Evaluating Static Analysis Tools for Detecting Buffer Overflows in C Code

Kendra June Kratkiewicz

A Thesis in the Field of Information Technology

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

March 2005

(corrected May 2005)

This work was sponsored by the United States Air Force under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **MAY 2005**	2. REPORT TYPE	3. DATES COVERED **00-00-2005 to 00-00-2005**

4. TITLE AND SUBTITLE **Evaluating Static Analysis Tools for Detecting Buffer Overflows in C Code**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Harvard University,Cambridge,MA,02138**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**299**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Copyright 2005 Kendra June Kratkiewicz

Abstract

This project evaluated five static analysis tools using a diagnostic test suite to determine their strengths and weaknesses in detecting a variety of buffer overflow flaws in C code. Detection, false alarm, and confusion rates were measured, along with execution time.

PolySpace demonstrated a superior detection rate on the basic test suite, missing only one out of a possible 291 detections. It may benefit from improving its treatment of signal handlers, and reducing both its false alarm rate (particularly for C library functions) and execution time. ARCHER performed quite well with no false alarms whatsoever; a few key enhancements, such as in its inter-procedural analysis and handling of C library functions, would boost its detection rate and should improve its performance on real-world code. Splint detected significantly fewer overflows and exhibited the highest false alarm rate. Improvements in its loop handling, and reductions in its false alarm rate would make it a much more useful tool. UNO had no false alarms, but missed a broad variety of overflows amounting to nearly half of the possible detections in the test suite. It would need improvement in many areas to become a very useful tool. BOON was clearly at the back of the pack, not even performing well on the subset of test cases where it could have been expected to function.

The project also provides a buffer overflow taxonomy, along with a test suite generator and other tools, that can be used by others to evaluate code analysis tools with respect to buffer overflow detection.

This work was sponsored by the United States Air Force under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

Acknowledgments

I would like to thank Dr. Richard Lippmann for offering to serve as my thesis director, and for his input and guidance along the way. I am also very grateful to many of my Lincoln Laboratory colleagues, including Dr. Robert Cunningham, for allowing me the flexibility in my work schedule needed to complete this project; Tim Leek, for discussions on the buffer overflow taxonomy and static analysis tools, as well as for the help he provided in getting the tools installed and running; Chris Scott, for database discussions, assistance with code review, general idea bouncing, and moral support; and Cindy McLain and Bracha Epstein, for the interest they took in my project and for the encouragement they provided.

Chris Hote, of PolySpace Technologies, was particularly gracious. I would not have been able to evaluate PolySpace without the temporary license he provided (and extended several times). He was always very helpful and responsive when I encountered problems with or had questions about their tool. Merci.

Many thanks to my husband, for diligently reviewing this document and providing constructive feedback, as well as for all the extra parenting and household duties he assumed in support of my goal. Many apologies to my children, for all the events I missed and for all the time I couldn’t spend with them while I pursued this degree.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

This project evaluated a selection of static analysis tools to determine their effectiveness and accuracy in detecting a variety of buffer overflow flaws in C code. The evaluation ran each tool against a carefully constructed suite of diagnostic test cases, measuring detection, false alarm, and confusion rates (the inability of the tools to accurately distinguish between a bad program and a corresponding patched program), subsequently comparing and analyzing the results to assess the strengths and weaknesses of the various tools. Chapter 1 provides the background necessary to put the project in context and describes the layout of the remainder of the document.

Background

Programmers sometimes make errors in code that may cause a program to crash or to behave in unexpected ways. For instance, they may fail to initialize a variable, to handle unusual user input, or to check that a buffer is large enough for the data they write into it. Such flaws are a security concern, or vulnerability, when they could allow users of the program to improperly influence the program's behavior. This happens when some form of input to the program (e.g., command line input, files, or packets) exercises the vulnerability, precipitating a crash or erroneous behavior. A malicious user may be able to exploit a vulnerability in such a way as to inject and execute his or her own code, possibly gaining privileges or information that he or she should not have. When widely used programs or servers, such as Sendmail or OpenSSH, contain vulnerabilities, the potential exists for large numbers of machines to be remotely exploited by attackers. Attackers may use the opportunity to launch distributed denial of service attacks, propagate worms, steal private data, or otherwise wreak havoc on machines and networks. Ideally, developers should discover and fix these vulnerabilities before programs are released.

Unfortunately, the scope of the problem is very large, and no universal solution exists. Types of flaws that exist in programs, as well as the ways in which they may be exploited, are vast and varied. Numerous approaches have been developed to attack the problem from different angles. The use of newer languages may eliminate entire classes of type and memory related programming flaws, but cannot prevent flaws in logic. Dynamic techniques aim to discover flaws during testing, but require a fully installed and executable program, and can discover only those flaws that the particular tests inputs and execution paths happen to expose. Other runtime solutions, built into languages and operating systems, may prevent flaws from being exploited by halting when illegal or undesirable operations, such as writing outside allocated memory, are about to take place; this, however, has the effect of turning a flaw into a denial of service, which may be equally problematic. Static analysis is another approach, comprising various compile-time techniques for detecting flaws by examining source code. Misha Zitser's master's thesis provides an informative survey of these numerous approaches to detecting flaws and preventing their exploitation (Zitser, 2003).

Static analysis would seem to be the ideal approach. It could be used during development to prevent the release of flaws in the first place, and could also be applied to already-released bodies of open source software. It does not require running the program, and does not depend upon developing large test suites in order to expose the flaws. A variety of static analysis techniques exist and have been incorporated into various research and commercial tools. It is not clear, however, that any of these solutions are comprehensive, effective, or accurate, since little data has been published on the actual performance of these tools. A truly useful and effective tool would need to detect a wide variety of flaws, have a high detection rate and low false alarm rate, be easy to use, and be scalable to large, real-world programs. Zitser's thesis and a related paper (Zitser, Lippmann, & Leek, 2004) present an evaluation of five static analysis tools, discussing how effectively they detect one particular class of software flaws: buffer overflows in C code.

Buffer Overflows

Buffer overflows are of particular interest as they are potentially exploitable by malicious users, and have historically accounted for a significant percentage of the software vulnerabilities that are published each year (Wagner, Foster, Brewer, & Aiken, 2000; Zitser, 2003), such as in NIST's ICAT Metabase (ICAT, 2004), CERT advisories (CERT, 2004), Bugtraq (Security Focus, 2004a), and other security forums. These buffer overflows have been the basis for many damaging exploits, such as the Sapphire/Slammer (Moore, 2003) and Blaster (PSS Security Response Team, 2003) worms.

A buffer overflow vulnerability occurs when data can be written outside the memory allocated for a buffer, either past the end or before the beginning. There are many ways that programmers may introduce such flaws into programs, such as by failing to limit the size of data written into a buffer (e.g., calling the `strcpy` function without checking to see if the source string will fit into the destination buffer), or miscalculating the size or limit of data being written into a buffer (e.g., failing to account for the null terminator that will be written to the buffer by the `strcat` function, causing the size calculation to be "off-by-one"). Buffer overflows may occur on the stack, on the heap, in the data segment, or the BSS segment (the memory area a program uses for uninitialized global data), and may overwrite from one byte to an unlimited number of bytes of memory outside the buffer. Amazingly, even a one-byte overflow can be enough to allow an exploit (klog, 1999).

Buffer overflows allow a malicious user to overwrite other pieces of information, such as a return address on the stack, a function pointer, or a data pointer, which may then alter the program's control flow. A denial of service attack uses this technique simply to crash the program. Worse yet, an attacker may inject his or her own attack code (through the same or another buffer overflow, or by other means, such as unchecked input parameters), and redirect the flow of control to the injected code. The attack code may then open a shell, enabling the attacker to execute arbitrary commands and completely control the victim's machine. Buffer overflows have been described at length in a great number of papers, including Zitser's thesis (Zitser, 2003). Many detailed descriptions of exploiting buffer overflows can be found online at the "subterrain" website (Subterrain, 2004).

Previous Evaluation

The Zitser thesis (Zitser, 2003) and related paper (Zitser, Lippmann, & Leek, 2004) evaluate the ability of several static analysis tools to detect fourteen known, historical vulnerabilities (all buffer overflows) in open source software packages. Due to the difficulty of using some of the tools on large-scale programs, they extracted the relevant code into smaller "model" programs, on the order of a few hundred or a thousand lines, containing just enough of the context and mechanics required to encapsulate the vulnerabilities. For each bad model program, they created a corresponding patched program, making only those changes necessary to correct the flaws. They ran static analysis tools against both the bad and the patched programs to see if the tools detected buffer overflows in the former or reported false alarms in the latter.

Zitser's evaluation included five static analysis tools: ARCHER (Xie, Chou, & Engler, 2003), BOON (Wagner et al., 2000), Splint (Evans & Larochelle, 2002; Larochelle & Evans 2001), UNO (Holzmann, 2002), and PolySpace C Verifier (PolySpace Technologies, 2003), the only commercial tool in the set. These tools all claim to employ some sophisticated static analysis techniques, including symbolic analysis, abstract interpretation, model checking, integer range analysis, and inter-procedural analysis. The study excluded tools based on simpler lexical analysis methods, such as RATS (Secure Software, 2003), ITS4 (Viega, Bloch, Kohno, & McGraw, 2000), Flawfinder (Wheeler), and BugScan (HBGary, 2003). These were of less interest because superficial lexical analysis techniques, which develop no deeper understanding of the code, are virtually guaranteed to produce unacceptable false alarm rates.

The results of the Zitser experiments were not encouraging, with only one out of the five evaluated systems performing statistically better than random guessing (i.e., a system using random guessing to label half of a program's lines of code as containing flaws). Not only did the tools fail to detect a significant number of flaws, but they also produced a large number of false alarms, indicating flaws where none actually existed. Equally discouraging was the confusion rate.

Given the small number of model programs, and the fact that the buffer overflows in these programs were embedded in complex code, it is difficult to draw many conclusions as to why the tools performed as poorly as they did. More experiments are required to gain a deeper understanding of the strengths and weaknesses of static analysis tools, and therefore to understand how they must be improved to become more useful. The tools must be run against a much larger and broader set of test cases. These test cases should be structured in such a way as to provide more granular and diagnostic information about the detection, false alarm, and confusion rates of the tools.

Thesis Goals

This project performed a more comprehensive analysis of the five tools evaluated in the previous study, to look more in-depth at the tools' performance with a broader, more diagnostic test suite. The goal was to understand what types of overflows the tools do and do not detect, when they report false alarms, and when they cannot distinguish between bad and patched code. Although this project evaluated only static analysis tools, it provides a taxonomy and test suite useful for evaluating dynamic analysis tools as well.

An in-depth analysis of the ability of static analysis tools to detect buffer overflows requires a buffer overflow taxonomy that can be used to classify particular instances of buffer overflows. This makes it possible to construct a suite of test cases that embodies as many different buffer overflows as possible. Experiments can then be performed by running the test suite through a selection of static analysis tools. The results may be analyzed to assess the strengths and weaknesses of the tools and techniques, discovering where improvements must be made to make the tools more useful. The following chapters further discuss these main project components.

Chapter 2 describes the buffer overflow taxonomy developed and used for this project. Chapter 3 discusses the construction of the test suite, while Chapter 4 explains the process used to evaluate the tools' performance on the test suite. The overall results and an analysis of each tool's behavior are presented in Chapter 5. The summary and conclusions appear in Chapter 6. Appendix 1 contains all the code and accessory files written during the course of this project.

Chapter 2 Buffer Overflow Taxonomy

A taxonomy is a classification, or a "systematic arrangement in groups or categories according to established criteria" (Merriam-Webster, 1997). Taxonomies have both explanatory and predictive value; the separation and ordering of specimens allows us to form generalizations about them, and to make predictions about unseen specimens by extrapolating from known specimens (Krsul, 1998). Using a comprehensive taxonomy to classify the buffer overflows in test programs presented to static analysis tools allows may allow us to generalize about the capabilities of the tools (e.g., "It appears that tool X cannot detect inter-procedural buffer overflows."), and possibly to predict how they would perform against other programs containing similar buffer overflows (e.g., "Tool X will not likely detect buffer overflow B, since it is inter-procedural.").

Krsul provides a comprehensive treatment of taxonomies and their importance as a conduit for research findings or bodies of knowledge in developing fields. In order to ensure repeatability of classifications, he further presents the following requirements for taxonomy attributes, or the characteristics of the specimens to be classified (Krsul, 1998):

> Objectivity: . . . The attribute being measured should be clearly observable.
>
> Determinism: There must be a clear procedure that can be followed to extract the feature.
>
> Repeatability: Several people independently extracting the same feature for the object must agree on the value observed.
>
> Specificity: The value for the feature must be unique and unambiguous.

Zitser presents a buffer overflow taxonomy comprising thirteen attributes, and uses it to classify the buffer overflows occurring in the model programs he created for his evaluation of static analysis tools (Zitser, 2003). I tried to use the same taxonomy to classify other buffer overflow specimens, but found that some areas lacked specificity and repeatability. This project modified and expanded that taxonomy to address the problems encountered, while still attempting to keep it small enough and simple enough for practical application. The new taxonomy consists of the twenty-two attributes listed in Table 1.

Note that the taxonomy developed for this project is specifically designed for applying to a diagnostic test suite, and may not be suitable as-is for classifying arbitrary buffer overflows in the wild. Buffer overflows occurring in real code may be more complex than can be classified with this taxonomy. This also means that the generalizations and predictions we can make based on the diagnostic test suite results will be limited. For instance, if a tool fails to detect every inter-procedural buffer overflow in the test suite, we may predict with some confidence that it will fail to detect inter-procedural buffer overflows in real code because they are likely to be more complex and even harder to detect. However, if the tool were to detect every inter-procedural buffer overflow in the test suite, we could not predict that it would detect such buffer overflows in real code.

Attribute Number	**Attribute Name**
1	Write/Read
2	Upper/Lower Bound
3	Data Type
4	Memory Location
5	Scope
6	Container
7	Pointer
8	Index Complexity
9	Address Complexity
10	Length/Limit Complexity
11	Alias of Buffer Address
12	Alias of Buffer Index
13	Local Control Flow
14	Secondary Control Flow
15	Loop Structure
16	Loop Complexity
17	Asynchrony
18	Taint
19	Runtime Environment Dependence
20	Magnitude
21	Continuous/Discrete
22	Signed/Unsigned Mismatch

Table 1. Buffer Overflow Taxonomy Attributes

For each attribute (except for Magnitude), the zero value is assigned to the simplest or “baseline” buffer overflow, as depicted in the program below:

```
int main(int argc, char *argv[])
{
  char buf[10];

  /*  BAD  */
  buf[10] = 'A';

  return 0;
}
```

The buffer access in this program is a write operation (Write/Read: 0) beyond the upper bound (Upper/Lower Bound: 0) of a stack-based (Memory Location: 0) character (Data Type: 0) buffer that is defined and overflowed within the same function (Scope: 0). The buffer does not lie within another container (Container: 0), is addressed directly (Pointer, Alias of Buffer Address: 0, Address Complexity: 0), is indexed with a constant (Alias of Buffer Index: 0, Index Complexity: 0). No C library function is used to effect the overflow (Length/Limit Complexity: 0). The overflow execution is not affected by any conditional or complicated control flow (Local Control Flow: 0, Secondary Control Flow: 0), asynchronous program constructs (Asynchrony: 0), or runtime dependencies (Runtime Environment Dependence: 0). The overflow writes to a discrete location (Continuous/Discrete: 0) one byte beyond the buffer boundary (Magnitude: 1), and cannot be manipulated by an external user (Taint: 0). Lastly, it does not involve a signed vs. unsigned type mismatch (Signed/Unsigned Mismatch: 0).

Appending the value digits for each of the twenty-two attributes forms a string that classifies a buffer overflow with respect to the taxonomy. For example, the sample program shown above would be classified as “0000000000000000000100.” Test programs can be tagged with classification strings, which can be referred to when analyzing the results of running the various tools against the test cases.

While the Zitser test cases were program pairs consisting of a bad program and a corresponding patched program, my evaluation uses program quadruplets. The four versions of each test case correspond to the four possible values of the Magnitude attribute; one of these represents the patched program (no overflow), while the remaining three indicate buffer overflows of varying sizes (one, eight, and 4096 bytes). All four programs in the quadruplet are tagged with identical classifications except for the value of the Magnitude attribute. Refer to the section on the Magnitude attribute later in this chapter for more information about the Magnitude values.

The sections that follow describe each attribute in detail and provide some insight as to why the attributes are of interest.

Attribute 1: Write/Read

This attribute poses the question "Is the buffer access an illegal write or an illegal read?" While detecting illegal writes is probably of more interest in preventing buffer overflow exploits, it is possible that illegal reads could allow unauthorized access to information or could constitute one operation in a multi-step exploit. Table 2 enumerates the possible values for the Write/Read attribute and gives examples that assume a ten byte buffer.

Value	**Description**	**Example**
0	write	`buf[10] = ‘A’`
1	read	`c = buf[10]`

Table 2. Write/Read Attribute Values

Attribute 2: Upper/Lower Bound

This attribute describes which buffer bound gets violated, the upper or the lower. While the term “buffer overflow” leads one to envision accessing beyond the upper bound of a buffer, it is equally possible to underflow a buffer, or access below its lower bound (Debian, 2004; Security Focus, 2004b). Table 3 documents the defined values for the Upper/Lower Bound attribute and shows examples assuming a ten-byte buffer.

Value	**Description**	**Example**
0	upper	`buf[10]`
1	lower	`buf[-1]`

Table 3. Upper/Lower Bound Attribute Values

Attribute 3: Data Type

The Data Type attribute, whose possible values and examples of which are shown in Table 4, describes the type of data stored in the buffer, as indicated in the buffer definition. Character buffers are often manipulated with unsafe string functions in C, and some tools may focus on detecting overflows of those buffers; buffers of all types may be overflowed, however, and should be analyzed.

Value	**Description**	**Example**
0	character	`char buf[10];`
1	integer	`int buf[10];`
2	floating point	`float buf[10];`
3	wide character	`wchar_t buf[10];`
4	pointer	`char * buf[10];`
5	unsigned integer	`unsigned int buf[10];`
6	unsigned character	`unsigned char buf[10];`

Table 4. Data Type Attribute Values

Attribute 4: Memory Location

The Memory Location attribute describes where the overflowed buffer resides. Table 5 lists the possible locations and provides examples. Non-static variables defined locally to a function are on the stack, while dynamically allocated buffers (e.g., those allocated by calling a `malloc` function) are on the heap. The data region holds initialized global or static variables, while the BSS region contains uninitialized global or static variables. Shared memory is typically allocated, mapped into and out of a program's address space, and released via operating system specific functions (e.g., shmget, shmat, shmdt, and shmctl on Linux). While a typical buffer overflow exploit may strive to overwrite a function return value on the stack, buffers in other locations have been exploited and should be considered as well (Conover, 1999).

Value	**Description**	**Example**
0	on the stack	`void function1() { char buf[10]; …}`
1	on the heap	`void function1() { char * buf; buf = (char *)malloc(10*sizeof(char)); …}`
2	in data region	`void function1() { static char buf[10] = “0123456789”; …}`
3	in BSS data	`void function1() { static char buf[10]; …}`
4	in shared memory	see text description of this attribute

Table 5. Memory Location Attribute Values

Attribute 5: Scope

The Scope attribute, as detailed in Table 6, describes the difference between where the buffer is allocated and where is it overrun. The scope is the same if the buffer is allocated and overrun within the same function. Inter-procedural scope describes a buffer that is allocated in one function and overrun in another function within the same file. Global scope indicates that the buffer is allocated as a global variable, and is overrun in a function within the same file. The scope is inter-file/inter-procedural if the buffer is allocated in a function in one file, and overrun in a function in another file. Inter-file/global scope describes a buffer that is allocated as a global in one file, and overrun in a function in another file.

Any scope other than the same may involve passing the buffer address as an argument to another function; in this case, the Alias of Buffer Address attribute must also be set accordingly. More sophisticated static analysis techniques may be required to detect buffer overflows or to avoid false alarms when the scope is not the same (Myers, 1981).

Note that the test suite used in this evaluation does not contain an example for "inter-file/global." This example, being more complicated to set up, was deferred in the interest of time.

Value	**Description**	**Example**

0	same	`void function1() {`
		`    char buf[10];`
		`    buf[10] = 'A'; }`

1	inter-procedural	`void function1() {`
		`    char buf[10];`
		`    function2(buf); }`
		`void function2(char * arg1) {`
		`    arg1[10] = 'A'; }`

2	global	`static char buf[10];`
		`void function1() {`
		`    buf[10] = 'A'; }`

3	inter-file/inter-procedural	`File 1:`
		`void function1() {`
		`    char buf[10];`
		`    function2(buf); }`
		`File 2:`
		`void function2(char * arg1) {`
		`    arg1[10] = 'A'; }`

4	inter-file/global	`File1:`
		`static char buf[10];`
		`File 2:`
		`extern char buf[];`
		`void function1() {`
		`    buf[10] = 'A'; }`

Table 6. Scope Attribute Values

Attribute 6: Container

The Container attribute asks, "Is the buffer inside of a container?" Table 7 enumerates the various containers considered in this evaluation and provides examples. Buffers may stand alone, or may be contained in arrays, structures, or unions. The buffer-containing structures and unions may be further contained in arrays.

The ability of static analysis tools to detect overflows within containers (e.g., overrunning one array element into the next, or one structure field into the next) and beyond container boundaries (i.e., beyond the memory allocated for the container as a whole) may vary according to how the tools model these containers and their contents.

Value	Description	Example
0	no	`char buf[10];`
1	array	`char buf[5][10];`

2	struct	`typedef struct {`
		`  char buf[10];`
		`} my_struct;`

3	union	`typedef union {`
		`  char buf[10];`
		`  int intval;`
		`} my_union;`

4	array of structs	`my_struct array_buf[5];`
5	array of unions	`my_union array_buf[5];`

Table 7. Container Attribute Values

Attribute 7: Pointer

The Pointer attribute indicates whether or not the buffer access uses a pointer dereference. Table 8 lists the possible values and some examples. Note that it is possible to use a pointer dereference with or without an array index; the Index Complexity attribute must be set accordingly.

In order to know if the memory location referred to by a dereferenced pointer is within buffer bounds, a code analysis tool must keep track of what pointers point to; this points-to analysis is a significant challenge (Landi, 1992).

Value	**Description**	**Example**
0	no	`buf[10]`
1	yes	`*pBuf  or (*pBuf)[10]`

Table 8. Pointer Attribute Values

Attribute 8: Index Complexity

This attribute describes the complexity of the array index, if any, of the buffer access causing the overflow. Possible values and examples are shown in Table 9. Note that this attribute applies only to the user program, and is not used to describe how buffer accesses are performed inside C library functions (for which the source may not be readily available).

Handling the variety of expressions that may be used for array indices is yet another challenge faced by code analysis tools.

Value	**Description**	**Example**
0	constant	`buf[10]`
1	variable	`buf[i]`
2	linear expression	`buf[5*i + 2]`
3	non-linear expression	`buf[i%3] or buf[i*i]`
4	function return value	`buf[strlen(buf)]`
5	array contents	`buf[array[i]]`
6	not applicable	`*pbuf = 'A'`

Table 9. Index Complexity Attribute Values

Attribute 9: Address Complexity

The Address Complexity attribute poses the question, “How complex is the address or pointer computation, if any, of the buffer being overflowed?” Table 10 describes the values and provides examples. Again, this attribute is used to describe the user program only, and is not applied to C library function internals.

Just as with array indices, code analysis tools must be able to handle a wide variety of expressions with varying degrees of complexity in order to accurately determine if the address accessed is beyond allocated buffer boundaries.

Value	Description	Example
0	constant	`buf[x], (buf+2)[x], (0x80097E34)[x], *(pBuf+2), strcpy(buf+2, src)`
1	variable	`(buf+i)[x], (bufAddrVar)[x], *(pBuf+i), strcpy(buf+i, src)`
2	linear expression	`(buf+(5*i + 2))[x], *(pBuf+(5*i + 2)), strcpy(buf+(5*i + 2), src)`
3	non-linear expression	`(buf+(i%3))[x], *(pBuf+(i*i)), strcpy(buf+(i%3), src)`
4	function return value	`(buf+f())[x], (getBufAddr())[x], *(pBuf+f()), *(getBufPtr()), strcpy(buf+f(), src), strcpy(getBufAddr(), src)`
5	array contents	`(buf+array[i])[x], (array[i])[x], *(pBuf+ array[i]), strcpy(buf+ array[i], src)`

Table 10. Address Complexity Attribute Values

Attribute 10: Length Complexity

The Length Complexity attribute describes the complexity of the length or limit passed to the C library function, if any, that overflows the buffer. Table 11 lists the possible values and provides examples.

Note that if a C library function overflows the buffer, the overflow is by definition inter-file/inter-procedural in scope, and involves at least one alias of the buffer address. In this case, the Scope and Alias of Buffer Address attributes must be set accordingly.

As with array indices and buffer addresses, C programs may contain arbitrarily complex expressions for the lengths or limits passed in C library function calls. Code analysis tools must be able to handle these in order to accurately detect buffer overflows. In addition, the code analysis tools may need to provide their own wrappers for or models of C library functions in order to perform a complete analysis (Bush, Pincus, & Sielaff, 2000).

Value	**Description**	**Example**

0	not applicable	`buf[10]`
		`(no library function called)`

1	none	`strcpy(buf, src)`
2	constant	`strncpy(buf, src, 10)`
3	variable	`strncpy(buf, src, i)`
4	linear expression	`strncpy(buf, src, 5*i + 2)`
5	non-linear expression	`strncpy(buf, src, i%3)`
6	function return value	`strncpy(buf, src, getSize())`
7	array contents	`strncpy(buf, src, array[i])`

Table 11. Length Complexity Attribute Values

Attribute 11: Alias of Buffer Address

This attribute indicates if the buffer is accessed directly or through one or two levels of aliasing. Assigning the original buffer address to a second variable and subsequently using the second variable to access the buffer constitutes one level of aliasing, as does passing the original buffer address to a second function. Similarly, assigning the second variable to a third and accessing the buffer through the third variable would be classified as two levels of aliasing, as would passing the buffer address to a third function from the second. See Table 12 for values and examples.

As mentioned in the Pointer attribute section, keeping track of aliases and what pointers point to is a significant challenge for code analysis tools.

Value	Description	Example

0	no	`char buf[10];`
		`buf[10] = ‘A’;`

1	one alias	`char buf[10];`
		`char * alias_one;`
		`alias_one = buf;`
		`alias_one[10] = ‘A’;`

2	two aliases	`char buf[10];`
		`char * alias_one;`
		`char * alias_two;`
		`alias_one = buf;`
		`alias_two = alias_one;`
		`alias_two[10] = ‘A’;`

Table 12. Alias of Buffer Address Attribute Values

Attribute 12: Alias of Index

The Alias of Index attribute is similar to the Alias of Address attribute, but applies to the index, if any, used in the buffer access rather than the buffer address itself. This attribute indicates whether or not the index is aliased. If the index is a constant or the results of a computation or function call, or if the index is a variable to which is directly assigned a constant value or the results of a computation or function call, then there is no aliasing of the index. If, however, the index is a variable to which the value of a second variable is assigned, then there is one level of aliasing. Adding a third variable assignment increases the level of aliasing to two. If no index is used in the buffer access, then this attribute is not applicable. Table 13 details the possible values and provides examples (assuming a ten byte character buffer).

Value	Description	Example
0	no	`int i = 10;` `buf[10] = ‘A’;` `buf[i] = ‘A’;`
1	one alias	`int i, j;` `i = 10; j = i;` `buf[j] = ‘A’;`
2	two aliases	`int i, j, k;` `i = 10; j = i; k = j;` `buf[k] = ‘A’;`
3	not applicable	`char * ptr;` `ptr = buf + 10;` `*ptr = ‘A’;`

Table 13. Alias of Index Attribute Values

Attribute 13: Local Control Flow

This attribute describes what kind of program control flow, if any, most immediately surrounds or affects the overflow. Possible values and examples are depicted in Table 14. For the values “if”, “switch”, and “cond”, the buffer overflow is located within the conditional construct. “Goto/label” signifies that the overflow occurs at or after the target label of a `goto` statement. Similarly, “setjmp/longjmp” means that the overflow is at or after a `longjmp` address. Buffer overflows that occur within functions reached via function pointers are assigned the “function pointer” value, and those within recursive functions receive the value “recursion”.

The values “function pointer” and “recursion” necessarily imply a global or inter-procedural scope, and may involve an address alias. The Scope and Alias of Address attributes should be set accordingly.

Control flow involves either branching or jumping to another context within the program; hence, only path-sensitive code analysis can determine whether or not the overflow is actually reachable (Xie et al., 2003). A code analysis tool must be able to follow function pointers and have techniques for handling recursive functions in order to detect buffer overflows with the last two values for this attribute.

Value	**Description**	**Example**
0	none	`buf[10] = ‘A’;`

1	if	`if (flag) {`
		`    buf[10] = ‘A’; }`

2	switch	`switch (value) {`
		`    case 1:`
		`        buf[10] = ‘A’;`
		`        break;`
		`      … }`

3	cond	`flag ? buf[10] = ‘A’ : 0;`

4	goto/label	`    goto my_label;`
		`my_label:`
		`    buf[10] = ‘A’;`

5	setjmp/longjmp	`if (setjmp(env) != 0) {`
		`    … }`
		`buf[4105] = 'A';`
		`longjmp(env, 1);`

6	function pointer	`In main:`
		`void (*fptr)(char *);`
		`char buf[10];`
		`fptr = function1;`
		`fptr(buf);`
		`void function1(char * arg1) {`
		`    arg1[10] = 'A'; }`

7	recursion	`In main:`
		`function1(buf);`
		`void function1(char *arg1, int counter) {`
		`    if (counter > 0) {`
		`        function1(arg1, counter - 1);  }`
		`    arg1[10] = ‘A’;`
		`    … }`

Table 14. Local Control Flow Attribute Values

Attribute 14: Secondary Control Flow

The types of control flow described by the Secondary Control Flow attribute are the same as the types described by the Local Control Flow; the difference is the location of the buffer overflow with respect to the control flow construct. Secondary control flow either precedes the overflow, or contains nested local control flow that affects the overflow (e.g., nested `if` statements, or an `if` statement within a `case` of a `switch`). See Table 15 for a list of values.

Only control flow that affects whether or not the overflow occurs is classified. In other words, a preceding `if` statement that has no bearing on the overflow is not labeled as any kind of secondary control flow. Some types of secondary control flow may occur without any local control flow, but some may not. If not, the Local Control Flow attribute should be set accordingly.

Some code analysis tools perform path-sensitive analyses, and some do not. Even those that do often must make simplifying approximations in order to keep the problem tractable and the solution scalable. This may mean throwing away some information, and thereby sacrificing precision, at points in the program where previous branches rejoin (Gregor & Schupp, 2003). Test cases containing secondary control flow that precedes the buffer overflow may highlight the capabilities or limitations of these varying techniques.

Value	Description	Example

0	none	`if (feel_like_it) {`
		`    do_something_unrelated(); }`
		`buf[10] = ‘A’;`

1	if	`if (flag1) {`
		`    flag2 ? buf[10] = ‘A’ : 0; }`

2	switch	`switch (value) {`
		`    case 1:`
		`        flag ? buf[10] = ‘A’ : 0;`
		`        break;`
		`     … }`

3	cond	`i = (j > 10) ? 10 : j;`
		`buf[i] = 'A';`

4	goto/label	`    goto my_label;`
		`my_label:`
		`    flag ? buf[10] = ‘A’ : 0;`

5	setjmp/longmp	`if (setjmp(env) != 0) {`
		`    … }`
		`flag ? buf[10] = ‘A’ : 0;`
		`longjmp(env, 1);`

6	function pointer	`In main:`
		`void (*fptr)(char *);`
		`char buf[10];`
		`fptr = function1;`
		`fptr(buf);`
		`void function1(char * arg1) {`
		`    flag ? arg1[10] = ‘A’ : 0; }`

7	recursion	`In main:`
		`function1(buf);`
		`void function1(char *arg1, int counter) {`
		`    if (counter > 0) {`
		`        function1(arg1, counter - 1); }`
		`    flag ? arg1[10] = ‘A’ : 0;`
		`    … }`

Table 15. Secondary Control Flow Attribute Values

Attribute 15: Loop Structure

The Loop Structure attribute describes the type of loop construct, if any, within which the overflow occurs. Table 16 shows the possible values and some examples. This taxonomy defines a “standard” loop as one that has an initialization, a loop exit test, and an increment or decrement of a loop variable, all in typical format and locations. A “non-standard” loop deviates from the standard loop in one or more of these three areas.

Omitting the initialization and/or the increment when they're not applicable does not constitute a deviation from the standard.

Non-standard loops may necessitate the introduction of secondary control flow (such as additional `if` statements). In these cases, the Secondary Control Flow attribute should be set accordingly. Any value other than "none" for this attribute requires that the Loop Complexity attribute be set to something other than "not applicable."

Loops may execute for a large number or even an infinite number of iterations, or may have exit criteria that depend on runtime conditions; therefore, it may be impossible or impractical for static analysis tools to simulate or analyze loops to completion. Different tools have different methods for handling loops; for example, some may attempt to simulate a loop for a fixed number of iterations, while others may employ heuristics to recognize and handle common loop constructs. The approach taken will likely affect a tool's capabilities to detect overflows that occur within various loop structures.

Value	Description	Example
0	none	`buf[10] = ‘A’;`
1	standard for	`for (i=0; i<11; i++) { buf[i] = ‘A’; }`
2	standard do-while	`i=0; do { buf[i] = ‘A’; i++; } while (i<11);`
3	standard while	`i=0; while (i<11) { buf[i] = ‘A’; i++; }`
4	non-standard for	`i=0 for ( ; i<11; i++) { buf[i] = ‘A’; }`
5	non-standard do-while	`i=0; do { buf[i] = ‘A’; i++; if (i>10) break; } while (1);`
6	non-standard while	`i=0; while (++i) { buf[i] = ‘A’; if (i>10) break; }`

Table 16. Loop Structure Attribute Values

Attribute 16: Loop Complexity

The Loop Complexity attribute, detailed in Table 17, indicates how many of the three loop components described under Loop Structure (i.e., init, test, and increment) are more complex than the following baseline:

init: initializes to a constant

test: tests against a constant

increment: increments or decrements by one

If the overflow does not occur within a loop, this attribute is not applicable. If none of the three loop components exceeds baseline complexity, the value assigned is “none.” If one of the components is more complex, the appropriate value is “one,” and so on. Of interest here is whether or not the tools handle loops with varying complexity in general, rather than which particular loop components are handled or not.

For any value other than “not applicable,” the Loop Structure attribute must be set to one of the standard or non-standard loop values.

Value	**Description**	**Example**
0	not applicable	`buf[10] = ‘A’;`

1	none	`for (i=0; i<11; i++) {`
		`    buf[i] = ‘A’; }`

2	one	`init = 0;`
		`for (i=init; i<11; i++) {`
		`    buf[i] = ‘A’; }`

3	two	`init = 0;`
		`test = 11;`
		`for (i=init; i<test; i++) {`
		`    buf[i] = ‘A’; }`

4	three	`init = 0;`
		`test = 11;`
		`inc = k – 10;`
		`for (i=init; i<test; i += inc) {`
		`    buf[i] = ‘A’; }`

Table 17. Loop Complexity Attribute Values

Attribute 17: Asynchrony

The Asynchrony attribute asks if the buffer overflow is potentially obfuscated by an asynchronous program construct. Table 18 lists the possible values, along with examples of functions that may be used to realize the constructs. These functions are often operating system specific. A code analysis tool may need detailed, embedded knowledge of these constructs and the O/S-specific functions in order to properly detect overflows that occur only under these special circumstances.

Value	**Description**	**Example**
0	no	n/a
1	threads	`pthread_create, pthread_exit`
2	forked process	`fork, wait, exit`
3	signal handler	`signal`

Table 18. Asynchrony Attribute Values

Attribute 18: Taint

The Taint attribute describes how a buffer overflow may be influenced externally. Possible values are shown in Table 19, along with examples of related functions. These functions may be operating system specific. The occurrence of a buffer overflow may depend on command line or `stdin` input from a user, the value of environment variables, file contents, data received through a socket or service, or properties of the process environment, such as the current working directory. All of these can be influenced by users external to the program, and are therefore considered “taintable.”

These may be the most crucial overflows to detect, as it is ultimately the ability of the external user to influence program operation that makes exploits possible. As with asynchronous constructs, code analysis tools may require detailed modeling of these functions in order to properly detect related overflows.

Note that the test suite used in this evaluation does not contain an example for “socket/service.” This example, being more complicated to set up, was deferred in the interest of time.

Value	**Description**	**Example**
0	no	n/a
1	argc/argv	using values from `argv`
2	environment variables	`getenv`
3	file read (or stdin)	`fgets, fread, read`
4	socket/service	`recv`
5	process environment	`getcwd`

Table 19. Taint Attribute Values

Attribute 19: Runtime Environment Dependence

This attribute, detailed in Table 20, indicates whether or not the occurrence of the overrun depends on something determined at runtime. If the overrun is certain to occur on every execution of the program, it is not dependent on the runtime environment; otherwise, it is. The examples discussed under the Taint attribute are examples of overflows that may be runtime dependent. Another example would be an overflow that may or may not occur, depending on the value of randomly generated number.

Intuition suggests that it should be easier to detect overflows and avoid false alarms when runtime behavior is guaranteed to be the same for each execution.

Value	Description
0	no
1	yes

Table 20. Runtime Environment Dependence Attribute Values

Attribute 20: Magnitude

The Magnitude attribute indicates the size of the overflow. “None” indicates that there is no overflow; it is used when classifying patched programs that correspond to bad programs containing overflows. The remaining values indicate how many bytes of memory the program will attempt to write outside the allocated buffer. Values and examples (assuming a ten byte buffer) are depicted in Table 2.

One would expect static analysis tools to detect buffer overflows without regard to the size of the overflow, unless they contain an off-by-one error in their modeling of library functions. The same is not true of dynamic analysis tools that use runtime instrumentation to detect memory violations; different methods may be sensitive to different sizes of overflows, which may or may not breach page boundaries, etc. The various overflow sizes were chosen with future dynamic tool evaluations in mind. Overflows of one byte test both the accuracy of static analysis modeling, and the sensitivity of dynamic instrumentation. Eight and 4096 byte overflows are aimed more exclusively at dynamic tool testing, and are designed to cross word-aligned and page boundaries.

Value	**Description**	**Example**
0	none	`buf[9] = 'A';`
1	1 byte	`buf[10] = 'A';`
2	8 bytes	`buf[17] = 'A';`
3	4096 bytes	`buf[4105] = 'A';`

Table 21. Magnitude Attribute Values

Attribute 21: Continuous/Discrete

This attribute indicates whether the buffer overflow is continuous or discrete. A continuous overflow accesses consecutive elements within the buffer before overflowing past the bounds, whereas a discrete overflow jumps directly out of the buffer. See Table 22 for values and examples (assuming a ten byte buffer).

Loop constructs are likely candidates for containing continuous overflows. C library functions that overflow a buffer while copying memory or string contents into it demonstrate continuous overflows. An overflow labeled as continuous should have the loop-related attributes or the Length Complexity attribute (indicating the complexity of the length or limit passed to a C library function) set accordingly.

Some dynamic techniques rely on “canaries” at buffer boundaries to detect continuous overflows; tools that rely on such techniques may miss discrete overflows (Bulba & Kil3r, 2000).

Value	**Description**	**Example**
0	discrete	`buf[10] = ‘A’;`
1	continuous	`for (i=0; i<11; i++) { buf[i] = ‘A’; }`

Table 22. Continuous/Discrete Attribute Values

Attribute 22: Signed/Unsigned Mismatch

This attribute indicates if the buffer overflow is caused by a signed vs. unsigned type mismatch. Table 23 depicts values and examples (based on a ten byte buffer). Typically, a signed value is used where an unsigned value is expected, and gets interpreted as a very large unsigned or positive value. For instance, the second example in Table 23 shows a size being calculated and passed to the `memcpy` function. Since the buffer is ten bytes long, the size calculated and passed to `memcpy` is negative one. The `memcpy` function, however, expects an unsigned value for the size parameter, and interprets the negative one as a huge positive number, causing an enormous buffer overflow. Several real exploits have been based on this type of overflow (Security Focus, 2004c).

Value	Description	Example
0	no	`memcpy(buf, src, 11);`

1	yes	`signed int size = 9 – sizeof(buf);`
		`memcpy(buf, src, size);`

Table 23. Signed/Unsigned Mismatch Attribute Values

Chapter 3 Test Suite

This chapter covers the test suite developed and used for this evaluation. It discusses the test suite design issues, and describes the program written to automatically generate the test cases.   It presents the basic test suite generated and used for the evaluation, and provides an explanation of the quality control measures used to ensure the validity of the test suite.

Design Considerations

This section discusses some of the issues considered when designing the test suite used for the evaluation.

Lack of Specific Tool Bias

Test cases were not specifically tailored to take advantage of any particular tool’s strengths or shortcomings, so as not to unfairly affect that tool’s performance “score”.

Distribution across the Taxonomy

Ideally, the test suite would have at least one instance of each possible buffer overflow that could be described by the taxonomy. Unfortunately, this is completely impractical, there being millions of such possibilities. Instead, a “basic” set of test cases was built by first choosing a simple, baseline example of a buffer overflow, and then varying its characteristics one at a time. The Basic Test Suite section of this chapter describes the actual test suite generated and used for the evaluation, and presents its distribution statistics.

Scoring

Detection, false alarm, and confusion rates were measured. This means that for each bad (containing a buffer overflow) program constructed, it was necessary to construct a corresponding patched program with no overflow. The patched program varies as little as possible from the bad program, containing only those changes necessary to minimally correct the flaw.

Naming and Documentation

Test cases were named and documented so as to make it clear which type of buffer overflow is represented by each test case, as well as where exactly the buffer overflow occurs within the program. This is necessary to analyze and score the tools' performance, as well as to make the test case suite comprehensible and useful to others.

Test program names consist of three parts separated with hyphens: the "basic" prefix, the five-digit test case number, and a suffix that indicates the magnitude of the overflow ("ok" for no overflow, "min" for one byte overflows, "med" for eight bytes, and "large" for 4096 bytes). For example, there are four versions of the baseline test case, named basic-00001-ok.c, basic-00001-min.c, basic-00001-med.c, and basic-00001-large.c.

The first line of every test case file is a comment containing the classification string, which is useful for automated processing. An entire block of comment lines follows, expanding the classification string into more human-readable documentation. In addition, a “BAD” comment line precedes the lines containing overflows in the bad programs, while an “OK” comment line precedes the corresponding lines in the patched programs. The example below illustrates this documentation for the patched, baseline test case:

```
/* Taxonomy Classification: 0000000000000000000000 */

/*
 *  WRITE/READ                   0    write
 *  WHICH BOUND                  0    upper
 *  DATA TYPE                    0    char
 *  MEMORY LOCATION              0    stack
 *  SCOPE                        0    same
 *  CONTAINER                    0    no
 *  POINTER                      0    no
 *  INDEX COMPLEXITY             0    constant
 *  ADDRESS COMPLEXITY           0    constant
 *  LENGTH COMPLEXITY            0    N/A
 *  ADDRESS ALIAS                0    none
 *  INDEX ALIAS                  0    none
 *  LOCAL CONTROL FLOW           0    none
 *  SECONDARY CONTROL FLOW       0    none
 *  LOOP STRUCTURE               0    no
 *  LOOP COMPLEXITY              0    N/A
 *  ASYNCHRONY                   0    no
 *  TAINT                        0    no
 *  RUNTIME ENV. DEPENDENCE      0    no
 *  MAGNITUDE                    0    no overflow
 *  CONTINUOUS/DISCRETE          0    discrete
 *  SIGNEDNESS                   0    no
 */

int main(int argc, char *argv[])
{
  char buf[10];

  /*  OK  */
  buf[9] = 'A';

  return 0;
}
```

Consistency

Care was taken to make the test cases consistent in their programming style and use of programming idioms in order to avoid introducing variations that might affect the tools' performance. Additionally, the test cases are consistent whenever possible in terms of runtime requirements (e.g., command line arguments or files expected); this facilitates automatically running the test cases through code analysis or quality control tools.

Quality Control

Except for the intentional buffer overflows, the test cases must be otherwise error free. They must compile without unrelated warnings or errors to be sure that the tools' ability to detect the specific buffer overflows constructed is the only thing being measured. They must also be executable, so that analyses can demonstrate that bad programs contain overflows and patched programs do not. A side benefit of the "executable" requirement is that the test suite will also be useful for dynamic analysis tool evaluations. Another quality control consideration is the problem of ensuring that the test cases are properly classified according to the taxonomy. This is exceedingly time consuming and error prone when done manually (by human inspection); it is, therefore, desirable to construct and document test cases in such a way as to facilitate the automation of this task as much as possible. The Test Suite Quality Control section later in this chapter describes the quality control measures that were taken for this evaluation.

Test Suite Generator

Zitser's thesis mentions a set of fifty simple test cases constructed as part of his project, which would be used by other researchers to evaluate static analysis tools (Zitser, 2003). My thesis project was partly inspired by my experience with trying to use and verify the correctness of those test cases. Realizing the time consuming and error prone nature of manually constructing and documenting such test cases, I chose instead to write an automatic test case generator. The goal was to generate several hundred test cases, and to verify more easily that they were classified and documented correctly. Many of the test cases generated by my program are based on test cases from Zitser's simple test suite.

Generator Command Line

The test suite generator, *gen_basic_tests.pl*, appears in its entirety in the Perl Files section of Appendix 1. Command line invocation is as follows:

```
gen_basic_tests.pl [--combo] <attribute list>
```

The attribute list consists of taxonomy attribute numbers separated by spaces. Without the combo option, the generator produces test cases for each attribute in the list, varying one attribute at a time. For instance, if the command line specifies attributes three and four, the generator produces one set of test cases that starts with the baseline test case and varies the Data Type attribute, and another independent set that starts with the baseline test case and varies the Memory Location attribute. If the combo option is specified, the generator produces a set of test cases that varies the attributes in combination. For example, if attributes three and four are specified (in that order) in conjunction with the combo option, the generator produces an intermediate set of test cases that start with the baseline test case and vary the Data Type attribute; it produces the final set by starting with each of the intermediate test cases and varying the Memory Location attribute. This process yields combinations of simultaneously varying Data Type and Memory Location attributes.

Generator Program Structure

Figure 1 presents a high-level picture of the generator's structure. The *main* function uses *get_fseq_num* to find the next, unused test case number, and passes that to *make_files*, along with a list of attributes to vary. If the combo option was specified, it passes the entire list at once; otherwise, it loops and passes the attributes one at a time.

The *make_files* function passes the attributes list to *handle_attributes*, and gets back an array of test case information. Each element of the array holds the information necessary to generate one test case, which consists of four files (ok, min, med, and large versions). For each element in the array, *make_files* calls upon *get_filenames*, passing it the current test case number, to get the set of four file names to use. It proceeds to write out the file contents based on the test case information held in the current array element. The *substitute_values* function substitutes current test case values for place-holder values in templates that represent different pieces of the final program. Table 24 describes in more detail the various components from which a test case file is pieced together, while the examples in Table 25 further illustrate the concept of place-holders. The *make_files* function also instructs *expand_tax_class* to place the taxonomy classification documentation into the file along with the code, and generates any additional files specified by the test case (e.g., an input file required to produce the overflow for the test case with the “file read” value of the Taint attribute).

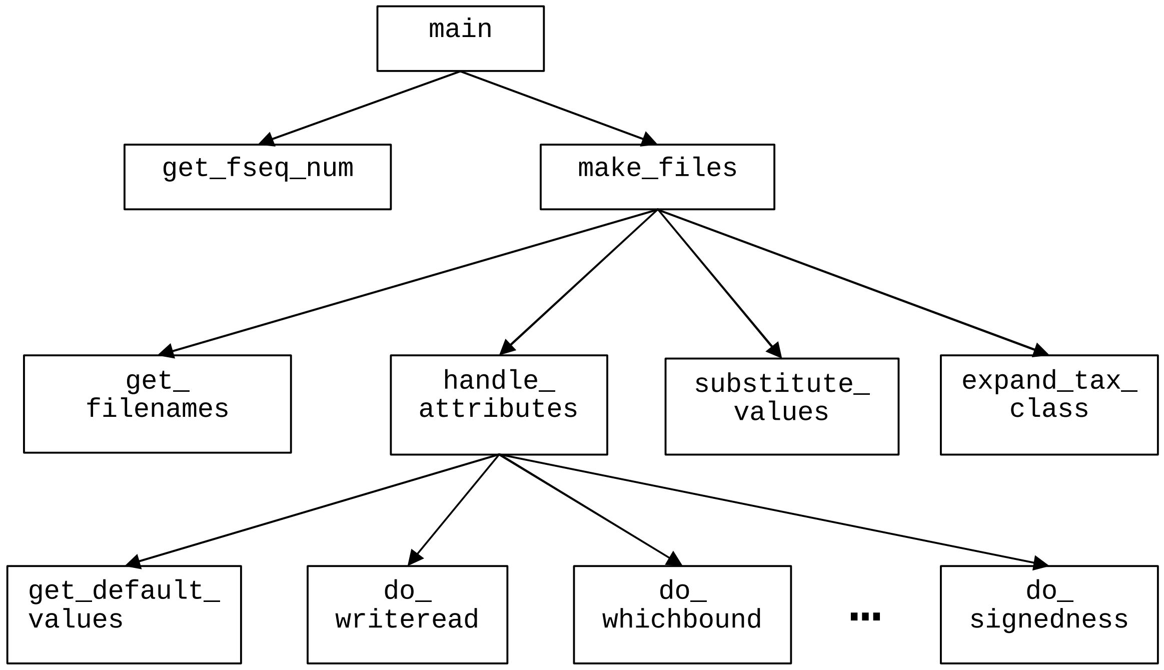


Figure 1. High-level Structure of gen_basic_tests.pl

Component	Description
taxonomy classification	The documentation described under the Naming and Document section of this chapter
FILE_HEADER	The copyright notice
INCL	All required #include statements
FILE_GLOBAL	Declarations global to the file
BEFORE_MAIN	All code that must appear before the *main* function
MAIN_OPEN	The function signature and opening brace for the *main* function
OTHER_DECLS	All variable declarations other than the buffer to be overflowed
BUF_DECL	The declaration of the buffer to be overflowed
PRE_ACCESS	All code in the *main* function that precedes the overflowing or patched buffer access
ACCESS	The overflowing or patched buffer access
POST_ACCESS	All code in the *main* function that follows the overflowing or patched buffer access
MAIN_CLOSE	The closing brace of the *main* function

Table 24. Test Case File Components

Template Name	Template String	Place-holders
TAXONOMY_CLASSIFICATION	`"/* Taxonomy Classification: VALUE */\n\n"`	VALUE
BUF_DECL	`"  TYPE buf[$BUF_SIZE];\n\n"`	TYPE, BUF_SIZE
BUF_ACCESS	`"COMMENT\n  buf[INDEX] = WRITE_VALUE;\n"`	COMMENT, INDEX, WRITE_VALUE

Table 25. Examples of Templates and Place-holders

The *handle_attributes* function calls upon *get_default_values* to retrieve the templates and initial values that reflect the baseline test case. These values are passed to the appropriate *do_<attribute>* functions for modification. If the "combo" command line option is not specified, there is only one attribute in the list passed to *handle_attributes;* hence, the default values are passed into the *do_<attribute>* function corresponding to that attribute, they are modified as required to produce test cases that vary that attribute, and the final results are handed back to *handle_attributes*, which in turn hands them back to *make_files*. If the combo option is specified, then there are multiple attributes in the list passed to *handle_attributes.* In this case, *handle_attributes* passes the default values to the first attribute handler, passes the results from the first handler to the second handler, and so on; after the last attribute is handled, *handle_attributes* passes the final results back to *make_files.* In this way, the modifications build upon each other and produce attribute variations in combination.

There are twenty-two attribute handler functions, each one knowing how to modify the templates and values passed into it to produce a new set of test cases. The new set contains at least one test case for each possible value of the attribute (not including the baseline value; the baseline test case is produced only once as part of the Attribute 1 test cases to avoid duplicates). As an example, the Data Type attribute has six values other than the default value of "character." Its handler function, *do_datatype*, produces six new test cases that are just like the one passed into the function, except the data type of the buffer contents has been changed to a new type, such as "int," "float," "wchar_t," etc. It accomplishes this by assigning new values to the TYPE and WRITE_VALUE place-holders that appear in the BUF_DECL and BUF_ACCESS templates, respectively (Table 25). When *make_files* calls *substitute_values*, the latter function takes the place-holder values for a particular test case and substitutes them for the place-holder names where they appear in the templates.

Sometimes the handler function produces more than one test case for a particular attribute value if there are multiple ways to illustrate the value and it is desirable to test more than one of them. Consider, for example, the Alias of Buffer Address attribute. There are two different ways in which an overflow could involve one alias of the buffer address: the address is assigned to and accessed via a second variable (Table 12), or the address is passed as an argument to a second function which overflows the buffer. In this case, *do_aliasaddress* produces two test cases for the "one alias" value: one that illustrates each of the described situations.

While *gen_basic_tests.pl* was designed and written with the combination capability in mind, generating a basic, rather than combination, test suite was the first priority. Combinations were not actually generated and evaluated for this project, due to lack of time. I wrote the attribute handler functions such that they can operate on any incoming test case (i.e., not assuming that the incoming test case is the baseline case) whenever it was relatively easy to do so. If building the combination capability into a handler would require significantly more effort, I deferred the complexity and wrote the handler to operate on the baseline case only. These locations in the code are marked with a "COMBO NOTE" comment. Other comments about desirable future capabilities are marked "FUTURE."

Basic Test Suite

The batch file *gen_suite_here.bat*, used to generate the basic test suite for this evaluation, appears in the BAT Files section of Appendix 1. Attributes 19 (Runtime Environment Dependence) and 20 (Magnitude) are not listed on the command line, since the test cases produced for all the other attributes cover these two attributes' values at the same time. Since the combo option is not specified, the test cases are produced independently for each attribute, varying from the baseline test case. The test cases are all generated in the same directory, and are consecutively numbered from "00001" to "00291." Since each test case comprises a quadruplet of files, and one test input file is generated for a "file read" test case, a total of 1165 files are generated when this batch file runs. A representative sample of generated test cases appears in Appendix 1, including all four Magnitude versions of the baseline test case (basic-00001-ok.c, basic-00001-min.c, basic-00001-med.c, and basic-00001-large.c).

Note that two other test generation batch files are included in Appendix 1: *gen_suite_dirs.bat* generates the same test cases as *gen_suite_here.bat*, but groups the test cases in directories by attribute; *gen_combo_dirs.bat* illustrates how to use *gen_basic_tests.pl* to generate a set of test cases combining the Data Type attribute with each other attribute in turn.

The Perl program *analyze_coverage.pl*, also depicted in Appendix 1, analyzes the test case files in a given directory and reports taxonomy coverage statistics to `stdout`. Appendix 2 contains the output generated from running the analyzer on the basic test suite. The report details how many test case files in the given directory contain examples of each attribute value. The analyzer discovers this information by parsing the taxonomy classification string (described in the Naming and Documentation section earlier in this chapter) appearing on the first line of each test case file.

Due to the strategy of generating test cases by varying one attribute at a time with respect to a baseline test case, the taxonomy coverage is, in general, heavily weighted toward the baseline attribute values.   Many attribute values are represented by four test case files, or one test case quadruplet. The distribution is more evenly spread across values for some attributes, because examples for some attribute values can only exist in combination with others. For example, the different loop complexities can only be demonstrated in combination with the various loop structures. The coverage across the Magnitude attribute values is perfectly even, since, by design, each test case quadruplet includes one example of each of the four Magnitude values.

As noted in Chapter 2, there is no coverage of two different attribute values: Scope – inter-file/global, and Taint – socket. These were more complicated to set up, and were deferred in the interest of time. They should be added to the test suite in the future to make it more complete. In general, future work on the test suite should strive for a more even distribution across the various attribute values.

Test Suite Quality Control

In order for the evaluation results and analysis to be valid, it is imperative that the test suite be accurate. The Quality Control section earlier in this chapter discusses the quality control issues taken into consideration when designing the test suite. This section describes the considerable measures taken to address these issues and to ensure an accurate test suite. Figure 2 depicts the process followed.

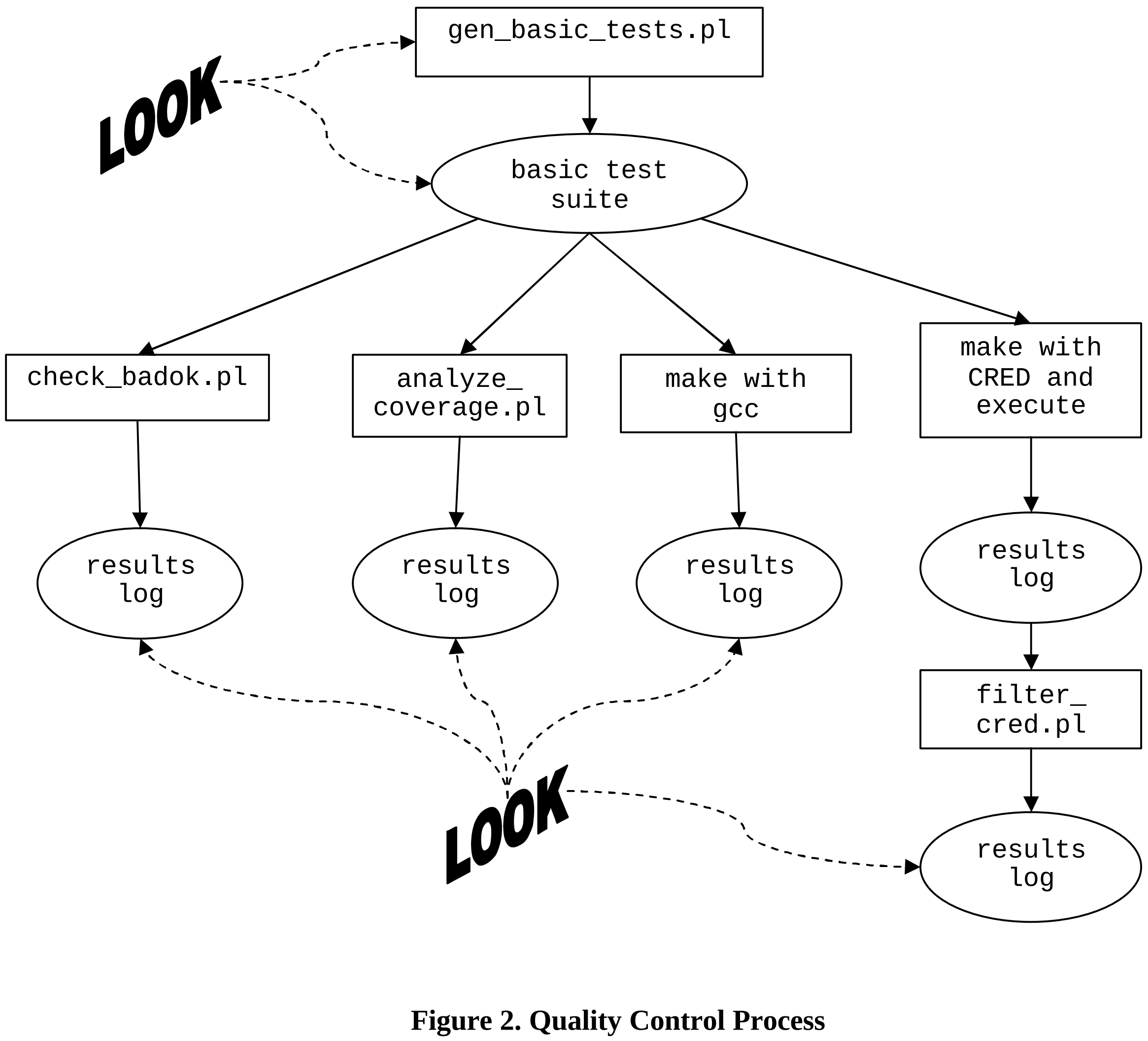


Figure 2. Quality Control Process

A colleague and I each performed a code review of the test suite generator, *gen_basic_tests.pl*, with an eye toward verifying that the taxonomy classification digits were set properly for each test case generated. The code review revealed one improperly set value in *do_lencomplex*, and several places where inter-attribute dependencies were not accounted for, which I subsequently fixed. Additionally, while developing and testing the generator one attribute handler at a time, I visually inspected the test cases being produced and verified that they looked as expected.

Several quality control tools were developed and run against the generated test suite; the code for all of these tools appears in Appendix 1. The *check_badok.pl* program verifies that each bad program (named –min, -med, or –large) contains a "BAD" comment line, and that each patched program (named –ok) contains an "OK" comment line. It prints a pass/fail report for each test case file, and this was visually inspected to verify that no failures occurred.

The *analyze_coverage.pl* program was introduced earlier in this chapter in the Basic Test Suite section. While its primary purpose was to document the taxonomy coverage, the results were also inspected to verify that there were as many test cases as expected for each attribute value. This served to double-check that the taxonomy classification was documented correctly in each test case file.

Each test case was compiled with gcc, the GNU C compiler, on Linux (GCC Home Page, 2004). See the listing for *Makefile* in Appendix 1 for details on the options used. The strictest warning options were specified, and the output was inspected to verify that the programs compiled without warnings or errors. The only test case in the suite that produced a warning when compiled was basic-00045-large.c; this test case initializes a variable to a string that is 4105 characters long, which the compiler warns is "greater than the length '509' ISO C89 compilers are required to support." The warning is unavoidable given what the test case needs to do, so the test case was retained intact.

Each test case file was also compiled with CRED, a fine-grained bounds-checking modification to gcc that instruments the resulting executable such that bounds violations can be detected and reported at runtime (Ruwase & Lam, 2004). The instrumented test case programs were then executed, and the CRED output collected (see Appendix 3 for sample CRED output). The *Makefile* listing in Appendix 1 shows how the CRED compilation and execution was accomplished. The *filter_cred.pl* program was developed to filter through all the CRED output to determine whether or not CRED reported a bounds error for the appropriate line of source code for each executed test case.

The *filter_cred.pl* output was visually inspected to look for any of the possible failure messages: "ERROR: OK overflowed," "ERROR: BAD did not overflow," or "ERROR: overflow on wrong line." The "bad did not overflow" error was reported for five of the test cases: 00013 (shared memory), 00177 (threads), 00181 (environment variable), 00182 (file read – `fgets` function), and 00183 (process environment – `getcwd` function). These were investigated individually, and in each case it was determined that the lack of a bounds error detection was a CRED shortcoming, rather than an error in the test cases; for each of these five test cases the large overflow version, and sometimes other versions, would segfault when executed from the command line.

Chapter 4 Tool Evaluation

This chapter describes how the evaluated static analysis tools were run, and the results collected and imported into a database. A general overview of the process is presented first, followed by sections specific to each tool. The last section describes the database and related scripts.

The evaluation ran the basic test suite, consisting of 291 test case quadruplets, through five static analysis tools (ARCHER, BOON, PolySpace, Splint, and UNO), and collected the output from each tool. Tool-specific Perl programs parsed the collected output, recording how the tools performed on each test case. The recorded results indicated whether the tool detected a buffer overflow on the commented buffer access line in each of the four versions (-ok, -min, -med, -large) of every test case. Additional scripts prepared the database and imported these results. Queries were then run against the database as part of the results analysis; Chapter 5 describes this analysis.

Figure 3 depicts this tool evaluation process for one generic tool represented by "<tool>." For this evaluation, there were five such paths through the process: one for each of ARCHER, BOON, PolySpace, Splint, and UNO; simply substitute the names of these tools wherever "<tool>" appears in the figure. The scripts used to run each of the tools against the basic test suite appear in the SH Files section of Appendix 1. Sample output from each tool appears in Appendix 4. All of the Perl programs written to tabulate the results and generate data suitable for importing into the database are shown in the Perl

Files section of Appendix 1. The *gen_testcase_db_info.pl* program and *setupDBtables.sql* are described in the Database Description section, later in this chapter.

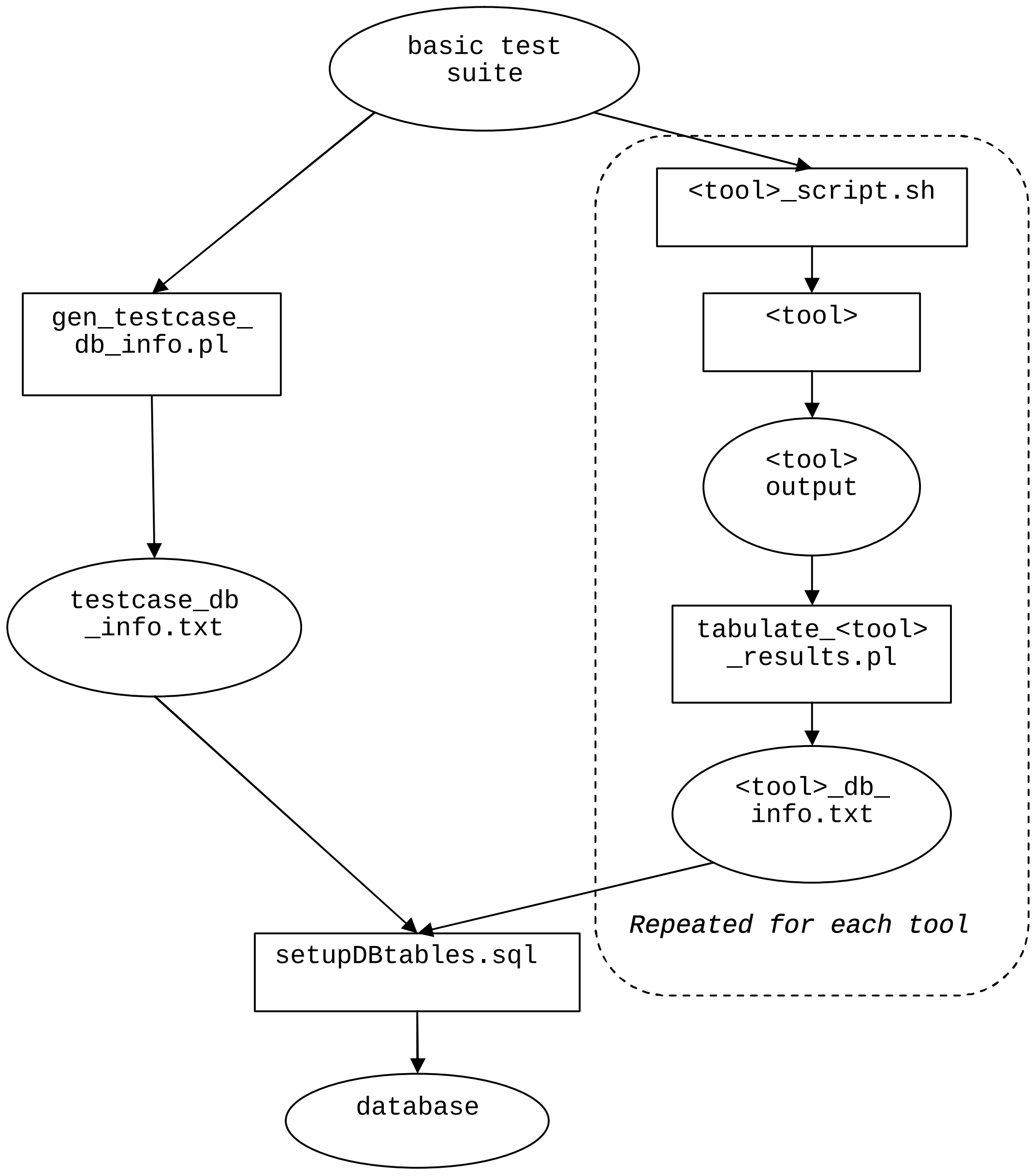


Figure 3. Tool Evaluation Process

ARCHER Evaluation

The script *Archer_script.sh* (see Appendix 1) runs ARCHER against each test case file in the suite, directing the results to an “Archer_results” directory. The results directory contains one results output file for each test case input file; the results file name is the same as the test case file name with the “.c” extension removed.

Appendix 4 contains the ARCHER output file *basic-00001-min*; this file illustrates how an ARCHER error report contains the source file name, the function name, and the line number where the error occurred, along with a descriptive phrase and details of the offending access. The *tabulate_archer_results.pl* program (see Appendix 1) parses through the results files looking for any of the known phrases that indicate a buffer overflow: “Array bounds error,” “array bounds error,” or “out-of-bound array or string access.” If one of these phrases is found, and if the reported line number matches the line number of the commented buffer access in the corresponding test case file, then a detection is recorded for that test case file.

After all of the results files have been processed, *tabulate_archer_results.pl* prints a summary report which can be redirected into a file (e.g., *archer_db_info.txt*). The summary report contains one row for each test case; each row comprises four columns, where each column represents one of the four Magnitude variations (-ok, -min, -med, -large). Each column contains a “1” if ARCHER reported a buffer overflow for that test case file, or a “0” if it did not. A few lines from this report are shown below:

```
Test case   OK    MIN   MED   LARGE
basic-00001 0     1     1     1
basic-00002 0     1     1     1
basic-00003 0     1     1     1
    .
    .
    .
basic-00013 0     0     0     0
basic-00014 0     0     0     0
basic-00015 0     1     1     1
```

This summary report excerpt indicates that for test cases basic-00001, basic-00002, basic-00003, and basic-00015, ARCHER reported buffer overflows in each of the -min, -med, and -large versions, but not in the -ok versions. ARCHER did not report buffer overflows in any Magnitude version of test cases basic-00013 or basic-00014. This summary report gets imported into the database, as described in the Database Description section later in this chapter.

BOON Evaluation

The script *Boon_script.sh* (see Appendix 1) runs BOON against each test case file in the suite, directing the results to a "Boon_results" directory. The results directory contains one results output file for each test case input file; the results file name is the same as the test case file name with the ".c" extension removed.

Appendix 4 contains the BOON output file *basic-00045-min*; this file illustrates how a BOON error report contains a descriptive phrase and the name of the buffer and function in which the error occurs; subsequent lines provide more details of the offending access. It does not, however, report a line number. The *tabulate_boon_results.pl* program (see Appendix 1) parses through the results files looking for the phrase "buffer overflow." If it is found, a detection is recorded for that test case file. Since BOON does not report line numbers for the errors, *tabulate_boon_results.pl* cannot validate that the reported error corresponds to the commented buffer access in the test case file; it assumes that any reported error is a valid detection.

After all of the results files have been processed, *tabulate_boon_results.pl* prints a summary report which can be redirected into a file (e.g., *boon_db_info.txt*). The summary report has the same format as the ARCHER summary report described in the previous section. This summary report gets imported into the database, as described in the Database Description section later in this chapter.

PolySpace Evaluation

The script *PolySpace_script.sh* (see Appendix 1) runs PolySpace against each test case file in the suite, directing the results to a "PolySpace_results" directory. The contents of the PolySpace results directory are more complex than those of the other tools, containing at the first level a directory for each test case input file; the directory name is the same as the test case file name with the ".c" extension removed. There are various files and other directories under the first-level directory; the only file needed to evaluate the results is a lower-level file named *polyspace_RTE_View.txt.* For ease of processing and to reduce storage requirements, I copied this file to the first-level directory for each test case, and removed all other output.

Appendix 4 contains the PolySpace output file *basic-00001-min*; this file is best viewed in columnar format, as when loaded into a spreadsheet. The results are hierarchical, with sections for higher-level functions appearing first, and those for lower-level functions later. The function sections are indented, or preceded with a varying number of vertical bars, to reflect the function call depth. Capitalized abbreviations denote error reports; Table 26 describes the abbreviations related to buffer overflows. These reports are also classified as red, orange, or green, as indicated by a "1" appearing in the R, O, or Gn column, respectively. A red report is a certain error, or one that occurs on each execution of the statement. An orange report is a possible error, or one that may occur on some executions but not on others. Green indicates that the statement is always safe on every execution. A line number is also provided with the error reports.

Abbreviation	**Description**
COR	illegal pointer access to variable or structure field (cannot use a pointer to one object to access into another)
IDP	dereferenced pointer is out of bounds
NIP	dereferencing a non-initialized pointer
OBAI	out of bounds array index

Table 26. PolySpace Error Abbreviations

The *tabulate_polyspace_results.pl* program (see Appendix 1) parses through the results files looking for any of the known abbreviations that indicate a buffer overflow. If one of these abbreviations is found, if the color is red or orange, and if the reported line number matches the line number of the commented buffer access in the corresponding test case file, then a detection is recorded for that test case file. The COR abbreviation has more than one meaning; therefore, if this abbreviation is found, the phrase "illegal pointer access" must also appear in order to count it as a detection.

An exception to the matching line number requirement is the case where the buffer overflow occurs in a C library function. PolySpace reports the error in the library function itself, rather than on the line in the test case file where the function is called. Therefore, *tabulate_polyspace_results.pl* examines the test case file to determine if it calls a C library function. If so, it looks for the appropriate error abbreviations reported within that function section of the results file. If any are found, a detection is counted, irrespective of the associated line number.

After all of the results files have been processed, *tabulate_polyspace_results.pl* prints a summary report which can be redirected into a file (e.g., *polyspace_db_info.txt*). The summary report has the same format as the ARCHER summary report described in a previous section. This summary report gets imported into the database, as described in the Database Description section later in this chapter.

When first run against the basic-00006 test case (Data Type: wide character), PolySpace could not complete its analysis; it reported a syntax error when processing the header file *wctype.h*. A PolySpace representative investigated the problem and suggested a work-around (additional command line options to ignore constant overflows and to specify a particular include file). The revised script *PolySpace_script_basic-00006.sh* (see Appendix 1) was used to rerun PolySpace against this test case only, and full results were obtained.

Splint Evaluation

The script *Splint_script.sh* (see Appendix 1) runs Splint against each test case file in the suite, directing the results to a "Splint_results" directory. The results directory contains one results output file for each test case input file; the results file name is the same as the test case file name with the ".c" extension removed.

Appendix 4 contains the Splint output file *basic-00001-min*; this file illustrates how a Splint error report contains the source file name, line number where the error occurred, and a descriptive phrase, followed by additional lines of detail about the offending access. The *tabulate_splint_results.pl* program (see Appendix 1) parses through the results files looking for the phrases "Possible out-of-bounds." If found, and if the reported line number matches the line number of the commented buffer access in the corresponding test case file, then a detection is recorded for that test case file.

After all of the results files have been processed, *tabulate_splint_results.pl* prints a summary report which can be redirected into a file (e.g., *splint_db_info.txt*). The summary report has the same format as the ARCHER summary report described in a previous section. This summary report gets imported into the database, as described in the Database Description section later in this chapter.

UNO Evaluation

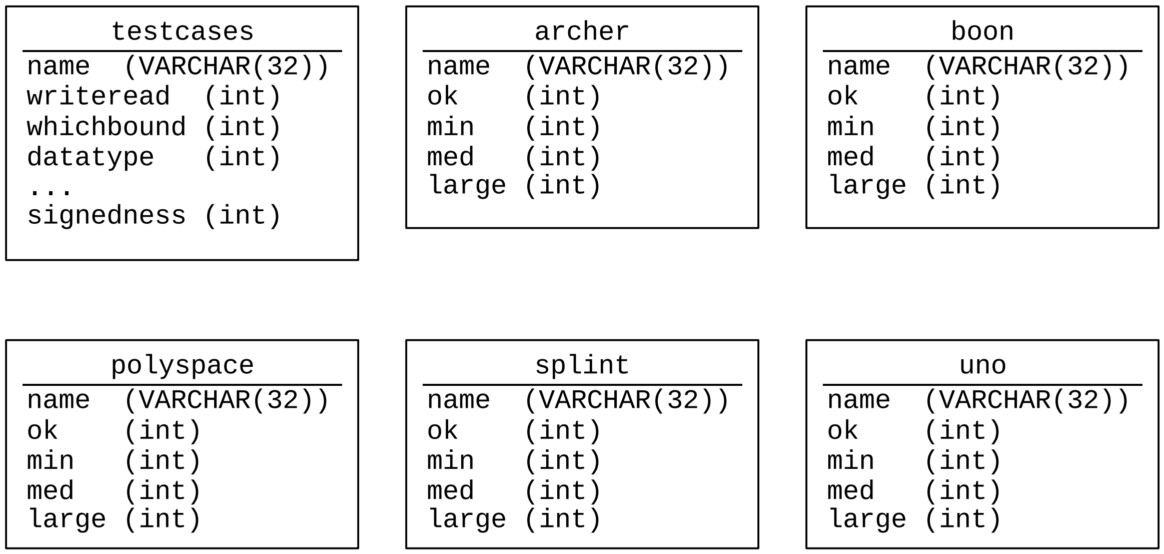
The script *Uno_script.sh* (see Appendix 1) runs UNO against each test case file in the suite, directing the results to a "Uno_results" directory. The results directory contains one results output file for each test case input file; the results file name is the same as the test case file name with the ".c" extension removed.

Appendix 4 contains the UNO output file *basic-00001-min*; this file illustrates how an UNO error report sometimes contains the source file name, line number where the error occurred, and a descriptive phrase, followed by additional details about the offending access. Some error reports have a different format, in which the source file name and line number appear on the line following the initial error report. The *tabulate_uno_results.pl* program (see Appendix 1) parses through the results files looking for the phrases “array index error,” “array index can be negative,” and “array index can exceed upper bound.” If found, and if the reported line number matches the line number of the commented buffer access in the corresponding test case file, then a detection is recorded for that test case file.

After all of the results files have been processed, *tabulate_uno_results.pl* prints a summary report which can be redirected into a file (e.g., *uno_db_info.txt*). The summary report has the same format as the ARCHER summary report described in a previous section. This summary report gets imported into the database, as described in the Database Description section below.

Database Description

The database developed for this evaluation has a very simple schema, as illustrated in Figure 4, and was implemented using MySQL ® (MySQL, 2005). The file *setupDBtables.sql* (see Appendix 1) contains all of the commands necessary to set up the database tables and import the data.



```
testcases
name  (VARCHAR(32))
writeread  (int)
whichbound (int)
datatype   (int)
...
signedness (int)

archer
name  (VARCHAR(32))
ok    (int)
min   (int)
med   (int)
large (int)

boon
name  (VARCHAR(32))
ok    (int)
min   (int)
med   (int)
large (int)

polyspace
name  (VARCHAR(32))
ok    (int)
min   (int)
med   (int)
large (int)

splint
name  (VARCHAR(32))
ok    (int)
min   (int)
med   (int)
large (int)

uno
name  (VARCHAR(32))
ok    (int)
min   (int)
med   (int)
large (int)
```

Figure 4. Database Schema

The *gen_testcase_db_info.pl* program (see Appendix 1) collects the taxonomy classification information from each test case file, and prints it out in the format required for direct import into the database. Each row of the output becomes a row in the *testcases* table, with the test case name (e.g., basic-00001) appearing in the *name* field, and the integer value for each attribute appearing in the appropriate column for that attribute. Only the Magnitude attribute does not have a column in this table, since all four values are used in test case. These four values are called out individually in the tool result tables.

Each tool has a corresponding table in the database to hold its results. As described in the earlier sections of this chapter, a tabulation program for each tool prints out a summary report indicating whether the tool reported a buffer overflow in each of the test case files. The Evaluating ARCHER section contains an example illustrating the format of this summary report, which is suitable for direct importation into the tool's database table. Each row of the report becomes a row in the tool's table, with the test case name (e.g., basic-00001) appearing in the *name* field, and the detection integer value ("1" for detection, "0" otherwise) appearing in the appropriate Magnitude column for a given test case variation.

The *setupDBtables.sql* script creates all six of the database tables, and then imports data from the various text files produced by the previously described Perl programs: *testcase_db_info.txt* goes into the testcases table, *archer_db_info.txt* goes into the archer table, and so on.

Chapter 5 Results and Analysis

This chapter explains how the tool results were analyzed to evaluate the tools' performance. It describes the metrics used to quantify the tools' performance, as well as the queries run against the database to obtain the required data. It offers some general observations, and compares the self-described capabilities of the tools to their actual performance.

Magnitude Analysis

The first analysis evaluated whether the tools performed differently for different overflow sizes within a test case. The expectation was that static analysis tools should perform the same regardless of overflow size. If a tool contained an off-by-one error in its model for a C library function, however, it might fail to detect a one-byte overflow, or might report a false alarm on a legitimate access to the full extent of the buffer.

The *query_minmedlarge_same* directory of ".sql" files contains one file for each tool evaluated (e.g., *archer.sql*). The files contain the commands and queries necessary to answer the Magnitude performance question. For each tool, the database is queried to check for test cases where the tool detected an overflow in at least one of the three bad versions (-min, -med, or –large), but not in all three. The batch file *run_query.bat* (see the BAT Files section of Appendix 1) runs each of the five SQL command files in the current directory against the database, saving the results back into the same directory.

The ARCHER version of all SQL command files is presented in the SQL Files section of Appendix 1. Simply substitute the other tool names for “archer” to obtain the commands used for the other tools.

This analysis confirmed the hypothesis, demonstrating that all five tools performed the same regardless of overflow size. If a tool detected any one of the real overflows within a test case (one byte, eight bytes, or 4096 bytes), then it detected all of them. This simplifies all further analysis, allowing us to condense those three results to one result representing performance on the “bad” programs as a whole.

Overall Performance

Three overall performance metrics for each tool were computed from the results (see Table 27). The *detection rate* indicates how well a tool detects the known buffer overflows in the bad programs (Magnitude values 1 byte, 8 bytes, and 4096 bytes), while the *false alarm rate* indicates how often a tool reports a buffer overflow in the patched programs (Magnitude value “none”). The *confusion rate* indicates how well a tool can distinguish between the bad and patched programs. When a tool reports a detection in both the patched and bad versions of a test case, the tool has demonstrated “confusion.” A tool may have a high detection rate, but if it also reports a false alarm on every patched program for which it detects the overflow in the corresponding bad programs, then it will have a high confusion rate. A high confusion rate makes the tool less useful, as users will become quickly frustrated with too many false alarms, and with alarms that persist after the overflow has been fixed. The formulas used to compute these three metrics are shown below:

$$\text{detection rate} = \frac{\text{(\# of test cases where tool reports overflow in bad version)}}{\text{(\# of test cases tool evaluated)}}$$

$$\text{false alarm rate} = \frac{\text{(\# of test cases where tool reports overflow in patched version)}}{\text{(\# of test cases tool evaluated)}}$$

$$\text{confusion rate} = \frac{\begin{array}{c}\text{(\# of test cases where tool reports overflow in bad version}\\ \text{AND reports overflow in patched version)}\end{array}}{\text{(\# of test cases where tool reports overflow in bad version)}}$$

The query directories *query_overall_detection*, *query_overall_falsealarm*, and *query_overall_confusion* contain all the SQL command files used to compute the three overall performance metrics (see Appendix 1 for the ARCHER example).

Tool	Total Detections	Detection Rate	Total False Alarms	False Alarm Rate	Total Confusions	Confusion Rate
ARCHER	264	90.72%	0	0.00%	0	0.00%
BOON	2	0.69%	0	0.00%	0	0.00%
PolySpace	290	99.66%	7	2.41%	7	2.41%
Splint	164	56.36%	35	12.03%	35	21.34%
UNO	151	51.89%	0	0.00%	0	0.00%

Table 27. Overall Performance on Basic Test Suite (291 Test Cases)

As seen in Table 27, ARCHER and PolySpace both have high detection rates, exceeding ninety percent. PolySpace's detection rate is nearly perfect, having missed only one out of the 291 possible detections. PolySpace produced seven false alarms, whereas ARCHER produced none. Splint and UNO each detected roughly half of the overflows; Splint, however, produced a substantial number of false alarms, while UNO produced none. Splint also exhibited a fairly high confusion rate: in over twenty percent of the cases where it properly detected an overflow, it also reported an error in the patched program. PolySpace's confusion rate was substantially lower, while the other three tools had no confusions. BOON's detection rate across the test suite was extremely low.

It is important to note that it was not necessarily the design goal of each tool to detect every possible buffer overflow. BOON, for instance, focuses only on the misuse of string manipulation functions, and therefore could not be expected to detect any of the other overflows. It is also important to realize that these performance rates are not necessarily predictive of how the tools would perform on buffer overflows in actual, released code. The basic test suite used in this evaluation was designed for diagnostic purposes, and the taxonomy coverage exhibited is not representative of that which would be seen in real-world buffer overflows.

In fact, the overall performance rates measured in this evaluation differ significantly from those reported in the Zister evaluation, which ran against excerpts from actual, open-source code (Zitser, 2003). Figure 5 presents a plot of detection rate vs. false alarm rate for each tool, similar to the one presented in Zitser's thesis. Each tool's performance is plotted with a single data point representing its achieved detection rate and false alarm rate. The diagonal line represents the hypothetical performance of a random guesser that decides with equal probability if each commented buffer access in the test programs results in an overflow or not. As Zitser points out, we would hope that a tool's performance would be better than what we could achieve with random guessing. The difference between a tool's detection rate and the random guesser's is only statistically significant if it lies more than two standard deviations away from the random guesser line at the same false alarm rate. In this evaluation, every tool except BOON performs significantly better than a random guesser, whereas in the Zitser evaluation, only PolySpace was significantly better. This difference in performance reflects the difference in taxonomy coverage between the two test suites, and the fact that the tools detect some types of overflows better than others.

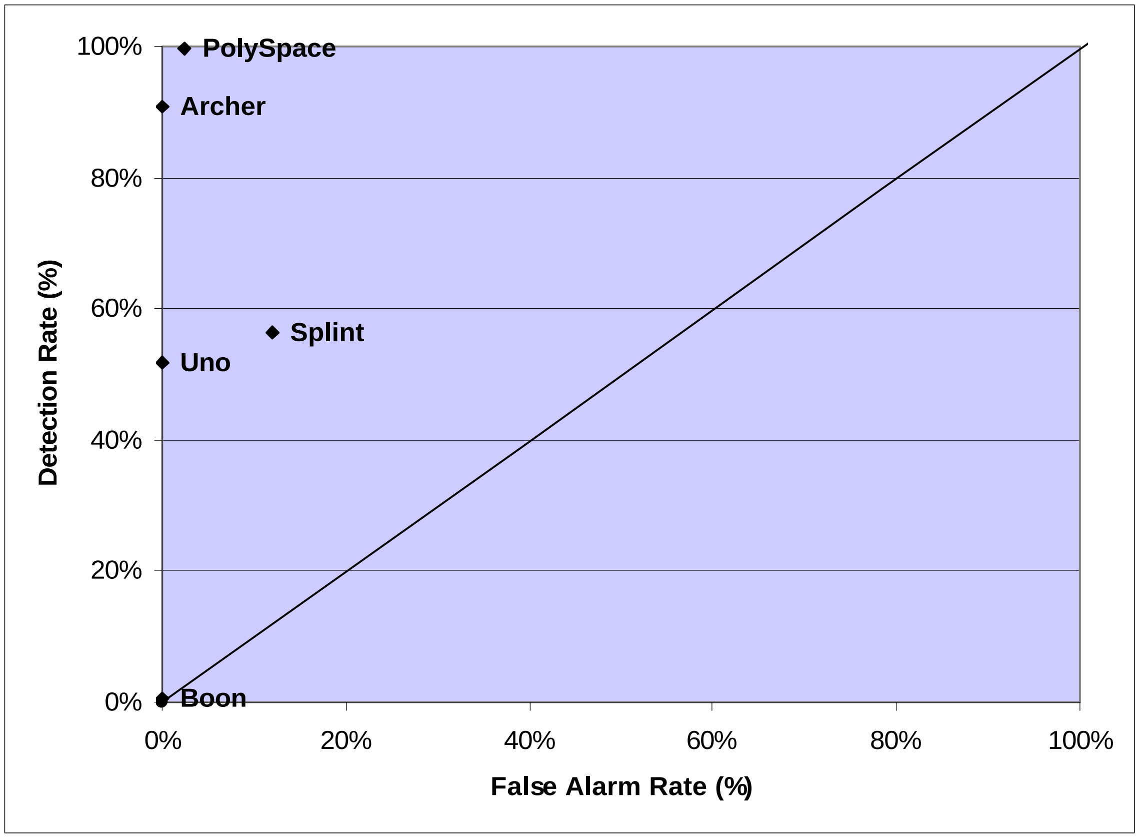


Figure 5. False Alarm and Detection Rates per Tool

Analysis by Attribute Value

In addition to computing the overall performance metrics, looking at the performance by attribute value provides other useful information. For instance, we can compare how the tools perform on buffer overflows with stack-based buffers vs. heap-based buffers, index-based accesses vs. pointer-based accesses, different types of loops, C library function calls, etc. This allows a diagnostic look into the tools' performance, in order to understand the strengths and weaknesses of their various approaches.

The query directories *query_attrval_detections, query_attrval_falsealarms,* and *query_attrval_confusions* contain all the SQL command files necessary to look at detection rates, false alarm rates, and confusion rates by attribute value (see Appendix 1 for the ARCHER examples). As described previously, *run_query.bat* was used to execute all the queries and save the results. Another Perl program, *gen_table_each_value.pl* (see Appendix 1), processes the query output to put it into tabular format, which may then be imported into a spreadsheet for easier viewing.

There are several problems with trying to analyze the data in this format, the first being that it is very difficult to look at such a large table full of numbers and make sense out of it. Secondly, the performance rates presented can lead to false impressions; while a detection rate of 100 percent for a particular attribute value may appear impressive, there may have been only one test case in the suite classified with that attribute value. Lastly, this format does not enable the discovery of what attribute values were held in common among the test cases for which a tool missed detections or reported false alarms. This makes it difficult to form useful generalizations about the tools' performance.

For these reasons, the attribute value analysis results are not presented here. The next section describes an alternative analysis that proved more useful.

Analysis by Test Case and Non-baseline Attribute Values

What we would really like to know about a tool's performance is what it got wrong and why. The best way to see this from the evaluation results is to arrange the data in a table such that one can read across the row for a test case. This allows one to see which tools missed detecting the overflows in the bad versions, which tools reported false alarms in the patched version, and what were the non-baseline attribute values for that test case. Inspecting the results in this format enables us to discover patterns of behavior across and within tool results, and to see what attributes a set of test cases has in common that might explain the behavior. We can also determine if some combination of tools would provide comprehensive coverage of varied buffer overflows, or if there are holes in coverage that none of the tools address.

Such a chart was prepared and used for the analysis portion of this project. For incorporation into this document, the table has been broken down into multiple tables in Appendix 5; all of the test cases constructed to embody the possible values for a particular attribute appear together as a set in one table. The last column in those tables contains the descriptions of only the non-baseline attribute values for each test case. These descriptions were generated by *gen_testcase_desc.pl* (shown in Appendix 1). The following sections provide general observations and per-tool discussions of these results.

General Observations

Some general observations can be made from inspecting the results as a whole. Rather than appearing as a random scattering of “X’s” in the results tables, missed detections and false alarms tend to group in certain attribute sets and follow logical patterns. For instance, it is often the case that if one tool missed a detection on a particular test case, some of the other tools missed it as well. For five test cases, basic-00183 (Taint: process environment) and basic-288 through basic-291 (Signed/Unsigned Mismatch: yes), only PolySpace did not miss detections in the bad programs. No attribute sets and no individual test cases have perfect detections across all five tools (mainly due to the BOON results), but eight attribute sets contain no false alarms at all. Without the BOON results, looking exclusively at the results of the other four tools, three of the attribute sets and 108 individual test cases had perfect detections across the four tools.

Since PolySpace missed only one detection, and three of the other tools did detect the overflow in that test case, one could obtain perfect detection across the evaluation test suite by using PolySpace as the primary authority, and using one of the other tool’s results only when PolySpace did not detect an overflow. ARCHER or UNO would be the best choice for this, as neither of those would bring any additional false alarms with them.

Similarly combining the ARCHER and Splint results would produce a detection rate of ninety-eight percent; ARCHER missed twenty-seven detections, and Splint detected all but five of those (the five missed being those five that were only detected by PolySpace). Unfortunately, Splint would also bring along its thirty-five false alarms.

Execution times for the five tools were also measured, and are compared Table 28. PolySpace's highest detection rate comes at the cost of a dramatically higher execution time than that of any other tool. ARCHER demonstrated both the second highest detection rate and the second highest execution time. Splint and UNO, with detection rates in the middle of the pack, had the two fastest execution times. BOON's slightly longer execution time did not result in a higher detection rate.

Tool	**Total Time (secs)**	**Average Time per Test Case (secs)**
ARCHER	288	0.247
BOON	73	0.063
PolySpace	200,820 (55.78 hrs)	172.526
Splint	24	0.021
UNO	27	0.023

Table 28. Tool Execution Times

The following sections discuss each tool's performance, especially as compared to the tools' advertised capabilities.

ARCHER Analysis

ARCHER's strategy is to find as many bugs as possible while minimizing the number of false alarms. It claims to be inter-procedural, path-senstive, context-sensitive, and aware of pointer aliases. It performs a fully-symbolic, bottom-up data flow analysis, while maintaining symbolic constraints between variables (handled by a linear constraint solver). ARCHER checks array accesses, pointer dereferences, and function calls that take a pointer and size. It is hard-coded to recognize and handle a small number of memory-related functions, such as `malloc`. (Xie et al., 2003)

The authors admit to many limitations of the tool, many of which they would like to address in future versions. ARCHER does not handle function pointers, and imposes a five second limit on the analysis of any particular function. Furthermore, it loses precision after function calls, as it does not perform a proper inter-procedural side effects analysis and has a very simple alias analysis. It does not understand C library string functions, nor does it keep track of null pointers or the length of null-terminated strings. Its linear constraint solver is limited to handling at most two non-linearly related variables. Finally, some of the techniques it uses to reduce false alarms will necessarily result in missed detections. For instance, if no bounds information is known about a variable used as an array index, ARCHER assumes the array access is trusted and does not issue a warning. Similarly, it only performs a bounds check on the length and offset of a pointer dereference if bounds information is available; otherwise it remains quiet and issues no warning. (Xie et al., 2003)

With close to a ninety-one percent detection rate and no false alarms, ARCHER stands up well to its stated strategy. Most of its twenty-seven missed detections are easily explained by its admitted limitations. Twenty of these were inter-procedural, and this seems to be ARCHER's main weakness. The twenty inter-procedural misses include fourteen cases that call C library functions. While the authors admit to ignoring string functions per se, one might have expected `memcpy` to be one of the few hard-coded for special handling. The other inter-procedural misses include cases involving shared memory (basic-00013), function pointers (which ARCHER does not handle – basic-00064), recursion (which may be affected by loss of precision after function calls – basic-00065), and simple cases of passing a buffer address through one or two functions (one or two levels of address aliasing – basic-00014, basic-00054, and basic-00056). One might have expected ARCHER to detect the latter three; perhaps the fact that the functions did not also take a size argument precluded ARCHER from analyzing them.

Of the remaining seven misses, three involve function return values (basic-00038, basic-00043, and basic-00180); these may have been affected by ARCHER's lack of proper inter-procedural side effects analysis. Two other cases depend on array contents (basic-00039 and basic-00044); while ARCHER does keep track of constraints on some variable values, presumably it does not model all memory contents, and therefore would be expected to miss these. The remaining two cases again involved function pointers (basic-00071) and recursion (basic-00072).

While some of the missed detections occurred on cases whose features may not be widespread in real code (such as recursion), the use of C library functions and other inter-procedural mechanisms are surely prevalent. Indeed, ARCHER's poor showing in the Zitser evaluation is directly attributable to the preponderance of these features (Zitser, 2003). ARCHER detected only one overflow in Zitser's evaluation, which was based on overflows in real code. Of the thirteen programs for which ARCHER reported no overflows, twelve contained buffer overflows that would be classified according to this evaluation's taxonomy as having inter-procedural scope, and nine of those involve calls to C library functions. To perform well against a body of real code, ARCHER needs to handle C library functions and other inter-procedural buffer overflows well.

BOON Analysis

BOON's analysis is flow-insensitive and context-insensitive for scalability and simplicity, focusing exclusively on the misuse of string manipulation functions. The authors intentionally sacrificed precision for scalability. BOON will not detect overflows caused by using primitive pointer operations; it ignores pointer dereferencing, pointer aliasing, arrays of pointers, function pointers, and unions. The authors expect a high false alarm rate due to the loss of precision resulting from the compromises made for scalability. (Wagner et al., 2000)

In this evaluation, BOON properly detected two overflows out of fourteen that might have been expected, with no false alarms. Here it should be noted that the automated tabulation program, *tabulate_boon_results*, counted four detections and two false alarms. As described previously, however, BOON does not include line numbers with its error reports, so the tabulation program is forced to assume that any error report pertains to the commented buffer accesses in the test case files. Since BOON had so few detections and false alarms, it was practical to inspect them manually; this inspection revealed that two of the detections (basic-00013 and basic-00181), and both of the false alarms (again, basic-00013 and basic-00181) were reported on buffers other than the one accessed or overflowed in the test case program. Therefore, those two detections and two false alarms were removed from the final results

The two overflows detected by BOON comprise one of the two cases that call `strcpy` (basic-00045), and once case that calls `fgets` (basic-00182). BOON failed to detect the second case that calls `strcpy` (basic-00046), all six cases that call `strncpy` (basic-00047 through basic-00052), the case that calls `getcwd` (basic-00183), and all four cases that call `memcpy` (basic-00288 through basic-00291). Despite the heavy use of C library string functions in Zitser's model programs, BOON achieved only two detections in that evaluation as well (Zitser, 2003). BOON appears to be the least useful tool included in these evaluations.

PolySpace Analysis

PolySpace is the only commercial tool included in this evaluation; the details of its methods and implementation are proprietary. We do know, however, that its methods are based in part on several published works, including: symbolic analysis, or abstract interpretation (Cousot, 1976); escape analysis, for determining inter-procedural side effects (Deutsch, 1997); and inter-procedural alias analysis for pointers (Deutsch, 1994). It can detect dead or unreachable code. Like other tools, it may lose precision at junctions in code where previously branched paths rejoin, a compromise necessary to keep the analysis tractable.

PolySpace missed only one detection in this evaluation (basic-00179), which was a case involving a signal handler. The PolySpace output for this test case labeled the signal handler function with the code “UNP,” meaning “unreachable procedure.” Presumably PolySpace employs techniques similar to those described by Cousot, in which “only connected flowcharts are considered.” (Cousot, 1976) Since there is no direct invocation of the signal handler function from the `main` function, the signal handler's flow graph would not be connected to that of the `main` function. Interestingly, PolySpace did detect the overflow in the example involving threads (basic-00177). This case is quite similar; a function call inside `main` registers the thread handler function, but does not invoke it directly. This would leave the thread handler function similarly disconnected from the `main` function. It appears that PolySpace may have special handling that recognizes the reachability of thread functions, but not that of signal handlers.

PolySpace reported seven false alarms across the test suite, all of them qualified as orange, or possible overflows on some executions. These included all four of the Taint cases (basic-00180 through basic-00183), the shared memory case (basis-00013), the case that uses array contents for the buffer address (basic-00044), and one of the cases that calls `strcpy` (basic-00045). The false alarm on the array contents case is not too surprising, as it is impractical for a tool to track the contents of every location in memory. PolySpace does not, however, report a false alarm on the other two cases involving array contents (index - basic-00039, and length - basic-00052). The other six false alarms are on test cases that in some way involve calls to C library or O/S-specific function calls. Not all such cases produced false alarms, however. For instance, only one out of the two `strcpy` cases produced a false alarm: the one that copies directly from a constant string (e.g., "AAAA"). Without more insight into the PolySpace implementation, it is difficult to explain why these particular cases produced false alarms.

PolySpace did not perform as well in the Zitser evaluation as it did in this evaluation (Zitser, 2003). Again, without more knowledge of the tool's internals, it is difficult to know why its detection rate was lower in Zitser's evaluation. Presumably the additional complexity of the real code forced PolySpace into situations where approximations had to be made to keep the problem tractable, but at the expense of precision. The majority of the false alarms it reported in the Zitser evaluation were on overflows similar to those for which it reported false alarms in this evaluation: those involving memory contents and C library functions.

PolySpace's sophisticated analysis techniques clearly result in a superior detection rate on these test cases as compared to the other tools. This performance does not come without additional cost, however, both in money and in time. While the other tools completed their analyses across the entire test suite on the order of seconds or minutes, PolySpace ran for nearly two days and eight hours, averaging close to three minutes of analysis time per test case file. This long execution time may make it difficult to incorporate into the development cycle.

Splint Analysis

Splint employs "lightweight" static analysis and heuristics that are practical, but neither sound nor complete. Like many other tools, it trades off precision for scalability. It implements limited flow-sensitive control flow, merging possible paths at branch points. Splint uses heuristics to recognize loop idioms and determine loop bounds without resorting to more costly, and also more accurate, abstract evaluation. An annotated C library is provided, but the tool relies on the user to properly annotate all other functions in order to have any kind of inter-procedural analysis. Splint exhibited high false alarm rates in the developers' own tests. (Evans & Larochelle, 2002; Larochelle & Evans 2001)

The basis test suite used in this evaluation was not annotated for Splint for two reasons. First, it is a more fair comparison of the tools to run them all against the same source code, with no special accommodations for any particular tool. Second, expecting developers to completely and correctly annotate their programs for Splint seems as unrealistic as expecting them to write flawless code in the first place.

Not surprisingly, Splint exhibited the highest false alarm rate of any tool on the basic test suite. Many of the thirty-five false alarms are attributable to inter-procedural cases; cases involving increased complexity of the index, address, or length; and more complex containers and flow control constructs. The vast majority, 120 out of 127, of missed detections are attributable to loops. It misses detections in all of the non-standard for loop cases (both discrete and continuous), as well as in most of the other continuous loop cases. The only continuous loop cases it detects correctly are the standard for loops, and it also produces false alarms on nearly all of those. In addition, it misses the lower bound case (basic-00003), the "cond" case of local flow control (basic-00061), the taint case that calls `getcwd` (basic-00183), and all four of the signed/unsigned mismatch cases (basic-00288 through basic-291).

While Splint's detection rate was similar in this evaluation and the Zitser evaluation, its false alarm rate was much higher in the latter (Zitser, 2003). Again, this is presumably because code that is more complex results in more complex situations where precision is sacrificed in the interest of scalability, with the loss of precision leading to increased false alarms.

Splint’s weakest area in my evaluation was its loop handling. Enhancing its loop heuristics to more accurately recognize and handle non-standard for loops, as well as continuous loops of all varieties, would significantly improve its performance. The high confusion rate may be a source of frustration to developers, and may act as a deterrent to its use. Improvements in this regard are also important.

UNO Analysis

UNO is an acronym for *u*ninitialized variables, *n*ull-pointer dereferencing, and *o*ut-of-bounds array indexing, which are the three types of problems it is designed to address. Extensibility was another main goal, the design allowing others to define and implement additional checks to enhance the system. UNO implements a two-pass analysis; the first pass performs intra-procedural analysis within each function, while the second pass performs a global analysis across the entire program. It appears that the second pass focuses only on global pointer dereferencing, in order to detect null pointer usage; therefore, UNO would not seem to be inter-procedural with respect to out-of-bounds array indexing. UNO determines path infeasibility, and uses this information to suppress warnings and take shortcuts in its searches. It handles constants and scalars, but not computed indices (expressions on variables, or function calls), and easily loses precision on conservatively-computed value ranges. It does not handle function pointers, nor does it attempt to compute possible function return values. Lastly, UNO does not handle the `setjmp/longjmp` construct. (Holzmann, 2002)

UNO produced no false alarms in the basic test suite, but did miss nearly half of the possible detections (140 out of 291), most of which would be expected based on the tool's description. This included every inter-procedural case, every container case, nearly every index complexity case (the only one it detected was the simple variable), every address and length complexity case, every address alias case, the function and recursion cases, every signed/unsigned mismatch, nearly every continuous loop, and a small assortment of others. It performed well on the various data types, index aliasing, and discrete loops.

Given the broad variety of detections missed in the basic test suite, it is not surprising that UNO exhibited the worst performance out of the five tools in Zitser's evaluation (Zitser, 2003). UNO appears to have too many holes in its coverage to be a truly useful tool, especially in light of the other tools that performed better in the evaluations, and whose deficiencies are narrower and perhaps more easily remedied.

Chapter 6 Summary and Conclusions

This project evaluated five static analysis tools using a diagnostic test suite to determine their strengths and weaknesses in detecting a variety of buffer overflow flaws in C code. The diagnostic test suite comprises small test cases that vary slightly from a simple baseline. The performance observed is, therefore, a "best case" indication of how the tools perform, and cannot be generally extrapolated to determine performance on more complex, real-world programs. We may predict with some confidence that if a tool fails to detect a particular type of overflow in the basic test suite, it will also fail to detect similar overflows in more complex, real-world code; we may not conversely predict that if a tool does detect a particular type of overflow in the basic test suite, it will also detect similar overflows in more complex, real-world code. The results of this evaluation may be used to help explain tool performance on real code, such as in the Zitser evaluation (Zitser, 2003), and to provide guidance for improving the tools.

PolySpace demonstrated a superior detection rate on the basic test suite, missing only one out of a possible 291 detections. It may benefit from improving its treatment of signal handlers, and reducing both its false alarm rate (particularly for C library functions) and execution time. ARCHER performed quite well with no false alarms whatsoever; a few key enhancements, such as in its inter-procedural analysis and handling of C library functions, would boost its detection rate and should improve its performance on real-world code. Splint detected significantly fewer overflows and exhibited the highest false alarm rate. Improvements in its loop handling, and reductions in its false alarm rate would make it a much more useful tool. UNO had no false alarms, but missed a broad variety of overflows amounting to nearly half of the possible detections in the test suite. It would need improvement in many areas to become a very useful tool. BOON was clearly at the back of the pack, not even performing well on the subset of test cases where it could have been expected to function.

While some of the differences in tool performance between this evaluation and the Zitser evaluation can be explained directly using these diagnostic findings, some questions remain. It appears that overall tool performance may decline as code size and/or complexity increases. It would, therefore, be a useful exercise to evaluate the tools against test cases of varied size and complexity in between the basic test suite and Zitser's model programs. This might be accomplished either by starting with the basic test suite and building up in size and complexity, or by starting with Zitser's model programs and breaking down the size and complexity.

The basic test suite itself could be improved by adding test cases to cover the two attribute values not currently represented (Scope – inter-file/global, and Taint – socket), and by adding more test cases for those attribute values currently represented by only one example. The basic test suite may also be used to evaluate the performance of dynamic analysis approaches. The test suite and related tools will be made available for anyone wishing to use them.

References

Bulba and Kil3r (2000). Bypassing StackGuard and StackShield, *Phrack Magazine*, 10 (56), http://www.phrack.org/phrack/56/p56-0x05

Bush, W., Pincus, J. and Sielaff, D. (2000). A static analyzer for finding dynamic programming errors, *Software—Practice and Experience*, 30, 775-802

CERT (2004). CERT Coordination Center Advisories, http://www.cert.org/advisories/, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA

Conover, M. and Security Team (1999). w00w00 on Heap Overflows, http://www.w00w00.org/files/articles/heaptut.txt

Cousot, P. and Cousot, R. (1976). Static determination of dynamic properties of programs, *Proceedings of the 2nd International Symposium on Programming*, Paris, France, 106--130

Debian (2004). Debian Security Advisory DSA-266-1 krb5, http://www.debian.org/security/2003/dsa-266

Deutsch, A. (1994). Interprocedural may-alias analysis for pointers: beyond k-limiting, *Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation*, Orlando, Florida, 230—241

Deutsch, A. (1997). On the complexity of escape analysis, *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Paris, France, 358--371

Evans, D. and Larochelle, D. (2002). Improving security using extensible lightweight static analysis, *IEEE Software*, 19 (1), 42--51

GCC Home Page (2004). Free Software Foundation, Boston, MA, http://gcc.gnu.org/

Gregor, D., and Schupp, S. (2003). Retaining path-sensitive relations across control-flow merges, Technical Report 03-15, Rensselaer Polytechnic Institute Troy, N.Y., 17 pages

HBGary, LLC (2003). BugScan Technical White Paper, http://www.hbgary.com/text/BugScan_Technical_White_Paper.pdf, Sunnyvale, CA, 7 pages

Holzmann, G. (2002). UNO: Static source code checking for user-defined properties, Bell Labs Technical Report, Bell Laboratories, Murray Hill, NJ, 27 pages

ICAT (2004). The ICAT Metabase, http://icat.nist.gov/icat.cfm, National Institute of Standards and Technology, Computer Security Division, Gaithersburg, MD

klog (1999). The frame pointer overwrite, *Phrack Magazine*, 9 (55), http://www.tegatai.com/~jbl/overflow-papers/P55-08

Krsul, I. (1998). Software Vulnerability Analysis, PhD Thesis, Purdue University, West Lafayette, Indiana, 171 pages

Landi, W. (1992). Undecidability of static analysis, *ACM Letters on Programming Languages and Systems*, 1 (4), 323--337

Larochelle, D. and Evans, D. (2001). Statically detecting likely buffer overflow vulnerabilities, *Proceedings of the 10th USENIX Security Symposium*, Washington, DC, 177--190

Merriam-Webster, Incorporated (1997). Merriam-Webster's Collegiate Dictionary, Tenth Edition, Springfield, MA

Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., and Weaver, N. (2003). The Spread of the Sapphire/Slammer Worm, http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html

Myers, E. (1981). A precise inter-procedural data flow algorithm, *Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Williamsburg, VA, 219-230

MySQL (2005). MySQL web site, http://dev.mysql.com/, MySQL AB

PolySpace Technologies (2003). PolySpace C Developer Edition, http://www.polyspace.com/datasheets/c_psde.htm, Paris, France

PSS Security Response Team (2003). PSS Security Response Team Alert - New Worm: W32.Blaster.worm, http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/alerts/msblaster.asp, Microsoft Corporation, Redmond, WA

Ruwase, O. and Lam, M. (2004). A practical dynamic buffer overflow detector, *Proceedings of the 11th Annual Network and Distributed System Security Symposium*, San Diego, CA, 159--169

Secure Software (2003). RATS, http://www.securesw.com/download_form_rats.htm, Chantilly, VA

Security Focus (2004a). The Bugtraq mailing list, http://www.securityfocus.com/archive/1, SecurityFocus, Semantec Corporation, Cupertino, CA

Security Focus (2004b). ssldump SSLv2 Challenge Buffer Underflow Vulnerability, http://www.securityfocus.com/bid/5693/discussion/, SecurityFocus, Semantec Corporation, Cupertino, CA

Security Focus (2004c). Apache Chunked-Encoding Memory Corruption Vulnerability, http://www.securityfocus.com/bid/5033/discussion/, SecurityFocus, Semantec Corporation, Cupertino, CA

Subterrain (2004). The subterrain website, http://www.subterrain.net/overflow-papers/

Viega, J., Bloch, J. T., Kohno, Y., and McGraw, G. (2000). ITS4: A static vulnerability scanner for C and C++ code, *ACM Transactions on Information and System Security*, 5 (2), 11 pages

Wagner, D., Foster, J.S., Brewer, E.A., and Aiken, A. (2000). A first step towards automated detection of buffer overrun vulnerabilities, *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, 3--17

Wheeler, D. A. Flawfinder, http://www.dwheeler.com/flawfinder

Xie, Y., Chou, A., and Engler, D. (2003). ARCHER: Using symbolic, path-sensitive analysis to detect memory access errors, *Proceedings of the 9th European Software Engineering Conference/10th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Helsinki, Finland, 327--336

Zitser, M. (2003). Securing Software: An Evaluation of Static Source Code Analyzers, Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, 130 pages

Zitser, M., Lippmann, R., and Leek, T. (2004). Testing static analysis tools using exploitable buffer overflows from open-source code, *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Newport Beach, CA, 97--106

Appendix 1 Code

BAT files

gen_combo_dirs.bat

```
mkdir writeread_datatype
cd writeread_datatype
..\..\gen_basic_tests.pl --combo 1 3
cd ..

mkdir whichbound_datatype
cd whichbound_datatype
..\..\gen_basic_tests.pl --combo 2 3
cd ..

mkdir memloc_datatype
cd memloc_datatype
..\..\gen_basic_tests.pl --combo 4 3
cd ..

mkdir scope_datatype
cd scope_datatype
..\..\gen_basic_tests.pl --combo 5 3
cd ..

mkdir container_datatype
cd container_datatype
..\..\gen_basic_tests.pl --combo 6 3
cd ..

mkdir pointer_datatype
cd pointer_datatype
..\..\gen_basic_tests.pl --combo 7 3
cd ..

mkdir indexcomplex_datatype
cd indexcomplex_datatype
..\..\gen_basic_tests.pl --combo 8 3
cd ..

mkdir addrcomplex_datatype
cd addrcomplex_datatype
..\..\gen_basic_tests.pl --combo 9 3
cd ..
```

```
mkdir aliasaddr_datatype
cd aliasaddr_datatype
..\..\gen_basic_tests.pl --combo 11 3
cd ..

mkdir aliasindex_datatype
cd aliasindex_datatype
..\..\gen_basic_tests.pl --combo 12 3
cd ..

mkdir localflow_datatype
cd localflow_datatype
..\..\gen_basic_tests.pl --combo 13 3
cd ..

mkdir secondaryflow_datatype
cd secondaryflow_datatype
..\..\gen_basic_tests.pl --combo 14 3
cd ..

mkdir loopstructure_datatype
cd loopstructure_datatype
..\..\gen_basic_tests.pl --combo 15 3
cd ..

mkdir loopcomplex_datatype
cd loopcomplex_datatype
..\..\gen_basic_tests.pl --combo 16 3
cd ..

mkdir asynchrony_datatype
cd asynchrony_datatype
..\..\gen_basic_tests.pl --combo 17 3
cd ..

mkdir taint_datatype
cd taint_datatype
..\..\gen_basic_tests.pl --combo 18 3
cd ..

mkdir continuousdiscrete_datatype
cd continuousdiscrete_datatype
..\..\gen_basic_tests.pl --combo 21 3
cd ..
```

gen_suite_dirs.bat

```
mkdir writeread
cd writeread
..\..\gen_basic_tests.pl 1
cd ..

mkdir whichbound
cd whichbound
..\..\gen_basic_tests.pl 2
cd ..

mkdir datatype
cd datatype
..\..\gen_basic_tests.pl 3
cd ..

mkdir memloc
cd memloc
..\..\gen_basic_tests.pl 4
cd ..

mkdir scope
cd scope
..\..\gen_basic_tests.pl 5
cd ..

mkdir container
cd container
..\..\gen_basic_tests.pl 6
cd ..

mkdir pointer
cd pointer
..\..\gen_basic_tests.pl 7
cd ..

mkdir indexcomplex
cd indexcomplex
..\..\gen_basic_tests.pl 8
cd ..

mkdir addrcomplex
cd addrcomplex
..\..\gen_basic_tests.pl 9
cd ..

mkdir lencomplex
cd lencomplex
..\..\gen_basic_tests.pl 10
cd ..

mkdir aliasaddr
cd aliasaddr
..\..\gen_basic_tests.pl 11
```

```
cd ..

mkdir aliasindex
cd aliasindex
..\..\gen_basic_tests.pl 12
cd ..

mkdir localflow
cd localflow
..\..\gen_basic_tests.pl 13
cd ..

mkdir secondaryflow
cd secondaryflow
..\..\gen_basic_tests.pl 14
cd ..

mkdir loopstructure
cd loopstructure
..\..\gen_basic_tests.pl 15
cd ..

mkdir loopcomplex
cd loopcomplex
..\..\gen_basic_tests.pl 16
cd ..

mkdir asynchrony
cd asynchrony
..\..\gen_basic_tests.pl 17
cd ..

mkdir taint
cd taint
..\..\gen_basic_tests.pl 18
cd ..

mkdir continuousdiscrete
cd continuousdiscrete
..\..\gen_basic_tests.pl 21
cd ..

mkdir signedness
cd signedness
..\..\gen_basic_tests.pl 22
cd ..
```

gen_suite_here.bat

```
..\scripts\gen_basic_tests.pl 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 21 22
```

run_query.bat

```
echo archer > results.txt
c:\mysql\bin\mysql -u [user] –p[pwd] < archer.sql >> results.txt
echo boon >> results.txt
c:\mysql\bin\mysql -u [user] -p[pwd] < boon.sql >> results.txt
echo polyspace >> results.txt
c:\mysql\bin\mysql -u [user] -p[pwd] < polyspace.sql >> results.txt
echo splint >> results.txt
c:\mysql\bin\mysql -u [user] -p[pwd] < splint.sql >> results.txt
echo uno >> results.txt
c:\mysql\bin\mysql -u [user] -p[pwd] < uno.sql >> results.txt
```

C Files

basic-00001-ok.c

```
/* Taxonomy Classification: 0000000000000000000000 */

/*
 *  WRITE/READ                  0    write
 *  WHICH BOUND                 0    upper
 *  DATA TYPE                   0    char
 *  MEMORY LOCATION             0    stack
 *  SCOPE                       0    same
 *  CONTAINER                   0    no
 *  POINTER                     0    no
 *  INDEX COMPLEXITY            0    constant
 *  ADDRESS COMPLEXITY          0    constant
 *  LENGTH COMPLEXITY           0    N/A
 *  ADDRESS ALIAS               0    none
 *  INDEX ALIAS                 0    none
 *  LOCAL CONTROL FLOW          0    none
 *  SECONDARY CONTROL FLOW      0    none
 *  LOOP STRUCTURE              0    no
 *  LOOP COMPLEXITY             0    N/A
 *  ASYNCHRONY                  0    no
 *  TAINT                       0    no
 *  RUNTIME ENV. DEPENDENCE     0    no
 *  MAGNITUDE                   0    no overflow
 *  CONTINUOUS/DISCRETE         0    discrete
 *  SIGNEDNESS                  0    no
 */

/* Copyright 2004 M.I.T. */


int main(int argc, char *argv[])
{
  char buf[10];


  /*  OK  */
  buf[9] = 'A';


  return 0;
}
```

basic-00001-min.c

```
/* Taxonomy Classification: 0000000000000000000100 */

/*
 *  WRITE/READ                   0    write
 *  WHICH BOUND                  0    upper
 *  DATA TYPE                    0    char
 *  MEMORY LOCATION              0    stack
 *  SCOPE                        0    same
 *  CONTAINER                    0    no
 *  POINTER                      0    no
 *  INDEX COMPLEXITY             0    constant
 *  ADDRESS COMPLEXITY           0    constant
 *  LENGTH COMPLEXITY            0    N/A
 *  ADDRESS ALIAS                0    none
 *  INDEX ALIAS                  0    none
 *  LOCAL CONTROL FLOW           0    none
 *  SECONDARY CONTROL FLOW       0    none
 *  LOOP STRUCTURE               0    no
 *  LOOP COMPLEXITY              0    N/A
 *  ASYNCHRONY                   0    no
 *  TAINT                        0    no
 *  RUNTIME ENV. DEPENDENCE      0    no
 *  MAGNITUDE                    1    1 byte
 *  CONTINUOUS/DISCRETE          0    discrete
 *  SIGNEDNESS                   0    no
 */

/* Copyright 2004 M.I.T. */


int main(int argc, char *argv[])
{
  char buf[10];


  /*  BAD  */
  buf[10] = 'A';


  return 0;
}
```

basic-00001-med.c

```
/* Taxonomy Classification: 0000000000000000000200 */

/*
 *  WRITE/READ                 0    write
 *  WHICH BOUND                0    upper
 *  DATA TYPE                  0    char
 *  MEMORY LOCATION            0    stack
 *  SCOPE                      0    same
 *  CONTAINER                  0    no
 *  POINTER                    0    no
 *  INDEX COMPLEXITY           0    constant
 *  ADDRESS COMPLEXITY         0    constant
 *  LENGTH COMPLEXITY          0    N/A
 *  ADDRESS ALIAS              0    none
 *  INDEX ALIAS                0    none
 *  LOCAL CONTROL FLOW         0    none
 *  SECONDARY CONTROL FLOW     0    none
 *  LOOP STRUCTURE             0    no
 *  LOOP COMPLEXITY            0    N/A
 *  ASYNCHRONY                 0    no
 *  TAINT                      0    no
 *  RUNTIME ENV. DEPENDENCE    0    no
 *  MAGNITUDE                  2    8 bytes
 *  CONTINUOUS/DISCRETE        0    discrete
 *  SIGNEDNESS                 0    no
 */

/* Copyright 2004 M.I.T. */


int main(int argc, char *argv[])
{
  char buf[10];


  /*  BAD  */
  buf[17] = 'A';


  return 0;
}
```

basic-00001-large.c

```
/* Taxonomy Classification: 0000000000000000000300 */

/*
 *  WRITE/READ                 0    write
 *  WHICH BOUND                0    upper
 *  DATA TYPE                  0    char
 *  MEMORY LOCATION            0    stack
 *  SCOPE                      0    same
 *  CONTAINER                  0    no
 *  POINTER                    0    no
 *  INDEX COMPLEXITY           0    constant
 *  ADDRESS COMPLEXITY         0    constant
 *  LENGTH COMPLEXITY          0    N/A
 *  ADDRESS ALIAS              0    none
 *  INDEX ALIAS                0    none
 *  LOCAL CONTROL FLOW         0    none
 *  SECONDARY CONTROL FLOW     0    none
 *  LOOP STRUCTURE             0    no
 *  LOOP COMPLEXITY            0    N/A
 *  ASYNCHRONY                 0    no
 *  TAINT                      0    no
 *  RUNTIME ENV. DEPENDENCE    0    no
 *  MAGNITUDE                  3    4096 bytes
 *  CONTINUOUS/DISCRETE        0    discrete
 *  SIGNEDNESS                 0    no
 */

/* Copyright 2004 M.I.T. */


int main(int argc, char *argv[])
{
  char buf[10];


  /*  BAD  */
  buf[4105] = 'A';


  return 0;
}
```

basic-00002-min.c

```
/* Taxonomy Classification: 1000000000000000000100 */

/*
 *  WRITE/READ                   1    read
 *  WHICH BOUND                  0    upper
 *  DATA TYPE                    0    char
 *  MEMORY LOCATION              0    stack
 *  SCOPE                        0    same
 *  CONTAINER                    0    no
 *  POINTER                      0    no
 *  INDEX COMPLEXITY             0    constant
 *  ADDRESS COMPLEXITY           0    constant
 *  LENGTH COMPLEXITY            0    N/A
 *  ADDRESS ALIAS                0    none
 *  INDEX ALIAS                  0    none
 *  LOCAL CONTROL FLOW           0    none
 *  SECONDARY CONTROL FLOW       0    none
 *  LOOP STRUCTURE               0    no
 *  LOOP COMPLEXITY              0    N/A
 *  ASYNCHRONY                   0    no
 *  TAINT                        0    no
 *  RUNTIME ENV. DEPENDENCE      0    no
 *  MAGNITUDE                    1    1 byte
 *  CONTINUOUS/DISCRETE          0    discrete
 *  SIGNEDNESS                   0    no
 */

/* Copyright 2004 M.I.T. */


int main(int argc, char *argv[])
{
  char read_value;
  char buf[10];


  /*  BAD  */
  read_value = buf[10];


  return 0;
}
```

basic-00003-min.c

```
/* Taxonomy Classification: 0100000000000000000100 */

/*
 *  WRITE/READ                   0    write
 *  WHICH BOUND                  1    lower
 *  DATA TYPE                    0    char
 *  MEMORY LOCATION              0    stack
 *  SCOPE                        0    same
 *  CONTAINER                    0    no
 *  POINTER                      0    no
 *  INDEX COMPLEXITY             0    constant
 *  ADDRESS COMPLEXITY           0    constant
 *  LENGTH COMPLEXITY            0    N/A
 *  ADDRESS ALIAS                0    none
 *  INDEX ALIAS                  0    none
 *  LOCAL CONTROL FLOW           0    none
 *  SECONDARY CONTROL FLOW       0    none
 *  LOOP STRUCTURE               0    no
 *  LOOP COMPLEXITY              0    N/A
 *  ASYNCHRONY                   0    no
 *  TAINT                        0    no
 *  RUNTIME ENV. DEPENDENCE      0    no
 *  MAGNITUDE                    1    1 byte
 *  CONTINUOUS/DISCRETE          0    discrete
 *  SIGNEDNESS                   0    no
 */

/* Copyright 2004 M.I.T. */


int main(int argc, char *argv[])
{
  char buf[10];


  /*  BAD  */
  buf[-1] = 'A';


  return 0;
}
```

basic-00006-min.c

```
/* Taxonomy Classification: 0030000000000000000100 */

/*
 *  WRITE/READ                 0    write
 *  WHICH BOUND                0    upper
 *  DATA TYPE                  3    wchar
 *  MEMORY LOCATION            0    stack
 *  SCOPE                      0    same
 *  CONTAINER                  0    no
 *  POINTER                    0    no
 *  INDEX COMPLEXITY           0    constant
 *  ADDRESS COMPLEXITY         0    constant
 *  LENGTH COMPLEXITY          0    N/A
 *  ADDRESS ALIAS              0    none
 *  INDEX ALIAS                0    none
 *  LOCAL CONTROL FLOW         0    none
 *  SECONDARY CONTROL FLOW     0    none
 *  LOOP STRUCTURE             0    no
 *  LOOP COMPLEXITY            0    N/A
 *  ASYNCHRONY                 0    no
 *  TAINT                      0    no
 *  RUNTIME ENV. DEPENDENCE    0    no
 *  MAGNITUDE                  1    1 byte
 *  CONTINUOUS/DISCRETE        0    discrete
 *  SIGNEDNESS                 0    no
 */

/* Copyright 2004 M.I.T. */

#include <wchar.h>

int main(int argc, char *argv[])
{
  wchar_t buf[10];


  /*  BAD  */
  buf[10] = L'A';


  return 0;
}
```

basic-00010-min.c

```
/* Taxonomy Classification: 0001000000000000000100 */

/*
 *  WRITE/READ                 0    write
 *  WHICH BOUND                0    upper
 *  DATA TYPE                  0    char
 *  MEMORY LOCATION            1    heap
 *  SCOPE                      0    same
 *  CONTAINER                  0    no
 *  POINTER                    0    no
 *  INDEX COMPLEXITY           0    constant
 *  ADDRESS COMPLEXITY         0    constant
 *  LENGTH COMPLEXITY          0    N/A
 *  ADDRESS ALIAS              0    none
 *  INDEX ALIAS                0    none
 *  LOCAL CONTROL FLOW         0    none
 *  SECONDARY CONTROL FLOW     0    none
 *  LOOP STRUCTURE             0    no
 *  LOOP COMPLEXITY            0    N/A
 *  ASYNCHRONY                 0    no
 *  TAINT                      0    no
 *  RUNTIME ENV. DEPENDENCE    0    no
 *  MAGNITUDE                  1    1 byte
 *  CONTINUOUS/DISCRETE        0    discrete
 *  SIGNEDNESS                 0    no
 */

/* Copyright 2004 M.I.T. */

#include <stdlib.h>
#include <assert.h>

int main(int argc, char *argv[])
{
  char * buf;

  buf = (char *) malloc(10 * sizeof(char));
  assert (buf != NULL);

  /*  BAD  */
  buf[10] = 'A';


  return 0;
}
```

basic-00013-min.c

```
/* Taxonomy Classification: 0004100000000000000100 */

/*
 *  WRITE/READ                 0    write
 *  WHICH BOUND                0    upper
 *  DATA TYPE                  0    char
 *  MEMORY LOCATION            4    shared
 *  SCOPE                      1    inter-procedural
 *  CONTAINER                  0    no
 *  POINTER                    0    no
 *  INDEX COMPLEXITY           0    constant
 *  ADDRESS COMPLEXITY         0    constant
 *  LENGTH COMPLEXITY          0    N/A
 *  ADDRESS ALIAS              0    none
 *  INDEX ALIAS                0    none
 *  LOCAL CONTROL FLOW         0    none
 *  SECONDARY CONTROL FLOW     0    none
 *  LOOP STRUCTURE             0    no
 *  LOOP COMPLEXITY            0    N/A
 *  ASYNCHRONY                 0    no
 *  TAINT                      0    no
 *  RUNTIME ENV. DEPENDENCE    0    no
 *  MAGNITUDE                  1    1 byte
 *  CONTINUOUS/DISCRETE        0    discrete
 *  SIGNEDNESS                 0    no
 */

/* Copyright 2004 M.I.T. */

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <assert.h>
#include <stdlib.h>

int getSharedMem()
{
  return (shmget(IPC_PRIVATE, 10, 0xffffffff));
}

void relSharedMem(int memID)
{
  struct shmid_ds temp;
  shmctl(memID, IPC_RMID, &temp);
}

int main(int argc, char *argv[])
{
  int memIdent;
  char * buf;

  memIdent = getSharedMem();
  assert(memIdent != -1);
  buf = ((char *) shmat(memIdent, NULL, 0));
  assert(((int)buf) != -1);

  /*  BAD  */
  buf[10] = 'A';

  shmdt((void *)buf);
  relSharedMem(memIdent);

  return 0;
}
```

basic-00014-min.c

```
/* Taxonomy Classification: 0000100000100000000100 */

/*
 *  WRITE/READ                  0    write
 *  WHICH BOUND                 0    upper
 *  DATA TYPE                   0    char
 *  MEMORY LOCATION             0    stack
 *  SCOPE                       1    inter-procedural
 *  CONTAINER                   0    no
 *  POINTER                     0    no
 *  INDEX COMPLEXITY            0    constant
 *  ADDRESS COMPLEXITY          0    constant
 *  LENGTH COMPLEXITY           0    N/A
 *  ADDRESS ALIAS               1    yes, one level
 *  INDEX ALIAS                 0    none
 *  LOCAL CONTROL FLOW          0    none
 *  SECONDARY CONTROL FLOW      0    none
 *  LOOP STRUCTURE              0    no
 *  LOOP COMPLEXITY             0    N/A
 *  ASYNCHRONY                  0    no
 *  TAINT                       0    no
 *  RUNTIME ENV. DEPENDENCE     0    no
 *  MAGNITUDE                   1    1 byte
 *  CONTINUOUS/DISCRETE         0    discrete
 *  SIGNEDNESS                  0    no
 */

/* Copyright 2004 M.I.T. */


void function1(char * buf)
{
  /*  BAD  */
  buf[10] = 'A';
}

int main(int argc, char *argv[])
{
  char buf[10];


  function1(buf);


  return 0;
}
```

basic-00020-min.c

```
/* Taxonomy Classification: 0000020000000000000100 */

/*
 *  WRITE/READ                   0    write
 *  WHICH BOUND                  0    upper
 *  DATA TYPE                    0    char
 *  MEMORY LOCATION              0    stack
 *  SCOPE                        0    same
 *  CONTAINER                    2    struct
 *  POINTER                      0    no
 *  INDEX COMPLEXITY             0    constant
 *  ADDRESS COMPLEXITY           0    constant
 *  LENGTH COMPLEXITY            0    N/A
 *  ADDRESS ALIAS                0    none
 *  INDEX ALIAS                  0    none
 *  LOCAL CONTROL FLOW           0    none
 *  SECONDARY CONTROL FLOW       0    none
 *  LOOP STRUCTURE               0    no
 *  LOOP COMPLEXITY              0    N/A
 *  ASYNCHRONY                   0    no
 *  TAINT                        0    no
 *  RUNTIME ENV. DEPENDENCE      0    no
 *  MAGNITUDE                    1    1 byte
 *  CONTINUOUS/DISCRETE          0    discrete
 *  SIGNEDNESS                   0    no
 */

/* Copyright 2004 M.I.T. */


typedef struct
{
  char buf1[10];
  char buf2[10];
} my_struct;

int main(int argc, char *argv[])
{
  my_struct s;


  /*  BAD  */
  s.buf2[10] = 'A';


  return 0;
}
```

basic-00039-min.c

```
/* Taxonomy Classification: 0000000500000000000100 */

/*
 *  WRITE/READ                  0    write
 *  WHICH BOUND                 0    upper
 *  DATA TYPE                   0    char
 *  MEMORY LOCATION             0    stack
 *  SCOPE                       0    same
 *  CONTAINER                   0    no
 *  POINTER                     0    no
 *  INDEX COMPLEXITY            5    array contents
 *  ADDRESS COMPLEXITY          0    constant
 *  LENGTH COMPLEXITY           0    N/A
 *  ADDRESS ALIAS               0    none
 *  INDEX ALIAS                 0    none
 *  LOCAL CONTROL FLOW          0    none
 *  SECONDARY CONTROL FLOW      0    none
 *  LOOP STRUCTURE              0    no
 *  LOOP COMPLEXITY             0    N/A
 *  ASYNCHRONY                  0    no
 *  TAINT                       0    no
 *  RUNTIME ENV. DEPENDENCE     0    no
 *  MAGNITUDE                   1    1 byte
 *  CONTINUOUS/DISCRETE         0    discrete
 *  SIGNEDNESS                  0    no
 */

/* Copyright 2004 M.I.T. */


int main(int argc, char *argv[])
{
  int index_array[5];
  char buf[10];

  index_array[0] = 10;

  /*  BAD  */
  buf[index_array[0]] = 'A';


  return 0;
}
```

basic-00059-min.c

```
/* Taxonomy Classification: 0000000000001000000100 */

/*
 *  WRITE/READ                 0    write
 *  WHICH BOUND                0    upper
 *  DATA TYPE                  0    char
 *  MEMORY LOCATION            0    stack
 *  SCOPE                      0    same
 *  CONTAINER                  0    no
 *  POINTER                    0    no
 *  INDEX COMPLEXITY           0    constant
 *  ADDRESS COMPLEXITY         0    constant
 *  LENGTH COMPLEXITY          0    N/A
 *  ADDRESS ALIAS              0    none
 *  INDEX ALIAS                0    none
 *  LOCAL CONTROL FLOW         1    if
 *  SECONDARY CONTROL FLOW     0    none
 *  LOOP STRUCTURE             0    no
 *  LOOP COMPLEXITY            0    N/A
 *  ASYNCHRONY                 0    no
 *  TAINT                      0    no
 *  RUNTIME ENV. DEPENDENCE    0    no
 *  MAGNITUDE                  1    1 byte
 *  CONTINUOUS/DISCRETE        0    discrete
 *  SIGNEDNESS                 0    no
 */

/* Copyright 2004 M.I.T. */


int main(int argc, char *argv[])
{
  int flag;
  char buf[10];

  flag = 1;

  if (flag)
  {
    /*  BAD  */
    buf[10] = 'A';
  }


  return 0;
}
```

basic-00287-min.c

```
/* Taxonomy Classification: 0000000100000164000110 */

/*
 *  WRITE/READ                 0    write
 *  WHICH BOUND                0    upper
 *  DATA TYPE                  0    char
 *  MEMORY LOCATION            0    stack
 *  SCOPE                      0    same
 *  CONTAINER                  0    no
 *  POINTER                    0    no
 *  INDEX COMPLEXITY           1    variable
 *  ADDRESS COMPLEXITY         0    constant
 *  LENGTH COMPLEXITY          0    N/A
 *  ADDRESS ALIAS              0    none
 *  INDEX ALIAS                0    none
 *  LOCAL CONTROL FLOW         0    none
 *  SECONDARY CONTROL FLOW     1    if
 *  LOOP STRUCTURE             6    non-standard while
 *  LOOP COMPLEXITY            4    three
 *  ASYNCHRONY                 0    no
 *  TAINT                      0    no
 *  RUNTIME ENV. DEPENDENCE    0    no
 *  MAGNITUDE                  1    1 byte
 *  CONTINUOUS/DISCRETE        1    continuous
 *  SIGNEDNESS                 0    no
 */

/* Copyright 2004 M.I.T. */


int main(int argc, char *argv[])
{
  int init_value;
  int test_value;
  int inc_value;
  int loop_counter;
  char buf[10];

  init_value = 0;
  test_value = 10;
  inc_value = 10 - (10 - 1);

  loop_counter = init_value;
  while(loop_counter += inc_value)
  {
    /*  BAD  */
    buf[loop_counter] = 'A';
    if (loop_counter >= test_value) break;
  }

  return 0;
}
```

basic-00288-min.c

```
/* Taxonomy Classification: 0000300602130000000111 */

/*
 *  WRITE/READ                 0    write
 *  WHICH BOUND                0    upper
 *  DATA TYPE                  0    char
 *  MEMORY LOCATION            0    stack
 *  SCOPE                      3    inter-file/inter-proc
 *  CONTAINER                  0    no
 *  POINTER                    0    no
 *  INDEX COMPLEXITY           6    N/A
 *  ADDRESS COMPLEXITY         0    constant
 *  LENGTH COMPLEXITY          2    constant
 *  ADDRESS ALIAS              1    yes, one level
 *  INDEX ALIAS                3    N/A
 *  LOCAL CONTROL FLOW         0    none
 *  SECONDARY CONTROL FLOW     0    none
 *  LOOP STRUCTURE             0    no
 *  LOOP COMPLEXITY            0    N/A
 *  ASYNCHRONY                 0    no
 *  TAINT                      0    no
 *  RUNTIME ENV. DEPENDENCE    0    no
 *  MAGNITUDE                  1    1 byte
 *  CONTINUOUS/DISCRETE        1    continuous
 *  SIGNEDNESS                 1    yes
 */

/* Copyright 2004 M.I.T. */

#include <string.h>

int main(int argc, char *argv[])
{
  char src[11];
  char buf[10];

  memset(src, 'A', 11);
  src[11 - 1] = '\0';

  /*  BAD  */
  memcpy(buf, src, -1);



  return 0;
}
```

Makefiles

Makefile

```
sources := $(wildcard *.c)
exes := $(patsubst %.c,%.exe,$(sources))

%.exe : %.c
	$(CC) -lpthread -o $@ $<

.PHONY: cred
cred: CC = /home/kendra/gcc-cred-hw/install/bin/gcc -fbounds-checking
cred : $(exes)
	-$(foreach file,$(exes),./$(file) 9 10 17 4105 2>&1 |
../filter_cred.pl;)

.PHONY: gcc
gcc : CC=gcc -Wall -pedantic
gcc : $(exes)


clean :
	-rm -f $(exes)
```

Perl Files

analyze_coverage.pl

```
#!/usr/bin/perl -w

use strict;

# Copyright 2004 M.I.T.

##----------------------------------------------------------------
## File globals
##----------------------------------------------------------------
my $OK_SIZE = 0;
my $MIN_SIZE = 1;
my $MED_SIZE = 8;
my $LARGE_SIZE = 4096;

# associate taxonomy characteristics with their values in an array
my @TaxonomyInfo = (
    ['WRITE/READ', {0 => 'write', 1 => 'read'}],
    ['WHICH BOUND', {0 => 'upper', 1 => 'lower'}],
    ['DATA TYPE', {0 => 'char', 1 => 'int', 2 => 'float',
                   3 => 'wchar', 4 => 'pointer', 5 => 'unsigned int',
                   6 => 'unsigned char'}],
    ['MEMORY LOCATION', {0 => 'stack', 1 => 'heap',
                         2 => 'data segment',
                         3 => 'bss', 4 => 'shared'}],
    ['SCOPE', {0 => 'same', 1 => 'inter-procedural', 2 => 'global',
               3 => 'inter-file/inter-proc',
               4 => 'inter-file/global'}],
    ['CONTAINER', {0 => 'no', 1 => 'array', 2 => 'struct',
                   3 => 'union', 4 => 'array of structs',
                   5 => 'array of unions'}],
    ['POINTER', {0 => 'no', 1 => 'yes'}],
    ['INDEX COMPLEXITY', {0 => 'constant', 1 => 'variable',
                          2 => 'linear expr', 3 => 'non-linear expr',
                          4 => 'function return value',
                          5 => 'array contents', 6 => 'N/A'}],
    ['ADDRESS COMPLEXITY', {0 => 'constant', 1 => 'variable',
                            2 => 'linear expr', 3 => 'non-linear expr',
                            4 => 'function return value',
                            5 => 'array contents'}],
    ['LENGTH COMPLEXITY', {0 => 'N/A', 1 => 'none', 2 => 'constant',
                           3 => 'variable', 4 => 'linear expr',
                           5 => 'non-linear expr',
                           6 => 'function return value',
                           7 => 'array contents'}],
    ['ADDRESS ALIAS', {0 => 'none', 1 => 'yes, one level',
                       2 => 'yes, two levels'}],
    ['INDEX ALIAS', {0 => 'none', 1 => 'yes, one level',
                     2 => 'yes, two levels', 3 => 'N/A'}],
    ['LOCAL CONTROL FLOW', {0 => 'none', 1 => 'if', 2 => 'switch',
                            3 => 'cond', 4 => 'goto/label',
                            5 => 'longjmp', 6 => 'function pointer',
                            7 => 'recursion'}],
    ['SECONDARY CONTROL FLOW', {0 => 'none', 1 => 'if', 2 => 'switch',
                                3 => 'cond', 4 => 'goto/label',
                                5 => 'longjmp',
                                6 => 'function pointer',
                                7 => 'recursion'}],
    ['LOOP STRUCTURE', {0 => 'no', 1 => 'for', 2 => 'do-while',
                        3 => 'while', 4 => 'non-standard for',
                        5 => 'non-standard do-while',
                        6 => 'non-standard while'}],
    ['LOOP COMPLEXITY', {0 => 'N/A', 1 => 'none', 2 => 'one',
                         3 => 'two', 4 => 'three'}],
    ['ASYNCHRONY', {0 => 'no', 1 => 'threads', 2 => 'forked process',
                    3 => 'signal handler'}],
    ['TAINT', {0 => 'no', 1 => 'argc/argv',
               2 => 'environment variable', 3 => 'file read',
               4 => 'socket', 5 => 'process environment'}],
    ['RUNTIME ENV. DEPENDENCE', {0 => 'no', 1 => 'yes'}],
    ['MAGNITUDE', {0 => 'no overflow', 1 => "$MIN_SIZE byte",
                   2 => "$MED_SIZE bytes", 3 => "$LARGE_SIZE bytes"}],
    ['CONTINUOUS/DISCRETE', {0 => 'discrete', 1 => 'continuous'}],
    ['SIGNEDNESS', {0 => 'no', 1 => 'yes'}]);

# definitions for array indices related to the TaxonomyInfo array
use constant NAME_INDEX => 0;
use constant VALUES_INDEX => 1;

my @CoverageStats =
    ({0 => 0, 1 => 0},  # write/read
    {0 => 0, 1 => 0},  # which bound
    {0 => 0, 1 => 0, 2 => 0, 3 => 0, 4 => 0, 5 => 0, 6 => 0}, # data
type
    {0 => 0, 1 => 0, 2 => 0, 3 => 0, 4 => 0}, # memory location
    {0 => 0, 1 => 0, 2 => 0, 3 => 0, 4 => 0},  # scope
    {0 => 0, 1 => 0, 2 => 0, 3 => 0, 4 => 0, 5 => 0},  # container
    {0 => 0, 1 => 0},  # pointer
    {0 => 0, 1 => 0, 2 => 0, 3 => 0, 4 => 0, 5 => 0, 6 => 0},  # index
complexity
    {0 => 0, 1 => 0, 2 => 0, 3 => 0, 4 => 0, 5 => 0},  # addr
complexity
    {0 => 0, 1 => 0, 2 => 0, 3 => 0, 4 => 0, 5 => 0, 6 => 0, 7 => 0},
# length complexity
    {0 => 0, 1 => 0, 2 => 0},  # address alias
    {0 => 0, 1 => 0, 2 => 0, 3 => 0},  # index alias
    {0 => 0, 1 => 0, 2 => 0, 3 => 0, 4 => 0, 5 => 0, 6 => 0, 7 => 0},
# local control flow
    {0 => 0, 1 => 0, 2 => 0, 3 => 0, 4 => 0, 5 => 0, 6 => 0, 7 => 0},
# secondary control flow
    {0 => 0, 1 => 0, 2 => 0, 3 => 0, 4 => 0, 5 => 0, 6 => 0},  # loop
structure
    {0 => 0, 1 => 0, 2 => 0, 3 => 0, 4 => 0},  # loop complexity
    {0 => 0, 1 => 0, 2 => 0, 3 => 0},  # asynchrony
    {0 => 0, 1 => 0, 2 => 0, 3 => 0, 4 => 0, 5 => 0},  # taint
    {0 => 0, 1 => 0},  # runtime env dep
    {0 => 0, 1 => 0, 2 => 0, 3 => 0},  # magnitude
    {0 => 0, 1 => 0},  # continuous/discrete
    {0 => 0, 1 => 0}); # signedness

##----------------------------------------------------------------
## main program
##----------------------------------------------------------------

#----------------------------------------------------------------
# process arguments
#----------------------------------------------------------------
# check to see if we have the right number of arguments
if (@ARGV < 1)
{
    usage();
}

# check arguments for validity
my $dir = $ARGV[0];
-e $dir or die "Sorry.  Directory $dir does not exist.\n";
-d $dir or die "Sorry.  $dir is not a directory.\n";

#----------------------------------------------------------------
# here's the beef
#----------------------------------------------------------------
opendir(THEDIR, $dir) or die "Sorry.  Could not open $dir.\n";
my @allfiles = readdir THEDIR;
closedir THEDIR;

# loop through all files in directory
my $comment = undef;
foreach my $file (@allfiles)
{
    # only process the actual test case files; skip the rest
    if ($file =~ /(min.c|med.c|large.c|ok.c)/)
    {
        #print "Processing: $file ...\n";
    }
    else
    {
        #print "Skipped $file\n";
        next;
    }

    # open the file and look for the taxonomy classification string
    open(THISFILE, "<", "$dir/$file") or die "Sorry.  Could not open
$dir/$file.\n";
    my $foundit = 0;
    my $class_value = undef;
    while (<THISFILE>)
    {
        if ($_ =~ /Taxonomy Classification: (\d*) /)
        {
            $foundit = 1;
            $class_value = $1;
            last;
        }
    }

    # error if we couldn't find it; otherwise, process the string
    if (!$foundit)
    {
        print "FAILED to find taxonomy info in $file\n";
    }
    else
    {
        # for each digit in the string, increment a counter associated
with its value
        my $this_char;
        for (my $i = 0; $i < length $class_value; $i++)
        {
            $this_char = substr($class_value, $i, 1);
            $CoverageStats[$i]->{$this_char}++;
        }
    }
}

# now that we've processed all the files in the directory, print out
the statistics
print "\nCoverage Statistics:\n\n";
for (my $i = 0; $i < scalar @CoverageStats; $i++)
{
    printf "%-25s\n", $TaxonomyInfo[$i]->[NAME_INDEX];
    foreach my $value (sort keys %{$CoverageStats[$i]})
    {
        printf "\t%2u\t", $CoverageStats[$i]->{$value};
        print "$TaxonomyInfo[$i]->[VALUES_INDEX]->{$value}\n";
    }
    print "\n";
}

exit(0);

##----------------------------------------------------------------
## subroutines  (alphabetized)
##----------------------------------------------------------------
##----------------------------------------------------------------
## usage : print out an optional error message, followed by usage
## information, and then die
##----------------------------------------------------------------
sub usage
{
    if (scalar(@_) > 0)
    {
        print shift;
    }

    die "\nUsage:  analyze_coverage.pl <dir>\n";
```

```
};
__END__

=cut

=head1 NAME

    # give your program name and a very short description
    analyze_coverage.pl - Calculates how many examples of each taxonomy
      attribute/value exist in the given directory, and prints report
      to stdout.

=head1 SYNOPSIS

    # show examples of how to use your program
    ./analyze_coverage.pl .

=head1 DESCRIPTION

    # describe in more detail what your_program does

=head1 OPTIONS

    # document your_program's options in sub-sections here
=head2 --option1

    # describe what option does

=head1 AUTHOR

    # provide your name and contact information
    Kendra Kratkiewicz, MIT Lincoln Laboratory

=cut
```

check_badok.pl

```
#!/usr/bin/perl -w

use strict;

# Copyright 2004 M.I.T.

##----------------------------------------------------------------
## File globals
##----------------------------------------------------------------


##----------------------------------------------------------------
## main program
##----------------------------------------------------------------

#----------------------------------------------------------------
# process arguments
#----------------------------------------------------------------
# check to see if we have the right number of arguments
if (@ARGV < 1)
{
    usage();
}

# check arguments for validity
my $dir = $ARGV[0];
-e $dir or die "Sorry.  Directory $dir does not exist.\n";
-d $dir or die "Sorry.  $dir is not a directory.\n";

#----------------------------------------------------------------
# here's the beef
#----------------------------------------------------------------
opendir(THEDIR, $dir) or die "Sorry.  Could not open $dir.\n";
my @allfiles = readdir THEDIR;
closedir THEDIR;

# process each test case file; skip the rest
my $comment = undef;
foreach my $file (@allfiles)
{
    # set the comment to OK or BAD depending on the test case
    print "Processing: $file ...";
    if ($file =~ /ok.c/)
    {
        $comment = "OK";
    }
    elsif ($file =~ /(min.c|med.c|large.c)/)
    {
        $comment = "BAD";
    }
    else
    {
        print "skipped\n";
        next;
    }

    # if the file contains the appropriate comment line, it passes
    open(THISFILE, "<", $file) or die "Sorry.  Could not open
$file.\n";
    my $foundit = 0;
    while (<THISFILE>)
    {
        if ($_ =~ /\/\*\s*?$comment\s*?\*\//)
        {
            print "passed\n";
            $foundit = 1;
            last;
        }
    }
    # otherwise, it fails
    if (!$foundit)
    {
        print "FAILED\n";
    }
    close(THISFILE);
}

exit(0);

##----------------------------------------------------------------
## subroutines  (alphabetized)
##----------------------------------------------------------------
##----------------------------------------------------------------
## usage : print out an optional error message, followed by usage
## information, and then die
##----------------------------------------------------------------
sub usage
{
    if (scalar(@_) > 0)
    {
        print shift;
    }

    die "\nUsage:  check_badok.pl <dir>\n";
};
__END__

=cut

=head1 NAME

    # give your program name and a very short description
    check_badok.pl - Checks to see if each test case in the given
directory
    contains the appropriate BAD/OK label.

=head1 SYNOPSIS
```

```
    # show examples of how to use your program
    ./check_badok.pl .

=head1 DESCRIPTION

    # describe in more detail what your_program does
    check_badok.pl prints a report to stdout indicating which files in
the
    given directory were processed, and whether each one passed or
failed.

=head1 OPTIONS

    # document your_program's options in sub-sections here
=head2 --option1

    # describe what option does

=head1 AUTHOR

    # provide your name and contact information
    Kendra Kratkiewicz, MIT Lincoln Laboratory

=cut
```

filter_cred.pl

```
#!/usr/bin/perl -w

use strict;
use Getopt::Long;
use Pod::Usage;

# Copyright 2004 M.I.T.

# MAIN
{
    my %errors;  # hash holds file names/line numbers of bounds errors

    # check to see if we have the right number of arguments
    if (@ARGV < 1)
    {
          usage();
    }

    # check arguments for validity
    my $dir = $ARGV[0];
    -e $dir or die "Sorry.  Directory $dir does not exist.\n";
    -d $dir or die "Sorry.  $dir is not a directory.\n";

    # initialize options to default values
    my $opt_outfile = "CRED_test_output.log";

    # get caller-provided values
    if(!GetOptions("outfile=s" => \$opt_outfile))
    {
          usage("Unable to parse options.\n");
    }

    # as long as we're getting input...
    while (<>)
    {
        # look for lines that have a file name, line number, and then
        # the phrase 'Bounds error'
        if (/(basic-\d*?-\w*?.c):(\d*?):Bounds error/)
        {
            # Use the file name as a hash key, and the line number
            # will be its value.  If this key already exists, we have
            # a problem, because there should only be one bounds error
            # per file.
            if (exists($errors{$1}))
            {
                print "*** WARNING: more than one bounds error reported
in $1 ***\n";
            }
            else
            {
                $errors{$1} = $2;
            }
        }
    }

    # read the list of files in the directory
    opendir(THEDIR, $dir) or die "Sorry.  Could not open $dir.\n";
    my @allfiles = readdir THEDIR;
    closedir THEDIR;

    # for each test file, find the overflow line number (labeled with
    # with the OK/BAD comment); figure out if a bounds error was
    # correctly or incorrectly reported or not reported
    my $comment;
    foreach my $file (@allfiles)
    {
        trim($file);  # get rid of leading/trailing whitespace

        if ($file =~ /ok.c/)
        {
            $comment = "OK";
        }
        elsif ($file =~ /(min.c|med.c|large.c)/)
        {
            $comment = "BAD";
        }
        else
        {
            next;
        }

        # find the line number of the buffer access
        print "Processing: $file ...";
        open(THISFILE, "<", $file) or die "Sorry.  Could not open
$file.\n";
        my $foundit = 0;
        my $linenum = 1;
        while (<THISFILE>)
        {
            ++$linenum;
            if ($_ =~ /\/\*\s*?$comment\s*?\*\//)
            {
            $foundit = 1;
            last;
            }
        }
        if (!$foundit)
        {
            print "*** ERROR: no $comment comment ***\n";
        }
        close(THISFILE);

        # look up the file name in the errors hash
        # if this is a BAD comment file, it should be there; if it's an
        # OK comment file, it should not be there.
        my $overflowed = exists($errors{$file});
        if (($comment eq "OK") && $overflowed)
        {
            print "*** ERROR: OK overflowed ***\n";
        }
        elsif (($comment eq "BAD") && (!$overflowed))
        {
            print "*** ERROR: BAD did not overflow***\n";
        }
        elsif (($comment eq "BAD") && ($errors{$file} != $linenum))
        {
            print "*** ERROR: overflow on wrong line ***\n";
        }
        else
        {
            print "PASSED\n";
        }
    }
}

##----------------------------------------------------------------
## trim : trim leading and trailing whitespace off the given
## string - modifies in place
##----------------------------------------------------------------
sub trim
{
    $_[0] =~ s/^\s+//;
    $_[0] =~ s/\s+$//;
};

##----------------------------------------------------------------
## usage : print out an optional error message, followed by usage
## information, and then die
##----------------------------------------------------------------
sub usage
{
    if (scalar(@_) > 0)
    {
        print shift;
    }

    die "\nUsage:  ./filter_cred.pl [--outfile=<outfile>]\n";
};


__END__

=head1 NAME
    filter_cred.pl - filters the given CRED output to determine if each
OK test
    case executed with no bounds errors, and if each BAD test case did
overflow
    as expected.  Writes output to CRED_test_output.log by default.

=head1  SYNOPSIS
    ./filter_cred.pl [--outfile=<outfile>]
```

```
=head1 OPTIONS

=head2 --outfile=<outfile>
    Specifies output file name to use instead of the default.

=head1 DESCRIPTION
    filter_cred.pl will warn if more than one bounds error is reported
on a
    test case, if an OK case overflowed, if a BAD case did not
overflow, or if
    a BAD case overflowed but on the wrong line (not the line marked
with the
    BAD comment).

=head1 EXAMPLE USAGE

=head1 AUTHOR

Kendra Kratkiewicz - 04/08/04 - MIT Lincoln Laboratory
```

gen_basic_tests.pl

```perl
#!/usr/bin/perl -w

use strict;
use Getopt::Long;

# Copyright 2004 M.I.T.

##--------------------------------------------------------------
## File globals
##--------------------------------------------------------------
# definitions for overflow size variations
my $OK_OVERFLOW = 0;
my $MIN_OVERFLOW = 1;
my $MED_OVERFLOW = 2;
my $LARGE_OVERFLOW = 3;

# buffer and overflow sizes defined
my $BUF_SIZE = 10;
my $ARRAY_SIZE = 5; # for when we have an array of buffers
my $OK_SIZE = 0;
my $MIN_SIZE = 1;
my $MED_SIZE = 8;
my $LARGE_SIZE = 4096;

# comments for the buffer accesses
use constant OK_COMMENT => "  /*  OK  */";
use constant BAD_COMMENT => "  /*  BAD  */";

# associate overflow type with size and comment in a hash
my %OverflowInfo = ($OK_OVERFLOW => [$OK_SIZE, OK_COMMENT],
                    $MIN_OVERFLOW => [$MIN_SIZE, BAD_COMMENT],
                    $MED_OVERFLOW => [$MED_SIZE, BAD_COMMENT],
                    $LARGE_OVERFLOW => [$LARGE_SIZE, BAD_COMMENT]);

# definitions for array indices related to the OverflowInfo hash
use constant SIZE_INDEX => 0;
use constant COMMENT_INDEX => 1;

# pieces of the test case programs
use constant FILE_HEADER =>
"/*
Copyright 2004 M.I.T.

Permission is hereby granted, without written agreement or royalty fee,
to use,
copy, modify, and distribute this software and its documentation for
any
purpose, provided that the above copyright notice and the following
three
paragraphs appear in all copies of this software.
```

```
IN NO EVENT SHALL M.I.T. BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT,
SPECIAL,
INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS
SOFTWARE
AND ITS DOCUMENTATION, EVEN IF M.I.T. HAS BEEN ADVISED OF THE
POSSIBILITY OF
SUCH DAMANGE.

M.I.T. SPECIFICALLY DISCLAIMS ANY WARRANTIES INCLUDING, BUT NOT LIMITED
TO
THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
PURPOSE,
AND NON-INFRINGEMENT.

THE SOFTWARE IS PROVIDED ON AN \"AS-IS\" BASIS AND M.I.T. HAS NO
OBLIGATION TO
PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
*/\n\n";

use constant TAXONOMY_CLASSIFICATION => "/* Taxonomy Classification:
VALUE */\n\n";
use constant MAIN_OPEN => "int main\(int argc, char *argv[]\)\n{\n";
use constant MAIN_CLOSE => "\n  return 0;\n}\n";

my $BUF_DECL = "  TYPE buf[$BUF_SIZE];\n\n";
my $BUF_ACCESS = "COMMENT\n  buf[INDEX] = WRITE_VALUE;\n";

# definitions for the taxonomy characteristics digits
my $WRITEREAD_DIGIT = 0;
my $WHICHBOUND_DIGIT = 1;
my $DATATYPE_DIGIT = 2;
my $MEMLOC_DIGIT = 3;
my $SCOPE_DIGIT = 4;
my $CONTAINER_DIGIT = 5;
my $POINTER_DIGIT = 6;
my $INDEXCOMPLEX_DIGIT = 7;
my $ADDRCOMPLEX_DIGIT = 8;
my $LENCOMPLEX_DIGIT = 9;
my $ALIASADDR_DIGIT = 10;
my $ALIASINDEX_DIGIT = 11;
my $LOCALFLOW_DIGIT = 12;
my $SECONDARYFLOW_DIGIT = 13;
my $LOOPSTRUCTURE_DIGIT = 14;
my $LOOPCOMPLEX_DIGIT = 15;
my $ASYNCHRONY_DIGIT = 16;
my $TAINT_DIGIT = 17;
my $RUNTIMEENVDEP_DIGIT = 18;
my $MAGNITUDE_DIGIT = 19;
my $CONTINUOUSDISCRETE_DIGIT = 20;
my $SIGNEDNESS_DIGIT = 21;

# associate taxonomy characteristics with their values in an array
my @TaxonomyInfo = (
    ['WRITE/READ', {0 => 'write', 1 => 'read'}],
    ['WHICH BOUND', {0 => 'upper', 1 => 'lower'}],
['DATA TYPE', {0 => 'char', 1 => 'int', 2 => 'float',
               3 => 'wchar', 4 => 'pointer', 5 => 'unsigned int',
               6 => 'unsigned char'}],
['MEMORY LOCATION', {0 => 'stack', 1 => 'heap',
                     2 => 'data segment',
                     3 => 'bss', 4 => 'shared'}],
['SCOPE', {0 => 'same', 1 => 'inter-procedural', 2 => 'global',
           3 => 'inter-file/inter-proc',
           4 => 'inter-file/global'}],
['CONTAINER', {0 => 'no', 1 => 'array', 2 => 'struct',
               3 => 'union', 4 => 'array of structs',
               5 => 'array of unions'}],
['POINTER', {0 => 'no', 1 => 'yes'}],
['INDEX COMPLEXITY', {0 => 'constant', 1 => 'variable',
                      2 => 'linear expr', 3 => 'non-linear expr',
                      4 => 'function return value',
                      5 => 'array contents', 6 => 'N/A'}],
['ADDRESS COMPLEXITY', {0 => 'constant', 1 => 'variable',
                        2 => 'linear expr', 3 => 'non-linear expr',
                        4 => 'function return value',
                        5 => 'array contents'}],
['LENGTH COMPLEXITY', {0 => 'N/A', 1 => 'none', 2 => 'constant',
                       3 => 'variable', 4 => 'linear expr',
                       5 => 'non-linear expr',
                       6 => 'function return value',
                       7 => 'array contents'}],
['ADDRESS ALIAS', {0 => 'none', 1 => 'yes, one level',
                   2 => 'yes, two levels'}],
['INDEX ALIAS', {0 => 'none', 1 => 'yes, one level',
                 2 => 'yes, two levels', 3 => 'N/A'}],
['LOCAL CONTROL FLOW', {0 => 'none', 1 => 'if', 2 => 'switch',
                        3 => 'cond', 4 => 'goto/label',
                        5 => 'longjmp', 6 => 'function pointer',
                        7 => 'recursion'}],
['SECONDARY CONTROL FLOW', {0 => 'none', 1 => 'if', 2 => 'switch',
                            3 => 'cond', 4 => 'goto/label',
                            5 => 'longjmp',
                            6 => 'function pointer',
                            7 => 'recursion'}],
['LOOP STRUCTURE', {0 => 'no', 1 => 'for', 2 => 'do-while',
                    3 => 'while', 4 => 'non-standard for',
                    5 => 'non-standard do-while',
                    6 => 'non-standard while'}],
['LOOP COMPLEXITY', {0 => 'N/A', 1 => 'zero', 2 => 'one',
                     3 => 'two', 4 => 'three'}],
['ASYNCHRONY', {0 => 'no', 1 => 'threads', 2 => 'forked process',
                3 => 'signal handler'}],
['TAINT', {0 => 'no', 1 => 'argc/argv',
           2 => 'environment variable',
           3 => 'file read', 4 => 'socket',
           5 => 'process environment'}],
['RUNTIME ENV. DEPENDENCE', {0 => 'no', 1 => 'yes'}],
['MAGNITUDE', {0 => 'no overflow', 1 => "$MIN_SIZE byte",
               2 => "$MED_SIZE bytes", 3 => "$LARGE_SIZE bytes"}],
['CONTINUOUS/DISCRETE', {0 => 'discrete', 1 => 'continuous'}],
    ['SIGNEDNESS', {0 => 'no', 1 => 'yes'}]);

# definitions for array indices related to the TaxonomyInfo array
use constant NAME_INDEX => 0;
use constant VALUES_INDEX => 1;

# associate taxonomy attributes with functions that handle them
my %AttributeFunctions =
(
    $WRITEREAD_DIGIT => \&do_writeread,
    $WHICHBOUND_DIGIT => \&do_whichbound,
    $DATATYPE_DIGIT => \&do_datatype,
    $MEMLOC_DIGIT => \&do_memloc,
    $SCOPE_DIGIT => \&do_scope,
    $CONTAINER_DIGIT => \&do_container,
    $POINTER_DIGIT => \&do_pointer,
    $INDEXCOMPLEX_DIGIT => \&do_indexcomplex,
    $ADDRCOMPLEX_DIGIT => \&do_addrcomplex,
    $LENCOMPLEX_DIGIT => \&do_lencomplex,
    $ALIASADDR_DIGIT => \&do_aliasaddr,
    $ALIASINDEX_DIGIT => \&do_aliasindex,
    $LOCALFLOW_DIGIT => \&do_localflow,
    $SECONDARYFLOW_DIGIT => \&do_secondaryflow,
    $LOOPSTRUCTURE_DIGIT => \&do_loopstructure,
    $LOOPCOMPLEX_DIGIT => \&do_loopcomplex,
    $ASYNCHRONY_DIGIT => \&do_asynchrony,
    $TAINT_DIGIT => \&do_taint,
    $CONTINUOUSDISCRETE_DIGIT => \&do_continuousdiscrete,
    $SIGNEDNESS_DIGIT => \&do_signedness,
    $RUNTIMEENVDEP_DIGIT => \&do_runtimeenvdep,
    $MAGNITUDE_DIGIT => \&do_magnitude,
);

##---------------------------------------------------------------
## main program
##---------------------------------------------------------------

#---------------------------------------------------------------
# process options
# (future options: clean dir, starting file num, output directory)
#---------------------------------------------------------------
# initialize to default values
my $Opt_Combo = undef;

# get caller-provided values
if(!GetOptions("combo" => \$Opt_Combo))
{
    usage("Unable to parse options.\n");
}

#---------------------------------------------------------------
# process arguments
#---------------------------------------------------------------
# get remaining command line arguments - they are the attributes to
vary
```

```
my @vary_these = @ARGV;

#---------------------------------------------------------------
# here's the beef
#---------------------------------------------------------------

# find the file sequence number to start with
my $fseq_num = get_fseq_num();

# if the caller wants a combo, pass all attributes in one call
if ($Opt_Combo)
{
    $fseq_num = make_files(\@vary_these, $fseq_num);
}
else
{
    # the caller wants singles, not combos, so pass one attribute at a
time
    foreach my $attribute (@vary_these)
    {
        $fseq_num = make_files([$attribute], $fseq_num);
    }
}

exit(0);

##---------------------------------------------------------------
## subroutines  (alphabetized)
##---------------------------------------------------------------
##---------------------------------------------------------------
## do_addrcomplex : produces all the test case variants for the
##  "address complexity" attribute.
##---------------------------------------------------------------
sub do_addrcomplex
{   # COMBO NOTE: these will be affected by whether or not we're using
    # a pointer or a function call instead of a buffer index.  For now
    # it assumes a buffer index.  This needs to be updated for other
    # combos to work.

    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # variable = 1
        my $var_values_ref = get_hash_copy($entry);
        $var_values_ref->{'TAX'}->[$ADDRCOMPLEX_DIGIT] = 1;
        $var_values_ref->{'OTHER_DECL'} = "  int i;\n" .
            $var_values_ref->{'OTHER_DECL'};
        $var_values_ref->{'PRE_ACCESS'} =  "  i = INDEX;\n" .
            $var_values_ref->{'PRE_ACCESS'};
        $var_values_ref->{'ACCESS'} = "COMMENT\n  (buf + i)[0] =
            WRITE_VALUE;\n";
        push @$results_ref, $var_values_ref;
        # linear expression = 2
        my $linexp_values_ref = get_hash_copy($entry);
        $linexp_values_ref->{'TAX'}->[$ADDRCOMPLEX_DIGIT] = 2;
        $linexp_values_ref->{'OTHER_DECL'} = "  int i;\n" .
            $linexp_values_ref->{'OTHER_DECL'};
        # need to calculate the values that will get substituted to fit
        # linear expr: 4 * FACTOR2 + REMAIN = previous value for INDEX
        my $factor1 = 4;
        foreach my $index_value (keys %{$linexp_values_ref->{'MULTIS'}-
>{'INDEX'}})
        {
            $linexp_values_ref->{'MULTIS'}->{'FACTOR2'}->{$index_value}
= int $linexp_values_ref->{'MULTIS'}->{'INDEX'}-  >{$index_value} /
$factor1;
            $linexp_values_ref->{'MULTIS'}->{'REMAIN'}->{$index_value}
= $linexp_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} -
($linexp_values_ref->{'MULTIS'}->{'FACTOR2'}->{$index_value} *
$factor1);
        }
        $linexp_values_ref->{'PRE_ACCESS'} =  "  i = FACTOR2;\n" .
              $linexp_values_ref->{'PRE_ACCESS'};
        $linexp_values_ref->{'ACCESS'} =
      "COMMENT\n  (buf + ($factor1 * i))[REMAIN] = WRITE_VALUE;\n";
        push @$results_ref, $linexp_values_ref;

        # non-linear expression = 3
        my $nonlinexp_values_ref = get_hash_copy($entry);
        $nonlinexp_values_ref->{'TAX'}->[$ADDRCOMPLEX_DIGIT] = 3;
        $nonlinexp_values_ref->{'OTHER_DECL'} = "  int i;\n" .
            $nonlinexp_values_ref->{'OTHER_DECL'};
        foreach my $index_value (keys %{$nonlinexp_values_ref-
>{'MULTIS'}->{'INDEX'}})
        {
            $nonlinexp_values_ref->{'MULTIS'}->{'MOD'}->{$index_value}
= int $nonlinexp_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} + 1;
        }
        $nonlinexp_values_ref->{'PRE_ACCESS'} =  "  i = MOD;\n" .
              $nonlinexp_values_ref->{'PRE_ACCESS'};
        $nonlinexp_values_ref->{'ACCESS'} =
            "COMMENT\n  (buf + (INDEX % i))[0] = WRITE_VALUE;\n";
        push @$results_ref, $nonlinexp_values_ref;

         # function return value = 4
        my $funcret_values_ref = get_hash_copy($entry);
        $funcret_values_ref->{'TAX'}->[$ADDRCOMPLEX_DIGIT] = 4;
        $funcret_values_ref->{'BEFORE_MAIN'} =
            "TYPE * function1\(TYPE * arg1)\n{\n" .
            "  return arg1;\n}\n\n" .
            $funcret_values_ref->{'BEFORE_MAIN'};
        $funcret_values_ref->{'ACCESS'} =
            "COMMENT\n  (function1(buf))[INDEX] = WRITE_VALUE;\n";
        push @$results_ref, $funcret_values_ref;

        #  array contents = 5
        my $arraycontent_values_ref = get_hash_copy($entry);
        $arraycontent_values_ref->{'TAX'}->[$ADDRCOMPLEX_DIGIT] = 5;
        $arraycontent_values_ref->{'OTHER_DECL'} =
            "  TYPE * addr_array[$ARRAY_SIZE];\n" .
            $arraycontent_values_ref->{'OTHER_DECL'};
        $arraycontent_values_ref->{'PRE_ACCESS'} =
            "  addr_array[0] = buf;\n" .
            $arraycontent_values_ref->{'PRE_ACCESS'};
        $arraycontent_values_ref->{'ACCESS'} =
            "COMMENT\n  (addr_array[0])[INDEX] = WRITE_VALUE;\n";
        push @$results_ref, $arraycontent_values_ref;
    }

    return $results_ref;
}

##------------------------------------------------------------------
## do_aliasaddr : produces all the test case variants for the
##  "alias of buffer address" attribute.
##------------------------------------------------------------------
sub do_aliasaddr
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # one alias = 1 (there are two variations of this)
        # first variation: simple alias to original buffer
        my $onealiasvar1_values_ref = get_hash_copy($entry);
        $onealiasvar1_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1;
        $onealiasvar1_values_ref->{'OTHER_DECL'} =
            "  TYPE * buf_alias;\n" .
            $onealiasvar1_values_ref->{'OTHER_DECL'};
        $onealiasvar1_values_ref->{'PRE_ACCESS'} =
            "  buf_alias = buf;\n" .
            $onealiasvar1_values_ref->{'PRE_ACCESS'};
        $onealiasvar1_values_ref->{'ACCESS'} =~ s/buf/buf_alias/;
        push @$results_ref, $onealiasvar1_values_ref;

        # second variation: buffer passed into one function
        my $onealiasvar2_values_ref = get_hash_copy($entry);
        $onealiasvar2_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1;
        $onealiasvar2_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 1;  #inter-
procedural
        $onealiasvar2_values_ref->{'BEFORE_MAIN'} =
            "void function1\(TYPE * buf\)\n{\n" .
            $onealiasvar2_values_ref->{'ACCESS'} . "}\n\n" .
            $onealiasvar2_values_ref->{'BEFORE_MAIN'};
        $onealiasvar2_values_ref->{'ACCESS'} = "  function1\(buf\);\n";
        push @$results_ref, $onealiasvar2_values_ref;

        # two aliases = 2 (there are two variations of this)
        # first variation: simple double alias to original buffer
        my $twoaliasvar1_values_ref = get_hash_copy($entry);
        $twoaliasvar1_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 2;
        $twoaliasvar1_values_ref->{'OTHER_DECL'} =
            "  TYPE * buf_alias1;\n" .
            "  TYPE * buf_alias2;\n" .
            $twoaliasvar1_values_ref->{'OTHER_DECL'};
        $twoaliasvar1_values_ref->{'PRE_ACCESS'} =
            "  buf_alias1 = buf;\n" .
            "  buf_alias2 = buf_alias1;\n" .
            $twoaliasvar1_values_ref->{'PRE_ACCESS'};
        $twoaliasvar1_values_ref->{'ACCESS'} =~ s/buf/buf_alias2/;
        push @$results_ref, $twoaliasvar1_values_ref;

        # second variation: buffer passed into two functions
        my $twoaliasvar2_values_ref = get_hash_copy($entry);
        $twoaliasvar2_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 2;
        $twoaliasvar2_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 1;  #inter-
procedural
        $twoaliasvar2_values_ref->{'BEFORE_MAIN'} =
            "void function2\(TYPE * buf\)\n{\n" .
            $twoaliasvar2_values_ref->{'ACCESS'} . "}\n\n" .
            "void function1\(TYPE * buf\)\n{\n" .
            "  function2\(buf\);\n}\n\n" .
            $twoaliasvar2_values_ref->{'BEFORE_MAIN'};
        $twoaliasvar2_values_ref->{'ACCESS'} = "  function1\(buf\);\n";
        push @$results_ref, $twoaliasvar2_values_ref;
    }

    return $results_ref;
}

##----------------------------------------------------------------
## do_aliasindex : produces all the test case variants for the
##  "alias of buffer index" attribute.
##----------------------------------------------------------------
sub do_aliasindex
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # one alias = 1
        my $onealias_values_ref = get_hash_copy($entry);
        $onealias_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 1;
        $onealias_values_ref->{'OTHER_DECL'} =
            "  int i;\n  int j;\n" .
            $onealias_values_ref->{'OTHER_DECL'};
        $onealias_values_ref->{'PRE_ACCESS'} =
            "  i = INDEX;\n  j = i;\n" .
            $onealias_values_ref->{'PRE_ACCESS'};
        $onealias_values_ref->{'ACCESS'} =~ s/INDEX/j/;
        push @$results_ref, $onealias_values_ref;

        # two aliases = 2
        my $twoalias_values_ref = get_hash_copy($entry);
        $twoalias_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 2;
        $twoalias_values_ref->{'OTHER_DECL'} =
            "  int i;\n  int j;\n  int k;\n" .
            $twoalias_values_ref->{'OTHER_DECL'};
        $twoalias_values_ref->{'PRE_ACCESS'} =
            "  i = INDEX;\n  j = i;\n  k = j;\n" .
            $twoalias_values_ref->{'PRE_ACCESS'};
        $twoalias_values_ref->{'ACCESS'} =~ s/INDEX/k/;
        push @$results_ref, $twoalias_values_ref;
    }

    return $results_ref;
}

##----------------------------------------------------------------
## do_asynchrony : produces all the test case variants for the
##  "asynchrony" attribute.
##----------------------------------------------------------------
sub do_asynchrony
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # threads = 1
        my $thread_values_ref = get_hash_copy($entry);
        $thread_values_ref->{'TAX'}->[$ASYNCHRONY_DIGIT] = 1;
        $thread_values_ref->{'INCL'} =
            "#include <pthread.h>\n" . $thread_values_ref->{'INCL'};
        $thread_values_ref->{'BEFORE_MAIN'} =
            "void * thread_function1(void * arg1)\n{\n" .
            $thread_values_ref->{'BUF_DECL'} .
            $thread_values_ref->{'OTHER_DECL'} .
            $thread_values_ref->{'PRE_ACCESS'} .
            $thread_values_ref->{'ACCESS'} .
            $thread_values_ref->{'POST_ACCESS'} .
            "  pthread_exit((void *)NULL);\n\n" .
            "  return 0;\n}\n\n" .
            $thread_values_ref->{'BEFORE_MAIN'};
        $thread_values_ref->{'BUF_DECL'} = "";
        $thread_values_ref->{'OTHER_DECL'} =
            "  pthread_t  thread1;\n\n";
        $thread_values_ref->{'PRE_ACCESS'} =
            "  pthread_create(&thread1, NULL, &thread_function1, (void
      *)NULL);\n" .
            "  pthread_exit((void *)NULL);\n";
        $thread_values_ref->{'ACCESS'} = "";
        $thread_values_ref->{'POST_ACCESS'} = "";
        push @$results_ref, $thread_values_ref;

        # forked process = 2
        my $fork_values_ref = get_hash_copy($entry);
        $fork_values_ref->{'TAX'}->[$ASYNCHRONY_DIGIT] = 2;
        $fork_values_ref->{'TAX'}->[$LOCALFLOW_DIGIT] = 1;      # if
        $fork_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 1;  # if
        $fork_values_ref->{'INCL'} = "#include <sys/types.h>\n" .
                                     "#include <sys/wait.h>\n" .
                                     "#include <unistd.h>\n" .
                                     "#include <stdlib.h>\n" .
            $fork_values_ref->{'INCL'};
        $fork_values_ref->{'OTHER_DECL'} =
            "  pid_t pid;\n  int child_status;\n\n";
        $fork_values_ref->{'PRE_ACCESS'} =
            $fork_values_ref->{'PRE_ACCESS'} .
            "  pid = fork();\n" .
           "  if (pid == 0)\n  {\n    sleep(3);\n    exit(0);\n  }\n" .
            "  else if (pid != -1)\n  {\n    wait(&child_status);\n" .
            "    if (WIFEXITED(child_status))\n    {\n";
        $fork_values_ref->{'ACCESS'} =
            indent(6, $fork_values_ref->{'ACCESS'});
        $fork_values_ref->{'POST_ACCESS'} =
            "    }\n  }\n" . $fork_values_ref->{'POST_ACCESS'};
        push @$results_ref, $fork_values_ref;

        # signal handler = 3
        my $sighand_values_ref = get_hash_copy($entry);
        $sighand_values_ref->{'TAX'}->[$ASYNCHRONY_DIGIT] = 3;
        $sighand_values_ref->{'INCL'} =
            "#include <signal.h>\n" . "#include <sys/time.h>\n" .
            "#include <unistd.h>\n" . $sighand_values_ref->{'INCL'};
        $sighand_values_ref->{'BEFORE_MAIN'} =
            "void sigalrm_handler(int arg1)\n{\n" .
            $sighand_values_ref->{'BUF_DECL'} .
            $sighand_values_ref->{'OTHER_DECL'} .
            $sighand_values_ref->{'PRE_ACCESS'} .
            $sighand_values_ref->{'ACCESS'} .
            $sighand_values_ref->{'POST_ACCESS'} .
            "  return;\n}\n\n" .
            $sighand_values_ref->{'BEFORE_MAIN'};
        $sighand_values_ref->{'BUF_DECL'} = "";
        $sighand_values_ref->{'OTHER_DECL'} =
            "  struct itimerval new_timeset, old_timeset;\n";
        $sighand_values_ref->{'PRE_ACCESS'} =
            "  signal(SIGALRM, &sigalrm_handler);\n" .
            "  new_timeset.it_interval.tv_sec = 1;\n" .
            "  new_timeset.it_interval.tv_usec = 0;\n" .
            "  new_timeset.it_value.tv_sec = 1;\n" .
            "  new_timeset.it_value.tv_usec = 0;\n" .
           "  setitimer(ITIMER_REAL, &new_timeset, &old_timeset );\n" .
            "  pause();\n\n";
        $sighand_values_ref->{'ACCESS'} = "";
        $sighand_values_ref->{'POST_ACCESS'} = "";
        push @$results_ref, $sighand_values_ref;
    }

    return $results_ref;
}

##---------------------------------------------------------------
## do_container : produces all the test case variants for the
##  "container" attribute.
##-----------------------------------------------------------------
sub do_container
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # array = 1
        my $array_values_ref = get_hash_copy($entry);
        $array_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 1;
        $array_values_ref->{'BUF_DECL'} =
            "  TYPE buf[$ARRAY_SIZE][$BUF_SIZE];\n\n";
        # COMBO NOTE: array_index would be affected for underflow
        # Needs to be updated for underflow combos to work.
        my $array_index1 = $ARRAY_SIZE - 1;
        $array_values_ref->{'ACCESS'} =
            "COMMENT\n  buf[$array_index1][INDEX] = WRITE_VALUE;\n";
        push @$results_ref, $array_values_ref;

        # struct = 2
        # variation 1: the buffer is the only thing in the struct
        my $struct1_values_ref = get_hash_copy($entry);
        $struct1_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 2;
        $struct1_values_ref->{'BEFORE_MAIN'} = "typedef struct\n{\n" .
                               "  TYPE buf[$BUF_SIZE];\n" .
                               "} my_struct;\n\n" .
                         $struct1_values_ref->{'BEFORE_MAIN'};
        $struct1_values_ref->{'BUF_DECL'} = "  my_struct s;\n\n";
        # COMBO NOTE: access would be different for a read or a pointr.
        # Needs to be updated for read and pointer combos to work.
        $struct1_values_ref->{'ACCESS'} =
            "COMMENT\n  s.buf[INDEX] = WRITE_VALUE;\n";
        push @$results_ref, $struct1_values_ref;

        # struct = 2
        # variation 2: 2 buffers, 1st overflows
        my $struct2_values_ref = get_hash_copy($entry);
        $struct2_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 2;
        $struct2_values_ref->{'BEFORE_MAIN'} = "typedef struct\n{\n" .
                               "  TYPE buf1[$BUF_SIZE];\n" .
                               "  TYPE buf2[$BUF_SIZE];\n" .
                               "} my_struct;\n\n" .
                         $struct2_values_ref->{'BEFORE_MAIN'};
        $struct2_values_ref->{'BUF_DECL'} = "  my_struct s;\n\n";
        # COMBO NOTE: access would be different for a read or a pointr.
        # Needs to be updated for read and pointer combos to work.
        $struct2_values_ref->{'ACCESS'} =
            "COMMENT\n  s.buf1[INDEX] = WRITE_VALUE;\n";
        push @$results_ref, $struct2_values_ref;

        # struct = 2
        # variation 3: 2 buffers, 2nd overflows
        my $struct3_values_ref = get_hash_copy($entry);
        $struct3_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 2;
        $struct3_values_ref->{'BEFORE_MAIN'} = "typedef struct\n{\n" .
                                    "  TYPE buf1[$BUF_SIZE];\n" .
                                    "  TYPE buf2[$BUF_SIZE];\n" .
                                    "} my_struct;\n\n" .
                              $struct3_values_ref->{'BEFORE_MAIN'};
        $struct3_values_ref->{'BUF_DECL'} = "  my_struct s;\n\n";
        # COMBO NOTE: access would be different for a read or a pointr.
        # Needs to be updated for read and pointer combos to work.
        $struct3_values_ref->{'ACCESS'} =
            "COMMENT\n  s.buf2[INDEX] = WRITE_VALUE;\n";
        push @$results_ref, $struct3_values_ref;

        # struct = 2
        # variation 4: buffer then integer
        my $struct4_values_ref = get_hash_copy($entry);
        $struct4_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 2;
        $struct4_values_ref->{'BEFORE_MAIN'} = "typedef struct\n{\n" .
                                    "  TYPE buf[$BUF_SIZE];\n" .
                                    "  int int_field;\n" .
                                    "} my_struct;\n\n" .
                           $struct4_values_ref->{'BEFORE_MAIN'};
        $struct4_values_ref->{'BUF_DECL'} = "  my_struct s;\n\n";
        # COMBO NOTE: access would be different for a read or a pointr.
        # Needs to be updated for read and pointer combos to work.
        $struct4_values_ref->{'ACCESS'} =
            "COMMENT\n  s.buf[INDEX] = WRITE_VALUE;\n";
        push @$results_ref, $struct4_values_ref;

        # struct = 2
        # variation 5: integer then buffer
        my $struct5_values_ref = get_hash_copy($entry);
        $struct5_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 2;
        $struct5_values_ref->{'BEFORE_MAIN'} = "typedef struct\n{\n" .
                                    "  int int_field;\n" .
                                    "  TYPE buf[$BUF_SIZE];\n" .
                                    "} my_struct;\n\n" .
                            $struct5_values_ref->{'BEFORE_MAIN'};
        $struct5_values_ref->{'BUF_DECL'} = "  my_struct s;\n\n";
        # COMBO NOTE: access would be different for a read or a pointr.
        # Needs to be updated for read and pointer combos to work.
        $struct5_values_ref->{'ACCESS'} =
            "COMMENT\n  s.buf[INDEX] = WRITE_VALUE;\n";
        push @$results_ref, $struct5_values_ref;

        # struct = 2
        # variation 6: buffer then integer, use integer as index
        my $struct6_values_ref = get_hash_copy($entry);
        $struct6_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 2;
        $struct6_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 1; #
variable
        $struct6_values_ref->{'BEFORE_MAIN'} = "typedef struct\n{\n" .
                                    "  TYPE buf[$BUF_SIZE];\n" .
                                    "  int int_field;\n" .
                                    "} my_struct;\n\n" .
                              $struct6_values_ref->{'BEFORE_MAIN'};
        $struct6_values_ref->{'BUF_DECL'} = "  my_struct s;\n\n";
        $struct6_values_ref->{'PRE_ACCESS'} =
            "  s.int_field = INDEX;\n" .
            $struct6_values_ref->{'PRE_ACCESS'};
        # COMBO NOTE: access would be different for a read or a pointr.
        # Needs to be updated for read and pointer combos to work.
        $struct6_values_ref->{'ACCESS'} =
            "COMMENT\n  s.buf[s.int_field] = WRITE_VALUE;\n";
        push @$results_ref, $struct6_values_ref;

        # struct = 2
        # variation 7: integer then buffer, use integer as index
        my $struct7_values_ref = get_hash_copy($entry);
        $struct7_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 2;
        $struct7_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 1; #
variable
        $struct7_values_ref->{'BEFORE_MAIN'} = "typedef struct\n{\n" .
                                   "  int int_field;\n" .
                                   "  TYPE buf[$BUF_SIZE];\n" .
                                   "} my_struct;\n\n" .
                              $struct7_values_ref->{'BEFORE_MAIN'};
        $struct7_values_ref->{'BUF_DECL'} = "  my_struct s;\n\n";
        $struct7_values_ref->{'PRE_ACCESS'} =
            "  s.int_field = INDEX;\n" .
            $struct7_values_ref->{'PRE_ACCESS'};
        # COMBO NOTE: access would be different for a read or a pointr.
        # Needs to be updated for read and pointer combos to work.
        $struct7_values_ref->{'ACCESS'} =
            "COMMENT\n  s.buf[s.int_field] = WRITE_VALUE;\n";
        push @$results_ref, $struct7_values_ref;

        # union = 3
        # variation 1: buffer first
        my $union1_values_ref = get_hash_copy($entry);
        $union1_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 3;
        $union1_values_ref->{'BEFORE_MAIN'} = "typedef union\n{\n" .
                                   "  TYPE buf[$BUF_SIZE];\n" .
                                   "  int intval;\n" .
                                   "} my_union;\n\n" .
                              $union1_values_ref->{'BEFORE_MAIN'};
        $union1_values_ref->{'BUF_DECL'} = "  my_union u;\n\n";
        # COMBO NOTE: access would be different for a read or a pointr.
        # Needs to be updated for read and pointer combos to work.
        $union1_values_ref->{'ACCESS'} =
            "COMMENT\n  u.buf[INDEX] = WRITE_VALUE;\n";
        push @$results_ref, $union1_values_ref;

        # union = 3
        # variation 2: buffer second
        my $union2_values_ref = get_hash_copy($entry);
        $union2_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 3;
        $union2_values_ref->{'BEFORE_MAIN'} = "typedef union\n{\n" .
                                   "  int intval;\n" .
                                   "  TYPE buf[$BUF_SIZE];\n" .
```

```
                               "} my_union;\n\n" .
                          $union2_values_ref->{'BEFORE_MAIN'};
$union2_values_ref->{'BUF_DECL'} = "  my_union u;\n\n";
# COMBO NOTE: access would be different for a read or a pointr.
# Needs to be updated for read and pointer combos to work.
$union2_values_ref->{'ACCESS'} =
    "COMMENT\n  u.buf[INDEX] = WRITE_VALUE;\n";
push @$results_ref, $union2_values_ref;

# array of structs = 4
# variation 1: the buffer is the only thing in the struct
my $structarray1_values_ref = get_hash_copy($entry);
$structarray1_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 4;
$structarray1_values_ref->{'BEFORE_MAIN'} =
    "typedef struct\n{\n" .
                               "  TYPE buf[$BUF_SIZE];\n" .
                               "} my_struct;\n\n" .
                     $structarray1_values_ref->{'BEFORE_MAIN'};
$structarray1_values_ref->{'BUF_DECL'} =
    "  my_struct array_buf[$ARRAY_SIZE];\n\n";
# COMBO NOTE: access would be different for a read or a pointr.
# Needs to be updated for read and pointer combos to work.
# COMBO NOTE: array_index would be different for underflow.
# Needs to be updated for underflow combos to work.
my $array_index2 = $ARRAY_SIZE - 1;
$structarray1_values_ref->{'ACCESS'} =
          "COMMENT\n  array_buf[$array_index2].buf[INDEX] =
          WRITE_VALUE;\n";
push @$results_ref, $structarray1_values_ref;

# array of structs = 4
# variation 2: 2 buffers, 1st overflows
my $structarray2_values_ref = get_hash_copy($entry);
$structarray2_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 4;
$structarray2_values_ref->{'BEFORE_MAIN'} =
    "typedef struct\n{\n" .
                               "  TYPE buf1[$BUF_SIZE];\n" .
                               "  TYPE buf2[$BUF_SIZE];\n" .
                               "} my_struct;\n\n" .
                     $structarray2_values_ref->{'BEFORE_MAIN'};
$structarray2_values_ref->{'BUF_DECL'} =
    "  my_struct array_buf[$ARRAY_SIZE];\n\n";
# COMBO NOTE: access would be different for a read or a pointr.
# Needs to be updated for read and pointer combos to work.
# COMBO NOTE: array_index would be different for underflow.
# Needs to be updated for underflow combos to work.
my $array_index3 = $ARRAY_SIZE - 1;
$structarray2_values_ref->{'ACCESS'} =
    "COMMENT\n  array_buf[$array_index3].buf1[INDEX] =
    WRITE_VALUE;\n";
push @$results_ref, $structarray2_values_ref;

# array of structs = 4
# variation 3: 2 buffers, 2nd overflows
my $structarray3_values_ref = get_hash_copy($entry);
```

```
$structarray3_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 4;
$structarray3_values_ref->{'BEFORE_MAIN'} =
    "typedef struct\n{\n" .
                           "  TYPE buf1[$BUF_SIZE];\n" .
                           "  TYPE buf2[$BUF_SIZE];\n" .
                           "} my_struct;\n\n" .
                     $structarray3_values_ref->{'BEFORE_MAIN'};
$structarray3_values_ref->{'BUF_DECL'} =
    "  my_struct array_buf[$ARRAY_SIZE];\n\n";
# COMBO NOTE: access would be different for a read or a pointr.
# Needs to be updated for read and pointer combos to work.
# COMBO NOTE: array_index would be different for underflow.
# Needs to be updated for underflow combos to work.
my $array_index4 = $ARRAY_SIZE - 1;
$structarray3_values_ref->{'ACCESS'} =
    "COMMENT\n  array_buf[$array_index4].buf2[INDEX] =
    WRITE_VALUE;\n";
push @$results_ref, $structarray3_values_ref;

# array of structs = 4
# variation 4: buffer then integer
my $structarray4_values_ref = get_hash_copy($entry);
$structarray4_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 4;
$structarray4_values_ref->{'BEFORE_MAIN'} =
    "typedef struct\n{\n" .
                           "  TYPE buf[$BUF_SIZE];\n" .
                           "  int int_field;\n" .
                           "} my_struct;\n\n" .
                     $structarray4_values_ref->{'BEFORE_MAIN'};
$structarray4_values_ref->{'BUF_DECL'} =
    "  my_struct array_buf[$ARRAY_SIZE];\n\n";
# COMBO NOTE: access would be different for a read or a pointr.
# Needs to be updated for read and pointer combos to work.
# COMBO NOTE: array_index would be different for underflow.
# Needs to be updated for underflow combos to work.
my $array_index5 = $ARRAY_SIZE - 1;
$structarray4_values_ref->{'ACCESS'} =
    "COMMENT\n  array_buf[$array_index5].buf[INDEX] =
    WRITE_VALUE;\n";
push @$results_ref, $structarray4_values_ref;

# array of structs = 4
# variation 5: integer then buffer
my $structarray5_values_ref = get_hash_copy($entry);
$structarray5_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 4;
$structarray5_values_ref->{'BEFORE_MAIN'} =
    "typedef struct\n{\n" .
                           "  int int_field;\n" .
                           "  TYPE buf[$BUF_SIZE];\n" .
                           "} my_struct;\n\n" .
                     $structarray5_values_ref->{'BEFORE_MAIN'};
$structarray5_values_ref->{'BUF_DECL'} =
    "  my_struct array_buf[$ARRAY_SIZE];\n\n";
# COMBO NOTE: access would be different for a read or a pointr.
# Needs to be updated for read and pointer combos to work.
```

```
        # COMBO NOTE: array_index would be different for underflow.
        # Needs to be updated for underflow combos to work.
        my $array_index6 = $ARRAY_SIZE - 1;
        $structarray5_values_ref->{'ACCESS'} =
            "COMMENT\n  array_buf[$array_index6].buf[INDEX] =
            WRITE_VALUE;\n";
        push @$results_ref, $structarray5_values_ref;

        # array of unions = 5
        # variation 1: buffer first
        my $unionarray1_values_ref = get_hash_copy($entry);
        $unionarray1_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 5;
        $unionarray1_values_ref->{'BEFORE_MAIN'} =
            "typedef union\n{\n" .
                                       "  TYPE buf[$BUF_SIZE];\n" .
                                       "  int intval;\n" .
                                       "} my_union;\n\n" .
                              $unionarray1_values_ref->{'BEFORE_MAIN'};
        $unionarray1_values_ref->{'BUF_DECL'} =
            "  my_union array_buf[$ARRAY_SIZE];\n\n";
        # COMBO NOTE: access would be different for a read or a pointr.
        # Needs to be updated for read and pointer combos to work.
        # COMBO NOTE: array_index would be different for underflow.
        # Needs to be updated for underflow combos to work.
        my $array_index7 = $ARRAY_SIZE - 1;
        $unionarray1_values_ref->{'ACCESS'} =
            "COMMENT\n  array_buf[$array_index7].buf[INDEX] =
            WRITE_VALUE;\n";
        push @$results_ref, $unionarray1_values_ref;

        # array of unions = 5
        # variation 2: buffer second
        my $unionarray2_values_ref = get_hash_copy($entry);
        $unionarray2_values_ref->{'TAX'}->[$CONTAINER_DIGIT] = 5;
        $unionarray2_values_ref->{'BEFORE_MAIN'} =
            "typedef union\n{\n" .
                                       "  int intval;\n" .
                                       "  TYPE buf[$BUF_SIZE];\n" .
                                       "} my_union;\n\n" .
                              $unionarray2_values_ref->{'BEFORE_MAIN'};
        $unionarray2_values_ref->{'BUF_DECL'} =
            "  my_union array_buf[$ARRAY_SIZE];\n\n";
        # COMBO NOTE: access would be different for a read or a pointr.
        # Needs to be updated for read and pointer combos to work.
        # COMBO NOTE: array_index would be different for underflow.
        # Needs to be updated for underflow combos to work.
        my $array_index8 = $ARRAY_SIZE - 1;
        $unionarray2_values_ref->{'ACCESS'} =
            "COMMENT\n  array_buf[$array_index8].buf[INDEX] =
            WRITE_VALUE;\n";
        push @$results_ref, $unionarray2_values_ref;
    }

    return $results_ref;
}
```

```
##------------------------------------------------------------
## do_continuousdiscrete : produces all the test case variants for the
##  "continuous/discrete" attribute.
##------------------------------------------------------------
sub do_continuousdiscrete
{
    # NOTE: these are all continuous versions of what's produced by
    # do_loopstructure and do_loopcomplex
    my $results_ref = [];

    # first, the variations on do_loopstructure
    my $loopstruct_array = do_loopstructure($_[0]);
    foreach my $variant_ref (@$loopstruct_array)
    {
        $variant_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] = 1;
        $variant_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 1;   # variable
        $variant_ref->{'ACCESS'} =~ s/INDEX/loop_counter/;
        push @$results_ref, $variant_ref;
    }

    # next, the variations on do_loopcomplex
    my $loopcomplex_array = do_loopcomplex($_[0]);
    foreach my $variant_ref (@$loopcomplex_array)
    {
        $variant_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] = 1;
        $variant_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 1;   # variable
        $variant_ref->{'ACCESS'} =~ s/INDEX/loop_counter/;
        push @$results_ref, $variant_ref;
    }

    return $results_ref;
}

##------------------------------------------------------------
## do_datatype : produces all the test case variants for the "data
type"
##  attribute.
##------------------------------------------------------------
sub do_datatype
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # integer = 1
        my $int_values_ref = get_hash_copy($entry);
        $int_values_ref->{'TAX'}->[$DATATYPE_DIGIT] = 1;
        $int_values_ref->{'SINGLES'}->{'TYPE'} = "int";
        $int_values_ref->{'SINGLES'}->{'WRITE_VALUE'} = "55";
        push @$results_ref, $int_values_ref;

        # float = 2
        my $float_values_ref = get_hash_copy($entry);
        $float_values_ref->{'TAX'}->[$DATATYPE_DIGIT] = 2;
        $float_values_ref->{'SINGLES'}->{'TYPE'} = "float";
        $float_values_ref->{'SINGLES'}->{'WRITE_VALUE'} = "55.55";
        push @$results_ref, $float_values_ref;

        # wchar = 3
        my $wchar_values_ref = get_hash_copy($entry);
        $wchar_values_ref->{'TAX'}->[$DATATYPE_DIGIT] = 3;
        $wchar_values_ref->{'SINGLES'}->{'TYPE'} = "wchar_t";
        $wchar_values_ref->{'SINGLES'}->{'WRITE_VALUE'} = "L\'A\'";
        $wchar_values_ref->{'INCL'} = "#include <wchar.h>\n" .
            $wchar_values_ref->{'INCL'};
        push @$results_ref, $wchar_values_ref;

        # pointer = 4
        my $ptr_values_ref = get_hash_copy($entry);
        $ptr_values_ref->{'TAX'}->[$DATATYPE_DIGIT] = 4;
        $ptr_values_ref->{'SINGLES'}->{'TYPE'} = "char *";
        $ptr_values_ref->{'SINGLES'}->{'WRITE_VALUE'} = "\"A\"";
        push @$results_ref, $ptr_values_ref;

        # unsigned integer = 5
        my $uint_values_ref = get_hash_copy($entry);
        $uint_values_ref->{'TAX'}->[$DATATYPE_DIGIT] = 5;
        $uint_values_ref->{'SINGLES'}->{'TYPE'} = "unsigned int";
        $uint_values_ref->{'SINGLES'}->{'WRITE_VALUE'} = "55";
        push @$results_ref, $uint_values_ref;

        # unsigned char = 6
        my $uchar_values_ref = get_hash_copy($entry);
        $uchar_values_ref->{'TAX'}->[$DATATYPE_DIGIT] = 6;
        $uchar_values_ref->{'SINGLES'}->{'TYPE'} = "unsigned char";
        push @$results_ref, $uchar_values_ref;
    }

    return $results_ref;
}

##-------------------------------------------------------------
## do_indexcomplex : produces all the test case variants for the
##  "index complexity" attribute.
##-------------------------------------------------------------
sub do_indexcomplex
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # variable = 1
        my $var_values_ref = get_hash_copy($entry);
        $var_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 1;
        $var_values_ref->{'OTHER_DECL'} = "  int i;\n" .
            $var_values_ref->{'OTHER_DECL'};
        $var_values_ref->{'PRE_ACCESS'} =  "  i = INDEX;\n" .
```

```
              $var_values_ref->{'PRE_ACCESS'};
        $var_values_ref->{'ACCESS'} =
            "COMMENT\n  buf[i] = WRITE_VALUE;\n";
        push @$results_ref, $var_values_ref;

        # linear expression = 2
        my $linexp_values_ref = get_hash_copy($entry);
        $linexp_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 2;
        $linexp_values_ref->{'OTHER_DECL'} = "  int i;\n" .
            $linexp_values_ref->{'OTHER_DECL'};
        # need to calculate the values that will get substituted to fit
        # linear expr: 4 * FACTOR2 + REMAIN = previous value for INDEX
        my $factor1 = 4;
        foreach my $index_value (keys %{$linexp_values_ref->{'MULTIS'}-
>{'INDEX'}})
        {
            $linexp_values_ref->{'MULTIS'}->{'FACTOR2'}->{$index_value}
= int $linexp_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} /
$factor1;
            $linexp_values_ref->{'MULTIS'}->{'REMAIN'}->{$index_value}
= $linexp_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} -
  ($linexp_values_ref->{'MULTIS'}->{'FACTOR2'}->{$index_value} *
$factor1);
        }
        $linexp_values_ref->{'PRE_ACCESS'} =  "  i = FACTOR2;\n" .
                                $linexp_values_ref->{'PRE_ACCESS'};
        $linexp_values_ref->{'ACCESS'} =
            "COMMENT\n  buf[($factor1 * i) + REMAIN] = WRITE_VALUE;\n";
        push @$results_ref, $linexp_values_ref;

        # non-linear expression = 3
        my $nonlinexp_values_ref = get_hash_copy($entry);
        $nonlinexp_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 3;
        $nonlinexp_values_ref->{'OTHER_DECL'} = "  int i;\n" .
            $nonlinexp_values_ref->{'OTHER_DECL'};
        foreach my $index_value (keys %{$nonlinexp_values_ref-
>{'MULTIS'}->{'INDEX'}})
        {
            $nonlinexp_values_ref->{'MULTIS'}->{'MOD'}->{$index_value}
= $nonlinexp_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} + 1;
        }
        $nonlinexp_values_ref->{'PRE_ACCESS'} =
            "  i = MOD;\n" . $nonlinexp_values_ref->{'PRE_ACCESS'};
        $nonlinexp_values_ref->{'ACCESS'} =
            "COMMENT\n  buf[INDEX % i] = WRITE_VALUE;\n";
        push @$results_ref, $nonlinexp_values_ref;

         # function return value = 4
        my $funcret_values_ref = get_hash_copy($entry);
        $funcret_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 4;
        $funcret_values_ref->{'BEFORE_MAIN'} =
            "int function1\(int arg1)\n{\n" .
            "  return arg1;\n}\n\n" .
            $funcret_values_ref->{'BEFORE_MAIN'};
        $funcret_values_ref->{'ACCESS'} =
            "COMMENT\n  buf[function1(INDEX)] = WRITE_VALUE;\n";
        push @$results_ref, $funcret_values_ref;

        #  array contents = 5
        my $arraycontent_values_ref = get_hash_copy($entry);
        $arraycontent_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 5;
        $arraycontent_values_ref->{'OTHER_DECL'} =
            "  int index_array[$ARRAY_SIZE];\n" .
            $arraycontent_values_ref->{'OTHER_DECL'};
        $arraycontent_values_ref->{'PRE_ACCESS'} =
            "  index_array[0] = INDEX;\n" .
            $arraycontent_values_ref->{'PRE_ACCESS'};
        $arraycontent_values_ref->{'ACCESS'} =
            "COMMENT\n  buf[index_array[0]] = WRITE_VALUE;\n";
        push @$results_ref, $arraycontent_values_ref;

        # not applicable = 6
        # this is covered by the pointer case
        #don't need another one here
    }

   return $results_ref;
}

##----------------------------------------------------------------
## do_lencomplex : produces all the test case variants for the
##  "length/limit complexity" attribute.
##----------------------------------------------------------------
sub do_lencomplex
{   # COMBO NOTE: these would be affected by data type.
    # Needs to be updated
    # for data type combos to work.
    # FUTURE: add option for generating whole set of string functions.
    # Note: we set the index complexity to N/A when passing the buffer
    #  address to a library function.  We don't know what the library
    #  function does, but our own code is not using an index.
    # However, we do set the continuous attribute if the library
    # function would be copying a string past the end of a buffer.
    # Also note that we set scope to inter-file/inter-procedural, and
    # alias addr to one alias.
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # COMBO NOTE: what function would we use for other data types?
        # none = 1
        # variation 1: copy directly from a string
        my $none1_values_ref = get_hash_copy($entry);
        $none1_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 1;
        $none1_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; # N/A -
no index
        $none1_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A - no
index
```

```
        $none1_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; # inter-
file/inter-proc
        $none1_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1 alias of
addr
        $none1_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] = 1;
#continuous
        $none1_values_ref->{'INCL'} = "#include <string.h>\n" .
            $none1_values_ref->{'INCL'};
        my $source_str;
        foreach my $index_value (keys %{$none1_values_ref->{'MULTIS'}-
>{'INDEX'}})
        {
            $source_str = "\"" .
                  ('A' x $none1_values_ref->{'MULTIS'}->{'INDEX'}-
                  >{$index_value}) . "\"";
            $none1_values_ref->{'MULTIS'}->{'SOURCE'}->{$index_value} =
                  $source_str;
        }
        $none1_values_ref->{'ACCESS'} =
            "COMMENT\n  strcpy(buf, SOURCE);\n";
        push @$results_ref, $none1_values_ref;

        # none = 1
        # variation 2: copy from another buffer
        my $none2_values_ref = get_hash_copy($entry);
        $none2_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 1;
        $none2_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; # N/A -
no index
        $none2_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A - no
index
        $none2_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; # inter-
file/inter-proc
        $none2_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1 alias of
addr
        $none2_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] = 1;
#continuous
        $none2_values_ref->{'INCL'} = "#include <string.h>\n" .
            $none2_values_ref->{'INCL'};
        foreach my $index_value (keys %{$none2_values_ref->{'MULTIS'}-
>{'INDEX'}})
        {
            $none2_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} +=
                   1;
        }
        $none2_values_ref->{'OTHER_DECL'} = "  char src[INDEX];\n" .
              $none2_values_ref->{'OTHER_DECL'};
        $none2_values_ref->{'PRE_ACCESS'} =
            "  memset(src, 'A', INDEX);\n" .
            "  src[INDEX - 1] = '\\0';\n" .
            $none2_values_ref->{'PRE_ACCESS'};
        $none2_values_ref->{'ACCESS'} =
            "COMMENT\n  strcpy(buf, src);\n";
        push @$results_ref, $none2_values_ref;

        # constant = 2
        my $const_values_ref = get_hash_copy($entry);
        $const_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 2;
        $const_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; # N/A -
no index
        $const_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A - no
index
        $const_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; # inter-
file/inter-proc
        $const_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1 alias of
addr
        $const_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] = 1;
#continuous
        $const_values_ref->{'INCL'} = "#include <string.h>\n" .
            $const_values_ref->{'INCL'};
        foreach my $index_value (keys %{$const_values_ref->{'MULTIS'}-
>{'INDEX'}})
        {
            $const_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} +=
                   1;
        }
        $const_values_ref->{'OTHER_DECL'} = "  char src[INDEX];\n" .
            $const_values_ref->{'OTHER_DECL'};
        $const_values_ref->{'PRE_ACCESS'} =
            "  memset(src, 'A', INDEX);\n" .
            "  src[INDEX - 1] = '\\0';\n" .
            $const_values_ref->{'PRE_ACCESS'};
        $const_values_ref->{'ACCESS'} =
            "COMMENT\n  strncpy(buf, src, INDEX);\n";
        push @$results_ref, $const_values_ref;

        # variable = 3
        my $var_values_ref = get_hash_copy($entry);
        $var_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 3;
        $var_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; # N/A - no
index
        $var_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A - no
index
        $var_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; # inter-
file/inter-proc
        $var_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1 alias of
addr
        $var_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] = 1;
#continuous
        $var_values_ref->{'INCL'} =
            "#include <string.h>\n" . $var_values_ref->{'INCL'};
        foreach my $index_value (keys %{$var_values_ref->{'MULTIS'}-
>{'INDEX'}})
        {
            $var_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} +=
                  1;
        }
        $var_values_ref->{'OTHER_DECL'} =
            "  size_t len;\n  char src[INDEX];\n" .
            $var_values_ref->{'OTHER_DECL'};
        $var_values_ref->{'PRE_ACCESS'} =
            "  memset(src, 'A', INDEX);\n" .
            "  src[INDEX - 1] = '\\0';\n" .
            "  len = INDEX;\n" .
            $var_values_ref->{'PRE_ACCESS'};
        $var_values_ref->{'ACCESS'} =
            "COMMENT\n  strncpy(buf, src, len);\n";
        push @$results_ref, $var_values_ref;

        # linear expression = 4
        my $linexp_values_ref = get_hash_copy($entry);
        $linexp_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 4;
        $linexp_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; # N/A -
no index
        $linexp_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A -
no index
        $linexp_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; # inter-
file/inter-proc
        $linexp_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1 alias
of addr
        $linexp_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] = 1;
#continuous
        $linexp_values_ref->{'INCL'} =
            "#include <string.h>\n" . $linexp_values_ref->{'INCL'};
        foreach my $index_value (keys %{$linexp_values_ref->{'MULTIS'}-
>{'INDEX'}})
        {
            $linexp_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value}
                  += 1;
        }
        $linexp_values_ref->{'OTHER_DECL'} =
            "  int i;\n  char src[INDEX];\n" .
            $linexp_values_ref->{'OTHER_DECL'};
        # need to calculate the values that will get substituted to fit
        # linear expr: 4 * FACTOR2 + REMAIN = previous value for INDEX
        my $factor1 = 4;
        foreach my $index_value (keys %{$linexp_values_ref->{'MULTIS'}-
>{'INDEX'}})
        {
            $linexp_values_ref->{'MULTIS'}->{'FACTOR2'}->{$index_value}
= int $linexp_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} /
$factor1;
            $linexp_values_ref->{'MULTIS'}->{'REMAIN'}->{$index_value}
= $linexp_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} -
  ($linexp_values_ref->{'MULTIS'}->{'FACTOR2'}->{$index_value} *
$factor1);
        }
        $linexp_values_ref->{'PRE_ACCESS'} =
            "  memset(src, 'A', INDEX);\n" .
            "  src[INDEX - 1] = '\\0';\n" .
            "  i = FACTOR2;\n" .
            $linexp_values_ref->{'PRE_ACCESS'};
        $linexp_values_ref->{'ACCESS'} =
            "COMMENT\n  strncpy(buf, src, ($factor1 * i) + REMAIN);\n";
        push @$results_ref, $linexp_values_ref;
        # non-linear expression = 5
        my $nonlinexp_values_ref = get_hash_copy($entry);
        $nonlinexp_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 5;
        $nonlinexp_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; #
N/A - no index
        $nonlinexp_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A
- no index
        $nonlinexp_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; # inter-
file/inter-proc
        $nonlinexp_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1
alias of addr
        $nonlinexp_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] =
1; #continuous
        $nonlinexp_values_ref->{'INCL'} = "#include <string.h>\n" .
            $nonlinexp_values_ref->{'INCL'};
        foreach my $index_value (keys %{$nonlinexp_values_ref-
>{'MULTIS'}->{'INDEX'}})
        {
            $nonlinexp_values_ref->{'MULTIS'}->{'INDEX'}-
>{$index_value} += 1;
        }
        $nonlinexp_values_ref->{'OTHER_DECL'} =
            "  int i;\n  char src[INDEX];\n" .
            $nonlinexp_values_ref->{'OTHER_DECL'};
        foreach my $index_value (keys %{$nonlinexp_values_ref-
>{'MULTIS'}->{'INDEX'}})
        {
            $nonlinexp_values_ref->{'MULTIS'}->{'MOD'}->{$index_value}
= int $nonlinexp_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} + 1;
        }
        $nonlinexp_values_ref->{'PRE_ACCESS'} =
            "  memset(src, 'A', INDEX);\n" .
            "  src[INDEX - 1] = '\\0';\n" .
            "  i = MOD;\n" .
            $nonlinexp_values_ref->{'PRE_ACCESS'};
        $nonlinexp_values_ref->{'ACCESS'} =
            "COMMENT\n  strncpy(buf, src, INDEX % i);\n";
        push @$results_ref, $nonlinexp_values_ref;

        # function return value = 6
        my $funcret_values_ref = get_hash_copy($entry);
        $funcret_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 6;
        $funcret_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; # N/A
- no index
        $funcret_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A -
no index
        $funcret_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; # inter-
file/inter-proc
        $funcret_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1 alias
of addr
        $funcret_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] = 1;
#continuous
        $funcret_values_ref->{'INCL'} = "#include <string.h>\n" .
            $funcret_values_ref->{'INCL'};
        foreach my $index_value (keys %{$funcret_values_ref-
>{'MULTIS'}->{'INDEX'}})
        {
            $funcret_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value}
                  += 1;
        }
        $funcret_values_ref->{'OTHER_DECL'} =
            "  char src[INDEX];\n" .
            $funcret_values_ref->{'OTHER_DECL'};
        $funcret_values_ref->{'BEFORE_MAIN'} =
            "int function1\(int arg1)\n{\n" .
            "  return arg1;\n}\n\n" .
            $funcret_values_ref->{'BEFORE_MAIN'};
        $funcret_values_ref->{'PRE_ACCESS'} =
            "  memset(src, 'A', INDEX);\n" .
            "  src[INDEX - 1] = '\\0';\n" .
            $funcret_values_ref->{'PRE_ACCESS'};
        $funcret_values_ref->{'ACCESS'} =
            "COMMENT\n  strncpy(buf, src, function1(INDEX));\n";
        push @$results_ref, $funcret_values_ref;

        #  array contents = 7
        my $arraycontent_values_ref = get_hash_copy($entry);
        $arraycontent_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 7;
        $arraycontent_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; #
N/A - no index
        $arraycontent_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; #
N/A - no index
        $arraycontent_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; # inter-
file/inter-proc
        $arraycontent_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1
alias of addr
        $arraycontent_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT]
= 1; #continuous
        $arraycontent_values_ref->{'INCL'} = "#include <string.h>\n" .
            $arraycontent_values_ref->{'INCL'};
        foreach my $index_value (keys %{$arraycontent_values_ref-
>{'MULTIS'}->{'INDEX'}})
        {
            $arraycontent_values_ref->{'MULTIS'}->{'INDEX'}-
>{$index_value} += 1;
        }
        $arraycontent_values_ref->{'OTHER_DECL'} =
            "  int index_array[$ARRAY_SIZE];\n  char src[INDEX];\n" .
            $arraycontent_values_ref->{'OTHER_DECL'};
        $arraycontent_values_ref->{'PRE_ACCESS'} =
            "  memset(src, 'A', INDEX);\n" .
            "  src[INDEX - 1] = '\\0';\n" .
            "  index_array[0] = INDEX;\n" .
            $arraycontent_values_ref->{'PRE_ACCESS'};
        $arraycontent_values_ref->{'ACCESS'} =
            "COMMENT\n  strncpy(buf, src, index_array[0]);\n";
        push @$results_ref, $arraycontent_values_ref;
    }
```

```
   return $results_ref;
}

##-----------------------------------------------------------------
## do_localflow : produces all the test case variants for the
##  "local control flow" attribute.
##-----------------------------------------------------------------
sub do_localflow
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # if = 1
        my $if_values_ref = get_hash_copy($entry);
        $if_values_ref->{'TAX'}->[$LOCALFLOW_DIGIT] = 1;
        $if_values_ref->{'OTHER_DECL'} = "  int flag;\n" .
            $if_values_ref->{'OTHER_DECL'};
        $if_values_ref->{'PRE_ACCESS'} = "  flag = 1;\n" .
              $if_values_ref->{'PRE_ACCESS'} .
              "  if (flag)\n  {\n";
        $if_values_ref->{'ACCESS'} =
            indent(2, $if_values_ref->{'ACCESS'});
        $if_values_ref->{'POST_ACCESS'} = "  }\n" .
            $if_values_ref->{'POST_ACCESS'};
        push @$results_ref, $if_values_ref;

        # switch = 2
        my $switch_values_ref = get_hash_copy($entry);
        $switch_values_ref->{'TAX'}->[$LOCALFLOW_DIGIT] = 2;
        $switch_values_ref->{'OTHER_DECL'} = "  int flag;\n" .
            $switch_values_ref->{'OTHER_DECL'};
        $switch_values_ref->{'PRE_ACCESS'} = "  flag = 1;\n" .
            $switch_values_ref->{'PRE_ACCESS'} .
            "  switch (flag)\n  {\n    case 1:\n";
        $switch_values_ref->{'ACCESS'} =
            indent(4, $switch_values_ref->{'ACCESS'});
        $switch_values_ref->{'POST_ACCESS'} =
            "      break;\n    default:\n      break;\n  }\n" .
            $switch_values_ref->{'POST_ACCESS'};
        push @$results_ref, $switch_values_ref;

        # cond = 3
        my $cond_values_ref = get_hash_copy($entry);
        $cond_values_ref->{'TAX'}->[$LOCALFLOW_DIGIT] = 3;
        $cond_values_ref->{'OTHER_DECL'} = "  int flag;\n" .
            $cond_values_ref->{'OTHER_DECL'};
        $cond_values_ref->{'PRE_ACCESS'} = "  flag = 1;\n" .
            $cond_values_ref->{'PRE_ACCESS'};
        # strip off the leading comment and the trailing ;\n
        $cond_values_ref->{'ACCESS'} =~ s/COMMENT\n//;
        substr($cond_values_ref->{'ACCESS'}, -2) =~ s/;\n//;
        $cond_values_ref->{'ACCESS'} = "COMMENT\n" .
            "  flag ? $cond_values_ref->{'ACCESS'} : 0;\n";
        push @$results_ref, $cond_values_ref;

        # goto = 4
        my $goto_values_ref = get_hash_copy($entry);
        $goto_values_ref->{'TAX'}->[$LOCALFLOW_DIGIT] = 4;
        $goto_values_ref->{'PRE_ACCESS'} =
            "  goto label1;\n  return 0;\n" .
            $goto_values_ref->{'PRE_ACCESS'} .
            "label1:\n";
        push @$results_ref, $goto_values_ref;

        # longjmp = 5;
        my $lj_values_ref = get_hash_copy($entry);
        $lj_values_ref->{'TAX'}->[$LOCALFLOW_DIGIT] = 5;
        $lj_values_ref->{'INCL'} = "#include <setjmp.h>\n" .
            $lj_values_ref->{'INCL'};
        $lj_values_ref->{'OTHER_DECL'} =
            "  jmp_buf env;\n" . $lj_values_ref->{'OTHER_DECL'};
        $lj_values_ref->{'PRE_ACCESS'} =
            "  if (setjmp(env) != 0)\n  {\n    return 0;\n  }\n" .
            $lj_values_ref->{'PRE_ACCESS'};
        $lj_values_ref->{'POST_ACCESS'} =
            $lj_values_ref->{'POST_ACCESS'} .
            "  longjmp(env, 1);\n";
        push @$results_ref, $lj_values_ref;

        # function pointer = 6
        my $funcptr_values_ref = get_hash_copy($entry);
        $funcptr_values_ref->{'TAX'}->[$LOCALFLOW_DIGIT] = 6;
        $funcptr_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 1;   #inter-
procedural
        $funcptr_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1;   #1
alias of addr
        $funcptr_values_ref->{'BEFORE_MAIN'} =
            "void function1\(TYPE * buf\)\n{\n" .
            $funcptr_values_ref->{'ACCESS'} .
            "}\n\n" . $funcptr_values_ref->{'BEFORE_MAIN'};
        $funcptr_values_ref->{'OTHER_DECL'} =
            "  void (*fptr)(TYPE *);\n" .
            $funcptr_values_ref->{'OTHER_DECL'};
        $funcptr_values_ref->{'PRE_ACCESS'} = "  fptr = function1;\n" .
            $funcptr_values_ref->{'PRE_ACCESS'};
        $funcptr_values_ref->{'ACCESS'} = "  fptr\(buf\);\n";
        push @$results_ref, $funcptr_values_ref;

        # recursion = 7
        my $recur_values_ref = get_hash_copy($entry);
        $recur_values_ref->{'TAX'}->[$LOCALFLOW_DIGIT] = 7;
        $recur_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 1;   #inter-
procedural
        $recur_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1;   #1 alias
of addr
        $recur_values_ref->{'BEFORE_MAIN'} =
            "void function1\(TYPE * buf, int counter\)\n{\n" .
            "  if \(counter > 0\)\n  {\n" .
            "    function1\(buf, counter - 1\);\n  }\n" .
            $recur_values_ref->{'ACCESS'} .
            "}\n\n" . $recur_values_ref->{'BEFORE_MAIN'};
        $recur_values_ref->{'ACCESS'} = "  function1\(buf, 3\);\n";
        push @$results_ref, $recur_values_ref;
    }

    return $results_ref;
}

##-----------------------------------------------------------------
## do_loopcomplex : produces all the test case variants for the
##  "loop complexity" attribute.
##-----------------------------------------------------------------
sub do_loopcomplex
{
    my $results_ref = [];

    # NOTE: these are all variations on what's produced by
    # do_loopstructure

    # 0 is baseline - no loop
    # 1 is standard complexity, which is already produced by
    # loopstructure

    #  2 one - one of the three is more complex than the baseline
    #  first we'll change the initialization
    my $complex1_init_array = do_loopstructure($_[0]);
    foreach my $variant_ref (@$complex1_init_array)
    {
        $variant_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 2;
        $variant_ref->{'OTHER_DECL'} = "  int init_value;\n" .
                                        $variant_ref->{'OTHER_DECL'};
        $variant_ref->{'PRE_ACCESS'} = "  init_value = 0;\n" .
                                        $variant_ref->{'PRE_ACCESS'};
        $variant_ref->{'SINGLES'}->{'LOOP_INIT'} =~
            s/=\s*?0/= init_value/;
        push @$results_ref, $variant_ref;
    }

    #  2 one - one of the three is more complex than the baseline
    #  next we'll change the test
    my $complex1_test_array = do_loopstructure($_[0]);
    foreach my $variant_ref (@$complex1_test_array)
    {
        $variant_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 2;
        $variant_ref->{'OTHER_DECL'} = "  int test_value;\n" .
                                        $variant_ref->{'OTHER_DECL'};
        $variant_ref->{'PRE_ACCESS'} = "  test_value = INDEX;\n" .
                                        $variant_ref->{'PRE_ACCESS'};
        $variant_ref->{'SINGLES'}->{'LOOP_TEST'} =~
            s/INDEX/test_value/;
        push @$results_ref, $variant_ref;
    }
```

```
#  2 one - one of the three is more complex than the baseline
#  last we'll change the inc
my $complex1_inc_array = do_loopstructure($_[0]);
foreach my $variant_ref (@$complex1_inc_array)
{
    $variant_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 2;
    $variant_ref->{'OTHER_DECL'} = "  int inc_value;\n" .
                                    $variant_ref->{'OTHER_DECL'};
    $variant_ref->{'PRE_ACCESS'} =
        "  inc_value = INDEX - (INDEX - 1);\n" .
                                    $variant_ref->{'PRE_ACCESS'};
    $variant_ref->{'SINGLES'}->{'LOOP_INC'} =
        "loop_counter += inc_value";
    push @$results_ref, $variant_ref;
}

#  3 two - two of the three are more complex than the baseline
#  first we'll change the initialization and test
my $complex2_inittest_array = do_loopstructure($_[0]);
foreach my $variant_ref (@$complex2_inittest_array)
{
    $variant_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 3;
    $variant_ref->{'OTHER_DECL'} =
        "  int init_value;\n  int test_value;\n" .
                                   $variant_ref->{'OTHER_DECL'};
    $variant_ref->{'PRE_ACCESS'} =
        "  init_value = 0;\n  test_value = INDEX;\n" .
                                    $variant_ref->{'PRE_ACCESS'};
    $variant_ref->{'SINGLES'}->{'LOOP_INIT'} =~
        s/=\s*?0/= init_value/;
    $variant_ref->{'SINGLES'}->{'LOOP_TEST'} =~
         s/INDEX/test_value/;
    push @$results_ref, $variant_ref;
}

#  3 two - two of the three are more complex than the baseline
#  next we'll change the initialization and increment
my $complex2_initinc_array = do_loopstructure($_[0]);
foreach my $variant_ref (@$complex2_initinc_array)
{
    $variant_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 3;
    $variant_ref->{'OTHER_DECL'} =
        "  int init_value;\n  int inc_value;\n" .
                                   $variant_ref->{'OTHER_DECL'};
    $variant_ref->{'PRE_ACCESS'} = "  init_value = 0;\n" .
                          "  inc_value = INDEX - (INDEX - 1);\n" .
                                    $variant_ref->{'PRE_ACCESS'};
    $variant_ref->{'SINGLES'}->{'LOOP_INIT'} =~
        s/=\s*?0/= init_value/;
    $variant_ref->{'SINGLES'}->{'LOOP_INC'} =
        "loop_counter += inc_value";
    push @$results_ref, $variant_ref;
}

#  3 two - two of the three are more complex than the baseline
```

```
    #  last we'll change the test and increment
    my $complex2_testinc_array = do_loopstructure($_[0]);
    foreach my $variant_ref (@$complex2_testinc_array)
    {
        $variant_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 3;
        $variant_ref->{'OTHER_DECL'} =
            "  int test_value;\n  int inc_value;\n" .
                                $variant_ref->{'OTHER_DECL'};
        $variant_ref->{'PRE_ACCESS'} = "  test_value = INDEX;\n" .
                              "  inc_value = INDEX - (INDEX - 1);\n" .
                                $variant_ref->{'PRE_ACCESS'};
        $variant_ref->{'SINGLES'}->{'LOOP_TEST'} =~
            s/INDEX/test_value/;
        $variant_ref->{'SINGLES'}->{'LOOP_INC'} =
            "loop_counter += inc_value";
        push @$results_ref, $variant_ref;
    }

    #  4 three - all three are more complex than the baseline
    #  change the init, test, and increment
    my $complex3_inittestinc_array = do_loopstructure($_[0]);
    foreach my $variant_ref (@$complex3_inittestinc_array)
    {
        $variant_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 4;
        $variant_ref->{'OTHER_DECL'} =
            "  int init_value;\n  int test_value;\n" .
                                "  int inc_value;\n" .
                                $variant_ref->{'OTHER_DECL'};
        $variant_ref->{'PRE_ACCESS'} = "  init_value = 0;\n" .
                                "  test_value = INDEX;\n" .
                              "  inc_value = INDEX - (INDEX - 1);\n" .
                                $variant_ref->{'PRE_ACCESS'};
        $variant_ref->{'SINGLES'}->{'LOOP_INIT'} =~
            s/=\s*?0/= init_value/;
        $variant_ref->{'SINGLES'}->{'LOOP_TEST'} =~
            s/INDEX/test_value/;
        $variant_ref->{'SINGLES'}->{'LOOP_INC'} =
            "loop_counter += inc_value";
        push @$results_ref, $variant_ref;
    }

    return $results_ref;
}

##--------------------------------------------------------------
## do_loopstructure : produces all the test case variants for the
##  "loop structure" attribute.
##--------------------------------------------------------------
sub do_loopstructure
{
    my $standard_loop_init = "loop_counter = 0";
    my $standard_loop_test = "loop_counter <= INDEX";
    my $standard_loop_postinc = "loop_counter++";
    my $standard_loop_preinc = "++loop_counter";
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # standard for = 1
        my $for_values_ref = get_hash_copy($entry);
        $for_values_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] = 1;
        $for_values_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 1; #standard
        $for_values_ref->{'SINGLES'}->{'LOOP_INIT'} =
            $standard_loop_init;
        $for_values_ref->{'SINGLES'}->{'LOOP_TEST'} =
            $standard_loop_test;
        $for_values_ref->{'SINGLES'}->{'LOOP_INC'} =
            $standard_loop_postinc;
        $for_values_ref->{'OTHER_DECL'} = "  int loop_counter;\n" .
            $for_values_ref->{'OTHER_DECL'};
        $for_values_ref->{'PRE_ACCESS'} =
            $for_values_ref->{'PRE_ACCESS'} .
            "  for(LOOP_INIT; LOOP_TEST; LOOP_INC)\n  {\n";
        $for_values_ref->{'ACCESS'} =
            indent(2, $for_values_ref->{'ACCESS'});
        $for_values_ref->{'POST_ACCESS'} =
            "  }\n" . $for_values_ref->{'POST_ACCESS'};
        push @$results_ref, $for_values_ref;

        # standard do-while = 2
        my $dowhile_values_ref = get_hash_copy($entry);
        $dowhile_values_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] = 2;
        $dowhile_values_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 1;
#standard
        $dowhile_values_ref->{'SINGLES'}->{'LOOP_INIT'} =
            $standard_loop_init;
        $dowhile_values_ref->{'SINGLES'}->{'LOOP_TEST'} =
            $standard_loop_test;
        $dowhile_values_ref->{'SINGLES'}->{'LOOP_INC'} =
            $standard_loop_postinc;
        $dowhile_values_ref->{'OTHER_DECL'} = "  int loop_counter;\n" .
$dowhile_values_ref->{'OTHER_DECL'};
        $dowhile_values_ref->{'PRE_ACCESS'} = $dowhile_values_ref-
>{'PRE_ACCESS'} .
                                        "  LOOP_INIT;\n  do\n  {\n";
        $dowhile_values_ref->{'ACCESS'} =
            indent(2, $dowhile_values_ref->{'ACCESS'});
        $dowhile_values_ref->{'POST_ACCESS'} =
            "    LOOP_INC;\n  }\n  while(LOOP_TEST);\n" .
            $dowhile_values_ref->{'POST_ACCESS'};
        push @$results_ref, $dowhile_values_ref;

        # standard while = 3
        my $while_values_ref = get_hash_copy($entry);
        $while_values_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] = 3;
        $while_values_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 1; #standard
        $while_values_ref->{'SINGLES'}->{'LOOP_INIT'} =
            $standard_loop_init;
        $while_values_ref->{'SINGLES'}->{'LOOP_TEST'} =
             $standard_loop_test;
        $while_values_ref->{'SINGLES'}->{'LOOP_INC'} =
            $standard_loop_postinc;
        $while_values_ref->{'OTHER_DECL'} =
            "  int loop_counter;\n" .
            $while_values_ref->{'OTHER_DECL'};
        $while_values_ref->{'PRE_ACCESS'} =
            $while_values_ref->{'PRE_ACCESS'} .
            "  LOOP_INIT;\n  while(LOOP_TEST)\n  {\n";
        $while_values_ref->{'ACCESS'} =
            indent(2, $while_values_ref->{'ACCESS'});
        $while_values_ref->{'POST_ACCESS'} =
            "    LOOP_INC;\n  }\n" .
            $while_values_ref->{'POST_ACCESS'};
        push @$results_ref, $while_values_ref;

        # non-standard for = 4
        # first variation: move the init clause
        my $nsfor1_values_ref = get_hash_copy($entry);
        $nsfor1_values_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] = 4;
        $nsfor1_values_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 1;
#standard
        $nsfor1_values_ref->{'SINGLES'}->{'LOOP_INIT'} =
            $standard_loop_init;
        $nsfor1_values_ref->{'SINGLES'}->{'LOOP_TEST'} =
            $standard_loop_test;
        $nsfor1_values_ref->{'SINGLES'}->{'LOOP_INC'} =
            $standard_loop_postinc;
        $nsfor1_values_ref->{'OTHER_DECL'} =
            "  int loop_counter;\n" .
            $nsfor1_values_ref->{'OTHER_DECL'};
        $nsfor1_values_ref->{'PRE_ACCESS'} =
            $nsfor1_values_ref->{'PRE_ACCESS'} .
                                     "  LOOP_INIT;\n" .
                          "  for( ; LOOP_TEST; LOOP_INC)\n  {\n";
        $nsfor1_values_ref->{'ACCESS'} =
            indent(2, $nsfor1_values_ref->{'ACCESS'});
        $nsfor1_values_ref->{'POST_ACCESS'} = "  }\n" .
             $nsfor1_values_ref->{'POST_ACCESS'};
        push @$results_ref, $nsfor1_values_ref;

        # non-standard for = 4
        # second variation: move the test clause
        # Note that the explicit "if" counts as secondary flow control
        my $nsfor2_values_ref = get_hash_copy($entry);
        $nsfor2_values_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] = 4;
        $nsfor2_values_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 1;
#standard
        $nsfor2_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 1;  # if
        $nsfor2_values_ref->{'SINGLES'}->{'LOOP_INIT'} =
            $standard_loop_init;
        $nsfor2_values_ref->{'SINGLES'}->{'LOOP_TEST'} =
            "loop_counter > INDEX";
        $nsfor2_values_ref->{'SINGLES'}->{'LOOP_INC'} =
            $standard_loop_postinc;
        $nsfor2_values_ref->{'OTHER_DECL'} =
            "  int loop_counter;\n" .
            $nsfor2_values_ref->{'OTHER_DECL'};
        $nsfor2_values_ref->{'PRE_ACCESS'} =
            $nsfor2_values_ref->{'PRE_ACCESS'} .
            "  for(LOOP_INIT; ; LOOP_INC)\n  {\n" .
            "    if (LOOP_TEST) break;\n";
        $nsfor2_values_ref->{'ACCESS'} =
            indent(2, $nsfor2_values_ref->{'ACCESS'});
        $nsfor2_values_ref->{'POST_ACCESS'} = "  }\n" .
            $nsfor2_values_ref->{'POST_ACCESS'};
        push @$results_ref, $nsfor2_values_ref;

        # non-standard for = 4
        # third variation: move the increment clause
        my $nsfor3_values_ref = get_hash_copy($entry);
        $nsfor3_values_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] = 4;
        $nsfor3_values_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 1;
#standard
        $nsfor3_values_ref->{'SINGLES'}->{'LOOP_INIT'} =
            $standard_loop_init;
        $nsfor3_values_ref->{'SINGLES'}->{'LOOP_TEST'} =
            $standard_loop_test;
        $nsfor3_values_ref->{'SINGLES'}->{'LOOP_INC'} =
            $standard_loop_postinc;
        $nsfor3_values_ref->{'OTHER_DECL'} = "  int loop_counter;\n" .
            $nsfor3_values_ref->{'OTHER_DECL'};
        $nsfor3_values_ref->{'PRE_ACCESS'} =
            $nsfor3_values_ref->{'PRE_ACCESS'} .
            "  for(LOOP_INIT; LOOP_TEST; )\n  {\n";
        $nsfor3_values_ref->{'ACCESS'} =
            indent(2, $nsfor3_values_ref->{'ACCESS'});
        $nsfor3_values_ref->{'POST_ACCESS'} = "    LOOP_INC;\n  }\n" .
            $nsfor3_values_ref->{'POST_ACCESS'};
        push @$results_ref, $nsfor3_values_ref;

        # non-standard for = 4
        # fourth variation: move all three clauses
        # Note that the explicit "if" counts as secondary flow control
        my $nsfor4_values_ref = get_hash_copy($entry);
        $nsfor4_values_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] = 4;
        $nsfor4_values_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 1;
#standard
        $nsfor4_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 1;    #
if
        $nsfor4_values_ref->{'SINGLES'}->{'LOOP_INIT'} =
            $standard_loop_init;
        $nsfor4_values_ref->{'SINGLES'}->{'LOOP_TEST'} =
            "loop_counter > INDEX";
        $nsfor4_values_ref->{'SINGLES'}->{'LOOP_INC'} =
            $standard_loop_postinc;
        $nsfor4_values_ref->{'OTHER_DECL'} = "  int loop_counter;\n" .
            $nsfor4_values_ref->{'OTHER_DECL'};
        $nsfor4_values_ref->{'PRE_ACCESS'} =
            $nsfor4_values_ref->{'PRE_ACCESS'} .
                                         "  LOOP_INIT;\n" .
                                         "  for( ; ; )\n  {\n" .
                                         "    if (LOOP_TEST) break;\n";
        $nsfor4_values_ref->{'ACCESS'} =
            indent(2, $nsfor4_values_ref->{'ACCESS'});
        $nsfor4_values_ref->{'POST_ACCESS'} =
            "    LOOP_INC;\n  }\n" .
            $nsfor4_values_ref->{'POST_ACCESS'};
        push @$results_ref, $nsfor4_values_ref;

        # non-standard do-while = 5
        # first variation: move the increment (combine with test)
        my $nsdowhile1_values_ref = get_hash_copy($entry);
        $nsdowhile1_values_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] = 5;
        $nsdowhile1_values_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 1;
#standard
        $nsdowhile1_values_ref->{'SINGLES'}->{'LOOP_INIT'} =
            $standard_loop_init;
        $nsdowhile1_values_ref->{'SINGLES'}->{'LOOP_TEST'} =
            $standard_loop_test;
        $nsdowhile1_values_ref->{'SINGLES'}->{'LOOP_INC'} =
            $standard_loop_preinc;
        $nsdowhile1_values_ref->{'OTHER_DECL'} =
            "  int loop_counter;\n" .
            $nsdowhile1_values_ref->{'OTHER_DECL'};
        $nsdowhile1_values_ref->{'PRE_ACCESS'} =
            $nsdowhile1_values_ref->{'PRE_ACCESS'} .
                                         "  LOOP_INIT;\n  do\n  {\n";
        $nsdowhile1_values_ref->{'ACCESS'} =
            indent(2, $nsdowhile1_values_ref->{'ACCESS'});
        $nsdowhile1_values_ref->{'POST_ACCESS'} =
            "  }\n  while((LOOP_INC) && (LOOP_TEST));\n" .
            $nsdowhile1_values_ref->{'POST_ACCESS'};
        push @$results_ref, $nsdowhile1_values_ref;

        # non-standard do-while = 5
        # second variation: move the test
        # Note that the explicit "if" counts as secondary flow control
        my $nsdowhile2_values_ref = get_hash_copy($entry);
        $nsdowhile2_values_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] = 5;
        $nsdowhile2_values_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 1;
#standard
        $nsdowhile2_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 1;
# if
        $nsdowhile2_values_ref->{'SINGLES'}->{'LOOP_INIT'} =
            $standard_loop_init;
        $nsdowhile2_values_ref->{'SINGLES'}->{'LOOP_TEST'} =
            "loop_counter > INDEX";
        $nsdowhile2_values_ref->{'SINGLES'}->{'LOOP_INC'} =
            $standard_loop_postinc;
        $nsdowhile2_values_ref->{'OTHER_DECL'} =
            "  int loop_counter;\n" .
            $nsdowhile2_values_ref->{'OTHER_DECL'};
        $nsdowhile2_values_ref->{'PRE_ACCESS'} =
            $nsdowhile2_values_ref->{'PRE_ACCESS'} .
            "  LOOP_INIT;\n  do\n  {\n";
        $nsdowhile2_values_ref->{'ACCESS'} =
            indent(2, $nsdowhile2_values_ref->{'ACCESS'});
        $nsdowhile2_values_ref->{'POST_ACCESS'} =
            "    LOOP_INC;\n" . "    if (LOOP_TEST) break;\n" .
            "  }\n  while(1);\n" .
            $nsdowhile2_values_ref->{'POST_ACCESS'};
        push @$results_ref, $nsdowhile2_values_ref;

        # non-standard do-while = 5
        # third variation: move both test and increment
        # Note that the explicit "if" counts as secondary flow control
        my $nsdowhile3_values_ref = get_hash_copy($entry);
        $nsdowhile3_values_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] = 5;
        $nsdowhile3_values_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 1;
#standard
        $nsdowhile3_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 1;
# if
        $nsdowhile3_values_ref->{'SINGLES'}->{'LOOP_INIT'} =
            $standard_loop_init;
        $nsdowhile3_values_ref->{'SINGLES'}->{'LOOP_TEST'} =
            "loop_counter >= INDEX";
        $nsdowhile3_values_ref->{'SINGLES'}->{'LOOP_INC'} =
            $standard_loop_preinc;
        $nsdowhile3_values_ref->{'OTHER_DECL'} =
            "  int loop_counter;\n" .
            $nsdowhile3_values_ref->{'OTHER_DECL'};
        $nsdowhile3_values_ref->{'PRE_ACCESS'} =
            $nsdowhile3_values_ref->{'PRE_ACCESS'} .
            "  LOOP_INIT;\n  do\n  {\n";
        $nsdowhile3_values_ref->{'ACCESS'} =
            indent(2, $nsdowhile3_values_ref->{'ACCESS'});
        $nsdowhile3_values_ref->{'POST_ACCESS'} =
            "    if (LOOP_TEST) break;\n" .
            "  }\n  while(LOOP_INC);\n" .
            $nsdowhile3_values_ref->{'POST_ACCESS'};
        push @$results_ref, $nsdowhile3_values_ref;

        # non-standard while = 6
        # first variation: move test
        # Note that the explicit "if" counts as secondary flow control
        my $nswhile1_values_ref = get_hash_copy($entry);
        $nswhile1_values_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] = 6;
        $nswhile1_values_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 1;
#standard
        $nswhile1_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 1;    #
if
        $nswhile1_values_ref->{'SINGLES'}->{'LOOP_INIT'} =
            $standard_loop_init;
        $nswhile1_values_ref->{'SINGLES'}->{'LOOP_TEST'} =
            "loop_counter > INDEX";
        $nswhile1_values_ref->{'SINGLES'}->{'LOOP_INC'} =
            $standard_loop_postinc;
        $nswhile1_values_ref->{'OTHER_DECL'} =
            "  int loop_counter;\n" .
            $nswhile1_values_ref->{'OTHER_DECL'};
        $nswhile1_values_ref->{'PRE_ACCESS'} =
            $nswhile1_values_ref->{'PRE_ACCESS'} .
            "  LOOP_INIT;\n  while(1)\n  {\n";
        $nswhile1_values_ref->{'ACCESS'} =
            indent(2, $nswhile1_values_ref->{'ACCESS'});
        $nswhile1_values_ref->{'POST_ACCESS'} =
            "    LOOP_INC;\n" . "    if (LOOP_TEST) break;\n  }\n" .
            $nswhile1_values_ref->{'POST_ACCESS'};
        push @$results_ref, $nswhile1_values_ref;

        # non-standard while = 6
        # second variation: move increment
        my $nswhile2_values_ref = get_hash_copy($entry);
        $nswhile2_values_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] = 6;
        $nswhile2_values_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 1;
#standard
        $nswhile2_values_ref->{'SINGLES'}->{'LOOP_INIT'} =
            $standard_loop_init;
        $nswhile2_values_ref->{'SINGLES'}->{'LOOP_TEST'} =
            $standard_loop_test;
        $nswhile2_values_ref->{'SINGLES'}->{'LOOP_INC'} =
            $standard_loop_preinc;
        $nswhile2_values_ref->{'OTHER_DECL'} =
            "  int loop_counter;\n" .
            $nswhile2_values_ref->{'OTHER_DECL'};
        $nswhile2_values_ref->{'PRE_ACCESS'} =
            $nswhile2_values_ref->{'PRE_ACCESS'} .
            "  LOOP_INIT;\n  while((LOOP_INC) && (LOOP_TEST))\n  {\n";
        $nswhile2_values_ref->{'ACCESS'} =
            indent(2, $nswhile2_values_ref->{'ACCESS'});
        $nswhile2_values_ref->{'POST_ACCESS'} = "  }\n" .
            $nswhile2_values_ref->{'POST_ACCESS'};
        push @$results_ref, $nswhile2_values_ref;

        # non-standard while = 6
        # third variation: move both test and increment
        # Note that the explicit "if" counts as secondary flow control
        my $nswhile3_values_ref = get_hash_copy($entry);
        $nswhile3_values_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] = 6;
        $nswhile3_values_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] = 1;
#standard
        $nswhile3_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 1;    #
if
        $nswhile3_values_ref->{'SINGLES'}->{'LOOP_INIT'} =
            $standard_loop_init;
        $nswhile3_values_ref->{'SINGLES'}->{'LOOP_TEST'} =
            "loop_counter >= INDEX";
        $nswhile3_values_ref->{'SINGLES'}->{'LOOP_INC'} =
            $standard_loop_preinc;
        $nswhile3_values_ref->{'OTHER_DECL'} =
            "  int loop_counter;\n" .
            $nswhile3_values_ref->{'OTHER_DECL'};
        $nswhile3_values_ref->{'PRE_ACCESS'} =
            $nswhile3_values_ref->{'PRE_ACCESS'} .
            "  LOOP_INIT;\n  while(LOOP_INC)\n  {\n";
        $nswhile3_values_ref->{'ACCESS'} =
            indent(2, $nswhile3_values_ref->{'ACCESS'});
        $nswhile3_values_ref->{'POST_ACCESS'} =
            "    if (LOOP_TEST) break;\n  }\n" .
            $nswhile3_values_ref->{'POST_ACCESS'};
        push @$results_ref, $nswhile3_values_ref;
    }

    return $results_ref;
}

##----------------------------------------------------------------
## do_magnitude : produces all the test case variants for the
##  "magnitude" attribute.
##----------------------------------------------------------------
sub do_magnitude
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    # don't need to produce anything here, because we automatically
    # vary the magnitude across all the other test cases we produce

    return $results_ref;
}

##----------------------------------------------------------------
## do_memloc : produces all the test case variants for the "memory
##  location" attribute.
##----------------------------------------------------------------
sub do_memloc
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # heap = 1
        my $heap_values_ref = get_hash_copy($entry);
        $heap_values_ref->{'TAX'}->[$MEMLOC_DIGIT] = 1;
        $heap_values_ref->{'INCL'} =
            "#include <stdlib.h>\n#include <assert.h>\n" .
            $heap_values_ref->{'INCL'};
        $heap_values_ref->{'BUF_DECL'} = "  TYPE * buf;\n\n";
        $heap_values_ref->{'PRE_ACCESS'} =
            "  buf = \(TYPE *\) malloc\($BUF_SIZE\ *
sizeof\(TYPE\)\);\n" .
            "  assert \(buf != NULL\);\n" .
            $heap_values_ref->{'PRE_ACCESS'};
        push @$results_ref, $heap_values_ref;

        # data region = 2
        my $data_values_ref = get_hash_copy($entry);
        $data_values_ref->{'TAX'}->[$MEMLOC_DIGIT] = 2;
        # COMBO NOTE: need to make this init value more generic and
dependent on TYPE.
        # Needs to be updated in order for data type combos to work.
        $data_values_ref->{'BUF_DECL'} = "  static " .
            $data_values_ref->{'BUF_DECL'};
        $data_values_ref->{'BUF_DECL'} =~ s/\;/ = \"\"\;/;
        push @$results_ref, $data_values_ref;

        # bss = 3
        my $bss_values_ref = get_hash_copy($entry);
        $bss_values_ref->{'TAX'}->[$MEMLOC_DIGIT] = 3;
        $bss_values_ref->{'BUF_DECL'} = "  static " .
            $bss_values_ref->{'BUF_DECL'};
        push @$results_ref, $bss_values_ref;

        # shared = 4
        my $shared_values_ref = get_hash_copy($entry);
        $shared_values_ref->{'TAX'}->[$MEMLOC_DIGIT] = 4;
        $shared_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 1; # inter-
procedural
        $shared_values_ref->{'INCL'} =
            "#include <sys/types.h>\n#include <sys/ipc.h>\n" .
            "#include <sys/shm.h>\n#include <assert.h>\n" .
            "#include <stdlib.h>\n" . $shared_values_ref->{'INCL'};
        $shared_values_ref->{'BEFORE_MAIN'} =
            "int getSharedMem()\n{\n" .
            "  return (shmget(IPC_PRIVATE, $BUF_SIZE,
0xffffffff));\n}\n\n" .
            "void relSharedMem(int memID)\n{\n  struct shmid_ds
temp;\n" .
            "  shmctl(memID, IPC_RMID, &temp);\n}\n\n" .
            $shared_values_ref->{'BEFORE_MAIN'};
        $shared_values_ref->{'OTHER_DECL'} =
            "  int memIdent;\n" . $shared_values_ref->{'OTHER_DECL'};
        $shared_values_ref->{'BUF_DECL'} = "  TYPE * buf;\n\n";
        $shared_values_ref->{'PRE_ACCESS'} =
          "  memIdent = getSharedMem();\n  assert(memIdent != -1);\n" .
            "  buf = ((TYPE *) shmat(memIdent, NULL, 0));\n
      assert(((int)buf) != -1);\n" .
            $shared_values_ref->{'PRE_ACCESS'};
        $shared_values_ref->{'POST_ACCESS'} =
            $shared_values_ref->{'POST_ACCESS'} .
            "  shmdt((void *)buf);\n  relSharedMem(memIdent);\n";
        push @$results_ref, $shared_values_ref;
    }

    return $results_ref;
}

##-------------------------------------------------------------
## do_pointer : produces all the test case variants for the "pointer"
##  attribute.
##-------------------------------------------------------------
sub do_pointer
```

```
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # yes = 1
        my $pointer_values_ref = get_hash_copy($entry);
        $pointer_values_ref->{'TAX'}->[$POINTER_DIGIT] = 1;
        $pointer_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; # N/A
- no index
        $pointer_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A -
no index
        $pointer_values_ref->{'ACCESS'} =
            "COMMENT\n  *(buf + INDEX) = WRITE_VALUE;\n";
        push @$results_ref, $pointer_values_ref;
    }

    return $results_ref;
}

##----------------------------------------------------------------
## do_runtimeenvdep : produces all the test case variants for the
##  "runtime environment dependent" attribute.
##----------------------------------------------------------------
sub do_runtimeenvdep
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    # we don't need to produce anything here, because we cover both
    # possibilities in the other test cases we produce

    return $results_ref;
}

##----------------------------------------------------------------
## do_scope : produces all the test case variants for the "scope"
##  attribute.
##----------------------------------------------------------------
sub do_scope
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # inter-procedural = 1
        my $interproc_values_ref = get_hash_copy($entry);
        $interproc_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 1;
        $interproc_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1;   # 1
function arg alias
        $interproc_values_ref->{'BEFORE_MAIN'} =
            "void function1\(TYPE * buf\)\n{\n" .
            $interproc_values_ref->{'ACCESS'} .
            "}\n\n" . $interproc_values_ref->{'BEFORE_MAIN'};
        $interproc_values_ref->{'ACCESS'} = "  function1\(buf\);\n";
        push @$results_ref, $interproc_values_ref;

        # global = 2
        # generate 2 variations, because global declaration has to be
        #   either bss (uninitialized) or data segment (initialized)
        my $global1_values_ref = get_hash_copy($entry);
        $global1_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 2;
        $global1_values_ref->{'TAX'}->[$MEMLOC_DIGIT] = 3; # bss
        $global1_values_ref->{'FILE_GLOBAL'} = "static " .
            $global1_values_ref->{'BUF_DECL'};
        $global1_values_ref->{'BUF_DECL'} = "";
        push @$results_ref, $global1_values_ref;

        my $global2_values_ref = get_hash_copy($entry);
        $global2_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 2;
        $global2_values_ref->{'TAX'}->[$MEMLOC_DIGIT] = 2; # data
segment
        $global2_values_ref->{'FILE_GLOBAL'} = "static " .
            $global2_values_ref->{'BUF_DECL'};
        $global2_values_ref->{'FILE_GLOBAL'} =~ s/\;/ = \"\"\;/;
        $global2_values_ref->{'BUF_DECL'} = "";
        push @$results_ref, $global2_values_ref;

        # Note: there are some examples of inter-file/inter-proc in
        # test cases that call library functions.
        # FUTURE: add some examples of these two variations where the
        # overwriting function is user-defined, not a library function.
        # inter-file/inter-proc = 3
        # inter-file/global = 4
    }

    return $results_ref;
}

##-------------------------------------------------------------
## do_secondaryflow : produces all the test case variants for the
##  "secondary control flow" attribute.
##-------------------------------------------------------------
sub do_secondaryflow
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # if = 1
        my $if_values_ref = get_hash_copy($entry);
        $if_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 1;
        $if_values_ref->{'PRE_ACCESS'} =
            "  if (sizeof buf > INDEX + 1)\n  {\n    " .
            "return 0;\n  }\n" .
            $if_values_ref->{'PRE_ACCESS'};
        push @$results_ref, $if_values_ref;
        # switch = 2
        my $switch_values_ref = get_hash_copy($entry);
        $switch_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 2;
        $switch_values_ref->{'PRE_ACCESS'} =
            "  switch (sizeof buf > INDEX + 1)\n  {\n    case 1:\n" .
            "      return 0;\n    default:\n      break;\n  }\n" .
            $switch_values_ref->{'PRE_ACCESS'};
        push @$results_ref, $switch_values_ref;

        # cond = 3
        my $cond_values_ref = get_hash_copy($entry);
        $cond_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 3;
        $cond_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 1; #
variable
        $cond_values_ref->{'OTHER_DECL'} = "  int i;\n" .
            $cond_values_ref->{'OTHER_DECL'};
        $cond_values_ref->{'PRE_ACCESS'} =
            "  i = (sizeof buf > INDEX + 1) ? 0 : INDEX;\n" .
            $cond_values_ref->{'PRE_ACCESS'};
        $cond_values_ref->{'ACCESS'} =~ s/INDEX/i/;
        push @$results_ref, $cond_values_ref;

        # goto = 4
        my $goto_values_ref = get_hash_copy($entry);
        $goto_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 4;
        $goto_values_ref->{'TAX'}->[$LOCALFLOW_DIGIT] = 1;  # local if
        $goto_values_ref->{'OTHER_DECL'} = "  int flag;\n" .
            $goto_values_ref->{'OTHER_DECL'};
        $goto_values_ref->{'PRE_ACCESS'} =
            "  goto label1;\n  return 0;\n" .
            $goto_values_ref->{'PRE_ACCESS'} .
            "label1:\n  flag = 1;\n" .
            "  if (flag)\n  {\n";
        $goto_values_ref->{'ACCESS'} =
            indent(2, $goto_values_ref->{'ACCESS'});
        $goto_values_ref->{'POST_ACCESS'} = "  }\n" .
            $goto_values_ref->{'POST_ACCESS'};
        push @$results_ref, $goto_values_ref;

        # longjmp = 5;
        my $lj_values_ref = get_hash_copy($entry);
        $lj_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 5;
        $lj_values_ref->{'TAX'}->[$LOCALFLOW_DIGIT] = 1;  # local if
        $lj_values_ref->{'INCL'} = "#include <setjmp.h>\n" .
            $lj_values_ref->{'INCL'};
        $lj_values_ref->{'OTHER_DECL'} =
            "  jmp_buf env;\n  int flag;\n" .
            $lj_values_ref->{'OTHER_DECL'};
        $lj_values_ref->{'PRE_ACCESS'} =
            "  if (setjmp(env) != 0)\n  {\n    return 0;\n  }\n" .
            $lj_values_ref->{'PRE_ACCESS'} .
            "  flag = 1;\n  if (flag)\n  {\n";
        $lj_values_ref->{'ACCESS'} =
            indent(2, $lj_values_ref->{'ACCESS'});
        $lj_values_ref->{'POST_ACCESS'} =
            "  }\n" . $lj_values_ref->{'POST_ACCESS'} .
            "  longjmp(env, 1);\n";
        push @$results_ref, $lj_values_ref;

        # function pointer = 6
        my $funcptr_values_ref = get_hash_copy($entry);
        $funcptr_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 6;
        $funcptr_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 1;    #
variable
        $funcptr_values_ref->{'BEFORE_MAIN'} =
            "int function1\(\)\n{\n" .
            "  return INDEX;\n}\n\n" .
            $funcptr_values_ref->{'BEFORE_MAIN'};
        $funcptr_values_ref->{'OTHER_DECL'} =
            "  int i;\n  int (*fptr)();\n" .
            $funcptr_values_ref->{'OTHER_DECL'};
        $funcptr_values_ref->{'PRE_ACCESS'} =
            "  fptr = function1;\n  i = fptr\(\);\n" .
            $funcptr_values_ref->{'PRE_ACCESS'};
        $funcptr_values_ref->{'ACCESS'} =
            "COMMENT\n  buf[i] = WRITE_VALUE;\n";
        push @$results_ref, $funcptr_values_ref;


        # recursion = 7
        my $recur_values_ref = get_hash_copy($entry);
        $recur_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 7;
        $recur_values_ref->{'BEFORE_MAIN'} =
            "int function1\(int counter\)\n{\n" .
            "  if \(counter > 0\)\n  {\n" .
            "    return function1\(counter - 1\);\n  }\n" .
            "  else\n  {\n    return INDEX;\n  }\n}\n\n" .
            $recur_values_ref->{'BEFORE_MAIN'};
        $recur_values_ref->{'OTHER_DECL'} =
            "  int i;\n  int (*fptr)(int);\n" .
            $recur_values_ref->{'OTHER_DECL'};
        $recur_values_ref->{'PRE_ACCESS'} =
            "  fptr = function1;\n  i = fptr\(3\);\n" .
            $recur_values_ref->{'PRE_ACCESS'};
        $recur_values_ref->{'ACCESS'} =
            "COMMENT\n  buf[i] = WRITE_VALUE;\n";
        push @$results_ref, $recur_values_ref;
    }

    return $results_ref;
}

##----------------------------------------------------------------
## do_signedness : produces all the test case variants for the
##  "signedness" attribute.
##----------------------------------------------------------------
sub do_signedness
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # yes = 1
        # variation 1: directly use a negative constant for an unsigned
length
        my $sign1_values_ref = get_hash_copy($entry);
        $sign1_values_ref->{'TAX'}->[$SIGNEDNESS_DIGIT] = 1;
        $sign1_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 2;  #
constant
        $sign1_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; # N/A -
no index
        $sign1_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A - no
index
        $sign1_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; # inter-
file/inter-proc
        $sign1_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1 alias of
addr
        $sign1_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] = 1;
#continuous
        $sign1_values_ref->{'INCL'} = "#include <string.h>\n" .
            $sign1_values_ref->{'INCL'};
        foreach my $index_value (keys %{$sign1_values_ref->{'MULTIS'}-
>{'INDEX'}})
        {
            $sign1_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} +=
                  1;
        }
        $sign1_values_ref->{'MULTIS'}->{'SIGNED_LEN'} =
            {$OK_OVERFLOW => $BUF_SIZE,
                                 $MIN_OVERFLOW => "-1",
                                 $MED_OVERFLOW => "-1",
                         $LARGE_OVERFLOW => "-1"};
        $sign1_values_ref->{'OTHER_DECL'} = "  char src[INDEX];\n" .
                                   $sign1_values_ref->{'OTHER_DECL'};
      $sign1_values_ref->{'PRE_ACCESS'} =
            "  memset(src, 'A', INDEX);\n" .
            "  src[INDEX - 1] = '\\0';\n" .
            $sign1_values_ref->{'PRE_ACCESS'};
        $sign1_values_ref->{'ACCESS'} =
            "COMMENT\n  memcpy(buf, src, SIGNED_LEN);\n\n";
        push @$results_ref, $sign1_values_ref;

        # yes = 1
        # variation 2: use a negative variable
        my $sign2_values_ref = get_hash_copy($entry);
        $sign2_values_ref->{'TAX'}->[$SIGNEDNESS_DIGIT] = 1;
        $sign2_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 3;  #
variable
        $sign2_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; # N/A -
no index
        $sign2_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A - no
index
```

```
        $sign2_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; # inter-
file/inter-proc
        $sign2_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1 alias of
addr
        $sign2_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] = 1;
#continuous
        $sign2_values_ref->{'INCL'} =
            "#include <string.h>\n" . $sign2_values_ref->{'INCL'};
        foreach my $index_value (keys %{$sign2_values_ref->{'MULTIS'}-
>{'INDEX'}})
        {
            $sign2_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} +=
                  1;
        }
        $sign2_values_ref->{'MULTIS'}->{'SIGNED_LEN'} =
            {$OK_OVERFLOW => $BUF_SIZE,
                                 $MIN_OVERFLOW => "-1",
                                 $MED_OVERFLOW => "-1",
                         $LARGE_OVERFLOW => "-1"};
        $sign2_values_ref->{'OTHER_DECL'} =
            "  int size;\n  char src[INDEX];\n" .
                               $sign2_values_ref->{'OTHER_DECL'};
        $sign2_values_ref->{'PRE_ACCESS'} =
            "  memset(src, 'A', INDEX);\n" .
            "  src[INDEX - 1] = '\\0';\n" .
            "  size = SIGNED_LEN;\n" .
            $sign2_values_ref->{'PRE_ACCESS'};
        $sign2_values_ref->{'ACCESS'} =
            "COMMENT\n  memcpy(buf, src, size);\n\n";
        push @$results_ref, $sign2_values_ref;

        # yes = 1
        # variation 3: use a negative variable, plus test
        my $sign3_values_ref = get_hash_copy($entry);
        $sign3_values_ref->{'TAX'}->[$SIGNEDNESS_DIGIT] = 1;
        $sign3_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 3;  #
variable
        $sign3_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; # N/A -
no index
        $sign3_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A - no
index
        $sign3_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; # inter-
file/inter-proc
        $sign3_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1 alias of
addr
        $sign3_values_ref->{'TAX'}->[$LOCALFLOW_DIGIT] = 1;   # if
        $sign3_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] = 1;
#continuous
        $sign3_values_ref->{'INCL'} =
            "#include <string.h>\n" . $sign3_values_ref->{'INCL'};
        foreach my $index_value (keys %{$sign3_values_ref->{'MULTIS'}-
>{'INDEX'}})
        {
            $sign3_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} +=
                  1;
        }
        $sign3_values_ref->{'MULTIS'}->{'SIGNED_LEN'} =
            {$OK_OVERFLOW => $BUF_SIZE,
                                 $MIN_OVERFLOW => "-1",
                                 $MED_OVERFLOW => "-1",
                         $LARGE_OVERFLOW => "-1"};
        $sign3_values_ref->{'OTHER_DECL'} =
            "  int copy_size;\n  int buf_size;\n" .
            "  char src[INDEX];\n" .
            $sign3_values_ref->{'OTHER_DECL'};
        $sign3_values_ref->{'PRE_ACCESS'} =
            $sign3_values_ref->{'PRE_ACCESS'} .
            "  memset(src, 'A', INDEX);\n" .
            "  src[INDEX - 1] = '\\0';\n" .
            "  copy_size = SIGNED_LEN;\n" .
            "  buf_size = sizeof buf;\n" .
            "  if (copy_size <= buf_size)\n  {\n";
        $sign3_values_ref->{'ACCESS'} =
            indent(2, "COMMENT\n  memcpy(buf, src, copy_size);\n\n");
        $sign3_values_ref->{'POST_ACCESS'} = "  }\n" .
            $sign3_values_ref->{'POST_ACCESS'};
        push @$results_ref, $sign3_values_ref;

        # yes = 1
        # variation 4: use a negative variable, plus test with a cast
        my $sign4_values_ref = get_hash_copy($entry);
        $sign4_values_ref->{'TAX'}->[$SIGNEDNESS_DIGIT] = 1;
        $sign4_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 3;  #
variable
        $sign4_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; # N/A -
no index
        $sign4_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A - no
index
        $sign4_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; # inter-
file/inter-proc
        $sign4_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1 alias of
addr
        $sign4_values_ref->{'TAX'}->[$LOCALFLOW_DIGIT] = 1;   # if
        $sign4_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] = 1;
#continuous
        $sign4_values_ref->{'INCL'} =
            "#include <string.h>\n" . $sign4_values_ref->{'INCL'};
        foreach my $index_value (keys %{$sign4_values_ref->{'MULTIS'}-
>{'INDEX'}})
        {
            $sign4_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value} +=
                  1;
        }
        $sign4_values_ref->{'MULTIS'}->{'SIGNED_LEN'} =
            {$OK_OVERFLOW => $BUF_SIZE,
                                 $MIN_OVERFLOW => "-1",
                                 $MED_OVERFLOW => "-1",
                         $LARGE_OVERFLOW => "-1"};
        $sign4_values_ref->{'OTHER_DECL'} =
            "  int copy_size;\n  char src[INDEX];\n" .
            $sign4_values_ref->{'OTHER_DECL'};
        $sign4_values_ref->{'PRE_ACCESS'} =
            $sign4_values_ref->{'PRE_ACCESS'} .
            "  memset(src, 'A', INDEX);\n" .
            "  src[INDEX - 1] = '\\0';\n" .
            "  copy_size = SIGNED_LEN;\n" .
            "  if (copy_size <= (int)(sizeof buf))\n  {\n";
        $sign4_values_ref->{'ACCESS'} =
            indent(2, "COMMENT\n  memcpy(buf, src, copy_size);\n\n");
        $sign4_values_ref->{'POST_ACCESS'} = "  }\n" .
            $sign4_values_ref->{'POST_ACCESS'};
        push @$results_ref, $sign4_values_ref;
    }

    return $results_ref;
}

##-------------------------------------------------------------------
## do_taint : produces all the test case variants for the
##  "taint" attribute.
##-------------------------------------------------------------------
sub do_taint
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # argc/argv = 1
        my $argcargv_values_ref = get_hash_copy($entry);
        $argcargv_values_ref->{'TAX'}->[$TAINT_DIGIT] = 1;
        # depends on command line argument
        $argcargv_values_ref->{'TAX'}->[$RUNTIMEENVDEP_DIGIT] = 1;
        # if statement before overflow
        $argcargv_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 1;
        # index complexity = return value of a function
        $argcargv_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 4;
        $argcargv_values_ref->{'INCL'} =
            "#include <stdlib.h>\n" . $argcargv_values_ref->{'INCL'};

        # setup the argv index to depend on the size of the overflow
        $argcargv_values_ref->{'MULTIS'}->{'WHICH_ARGV'} =
                                     {$OK_OVERFLOW => 1,
                                      $MIN_OVERFLOW => 2,
                                      $MED_OVERFLOW => 3,
                                      $LARGE_OVERFLOW => 4};
        $argcargv_values_ref->{'PRE_ACCESS'} =
            "  if ((argc < 5) || (atoi(argv[WHICH_ARGV]) > INDEX))\n" .
            "  {\n    return 0;\n  }\n" .
            $argcargv_values_ref->{'PRE_ACCESS'};
        $argcargv_values_ref->{'ACCESS'} =~
            s/INDEX/atoi(argv\[WHICH_ARGV\])/;
        push @$results_ref, $argcargv_values_ref;

        # env var = 2
        my $envvar_values_ref = get_hash_copy($entry);
        $envvar_values_ref->{'TAX'}->[$TAINT_DIGIT] = 2;
        # depends on length of PATH
        $envvar_values_ref->{'TAX'}->[$RUNTIMEENVDEP_DIGIT] = 1;
        # if statement before overflow
        $envvar_values_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] = 1;
        # index complexity = variable
        $envvar_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 1;
        $envvar_values_ref->{'INCL'} =
            "#include <string.h>\n#include <stdlib.h>\n" .
            $envvar_values_ref->{'INCL'};
        $envvar_values_ref->{'OTHER_DECL'} =
            "  int i;\n  char * envvar;\n" .
            $envvar_values_ref->{'OTHER_DECL'};
        $envvar_values_ref->{'MULTIS'}->{'ENVVAR'} =
            {$OK_OVERFLOW => "STRINGLEN_OK",
                                 $MIN_OVERFLOW => "STRINGLEN_MIN",
                                 $MED_OVERFLOW => "STRINGLEN_MED",
                         $LARGE_OVERFLOW => "STRINGLEN_LARGE"};
        $envvar_values_ref->{'PRE_ACCESS'} =
            "  envvar = getenv(\"ENVVAR\");\n" .
            "  if (envvar != NULL)\n  {\n" .
            "    i = strlen(envvar);\n  }\n" .
            "  else\n  {\n    i = 0;\n  }\n\n" .
            "  if (i > INDEX)\n" .
            "  {\n    return 0;\n  }\n" .
            $envvar_values_ref->{'PRE_ACCESS'};
        $envvar_values_ref->{'ACCESS'} =~ s/INDEX/i/;
        push @$results_ref, $envvar_values_ref;

        # reading from file or stdin = 3
        my $fileread_values_ref = get_hash_copy($entry);
        $fileread_values_ref->{'TAX'}->[$TAINT_DIGIT] = 3;
        # depends on contents of file
        $fileread_values_ref->{'TAX'}->[$RUNTIMEENVDEP_DIGIT] = 1;
        # length/limit complexity = constant
        $fileread_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 2;
        # library functions are inter-file/inter-proc, 1 alias of addr,
no index
        $fileread_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; #inter-
file/inter-proc
        $fileread_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1 alias
of addr
        $fileread_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; # N/A
- no index
        $fileread_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A -
no index
        $fileread_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] = 1;
#continuous
        $fileread_values_ref->{'INCL'} =
            "#include <assert.h>\n#include <stdio.h>\n" .
            $fileread_values_ref->{'INCL'};
        $fileread_values_ref->{'OTHER_DECL'} = "  FILE * f;\n" .
            $fileread_values_ref->{'OTHER_DECL'};
        $fileread_values_ref->{'PRE_ACCESS'} =
            "  f = fopen(\"TestInputFile1\", \"r\");\n" .
            "  assert(f != NULL);\n" .
            $fileread_values_ref->{'PRE_ACCESS'};
        $fileread_values_ref->{'ACCESS'} =
            "COMMENT\n  fgets(buf, INDEX, f);\n\n";
        $fileread_values_ref->{'POST_ACCESS'} =
            "  fclose(f);\n" .
            $fileread_values_ref->{'POST_ACCESS'};
        foreach my $index_value (keys %{$fileread_values_ref-
>{'MULTIS'}->{'INDEX'}})
        {
            $fileread_values_ref->{'MULTIS'}->{'INDEX'}->{$index_value}
                  += 1;
        }
        # we need to generate the input file as well
        $fileread_values_ref->{'EXTRA_FILES'}->{'TestInputFile1'} =
            'A' x 5000;
        push @$results_ref, $fileread_values_ref;

        # FUTURE: fill in a sockets example here
        # reading from a socket = 4

        # process environment = 5
        my $processenv_values_ref = get_hash_copy($entry);
        $processenv_values_ref->{'TAX'}->[$TAINT_DIGIT] = 5;
        # depends on current working directory
        $processenv_values_ref->{'TAX'}->[$RUNTIMEENVDEP_DIGIT] = 1;
        # length/limit complexity = constant
        $processenv_values_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] = 2;
        # library functions are inter-file/inter-proc, and 1 alias of
addr
        $processenv_values_ref->{'TAX'}->[$SCOPE_DIGIT] = 3; #inter-
file/inter-proc
        $processenv_values_ref->{'TAX'}->[$ALIASADDR_DIGIT] = 1; #1
alias of addr
        $processenv_values_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] = 6; #
N/A - no index
        $processenv_values_ref->{'TAX'}->[$ALIASINDEX_DIGIT] = 3; # N/A
- no index
        $processenv_values_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT] =
             1; #continuous
        $processenv_values_ref->{'INCL'} = "#include <unistd.h>\n" .
            $processenv_values_ref->{'INCL'};
        $processenv_values_ref->{'ACCESS'} =
            "COMMENT\n  getcwd(buf, INDEX);\n\n";
        foreach my $index1_value (keys %{$processenv_values_ref-
>{'MULTIS'}->{'INDEX'}})
        {
            $processenv_values_ref->{'MULTIS'}->{'INDEX'}-
>{$index1_value} += 1;
        }
        push @$results_ref, $processenv_values_ref;
    }

    return $results_ref;
```

```
}

##--------------------------------------------------------------
## do_whichbound : produces all the test case variants for the "which
bound"
##  attribute.
##--------------------------------------------------------------
# COMBO NOTE:  lower bound will also affect the first index of a 2D
array.
# Needs to be updated for underflow combos to work.
sub do_whichbound
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # lower = 1
        my $lower_values_ref = get_hash_copy($entry);
        $lower_values_ref->{'TAX'}->[$WHICHBOUND_DIGIT] = 1;
        $lower_values_ref->{'MULTIS'}->{'INDEX'} =
                                     {$OK_OVERFLOW => 0,
                                      $MIN_OVERFLOW => 0 - $MIN_SIZE,
                                      $MED_OVERFLOW => 0 - $MED_SIZE,
                                   $LARGE_OVERFLOW => 0 - $LARGE_SIZE};
        push @$results_ref, $lower_values_ref;
    }

    return $results_ref;
}

##--------------------------------------------------------------
## do_writeread : produces all the test case variants for the
"write/read"
##  attribute.
##--------------------------------------------------------------
sub do_writeread
{
    my $start_with_array_ref = shift;
    my $results_ref = [];

    foreach my $entry (@{$start_with_array_ref})
    {
        # write = 0
        push @$results_ref, get_hash_copy($entry);

        # read = 1
        my $read_values_ref = get_hash_copy($entry);
        $read_values_ref->{'TAX'}->[$WRITEREAD_DIGIT] = 1;
        $read_values_ref->{'OTHER_DECL'} =
            "  TYPE read_value;\n" . $read_values_ref->{'OTHER_DECL'};
        $read_values_ref->{'ACCESS'} =
            "COMMENT\n  read_value = buf[INDEX];\n";
        push @$results_ref, $read_values_ref;
    }
    return $results_ref;
}

##----------------------------------------------------------------
## expand_tax_class : Write out expanded taxonomy classification
##  to the given open file using values in the given array ref.
##----------------------------------------------------------------
sub expand_tax_class
{
    my ($fh, $tax_values) = @_;

    print $fh "/*\n";

    for (my $i=0; $i < scalar @$tax_values; ++$i)
    {
        printf $fh " *  %-25s\t", $TaxonomyInfo[$i]->[NAME_INDEX];
        printf $fh "%2u\t", $tax_values->[$i];
        print $fh "$TaxonomyInfo[$i]->[VALUES_INDEX]->{$tax_values-
>[$i]}\n";
    }

    print $fh " */\n\n";
}

##----------------------------------------------------------------
## get_array_copy : given a reference to an array, returns a reference
to a new
##   copy of the array itself (deep copy of all entries)
##----------------------------------------------------------------
sub get_array_copy
{
    my $orig_array_ref = shift;
    my @new_array;

    foreach my $entry (@$orig_array_ref)
    {
        if (ref($entry) eq "HASH")
        {
            push @new_array, get_hash_copy($entry);
        }
        elsif (ref($entry) eq "ARRAY")
        {
            push @new_array, get_array_copy($entry);
        }
        else
        {
            push @new_array, $entry;
        }
    }
    return \@new_array;
}

##----------------------------------------------------------------
## get_default_values : returns a reference to a hash of default values
##  for various pieces of test case code
##--------------------------------------------------------------
sub get_default_values
{
    return {'INCL' => "\n", 'BUF_DECL' => $BUF_DECL,
            'OTHER_DECL' => "",
        'ACCESS' => $BUF_ACCESS,
        'SINGLES' => {'TYPE' => "char", 'WRITE_VALUE' => "\'A\'"},
        'MULTIS' => { 'INDEX' => {$OK_OVERFLOW => $BUF_SIZE + $OK_SIZE
                                                                - 1,
                             $MIN_OVERFLOW => $BUF_SIZE + $MIN_SIZE
                                                                - 1,
                             $MED_OVERFLOW => $BUF_SIZE + $MED_SIZE
                                                                - 1,
                             $LARGE_OVERFLOW => $BUF_SIZE +
                                                  $LARGE_SIZE - 1}
                   },
        'TAX' => get_init_tax_class_values(),
        'PRE_ACCESS' => "\n",
        'POST_ACCESS' => "\n",
        'FILE_GLOBAL' => "",
        'BEFORE_MAIN' => "",
        'EXTRA_FILES' => {}};
}

##--------------------------------------------------------------
## get_fseq_num : Figures out and returns the next unused
##   file sequence number. Returns 0 if they're all used up.
##--------------------------------------------------------------
sub get_fseq_num
{
    my $num = 1;
    my $fname;
    my $found_it = 0;

    # loop until we've exhausted all seq nums or we found an unused one
    while ($num <= 99999 && !$found_it)
    {   # get a file name for this seq num
        $fname = get_ok_filename($num);
        if (-e $fname)
        {   # if it exists already, increment to next seq num
            $num++;
        }
        else
        {   # otherwise, set the found flag
            $found_it = 1;
        }
    }

    # return the next unused seq num if we found one, else return 0
    if ($found_it)
    {
        return $num;
    }
    else
    {
        return 0;
    }
};

##--------------------------------------------------------------
## get_filenames : given a file sequence number, generates a
##  set of file names
##--------------------------------------------------------------
sub get_filenames
{   # get the seq num that was passed in
    my $num = shift;

    my $fname_ok = get_ok_filename($num);
    my $fname_min = get_min_filename($num);
    my $fname_med = get_med_filename($num);
    my $fname_large = get_large_filename($num);

    return ($fname_ok, $fname_min, $fname_med, $fname_large);
};

##--------------------------------------------------------------
## get_hash_copy : given a reference to a hash, returns a reference to
a new
##   copy of the hash itself (deep copy of all entries)
##--------------------------------------------------------------
sub get_hash_copy
{
    my $orig_hash_ref = shift;
    my %new_hash;

    foreach my $key (sort keys %{$orig_hash_ref})
    {
        if (ref($orig_hash_ref->{$key}) eq "HASH")
        {
            $new_hash{$key} = get_hash_copy($orig_hash_ref->{$key});
        }
        elsif (ref($orig_hash_ref->{$key}) eq "ARRAY")
        {
            $new_hash{$key} = get_array_copy($orig_hash_ref->{$key});
        }
        else
        {
            $new_hash{$key} = $orig_hash_ref->{$key};
        }
    }
    return \%new_hash;
}

##--------------------------------------------------------------
## get_init_tax_class_values : returns a reference to an array of
##  initialized taxonomy classification values
##--------------------------------------------------------------
sub get_init_tax_class_values
{
    return [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0];
};

##-----------------------------------------------------------------
## get_ok_filename : given a file sequence number, generates a file
name
##  for an ok (no overflow) file
##-----------------------------------------------------------------
sub get_ok_filename
{   # get the seq num that was passed in
    my $num = shift;

    return sprintf("basic-%05u-ok.c", $num);
};

##-----------------------------------------------------------------
## get_min_filename : given a file sequence number, generates a file
name
##  for a minimum overflow file
##-----------------------------------------------------------------
sub get_min_filename
{   # get the seq num that was passed in
    my $num = shift;

    return sprintf("basic-%05u-min.c", $num);
};

##-----------------------------------------------------------------
## get_med_filename : given a file sequence number, generates a file
name
##  for a medium overflow file
##-----------------------------------------------------------------
sub get_med_filename
{   # get the seq num that was passed in
    my $num = shift;

    return sprintf("basic-%05u-med.c", $num);
};

##-----------------------------------------------------------------
## get_large_filename : given a file sequence number, generates a file
name
##  for a large overflow file
##-----------------------------------------------------------------
sub get_large_filename
{   # get the seq num that was passed in
    my $num = shift;

    return sprintf("basic-%05u-large.c", $num);
};

##-----------------------------------------------------------------
## handle_attributes : given a reference to an array of attributes to
vary, produces
```

```
##  all the test case variants for those attributes.  Returns the
variants
##  as an array of hashes.  Array element 0, for instance, would hold
##  a hash that contains pieces of a test case program appropriate for
##  when the given attribute's value is 0.
##-----------------------------------------------------------------
sub handle_attributes
{   # get the attribute numbers that were passed in
    my $attributes = shift;

    # check that each attribute is valid (and subtract 1 from each so
    #   that they're zero-relative)
    for (my $i = 0; $i < scalar @$attributes; ++$i)
    {
        $attributes->[$i]--;
        if (!defined($AttributeFunctions{$attributes->[$i]}))
        {
            die "Error: unknown attribute\n";
        }
    }

    # start with default values and vary from there
    my $start_with_values_ref = [get_default_values()];
    foreach my $attr (@$attributes)
    {
        $start_with_values_ref =
            $AttributeFunctions{$attr}->($start_with_values_ref);
    }

    return $start_with_values_ref;
}

##-----------------------------------------------------------------
## indent : indents every line in the given string by the given number
of
##      spaces and returns the result in new string
##-----------------------------------------------------------------
sub indent
{
    my ($num_spaces, $oldstr) = @_;
    my $append_str = ' ' x $num_spaces;
    my $newstr =
      join("\n", map {join("",$append_str,$_)} split("\n", $oldstr));
    if (substr($oldstr, -1) eq "\n")
    {
        $newstr = join("",$newstr,"\n");
    }
    return $newstr;
}

##-----------------------------------------------------------------
## make_files : Generates the test case files for variations on the
##  given attribute, starting with the given file sequence number.
##  Returns the next ununused file sequence number.
##-----------------------------------------------------------------
```

```
sub make_files
{   # get the parameters that were passed in
    my ($attributes, $fseq_num) = @_;

    my $fh;
    my $this_tax_class;
    my $tax_class_value;
    my $this_buf_access;

    my $values_array_ref = handle_attributes($attributes);

    foreach my $values_hash_ref (@$values_array_ref)
    {
        # generate a set of output file names
        my ($filename_ok, $filename_min, $filename_med,
$filename_large) =
            get_filenames($fseq_num);

        # generate the set of files
        my %file_info = ($filename_ok => $OK_OVERFLOW,
                        $filename_min => $MIN_OVERFLOW,
                        $filename_med => $MED_OVERFLOW,
                        $filename_large => $LARGE_OVERFLOW);

        foreach my $file (keys %file_info)
        {
            print "Processing file: $file...\n";

            # set value of Magnitude to match the size of this overflow
            $values_hash_ref->{'TAX'}->[$MAGNITUDE_DIGIT] =
                  $file_info{$file};

            # make the taxonomy classification, replacing VALUE with
            # the real value
            $tax_class_value =
                $values_hash_ref->{'TAX'}->[$WRITEREAD_DIGIT] .
                $values_hash_ref->{'TAX'}->[$WHICHBOUND_DIGIT] .
                $values_hash_ref->{'TAX'}->[$DATATYPE_DIGIT] .
                $values_hash_ref->{'TAX'}->[$MEMLOC_DIGIT] .
                $values_hash_ref->{'TAX'}->[$SCOPE_DIGIT] .
                $values_hash_ref->{'TAX'}->[$CONTAINER_DIGIT] .
                $values_hash_ref->{'TAX'}->[$POINTER_DIGIT] .
                $values_hash_ref->{'TAX'}->[$INDEXCOMPLEX_DIGIT] .
                $values_hash_ref->{'TAX'}->[$ADDRCOMPLEX_DIGIT] .
                $values_hash_ref->{'TAX'}->[$LENCOMPLEX_DIGIT] .
                $values_hash_ref->{'TAX'}->[$ALIASADDR_DIGIT] .
                $values_hash_ref->{'TAX'}->[$ALIASINDEX_DIGIT] .
                $values_hash_ref->{'TAX'}->[$LOCALFLOW_DIGIT] .
                $values_hash_ref->{'TAX'}->[$SECONDARYFLOW_DIGIT] .
                $values_hash_ref->{'TAX'}->[$LOOPSTRUCTURE_DIGIT] .
                $values_hash_ref->{'TAX'}->[$LOOPCOMPLEX_DIGIT] .
                $values_hash_ref->{'TAX'}->[$ASYNCHRONY_DIGIT] .
                $values_hash_ref->{'TAX'}->[$TAINT_DIGIT] .
                $values_hash_ref->{'TAX'}->[$RUNTIMEENVDEP_DIGIT] .
                $values_hash_ref->{'TAX'}->[$MAGNITUDE_DIGIT] .
                $values_hash_ref->{'TAX'}->[$CONTINUOUSDISCRETE_DIGIT]
.
                $values_hash_ref->{'TAX'}->[$SIGNEDNESS_DIGIT];
            $this_tax_class = TAXONOMY_CLASSIFICATION;
            $this_tax_class =~ s/VALUE/$tax_class_value/;

            # perform all necessary substitutions for INDEX,
            #  WRITE_VALUE, TYPE, and COMMENT
            my %copy_values = %$values_hash_ref;
            substitute_values(\%copy_values, $file_info{$file});

            # open the file for writing
            open($fh, ">", $file);

            # write out the contents
            print $fh $this_tax_class;
            # write out the expanded taxonomy classification
            expand_tax_class($fh, $copy_values{'TAX'});
            print $fh FILE_HEADER;
            print $fh $copy_values{'INCL'};
            print $fh $copy_values{'FILE_GLOBAL'};
            print $fh $copy_values{'BEFORE_MAIN'};
            print $fh MAIN_OPEN;
            print $fh $copy_values{'OTHER_DECL'};
            print $fh $copy_values{'BUF_DECL'};
            print $fh $copy_values{'PRE_ACCESS'};
            print $fh $copy_values{'ACCESS'};
            print $fh $copy_values{'POST_ACCESS'};
            print $fh MAIN_CLOSE;

            # close the file
            close($fh);
        }

        # generate extra files if needed
        foreach my $extra_file (keys %{$values_hash_ref-
>{'EXTRA_FILES'}})
        {
            open ($fh, ">", $extra_file);
            print $fh $values_hash_ref->{'EXTRA_FILES'}->{$extra_file};
            close($fh);
        }

        # increment the file sequence number
        ++$fseq_num;
    }

    return $fseq_num;
};

##----------------------------------------------------------------
## substitute_values: given a hash of program strings and which size
overflow
##  we're working with, substitute for all of our placeholders such as
##  INDEX, WRITE_VALUE, TYPE, and COMMENT
```

```
##----------------------------------------------------------------
sub substitute_values
{
    my ($hash_ref, $which_overflow) = @_;

    foreach my $item ('FILE_GLOBAL', 'BEFORE_MAIN', 'OTHER_DECL',
'BUF_DECL',
        'PRE_ACCESS', 'ACCESS', 'POST_ACCESS')
    {
        $hash_ref->{$item} =~
            s/COMMENT/$OverflowInfo{$which_overflow}-
      >[COMMENT_INDEX]/g;

        # iterate through all of the single substitutions
        foreach my $single (keys %{$hash_ref->{'SINGLES'}})
        {
            $hash_ref->{$item} =~
                s/$single/$hash_ref->{'SINGLES'}->{$single}/g;
        }

        # iterate through all of the multi (size-related) substitutions
        foreach my $multi (keys %{$hash_ref->{'MULTIS'}})
        {
            $hash_ref->{$item} =~
                s/$multi/$hash_ref->{'MULTIS'}->{$multi}-
>{$which_overflow}/g;
        }
    }
};

##----------------------------------------------------------------
## usage : print out an optional error message, followed by usage
## information, and then die
##----------------------------------------------------------------
sub usage
{
    if (scalar(@_) > 0)
    {
        print shift;
    }

    die "\nUsage:  gen_basic_tests.pl [--combo] <attribute list> \n";
};
__END__

=cut

=head1 NAME

    # give your program name and a very short description
    gen_basic_tests.pl - Generates test cases for static analysis tool
test suite

=head1 SYNOPSIS
```

```
    # show examples of how to use your program
    ./gen_basic_tests.pl [--combo] <attribute list>
    ./gen_basic_tests.pl 3 4
    ./gen_basic_tests.pl --combo 3 4

=head1 DESCRIPTION

    # describe in more detail what your_program does
    Without the combo option, gen_basic_tests.pl produces test cases
for each
    attribute in the given list, varying one attribute at a time.  For
instance,
    if the command line specifies attributes 3 and 4, it produces one
set of
    test cases that starts with the baseline test case and varies the
Data Type
    attribute, and another independent set that starts with the
baseline test
    case and varies the Memory Location attribute.

=head1 OPTIONS

    # document your_program's options in sub-sections here
=head2 --combo

    # describe what option does
    If the combo option is specified, gen_basic_tests.pl produces a set
of test
    cases that varies the attributes in combination.  For example, if
attributes
    three and four are specified (in that order) in conjunction with
the combo
    option, it produces an intermediate set of test cases that start
with the
    baseline test case and vary the Data Type attribute; it produces
the final
    set by starting with each of the intermediate test cases and
varying the
    Memory Location attribute.  This process yields combinations of
    simultaneously varying Data Type and Memory Location attributes.

=head1 AUTHOR

    # provide your name and contact information
    Kendra Kratkiewicz, MIT Lincoln Laboratory

=cut
```

gen_table_each_value.pl

```
#!/usr/bin/perl -w

use strict;

# Copyright 2004 M.I.T.

##----------------------------------------------------------------
## File globals
##----------------------------------------------------------------
my %Results;    #holds statistics for each tool/each attribute/each
value

##----------------------------------------------------------------
## main program
##----------------------------------------------------------------

#----------------------------------------------------------------
# process arguments
#----------------------------------------------------------------
# check to see if we have the right number of arguments
if (@ARGV < 1)
{
    usage();
}

# check arguments for validity
my $file = $ARGV[0];
-e $file or die "Sorry.  $file does not exist.\n";

#----------------------------------------------------------------
# here's the beef
#----------------------------------------------------------------
# open the given input file
open(THISFILE, "<", $file) or die "Sorry.  Could not open $file.\n";

# process five blocks of results from the five tools
for (my $i=1; $i < 6; ++$i)
{
    process_tool_block();
}

close(THISFILE);

# now that we've processed all of the tool info, print out the
statistics
print "ATTRIBUTE\tVALUE\tDESCRIPTION\tARCHER TOTAL\tBOON TOTAL" .
       "\tPOLYSPACE TOTAL\tSPLINT TOTAL\tUNO TOTAL" .
       "\tARCHER RESULT\tBOON RESULT\tPOLYSPACE RESULT" .
       "\tSPLINT RESULT\tUNO RESULT\n";
for my $attribute (sort keys %Results)
{
    my $attrib_hash = $Results{$attribute};
    for my $value (sort keys %$attrib_hash)
    {
        my $value_hash = $attrib_hash->{$value};
        printf "$attribute\t$value\t$value_hash-
>{'desc'}\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
               $value_hash->{'archer'}->{'total'},
               $value_hash->{'boon'}->{'total'},
               $value_hash->{'polyspace'}->{'total'},
               $value_hash->{'splint'}->{'total'},
               $value_hash->{'uno'}->{'total'},
               $value_hash->{'archer'}->{'result'},
               $value_hash->{'boon'}->{'result'},
               $value_hash->{'polyspace'}->{'result'},
               $value_hash->{'splint'}->{'result'},
               $value_hash->{'uno'}->{'result'};
    }
}

exit(0);

##----------------------------------------------------------------
## subroutines  (alphabetized)
##----------------------------------------------------------------
##----------------------------------------------------------------
## process_addrcomplex : get the addrcomplex attribute/value results
##----------------------------------------------------------------
sub process_addrcomplex
{
    my $tool = shift;

    # 0 = constant
    save_values("addrcomplex", 0, "constant", $tool);

    # 1 = variable
    save_values("addrcomplex", 1, "variable", $tool);

    # 2 = linear exp
    save_values("addrcomplex", 2, "linear exp", $tool);

    # 3 = non-linear exp
    save_values("addrcomplex", 3, "non-linear exp", $tool);

    # 4 = func ret val
    save_values("addrcomplex", 4, "func ret val", $tool);

    # 5 = array contents
    save_values("addrcomplex", 5, "array contents", $tool);
};

##----------------------------------------------------------------
## process_aliasaddr : get the aliasaddr attribute/value results
##----------------------------------------------------------------
sub process_aliasaddr
{
    my $tool = shift;
    # 0 = none
    save_values("aliasaddr", 0, "none", $tool);

    # 1 = one
    save_values("aliasaddr", 1, "one", $tool);

    # 2 = two
    save_values("aliasaddr", 2, "two", $tool);
};

##----------------------------------------------------------------
## process_aliasindex : get the aliasindex attribute/value results
##----------------------------------------------------------------
sub process_aliasindex
{
    my $tool = shift;

    # 0 = none
    save_values("aliasindex", 0, "none", $tool);

    # 1 = one
    save_values("aliasindex", 1, "one", $tool);

    # 2 = two
    save_values("aliasindex", 2, "two", $tool);

    # 3 = N/A
    save_values("aliasindex", 3, "N/A", $tool);

};

##----------------------------------------------------------------
## process_asynchrony : get the asynchrony attribute/value results
##----------------------------------------------------------------
sub process_asynchrony
{
    my $tool = shift;

    # 0 = none
    save_values("asynchrony", 0, "none", $tool);

    # 1 = threads
    save_values("asynchrony", 1, "threads", $tool);

    # 2 = fork
    save_values("asynchrony", 2, "fork", $tool);

    # 3 = sig hand
    save_values("asynchrony", 3, "sig hand", $tool);
};

##----------------------------------------------------------------
## process_container : get the container attribute/value results
##----------------------------------------------------------------
```

```
sub process_container
{
    my $tool = shift;

    # 0 = none
    save_values("container", 0, "none", $tool);

    # 1 = array
    save_values("container", 1, "array", $tool);

    # 2 = struct
    save_values("container", 2, "struct", $tool);

    # 3 = union
    save_values("container", 3, "union", $tool);

    # 4 = array of structs
    save_values("container", 4, "array of structs", $tool);

    # 5 = array of unions
    save_values("container", 5, "array of unions", $tool);
};

##----------------------------------------------------------------
## process_continuousdiscrete : get the continuousdiscrete
attribute/value results
##----------------------------------------------------------------
sub process_continuousdiscrete
{
    my $tool = shift;

    # 0 = discrete
    save_values("continuousdiscrete", 0, "discrete", $tool);

    # 1 = continuous
    save_values("continuousdiscrete", 1, "continuous", $tool);
};

##----------------------------------------------------------------
## process_datatype : get the datatype attribute/value results
##----------------------------------------------------------------
sub process_datatype
{
    my $tool = shift;

    # 0 = char
    save_values("datatype", 0, "char", $tool);

    # 1 = int
    save_values("datatype", 1, "int", $tool);

    # 2 = float
    save_values("datatype", 2, "float", $tool);

    # 3 = wchar
    save_values("datatype", 3, "wchar", $tool);

    # 4 = pointer
    save_values("datatype", 4, "pointer", $tool);

    # 5 = unsigned int
    save_values("datatype", 5, "unsigned int", $tool);

    # 6 = unsigned char
    save_values("datatype", 6, "unsigned char", $tool);
};

##----------------------------------------------------------------
## process_indexcomplex : get the indexcomplex attribute/value results
##----------------------------------------------------------------
sub process_indexcomplex
{
    my $tool = shift;

    # 0 = constant
    save_values("indexcomplex", 0, "constant", $tool);

    # 1 = variable
    save_values("indexcomplex", 1, "variable", $tool);

    # 2 = linear exp
    save_values("indexcomplex", 2, "linear exp", $tool);

    # 3 = non-linear exp
    save_values("indexcomplex", 3, "non-linear exp", $tool);

    # 4 = func ret val
    save_values("indexcomplex", 4, "func ret val", $tool);

    # 5 = array contents
    save_values("indexcomplex", 5, "array contents", $tool);

    # 6 = N/A
    save_values("indexcomplex", 6, "N/A", $tool);
};

##----------------------------------------------------------------
## process_lencomplex : get the lencomplex attribute/value results
##----------------------------------------------------------------
sub process_lencomplex
{
    my $tool = shift;

    # 0 = N/A
    save_values("lencomplex", 0, "N/A", $tool);

    # 1 = none
    save_values("lencomplex", 1, "none", $tool);

    # 2 = constant
    save_values("lencomplex", 2, "constant", $tool);

    # 3 = variable
    save_values("lencomplex", 3, "variable", $tool);

    # 4 = linear exp
    save_values("lencomplex", 4, "linear exp", $tool);

    # 5 = non-linear exp
    save_values("lencomplex", 5, "non-linear exp", $tool);

    # 6 = func ret val
    save_values("lencomplex", 6, "func ret val", $tool);

    # 7 = array contents
    save_values("lencomplex", 7, "array contents", $tool);
};

##----------------------------------------------------------------
## process_localflow : get the localflow attribute/value results
##----------------------------------------------------------------
sub process_localflow
{
    my $tool = shift;

    # 0 = none
    save_values("localflow", 0, "none", $tool);

    # 1 = if
    save_values("localflow", 1, "if", $tool);

    # 2 = switch
    save_values("localflow", 2, "switch", $tool);

    # 3 = cond
    save_values("localflow", 3, "cond", $tool);

    # 4 = goto
    save_values("localflow", 4, "goto", $tool);

    # 5 = longjmp
    save_values("localflow", 5, "longjmp", $tool);

    # 6 = func ptr
    save_values("localflow", 6, "func ptr", $tool);

    # 7 = recursion
    save_values("localflow", 7, "recursion", $tool);
};

##----------------------------------------------------------------
## process_loopcomplex : get the loopcomplex attribute/value results
##----------------------------------------------------------------
sub process_loopcomplex
{
    my $tool = shift;

    # 0 = N/A
    save_values("loopcomplex", 0, "N/A", $tool);

    # 1 = none
    save_values("loopcomplex", 1, "none", $tool);

    # 2 = one
    save_values("loopcomplex", 2, "one", $tool);

    # 3 = two
    save_values("loopcomplex", 3, "two", $tool);

    # 4 = N/A
    save_values("loopcomplex", 4, "N/A", $tool);
};

##----------------------------------------------------------------
## process_loopstructure : get the loopstructure attribute/value
results
##----------------------------------------------------------------
sub process_loopstructure
{
    my $tool = shift;

    # 0 = none
    save_values("loopstructure", 0, "none", $tool);

    # 1 = standard for
    save_values("loopstructure", 1, "standard for", $tool);

    # 2 = standard do-while
    save_values("loopstructure", 2, "standard do-while", $tool);

    # 3 = standard while
    save_values("loopstructure", 3, "standard while", $tool);

    # 4 = non-standard for
    save_values("loopstructure", 4, "non-standard for", $tool);

    # 5 = non-standard do-while
    save_values("loopstructure", 5, "non-standard do-while", $tool);

    # 6 = non-standard while
    save_values("loopstructure", 6, "non-standard while", $tool);
};

##----------------------------------------------------------------
## process_memloc : get the memloc attribute/value results
##----------------------------------------------------------------
sub process_memloc
{
    my $tool = shift;
    # 0 = stack
    save_values("memloc", 0, "stack", $tool);

    # 1 = heap
    save_values("memloc", 1, "heap", $tool);

    # 2 = data
    save_values("memloc", 2, "data", $tool);

    # 3 = bss
    save_values("memloc", 3, "bss", $tool);

    # 4 = shared
    save_values("memloc", 4, "shared", $tool);
};

##----------------------------------------------------------------
## process_pointer : get the pointer attribute/value results
##----------------------------------------------------------------
sub process_pointer
{
    my $tool = shift;

    # 0 = no
    save_values("pointer", 0, "no", $tool);

    # 1 = yes
    save_values("pointer", 1, "yes", $tool);
};

##----------------------------------------------------------------
## process_runtimeenvdep : get the runtimeenvdep attribute/value
results
##----------------------------------------------------------------
sub process_runtimeenvdep
{
    my $tool = shift;

    # 0 = no
    save_values("runtimeenvdep", 0, "no", $tool);

    # 1 = yes
    save_values("runtimeenvdep", 1, "yes", $tool);
};

##----------------------------------------------------------------
## process_scope : get the scope attribute/value results
##----------------------------------------------------------------
sub process_scope
{
    my $tool = shift;

    # 0 = same
    save_values("scope", 0, "same", $tool);
    # 1 = inter-proc
    save_values("scope", 1, "inter-proc", $tool);

    # 2 = global
    save_values("scope", 2, "global", $tool);

    # 3 = inter-file/inter-proc
    save_values("scope", 3, "inter-file/inter-proc", $tool);
};

##----------------------------------------------------------------
## process_secondaryflow : get the secondaryflow attribute/value
results
##----------------------------------------------------------------
sub process_secondaryflow
{
    my $tool = shift;

    # 0 = none
    save_values("secondaryflow", 0, "none", $tool);

    # 1 = if
    save_values("secondaryflow", 1, "if", $tool);

    # 2 = switch
    save_values("secondaryflow", 2, "switch", $tool);

    # 3 = cond
    save_values("secondaryflow", 3, "cond", $tool);

    # 4 = goto
    save_values("secondaryflow", 4, "goto", $tool);

    # 5 = longjmp
    save_values("secondaryflow", 5, "longjmp", $tool);

    # 6 = func ptr
    save_values("secondaryflow", 6, "func ptr", $tool);

    # 7 = recursion
    save_values("secondaryflow", 7, "recursion", $tool);
};

##----------------------------------------------------------------
## process_signedness : get the signedness attribute/value results
##----------------------------------------------------------------
sub process_signedness
{
    my $tool = shift;

    # 0 = no
    save_values("signedness", 0, "no", $tool);

    # 1 = yes
    save_values("signedness", 1, "yes", $tool);
```

```
};

##------------------------------------------------------------------
## process_taint : get the taint attribute/value results
##------------------------------------------------------------------
sub process_taint
{
    my $tool = shift;

    # 0 = none
    save_values("taint", 0, "none", $tool);

    # 1 = argc/argv
    save_values("taint", 1, "argc/argv", $tool);

    # 2 = env var
    save_values("taint", 2, "env var", $tool);

    # 3 = file read
    save_values("taint", 3, "file read", $tool);

    # 5 = process env
    save_values("taint", 5, "process env", $tool);
};

##------------------------------------------------------------------
## process_tool_block : process a block of data from one tool
##------------------------------------------------------------------
sub process_tool_block
{
    # the first line is the name of the tool
    my $tool = <THISFILE>;
    chomp $tool;
    trim($tool);
    #print "Found tool $tool\n";

    # get the attribute data
    process_writeread($tool);
    process_whichbound($tool);
    process_datatype($tool);
    process_memloc($tool);
    process_scope($tool);
    process_container($tool);
    process_pointer($tool);
    process_indexcomplex($tool);
    process_addrcomplex($tool);
    process_lencomplex($tool);
    process_aliasaddr($tool);
    process_aliasindex($tool);
    process_localflow($tool);
    process_secondaryflow($tool);
    process_loopstructure($tool);
    process_loopcomplex($tool);
    process_asynchrony($tool);
    process_taint($tool);
    process_runtimeenvdep($tool);
    process_continuousdiscrete($tool);
    process_signedness($tool);
};

##----------------------------------------------------------------
## process_whichbound : get the whichbound attribute/value results
##----------------------------------------------------------------
sub process_whichbound
{
    my $tool = shift;

    # 0 = upper
    save_values("whichbound", 0, "upper", $tool);

    # 1 = lower
    save_values("whichbound", 1, "lower", $tool);
};

##----------------------------------------------------------------
## process_writeread : get the writeread attribute/value results
##----------------------------------------------------------------
sub process_writeread
{
    my $tool = shift;

    # 0 = write
    save_values("writeread", 0, "write", $tool);

    # 1 = read
    save_values("writeread", 1, "read", $tool);
};

##----------------------------------------------------------------
## save_values : get and save values under given
attribute/value/desc/tool
##----------------------------------------------------------------
sub save_values
{
    my ($attrib, $value, $desc, $tool) = @_;

    # skip "count" line and grab total number from next line
    <THISFILE>;
    my $total = <THISFILE>;
    chomp $total;
    trim($total);

    # skip "count" line and grab this tool's performance number from
    # next line
    <THISFILE>;
    my $result = <THISFILE>;
    chomp $result;
    trim($result);

    # save all this info in our Results hash
    $Results{$attrib}{$value}{'desc'} = $desc;
    $Results{$attrib}{$value}{$tool}{'total'} = $total;
    $Results{$attrib}{$value}{$tool}{'result'} = $result;
};

##----------------------------------------------------------------
## trim : trim leading and trailing whitespace off the given
## string - modifies in place
##----------------------------------------------------------------
sub trim
{
    $_[0] =~ s/^\s+//;
    $_[0] =~ s/\s+$//;
};

##----------------------------------------------------------------
## usage : print out an optional error message, followed by usage
## information, and then die
##----------------------------------------------------------------
sub usage
{
    if (scalar(@_) > 0)
    {
        print shift;
    }

    die "\nUsage:  gen_table_each_value.pl <input file>\n";
};
__END__

=cut

=head1 NAME

    # give your program name and a very short description
    gen_table_each_value.pl - Processes the input file, tabulates how
each
        tool performed on each value of each attribute, and prints
report.

=head1 SYNOPSIS

    # show examples of how to use your program
    ./gen_table_each_value.pl raw_detection_results.txt >
detection_table.txt

=head1 DESCRIPTION

    # describe in more detail what your_program does

=head1 OPTIONS

    # document your_program's options in sub-sections here
=head2 --option1
```

```
    # describe what option does

=head1 AUTHOR

    # provide your name and contact information
    Kendra Kratkiewicz, MIT Lincoln Laboratory

=cut
```

gen_testcase_db_info.pl

```
#!/usr/bin/perl -w

use strict;

# Copyright 2004 M.I.T.

##----------------------------------------------------------------
## File globals
##----------------------------------------------------------------
# There will be one entry for each test case (minus the suffix
# ok/min/med/large), and the value will be the taxonomy
# classification string.
my %TestCases;

##----------------------------------------------------------------
## main program
##----------------------------------------------------------------

#----------------------------------------------------------------
# process arguments
#----------------------------------------------------------------
# check to see if we have the right number of arguments
if (@ARGV < 1)
{
    usage();
}

# check arguments for validity
my $dir = $ARGV[0];
-e $dir or die "Sorry.  Directory $dir does not exist.\n";
-d $dir or die "Sorry.  $dir is not a directory.\n";

#----------------------------------------------------------------
# here's the beef
#----------------------------------------------------------------
opendir(THEDIR, $dir) or die "Sorry.  Could not open $dir.\n";
my @allfiles = readdir THEDIR;
closedir THEDIR;

# find and process one version of each test case in the given dir
my $testcase = undef;
foreach my $file (@allfiles)
{
    if ($file =~ /(\w*-\d*)-(min.c|med.c|large.c|ok.c)/)
    {
        #print "Processing: $file ...\n";
        $testcase = $1;
        if (exists($TestCases{$testcase}))
        {
            #print "\tAlready processed $testcase\n";
            next;
        }
    }
    else
    {
        #print "Skipped $file\n";
        next;
    }

    # open the file and find its taxonomy classification string
    open(THISFILE, "<", "$dir/$file") or die "Sorry.  Could not open
$dir/$file.\n";
    my $class_value = undef;
    while (<THISFILE>)
    {
        if ($_ =~ /Taxonomy Classification: (\d*) /)
        {
            $class_value = $1;
            last;
        }
    }

    if (!defined($class_value))
    {
        print "FAILED to find taxonomy info in $dir/$file\n";
    }
    else
    {
        # save the taxonomy classification string in the hash
        $TestCases{$testcase} = $class_value;
    }
}

# now that we've processed all the files in the directory, print out
the
# info we collected in database table format
for my $entry (sort keys %TestCases)
{
    # the first column is the test case name
    print "$entry";

    # next we want each digit of the taxonomy classification in its own
    # column except for the size digit - we want to skip that one
    my $tax_string = $TestCases{$entry};
    my $this_char;
    for (my $i = 0; $i < length $tax_string; $i++)
    {
        # skip the magnitude digit
        if ($i == 19)
        {
            next;
        }
        # for every other digit, precede it with a tab
        $this_char = substr($tax_string, $i, 1);
        print "\t$this_char";
    }
    print "\n";
```

```
}

exit(0);

##----------------------------------------------------------------
## subroutines  (alphabetized)
##----------------------------------------------------------------
##----------------------------------------------------------------
## usage : print out an optional error message, followed by usage
## information, and then die
##----------------------------------------------------------------
sub usage
{
    if (scalar(@_) > 0)
    {
        print shift;
    }

    die "\nUsage:  gen_testcase_db_info.pl <dir>\n";
};
__END__

=cut

=head1 NAME

    # give your program name and a very short description
    gen_testcase_db_info.pl - Finds all of the test cases in the given
        directory, collects their names and taxonomy classification,
and
        prints report to stdout in database table format (test case
name first,
        followed by taxonomy classification digits [except for the size
digit]
        all tab-delimited).

=head1 SYNOPSIS

    # show examples of how to use your program
    ./gen_testcase_db_info.pl .

=head1 DESCRIPTION

    # describe in more detail what your_program does

=head1 OPTIONS

    # document your_program's options in sub-sections here
=head2 --option1

    # describe what option does

=head1 AUTHOR

    # provide your name and contact information
    Kendra Kratkiewicz, MIT Lincoln Laboratory

=cut
```

gen_testcase_desc.pl

```
#!/usr/bin/perl -w

use strict;

# Copyright 2004 M.I.T.

##--------------------------------------------------------------
## File globals
##--------------------------------------------------------------
# There will be one entry for each test case (minus the suffix
# ok/min/med/large), and the value will be the taxonomy
# classification string.
my %TestCases;

my $OK_SIZE = 0;
my $MIN_SIZE = 1;
my $MED_SIZE = 8;
my $LARGE_SIZE = 4096;

# associate taxonomy characteristics with their values in an array
my @TaxonomyInfo = (
    ['WRITE/READ', {0 => 'write', 1 => 'read'}],
    ['WHICH BOUND', {0 => 'upper', 1 => 'lower'}],
    ['DATA TYPE', {0 => 'char', 1 => 'int', 2 => 'float',
                   3 => 'wchar', 4 => 'pointer', 5 => 'unsigned int',
                   6 => 'unsigned char'}],
    ['MEMORY LOCATION', {0 => 'stack', 1 => 'heap',
                      2 => 'data segment',
                      3 => 'bss', 4 => 'shared'}],
    ['SCOPE', {0 => 'same scope', 1 => 'inter-procedural',
               2 => 'global',
               3 => 'inter-file/inter-proc',
               4 => 'inter-file/global'}],
    ['CONTAINER', {0 => 'no container', 1 => 'array', 2 => 'struct',
                   3 => 'union', 4 => 'array of structs',
                   5 => 'array of unions'}],
    ['POINTER', {0 => 'no pointer', 1 => 'pointer'}],
    ['INDEX COMPLEXITY', {0 => 'index:constant', 1 => 'index:variable',
                          2 => 'index:linear expr',
                          3 => 'index:non-linear expr',
                          4 => 'index:func ret val',
                          5 => 'index:array contents',
                          6 => 'index:N/A'}],
    ['ADDRESS COMPLEXITY', {0 => 'addr:constant', 1 => 'addr:variable',
                            2 => 'addr:linear expr',
                            3 => 'addr:non-linear expr',
                            4 => 'addr:func ret val',
                            5 => 'addr:array contents'}],
    ['LENGTH COMPLEXITY', {0 => 'length:N/A', 1 => 'length:none',
                           2 => 'length:constant',
                           3 => 'length:variable',
                           4 => 'length:linear expr',
                         5 => 'length:non-linear expr',
                         6 => 'length:func ret val',
                         7 => 'length:array contents'}],
    ['ADDRESS ALIAS', {0 => 'aliasaddr:none', 1 => 'aliasaddr:one',
                       2 => 'aliasaddr:two'}],
    ['INDEX ALIAS', {0 => 'aliasindex:none', 1 => 'aliasindex:one',
                     2 => 'aliasindex:two',
                     3 => 'aliasindex:N/A'}],
    ['LOCAL CONTROL FLOW', {0 => 'localflow:none', 1 => 'localflow:if',
                            2 => 'localflow:switch',
                            3 => 'localflow:cond',
                            4 => 'localflow:goto',
                            5 => 'localflow:longjmp',
                            6 => 'localflow:funcptr',
                            7 => 'localflow:recursion'}],
    ['SECONDARY CONTROL FLOW', {0 => '2ndflow:none', 1 => '2ndflow:if',
                                2 => '2ndflow:switch',
                                3 => '2ndflow:cond',
                         4 => '2ndflow:goto', 5 => '2ndflow:longjmp',
                         6 => '2ndflow:funcptr',
                         7 => '2ndflow:recursion'}],
    ['LOOP STRUCTURE', {0 => 'no loop', 1 => 'for', 2 => 'do-while',
                        3 => 'while',
                        4 => 'non-standard for',
                        5 => 'non-standard do-while',
                        6 => 'non-standard while'}],
    ['LOOP COMPLEXITY', {0 => 'loopcomplex:N/A',
                         1 => 'loopcomplex:zero',
                         2 => 'loopcomplex:one',
                         3 => 'loopcomplex:two',
                         4 => 'loopcomplex:three'}],
    ['ASYNCHRONY', {0 => 'no asynchrony', 1 => 'threads',
                    2 => 'forked process',
                    3 => 'signal handler'}],
    ['TAINT', {0 => 'no taint', 1 => 'argc/argv', 2 => 'env var',
               3 => 'file read', 4 => 'socket', 5 => 'proc env'}],
    ['RUNTIME ENV. DEPENDENCE', {0 => 'not runtime env dep',
                                 1 => 'runtime env dep'}],
    ['MAGNITUDE', {0 => 'no overflow', 1 => "$MIN_SIZE byte",
                   2 => "$MED_SIZE bytes", 3 => "$LARGE_SIZE bytes"}],
    ['CONTINUOUS/DISCRETE', {0 => 'discrete', 1 => 'continuous'}],
    ['SIGNEDNESS', {0 => 'no sign', 1 => 'sign'}]);

# definitions for array indices related to the TaxonomyInfo array
use constant NAME_INDEX => 0;
use constant VALUES_INDEX => 1;


##----------------------------------------------------------------
## main program
##----------------------------------------------------------------

#----------------------------------------------------------------
# process arguments
#----------------------------------------------------------------
```

```
# check to see if we have the right number of arguments
if (@ARGV < 1)
{
    usage();
}

# check arguments for validity
my $dir = $ARGV[0];
-e $dir or die "Sorry.  Directory $dir does not exist.\n";
-d $dir or die "Sorry.  $dir is not a directory.\n";

#-------------------------------------------------------------
# here's the beef
#-------------------------------------------------------------
opendir(THEDIR, $dir) or die "Sorry.  Could not open $dir.\n";
my @allfiles = readdir THEDIR;
closedir THEDIR;

# find and process one version of each test case in the given dir
my $testcase = undef;
foreach my $file (@allfiles)
{
    if ($file =~ /(\w*-\d*)-(min.c|med.c|large.c|ok.c)/)
    {
        #print "Processing: $file ...\n";
        $testcase = $1;
        if (exists($TestCases{$testcase}))
        {
            #print "\tAlready processed $testcase\n";
            next;
        }
    }
    else
    {
        #print "Skipped $file\n";
        next;
    }

    # open the file and find its taxonomy classification string
    open(THISFILE, "<", "$dir/$file") or die "Sorry.  Could not open
$dir/$file.\n";
    my $class_value = undef;
    while (<THISFILE>)
    {
        if ($_ =~ /Taxonomy Classification: (\d*) /)
        {
            $class_value = $1;
            last;
        }
    }

    if (!defined($class_value))
    {
        print "FAILED to find taxonomy info in $dir/$file\n";
    }
    else
    {
        # save the taxonomy classification string in the hash
        $TestCases{$testcase} = $class_value;
    }
}

# now that we've processed all the files in the directory, print out
# the info we collected (one row for each test case in the format:
#  test case name <TAB> desc1,desc2,desc3 etc.
# where the descX's are English descriptions of the non-zero attribute
# values (non-base case)
for my $entry (sort keys %TestCases)
{
    # the first column is the test case name
    print "$entry\t";

    # next, for each non-zero digit of the taxonomy classification,
    # loop up and print out its English description (except for the
    # size digit
    # - we want to skip that one)
    my $tax_string = $TestCases{$entry};
    my $this_char;
    for (my $i = 0; $i < length $tax_string; $i++)
    {
        # skip the magnitude digit
        if ($i == 19)
        {
            next;
        }
        # for every other digit, if it's not zero, print out its
        # description (comma separate them)
        $this_char = substr($tax_string, $i, 1);
        if ($this_char != 0)
        {
            print "$TaxonomyInfo[$i]->[VALUES_INDEX]->{$this_char},";
        }
    }
    print "\n";
}

exit(0);

##----------------------------------------------------------------
## subroutines  (alphabetized)
##----------------------------------------------------------------
##----------------------------------------------------------------
## usage : print out an optional error message, followed by usage
## information, and then die
##----------------------------------------------------------------
sub usage
{
    if (scalar(@_) > 0)
    {
        print shift;
    }

    die "\nUsage:  gen_testcase_desc.pl <dir>\n";
};
__END__

=cut

=head1 NAME

    # give your program name and a very short description
    gen_testcase_desc.pl - Finds all of the test cases in the given
        directory, collects their names and taxonomy classification,
and
        prints to stdout a report that lists each test case and an
English
        description of its non-zero attributes (those that differ from
the
        base case).

=head1 SYNOPSIS

    # show examples of how to use your program
    ./gen_testcase_desc.pl .

=head1 DESCRIPTION

    # describe in more detail what your_program does

=head1 OPTIONS

    # document your_program's options in sub-sections here
=head2 --option1

    # describe what option does

=head1 AUTHOR

    # provide your name and contact information
    Kendra Kratkiewicz, MIT Lincoln Laboratory

=cut
```

tabulate_archer_results.pl

```
#!/usr/bin/perl -w

use strict;

# Copyright 2004 M.I.T.

##----------------------------------------------------------------
## File globals
##----------------------------------------------------------------
my %Errors;  # a hash that holds file names/line numbers of bounds
errors
my %Results; # a hash that holds all of the detection results
my $TotalErrors = 0; # total number of bounds errors detected

##----------------------------------------------------------------
## main program
##----------------------------------------------------------------

#----------------------------------------------------------------
# process arguments
#----------------------------------------------------------------
# check to see if we have the right number of arguments
if (@ARGV < 2)
{
    usage();
}

# check arguments for validity
my $src_dir = $ARGV[0];
-e $src_dir or die "Sorry.  Directory $src_dir does not exist.\n";
-d $src_dir or die "Sorry.  $src_dir is not a directory.\n";

my $results_dir = $ARGV[1];
-e $results_dir or die "Sorry.  Directory $results_dir does not
exist.\n";
-d $results_dir or die "Sorry.  $results_dir is not a directory.\n";

#----------------------------------------------------------------
# here's the beef
#----------------------------------------------------------------
# process the results directory to find reported errors
opendir(OUTDIR, $results_dir) or die "Sorry.  Could not open
$results_dir.\n";
my @resfiles = readdir OUTDIR;
closedir OUTDIR;

foreach my $resfile (@resfiles)
{
    if (!($resfile =~ /\w*-\d*-(ok|min|med|large)/))
    {
          #print "Skipping: $resfile\n";
          next;
    }
    #print "Processing results file: $resfile\n";
    find_errors("$results_dir/$resfile", $resfile);
}

# process the source directory against the errors we found
opendir(SRCDIR, $src_dir) or die "Sorry.  Could not open $src_dir.\n";
my @srcfiles = readdir SRCDIR;
closedir SRCDIR;

foreach my $srcfile (@srcfiles)
{
    if (!($srcfile =~ /(\w*-\d*)-(ok|min|med|large).c/))
    {
          #print "Skipping: $srcfile\n";
          next;
    }
    else
    {
        #print "Processing source file: $srcfile\n";
        process_src_file("$src_dir/$srcfile", $1, $2);
    }
}

# report all of the results recorded in our Results hash
# first print a header line
print "Test case\tOK\tMIN\tMED\tLARGE\n";
foreach my $testcase (sort keys %Results)
{
    my $ok_result = "-";
    if (exists($Results{$testcase}{"ok"}))
    {
        $ok_result = $Results{$testcase}{"ok"};
    }
    my $min_result = "-";
    if (exists($Results{$testcase}{"min"}))
    {
        $min_result = $Results{$testcase}{"min"};
    }
    my $med_result = "-";
    if (exists($Results{$testcase}{"med"}))
    {
        $med_result = $Results{$testcase}{"med"};
    }
    my $large_result = "-";
    if (exists($Results{$testcase}{"large"}))
    {
        $large_result = $Results{$testcase}{"large"};
    }
    printf ("$testcase\t%d\t%d\t%d\t%d\n", $ok_result, $min_result,
$med_result, $large_result);
}
#print "\nTotal Errors Detected = $TotalErrors\n";

exit(0);
```

```
##--------------------------------------------------------------
## subroutines  (alphabetized)
##--------------------------------------------------------------
##--------------------------------------------------------------
## find_errors : tabulate (by file name and line number) all the
##  errors found in the given results file.
##--------------------------------------------------------------
sub find_errors
{
    my ($filepath, $file) = @_;

      open(THISFILE, "<", $filepath) or die "Sorry.  Could not open
$filepath.\n";
      while (<THISFILE>)
      {
        # example error line:
        #/home/kendra/ThesisTestSuite/basic-00001-large.c:56:main:
ERROR:BUFFER:56:56:Array bounds error (len < off+1) (buf[4105], len =
10, off = 4105, max(len-off) = -4095)
          if ($_ =~ /([Aa]rray bounds error|out-of-bound array or
string access)/)
          {
            ++$TotalErrors;
            if ($_ =~ /$file.*\:([0-9]+)\:([0-9]+)\:/)
            {
                  $Errors{$file}{$1} = 1;
            }
          }
      }
      close(THISFILE);
}

##--------------------------------------------------------------
## get_badok_line : return the line number of the BAD/OK line (the line
##    after the BAD/OK comment).  Returns -1 if no such line found.
##--------------------------------------------------------------
sub get_badok_line
{
    my $file = shift;

      open(THISFILE, "<", $file) or die "Sorry.  Could not open
$file.\n";
      my $foundit = 0;
      my $linenum = 1;
      while (<THISFILE>)
      {
          ++$linenum;
          if ($_ =~ /\/\*\s*?(BAD|OK)\s*?\*\//)
          {
            $foundit = 1;
            last;
          }
      }
      close(THISFILE);
    if (!$foundit)
      {
          return -1;
      }
    else
    {
        return $linenum;
    }
}

##----------------------------------------------------------------
## process_src_file : process the given source file to determine if an
error
##        was reported for the annotated (BAD/OK) line
##----------------------------------------------------------------
sub process_src_file
{
    my ($src_file, $root, $suffix) = @_;

    my $badok_line = get_badok_line($src_file);
    #print "BAD/OK line number: $badok_line\n";

    # record in the Results hash whether or not an error was reported
    #  on the bad/ok line for this file
    $Results{$root}{$suffix} =
      exists($Errors{"$root-$suffix"}{$badok_line});
}

##----------------------------------------------------------------
## usage : print out an optional error message, followed by usage
## information, and then die
##----------------------------------------------------------------
sub usage
{
    if (scalar(@_) > 0)
    {
        print shift;
    }

    die "\nUsage:  tabulate_archer_results.pl <testcase src dir>
<archer results dir>\n";
};
__END__

=cut

=head1 NAME

    # give your program name and a very short description
    tabulate_archer_results.pl - parses Archer results to see if it
reported
    errors when it should have and didn't when it shouldn't have.
Prints
    report to stdout.
```

```
=head1 SYNOPSIS

    # show examples of how to use your program
    ./tabulate_archer_results.pl testcase_src_dir archer_results_dir

=head1 DESCRIPTION

    # describe in more detail what your_program does

=head1 OPTIONS

    # document your_program's options in sub-sections here
=head2 --option1

    # describe what option does

=head1 AUTHOR

    # provide your name and contact information
    Kendra Kratkiewicz, MIT Lincoln Laboratory

=cut
```

tabulate_boon_results.pl

```
#!/usr/bin/perl -w

use strict;

# Copyright 2004 M.I.T.

##----------------------------------------------------------------
## File globals
##----------------------------------------------------------------
my %Errors;  # a hash that holds file names/line numbers of bounds
errors
my %Results; # a hash that holds all of the detection results
my $TotalErrors = 0; # total number of bounds errors detected

##----------------------------------------------------------------
## main program
##----------------------------------------------------------------

#----------------------------------------------------------------
# process arguments
#----------------------------------------------------------------
# check to see if we have the right number of arguments
if (@ARGV < 2)
{
    usage();
}

# check arguments for validity
my $src_dir = $ARGV[0];
-e $src_dir or die "Sorry.  Directory $src_dir does not exist.\n";
-d $src_dir or die "Sorry.  $src_dir is not a directory.\n";

my $results_dir = $ARGV[1];
-e $results_dir or die "Sorry.  Directory $results_dir does not
exist.\n";
-d $results_dir or die "Sorry.  $results_dir is not a directory.\n";

#----------------------------------------------------------------
# here's the beef
#----------------------------------------------------------------
# process the results directory to find reported errors
opendir(OUTDIR, $results_dir) or die "Sorry.  Could not open
$results_dir.\n";
my @resfiles = readdir OUTDIR;
closedir OUTDIR;

foreach my $resfile (@resfiles)
{
    if (!($resfile =~ /\w*-\d*-(ok|min|med|large)/))
    {
          #print "Skipping: $resfile\n";
          next;
    }
    #print "Processing results file: $resfile\n";
    find_errors("$results_dir/$resfile", $resfile);
}

# process the source directory against the errors we found
opendir(SRCDIR, $src_dir) or die "Sorry.  Could not open $src_dir.\n";
my @srcfiles = readdir SRCDIR;
closedir SRCDIR;

foreach my $srcfile (@srcfiles)
{
    if (!($srcfile =~ /(\w*-\d*)-(ok|min|med|large).c/))
    {
          #print "Skipping: $srcfile\n";
          next;
    }
    else
    {
        #print "Processing source file: $srcfile\n";
        process_src_file("$src_dir/$srcfile", $1, $2);
    }
}

# report all of the results recorded in our Results hash
# first print a header line
print "Test case\tOK\tMIN\tMED\tLARGE\n";
foreach my $testcase (sort keys %Results)
{
    my $ok_result = "-";
    if (exists($Results{$testcase}{"ok"}))
    {
        $ok_result = $Results{$testcase}{"ok"};
    }
    my $min_result = "-";
    if (exists($Results{$testcase}{"min"}))
    {
        $min_result = $Results{$testcase}{"min"};
    }
    my $med_result = "-";
    if (exists($Results{$testcase}{"med"}))
    {
        $med_result = $Results{$testcase}{"med"};
    }
    my $large_result = "-";
    if (exists($Results{$testcase}{"large"}))
    {
        $large_result = $Results{$testcase}{"large"};
    }
    printf ("$testcase\t%d\t%d\t%d\t%d\n", $ok_result, $min_result,
$med_result, $large_result);
}
#print "\nTotal Errors Detected = $TotalErrors\n";

exit(0);
```

```
##-------------------------------------------------------------
## subroutines  (alphabetized)
##-------------------------------------------------------------
##-------------------------------------------------------------
## find_errors : tabulate (by file name and line number) all the
##  errors found in the given results file.
##-------------------------------------------------------------
sub find_errors
{
    my ($filepath, $file) = @_;

      open(THISFILE, "<", $filepath) or die "Sorry.  Could not open
$filepath.\n";
      while (<THISFILE>)
      {
# sample error:
#        POSSIBLE VULNERABILITIES:
#        Almost certainly a buffer overflow in `buf@main()':
#          10..10 bytes allocated, 4106..4106 bytes used.
#          <- siz(buf@main())
#          <- len(buf@main())
#        Slight chance of a buffer overflow in
`__assertion@__assert_fail()':
#          15..17 bytes allocated, 15..17 bytes used.
#          <- siz(__assertion@__assert_fail())
#          <- len(__assertion@__assert_fail())
#        Possibly a buffer overflow in `buf@main()':
#          10..10 bytes allocated, 1..4106 bytes used.
#          <- siz(buf@main())
#          <- len(buf@main())

          if ($_ =~ /buffer overflow/)
          {
            ++$TotalErrors;
            # Boon doesn't report line number, so we'll just report the
            # line number as unspecified and assume it would match –
            # best we can do
            $Errors{$file}{'unspecified'} = 1;
          }
      }
      close(THISFILE);
}

##-------------------------------------------------------------
## get_badok_line : return the line number of the BAD/OK line (the line
##    after the BAD/OK comment).  Returns -1 if no such line found.
##-------------------------------------------------------------
sub get_badok_line
{
    my $file = shift;

      open(THISFILE, "<", $file) or die "Sorry.  Could not open
$file.\n";
      my $foundit = 0;
      my $linenum = 1;
      while (<THISFILE>)
      {
          ++$linenum;
          if ($_ =~ /\/\*\s*?(BAD|OK)\s*?\*\//)
          {
            $foundit = 1;
            last;
          }
      }
      close(THISFILE);

    if (!$foundit)
      {
          return -1;
      }
    else
    {
        return $linenum;
    }
}

##------------------------------------------------------------------
## process_src_file : process the given source file to determine if an
error
##      was reported for the annotated (BAD/OK) line
##------------------------------------------------------------------
sub process_src_file
{
    my ($src_file, $root, $suffix) = @_;

    # we actually don't need the bad/ok line number, since boon doesn't
    # report line numbers anyway (so we have nothing to compare against
    #my $badok_line = get_badok_line($src_file);
    #print "BAD/OK line number: $badok_line\n";

    # record in the Results hash whether or not an error was reported
    # for this file
    $Results{$root}{$suffix} =
      exists($Errors{"$root-$suffix"}{'unspecified'});
}

##------------------------------------------------------------------
## usage : print out an optional error message, followed by usage
## information, and then die
##------------------------------------------------------------------
sub usage
{
    if (scalar(@_) > 0)
    {
        print shift;
    }

    die "\nUsage:  tabulate_boon_results.pl <testcase src dir> <boon
results dir>\n";
};
__END__

=cut

=head1 NAME

    # give your program name and a very short description
    tabulate_boon_results.pl - parses Boon results to see if it
reported
    errors when it should have and didn't when it shouldn't have.
Prints
    report to stdout.

=head1 SYNOPSIS

    # show examples of how to use your program
    ./tabulate_boon_results.pl testcase_src_dir boon_results_dir

=head1 DESCRIPTION

    # describe in more detail what your_program does

=head1 OPTIONS

    # document your_program's options in sub-sections here
=head2 --option1

    # describe what option does

=head1 AUTHOR

    # provide your name and contact information
    Kendra Kratkiewicz, MIT Lincoln Laboratory

=cut
```

tabulate_polyspace_results.pl

```
#!/usr/bin/perl -w

use strict;

# Copyright 2004 M.I.T.

##----------------------------------------------------------------
## File globals
##----------------------------------------------------------------
my %Errors;  # a hash that holds file names/line numbers of bounds
errors
my %Results; # a hash that holds all of the detection results
my $TotalErrors = 0; # total number of bounds errors detected

##----------------------------------------------------------------
## main program
##----------------------------------------------------------------

#----------------------------------------------------------------
# process arguments
#----------------------------------------------------------------
# check to see if we have the right number of arguments
if (@ARGV < 2)
{
    usage();
}

# check arguments for validity
my $src_dir = $ARGV[0];
-e $src_dir or die "Sorry.  Directory $src_dir does not exist.\n";
-d $src_dir or die "Sorry.  $src_dir is not a directory.\n";

my $results_dir = $ARGV[1];
-e $results_dir or die "Sorry.  Directory $results_dir does not
exist.\n";
-d $results_dir or die "Sorry.  $results_dir is not a directory.\n";

#----------------------------------------------------------------
# here's the beef
#----------------------------------------------------------------
# process the results directory to find reported errors
opendir(OUTDIR, $results_dir) or die "Sorry.  Could not open
$results_dir.\n";
my @resdirs = readdir OUTDIR;
closedir OUTDIR;

foreach my $resdir (@resdirs)
{
    if (!($resdir =~ /\w*-\d*-(ok|min|med|large)/))
    {
          #print "Skipping: $resdir\n";
          next;
    }
    #print "Processing results file: $resdir/polyspace_RTE_View.txt\n";
    find_errors("$results_dir/$resdir/polyspace_RTE_View.txt",
$resdir);
}

# For PolySpace, we also need to look for errors reported in library
# functions if the source file calls a library function to perform the
# overflow.  This is because PolySpace only reports the error in the
# library function, and does not tie it back to the line in the source
# program from which the library function was called.
foreach my $resdir (@resdirs)
{
    if (!($resdir =~ /\w*-\d*-(ok|min|med|large)/))
    {
          next;
    }
    find_lib_errors("$results_dir/$resdir/polyspace_RTE_View.txt",
$resdir, $src_dir);
}

# process the source directory against the errors we found
opendir(SRCDIR, $src_dir) or die "Sorry.  Could not open $src_dir.\n";
my @srcfiles = readdir SRCDIR;
closedir SRCDIR;

foreach my $srcfile (@srcfiles)
{
    if (!($srcfile =~ /(\w*-\d*)-(ok|min|med|large).c/))
    {
          #print "Skipping: $srcfile\n";
          next;
    }
    else
    {
        #print "Processing source file: $srcfile\n";
        process_src_file("$src_dir/$srcfile", $1, $2);
    }
}

# report all of the results recorded in our Results hash
# first print a header line
print "Test case\tOK\tMIN\tMED\tLARGE\n";
foreach my $testcase (sort keys %Results)
{
    my $ok_result = "-";
    if (exists($Results{$testcase}{"ok"}))
    {
        $ok_result = $Results{$testcase}{"ok"};
    }
    my $min_result = "-";
    if (exists($Results{$testcase}{"min"}))
    {
        $min_result = $Results{$testcase}{"min"};
    }
    my $med_result = "-";
    if (exists($Results{$testcase}{"med"}))
    {
        $med_result = $Results{$testcase}{"med"};
    }
    my $large_result = "-";
    if (exists($Results{$testcase}{"large"}))
    {
        $large_result = $Results{$testcase}{"large"};
    }
    printf ("$testcase\t%d\t%d\t%d\t%d\n", $ok_result, $min_result,
$med_result, $large_result);
}
#print "\nTotal Errors Detected = $TotalErrors\n";

exit(0);

##----------------------------------------------------------------
## subroutines  (alphabetized)
##----------------------------------------------------------------
##----------------------------------------------------------------
## find_errors : tabulate (by testcase name and line number) all the
##  errors found in the given results file.
##----------------------------------------------------------------
sub find_errors
{
    my ($filepath, $testcase) = @_;

      open(THISFILE, "<", $filepath) or die "Sorry.  Could not open
$filepath.\n";
    my @fields;
      while (<THISFILE>)
      {
        # example error line:
        # |  |  |  ! OBAI.0   1                       56    5
      array index within bounds     76
        # fields[0] will have the error name in it, if any
        # fields[1] is the "red" error indicator, while fields[2] is
the "orange" error
        # fields[5] is the line number, fields[8] is the detailed
comment
      @fields = split /\t/;
        # if the red or orange flag is set, and if the error reported
is one
        # that we count as a buffer overflow detection, then we'll save
the
        # line number
          if ((scalar @fields > 8) &&
            (($fields[1] ne "") || ($fields[2] ne "")) &&
            (($fields[0] =~ /(IDP|NIP|OBAI)/) ||
             (($fields[0] =~ /COR/) && ($fields[8] =~
                                          /illegal pointer access/))))
          {
            ++$TotalErrors;
            $Errors{$testcase}{$fields[5]} = 1;
            }
        }
        close(THISFILE);
}

##------------------------------------------------------------------
## find_lib_errors : tabulate (by testcase name and -1 for line number)
any
##  errors found in library functions called (on the BAD/OK line) by
the source
##  program.
##------------------------------------------------------------------
sub find_lib_errors
{
    my ($filepath, $testcase, $src_dir) = @_;

    # first get the name of the library function, if any, called by the
    # source program on the BAD/OK line
    my $lib_func_name = get_lib_func_name("$src_dir/$testcase.c");
    if (defined($lib_func_name))
    {
        #print "$src_dir/$testcase calls function: \t$lib_func_name\n";

        open(THISFILE, "<", $filepath)
            or die "Sorry.  Could not open $filepath.\n";

        my @fields;
        my $indent_level;
        # first, look for a report line that has the function name in
        # the first field
        while (<THISFILE>)
        {
            @fields = split /\t/;
            if ((scalar @fields > 8) &&
                ($fields[0] =~ /$lib_func_name/))
            {
                #print "\tfound line for $lib_func_name in results
file\n";
                # now get the indent level of this function (count the
                # number of vertical bars)
                $indent_level = get_indent_level($fields[0]);
                #print "\tindent level = $indent_level\n";
                last;
            }
        }

        # now examine every subsequent line of indent_level + 1 to see
        # if any bounds errors were reported within this library
        # function
        while (<THISFILE>)
        {
            @fields = split /\t/;
            if ((scalar @fields > 8) &&
                (get_indent_level($fields[0]) > $indent_level))
            {
                if (defined(line_has_error($_)))
                {
                    #print "\tlib function contained bounds error\n";
                    ++$TotalErrors;
                    # use -1 for line number
                    $Errors{$testcase}{-1} = 1;
                }
            }
            else
            {
                # if we hit a line that's not the right format or has a
                # different indent level, exit this loop
                last;
            }
        }
        close(THISFILE);
    }

}

##--------------------------------------------------------------
## get_badok_line : return the line number of the BAD/OK line (the line
##    after the BAD/OK comment).  Returns -1 if no such line found.
##--------------------------------------------------------------
sub get_badok_line
{
    my $file = shift;

      open(THISFILE, "<", $file) or die "Sorry.  Could not open
$file.\n";
      my $foundit = 0;
      my $linenum = 1;
      while (<THISFILE>)
      {
          ++$linenum;
          if ($_ =~ /\/\*\s*?(BAD|OK)\s*?\*\//)
          {
            $foundit = 1;
            last;
          }
      }
      close(THISFILE);

    if (!$foundit)
    {
          return -1;
    }
    else
    {
        return $linenum;
    }
}

##--------------------------------------------------------------
## get_indent_level : return the number of vertical bars in the given
## string
##---------------------------------------------------------------
sub get_indent_level
{
    my $str = shift;
    my $result = 0;

    my @fields = split(/ /, $str);
    for (my $i = 0; $i < scalar @fields; ++$i)
    {
        if ($fields[$i] =~ /\|/)
        {
            ++$result;
        }
    }

    return $result;
}

##---------------------------------------------------------------
## get_lib_func_name : returns the name of the library function called
on the
## BAD/OK line of the source program.  If no library function is
called,
## returns undef.
##---------------------------------------------------------------
sub get_lib_func_name
{
    my $src_file = shift;
    my $lib_func_name = undef;

    open(SRCFILE, "<", $src_file)
        or die "Sorry.  Could not open $src_file.\n";

    # first look at the taxonomy classification value to see if a
library
    # function is used to cause the overflow
    my $class_value = undef;
    while (<SRCFILE>)
    {
        if ($_ =~ /Taxonomy Classification: (\d*) /)
        {
            $class_value = $1;
            last;
        }
    }

    if (!defined($class_value))
    {
        print "FAILED to find taxonomy info in $src_file\n";
    }
    else
    {
        # now get the value of the length/limit complexity digit
        my $len_limit_complexity = substr($class_value, 9, 1);
        if ($len_limit_complexity > 0)
        {
            # this test case calls a library function, so keep going
            # find the BAD/OK line; the next line will have the call
            my $foundit = 0;
            while (<SRCFILE>)
            {
                if ($_ =~ /\/\*\s*?(BAD|OK)\s*?\*\//)
                {
                    $foundit = 1;
                    last;
                }
            }

            if ($foundit)
            {
                my $src_line = <SRCFILE>;
                if ($src_line =~ /(\w+)\s*\(/)
                {
                    $lib_func_name = $1;
                }
            }
        }
    }

    return $lib_func_name;
}

##----------------------------------------------------------------
## line_has_error : returns the reported line number of the error if
the
## given line from a PolySpace results file contains a buffer overflow
error
## report; returns 0 if no such error is reported on the line
##----------------------------------------------------------------
sub line_has_error
{
    my $line = shift;
    my $error_reported = undef;

    # example error line:
    # |  |  |  ! OBAI.0 1                       56    5           array
index within bounds     76
    # fields[0] will have the error name in it, if any
    # fields[1] is the "red" error indicator, while fields[2] is the
"orange" error
    # fields[5] is the line number, fields[8] is the detailed comment
    my @fields = split(/\t/, $line);
    # if the red or orange flag is set, and if the error reported is
one
    # that we count as a buffer overflow detection, then we'll save the
    # line number
    if ((scalar @fields > 8) &&
        (($fields[1] ne "") || ($fields[2] ne "")) &&
        (($fields[0] =~ /(IDP|NIP|OBAI)/) ||
         (($fields[0] =~ /COR/) &&
          ($fields[8] =~ /illegal pointer access/))))
    {
        $error_reported = $fields[5];
    }

    return $error_reported;
}

##---------------------------------------------------------------
## process_src_file : process the given source file to determine if an
error
##      was reported for the annotated (BAD/OK) line
##---------------------------------------------------------------
sub process_src_file
{
    my ($src_file, $root, $suffix) = @_;

    my $badok_line = get_badok_line($src_file);
    #print "BAD/OK line number: $badok_line\n";

    # record in the Results hash whether or not an error was reported
    # on the bad/ok line for this file (or if an error was reported in
    # a library function)
    $Results{$root}{$suffix} =
      (exists($Errors{"$root-$suffix"}{$badok_line}) ||
                                exists($Errors{"$root-$suffix"}{-1}));
}

##---------------------------------------------------------------
## usage : print out an optional error message, followed by usage
## information, and then die
##---------------------------------------------------------------
sub usage
{
    if (scalar(@_) > 0)
    {
        print shift;
    }

    die "\nUsage:  tabulate_polyspace_results.pl <testcase src dir>
<polyspace results dir>\n";
};
__END__

=cut

=head1 NAME

    # give your program name and a very short description
    tabulate_polyspace_results.pl - parses PolySpace results to see if
it reported
    errors when it should have and didn't when it shouldn't have.
Prints report
    to stdout.
```

```
=head1 SYNOPSIS

    # show examples of how to use your program
    ./tabulate_polyspace_results.pl testcase_src_dir
polyspace_results_dir

=head1 DESCRIPTION

    # describe in more detail what your_program does

=head1 OPTIONS

    # document your_program's options in sub-sections here
=head2 --option1

    # describe what option does

=head1 AUTHOR

    # provide your name and contact information
    Kendra Kratkiewicz, MIT Lincoln Laboratory

=cut
```

tabulate_splint_results.pl

```
#!/usr/bin/perl -w

use strict;

# Copyright 2004 M.I.T.

##----------------------------------------------------------------
## File globals
##----------------------------------------------------------------
my %Errors;  # a hash that holds file names/line numbers of bounds
errors
my %Results; # a hash that holds all of the detection results
my $TotalErrors = 0; # total number of bounds errors detected

##----------------------------------------------------------------
## main program
##----------------------------------------------------------------

#----------------------------------------------------------------
# process arguments
#----------------------------------------------------------------
# check to see if we have the right number of arguments
if (@ARGV < 2)
{
    usage();
}

# check arguments for validity
my $src_dir = $ARGV[0];
-e $src_dir or die "Sorry.  Directory $src_dir does not exist.\n";
-d $src_dir or die "Sorry.  $src_dir is not a directory.\n";

my $results_dir = $ARGV[1];
-e $results_dir or die "Sorry.  Directory $results_dir does not
exist.\n";
-d $results_dir or die "Sorry.  $results_dir is not a directory.\n";

#----------------------------------------------------------------
# here's the beef
#----------------------------------------------------------------
# process the results directory to find reported errors
opendir(OUTDIR, $results_dir) or die "Sorry.  Could not open
$results_dir.\n";
my @resfiles = readdir OUTDIR;
closedir OUTDIR;

foreach my $resfile (@resfiles)
{
    if (!($resfile =~ /\w*-\d*-(ok|min|med|large)/))
    {
          #print "Skipping: $resfile\n";
          next;
    }
    #print "Processing results file: $resfile\n";
    find_errors("$results_dir/$resfile", $resfile);
}

# process the source directory against the errors we found
opendir(SRCDIR, $src_dir) or die "Sorry.  Could not open $src_dir.\n";
my @srcfiles = readdir SRCDIR;
closedir SRCDIR;

foreach my $srcfile (@srcfiles)
{
    if (!($srcfile =~ /(\w*-\d*)-(ok|min|med|large).c/))
    {
          #print "Skipping: $srcfile\n";
          next;
    }
    else
    {
        #print "Processing source file: $srcfile\n";
        process_src_file("$src_dir/$srcfile", $1, $2);
    }
}

# report all of the results recorded in our Results hash
# first print a header line
print "Test case\tOK\tMIN\tMED\tLARGE\n";
foreach my $testcase (sort keys %Results)
{
    my $ok_result = "-";
    if (exists($Results{$testcase}{"ok"}))
    {
        $ok_result = $Results{$testcase}{"ok"};
    }
    my $min_result = "-";
    if (exists($Results{$testcase}{"min"}))
    {
        $min_result = $Results{$testcase}{"min"};
    }
    my $med_result = "-";
    if (exists($Results{$testcase}{"med"}))
    {
        $med_result = $Results{$testcase}{"med"};
    }
    my $large_result = "-";
    if (exists($Results{$testcase}{"large"}))
    {
        $large_result = $Results{$testcase}{"large"};
    }
    printf ("$testcase\t%d\t%d\t%d\t%d\n", $ok_result, $min_result,
$med_result, $large_result);
}
#print "\nTotal Errors Detected = $TotalErrors\n";

exit(0);
```

```
##----------------------------------------------------------------
## subroutines  (alphabetized)
##----------------------------------------------------------------
##----------------------------------------------------------------
## find_errors : tabulate (by file name and line number) all the
##  errors found in the given results file.
##----------------------------------------------------------------
sub find_errors
{
    my ($filepath, $file) = @_;

      open(THISFILE, "<", $filepath) or die "Sorry.  Could not open
$filepath.\n";
      while (<THISFILE>)
      {
        # sample error:
        # /home/kendra/ThesisTestSuite/basic-00001-large.c:56:3:
Possible out-of-bounds store:
          if ($_ =~ /Possible out-of-bounds/)
          {
            ++$TotalErrors;
            if ($_ =~ /$file.*\:([0-9]+)\:([0-9]+)\:/)
            {
                  $Errors{$file}{$1} = 1;
            }
          }
      }
      close(THISFILE);
}

##----------------------------------------------------------------
## get_badok_line : return the line number of the BAD/OK line (the line
##    after the BAD/OK comment).  Returns -1 if no such line found.
##----------------------------------------------------------------
sub get_badok_line
{
    my $file = shift;

      open(THISFILE, "<", $file) or die "Sorry.  Could not open
$file.\n";
      my $foundit = 0;
      my $linenum = 1;
      while (<THISFILE>)
      {
          ++$linenum;
          if ($_ =~ /\/\*\s*?(BAD|OK)\s*?\*\//)
          {
            $foundit = 1;
            last;
          }
      }
      close(THISFILE);

    if (!$foundit)
```

```
      {
          return -1;
      }
    else
    {
        return $linenum;
    }
}

##------------------------------------------------------------------
## process_src_file : process the given source file to determine if an
error
##      was reported for the annotated (BAD/OK) line
##------------------------------------------------------------------
sub process_src_file
{
    my ($src_file, $root, $suffix) = @_;

    my $badok_line = get_badok_line($src_file);
    #print "BAD/OK line number: $badok_line\n";

    # record in the Results hash whether or not an error was reported
    #  on the bad/ok line for this file
    $Results{$root}{$suffix} =
      exists($Errors{"$root-$suffix"}{$badok_line});
}

##------------------------------------------------------------------
## usage : print out an optional error message, followed by usage
## information, and then die
##------------------------------------------------------------------
sub usage
{
    if (scalar(@_) > 0)
    {
        print shift;
    }

    die "\nUsage:  tabulate_splint_results.pl <testcase src dir>
<splint results dir>\n";
};
__END__

=cut

=head1 NAME

    # give your program name and a very short description
    tabulate_splint_results.pl - parses Splint results to see if it
reported
    errors when it should have and didn't when it shouldn't have.
Prints
    report to stdout.

=head1 SYNOPSIS
```

```
    # show examples of how to use your program
    ./tabulate_splint_results.pl testcase_src_dir splint_results_dir

=head1 DESCRIPTION

    # describe in more detail what your_program does

=head1 OPTIONS

    # document your_program's options in sub-sections here
=head2 --option1

    # describe what option does

=head1 AUTHOR

    # provide your name and contact information
    Kendra Kratkiewicz, MIT Lincoln Laboratory

=cut
```

tabulate_uno_results.pl

```
#!/usr/bin/perl -w

use strict;

# Copyright 2004 M.I.T.

##----------------------------------------------------------------
## File globals
##----------------------------------------------------------------
my %Errors;  # a hash that holds file names/line numbers of bounds
errors
my %Results; # a hash that holds all of the detection results
my $TotalErrors = 0; # total number of bounds errors detected

##----------------------------------------------------------------
## main program
##----------------------------------------------------------------

#----------------------------------------------------------------
# process arguments
#----------------------------------------------------------------
# check to see if we have the right number of arguments
if (@ARGV < 2)
{
    usage();
}

# check arguments for validity
my $src_dir = $ARGV[0];
-e $src_dir or die "Sorry.  Directory $src_dir does not exist.\n";
-d $src_dir or die "Sorry.  $src_dir is not a directory.\n";

my $results_dir = $ARGV[1];
-e $results_dir or die "Sorry.  Directory $results_dir does not
exist.\n";
-d $results_dir or die "Sorry.  $results_dir is not a directory.\n";

#----------------------------------------------------------------
# here's the beef
#----------------------------------------------------------------
# process the results directory to find reported errors
opendir(OUTDIR, $results_dir) or die "Sorry.  Could not open
$results_dir.\n";
my @resfiles = readdir OUTDIR;
closedir OUTDIR;

foreach my $resfile (@resfiles)
{
    if (!($resfile =~ /\w*-\d*-(ok|min|med|large)/))
    {
          #print "Skipping: $resfile\n";
          next;
    }
    #print "Processing results file: $resfile\n";
    find_errors("$results_dir/$resfile", $resfile);
}

# process the source directory against the errors we found
opendir(SRCDIR, $src_dir) or die "Sorry.  Could not open $src_dir.\n";
my @srcfiles = readdir SRCDIR;
closedir SRCDIR;

foreach my $srcfile (@srcfiles)
{
    if (!($srcfile =~ /(\w*-\d*)-(ok|min|med|large).c/))
    {
          #print "Skipping: $srcfile\n";
          next;
    }
    else
    {
        #print "Processing source file: $srcfile\n";
        process_src_file("$src_dir/$srcfile", $1, $2);
    }
}

# report all of the results recorded in our Results hash
# first print a header line
print "Test case\tOK\tMIN\tMED\tLARGE\n";
foreach my $testcase (sort keys %Results)
{
    my $ok_result = "-";
    if (exists($Results{$testcase}{"ok"}))
    {
        $ok_result = $Results{$testcase}{"ok"};
    }
    my $min_result = "-";
    if (exists($Results{$testcase}{"min"}))
    {
        $min_result = $Results{$testcase}{"min"};
    }
    my $med_result = "-";
    if (exists($Results{$testcase}{"med"}))
    {
        $med_result = $Results{$testcase}{"med"};
    }
    my $large_result = "-";
    if (exists($Results{$testcase}{"large"}))
    {
        $large_result = $Results{$testcase}{"large"};
    }
    printf ("$testcase\t%d\t%d\t%d\t%d\n", $ok_result, $min_result,
$med_result, $large_result);
}
#print "\nTotal Errors Detected = $TotalErrors\n";

exit(0);
```

```
##----------------------------------------------------------------
## subroutines  (alphabetized)
##----------------------------------------------------------------
##----------------------------------------------------------------
## find_errors : tabulate (by file name and line number) all the
##  errors found in the given results file.
##----------------------------------------------------------------
sub find_errors
{
    my ($filepath, $file) = @_;

      open(THISFILE, "<", $filepath) or die "Sorry.  Could not open
$filepath.\n";
      while (<THISFILE>)
      {
        # sample errors:
        #
        # uno: basic-00178-large.c:  74 array index error 'buf[4105]'
(array-size: 10)
        # uno: foo.c: array index can be negative (%d)
        # uno: in fct main, array index can exceed upper-bound (10>9),
var 'loop_counter'
        #     statement  : basic-00259-min.c:64: buf[loop_counter]='A'
        #     declaration: basic-00259-min.c:54: int loop_counter;
          if ($_ =~ /array index error/)
          {
            ++$TotalErrors;
            if ($_ =~ /$file.*\:\s*([0-9]+)\s/)
            {
                $Errors{$file}{$1} = 1;
            }
          }
          elsif ($_ =~ /(array index can be negative|array index can
exceed upper-bound)/)
          {
            ++$TotalErrors;
            # look for line number on next line
            my $next_line = <THISFILE>;
            if ($next_line =~ /statement.*\:.*$file.*\:([0-9]+)\:/)
            {
                  $Errors{$file}{$1} = 1;
            }
          }
      }
      close(THISFILE);
}

##----------------------------------------------------------------
## get_badok_line : return the line number of the BAD/OK line (the line
##    after the BAD/OK comment).  Returns -1 if no such line found.
##----------------------------------------------------------------
sub get_badok_line
{
    my $file = shift;
```

```
      open(THISFILE, "<", $file) or die "Sorry.  Could not open
$file.\n";
      my $foundit = 0;
      my $linenum = 1;
      while (<THISFILE>)
      {
          ++$linenum;
          if ($_ =~ /\/\*\s*?(BAD|OK)\s*?\*\//)
          {
            $foundit = 1;
            last;
          }
      }
      close(THISFILE);

    if (!$foundit)
      {
          return -1;
      }
    else
    {
        return $linenum;
    }
}

##------------------------------------------------------------
## process_src_file : process the given source file to determine if an
error
##      was reported for the annotated (BAD/OK) line
##------------------------------------------------------------
sub process_src_file
{
    my ($src_file, $root, $suffix) = @_;

    my $badok_line = get_badok_line($src_file);
    #print "BAD/OK line number: $badok_line\n";

    # record in the Results hash whether or not an error was reported
    #  on the bad/ok line for this file
    $Results{$root}{$suffix} =
      exists($Errors{"$root-$suffix"}{$badok_line});
}

##------------------------------------------------------------
## usage : print out an optional error message, followed by usage
## information, and then die
##------------------------------------------------------------
sub usage
{
    if (scalar(@_) > 0)
    {
        print shift;
    }
    die "\nUsage:  tabulate_uno_results.pl <testcase src dir> <uno
results dir>\n";
};
__END__

=cut

=head1 NAME

    # give your program name and a very short description
    tabulate_uno_results.pl - parses Uno results to see if it reported
    errors when it should have and didn't when it shouldn't have.
Prints
    report to stdout.

=head1 SYNOPSIS

    # show examples of how to use your program
    ./tabulate_uno_results.pl testcase_src_dir uno_results_dir

=head1 DESCRIPTION

    # describe in more detail what your_program does

=head1 OPTIONS

    # document your_program's options in sub-sections here
=head2 --option1

    # describe what option does

=head1 AUTHOR

    # provide your name and contact information
    Kendra Kratkiewicz, MIT Lincoln Laboratory

=cut
```

SH Files

Archer_script.sh

```
#!/bin/sh
testdir=`pwd`
for file in `ls basic*.c`
do
    bs=`basename $file .c`
    rm -rf /home/kendra/temptree
    mkdir /home/kendra/temptree
    export LD_LIBRARY_PATH=/data/ia/tools/archer/global/global1.0/tree-
emit
    export MC_GLOBAL=/data/ia/tools/archer/global/global1.0
    export MCGCC=/data/ia/tools/archer/global/gcc-bin/bin/gcc
    /data/ia/tools/archer/global/gcc-bin/bin/gcc -fmc-
emit=/home/kendra/temptree \
        -c $file -lpthread
    cd /data/ia/tools/archer/archer
    /data/ia/tools/archer/archer/archer /home/kendra/temptree >
/home/kendra/Thesis/Archer_results/$bs
    cd $testdir
done
```

Boon_script.sh

```
#!/bin/sh
testdir=`pwd`
export PATH="$PATH:/data/ia/tools/boon"
for file in `ls basic*.c`
do
    bs=`basename $file .c`
    cd /data/ia/tools/boon
    ./preproc $testdir/$file
    ./boon $testdir/$bs.i >& /home/kendra/Thesis/Boon_results/$bs
    rm $testdir/$bs.i
done
```

PolySpace_script.sh

```
#!/bin/sh
for file in `ls basic*.c`
do
    bs=`basename $file .c`
    /work/tools/PolySpace/2.5/bin/polyspace-c \
    -continue-with-existing-host \
    -target i386 \
    -OS-target linux \
    -I /work/tools/PolySpace/2.5/include/include-linux \
    -permissive \
    -sources $file \
    -results-dir /home/kendra/PolySpace_results/$bs
done
```

PolySpace_script_basic-00006.sh

```
#!/bin/sh
for file in `ls basic*.c`
do
    bs=`basename $file .c`
    /work/tools/PolySpace/2.5/bin/polyspace-c \
    -continue-with-existing-host \
    -target i386 \
    -OS-target linux \
    -I /work/tools/PolySpace/2.5/include/include-linux \
    -permissive \
    -ignore-constant-overflows \
    -include /work/tools/PolySpace/2.5/include/include-
linux/sys/types.h \
    -sources $file \
    -results-dir /home/kendra/retest-basic-00006/PolySpace_results/$bs
done
```

Splint_script.sh

```
#!/bin/sh
testdir=`pwd`
for file in `ls basic*.c`
do
    bs=`basename $file .c`
    /usr/bin/splint +bounds +orconstraint -paramuse $file >&
/home/kendra/Thesis/Splint_results/$bs
done
```

Uno_script.sh

```
#!/bin/sh
for file in `ls basic*.c`
do
    bs=`basename $file .c`
    /usr/bin/uno -CPP=/usr/bin/cpp -w $file >&
/home/kendra/Thesis/Uno_results/$bs
done
```

SQL Files

query_attrval_confusions/archer.sql

```
use thesis;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.writeread=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.writeread=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.writeread=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.writeread=1 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.whichbound=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.whichbound=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.whichbound=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.whichbound=1 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=3 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=4 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=5 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=6 and A.min=1 and A.ok=1;
```

```
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.memloc=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.memloc=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.memloc=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.memloc=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.memloc=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.memloc=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.memloc=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.memloc=3 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.memloc=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.memloc=4 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.scope=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.scope=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.scope=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.scope=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.scope=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.scope=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.scope=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.scope=3 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.container=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.container=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.container=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.container=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=3 and A.min=1 and A.ok=1;
```

```
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.container=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=4 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.container=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=5 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.pointer=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.pointer=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.pointer=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.pointer=1 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=3 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=4 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=5 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.indexcomplex=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=6 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.addrcomplex=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.addrcomplex=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=1 and A.min=1 and A.ok=1;
```

```
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.addrcomplex=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.addrcomplex=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=3 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.addrcomplex=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=4 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.addrcomplex=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=5 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.lencomplex=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.lencomplex=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.lencomplex=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.lencomplex=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=3 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.lencomplex=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=4 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.lencomplex=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=5 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.lencomplex=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=6 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.lencomplex=7;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=7 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.aliasaddr=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasaddr=0 and A.min=1 and A.ok=1;
```

```
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.aliasaddr=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasaddr=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.aliasaddr=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasaddr=2 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.aliasindex=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasindex=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.aliasindex=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasindex=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.aliasindex=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasindex=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.aliasindex=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasindex=3 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.localflow=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.localflow=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.localflow=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.localflow=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=3 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.localflow=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=4 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.localflow=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=5 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.localflow=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=6 and A.min=1 and A.ok=1;
```

```
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.localflow=7;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=7 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.secondaryflow=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.secondaryflow=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.secondaryflow=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.secondaryflow=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=3 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.secondaryflow=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=4 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.secondaryflow=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=5 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.secondaryflow=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=6 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.secondaryflow=7;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=7 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.loopstructure=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.loopstructure=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.loopstructure=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.loopstructure=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=3 and A.min=1 and A.ok=1;
```

```
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.loopstructure=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=4 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.loopstructure=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=5 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.loopstructure=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=6 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.loopstructure=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.loopstructure=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.loopstructure=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.loopstructure=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=3 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.loopstructure=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=4 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.asynchrony=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.asynchrony=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.asynchrony=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.asynchrony=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.asynchrony=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.asynchrony=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.asynchrony=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.asynchrony=3 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.taint=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=0 and A.min=1 and A.ok=1;
```

```
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.taint=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=1 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.taint=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=2 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.taint=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=3 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.taint=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=5 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.runtimeenvdep=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.runtimeenvdep=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.runtimeenvdep=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.runtimeenvdep=1 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.continuousdiscrete=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.continuousdiscrete=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.continuousdiscrete=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.continuousdiscrete=1 and A.min=1 and A.ok=1;

select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.signedness=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.signedness=0 and A.min=1 and A.ok=1;
select count(*) from testcases T,archer A where T.name=A.name and
A.min=1 and T.signedness=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.signedness=1 and A.min=1 and A.ok=1;
```

query_attrval_detections/archer.sql

```
use thesis;

select count(*) from testcases where writeread=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.writeread=0 and A.min=1;
select count(*) from testcases where writeread=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.writeread=1 and A.min=1;

select count(*) from testcases where whichbound=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.whichbound=0 and A.min=1;
select count(*) from testcases where whichbound=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.whichbound=1 and A.min=1;

select count(*) from testcases where datatype=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=0 and A.min=1;
select count(*) from testcases where datatype=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=1 and A.min=1;
select count(*) from testcases where datatype=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=2 and A.min=1;
select count(*) from testcases where datatype=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=3 and A.min=1;
select count(*) from testcases where datatype=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=4 and A.min=1;
select count(*) from testcases where datatype=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=5 and A.min=1;
select count(*) from testcases where datatype=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=6 and A.min=1;

select count(*) from testcases where memloc=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.memloc=0 and A.min=1;
select count(*) from testcases where memloc=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.memloc=1 and A.min=1;
select count(*) from testcases where memloc=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.memloc=2 and A.min=1;
select count(*) from testcases where memloc=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.memloc=3 and A.min=1;
select count(*) from testcases where memloc=4;
select count(*) from testcases T,archer A where
```

```
    T.name=A.name and T.memloc=4 and A.min=1;

select count(*) from testcases where scope=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.scope=0 and A.min=1;
select count(*) from testcases where scope=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.scope=1 and A.min=1;
select count(*) from testcases where scope=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.scope=2 and A.min=1;
select count(*) from testcases where scope=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.scope=3 and A.min=1;

select count(*) from testcases where container=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=0 and A.min=1;
select count(*) from testcases where container=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=1 and A.min=1;
select count(*) from testcases where container=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=2 and A.min=1;
select count(*) from testcases where container=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=3 and A.min=1;
select count(*) from testcases where container=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=4 and A.min=1;
select count(*) from testcases where container=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=5 and A.min=1;

select count(*) from testcases where pointer=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.pointer=0 and A.min=1;
select count(*) from testcases where pointer=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.pointer=1 and A.min=1;

select count(*) from testcases where indexcomplex=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=0 and A.min=1;
select count(*) from testcases where indexcomplex=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=1 and A.min=1;
select count(*) from testcases where indexcomplex=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=2 and A.min=1;
select count(*) from testcases where indexcomplex=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=3 and A.min=1;
select count(*) from testcases where indexcomplex=4;
select count(*) from testcases T,archer A where
```

```
    T.name=A.name and T.indexcomplex=4 and A.min=1;
select count(*) from testcases where indexcomplex=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=5 and A.min=1;
select count(*) from testcases where indexcomplex=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=6 and A.min=1;

select count(*) from testcases where addrcomplex=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=0 and A.min=1;
select count(*) from testcases where addrcomplex=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=1 and A.min=1;
select count(*) from testcases where addrcomplex=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=2 and A.min=1;
select count(*) from testcases where addrcomplex=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=3 and A.min=1;
select count(*) from testcases where addrcomplex=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=4 and A.min=1;
select count(*) from testcases where addrcomplex=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=5 and A.min=1;

select count(*) from testcases where lencomplex=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=0 and A.min=1;
select count(*) from testcases where lencomplex=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=1 and A.min=1;
select count(*) from testcases where lencomplex=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=2 and A.min=1;
select count(*) from testcases where lencomplex=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=3 and A.min=1;
select count(*) from testcases where lencomplex=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=4 and A.min=1;
select count(*) from testcases where lencomplex=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=5 and A.min=1;
select count(*) from testcases where lencomplex=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=6 and A.min=1;
select count(*) from testcases where lencomplex=7;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=7 and A.min=1;

select count(*) from testcases where aliasaddr=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasaddr=0 and A.min=1;
```

```
select count(*) from testcases where aliasaddr=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasaddr=1 and A.min=1;
select count(*) from testcases where aliasaddr=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasaddr=2 and A.min=1;

select count(*) from testcases where aliasindex=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasindex=0 and A.min=1;
select count(*) from testcases where aliasindex=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasindex=1 and A.min=1;
select count(*) from testcases where aliasindex=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasindex=2 and A.min=1;
select count(*) from testcases where aliasindex=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasindex=3 and A.min=1;

select count(*) from testcases where localflow=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=0 and A.min=1;
select count(*) from testcases where localflow=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=1 and A.min=1;
select count(*) from testcases where localflow=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=2 and A.min=1;
select count(*) from testcases where localflow=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=3 and A.min=1;
select count(*) from testcases where localflow=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=4 and A.min=1;
select count(*) from testcases where localflow=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=5 and A.min=1;
select count(*) from testcases where localflow=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=6 and A.min=1;
select count(*) from testcases where localflow=7;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=7 and A.min=1;

select count(*) from testcases where secondaryflow=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=0 and A.min=1;
select count(*) from testcases where secondaryflow=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=1 and A.min=1;
select count(*) from testcases where secondaryflow=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=2 and A.min=1;
select count(*) from testcases where secondaryflow=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=3 and A.min=1;
select count(*) from testcases where secondaryflow=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=4 and A.min=1;
select count(*) from testcases where secondaryflow=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=5 and A.min=1;
select count(*) from testcases where secondaryflow=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=6 and A.min=1;
select count(*) from testcases where secondaryflow=7;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=7 and A.min=1;

select count(*) from testcases where loopstructure=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=0 and A.min=1;
select count(*) from testcases where loopstructure=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=1 and A.min=1;
select count(*) from testcases where loopstructure=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=2 and A.min=1;
select count(*) from testcases where loopstructure=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=3 and A.min=1;
select count(*) from testcases where loopstructure=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=4 and A.min=1;
select count(*) from testcases where loopstructure=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=5 and A.min=1;
select count(*) from testcases where loopstructure=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=6 and A.min=1;

select count(*) from testcases where loopstructure=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=0 and A.min=1;
select count(*) from testcases where loopstructure=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=1 and A.min=1;
select count(*) from testcases where loopstructure=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=2 and A.min=1;
select count(*) from testcases where loopstructure=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=3 and A.min=1;
select count(*) from testcases where loopstructure=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=4 and A.min=1;

select count(*) from testcases where asynchrony=0;
select count(*) from testcases T,archer A where
```

```
    T.name=A.name and T.asynchrony=0 and A.min=1;
select count(*) from testcases where asynchrony=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.asynchrony=1 and A.min=1;
select count(*) from testcases where asynchrony=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.asynchrony=2 and A.min=1;
select count(*) from testcases where asynchrony=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.asynchrony=3 and A.min=1;

select count(*) from testcases where taint=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=0 and A.min=1;
select count(*) from testcases where taint=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=1 and A.min=1;
select count(*) from testcases where taint=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=2 and A.min=1;
select count(*) from testcases where taint=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=3 and A.min=1;
select count(*) from testcases where taint=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=5 and A.min=1;

select count(*) from testcases where runtimeenvdep=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.runtimeenvdep=0 and A.min=1;
select count(*) from testcases where runtimeenvdep=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.runtimeenvdep=1 and A.min=1;

select count(*) from testcases where continuousdiscrete=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.continuousdiscrete=0 and A.min=1;
select count(*) from testcases where continuousdiscrete=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.continuousdiscrete=1 and A.min=1;

select count(*) from testcases where signedness=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.signedness=0 and A.min=1;
select count(*) from testcases where signedness=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.signedness=1 and A.min=1;
```

query_attrval_falsealarms/archer.sql

```
use thesis;

select count(*) from testcases where writeread=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.writeread=0 and A.ok=1;
select count(*) from testcases where writeread=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.writeread=1 and A.ok=1;

select count(*) from testcases where whichbound=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.whichbound=0 and A.ok=1;
select count(*) from testcases where whichbound=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.whichbound=1 and A.ok=1;

select count(*) from testcases where datatype=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=0 and A.ok=1;
select count(*) from testcases where datatype=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=1 and A.ok=1;
select count(*) from testcases where datatype=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=2 and A.ok=1;
select count(*) from testcases where datatype=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=3 and A.ok=1;
select count(*) from testcases where datatype=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=4 and A.ok=1;
select count(*) from testcases where datatype=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=5 and A.ok=1;
select count(*) from testcases where datatype=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.datatype=6 and A.ok=1;

select count(*) from testcases where memloc=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.memloc=0 and A.ok=1;
select count(*) from testcases where memloc=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.memloc=1 and A.ok=1;
select count(*) from testcases where memloc=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.memloc=2 and A.ok=1;
select count(*) from testcases where memloc=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.memloc=3 and A.ok=1;
select count(*) from testcases where memloc=4;
select count(*) from testcases T,archer A where
```

```
    T.name=A.name and T.memloc=4 and A.ok=1;

select count(*) from testcases where scope=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.scope=0 and A.ok=1;
select count(*) from testcases where scope=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.scope=1 and A.ok=1;
select count(*) from testcases where scope=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.scope=2 and A.ok=1;
select count(*) from testcases where scope=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.scope=3 and A.ok=1;

select count(*) from testcases where container=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=0 and A.ok=1;
select count(*) from testcases where container=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=1 and A.ok=1;
select count(*) from testcases where container=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=2 and A.ok=1;
select count(*) from testcases where container=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=3 and A.ok=1;
select count(*) from testcases where container=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=4 and A.ok=1;
select count(*) from testcases where container=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.container=5 and A.ok=1;

select count(*) from testcases where pointer=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.pointer=0 and A.ok=1;
select count(*) from testcases where pointer=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.pointer=1 and A.ok=1;

select count(*) from testcases where indexcomplex=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=0 and A.ok=1;
select count(*) from testcases where indexcomplex=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=1 and A.ok=1;
select count(*) from testcases where indexcomplex=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=2 and A.ok=1;
select count(*) from testcases where indexcomplex=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=3 and A.ok=1;
select count(*) from testcases where indexcomplex=4;
select count(*) from testcases T,archer A where
```

```
    T.name=A.name and T.indexcomplex=4 and A.ok=1;
select count(*) from testcases where indexcomplex=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=5 and A.ok=1;
select count(*) from testcases where indexcomplex=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.indexcomplex=6 and A.ok=1;

select count(*) from testcases where addrcomplex=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=0 and A.ok=1;
select count(*) from testcases where addrcomplex=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=1 and A.ok=1;
select count(*) from testcases where addrcomplex=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=2 and A.ok=1;
select count(*) from testcases where addrcomplex=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=3 and A.ok=1;
select count(*) from testcases where addrcomplex=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=4 and A.ok=1;
select count(*) from testcases where addrcomplex=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.addrcomplex=5 and A.ok=1;

select count(*) from testcases where lencomplex=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=0 and A.ok=1;
select count(*) from testcases where lencomplex=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=1 and A.ok=1;
select count(*) from testcases where lencomplex=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=2 and A.ok=1;
select count(*) from testcases where lencomplex=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=3 and A.ok=1;
select count(*) from testcases where lencomplex=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=4 and A.ok=1;
select count(*) from testcases where lencomplex=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=5 and A.ok=1;
select count(*) from testcases where lencomplex=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=6 and A.ok=1;
select count(*) from testcases where lencomplex=7;
select count(*) from testcases T,archer A where
    T.name=A.name and T.lencomplex=7 and A.ok=1;

select count(*) from testcases where aliasaddr=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasaddr=0 and A.ok=1;
```

```
select count(*) from testcases where aliasaddr=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasaddr=1 and A.ok=1;
select count(*) from testcases where aliasaddr=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasaddr=2 and A.ok=1;

select count(*) from testcases where aliasindex=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasindex=0 and A.ok=1;
select count(*) from testcases where aliasindex=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasindex=1 and A.ok=1;
select count(*) from testcases where aliasindex=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasindex=2 and A.ok=1;
select count(*) from testcases where aliasindex=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.aliasindex=3 and A.ok=1;

select count(*) from testcases where localflow=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=0 and A.ok=1;
select count(*) from testcases where localflow=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=1 and A.ok=1;
select count(*) from testcases where localflow=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=2 and A.ok=1;
select count(*) from testcases where localflow=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=3 and A.ok=1;
select count(*) from testcases where localflow=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=4 and A.ok=1;
select count(*) from testcases where localflow=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=5 and A.ok=1;
select count(*) from testcases where localflow=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=6 and A.ok=1;
select count(*) from testcases where localflow=7;
select count(*) from testcases T,archer A where
    T.name=A.name and T.localflow=7 and A.ok=1;

select count(*) from testcases where secondaryflow=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=0 and A.ok=1;
select count(*) from testcases where secondaryflow=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=1 and A.ok=1;
select count(*) from testcases where secondaryflow=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=2 and A.ok=1;
select count(*) from testcases where secondaryflow=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=3 and A.ok=1;
select count(*) from testcases where secondaryflow=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=4 and A.ok=1;
select count(*) from testcases where secondaryflow=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=5 and A.ok=1;
select count(*) from testcases where secondaryflow=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=6 and A.ok=1;
select count(*) from testcases where secondaryflow=7;
select count(*) from testcases T,archer A where
    T.name=A.name and T.secondaryflow=7 and A.ok=1;

select count(*) from testcases where loopstructure=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=0 and A.ok=1;
select count(*) from testcases where loopstructure=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=1 and A.ok=1;
select count(*) from testcases where loopstructure=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=2 and A.ok=1;
select count(*) from testcases where loopstructure=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=3 and A.ok=1;
select count(*) from testcases where loopstructure=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=4 and A.ok=1;
select count(*) from testcases where loopstructure=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=5 and A.ok=1;
select count(*) from testcases where loopstructure=6;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=6 and A.ok=1;

select count(*) from testcases where loopstructure=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=0 and A.ok=1;
select count(*) from testcases where loopstructure=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=1 and A.ok=1;
select count(*) from testcases where loopstructure=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=2 and A.ok=1;
select count(*) from testcases where loopstructure=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=3 and A.ok=1;
select count(*) from testcases where loopstructure=4;
select count(*) from testcases T,archer A where
    T.name=A.name and T.loopstructure=4 and A.ok=1;

select count(*) from testcases where asynchrony=0;
select count(*) from testcases T,archer A where
```

```
    T.name=A.name and T.asynchrony=0 and A.ok=1;
select count(*) from testcases where asynchrony=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.asynchrony=1 and A.ok=1;
select count(*) from testcases where asynchrony=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.asynchrony=2 and A.ok=1;
select count(*) from testcases where asynchrony=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.asynchrony=3 and A.ok=1;

select count(*) from testcases where taint=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=0 and A.ok=1;
select count(*) from testcases where taint=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=1 and A.ok=1;
select count(*) from testcases where taint=2;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=2 and A.ok=1;
select count(*) from testcases where taint=3;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=3 and A.ok=1;
select count(*) from testcases where taint=5;
select count(*) from testcases T,archer A where
    T.name=A.name and T.taint=5 and A.ok=1;

select count(*) from testcases where runtimeenvdep=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.runtimeenvdep=0 and A.ok=1;
select count(*) from testcases where runtimeenvdep=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.runtimeenvdep=1 and A.ok=1;

select count(*) from testcases where continuousdiscrete=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.continuousdiscrete=0 and A.ok=1;
select count(*) from testcases where continuousdiscrete=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.continuousdiscrete=1 and A.ok=1;

select count(*) from testcases where signedness=0;
select count(*) from testcases T,archer A where
    T.name=A.name and T.signedness=0 and A.ok=1;
select count(*) from testcases where signedness=1;
select count(*) from testcases T,archer A where
    T.name=A.name and T.signedness=1 and A.ok=1;
```

query_minmedlarge_same/archer.sql

```
use thesis;
select count(*) from archer where (min=1 and (med=0 or large=0)) or
                                  (med=1 and (min=0 or large=0)) or
                                  (large=1 and (min=0 or med=0));
```

query_overall_confusion/archer.sql

```
use thesis;
select count(*) from archer where ok=1 and min=1;
```

query_overall_detection/archer.sql

```
use thesis;
select count(*) from archer where min=1;
```

query_overall_falsealarm/archer.sql

```
use thesis;
select count(*) from archer where ok=1;
```

setupDBtables.sql

```
-- Creates entire thesis DB schema for MySQL.
-- All tables will be empty.

DROP TABLE testcases;
CREATE TABLE testcases (
    name            VARCHAR(32),
    writeread       int,
    whichbound      int,
    datatype        int,
    memloc          int,
    scope           int,
    container       int,
    pointer         int,
    indexcomplex    int,
    addrcomplex     int,
    lencomplex      int,
    aliasaddr       int,
    aliasindex      int,
    localflow       int,
    secondaryflow   int,
    loopstructure   int,
    loopcomplex     int,
    asynchrony      int,
    taint           int,
    runtimeenvdep   int,
    continuousdiscrete  int,
    signedness      int
);

DROP TABLE archer;
CREATE TABLE archer (
    name            VARCHAR(32),
    ok              int,
    min             int,
    med             int,
    large           int
);

DROP TABLE boon;
CREATE TABLE boon (
    name            VARCHAR(32),
    ok              int,
    min             int,
    med             int,
    large           int
);

DROP TABLE polyspace;
CREATE TABLE polyspace (
    name            VARCHAR(32),
    ok              int,
    min             int,
    med             int,
    large           int
);

DROP TABLE splint;
CREATE TABLE splint (
    name            VARCHAR(32),
    ok              int,
    min             int,
    med             int,
    large           int
);

DROP TABLE uno;
CREATE TABLE uno (
    name            VARCHAR(32),
    ok              int,
    min             int,
    med             int,
    large           int
);

LOAD DATA INFILE 'c:/Documents and Settings/Kendra Kratkiewicz/My
Documents/Thesis/results/testcase_db_info.txt' INTO TABLE testcases;
LOAD DATA INFILE 'c:/Documents and Settings/Kendra Kratkiewicz/My
Documents/Thesis/results/archer_db_info.txt' INTO TABLE archer IGNORE 1
lines;
LOAD DATA INFILE 'c:/Documents and Settings/Kendra Kratkiewicz/My
Documents/Thesis/results/boon_db_info.txt' INTO TABLE boon IGNORE 1
lines;
LOAD DATA INFILE 'c:/Documents and Settings/Kendra Kratkiewicz/My
Documents/Thesis/results/polyspace_db_info.txt' INTO TABLE polyspace
IGNORE 1 lines;
LOAD DATA INFILE 'c:/Documents and Settings/Kendra Kratkiewicz/My
Documents/Thesis/results/splint_db_info.txt' INTO TABLE splint IGNORE 1
lines;
LOAD DATA INFILE 'c:/Documents and Settings/Kendra Kratkiewicz/My
Documents/Thesis/results/uno_db_info.txt' INTO TABLE uno IGNORE 1
lines;
```

Appendix 2 Basic Test Suite Coverage Statistics

```
Coverage Statistics:

WRITE/READ
      1160  write
       4    read

WHICH BOUND
      1160  upper
       4    lower

DATA TYPE
      1140  char
       4    int
       4    float
       4    wchar
       4    pointer
       4    unsigned int
       4    unsigned char

MEMORY LOCATION
      1140  stack
       4    heap
       8    data segment
       8    bss
       4    shared

SCOPE
      1076  same
      24    inter-procedural
       8    global
      56    inter-file/inter-proc
       0    inter-file/global

CONTAINER
      1096  no
       4    array
      28    struct
       8    union
      20    array of structs
       8    array of unions

POINTER
      1160  no
       4    yes

INDEX COMPLEXITY
      644   constant
      440   variable
       4    linear expr
       4    non-linear expr
       8    function return value
       4    array contents
      60    N/A

ADDRESS COMPLEXITY
      1144  constant
       4    variable
       4    linear expr
       4    non-linear expr
       4    function return value
       4    array contents

LENGTH COMPLEXITY
      1108  N/A
       8    none
      16    constant
      16    variable
       4    linear expr
       4    non-linear expr
       4    function return value
       4    array contents

ADDRESS ALIAS
      1080  none
      76    yes, one level
       8    yes, two levels

INDEX ALIAS
      1096  none
       4    yes, one level
       4    yes, two levels
      60    N/A

LOCAL CONTROL FLOW
      1116  none
      24    if
       4    switch
       4    cond
       4    goto/label
       4    longjmp
       4    function pointer
       4    recursion

SECONDARY CONTROL FLOW
      740   none
      400   if
       4    switch
       4    cond
       4    goto/label
       4    longjmp
       4    function pointer
       4    recursion
```

LOOP STRUCTURE

332	no
64	for
64	do-while
64	while
256	non-standard for
192	non-standard do-while
192	non-standard while

LOOP COMPLEXITY

332	N/A
104	none
312	one
312	two
104	three

ASYNCHRONY

1152	no
4	threads
4	forked process
4	signal handler

TAINT

1148	no
4	argc/argv
4	environment variable
4	file read
0	socket
4	process environment

RUNTIME ENV. DEPENDENCE

1148	no
16	yes

MAGNITUDE

291	no overflow
291	1 byte
291	8 bytes
291	4096 bytes

CONTINUOUS/DISCRETE

692	discrete
472	continuous

SIGNEDNESS

1148	no
16	yes

Appendix 3 Sample CRED Output

```
Bounds Checking GCC v gcc-3.3.2-3.2 Copyright (C) 1995 Richard W.M.
Jones
Bounds Checking comes with ABSOLUTELY NO WARRANTY. For details see file
`COPYING' that should have come with the source to this program.
Bounds Checking is free software, and you are welcome to redistribute
it
under certain conditions. See the file `COPYING' for details.
For more information, set GCC_BOUNDS_OPTS to `-help'
basic-00001-large.c:56:Bounds error: array reference (4105) outside
bounds of the array.
basic-00001-large.c:56:  Pointer value: 0xbffff8b9
basic-00001-large.c:56:  Object `buf':
basic-00001-large.c:56:    Address in memory:    0xbfffe8b0 ..
0xbfffe8b9
basic-00001-large.c:56:    Size:                 10 bytes
basic-00001-large.c:56:    Element size:         1 bytes
basic-00001-large.c:56:    Number of elements:   10
basic-00001-large.c:56:    Created at:           basic-00001-large.c,
line 52
basic-00001-large.c:56:    Storage class:        stack
```

Appendix 4 Sample Tool Output

ARCHER Output

basic-00001-min

```
/home/kendra/temptree
Retrieving input file
/home/kendra/temptree/home/kendra/ThesisTestSuite/basic-00001-min.c
Analyzing main
/home/kendra/ThesisTestSuite/basic-00001-min.c:56:main:
ERROR:BUFFER:56:56:Array bounds error (len < off+1) (buf[10], len = 10,
off = 10, max(len-off) = 0)
===============================
| CURRENT STATE OF THE SOLVER |
===============================

The Store
---------
buf: ary(ptr_1, 1)
argv: ptr(ptr_0, [sym_2])
argc: sym_0

The Bindings
------------

The Pointers
------------
ptr_0 = (sym_3, sym_1, ...)
ptr_1 = (sym_4, 10, 1000000)

Dependencies
------------
sym_0: { argc }
sym_2: { argv }
sym_4: { ptr_1 }
sym_1: { ptr_0 }
sym_3: { ptr_0 }
ptr_0: { argv }
ptr_1: { buf }
```

BOON Output

basic-00045-min

```
--- Parsing `/home/kendra/ThesisTestSuite/basic-00045-min.i'...
--- Alpha-converting `/home/kendra/ThesisTestSuite/basic-00045-
min.i'...
--- Memorizing prototypes...
--- Analyzing protos in `/home/kendra/ThesisTestSuite/basic-00045-
min.i'...
--- Generating constraints...
--- Analyzing code in `/home/kendra/ThesisTestSuite/basic-00045-
min.i'...
--- Solving constraints...

real  0m0.005s
user  0m0.000s
sys   0m0.000s

POSSIBLE VULNERABILITIES:
Almost certainly a buffer overflow in `buf@main()':
  10..10 bytes allocated, 11..11 bytes used.
  <- siz(buf@main())
  <- len(buf@main())


CAVEATS:
```

basic-00001-min/polyspace_RTE_View.txt

```
Procedural entities     R     O     Gy    Gn    Line  Col   %
      Details     Macro Code  Reviewed    Comment
-polyspace  1                             100         0
|  -basic-00001-min     1                 1           100
      basic-00001-min.c 0
|  |  -main 1                 50    4     100   basic-00001-min.c
      0
|  |  |  ! OBAI.0 1                 56    5           array index
within bounds     76
|  -__polyspace__stdstubs                       1
      __polyspace__stdstubs.c 0
|  |  _init_globals                       1
      0
|  |  -_polyspace_random_char                   127   12
      __polyspace__stdstubs.c 0
|  |  |  X UNP.0                                            6
|  |  -_polyspace_random_int                    139   11
      __polyspace__stdstubs.c 0
|  |  |  X UNP.0                                            6
|  |  -_polyspace_random_long                   145   12
      __polyspace__stdstubs.c 0
|  |  |  X UNP.0                                            6
|  |  -_polyspace_random_long_double                        4165
      12          __polyspace__stdstubs.c 0
|  |  |  X UNP.0                                            6
|  |  -_polyspace_random_uchar                        133   21
      __polyspace__stdstubs.c 0
|  |  |  X UNP.0                                            6
|  |  -_polyspace_random_ulong                        151   21
      __polyspace__stdstubs.c 0
|  |  |  X UNP.0                                            6

************************************************************
*
* LIST OF VERIFIER OPTIONS
* Warning, these options may modify this view.
*
************************************************************

-Selected View :  alpha level


** SOURCE PROGRAM **
-target :   i386
-OS-target :      linux
```

```
** SESSION IDENTIFICATION **
-verif-version :  1.0


** PRECISION LEVEL **
-from :     "scratch"
-to : "Software Safety Analysis level 4"
-O :  2
-permissive


** C SPECIFIC **
-I :  /work/tools/PolySpace/2.5/include/include-linux
```

Splint Output

basic-00001-min

```
Splint 3.0.1.7 --- 24 Jan 2003

Command Line: Setting +orconstraint redundant with current value
basic-00001-min.c: (in function main)
basic-00001-min.c:56:3: Possible out-of-bounds store:
    buf[10]
    Unable to resolve constraint:
    requires 9 >= 10
     needed to satisfy precondition:
    requires maxSet(buf @ basic-00001-min.c:56:3) >= 10
  A memory write may write to an address beyond the allocated buffer.
(Use
  -boundswrite to inhibit warning)

Finished checking --- 1 code warning
```

UNO Output

basic-00001-min

```
uno: basic-00001-min.c:  56 array index error 'buf[10]' (array-size:
10)
uno: in fct main, local variable never used (evaluated) 'argv'
     statement  : basic-00001-min.c:50: main(int argc,char *argv[])
     declaration: basic-00001-min.c:51: char *argv[];
uno: in fct main, local variable never used (evaluated) 'argc'
     statement  : basic-00001-min.c:50: main(int argc,char *argv[])
     declaration: basic-00001-min.c:51: int argc;
```

Appendix 5 Performance Results by Test Case

This appendix presents the tools' performance by test case, broken down into multiple tables. Each table contains the set of test cases constructed to embody all the possible values for a particular attribute. For instance, the first table shows the results for the two test cases in the Write/Read attribute set: one for the "write" value (which is also the baseline test case), and one for the "read" value. This is the only table that includes the baseline test case.

The tables show which tools missed detections and which tools reported false alarms for each test case in the table, indicated with an "X" at the intersection of the tool's column and the test case's row. The tool columns are labeled only with the first letters of the tool names: "A" for ARCHER, "B" for BOON, "P" for PolySpace, "S" for Splint, and "U" for UNO. The last column to the right describes, for each test case, which attributes are classified with non-baseline values.

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00001							X				
basic-00002				X			X				read

Table 29. Write/Read Test Case Results

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00003							X		X		lower

Table 30. Upper/Lower Bound Test Case Results

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00004							X				int
basic-00005							X				float
basic-00006							X				wchar
basic-00007							X				pointer
basic-00008							X				unsigned int
basic-00009							X				unsigned char

Table 31. Data Type Test Case Results

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00010							X			X	heap
basic-00011				X			X				data segment
basic-00012							X				bss
basic-00013			X	X		X	X			X	shared[1]

Table 32. Memory Location Test Case Results

[1] Additional values: inter-procedural

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00014				X		X	X			X	inter-procedural[1]
basic-00015							X				global[2]
basic-00016							X				global[3]

Table 33. Scope Test Case Results

[1] Additional values: aliasaddr:one
[2] Additional values: bss
[3] Additional values: data segment

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00017				X			X			X	array
basic-00018							X			X	struct
basic-00019							X			X	struct
basic-00020							X			X	struct
basic-00021							X			X	struct
basic-00022							X			X	struct
basic-00023							X			X	struct[1]
basic-00024							X			X	struct[1]
basic-00025							X			X	union
basic-00026							X			X	union
basic-00027				X			X			X	array of structs
basic-00028				X			X			X	array of structs
basic-00029				X			X			X	array of structs
basic-00030				X			X			X	array of structs
basic-00031				X			X			X	array of structs
basic-00032				X			X			X	array of unions
basic-00033				X			X			X	array of unions

Table 34. Container Test Case Results

[1] Additional values: index:variable

Test Case	False Alarms					Missed Detections					Non-basline attribute values
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00034							X			X	pointer[1]

Table 35. Pointer Test Case Results

[1] Additional values: index:N/A, aliasindex:N/A

Test Case	False Alarms					Missed Detections					Non-basline attribute values
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00035							X				index:variable
basic-00036				X			X			X	index:linear expr
basic-00037				X			X			X	index:non-linear expr
basic-00038				X		X	X			X	index:func ret val
basic-00039						X	X			X	index:array contents

Table 36. Index Complexity Test Case Results

Test Case	False Alarms					Missed Detections					Non-basline attribute values
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00040							X			X	addr:variable
basic-00041				X			X			X	addr:linear expr
basic-00042				X			X			X	addr:non-linear expr
basic-00043				X		X	X			X	addr:func ret val
basic-00044			X			X	X			X	addr:array contents

Table 37. Address Complexity Test Case Results

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00045			X			X				X	length:none[1] (strcpy)
basic-00046				X		X	X			X	length:none[1] (strcpy)
basic-00047						X	X			X	length:constant[1] (strncpy)
basic-00048						X	X			X	length:variable[1] (strncpy)
basic-00049				X		X	X			X	length:linear expr[1] (strncpy)
basic-00050				X		X	X			X	length:non-linear expr[1] (strncpy)
basic-00051				X		X	X			X	length:func ret val[1] (strncpy)
basic-00052						X	X			X	length:array contents[1] (strncpy)

Table 38. Length Complexity Test Case Results

[1] Additional values: inter-file/inter-proc, index:N/A, aliasaddr:one, aliasindex:N/A, continuous

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00053							X			X	aliasaddr:one
basic-00054				X		X	X			X	aliasaddr:one[1]
basic-00055							X			X	aliasaddr:two
basic-00056				X		X	X			X	aliasaddr:two[1]

Table 39. Alias of Buffer Address Test Case Results

[1] Additional values: inter-prodecural

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00057							X				aliasindex:one
basic-00058							X				aliasindex:two

Table 40. Alias of Index Test Case Results

Test Case	False Alarms					Missed Detections					Non-basline attribute values
	A	B	P	S	U	A	B	P	S	U	
basic-00059							X				localflow:if
basic-00060				X			X				localflow: switch
basic-00061							X		X		localflow:cond
basic-00062							X				localflow:goto
basic-00063							X				localflow: longjmp
basic-00064				X		X	X			X	localflow: funcptr[1]
basic-00065				X		X	X			X	localflow: recursion[1]

Table 41. Local Control Flow Test Case Results

[1] Additional values: inter-procedural, aliasaddr:one

Test Case	False Alarms					Missed Detections					Non-basline attribute values
	A	B	P	S	U	A	B	P	S	U	
basic-00066							X				2ndflow:if
basic-00067							X				2ndflow:switch
basic-00068				X			X			X	2ndflow:cond[1]
basic-00069							X				2ndflow:goto[2]
basic-00070							X				2ndflow:longjmp[2]
basic-00071				X		X	X			X	2ndflow:funcptr[1]
basic-00072				X		X	X			X	2ndflow:recursion

Table 42. Secondary Control Flow Test Case Results

[1] Additional values: index:variable
[2] Additional values: localflow:if

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00073							X				for[1]
basic-00074							X				do-while[1]
basic-00075							X				while[1]
basic-00076							X				non-standard for[1]
basic-00077							X		X		non-standard for[2]
basic-00078							X		X		non-standard for[1]
basic-00079							X		X		non-standard for[2]
basic-00080							X				non-standard do-while[1]
basic-00081							X				non-standard do-while[2]
basic-00082							X				non-standard do-while[2]
basic-00083							X				non-standard while[2]
basic-00084							X				non-standard while[1]
basic-00085							X				non-standard while[2]

Table 43. Loop Structure Test Case Results

[1] Additional values: loopcomplex:zero
[2] Additional values: 2ndflow:if, loopcomplex:zero

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00086							X				loopcomplex:one[1]
basic-00087							X				loopcomplex:one[2]
basic-00088							X				loopcomplex:one[3]
basic-00089							X				loopcomplex:one[4]
basic-00090							X		X		loopcomplex:one[5]
basic-00091							X		X		loopcomplex:one[4]
basic-00092							X		X		loopcomplex:one[5]
basic-00093							X				loopcomplex:one[6]
basic-00094							X				loopcomplex:one[7]
basic-00095							X				loopcomplex:one[7]
basic-00096							X				loopcomplex:one[8]
basic-00097							X				loopcomplex:one[9]
basic-00098							X				loopcomplex:one[8]
basic-00099							X				loopcomplex:one[1]
basic-00100							X				loopcomplex:one[2]
basic-00101							X				loopcomplex:one[3]
basic-00102							X				loopcomplex:one[4]
basic-00103							X		X		loopcomplex:one[5]
basic-00104							X		X		loopcomplex:one[4]
basic-00105							X		X		loopcomplex:one[5]
basic-00106							X				loopcomplex:one[6]
basic-00107							X				loopcomplex:one[7]
basic-00108							X				loopcomplex:one[7]
basic-00109							X				loopcomplex:one[8]
basic-00110							X				loopcomplex:one[9]
basic-00111							X				loopcomplex:one[8]
basic-00112							X				loopcomplex:one[1]
basic-00113							X				loopcomplex:one[2]
basic-00114							X				loopcomplex:one[3]
basic-00115							X				loopcomplex:one[4]
basic-00116							X		X		loopcomplex:one[5]
basic-00117							X		X		loopcomplex:one[4]
basic-00118							X		X		loopcomplex:one[5]
basic-00119							X				loopcomplex:one[6]
basic-00120							X				loopcomplex:one[7]
basic-00121							X				loopcomplex:one[7]
basic-00122							X				loopcomplex:one[8]
basic-00123							X				loopcomplex:one[9]
basic-00124							X				loopcomplex:one[8]
basic-00125							X				loopcomplex:two[1]
basic-00126							X				loopcomplex:two[2]

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00127							X				loopcomplex:two[3]
basic-00128							X				loopcomplex:two[4]
basic-00129							X		X		loopcomplex:two[5]
basic-00130							X		X		loopcomplex:two[4]
basic-00131							X		X		loopcomplex:two[5]
basic-00132							X				loopcomplex:two[6]
basic-00133							X				loopcomplex:two[7]
basic-00134							X				loopcomplex:two[7]
basic-00135							X				loopcomplex:two[8]
basic-00136							X				loopcomplex:two[9]
basic-00137							X				loopcomplex:two[8]
basic-00138							X				loopcomplex:two[1]
basic-00139							X				loopcomplex:two[2]
basic-00140							X				loopcomplex:two[3]
basic-00141							X				loopcomplex:two[4]
basic-00142							X		X		loopcomplex:two[5]
basic-00143							X		X		loopcomplex:two[4]
basic-00144							X		X		loopcomplex:two[5]
basic-00145							X				loopcomplex:two[6]
basic-00146							X				loopcomplex:two[7]
basic-00147							X				loopcomplex:two[7]
basic-00148							X				loopcomplex:two[8]
basic-00149							X				loopcomplex:two[9]
basic-00150							X				loopcomplex:two[8]
basic-00151							X				loopcomplex:two[1]
basic-00152							X				loopcomplex:two[2]
basic-00153							X				loopcomplex:two[3]
basic-00154							X				loopcomplex:two[4]
basic-00155							X		X		loopcomplex:two[5]
basic-00156							X		X		loopcomplex:two[4]
basic-00157							X		X		loopcomplex:two[5]
basic-00158							X				loopcomplex:two[6]
basic-00159							X				loopcomplex:two[7]
basic-00160							X				loopcomplex:two[7]
basic-00161							X				loopcomplex:two[8]
basic-00162							X				loopcomplex:two[9]
basic-00163							X				loopcomplex:two[8]
basic-00164							X				loopcomplex:three[1]
basic-00165							X				loopcomplex:three[2]
basic-00166							X				loopcomplex:three[3]
basic-00167							X				loopcomplex:three[4]

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00168							X		X		loopcomplex:three[5]
basic-00169							X		X		loopcomplex:three[4]
basic-00170							X		X		loopcomplex:three[5]
basic-00171							X				loopcomplex:three[6]
basic-00172							X				loopcomplex:three[7]
basic-00173							X				loopcomplex:three[7]
basic-00174							X				loopcomplex:three[8]
basic-00175							X				loopcomplex:three[9]
basic-00176							X				loopcomplex:three[8]

Table 44. Loop Complexity Test Case Results

[1] Additional values: for
[2] Additional values: do-while
[3] Additional values: while
[4] Additional values: non-standard for
[5] Additional values: 2ndflow:if, non-standard for
[6] Additional values: non-standard do-while
[7] Additional values: 2ndflow:if, non-standard do-while
[8] Additional values: 2ndflow:if, non-standard while
[9] Additional values: non-standard while

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00177							X				threads
basic-00178							X				forked process[1]
basic-00179							X	X			signal handler

Table 45. Asynchrony Test Case Results

[1] Additional values: localflow:if, 2ndflow:if

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00180			X	X		X	X			X	argc/argv[1]
basic-00181			X				X				env var[2]
basic-00182			X			X				X	file read[3] (fgets)
basic-00183			X			X	X		X	X	proc env[3] getcwd)

Table 46. Taint Test Case Results

[1] Additional values: index:func ret val, 2ndflow:if, runtime env dep
[2] Additional values: index:variable, 2ndflow:if, runtime env dep
[3] Additional values: inter-file/inter-proc, index:N/A, length:constant, aliasaddr:one, aliasindex:N/A, runtime env dep, continuous

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00184							X				continuous[1,6]
basic-00185							X		X		continuous[1,7]
basic-00186							X		X		continuous[1,8]
basic-00187							X		X		continuous[1,9]
basic-00188							X		X	X	continuous[1,5,9]
basic-00189							X		X	X	continuous[1,9]
basic-00190							X		X	X	continuous[1,5,9]
basic-00191							X		X	X	continuous[1,10]
basic-00192							X		X		continuous[1,5,10]
basic-00193							X		X	X	continuous[1,5,10]
basic-00194							X		X		continuous[1,5,11]
basic-00195							X		X	X	continuous[1,11]
basic-00196							X		X	X	continuous[1,5,11]
basic-00197							X			X	continuous[2,6]
basic-00198							X		X		continuous[2,7]
basic-00199							X		X	X	continuous[2,8]
basic-00200							X		X	X	continuous[2,9]
basic-00201							X		X	X	continuous[2,5,9]
basic-00202							X		X	X	continuous[2,9]
basic-00203							X		X	X	continuous[2,5,9]
basic-00204							X		X	X	continuous[2,10]
basic-00205							X		X		continuous[2,5,10]
basic-00206							X		X	X	continuous[2,5,10]
basic-00207							X		X		continuous[2,5,11]
basic-00208							X		X	X	continuous[2,11]
basic-00209							X		X	X	continuous[2,5,11]
basic-00210				X			X			X	continuous[2,6]
basic-00211							X		X	X	continuous[2,7]
basic-00212							X		X	X	continuous[2,8]
basic-00213							X		X	X	continuous[2,9]
basic-00214							X		X	X	continuous[2,5,9]
basic-00215							X		X	X	continuous[2,9]
basic-00216							X		X	X	continuous[2,5,9]
basic-00217							X		X	X	continuous[2,10]
basic-00218							X		X	X	continuous[2,5,10]
basic-00219							X		X	X	continuous[2,5,10]
basic-00220							X		X	X	continuous[2,5,11]
basic-00221							X		X	X	continuous[2,11]
basic-00222							X		X	X	continuous[2,5,11]
basic-00223							X				continuous[2,6]
basic-00224							X		X		continuous[2,7]

Test Case	False Alarms					Missed Detections					Non-basline attribute values
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00225							X		X		continuous[2,8]
basic-00226							X		X		continuous[2,9]
basic-00227							X		X	X	continuous[2,5,9]
basic-00228							X		X	X	continuous[2,9]
basic-00229							X		X	X	continuous[2,5,9]
basic-00230							X		X	X	continuous[2,10]
basic-00231							X		X		continuous[2,5,10]
basic-00232							X		X	X	continuous[2,5,10]
basic-00233							X		X		continuous[2,5,11]
basic-00234							X		X	X	continuous[2,11]
basic-00235							X		X	X	continuous[2,5,11]
basic-00236				X			X			X	continuous[3,6]
basic-00237							X		X	X	continuous[3,7]
basic-00238							X		X	X	continuous[3,8]
basic-00239							X		X	X	continuous[3,9]
basic-00240							X		X	X	continuous[3,5,9]
basic-00241							X		X	X	continuous[3,9]
basic-00242							X		X	X	continuous[3,5,9]
basic-00243							X		X	X	continuous[3,10]
basic-00244							X		X	X	continuous[3,5,10]
basic-00245							X		X	X	continuous[3,5,10]
basic-00246							X		X	X	continuous[3,5,11]
basic-00247							X		X	X	continuous[3,11]
basic-00248							X		X	X	continuous[3,5,11]
basic-00249							X			X	continuous[3,6]
basic-00250							X		X		continuous[3,7]
basic-00251							X		X	X	continuous[3,8]
basic-00252							X		X	X	continuous[3,9]
basic-00253							X		X	X	continuous[3,5,9]
basic-00254							X		X	X	continuous[3,9]
basic-00255							X		X	X	continuous[3,5,9]
basic-00256							X		X	X	continuous[3,10]
basic-00257							X		X		continuous[3,5,10]
basic-00258							X		X	X	continuous[3,5,10]
basic-00259							X		X		continuous[3,5,11]
basic-00260							X		X	X	continuous[3,11]
basic-00261							X		X	X	continuous[3,5,11]
basic-00262				X			X			X	continuous[3,6]
basic-00263							X		X	X	continuous[3,7]
basic-00264							X		X	X	continuous[3,8]
basic-00265							X		X	X	continuous[3,9]

Test Case	False Alarms					Missed Detections					Non-basline attribute values
	A	B	P	S	U	A	B	P	S	U	
basic-00266							X		X	X	continuous[3,5,9]
basic-00267							X		X	X	continuous[3,9]
basic-00268							X		X	X	continuous[3,5,9]
basic-00269							X		X	X	continuous[3,10]
basic-00270							X		X	X	continuous[3,5,10]
basic-00271							X		X	X	continuous[3,5,10]
basic-00272							X		X	X	continuous[3,5,11]
basic-00273							X		X	X	continuous[3,11]
basic-00274							X		X	X	continuous[3,5,11]
basic-00275				X			X			X	continuous[4,6]
basic-00276							X		X	X	continuous[4,7]
basic-00277							X		X	X	continuous[4,8]
basic-00278							X		X	X	continuous[4,9]
basic-00279							X		X	X	continuous[4,5,9]
basic-00280							X		X	X	continuous[4,9]
basic-00281							X		X	X	continuous[4,5,9]
basic-00282							X		X	X	continuous[4,10]
basic-00283							X		X	X	continuous[4,5,10]
basic-00284							X		X	X	continuous[4,5,10]
basic-00285							X		X	X	continuous[4,5,11]
basic-00286							X		X	X	continuous[4,11]
basic-00287							X		X	X	continuous[4,5,11]

Table 47. Continuous/Discrete Test Case Results

[1] Additional values: index:variable, loopcomplex:zero
[2] Additional values: index:variable, loopcomplex:one
[3] Additional values: index:variable, loopcomplex:two
[4] Additional values: index:variable, loopcomplex:three
[5] Additional values: 2ndflow:if
[6] Additional values: for
[7] Additional values: do-while
[8] Additional values: while
[9] Additional values: non-standard for
[10] Additional values: non-standard do-while
[11] Additional values: non-standard while

Test Case	**False Alarms**					**Missed Detections**					**Non-basline attribute values**
	A	**B**	**P**	**S**	**U**	**A**	**B**	**P**	**S**	**U**	
basic-00288						X	X		X	X	sign[1] (memcpy)
basic-00289						X	X		X	X	sign[2] (memcpy)
basic-00290						X	X		X	X	sign[3] (memcpy)
basic-00291						X	X		X	X	sign[3] (memcpy)

Table 48. Signed/Unsigned Mismatch Test Case Results

[1] Additional values: inter-file/inter-proc, index:N/A, length:constant, aliasaddr:one, aliasindex:N/A, continuous
[2] Additional values: inter-file/inter-proc, index:N/A, length:variable, aliasaddr:one, aliasindex:N/A, continuous,
[3] Additional values: inter-file/inter-proc, index:N/A, length:variable, aliasaddr:one, aliasindex:N/A, localflow:if, continuous