REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE November 16, 1993

3. REPORT TYPE AND DATES COVERED Final Report: 24 Sep 1992-24 Sep 1994

4. TITLE AND SUBTITLE
Ada Support for The Mathematical Foundations of Software Engineering

5. FUNDING NUMBERS

6. AUTHOR(S)
John Beidler

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Computing Sciences Department
University of Scranton
Scranton, PA 18510

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U. S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

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

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Three topics were selected as targets for this project, program correctness, finite state devices, and program complexity (timing). A variety of artifacts were developed to support course material, programming assignments, and laboratory assignments in the mathematics of software engineering. The program correctness artifact centers on an artifact, called Assert. Assert is an Ada package that assists users in testing program assertions. The finite state device target is supported by several artifacts. One artifact in a course module, with laboratory and programming assignments, that centers on the use of finite state device concepts in programming and the classical representations of finite state devices in Ada. The second finite state device artifact is a Turing Machine simulator that simulates a turing machine with from one to three tapes. The timing target centered on generalizations of the classical Towers of Hanoi problem. The traditional Towers of Hanoi problem appears in many computing texts as a recursion example. Our study of the Towers of Hanoi problem led to the observation that there is no formal proof for the Towers of Hanoi problem when more than three spindles are used. This problem lends itself to substantial experimentation among the students as they compete to develop the program with the best timing results. All software developed through this grant has been forwarded for inclusion in the Public Ada Library (PAL).

14. SUBJECT TERMS
Program correctness, assertion testing, finite state automata, turing machines, Towers of Hanoi, computational complexity, program timing

15. NUMBER OF PAGES 39

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Final Report

Research Agreement No: DAAL03-92-G-0410

Ada Support for The Mathematical Foundations of Software Engineering

John Beidler
Prof. of Computing Sciences
University of Scranton
Scranton, PA 18510

beidler@cs.uofs.edu
(717) 941-7446 (Voice)
(717) 941-4250 (FAX)

Table of Contents

1 Executive Summary

Three topics were selected as targets for this project, program correctness, finite state devices, and program complexity (timing). A variety of artifacts were developed to support course material, programming assignments, and laboratory assignments in the mathematics of software engineering. Most of these artifacts have been placed in the PAL, Public Ada Library. A few more artifacts will be set to the PAL after they have been classroom tested.

The program correctness artifact centers on an artifact, called Assert. Assert is an Ada package that assists users in testing program assertions. The finite state device target is supported by several artifacts. One artifact in a course module, with laboratory and programming assignments, that centers on the use of finite state device concepts in programming and the classical representations of finite state devices in Ada.

The second finite state device artifact is a Turing Machine simulator, called tm. tm simulates a turing machine with from one to three tapes with a visual representation on a typical text screen. The simulator requires a VT100 compatible terminal (that includes any PC running ANSI.SYS, window systems using an X11 xterm window). tm was designed using an object oriented approach, hence the artifacts support three types of usage. Besides the typical turing machine building assignments, the may be used to demonstrate object oriented design and the turing simulator may be as a programming project, by withholding several package bodies and requiring the students to build the various packagê bodies.

The timing target centered on generalizations of the classical Towers of Hanoi problem. The traditional Towers of Hanoi problem appears in many computing texts as a recursion example. Our study of the Towers of Hanoi problem led to several surprises. First, the "well known" timing solution for the traditional three spindle version was formally proven only in 1981! Also, there are no formal proofs for the Towers of Hanoi with four or more spindles. There do exist estimates that appeared in 1941 as solutions to a Problems Section entry that appeared in 1939. However, none of the solutions contained formal proofs. This problem lends itself to substantial experimentation among the students as they compete to develop the program with the best timing results.

All software developed through this grant has been forwarded for inclusion in the Public Ada Library (PAL).

2 Overview

It was not enough to simply develop course materials directed at the mathematical foundations of software engineering, it was important to develop materials that could be integrated into existing courses without disrupting or displacing existing course materials. One way of achieving this is to develop material that builds upon topics that are already in the curriculum. Four topics that are frequently found in two core courses were select. The courses were the second course in computer science and the data structures and algorithm course, frequently referred to as the ACM Curriculum courses, CS 2 and CS 7. These courses were selected because all computing curriculum either contain these two courses, or cover these topics in other core courses.

Initially, the project concentrated on three topics, program correctness, finite state devices, and program timing. It was desirable to approach each topic from a new perspective that will not only interest the students, but interest the instructors as well. For example, consider the topic of program correctness. This is usually approached from a theoretical and very mathematical point of view. The approach take in this project was from a pragmatic point of view, using assertions to locate and correct errors. This point of view demonstrates that program correctness can play an important role throughout the entire software life cycle, including system evaluation and maintenance.

3 Assertions

Proving the correctness of a program is frequently viewed by software developers as an esoteric academic exercise. This point of view can be readily appreciated when one reviews the various presentation on program correctness that appears in current text books. Frequently, illustrations of applications of program correctness concepts are after-the-fact exercises of belaboring the obvious. In some cases errors exist in the "proofs". Frequently authors avoid, or provide a poor presentation, on one of the major tools of low level program correctness proofs, loop invariants. The current lack of utilization of program correctness techniques is unfortunate because program correctness techniques can be an invaluable software development aids that may be employed throughout the software development process, from the design phase through software maintenance and modification.

This paper describes our experience designing an Ada package that supports practical uses of program correctness throughout the software development process. The package, called ASSERT, was originally designed as a stand alone package to supports the pragmatic use of program correctness with its major concentration on the interface between the design and implementation phases of software projects. An early version of the **Assert** package was designed and developed by Jennifer Pollack, a senior Computer Science major at the University of Scranton.

She began the research for this project in the Spring of her Junior year and spent the summer reviewing the literature on program correctness. Because of her preparation, she had a prototype completed for her Senior project at the beginning of the Fall 1991 semester. This allowed us to concentrate on the issues surrounding methods of encouraging potential clients to use the ASSERT package. Section Two describes the **Assert** package and its various reporting modes. Section Three illustrates a typical use of the **Assert** package.

Section Four presents several pragmatic issues surrounding the use of the ASSERT package. The current version of the package was is slightly different from our original version of system. Initially, we found attempts to use the Assert package to be inconvenient. This is illustrated in this paper with several examples of assertions involving classical array based algorithms. Initially, the problem of testing assertions frequently doubled the amount of code that was written. That is, the code to perform assertion testing of a system was almost equal in size to the amount of code in the original system. The reason for the additional code was that we wanted assertion testing to be written in code that is independent from the original code for the obvious reason (same code = same error).

As we began to understand how to design good iterators, see [*Bei92a*], [*Bei92b*], [*Bei93*], the Assert package became much easier to use. With good iterators we found it easy to build assertion tests for a variety of homogeneous data structures. This will be demonstrated in the article with array iterators. With just two types of array iterators as tools for building assertions, we found we could easily build most assertion tests for many classical array based algorithms.

Just as our frustration was beginning to peak, this project was assisted by another project that involved the construction of a repertoire of Ada software components. Each of our software component packages contained appropriate iterators for the various components. Iterators became an invaluable tool for the construction of algorithms to test assertions about components. With this in hand we went back to the problems we had constructing assertions for array algorithms. First we constructed a simple package of array iterators, then used these iterators to construct assertions. The result was a dramatic reduction in the amount of code written to perform assertion testing.

The final section summarizes the results contained in this paper with some practical observation and plans for future spin-offs from this project. The practical observation is that the use of iterators in assertion testing provides a type of "proof-reading" approach to proving the correctness of a program. If two proof readers come to the same conclusions about a piece of text, the text is assumed correct. The noted mathematician, George Polya, wrote an article, see [*Pol76*], about the mathematics of proofreading. In the future we plan to analyze the use of the Assert package in light of Polya's article. By a proof-reading proof we mean that if two independent pieces of code produce the same answer there is a higher degree of confidence that the code is correct, especially if the two pieces of code are truly independent. To some extent, the use of independently written iterators to construct assertion tests provides a reliable and cost effective means of testing assertions.

Once we started using iterators to build tests for assertions, we found that it may be convenient to build a repertoire of packages of small pieces of code to support a collections of typical assertions about the structures of various components. We see the new nested library scheme in Ada-9X as a desirable framework for the packaging of assertion testing tools. We plan to proceed with this project when we gain access to a Ada-9X compiler. With Ada-9X's ability to nest packages, we believe we can use this feature to build collections of reusable assertion testing tools and components.

3.1 The `Assert` Package

The **ASSERT** package is quite simple in design. The specifications for the **ASSERT** package appear in Figure 1. A natural assumption made regarding the use of the **ASSERT** package is that clients understand the basics of program correctness. The four assertion testing procedures, Precondition, Postcondition, Invariant, and Assertion are essentially identical. The only difference between them is a unique prefix placed by the procedure in front of the output requested by the client. The implication is that the client uses the appropriately named procedure when requesting an assertion test and uses identical Prefix strings for procedures that correspond to the same structure. For example, given the statement sequence,

```
Sum := 0;
for i in 1 .. n loop
   Sum := Sum + A(i) ;
end loop ;
```

```
package Assert is

procedure Precondition ( Condition     : boolean ;
                         Prefix        : String ;
                         True_Message  : String ;
                         False_Message : String ) ;
   -----------------------------------------------------
   --  Pre-cond  : None
   --  Post-cond : if Condition then display True_Message
   --                 else display False_Message ;
   --              unless limited by current operating
   --              mode
   -----------------------------------------------------

procedure Postcondition ( Condition     : boolean ;
                          Prefix        : String ;
                          True_Message  : String ;
                          False_Message : String ) ;
   -----------------------------------------------------
   --  Pre-cond  : None
   --  Post-cond : if Condition then display True_Message
   --                 else display False_Message ;
   --              unless limited by current operating
   --              mode
   -----------------------------------------------------

procedure Invariant ( Condition     : boolean ;
                      Prefix        : String ;
                      True_Message  : String ;
                      False_Message : String ) ;
   -----------------------------------------------------
   --  Pre-cond  : None
   --  Post-cond : if Condition then display True_Message
   --                 else display False_Message ;
   --              unless limited by current operating
   --              mode
   -----------------------------------------------------

procedure Assertion ( Condition     : boolean ;
                      Prefix        : String ;
                      True_Message  : String ;
                      False_Message : String ) ;
   -----------------------------------------------------
   --  Pre-cond  : None
   --  Post-cond : if Condition then display True_Message
   --                 else display False_Message ;
   --              unless limited by current operating
   --              mode
   -----------------------------------------------------

Procedure Beta_Mode ;
   ------------------------------------------------------------
   --  Pre-cond  : None
   --  Post-cond : Place package in Alpha_Mode
   --  Comment   : In Beta_Mode if the package is turned "On",
   --              only the False_Message is printed when the
   --              condition being tested by an assertion
                          procedure fails..
   ------------------------------------------------------------

Procedure Alpha_Mode ;
   -----------------------------------------------------------
   --  Pre-cond  : None
   --  Post-cond : Place package in Alpha_Mode
   --  Comment   : In Alpha_Mode if the package is turned "On",
   --              either the True_Message of the False_Message
   --              is printed as each assertion test procedure
                     is called.
   -----------------------------------------------------------

Procedure Off ;
   -----------------------------------------------------------
   --  Pre-cond  : None
   --  Post-cond : Displaying of assertions is terminated
   -----------------------------------------------------------

Procedure On ;
   -----------------------------------------------------------
   --  Pre-cond  : None
   --  Post-cond : Displaying of assertions continued depending
   --              upon current package mode
   -----------------------------------------------------------

end Assert ;
```

Figure 1 ASSERT Specifications

may have the corresponding assertions,

```
Sum := 0;
ASSERT.Precondition ( Sum = Sum_OF (A, 0),
                "Sum loop", "start", "Failed");
for i in 1 .. n loop
   Sum := Sum + A(i) ;
   ASSERT.Invariant ( Sum = Sum_OF (A, i),
          "Sum loop", integer'image(i), "Failed");
end loop ;
ASSERT.Postcondition ( Sum = Sum_OF (A, n),
                      "Sum loop", "end", "Failed");
```

where the prefix, "Sum loop" appears in each line of the display. The display would appear as,

```
PRE : Sum loop start
INV : Sum loop  1
INV : Sum loop  2
INV : Sum loop  3
INV : Sum loop  4
INV : Sum loop  5
   . . .
POST: Sum loop end
```

The prefixes "**PRE :** ", "**INV :** ", "**POST:** ", and "**ASRT:** " are placed at the beginning of the line displayed by the procedure Precondition, Invariant, Postcondition, and Assertion, respectively. Combining the assertion prefixes with the client provided prefixes creates a display format that is easy to interpret. For example the nested pair of loops in the statement sequence,

```
Sum := 0 ;
ASSERT.Precondition (Sum = Answer (0),
                "Outer", "start", "Start fail") ;
for outer in 1 .. 10 loop
   ASSERT.Precondition (Sum = Answer (Outer-1),
        "    Inner", "nest begin", "nest  fail") ;
  ·for inner in 1 ,, outer loop
      Sum := Sum + 1 ;
      ASSERT.Invariant (Sum = (Answer (Outer
                            -1)+Inner), "   Inner",
         integer'image(inner), "nest  fail") ;
   end loop ;
   ASSERT.Postcondition (Sum = Answer (Outer),
                "    Inner",
                "nest end" & integer'image(outer),
                "nest  fail") ;
   ASSERT.Invariant (Sum = Answer (Outer),
                "Outer",
                integer'image(outer),
                "fail" & integer'image(outer) ) ;
end loop ;
ASSERT.Postcondition (Sum = Answer (10),
                "Outer", "start", "Start fail") ;
```

have assertions that produce the display,

```
PRE : Outer start
PRE :     Inner nest begin
INV :     Inner  1
POST:     Inner  next end 1
INV : Outer  1
PRE :     Inner nest begin
INV :     Inner  1
INV :     Inner  2
POST:     Inner  next end 1
INV : Outer  2
    . . .
```

Because of the potential verbosity of the display, the package has two global modes and two display modes. The global modes are **on** and **off**. The display modes are referred to as **Alpha** mode and **Beta** mode. The package displays assertion messages only when the global mode is **on**. No messages are displayed when the global mode is **off**. When the global mode is **on**, if the display mode is **Alpha**, every time an assertion testing procedure is called a message is displayed. In **Beta** mode messages are displayed only when a test fails. By selectively using the global **on/off** modes with the display **Alpha/Beta** modes clients may control the verbosity of the display.

The usefulness of the package depends somewhat on the cleverness of a client in performing meaningful assertion tests along with useful and distinct messages. A future modification of the package will be the inclusion of a **Silent** display mode. In **Silent** mode all assertion messages are placed in a file, which may be viewed at a later time.

3.2 Package Applications

Building an assertion testing package is easy, making it useful is another story. To illustrate, consider building the assertions to test a simple algorithm, like the bubble sort illustrated in Figure 2. A set of assertions to test the looping conditions for the bubble sort appears in Figure 3. Fortunately, the tests for all the assertions may be created using a single function, `Is_Sorted`, which also appears in Figure 3.

```
procedure Bubble_Sort (A : in out Int_Array) is

   temp, bub : integer ;

   begin
   tio.Put_Line ("Start sort") ;
   tio.Put_Line ("first precond") ;
   for index in 1 .. A'range'last-1 loop
      bub := index ;
      while (bub > 0)
            and then (A (bub) > A (bub+1)) loop
         temp       := A (bub);
         A (bub)    := A (bub+1) ;
         A (bub+1)  := temp ;
         bub        := bub - 1;
      end loop ;
   end loop;
   end Bubble_Sort ;
```

Figure 2 Bubble Sort Algorithm

Figure 4 contains a partial listing of sample output from the assertions in Figure 3. This simple example demonstrates one of two problems associated with using the **ASSERT** package, its propensity for

```
function Is_Sorted
   (A       : Int_Array;
    Start,
    Finish : integer ) return boolean is

   begin -- Is_Sorted
   for index in (Start+1) .. Finish loop
      if A(index-1) > A(Index) then
         return false ;
      end if ;
   end loop ;
   return true ;
   end Is_Sorted ;

procedure Bubble_Sort (A : in out Int_Array) is

   temp, bub : integer ;

   begin
   tio.Put_Line ("Start sort") ;
   ASSERT.PreCondition ( Is_Sorted (A, 1, 1),
                "Outer", "Start", "Start error");
   tio.Put_Line ("first precond") ;
   for index in 1 .. A'range'last-1 loop
      bub := index ;
      ASSERT.Precondition (Is_Sorted (A, 1, Index),
                "   Inner", "Start", "Start error");
      while (bub > 0)
            and then (A (bub) > A (bub+1)) loop
         temp      := A (bub);
         A (bub)   := A (bub+1) ;
         A (bub+1) := temp ;
         bub       := bub - 1;
         ASSERT.Invariant
            ( Is_Sorted (A, bub+1, Index+1),
             "   Inner",
             integer'image(bub+1),
             "invariant error");
      end loop ;
      ASSERT.Postcondition
         ( Is_Sorted (A, 1, Index+1),
          "   Inner", "Finish", "Finish error");
      ASSERT.Invariant ( Is_Sorted (A, 1, Index+1),
           "Outer", "OK", integer'image(index) );
   end loop;
   ASSERT.Postcondition
      ( Is_Sorted (A, 1, A'Range'last),
       "Outer", "OK", "end error" );
   end Bubble_Sort ;
```

Figure 3 Bubble Sort with Assertions

producing enormous amounts of output. If the procedure in Figure 3 was sorting five hundred numbers, the assertion tests would produce approximately 250,000 lines of output.

By selectively using the package's **On/Off** switch and the **Alpha/Beta** display modes a client can dramatically reduce the amount of output produced by the package. Since the major concern centers on assertion failures, the **Beta** display mode is normally the primary interest of clients. In **Beta** mode, only assertion failures produce output, a system that is mostly correct would produce very little output, and the output that is produced would be the output that is of most interest to clients.

Figure 4 Sample Output for Bubble Sort Assertions

The second fundamental problem with using the Assert package is the general problem of creating assertions. For the best possible results, assertion tests should be developed independently of the package being tested. There are sound formal reasons for the independent development of assertions. The formal reasons are addressed in Section ?. Informally, it is desirable for the independent development of assertion tests so that the code in the assertion tests is as distinct as possible from the code in the program. The basis for the assertion tests in Figure 3 is at least a function whose code is independent of the code in the procedure. A more desirable situation would be to have as much code as possible pre-written, which leads us to Section ?.

3.3 Iterators and Assertions

Many times, the coding effort involved in building assertion tests is potentially as large as the effort required to build the system being tested. This would explain why assertion testing is not a popular method of testing the correctness of programs. This difficulty may be overcome with the right software development tools. One family of tools that we have found to be very useful is the family of iterators over homogeneous data structures.

Frequently, assertions center on verifying relationships that hold regarding the contents of homogeneous structures. Often, algorithms to test assertions may be accomplished through the traversal of a structure while performing simple comparison tests. The traversal may be constructed with a predefined iterator. Fortunately, in our environment the packages for all homogeneous data structures include collections of the typical iterators over the structure. For example, binary tree packages include should include breadth first, depth first, and other typical iterators. In addition, we have a standard array tool package, The specifications for the array iterators appear in Figure 5.

When it comes to developing assertions, there are two advantages in using iterators when they are available and appropriate. The first advantage is that iterators can dramatically reduce the amount of code written to test assertions. A second advantage with iterators is that the traversal code for the structure exists, and may be presumed to be correct, hence increasing the probability that the assertion test is valid.

The selector version of the step and bisection array iterators, serve as the basis for many array traversal based assertions. To illustrate, consider the coding for the assertion testing function, `Is_Sorted`, in Figure 3. It is composed of a loop to traverse the part of the array being tested and an `if-else` structure that performs the actual test. Figure 6 illustrates a version of `Is_Sorted` built with the use of an iterator. In the iterator based version the client only writes a procedure, `Check_One`, to perform a single test and instantiates the iterator with that test.

The use of iterators actually serves two purposes. Beside the obvious benefit of reducing the amount of code written to perform the assertion tests, basing the assertion tests on iterators makes the code that performs the assertion tests dramatically different than the source code being tested. This difference helps provide some degree of independence between the system's code and the assertion testing code. That

```
package Array_Tools is

------------------------------------------------------------
--  This package contains three sets of array tools
--
--  Iterators
--    Bisection Constructor and Selector
--    Step Constructor and Selector
--
--  Searching Tools
--    Bisection Method Search
--    Sequential Search
--
--  Sorting tools
--    Bubble Sort
--    Shell Sort
--    Quick Sort
------------------------------------------------------------

type Bisect_Control_Type is (quit, lower, higher) ;

Zero_Increment : exception ; -- see Step_Iterator

generic
   type Object_Type is limited private ;
   type Array_Range is (<>) ;
   type Array_Type is array (Array_Range range <>) of Object_Type;
   type Pass_Thru_Type is limited private ;
   with Procedure Process ( A       : in out Array_Type ;
                            Index   : in     Array_Range ;
                            Control :    out Bisect_Control_Type ;
                            Pass    : in out Pass_Thru_Type ) ;
procedure Array_Bisect_Constructor
   ( A    : in out Array_Type ;
     Last :    out Array_Range ;
     Pass : in out Pass_Thru_Type ) ;

generic
   type Object_Type is limited private ;
   type Array_Range is (<>) ;
   type Array_Type is array (Array_Range range <>) of Object_Type;
   type Pass_Thru_Type is limited private ;
   with Procedure Process ( A        : in out Array_Type ;
                            Index    : in     Array_Range ;
                            Continue :    out boolean ;
                            Pass     : in out Pass_Thru_Type ) ;
procedure Array_Step_Constructor
   ( A         : in out Array_Type ;
     Start     : in     Array_Range ;
     Increment : in     integer := 1 ;
     Pass      : in out Pass_Thru_Type ) ;
   ------------------------------------------------------
   -- Exception : Zero_Increment if Increment = 0
   ------------------------------------------------------
     . . .
```

Figure 5 Array_Tool Package Specifications (Partial)

independence helps alleviate some of the concerns that the assertion testing code may be tainted by the system's code. That is, the group writing the assertion testing code may have a tendency to echo the system's code while writing assertion tests. As a result the assertion tests could contain the same errors that appear in the actual code. By writing assertion tests in a different way, building them on top of iterators, there is greater certainty that errors existing in the system's code are not echoed in the assertions.

4 Estimating The Number of Undiscovered Error

After testing a software system how many undiscovered errors still remain? After a certain amount of testing, locating, and correcting of errors can the testing information be used to form a mathematically sound estimate of the number of undiscovered errors? This paper describes a statistical framework for making estimates about the number of errors that may still remain in a software system as well as providing measures of the quality of the testing process. This paper also suggests several methodologies that may help to improve the accuracy of the results obtained when using the techniques described in this paper.

```
function Is_Sorted
   (A       : Int_Array;
    Start,
    Finish : integer ) return boolean is

   Answer : boolean := true ;

   procedure Check_One
      (A        : In      Int_Array ;
       Index    : in      positive ;
       Continue :     out boolean ;
       Answer   : in out boolean ) is

      begin -- Check_One
      if A(index-1) <= A(Index) then
           Continue := true ;
        else
           Continue := false ;
           Answer   := false ;
      end if ;
      end Check_One ;

   procedure Check_Range is new
      Array_Tools.Array_Step_Selector
         ( Object_Type    => integer ,
           Array_Range    => positive ,
           Array_Type     => Int_Array ,
           Pass_Thru_Type => boolean ,
           Process        => Check_One ) ;

   begin -- Is_Sorted
   Check_Range (A, Start+1, 1, Finish, Answer) ;
   return Answer ;
   end Is_Sorted ;
```

Figure 6 Is_Sorted Using An Iterator

Let X be the number of errors that exist in a software system. Two teams, Able and Baker, independently evaluate the system. As a result of their independent evaluation Team Able found A errors and Team Baker found B errors. Let C represent the count of the number of errors located by both teams. C is included in A and B. Let p be the probability that Team Able locates any given error and q be the probability that Team Baker locates any given error. p and q represent independent probabilities.

It is reasonable to expect that the number of errors located by Team Able, A, approximates pX, the number of errors Team Able is expected to find. Similarly, B approximates qX, the number of errors located by Team Baker, and C approximates pqX, the number of errors located by both teams. This leads to the following estimate for X the number of errors in the system,

$$X = \frac{pX \cdot qX}{pqX} \sim \frac{A \cdot B}{C}$$

The total number of discovered errors is $A + B - C$. An estimate of the number of undiscovered errors is

$$X - (A + B - C) \sim \frac{A \cdot B}{C} - (A + B - C)$$

$$= \frac{AB - AC - BC + C^2}{C}$$

$$= \frac{(A-C)(B-C)}{C}$$

Estimates for the values p and q are of interest, they produce measures of the quality of work performed by each team. The measure of the quality of the work performed by Team Able is

$$p = \frac{pqX}{qX} \sim \frac{C}{B}$$

The measure of the quality of work performed by Team Baker is

$$q = \frac{pqX}{pX} \sim \frac{C}{A}$$

To illustrate uses of these estimates, consider the independent evaluation of a software system by two teams, Able and Baker, that produces the values 30, 35, and 25 for ***A***, ***B***, and ***C***, respectively. The estimate for the total number of errors is 42. The number of errors discovered by the two teams is 30+35-25 = 40. The estimate for the number of undiscovered errors is 2! Finally, the measures for the quality of work performed by teams Able and Baker are 25/35 = 71% and 25/30 = 83%, respectively.

As a second example, suppose the independent evaluation of a software system by two teams produces the values 30, 35, and 15 for ***A***, ***B***, and ***C***, respectively. These values produce an estimate of 70 for the total number of errors in the system. Since 30+35-15 = 50 errors were discovered, and the estimate for the number of undiscovered errors is 20. The quality measures of the two teams evaluating the software are 15/35 = 43% and 15/30 = 50%.

This technique is based on work by Polya, see [*Pol76*], on the probabilities of the number of undiscovered errors in a text after being proofread independently by two proofreaders. In that paper Polya uses probabilities to determine an estimate of the number of undiscovered mistakes that exist in a manuscript after the manuscript has been independently evaluated by two proofreaders. To what extent can this approach be applied to software? At which points could this technique be applied during the software system life cycle? There are two considerations that must be addressed for Polya's "proofreading approach" to apply to the evaluation of software systems. One is the relationships that exist between problem domain of the software system and the domain of use of the system by the groups beta testing the system. The second issue is addressing whether the requirements of the mathematical foundation are met in the testing process. These issues are addressed in the next two sections.

4.1 Domains

Applying probabilities to proofreading is relatively easy when compared to applying the same approach to estimating the number of errors that exist in a software system. Specifically, in the case of proofreading the work domain of the two proofreaders is identical, the same manuscript. With large complex software systems the problem is complicated by a collection of domains. The software system functions over the domain of the problem space for which it was intended. Beta testers might only test a subdomain of the software's problem domain, that part of the problem domain that is of interest to them. For example, assume the software system was a spreadsheet system. Most users to not use all of the features of a spreadsheet, they concentrate on those features that are useful to them in their problem domain, which is a subdomain of the spreadsheet's capability. Two different beta testers may be testing overlapping, but generally unrelated subdomains of the software system's domain, see Figure 7.

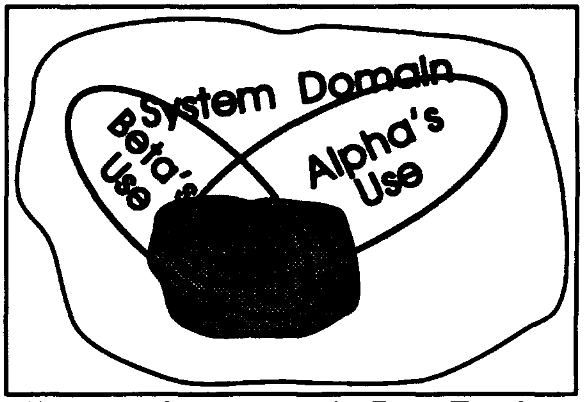

Figure 7 Incompatible Beta Testing Domains

Assume two beta testers are testing a software system and there tests are represented by Figure 7. If the software system developer has confidence in the work performed by the beta tester, then the estimates p and q helps measure the relationships between the subdomains of the beta testers. If $p = q = 0$, then the subdomains of the beta testers do not overlap. If $p < q = 1$, then the subdomain of beta tester Able is contained within the subdomain of beta tester Baker. The closer p and q are to one, The closer the subdomains of the two beta testers. Figure 8 illustrates an example where two beta testers are testing very similar domains.

The values p and q must be used carefully. On one hand, if a software developer expects a beta tester to perform a good job, the values p and q could provide a measure of the relationships of domains between beta testers.

On the other hand, if a software developer is confident that two beta testers are testing similar, or identical, subdomains, as illustrated in Figure 8, the values p and q may be good measures of the quality of work performed by the two beta testers. In this case,

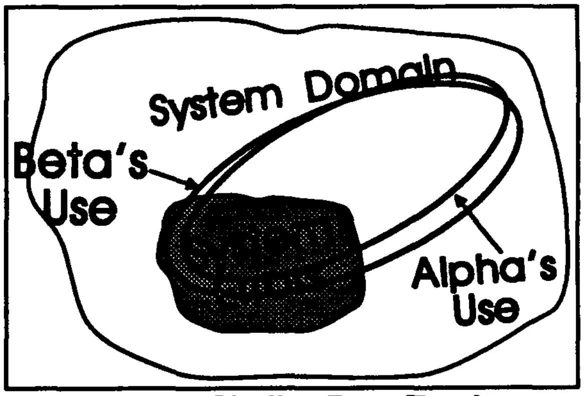

Figure 8 Similar Beta Testing Domains

$$\frac{(A - C)(B - C)}{C}$$

is an estimate of the number of errors that remain undetected in the in the beta tested subdomain of the system.

4.2 Independence

First of all, the results are estimates and must be carefully analyzed before the results may be interpreted. Whether the statistical foundation of these calculations is sound depend heavily on the evaluations of the software system being determined by the two teams working in a truly independent fashion. The statistical computations are based upon the probabilities of independent events. One might question whether the ability of the team to determine one error is independent of their ability to locate any other error. Assume that error i may be located with probability p_i. Then p is

$$p = \frac{\sum_{i \in X} p_i}{n(X)},$$

where $n(X)$ is the number of errors.

Intuitively, the more errors found in a system, the lower the confidence we should have in the software system. On the other hand, each time an error is located and corrected, without introducing a new error, the software system is closer to being correct. The obvious question is: Just how close?

The approach advocated in this paper suggests a methodology for obtaining an estimate on the number of errors that remain in the system and evaluations for the quality of testing being performed. Since the approach produces statistical estimates, there are a variety of factor that can adversely effect the estimates. Two factors that can effect the results and a way of producing multiple estimates are described below.

The results described above depend upon the probability of occurrence of independent events. There are two ways in which independence, or the lack there of, may effect the estimates produced. First, it is important that the two individuals, or groups, performing the testing do it in an independent fashion. If there is any direct, or indirect, communications between the two testing groups, the estimates are invalid. The second assumption, over which we have no control, is that the probability of a particular testing group of finding any given error is independent of locating any other error. One could argue that once a certain type of error is recognized in a software system, a tester might devise a process of locating similar errors, hence invalidating the independent probability of locating errors.

4.3 Beta Testing Equivalent Domains

It should be noted that the validity of the estimate of the number of undiscovered errors is valid only when the beta tester test the exact same subdomains of the problem. At first that may appear to be an impossibility. However, if assertion testing is placed into each program where the results of the assertion tests are reported to a file, the file may be analyzed relative to the beta test results provided by a tester. Now the errors indicated by the assertion tests may be measured relative to the errors reported by the beta tester. the assertion tests are performed on the identical domain as the domain of the beta tester, hence the estimates of the undiscovered errors within that subdomain could be fairly accurate.

4.4 Multiple Estimates

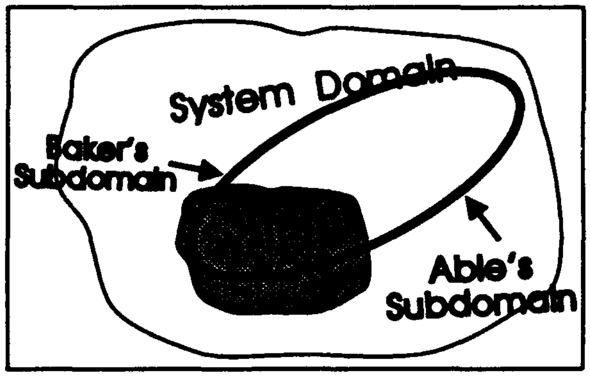

Figure 9 Identical Subdomains

When more than two beta testers is are used a collection of estimates may be formed. Assume the beta testers are testing in the same subdomain of the problem domain and there are X errors in that subdomain. Assume the three beta testers, γ, δ, and ϵ locate errors in a software system with probabilities p, q, and r, respectively. If A is the number of errors located by γ, B is the number of errors located by δ, and C the number of errors located by ϵ. Let D be the number of errors located by both γ and δ, E be the number of errors located by both δ and ϵ, F be the number of error located by both γ and ϵ, and G be the number of errors located by all three. then

$$A \sim pX,\ B \sim qX,\ C \sim rX,\ D \sim pqX,\ E \sim qrX,\ F \sim prX, \text{ and } G \sim pqrX.$$

There are five estimates that can be made for X,

$$\frac{A \cdot B \cdot C \cdot G}{D \cdot E \cdot F},\ \frac{A \cdot B}{D},\ \frac{B \cdot C}{E},\ \frac{A \cdot C}{F}, \text{ and } \frac{D \cdot E \cdot F}{G^2}.$$

and three estimates each for p, q, and r. The estimates for p are $\frac{D}{B}$, $\frac{G}{E}$, and $\frac{F}{C}$. Similar estimates may be obtained for q and r.

With three or more beta testers one may use the results to produce multiple estimates. These results may be further analyzed for consistency between the beta testers. Our experience indicates that with multiple beta testers, one can first use the results of the initial error analysis to eliminate results that are inconsistent to remove certain results from further consideration, then apply the approach described above to the remaining results.

5 Finite State Automata

It is desirable to introduce finite state automata early in the computing curriculum. Not only is it an important theoretical topic, but it is a fundamental design topic. In design, it is used to provide broad high level design characteristics of systems. It also plays an important role in object oriented design, where state transitions describe the state and change in states of objects in various object classes.

We introduced finite state automatons (FSAs) as a theoretical topic and made extensive use of CASE tools as a means of drawing state transition diagrams. We found three excellent CASE tools that freshman found easy to use. They are Open Select, Rationale ROSE, and Weilan's LeCASE. Along with the CASE tools we also discussed methods of implementing FSAs in programming languages. We concentrated on two schema for implementing FSAs, one using a state transition table and the second as a function.

Besides the two methods mentioned above, we also used a simulator, described in Section 6, to simulate FSAs. One laboratory assignment and one software development assignment were given to the class. The laboratory assignment used the simulator to build a machine that recognized regular expressions. For the laboratory assignment, the student were given a relatively simple task, like build a simulator that recognized all strings of zeros and one where the number of zeros was divisible by five.

The software development assignment requires the use of the FSA simulator, building a graphic representation using one of the CASE tools, and implementing the FSA in Ada and testing the implementation.

The FSA simulator is actually a special case of the Turing Machine Simulator described in the next section.

6 Turing Machine Simulator

We introduced turing machines in the CS 2 course, the second semester freshman computing course. Turing machines were introduced as part of a software development example. The introduction was done with an object oriented flavor by first discussing the component classes that make up a turing machine simulator: Tape class, Finite State Class, Transition controlled. The transition controller obtained information from the other objects, tape objects and finite state object, determined the next transition, then sent a message to each object informing it of its change in state (status).

6.1 Overview

Besides the description of a turing machine and its operations, the problem discussed the issues of user interfaces were discussed. The program presents a turing machine simulation by illustrating the movement of the tape across the screen and indicating the state transitions as they occur. The simulator had user definable controls. A user has the following options:

a. Single stepping through a simulation.
b. Speeding up the single step option with an ability to perform a fixed number of steps before halting the simulation.
c. Going to continuous simulation mode. Once in this mode, the simulator may not be halted.

The simulator also includes a step counter. The user may define a step limit. The limit is used to automatically terminate the simulator when the limit is exceeded.

The simulator terminates when either when the step counter exceeds the limit, or when no transition exists for the given state/symbol pair.

6.2 Construction

The simulator is put together as a collection of interrelated packages. The system is composed of three packages and a driving procedure,

a. The tm_tape package
b. The tm_state package
c. The tm_machine package
d. The tm procedure

The machine is entirely encapsulate in the tm_machine package. As such, the turing machine simulator may be used with other programs that may require the use of a turing machine to perform part of its task.

The tm procedure obtains the name of the file that contains the machine being simulated, calls the machine simulator, and passes to it the name of the file and the simulation parameters.

The tm_tape and tm_state package completely encapsulate tapes and states in an object oriented fashion. That is, states and tapes are entirely encapsulated within each package, including the information each tape and state must know to display themselves.

The tm package instantiates the tm_machine with the type of machine desires, a finite state automaton, or a one, two, or three tape turing machine. In turn, The tm_machine instantiates the tm_state package and the tm_tape package. In the case of finite state automatons, we made a special version of tm_machine that restricts the tm_tape to a one-way read-only tape. and simplifies the user description of finite state automata.

6.3 Use

Users define turing machines by completing turing tables. The tables are placed in an ASCII file, which is prepared before running the simulator. To assist users in defining turing machines, the simulator accepted ASCII files with one state transition per record. There are four record formats:

a. Comment records begin with a "--"
b. The first record in a sequence of records associated with a state simply contains the state name,

    old_state

c. The transitions for a specific state follow the record with the state name, one per record, with the format,

    current_symbol next_state new_symbol head_movement

d. A blank record terminates the transitions for a state.

The *old_state* and *next_state* were strings of up to eight characters in length. The current symbol and *new_symbol* were any printable ASCII character, except '*' which has a special meaning. The *head_movements* are '<', '-', or '>', which indicate that the read write head moved left, remains stationary, or moves right, respectively. The symbol '*' is a "DON'T CARE" indicator. That is, when it appears in the place of the *current_state* or *current_symbol* it means the value of this object may be anything. In the *next_state* or *new_symbol* position it means keep the current value. Since the state transition table is read from top to bottom, DON'T CARE indicators should appear after other state transitions that would override them.

```
-- A two tape turing machine
-- That duplicates the string of
-- zeros and ones bracketed by
-- dollar symbols on to the 2nd tape
start
  -- Start copies the first $
  $  -  mr   $  >  $  >

mr
  -- This state copies the rest of the
  -- symbols to the second $
  0  -  mr   0  > 0 >
  1  -  mr   1  > 1 >
  $  -  HALT $  - $ -
```

Figure 10 Turing Table Example

The system also allows users to place comments in the state transition tables. Comments may be placed in line, after a state transition, or are indicated by beginning a line with "--".

7 Generalized Towers of Hanoi

The classical *Towers of Hanoi* Problem, see Figure 11, is a game involving n disks and three spindles. The diameter of each disk is unique. The object of the game is to move the stack of n disks from the spindle containing the disks to a specified target spindle. The disks must be moved one at a time by removing any disk from the top of a stack on one spindle to another spindle. A disk may be placed on another spindle only if the spindle is empty or if the disks on the spindle are larger than the disk being moved. This problem is employed as an example in a number of mathematics and computing courses to demonstrate recursion or algorithm measurement. A fairly complete and traditional presentation on the *Towers of Hanoi* appears in [Knu??].

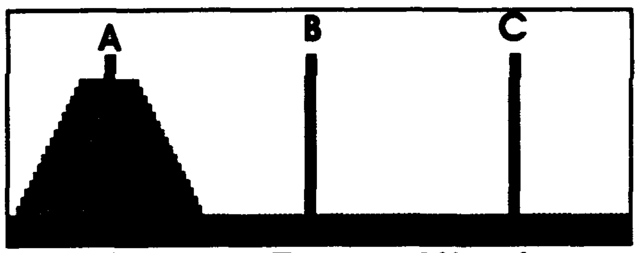

Figure 11 Towers of Hanoi

The *Towers of Hanoi* problem is frequently used in computing courses as a problem whose solution involves a non-trivial use of recursion. For n > 1 the algorithm for moving n disks is described as a recursive three step process, as illustrated in Figure 12:

Step 1: Recursively apply this algorithm to move n-1 disks from spindle A to spindle B using spindle C to assist in the process.
Step 2: Move the one disk on spindle A to spindle C.
Step 3: Recursively apply this algorithm to move the n-1 disks from spindle B to spindle C using spindle A to assist in the process.

Implementations of this algorithm in recursive programming language appear in a number of programming language and CS 1 texts, including [], [], []. Figure 13 contains a version of the solution written as a procedure in Ada. This procedure uses a screen display package that visually displays the disk movements as they are made.

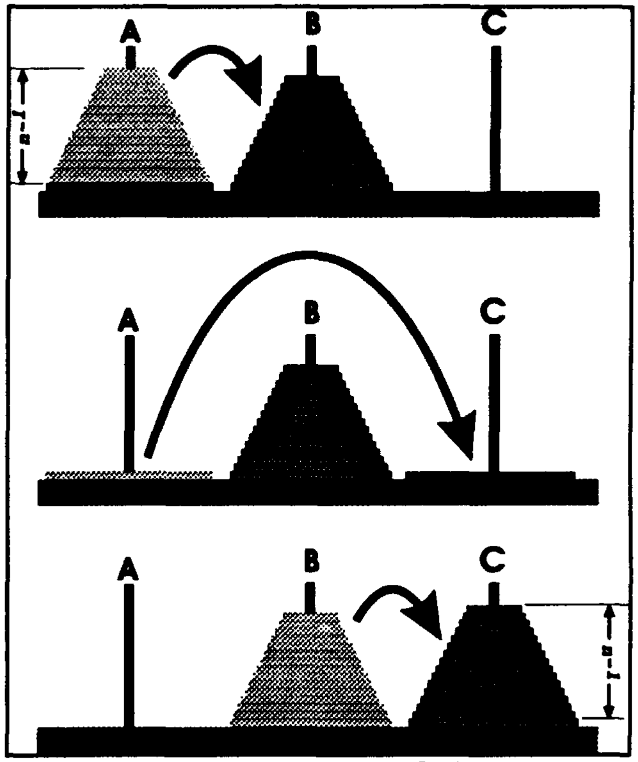

Figure 12 Recursive Solution to the Towers of Hanoi

```
procedure towers_of_hanoi
    ( from_spindle,
      help_spindle,
      to_spindle          : in
```

Figure 13 Sample Towers of Hanoi Program

Let H(n) be the minimum number of moves required to solve the *Towers of Hanoi* problem with n disks. Using induction it can be shown that $H(n) = 2^n-1$.

There are two obvious variations of the *Towers of Hanoi* problem, suggested in [Knu??]. One variation is that the disks are not all different, several disks may be identical. A second variation is to solve the problem with more than three spindles. A four spindle version of the *Towers of Hanoi* problem is used by several of our computing faculty as a software development assignment to test students' knowledge of recursion. We refer to the four spindle version of the *Towers of Hanoi* problem as the *Towers of Saigon*.

7.1 The Towers of Saigon

Independently, two faculty had used the Towers of Saigon as a programming assignment. In both cases, students were required not just to construct a correct program but to evaluate their programs. The programs written by students produced a large variety of timing results. Several faculty analyzed these results. This led to an analysis of the various solution strategies implemented by the students.

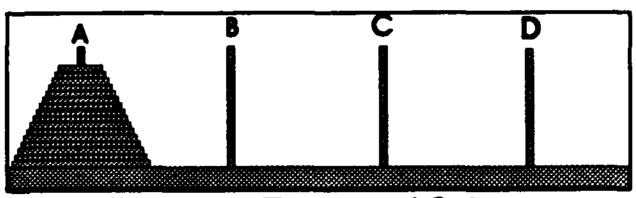

Figure 14 Towers of Saigon

Students employed Two basic solution strategies. These strategies evolved from specific suggestions made by the two faculty. We refer to these two solution strategies as the *n-2* strategy and the *split* strategy. One faculty member's suggestions led to the *n-2* strategy, the other faculty member's suggestions led to the *split* strategy. Figure 15 illustrates the *n-2* recursive strategy. In the *n-2* strategy a tower of *n* disks, $n > 2$, is moved in five steps:

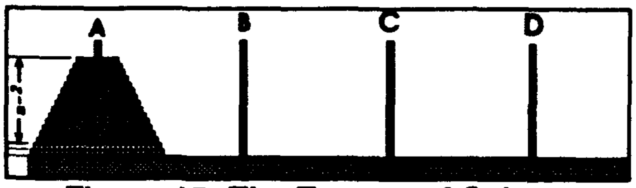

Figure 15 The Towers of Saigon n-2 Solution Strategy

Step 1: Recursively use this five step algorithm to move *n-2* disks to spindle B.
Step 2: Move one disk from spindle A to spindle C.
Step 3: Move the last disk from spindle A to spindle D.
Step 4: Move the disk on spindle C to spindle D.
Step 5: Recursively apply this five step algorithm to move the *n-2* disks on spindle B to spindle D.

When $n = 1$, the single disk may be move to the appropriate spindle. When $n = 2$, the two disks may be moved to the appropriate spindle in three moves.

The second strategy, the split strategy, makes specific use of the Towers of Hanoi (3 spindle) algorithm. With split strategy a number *k*, dependent on *n*, is selected. This strategy employs a three step process:

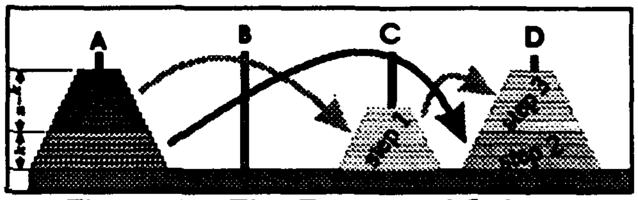

Figure 16 The Towers of Saigon Split Solution Strategy

Step 1': Recursively apply the split algorithm to move *n-k* disks from spindle A to spindle C.
Step 2': Apply the Towers of Hanoi solution to move the *k* disks from spindle A to spindle D using spindle B. Note that spindle C cannot be used because the disks on spindle C are smaller than the disks being moved during this step.
Step 3': Recursively apply the split algorithm to move the *n-k* disks from spindle C to spindle D.

Most students applied the split strategy by choosing $k = pn$ for some p, $0 <= p < 1$. Typical values selected for *p* were *1/2*, *1/3*, and *1/4*. Regardless of the choice made for *p*, for large values of n the split strategy clearly out performed the *n-2* strategy.

Several faculty began experimenting with the split strategy using various functions, $k=f(n)$, for selecting *k*. Independently, two faculty found the choice of $k = \sqrt{n}$ to be substantially better than other functions that were attempted. This lead to an interest in determining the best possible split strategy solution, or possibly, the best solution for all possible strategies.

Observe that the Split Strategy encapsulates all possible strategies. For example, the *n-2* strategy is an example of the Split Strategy with $k = 2$. Regardless of the strategy that one might adopt to solve the Towers of Saigon, that strategy must included the building of a tower of size *k* on an intermediate spindle while the remaining disks are moved to the target spindle using the remaining three spindles.

7.2 A Minimum Move Strategy for the Towers of Saigon

Let *H(n)* be the minimum number of moves required to solve the Towers of Hanoi problem with *n* disks. Let *S(n)* be the minimum number of moves required to solve the Towers of Saigon problem with *n* disks. The solution for moving *n* disks in the Towers of Saigon may be viewed as finding the optimum split location, *k*, so that *n-k* disks are moved using a four spindle algorithm to one of the two assisting spindles, then moving the lower *k* disks using a three spindle (Towers of Hanoi, *H(k)*) algorithm to the final spindle, and then moving the *n-k* disks to the final spindle using a four spindle algorithm. It is well known that the minimum number of moved for the three spindle Towers of Hanoi problem with *k* disks is 2^k-1. Clearly,

$$
\begin{aligned}
S(n) &= 0, \text{ for } n = 0, \\
&= 1, \text{ for } n = 1, \\
&= 2\,S(n-k) + H(k), \text{ for some } k,\ 0 < k < n, \text{ otherwise.}
\end{aligned}
$$

We wish to determine a formula for *k* in terms of *n* that will determine S(n), call it g(n), i.e., our best choice is k = g(n). Let $\Delta S(n) = S(n) - S(n-1)$. We wish to minimize $\Delta S(n)$. Observe that

$$\Delta S(n) = 2\,S(n-g(n)) + 2^{g(n)} - 1 - (2\,S(n-1-g(n-1)) + 2^{g(n-1)} - 1).$$

That is,

$$\Delta S(n) = 2\,[S\,(n-g(n)) - S\,(n-1-g(n-1))] + 2^{g(n)} - 2^{g(n-1)}.$$

If g(n) = g(n-1) then

$$\Delta S(n) = 2\,[S\,(n-g(n)) - S\,(n-1-g(n))].$$

If g(n) = g(n-1) + 1 then

$$\Delta S(n) = 2\,(S\,(n-[g(n-1)+1]\,) - S\,(n-1-g(n-1))) + 2^{g(n-1)+1} - 2^{g(n-1)} = 2^{g(n-1)}.$$

It is clear that to minimize $\Delta S(n)$, we wish to keep g(n) = g(n-1) until

$$2[S(n-g(n)) - S(n-1-g(n-1))] > 2^{g(n-1)}.$$

Hence g(1) = g(2)=1, g(3) = g(4) = g(5) = g(1) + 1, g(6) = g(7) = g(8) = g(9) = g(5) + 1, and so forth. That is, g(n) remains fixed one time more in each subsequent subsequence of values of g(n). That is, g(n-1) is the greatest integer such that

$$\sum_{i=1}^{g(n-1)} i \le n.$$

Thus g(n-1) satisfies $\frac{g(n-1)(g(n-1)+1)}{2} \le n$, and by the quadratic formula,

$$g(n-1) = \left\lfloor \frac{-1+(1+8n)^{1/2}}{2} \right\rfloor .$$

The sequence $\Delta S(n)$, s = 1, 2, 3, ..., is the sequence

$$\{a_n\} = 1, 2, 2, 4, 4, 4, 8, 8, 8, 8, \ldots, \underbrace{2^{i-1}, \ldots, 2^{i-1}}_{i}, \ldots$$

and

$$a_n = 2^{\left\lfloor \frac{-1+(1+8n)^{1/2}}{2} \right\rfloor} .$$

7.3 Observation

According to a tale, a group of monks made a deal with God. He would solve a 64 disk version of the Towers of Hanoi at the rate of one move a second before He would destroy the world. At that rate the world would be destroyed in 584,542,046,091 years. If the monks would have made the same deal with God using 64 disks and 4 spindles, the world would have come to an end in 19.3 days! What a difference a spindle makes.

7.4 Acknowledgements

I would like to acknowledge roles played by Professors Paul Jackowitz, Robert McCloskey (both of the Computing Sciences Department) and Prof. Steve Dougherty (of the Math. Dept.) for their continued interest and work on Split Strategy for the *Towers of Saigon* problem.

8 Software

All the software that was developed through this grant has been placed in the PAL (Public Ada Library). Portions of the software have also been processed with static logical analyzers and modified to conform with these analyzers. For completeness, the listing of the packages and systems developed through this grant are included below.

8.1 ASSERT

```
package Assert is

procedure Precondition ( Condition     : boolean ;
                         Prefix        : String ;
                         True_Message  : String ;
                         False_Message : String ) ;
    ---------------------------------------------------------
    -- Pre-cond  : None
    -- Post-cond : if Condition then display True_Message
    --                 else display False_Message ;
    --             unless limited by current operating
    --             mode
    ---------------------------------------------------------

procedure Postcondition ( Condition     : boolean ;
                          Prefix        : String ;
                          True_Message  : String ;
                          False_Message : String ) ;
    ---------------------------------------------------------
    -- Pre-cond  : None
    -- Post-cond : if Condition then display True_Message
    --                 else display False_Message ;
    --             unless limited by current operating
    --             mode
    ---------------------------------------------------------

procedure Invariant ( Condition     : boolean ;
                      Prefix        : String ;
                      True_Message  : String ;
                      False_Message : String ) ;
    ---------------------------------------------------------
    -- Pre-cond  : None
    -- Post-cond : if Condition then display True_Message
    --                 else display False_Message ;
    --             unless limited by current operating
    --             mode
    ---------------------------------------------------------

procedure Assertion ( Condition     : boolean ;
                      Prefix        : String ;
                      True_Message  : String ;
                      False_Message : String ) ;
    ---------------------------------------------------------
    -- Pre-cond  : None
    -- Post-cond : if Condition then display True_Message
    --                 else display False_Message ;
    --             unless limited by current operating
    --             mode
    ---------------------------------------------------------

Procedure Beta_Mode ;
    ---------------------------------------------------------
    -- Pre-cond  : None
    -- Post-cond : Place package in Alpha_Mode
    -- Comment   : In Beta_Mode if the package is turned "On",
    --             only the False_Message is printed when the
    --             condition being tested by an assertion
    --             procedure fails..
    ---------------------------------------------------------

Procedure Alpha_Mode ;
    ---------------------------------------------------------
    -- Pre-cond  : None
    -- Post-cond : Place package in Alpha_Mode
    -- Comment   : In Alpha_Mode if the package is turned "On",
    --             either the True_Message of the False_Message
    --             is printed as each assertion test procedure
    --             is called.
    ---------------------------------------------------------

Procedure Off ;
    ---------------------------------------------------------
    -- Pre-cond  : None
    -- Post-cond : Displaying of assertions is terminated
    ---------------------------------------------------------

Procedure On ;
    ---------------------------------------------------------
    -- Pre-cond  : None
    -- Post-cond : Displaying of assertions continued depending
    --             upon current package mode
    ---------------------------------------------------------

Procedure Close ;

end Assert ;
-------------------------------------------------------------

with Text_IO ;

package body Assert is

type Main_Switch_Type  is (on, off) ;
type Package_Mode_Type is (Alpha, Beta) ;

Main         : Main_Switch_Type  := on ;
Package_Mode : Package_Mode_Type := Alpha ;

Display_File : Text_IO.File_Type ;

-- generic
--    Type_Message : string ;
-- procedure Assertion_Test ( Condition     : boolean ;
--                            Prefix        : String ;
--                            True_Message  : String ;
--                            False_Message : String ) ;
--
procedure Assertion_Test ( Condition     : boolean ;
                           Type_Message  : string ;
                           Prefix        : String ;
                           True_Message  : String ;
                           False_Message : String ) is

   begin -- Assertion_Test
--    Text_IO.Put_Line ("Assert") ;
   if Main = on then
--       Text_IO.Put_Line ("Main on") ;
      if Condition then
--          Text_IO.Put_Line ("Condition true") ;
         if Package_Mode = Alpha then
--             Text_IO.Put_Line ("Alpha") ;
            Text_io.Put_Line
               ( Display_File, Type_Message & ": " & Prefix &
" " & True_Message) ;
         end if ;
      else
            Text_io.Put_Line
               ( Display_File, Type_Message & ": " & Prefix &
" " & False_Message) ;
      end if ;
   end if ;
   end Assertion_Test ;
   ----------------------------------------------------------

-- procedure Precondition  is new Assertion_Test ("PRE") ;

-- procedure Postcondition is new Assertion_Test ("POST") ;

-- procedure Invariant     is new Assertion_Test ("INV");

-- procedure Assertion     is new Assertion_Test ("ASRT");

procedure Precondition ( Condition     : boolean ;
                         Prefix        : String ;
                         True_Message  : String ;
                         False_Message : String ) is

   begin -- Precondition
   Assertion_Test
      (Condition, "PRE", Prefix, True_Message, False_Message) ;
   end Precondition ;

procedure Invariant ( Condition     : boolean ;
                      Prefix        : String ;
                      True_Message  : String ;
                      False_Message : String ) is

   begin -- Invariant
   Assertion_Test
      (Condition, "INV", Prefix, True_Message, False_Message) ;
   end Invariant ;

procedure Postcondition ( Condition     : boolean ;
                          Prefix        : String ;
                          True_Message  : String ;
                          False_Message : String ) is

   begin -- Postcondition
   Assertion_Test
      (Condition, "POST", Prefix, True_Message, False_Message) ;
   end Postcondition ;

procedure Assertion ( Condition     : boolean ;
                      Prefix        : String ;
                      True_Message  : String ;
                      False_Message : String ) is

   begin -- Assertion
   Assertion_Test
      (Condition, "ASRT", Prefix, True_Message, False_Message) ;
   end Assertion ;

Procedure Beta_Mode is

   begin -- Beta_Mode
   Package_Mode := Beta ;
   end Beta_Mode ;
   ----------------------------------------------------------
```

```
Procedure Alpha_Mode is

   begin -- Alpha_Mode
   Package_Mode := Alpha ;
   end Alpha_Mode ;
   ---------------------------------------------------

Procedure Off is

   begin -- Off
   Main := off ;
   end Off ;
   ---------------------------------------------------

Procedure On is

   begin -- On
   Main := on ;
   end On ;
   ---------------------------------------------------

Procedure Close is

   begin -- Close
   Text_IO.Close (Display_File) ;
   end Close ;

begin -- Assert
Text_IO.Open (Display_File, Text_IO.out_file, "DISPLAY.ASC") ;
-- Text_IO.Open (Display_File, Text_IO.out_file, "CON:") ;
end Assert ;
```

8.2 Turing Simulator

8.2.1 The Procedure tm

```
----------------------------------------------------------------
-- Multi Tape Turing Machine Simulator
-- The simulator operates on a one way infinite tape.
-- Place state transition table in a data file using the
-- following two record formats
--
-- <state_name>
-- <symbol 1..n>  <next_state>  <<new_symbol>  <head_move 1..n>
--
-- A <state_name> record is followed by one transition record
-- for each state transition.  The <head_move> is a '<', '-', or
-- a '>' to indicate that the head is to move left, stay in
-- position, or move right respectively.  The set of state
-- transitions is terminated with a record containing a '~'.
-- There are two types of tilde record, either
--      ~
-- or
--      ~  <next_state>  <new_symbol>  <head_move>
-- The second type of record is a catch all set of transitions.
-- That is, a tilde record indicates the transitions for this
-- state for all other state symbol pairs not listed
--
-- States are indicated by a sequence of 1..12 non blank
-- characters
----------------------------------------------------------------
with Turing_Machine_Sim ;
with String_Scanner ;
with integer_text_io, Text_IO, Screen_IO ;

procedure TM is

   package tio renames Text_IO ;
   package iio renames integer_text_io ;
   package sio renames Screen_IO ;
   package s_s is new String_Scanner ;

   subtype Actual_Tape_Range is positive range 1..3 ;
   type Headline_Array is
           array (Actual_Tape_Range) of string (1..80) ;

   No_Of_Tapes : Actual_Tape_Range := 3;

   Header : Headline_Array := (
"   O N E    T A P E    T U R I N G    M A C H I N E    S I M
U L A T O R         ",
"   T W O    T A P E    T U R I N G    M A C H I N E    S I M
U L A T O R         ",
" T H R E E    T A P E    T U R I N G    M A C H I N E    S I
M U L A T O R       ") ;

   package TM is new Turing_Machine_Sim
      (No_Of_Tapes => No_Of_Tapes,
       Machine_Row => 3+5*No_Of_Tapes,
       Machine_Col => 25,
       ST_Row      => 5+5*No_Of_Tapes,
       ST_Col      => 40 ) ;

   subtype Tape_Range is TM.Tape_Range ;

   File_Name : string (1..80) ;
   F_Size    : natural ;

   State     : TM.State_Name_String ;
   S_Size    : natural ;
   St_Tok    : TM.Action_Record ;
   Tape      : TM.Tape_Init_Array ;
   Max_Steps : natural ;
   Buf       : string (1..80) ;
   B_S       : natural ;

   procedure Message (Msg : string ) is

      Buf : string (1..40) ;
      B_S : natural ;

      begin -- Message
      sio.PutS(Msg, 20, 1) ;
      sio.PutS ("<cr> to continue", 21, 1) ;
      tio.Get_Line (Buf, B_S) ;
      end Message ;

   procedure Headline is

      begin -- Headline
      sio.PutS (Header(No_Of_Tapes), 1, 1) ;
      end Headline ;

   begin -- TM
   sio.Clear ;
   Headline ;
   sio.PutS ("Enter file name (<cr> to quit): ", 8, 1);
   tio.Get_Line (File_Name, F_Size) ;
   while F_Size > 0 loop
      begin
      TM.Get_Machine (File_Name(1..F_Size)) ;
      tio.Put ("<cr> to continue") ;
      tio.Get_Line (Buf, B_S) ;
      -- TM.Display_Table ;
      -- tio.Put ("<cr> to continue") ;
      -- tio.Get_Line (Buf, B_S) ;
      sio.Clear ;
      Headline ;
      sio.PutS("Initial Tape 1 (<cr> to quit): ",
                   10-2*No_Of_Tapes, 1);
      tio.Get_Line (Tape(1).Init, Tape(1).Size) ;
      while Tape(1).Size > 0 loop
         for i in 2..Tape_Range'last loop
            sio.PutS("Initial Tape" & integer'image (i) &
             " (<cr> to quit): ",8+2*i-2*No_Of_Tapes, 1);
            tio.Get_Line (Tape(i).Init, Tape(i).Size) ;
            if Tape(i).Size = 0 then
             Tape(i).Size := 1 ;
             Tape(i).Init(1) := '~' ;
            end if ;
         end loop ;
         sio.PutS("Start State: ", 10, 1) ;
         tio.Get_Line (State, S_Size) ;
         s_s.Input (State (1..S_Size)) ;
         State := s_s.Scan.Piece ;
         sio.PutS("Max. No. of Steps (0<cr> = 10000): ", 12, 1);
         iio.Get (Max_Steps) ;       tio.Skip_Line ;
         if Max_Steps = 0 then
          Max_Steps := 10000 ;
         end if ;
         sio.Clear ;
         Headline ;
         begin
            State := TM.Simulate (Tape, State,
                        true, true, 1.25, Max_Steps) ;
            Message ("Machine Halted        ") ;

         exception
            when TM.State_Error =>
            Message ("Machine Halted        ") ;
            when TM.Data_Error =>
            Message ("State Transition Table Error") ;
            when TM.Tape_Overflow =>
            Message
("Ran off right end of tape -- possible infinite loop") ;
            when TM.Tape_Underflow =>
            Message ("Ran off left end of tape") ;
            when TM.Tape_Error =>
            Message ("Unexpected tape failure") ;
            when TM.Time_Exceeded =>
            Message
("Exceeded Max. Steps -- Possible infinite loop") ;

         end ;
         sio.Clear ;
         Headline ;
         sio.PutS("Initial Tape 1 (<cr> to quit):
                  ", 10-2*No_Of_Tapes, 1);
         tio.Get_Line (Tape(1).Init, Tape(1).Size) ;
      end loop ;
       exception
       when others =>
          tio.New_Line ;
          tio.Put ("****** " & File_Name (1..F_Size)
                & " error -- <cr> to continue") ;
          tio.Get_Line (File_Name, F_Size) ;
      end  ;
      TM.End_Sim ;
      sio.Clear ;
      Headline ;
      sio.PutS ("Enter file name (<cr> to quit): ", 8, 1);
      tio.Get_Line (File_Name, F_Size) ;
   end loop ;

end TM ;
```

8.2.2 The package tm_machine_sim

```
with Turing_Tape, TM_State_Transition_Control ;

generic

   No_Of_Tapes          : positive := 1 ;
   Max_State_Name_Size  : positive := 12 ;
   Machine_Row          : natural := 8 ;
   Machine_Col          : natural := 18 ;
   ST_Row               : natural := 21 ;
   ST_Col               : natural := 40 ;

package Turing_Machine_Sim is

   package TM_Tape is new Turing_Tape
      ( No_Of_Tapes => No_Of_Tapes,
     Top_Row      => Machine_Row - 5*No_Of_Tapes,
     Head_Column => Machine_Col+2 ) ;

   package TM_ST   is new TM_State_Transition_Control
      ( No_Of_Tapes => No_Of_Tapes ) ;

subtype State_Name_String is TM_ST.State_Name_String ;

subtype Action_Record is TM_ST.Action_Record ;
subtype Tape_Range      is TM_ST.Tape_Range ;
subtype Symbol_Array    is TM_ST.Symbol_Array ;

type     Tape_Init_Rec is
   record
      Size : natural := 0;
      Init : string (1 .. 255) ;
   end record ;
type     Tape_Init_Array is array (Tape_Range) of Tape_Init_Rec
;

Tape_Range_Error : exception ;
Time_Exceeded : exception ;

State_Error : exception renames TM_ST.State_Error ;
Data_Error  : exception renames TM_ST.Data_Error ;

Tape_Overflow : exception renames TM_Tape.Tape_Overflow ;
Tape_Underflow: exception renames TM_Tape.Tape_Underflow ;
Tape_Error     : exception renames TM_Tape.Tape_Error ;

procedure Get_Machine (File_Name : string)
   renames TM_ST.Get_Machine ;

procedure Display_Table
   renames TM_ST.Display_Table ;

function Simulate
   (Input        : Tape_Init_Array ;
    Start        : State_Name_String ;
    On_Screen    : boolean := false ;
    Single_Step : boolean := false ;
    Pause        : Duration := 1.50 ;
    Max_Steps    : natural := 1000 ) return State_Name_String ;

-----------------------------------------------------------------
------
   -- PreCond : Get_Machine has been called to initialize a
machine
   -- PostCond: Returns the terminiting state of the machine
   -- Exceptions:
   --    Tape_Underflow
   --    Tape_Overflow
   --    Time_Exceeded

-----------------------------------------------------------------
------

function Tape_Size (Tape : Tape_Range) return natural
   renames TM_Tape.Tape_Size ;

function Tape_Piece (Tape : Tape_Range ;
                     Left : natural := 1 ;
                     Right: natural ) return string
   renames TM_Tape.Tape_Piece ;

function Head_Position (Tape : Tape_Range) return natural
   renames TM_Tape.Head_Position ;

function Transition (State   : State_Name_String;
                     Symbol : Symbol_Array) return Action_Record
   renames TM_ST.Transition ;

function Valid_State_Name (State : State_Name_String) return
boolean
   renames TM_ST.Valid_State_Name ;

procedure End_Sim
   renames TM_ST.End_Sim ;

end Turing_Machine_Sim ;
```

```
------------------------ B O D Y ------------------------
with String_Scanner ;
with Text_IO, Screen_IO ;

package body Turing_Machine_Sim is

   package S_S is new String_Scanner
      (Max_Substring_Size => Max_State_Name_Size) ;

   package tio renames Text_IO ;
   package sio renames Screen_IO ;

Initialised    : boolean := false ;
Simulating     : boolean := false ;
On_Screen      : boolean := false ;
Halted         : boolean := false ;

procedure Box (Row : natural) is

   begin -- Box
   sio.PutS ("+-", Row, Machine_Col) ;
   for i in 1..Max_State_Name_Size loop
      sio.PutC ('-', Row, Machine_Col+1+i) ;
   end loop ;
   sio.PutS ("-+", Row, Machine_Col+2+Max_State_Name_Size) ;
   end Box ;


procedure Display_State (State : State_Name_String ;
                  Step_No: natural ) is

   begin -- Display_State
   -- sio.PutS("Box                                          ", 1, 1)
;
   Box (Machine_Row) ;
   -- sio.PutS("Box 2                                        ", 1, 1);
   sio.PutS ("| " & State & " |", Machine_Row+1, Machine_Col) ;
   -- sio.PutS("Box 3                                        ", 1, 1);
   Box (Machine_Row+2) ;
   -- sio.PutS("Time                                         ", 1, 1);
   sio.PutSN ("Step No. ", Step_No, Machine_Row-1, 1) ;
   end Display_State ;

procedure Display_Action (Action : Action_Record) is

   begin -- Display_Action
      -- sio.PutS ("State Transition"                  , ST_Row,
                                 ST_Col) ;
      sio.PutS ("Next State: " & Action.Next_State, ST_Row+1,
                                 ST_Col) ;
      for i in Tape_Range loop
      sio.PutS ("Tape" & integer'image (i) & ": "
           & Action.Action(i).New_Sym & "  "
             & Action.Action (i).Head_Move,
            ST_Row+1+i , ST_Col ) ;
      end loop ;
   end Display_Action ;

procedure Erase_Action is

   begin -- Erase_Action
   sio.PutS ("                                 ", ST_Row, ST_Col);
   sio.PutS ("                                ", ST_Row+1,ST_Col);
   for i in Tape_Range loop
      sio.PutS ("                               ",ST_Row+1+i,ST_Col);
   end loop ;
   end Erase_Action ;

function Simulate
   (Input        : Tape_Init_Array ;
    Start        : State_Name_String ;
    On_Screen    : boolean := false ;
    Single_Step : boolean := false ;
    Pause        : Duration := 1.50 ;
    Max_Steps    : natural := 1000 ) return State_Name_String is

   Current_State : State_Name_String := Start ;
   Continuous     : boolean := not Single_Step ;
   User_Step      : natural := 0 ;
   Max_User_Steps: natural := 1 ;
   Step_No        : natural := 0 ;
   Current_Symbol: Symbol_Array ;
   Action         : Action_Record ;

   Buffer : string (1..80) ;
   B_Size : natural ;

   procedure Get_Tape_Symbols is

      begin -- Get_Tape_Symbols
      for i in Tape_Range loop
      TM_Tape.Read_Head (i, Current_Symbol(i)) ;
      end loop ;
      end Get_Tape_Symbols ;
```

```
    procedure Update_Tape (Tape : Tape_Range) is

       begin -- Update_Tape
       If Action.Action(Tape).New_Sym = '-' then
        TM_Tape.Write_Head (Tape, Current_Symbol (Tape)) ;
      else
        TM_Tape.Write_Head (Tape, Action.Action(Tape).New_Sym) ;
       end if ;
       Case Action.Action(Tape).Head_Move is
       when '>' => TM_Tape.Move_Head_Right(Tape) ;
       when '<' => TM_Tape.Move_Head_Left(Tape) ;
       when others => null ;
       end case ;
       end Update_Tape ;

    begin -- Simulate
    Halted := false ;
    for i in Tape_Range loop
       TM_Tape.Initialise(i, Input(i).Init (1..Input(i).Size) );
    end loop ;
    if On_Screen then
        -- sio.Clear ;
        -- sio.PutS ("Display on", 1,1) ;
        TM_Tape.Display_On ;

        -- sio.PutS ("Tape symbols", 1, 1) ;
        Display_State (Current_State, Step_No) ;
    end if ;
    Sim_Loop :
    loop
       -- s.. PutS ("Begin Loop                    ", 1, 1) ;
       Get_Tape_Symbols ;
       -- sio.PutS ("Transition                    ", 1, 1) ;
       Action := Transition (Current_State, Current_Symbol) ;
       -- Display_Table ;
       if On_Screen then
        -- sio.PutS ("Display Action", 1, 1) ;
        Display_Action (Action) ;
        if Continuous then
            delay Pause ;
          else
            User_Step := User_Step + 1 ;
            If User_Step >= Max_User_Steps then
             User_Step := 0 ;
             sio.PutS("C(ont)  Q(uit) ### : ", 23, 1) ;
             tio.Get_Line (Buffer, B_Size) ;
             if B_Size > 0 then
                 Case Buffer (1) is
                 when 'c' | 'C' => Continuous := true ;
                 when 'q' | 'Q' => exit Sim_Loop ;
                 when '0' .. '9' =>
                    Max_User_Steps :=
                        natural'value (Buffer(1..B_Size)) ;
                 when others =>
                    null ;
                 end case ;
             end if ;
            end if ;
        end if ;
        -- sio.PutS ("Erase action", 1, 1) ;
        Erase_Action ;
       end if ;
       for i in Tape_Range loop
       -- sio.PutS ("Update tape", 1, 1) ;
       Update_Tape (i) ;
       end loop ;
       Current_State := Action.Next_State ;
       Step_No := Step_No + 1 ;
       if On_Screen then
        -- sio.PutS ("Display step # and current state", 1, 1) ;
        Display_State (Current_State, Step_No) ;
       end if ;
       if Step_No > Max_Steps then
        raise Time_Exceeded ;
       end if ;
    end loop Sim_Loop ;
    return Current_State ;

    exception
       when State_Error =>
       Halted := true ;
       return Current_State ;
    end Simulate ;

end Turing_Machine_Sim ;
```

8.2.3 tm State_Transition_Control Package

```
with String_Scanner ;

generic
   No_Of_Tapes         : positive ; -- := 1 ;
   Max_State_Name_Size : positive := 12 ;

package TM_State_Transition_Control is

   package S_S is new String_Scanner
      (Max_Substring_Size => Max_State_Name_Size) ;

subType Token_Record      is S_S.Token_Type ;
subtype State_Name_String is string (1..Max_State_Name_Size) ;
subtype Tape_Range        is positive range 1..No_Of_Tapes ;
type    Symbol_Array      is array (Tape_Range) of character ;
type Tape_Action is
   record
      New_Sym   : character := ' ' ;
      Head_Move : character := ' ';
   end record ;
type Action_Array is array (Tape_Range) of Tape_Action ;

type Action_Record is
   record
      Next_State: State_Name_String := (Others => ' ');
      Action    : Action_Array ;
   end record ;

State_Error : exception ;
Data_Error  : exception ;

procedure Get_Machine (File_Name : string) ;
   ------------------------------------------------------------
   -- Precond : File_Name is the name of a file that contains a
   --           correct description of Turing Machine
   -- Postcond: The package is initialized with the TM's state
   --           transition table
   -- Exception: Data_Error iff file format error
   ------------------------------------------------------------

procedure Display_Table ;
   ------------------------------------------------------------
   -- Precond : Get_Machine was called and terminated
   --           successfully
   -- Postcond: Displays the state transition table
   ------------------------------------------------------------

procedure Display_On ;

procedure Display_Off ;

function Transition (State  : State_Name_String;
                     Symbol : Symbol_Array) return Action_Record ;
   ------------------------------------------------------------
   -- Precond : Get_Machine was called and terminated
   --           successfully
   -- Postcond: returns the transition for the (State, Symbol)
   --           pair
   -- Exception: State_Error if no state transition described
   ------------------------------------------------------------

function Valid_State_Name (State : State_Name_String)
                                          return boolean ;
   ------------------------------------------------------------
   -- Precond : Get_Machine was called and terminated
   --           succeddfully
   -- Postcond: returns true iff State is a valid state name
   ------------------------------------------------------------

procedure End_Sim ;
   ------------------------------------------------------------
   -- Precond : non
   -- Postcond: returns the package to its initial condition
   ------------------------------------------------------------

end TM_State_Transition_Control ;

------------------------ B O D Y -------------------------
with List_Pt_Lpt, List_Lpt_Lpt ;
with Text_IO, Screen_IO ;

package body TM_State_Transition_Control is

package tio renames Text_IO ;
package sio renames Screen_IO ;

type Transition_Record is
   record
      Symbol    : Symbol_Array ;
      Next_State: State_Name_String ;
      Action    : Action_Array ;
   end record ;

package T_L is new List_Pt_Lpt (Transition_Record) ;

type State_Record is
   record
      State  : State_Name_String ;
      T_List : T_L.List_Type ;
   end record ;

------------------------------------------------------------
-- Specifications of support procedures needed to instantiate
-- List_Lpt_Lpt with State_Records
------------------------------------------------------------
procedure Copy
   (Source : in     State_Record ;
    Target : in out State_Record) ;

procedure Move_And_Reset
   (Source : in out State_Record ;
    Target : in out State_Record) ;

procedure Zap (Source : in out State_Record) ;

function "=" (Left, Right : State_Record) return boolean ;

package S_L is new List_Lpt_Lpt
   (State_Record, Copy, Move_And_Reset, Zap, "=") ;

Initialized   : boolean := false ;
On_Screen     : boolean := false ;
Current_State : State_Name_String ;

State_List : S_L.List_Type ;

------------------------------------------------------------
-- Bodies of support procedures required to instantiate
-- List_Lpt_Lpt with State_Records
------------------------------------------------------------
procedure Copy
   (Source : in     State_Record ;
    Target : in out State_Record) is

   begin -- Copy
   Target.State := Source.State ;
   T_L.Copy (Source.T_List, Target.T_List) ;
   end Copy ;

procedure Move_And_Reset
   (Source : in out State_Record ;
    Target : in out State_Record) is

   begin -- Move_And_Reset
   Target.State := Source.State ;
   T_L.Move_And_Reset (Source.T_List, Target.T_List) ;
   end Move_And_Reset ;

procedure Zap (Source : in out State_Record) is

   begin -- Zap
   T_L.Zap (Source.T_List) ;
   end Zap ;

function "=" (Left, Right : State_Record) return boolean is

   begin -- "="
   return false ; -- Not used
   end "=" ;

procedure Get_Machine (File_Name : string) is

   In_File : tio.File_Type;
   Buffer : string (1..255) ;
   B_Size : natural ;
   State_Rec : State_Record ;
   Trans_Rec : Transition_Record ;
   Token_Buf : Token_Record ;

   procedure Get_Transition_Record is

      begin -- Get_Transition_Record
      for i in Tape_Range loop
      Trans_Rec.Symbol(i) := Token_Buf.Piece(1) ;
      tio.Put ("    " & Trans_Rec.Symbol(i) );
      if i /= Tape_Range'Last then
         Token_Buf := S_S.Scan ;
      end if ;
      end loop ;
      Trans_Rec.Next_State := S_S.Scan.Piece ;
      tio.Put ("    " & Trans_Rec.Next_State) ;
      for i in Tape_Range loop
      Token_Buf := S_S.Scan ;
      if Token_Buf.Size = 1 then
         Trans_Rec.Action(i).New_Sym := Token_Buf.Piece(1) ;
         tio.Put (" " & Trans_Rec.Action(i).New_Sym & "  ") ;
        else
         tio.Put (Token_Buf.Piece); tio.New_Line ;
         raise Data_Error ;
      end if ;
      Token_Buf := S_S.Scan ;
```

```
      if Token_Buf.Size = 1 then
          case Token_Buf.Piece(1) is
          when '<' | '>' | '-' =>
             Trans_Rec.Action(i).Head_Move :=
                                         Token_Buf.Piece(1) ;
             tio.Put (Trans_Rec.Action(i).Head_Move & "  ") ;
          when others            => raise Data_Error ;
          end case ;
        else
          tio.Put (Token_Buf.Piece); tio.New_Line ;
          raise Data_Error ;
      end if ;
      end loop ;
      if t_l.Is_Empty (State_Rec.T_List) then
       t_l.Insert_First (Trans_Rec, State_Rec.T_List) ;
     else
       t_l.Append (State_Rec.T_List, Trans_Rec) ;
      end if ;
      tio.New_Line ;
      end Get_Transition_Record ;

   begin -- Get_Machine
   S_L.Zap (State_List) ;
   tio.Open (In_File, tio.In_file, File_Name) ;
   tio.Put ("TM State Transition Table"); tio.new_line ;
   while not tio.End_Of_File (In_File) loop
      tio.Get_Line (In_File, Buffer, B_Size) ;
      S_S.Input (Buffer (1..B_Size) );
      State_Rec.State := S_S.Scan.Piece ;
      if State_Rec.State (1..2) = "--" then
       tio.Put (Buffer (1..B_Size)) ;
       tio.New_Line ;
     else
       tio.Put (State_Rec.State) ; tio.New_Line ;
       tio.Get_Line (In_File, Buffer, B_Size) ;
       S_S.Input (Buffer (1..B_Size) );
       Token_Buf := S_S.Scan ;
       while (Token_Buf.Size /= 0 ) loop
          -- and (Token_Buf.Piece(1) /= '-') loop
          if (Token_Buf.Size >= 2)
           and then (Token_Buf.Piece(1..2) = "--") then
           tio.Put (Buffer (1..B_Size)) ;
           tio.New_Line ;
            else
           Get_Transition_Record ;
          end if ;
          tio.Get_Line (In_File, Buffer, B_Size) ;
          S_S.Input (Buffer (1..B_Size) );
          Token_Buf := S_S.Scan ;
       end loop ;
       -- if (Token_Buf.Size = 1)
       --    and (Token_Buf.Piece(1) = '-') then
       --     if not S_S.End_Of_String then
       --          Get_Transition_Record ;
       --     end if ;
       --   else
       --     tio.Put (Token_Buf.Piece); tio.New_Line ;
       --     raise Data_Error ;
       -- end if ;
       if S_L.Is_Empty (State_List) then
           S_L.Insert_First (State_Rec, State_List) ;
         else
           S_L.Append (State_List, State_Rec) ;
       end if ;
       tio.New_Line ;
      end if ;
   end loop ;
   tio.Close (In_File) ;
   Initialized := true ;
   end Get_Machine ;

procedure Display_Table is

   type Not_Used_Record is
      record
      Not_Used : characte.
      end record ;

   Not_Used : Not_Used_Record ;

   procedure State_Action (State_Info : in     State_Record ;
                Not_Used    : in out Not_Used_Record ;
                Continue    :    out boolean ) ;

   procedure Traverse_States is new S_L.Rec_Sel_Iterator
      (Not_Used_Record, State_Action) ;

   procedure Transition_Action
                 (Action_Info : in     Transition_Record ;
                  Not_Used    : in out Not_Used_Record ;
                  Continue    :    out boolean ) ;

   procedure Traverse_Transitions is new T_L.Rec_Sel_Iterator
      (Not_Used_Record, Transition_Action) ;

   procedure Transition_Action
                 (Action_Info : in     Transition_Record ;
                  Not_Used    : in out Not_Used_Record ;
                  Continue    :    out boolean ) is

      begin -- Transition_Action
      for i in Tape_Range loop
      Text_IO.Put ("   " & Action_Info.Symbol(i) ) ;
      end loop ;
      Text_IO. Put ("   " &  Action_Info.Next_State ) ;
      for I in Tape_Range loop
          Text_IO. Put ("  " &  Action_Info.Action(i).New_Sym
            & "  " &  Action_Info.Action(i).Head_Move ) ;
      end loop ;
      Text_IO.New_Line ;
      Continue := true ;
      end Transition_Action ;

   procedure State_Action (State_Info : in     State_Record ;
                Not_Used    : in out Not_Used_Record ;
                Continue    :    out boolean ) is

      begin -- State_Action
      Text_IO.Put (State_Info.State); text_io.New_Line ;
      Traverse_Transitions (State_Info.T_List, Not_Used) ;
      Text_IO.New_Line ;
      Continue := true ;
      end State_Action ;

   begin -- Display_Table
   if Initialized then
       Traverse_States (State_List, Not_Used) ;
   end if ;
   end Display_Table ;

procedure Display_On is

   begin -- Display_On
   On_Screen := true ;
   end Display_On ;

procedure Display_Off is

   begin -- Display_Off
   On_Screen := false ;
   end Display_Off ;

function Transition (State  : State_Name_String ;
              Symbol : Symbol_Array) return Action_Record is

   type Return_Record is
      record
      Found : boolean := false ;
      Answer: Action_Record ;
      end record ;

   C_State : State_Name_String := State ;
   Answer  : Return_Record ;

   procedure Find_State (S_Rec   : in     State_Record ;
              C_State : In out State_Name_String ;
              Continue:    out boolean ) ;

   procedure Traverse_States is new S_L.Rec_Sel_Iterator
      (State_Name_String, Find_State) ;

   procedure  Find_Transition  (Act               :  in
Transition_Record ;
                    Answer   : in out Return_Record ;
                    Continue :    out boolean ) ;

   procedure Traverse_Actions is new T_L.Rec_Sel_Iterator
      (Return_Record, Find_Transition) ;

   function Match_Up (Sym1, Sym2 : Symbol_Array)
                                 return boolean is

      Answer : boolean := true ;

      begin -- Match_Up
      for i in Tape_Range loop
       if Sym1(i) /= '-' and then
          Sym2(i) /= '-' and then
          Sym1(i) /= Sym2(i) then
          Answer := false ;
          exit ;
       end if ;
      end loop ;
      return Answer ;
      end Match_Up ;

   procedure Find_State (S_Rec   : in     State_Record ;
              C_State : In out State_Name_String ;
              Continue:    out boolean ) Is

      begin -- Find_State
      if S_Rec.State = C_State then
       Traverse_Actions (S_Rec.T_List, Answer) ;
       Continue := false ;
     else
       Continue := true ;
      end if ;
      end Find_State ;

   procedure  Find_Transition  (Act               :  in
Transition_Record ;
                    Answer   : in out Return_Record ;
                    Continue :    out boolean ) is

      begin -- Find_Transition
      if Match_Up (Symbol, Act.Symbol) then
```

```
        Answer := (true, (Act.Next_State, Act.Action) ) ;
        Continue := false ;
      else
        Continue := true ;
       end if ;
       end Find_Transition ;

   begin -- Transition
   Traverse_States (State_List, C_State) ;
   if Answer.Found then
       return Answer.Answer ;
     else
       raise State_Error ;
   end if ;
   end Transition ;

function Valid_State_Name (State : State_Name_String) return
boolean is

   function Traverse (List : S_L.List_Type) return boolean is

      begin -- Traverse
      if State = S_L.Current_Object (List).State then
        return true ;
     elsif S_L.Is_Empty (S_L.Tail_Of(List)) then
        return false ;
     else
        return Traverse (S_L.Tail_Of(List)) ;
      end if ;
      end Traverse ;

   begin -- Valid_State_Name
   if S_L.Is_Empty (State_List) then
        return false ;
     else
        return Traverse (State_List) ;
   end if ;
   end Valid_State_Name ;

procedure End_Sim is

   begin -- End_Sim
   Initialised := false ;
   S_L.Zap (State_List) ;
   end End_Sim ;

end TM_State_Transition_Control ;
```

8.2.4 The Turing_Tape Package

```
-----------------------------------------------------------------
--  This package maintains and (optionally) displays Turing
--  machine tapes.
-----------------------------------------------------------------
generic

   No_Of_Tapes    : positive ;  -- 1..3
   Top_Row        : natural := 3 ;   -- Screen position of tape
   Head_Column    : natural := 20 ;  -- position of R/W head
   Max_Tape_Size  : integer := 1500 ;-- Max size of tape

package Turing_Tape is

subtype Tape_Range is positive range 1..No_Of_Tapes ;

Tape_Overflow : exception ;
Tape_Underflow: exception ;
Tape_Error    : exception ;

procedure Initialize (Tape_No : Tape_Range; Start : string) ;
   -------------------------------------------------------------
   -- PreCond  : None
   -- PostCond : Give a Turing tape an initial value
   -------------------------------------------------------------

procedure Display_On ;
   -------------------------------------------------------------
   -- PreCond  : None
   -- PostCond : Display the current tape on the screen
   -------------------------------------------------------------

procedure Display_Off ;
   -------------------------------------------------------------
   -- PreCond  : None
   -- PostCond : Stop displaying the tape
   -------------------------------------------------------------

procedure Read_Head (Tape_No : Tape_Range;
                     Symbol : in out character) ;
   -------------------------------------------------------------
   -- PreCond  : Tape was initialized
   -- PostCond : Return's the symbol currently being scanned by
   --             the R/W head
   -------------------------------------------------------------

procedure Write_Head (Tape_No : Tape_Range;
                      Symbol : in out character) ;
   -------------------------------------------------------------
   -- PreCond  : Tape was initialized
   -- PostCond : Replace the tape position currently being
   --            scanned by the R/W head with Symbol
   -------------------------------------------------------------

procedure Move_Head_Left (Tape_No : Tape_Range) ;
   -------------------------------------------------------------
   -- PreCond  : Tape was initialized
   -- PostCond : Move the R/W head one position left on the tape
   -- Exception: Tape_Underflow if the R/W head moves off the
   --            tape
   -------------------------------------------------------------

procedure Move_Head_Right (Tape_No : Tape_Range) ;
   -------------------------------------------------------------
   -- PreCond  : Tape was initialized
   -- PostCond : Move the R/W head one position right on the
   --            tape
   -- Exception: Tape_Overflow if the R/W/ head moves off the
   --            tape
   -------------------------------------------------------------

procedure End_Simulation ;
   -------------------------------------------------------------
   -- PreCond  : None
   -- PostCond : Marks the current tape as unitialized
   -------------------------------------------------------------

function Tape_Size (Tape_No : Tape_Range) return natural ;
   -------------------------------------------------------------
   -- PreCond  : None
   -- PostCond : returns the length of the non-blank portion of
   --            the tape
   -------------------------------------------------------------

function Tape_Piece (Tape_No : Tape_Range ;
               Left : natural := 1 ;
               Right: natural ) return string ;
   -------------------------------------------------------------
   -- PreCond  : None
   -- PostCond : Returns Tape (Left..Right) ;
   -------------------------------------------------------------

function Current_Symbol (Tape_No : Tape_Range)
                                         return character ;
   -------------------------------------------------------------
   -- PreCond  : Tace must be Initialized
   -- PostCond : Returns Symbol under the R/W head ;
   -------------------------------------------------------------

function Head_Position (Tape_No : Tape_Range) return natural ;
   -------------------------------------------------------------
   -- PreCond  : None
   -- PostCond : Returns Head position index ;
   -------------------------------------------------------------

end Turing_Tape ;

----------------------- B O D Y ---------------------------

with Screen_IO, Text_IO ;

package body Turing_Tape is

package sio renames Screen_IO ;

Separator : constant character := '|' ;
Corner    : constant character := '+' ;
Top       : constant character := '-' ;
R_W_Head  : constant character := '^' ;
Blank     : constant character := ' ' ;

Left_Edge : constant natural := Head_Column / 2 ;
Right_Edge: constant natural := (80 - Head_Column) / 2 ;

type Tape_Rec is
   record
      Tape       : string (1..Max_Tape_Size) ;
      Head       : natural := 1 ;
      Right_End  : natural := 0;
   end record ;
type Tape_Array is array (Tape_Range) of Tape_Rec ;

Initialized : boolean := false ;
Simulating  : boolean := false ;
On_Screen   : boolean := false ;

Data : Tape_Array ;

procedure Display_Tape_Symbols is

   Left    : natural ;
   Scr_Col : natural ;

   begin -- Display_Tape_Symbols
   for i in Tape_Range loop
      if Data(i).Head > Left_Edge then
       Left    := Data(i).Head-Left_Edge+1 ;
       Scr_Col := Head_Column - 2*(Left_Edge-1) ;
     else
       Left    := 1 ;
       Scr_Col := Head_Column - 2*(Data(i).Head-1) ;
      end if ;
      while (Scr_Col <=80) and (Left <= Data(i).Tape'last) loop
      sio.PutC (Data(i).Tape(Left), Top_Row+1 + 5*(i-1),
                       Scr_Col) ;
      Scr_Col := Scr_Col + 2 ;
      Left := Left + 1 ;
      end loop ;
   end loop ;
   end Display_Tape_Symbols ;

procedure Initialize (Tape_No : Tape_Range; Start : string) is

   begin -- Initialize
   Initialized := true ;
   Simulating  := false ;
   Data(Tape_No).Tape := (others => '-') ;
   Data(Tape_No).Tape (1..Start'last) := Start ;
   Data(Tape_No).Head := 1 ;
   Data(Tape_No).Right_End := Start'last ;
   if On_Screen then
       Display_On ;
   end if ;
   end Initialize ;

procedure Display_On is

   Col : natural ;

   begin -- Display_On
   On_Screen := true ;
   if Initialized then
        Initialized := false ;
        Simulating  := true ;
        for i in Tape_Range loop
        Col := Head_Column - 1;
```

```
      while Col <= 80 loop
         sio.PutC (Corner, Top_Row + 5*(i-1), Col) ;
         sio.PutC (Separator, Top_Row+1 + 5*(i-1), Col) ;
         sio.PutC (Corner, Top_Row+2 + 5*(i-1), Col) ;
         if Col < 80 then
          sio.PutC (Top, Top_Row + 5*(i-1), Col+1) ;
          sio.PutC (Top, Top_Row+2 + 5*(i-1), Col+1) ;
         end if ;
         Col := Col + 2 ;
      end loop ;
      sio.PutC (R_W_Head, Top_Row+3 + 5*(i-1), Head_Column) ;
      sio.PutC (Separator, Top_Row+4 + 5*(i-1), Head_Column) ;
      end loop ;
      Display_Tape_Symbols ;
   end if ;
   end Display_On ;

procedure Display_Off is

   begin -- Display_Off
   On_Screen := false ;
   end Display_Off ;

procedure Read_Head (Tape_No : Tape_Range ;
                     Symbol : in out character) is

   begin -- Read_Head
   if Initialised or Simulating then
       Initialised := false ;
       Simulating  := true ;
       Symbol := Data(Tape_No).Tape (Data(Tape_No).Head) ;
     else
       raise Tape_Error ;
   end if ;
   end Read_Head ;

procedure Write_Head (Tape_No : Tape_Range ;
                      Symbol : in out character) is

   begin -- Write_Head
   if Initialised or Simulating then
       Initialised := false ;
       Simulating  := true ;
       Data(Tape_No).Tape (Data(Tape_No).Head) := Symbol ;
       if On_Screen then
           Display_Tape_Symbols ;
       end if ;
     else
       raise Tape_Error ;
   end if ;
   end Write_Head ;

procedure Move_Head_Left (Tape_No : Tape_Range) is

   begin -- Move_Head_Left
   if Initialised or Simulating then
       Initialised := false ;
       Simulating  := true ;
       if Data(Tape_No).Head = 1 then
           raise Tape_Underflow ;
         else
        Data(Tape_No).Head := Data(Tape_No).Head - 1 ;
       end if ;
       if On_Screen then
        If Data(Tape_No).Head < Left_Edge then
            sio.PutS ("  ", Top_Row + 5*(Tape_No-1),
                      Head_Column-2*Data(Tape_No).Head-1) ;
            sio.PutS ("  ", Top_Row+1 + 5*(Tape_No-1),
                      Head_Column-2*Data(Tape_No).Head-1) ;
            sio.PutS ("  ", Top_Row+2 + 5*(Tape_No-1),
                      Head_Column-2*Data(Tape_No).Head-1) ;
           end if ;
           Display_Tape_Symbols ;
       end if ;
     else
       raise Tape_Error ;
   end if ;
   end Move_Head_Left ;

procedure Move_Head_Right (Tape_No : Tape_Range) is

   begin -- Move_Head_Right
   if Initialised or Simulating then
       Initialised := false ;
       Simulating  := true ;
       if Data(Tape_No).Head = Data(Tape_No).Tape'last then
           raise Tape_Overflow ;
         else
        Data(Tape_No).Head := Data(Tape_No).Head + 1 ;
        if Data(Tape_No).Head > Data(Tape_No).Right_End then
            Data(Tape_No).Right_End := Data(Tape_No).Head ;
        end if ;
       end if ;
       if On_Screen then
        If Data(Tape_No).Head <= Left_Edge then
            sio.PutS (Corner & Top, Top_Row + 5*(Tape_No-1),
                           Head_Column-2*Data(Tape_No).Head+1) ;
            sio.PutS (Separator & Blank,
                      Top_Row+1 + 5*(Tape_No-1),
                      Head_Column-2*Data(Tape_No).Head+1) ;
            sio.PutS (Corner & Top, Top_Row+2 + 5*(Tape_No-1),
                      Head_Column-2*Data(Tape_No).Head+1) ;
          end if ;
              Display_Tape_Symbols ;
          end if ;
        else
          raise Tape_Error ;
      end if ;
      end Move_Head_Right ;

procedure End_Simulation is

   begin -- End_Simulation
   Initialised := false ;
   Simulating := false ;
   On_Screen := false ;
   end End_Simulation ;

function Tape_Size (Tape_No : Tape_Range) return natural is

   begin -- Tape_Size
   return Data(Tape_No).Right_End ;
   end Tape_Size ;

function Tape_Piece (Tape_No : Tape_Range ;
                     Left  : natural := 1 ;
                     Right : natural ) return string is

   begin -- Tape_Piece
   return Data(Tape_No).Tape (Left..Right) ;
   end Tape_Piece ;

function Current_Symbol (Tape_No : Tape_Range)
                                           return character is

   begin -- Current_Symbol
   if Initialised or Simulating then
       return Data(Tape_No).Tape (Data(Tape_No).Head) ;
     else
       raise Tape_Error ;
   end if ;
   end Current_Symbol ;

function Head_Position (Tape_No : Tape_Range) return natural is

   begin -- Head_Position
   return Data(Tape_No).Head ;
   end Head_Position ;

end Turing_Tape ;
```

8.3 Towers of Hanoi

8.3.1 Programming Assignment

The Towers of Hanoi program was discussed in class. A copy of the Hanoi program is in

/home/faculty/beidler/Cmps144/hanoi.ada

You can see how the program executes by bringing up an xterm window and executing

/home/faculty/beidler/Cmps144/hanoi

Below is a graph of the number of disk moves required to complete the Towers of Hanoi problem with 1 through 8 disks, inclusive. Note that the number of moves required to perform the Towers of Hanoi is

$$M(d) = 2^{d} - 1$$

where d is the number of disks.

```
                                              127    255    511   1023   2047   4095

  |
80+
  |
  |
  |
  |
70+
  |
  |
  |                                     h
60+
  |
  |
  |
  |
50+
  |
  |
  |
  |
40+
  |
  |
  |                               h
30+
  |
  |
  |
  |
20+
  |
  |                         h
  |
```

```
 10+
   |                         h
   |
   |      h/s    h/s
---+------+------+------+------+------+------+------+------+------+------+-------
          1      2      3      4      5      6      7      8      9      10
```

The file /home/faculty/beidler/Cmps144/saigon.ada contains a partially completed Towers of Saigon program. The Towers of Saigon is like the Towers of Hanoi, but uses four spindles instead of three. Complete and run the program and do the following:

1. Run the program with all values between one and eight disks.

2. Cut and paste the graph above and plug in the letter 's' to roughly indicate the growth pattern for the number of moves required to solve the Towers of Saigon with between one and eight disks, inclusive.

3. Include with your submission for this assignment and estimate of the move function for the Towers of Saigon program.

4. Submit the information above, via email, with a .lst of your program.

8.3.2 Basic Towers of Hanoi Program

```
with tty, text_io, hanoi_board;

procedure hanoi is

   Number_Of_Disks : constant natural := 6 ;

   package New_board is new
      hanoi_board (No_Of_Spindles => 4,
                   No_Of_Disks    => Number_Of_Disks ) ;
   use New_board ;

   package tio renames Text_IO ;

   in_string        : string (1 .. 40);
   string_size      : integer;
   first_name       : positive := 1 ;
   second_name      : positive := 2 ;
   third_name       : positive := 3 ;

   Step_number : long_integer := 0 ;

   procedure towers_of_hanoi
      ( from_spindle,
        help_spindle,
        to_spindle      : in     positive ;
        number_of_disks : in     integer ) is

      begin -- of towers_of_hanoi
      if number_of_disks = 1 then
            Step_number := Step_Number + 1 ;
            Move_Disk (from_spindle, to_spindle);
            -- tio.put ("Step ");
            -- tio.put
            --(long_integer'image (Step_Number));
            -- tio.put (": Move a disk from ");
            -- tio.put
            -- (positive'image (from_spindle) );
            -- tio.put (" to ");
            -- tio.put (positive'image (to_Spindle) ) ;
            -- tio.new_line ;
        else
            -- Move all but the bottom disk to the "help" spindle
            towers_of_hanoi
               ( from_spindle    => from_spindle,
                 help_spindle    => to_spindle,
                 to_spindle      => help_spindle,
                 number_of_disks => number_of_disks - 1);
            --%% Move the bottom disk to the "to" spindle
            towers_of_hanoi
               ( from_spindle    => from_spindle,
                 help_spindle    => help_spindle,
                 to_spindle      => to_spindle,
                 number_of_disks => 1);
-- Move the disks from the "help" spindle on top of the big disk
            towers_of_hanoi
               ( from_spindle    => help_spindle,
                 help_spindle    => from_spindle,
                 to_spindle      => to_spindle,
                 number_of_disks => number_of_disks - 1);
      end if;
      end towers_of_hanoi;

   begin -- of hanoi
   -- tio.put("How many disks? ");
   -- tio.get_line(in_string, string_size);
   -- number_of_disks := integer'value(in_string);
   towers_of_hanoi(
       from_spindle    => first_name,
       help_spindle    => second_name,
       to_spindle      => third_name,
       number_of_disks => number_of_disks);
   tty.Put    (23,    1,    long_Integer'image(Step_Number)    )
;tty.Put(24,1, " ") ;
   end hanoi;
```

8.3.3 Towers of Hanoi Display Package

```
generic
   No_Of_Spindles  : positive ;
   No_of_disks : positive ;

package hanoi_board is

  Disk_Error : exception ;

  procedure move_disk ( from_spindle, to_spindle : in
positive ) ;

end hanoi_board;

--________________________________________
with screen_io ;
-- with tty ;
-- with text_io;

package body hanoi_board is

  Base_Row  : constant natural := 20 ;

  Disk_Size : constant natural := (80)/No_Of_Spindles ;
  subtype disk_string is string (1 .. Disk_Size) ;

  sample_disk : constant Disk_String := (others => '#') ;
  blanker     : constant Disk_String := (others => ' ') ;
  max_disks   : constant integer := (Disk_Size-2) / 2 ;

  subtype stack_range is integer range 1 .. max_disks;
  type spindle_array is array (stack_range) of natural ;
  type spindle_record is
     record
        top_disk : integer        := 0 ;
        Disk     : spindle_array := (others => 0);
     end record;
  subtype spindle_range is integer range 1 .. No_Of_Spindles;
  type board_type is array (spindle_range) of spindle_record;

  type spindle_column_type is array (1 .. No_Of_Spindles) of
integer ;

  spindle_column : spindle_column_type ;
  hanoi          : board_type ;
  from_number,
  to_number      : integer ;
  Count          : natural := 0 ;

  procedure Display_Disk
            (Row, Column : natural ; Disk : integer ) is

     begin -- Display_Disk
     if Disk <= 0 then
--     tty.Put (Row, Column+Disk, Blanker(1..(-Disk)) & "|" &
--                                 Blanker(1..(-Disk)) ) ;
      screen_io.Puts (Blanker(1..(-Disk)) & "|" &
           Blanker(1..(-Disk)), Row, Column+Disk ) ;
      else
--    tty.Put (Row, Column-Disk, Sample_Disk (1..Disk) & "#" &
--                                 Sample_Disk (1..Disk) ) ;
      screen_io.puts (Sample_Disk (1..Disk) & "#" &
           Sample_Disk (1..Disk), Row, Column-Disk) ;
     end if ;
     end Display_Disk ;

  procedure initialize_board (hanoi : in out board_type) is

     Procedure Display_Spindle
           (Column : natural ; Spindle : Spindle_Record) is

        begin -- Display_Spindle
        for i in Stack_Range loop
           Display_Disk (Base_Row - i, Column, Spindle.Disk(i));
        end loop ;
        end Display_Spindle ;

     begin -- of initialise_board
     spindle_column (1) :=  Disk_Size/2 ;
     for i in 2 .. No_Of_Spindles loop
        Spindle_Column (i) := Spindle_Column (i-1) + Disk_Size;
        -- text_io.Put(integer'image(Spindle_Column(i)));
     end loop ;

     for i in 1 .. No_Of_Disks loop
        Hanoi(1).Disk(i) := No_Of_Disks+1-i ;
        -- text_io.Put (integer'image(No_Of_Disks+1-i));
     end loop ;
     Hanoi(1).Top_Disk := No_Of_Disks ;

     --tty.Clear_Screen ;
     screen_io.Clear ;
     --tty.Put (Base_Row, 1,
     screen_io.Puts (
     "________________________________________"
     , Base_Row, 1) ;

     for i in 1 .. No_Of_Spindles loop
        Display_Spindle (Spindle_Column(i), Hanoi(i) ) ;
     end loop ;
     end initialise_board;

  procedure move_disk
       ( from_spindle, to_spindle : in     positive ) is

     Above, Below : integer ;

     begin
     Display_Disk (Base_Row-Hanoi (To_Spindle).Top_Disk - 1,
                   Spindle_Column (To_Spindle),

Hanoi(From_Spindle).Disk(Hanoi(From_Spindle).Top_Disk) );

     Display_Disk (Base_Row-Hanoi (From_Spindle).Top_Disk,
                   Spindle_Column (From_Spindle),

-Hanoi(From_Spindle).Disk(Hanoi(From_Spindle).Top_Disk) );

     Hanoi(To_Spindle).Top_Disk := Hanoi(To_Spindle).Top_Disk+1;
     Hanoi(To_Spindle).Disk(Hanoi(To_Spindle).Top_Disk) :=
        Hanoi(From_Spindle).Disk(Hanoi(From_Spindle).Top_Disk);
     Hanoi(From_Spindle).Disk(Hanoi(From_Spindle).Top_Disk)
                                           := 0 ;
     Hanoi(From_Spindle).Top_Disk :=
                          Hanoi(From_Spindle).Top_Disk-1 ;
     Count := Count + 1 ;
     --tty.Put (3, 50, "Moves = " & integer'image(Count)) ;
     screen_io.Puts ("Moves = " & integer'image(Count) , 3, 50);
     --tty.Put (22, 1, " ") ;
     screen_io.Puts (" ", 22, 1) ;
     -- delay 0.001  ;
     for i in 1 .. 32767 loop
     null ;
     end loop ;
     --tty.Put (3,3,"if            ") ;
     if (Hanoi(To_Spindle).Top_Disk > 1) then
      --tty.Put (3,3,"inner if") ;
      --tty.Put (4,3, integer'image (To_Spindle));
      --tty.Put
         (5,3, integer'image (Hanoi(To_Spindle).Top_Disk));
      --tty.Put (6,3, integer'image
         (Hanoi(To_Spindle).Disk(Hanoi(To_Spindle).Top_Disk)));
      --tty.Put (7,3, integer'image
        (Hanoi(To_Spindle).Disk(Hanoi(To_Spindle).Top_Disk-1)));
      Above :=
            Hanoi(To_Spindle).Disk(Hanoi(To_Spindle).Top_Disk);
      Below :=
          Hanoi(To_Spindle).Disk(Hanoi(To_Spindle).Top_Disk-1);
      if Above > Below then
          raise Disk_Error ;
      end if ;
      --tty.Put (3,3,"exit inner") ;
     end if ;
     --tty.Put (3,3, "exit outer") ;
     end move_disk;

begin -- hanoi_board
if No_Of_Disks > Max_Disks then
    raise constraint_error ;
  else
    initialise_board (hanoi) ;
end if ;
end hanoi_board;
```

8.3.4 Towers of Saigon Sample Code

```
-- Copyright (c) 1991,1992 John Beidler
--                          Computing Sciences Dept.
--                         Univ. of Scranton, Scranton, PA 18510
--
--                          (717) 941-7446 voice
--                          (717) 941-4250 FAX
--                          beidler@guinness.cs.uofs.edu
--                          beidler@scranton [Bitnet]
--
--  For use by non-profit educational institutions only.
--  This software is GUARANTEED.  Please report any errors.  All
--  corrections will be made as soon as possible (normally
--  within one working day).

with text_io, hanoi_board;

procedure Saigon is

   subtype Disk_Range is integer range 1 .. 9 ;

   package tio renames Text_IO ;
   package iio is new Text_Io.integer_io(integer) ;

   in_string       : string (1 .. 40);
   string_size     : integer;
   No_Of_Disks     : Disk_Range ;

   Step_number : long_integer := 0 ;

   Procedure Set_Display (Number_Of_Disks : in    integer) is

      package New_board is new
         hanoi_board (No_Of_Spindles => 4,
                      No_Of_Disks     => Number_Of_Disks ) ;
      use New_board ;

   procedure towers_of_Saigon
      ( from_spindle,
        help_spindle,
        Help_2_spindle,
        to_spindle       : in     positive ;
        Number_Of_disks : in     integer ) is

      begin -- of towers_of_Saigon
      -- tio.put (integer'image (No_Of_Disks) ); tio.New_Line;
      Case Number_Of_Disks is
        when 1 =>
           Move_Disk (From_Spindle, To_Spindle) ;
           Step_number := Step_Number + 1 ;
        when 2 =>
           Towers_Of_Saigon (From_Spindle,
                             Help_2_Spindle,
                             To_Spindle,
                             Help_Spindle,
                             1 ) ;
           Towers_Of_Saigon (From_Spindle,
                             Help_2_Spindle,
                             Help_Spindle,
                             To_Spindle,
                             1 ) ;
           Towers_Of_Saigon (Help_Spindle,
                             Help_2_Spindle,
                             From_Spindle,
                             To_Spindle,
                             1 ) ;
        when others =>
           Towers_Of_Saigon (From_Spindle,
                             Help_2_Spindle,
                             To_Spindle,
                             Help_Spindle,
                             Number_Of_Disks-2) ;
           Towers_Of_Saigon (From_Spindle,
                             Help_Spindle,
                             To_Spindle,
                             Help_2_Spindle,
                             1 ) ;
           Towers_Of_Saigon (From_Spindle,
                             Help_2_Spindle,
                             Help_Spindle,
                             To_Spindle,
                             1 ) ;
           Towers_Of_Saigon (Help_2_Spindle,
                             From_Spindle,
                             Help_Spindle,
                             To_Spindle,
                             1 ) ;
           Towers_Of_Saigon (Help_Spindle,
                             Help_2_Spindle,
                             From_Spindle,
                             To_Spindle,
                             Number_Of_Disks-2) ;
      end case ;
      end towers_of_Saigon;

      begin -- Set_Display
    towers_of_Saigon(
        from_spindle     => 1,
        help_spindle     => 2,
        Help_2_Spindle   => 3,
        to_spindle       => 4,
        Number_of_disks      => number_of_disks);
    end Set_Display ;

begin -- of Saigon
tio.put("How many disks? ");
iio.get (No_Of_Disks) ; tio.Skip_Line ;
Set_Display (No_Of_Disks) ;
-- tty.Put (23, 1, long_integer'image(Step_Number) )
-- tty.Put(24,1, " ") ;
Text_io.Get_Line (In_String, String_Size) ;
end Saigon;
```

9 References

Beidler, J. "Structuring Iterators for Reuse", To Appear in the Proceedings of Ada-Europe '93. Paris, France. July 1993.

-, **An Object-based Approach to Data Structures Using Ada**, a text in preparation.

-, "Building on the Booch Components: What Can Be Learned When Modifying Real World Software Tools for Educational Use", **Proceeding of Tri-Ada ;92**, Nov. 1992.

-, "A Role for Iterators as A Tool for Software Reuse", **Proceedings of WAdaS '92**, July 1992.

-, "A Sequence of Integrated Laboratory Assignments for Freshmen". **Proceeding of ASEET-6 Symposium**, Washington, D.C. Sept. 1991.

-, P. Jackowitz, and R. Plishka, "A Graphics-based Editor for Parallel Systems", **Proceeding of the Third Annual CASE Workshop**, Cambridge, MA, July, 1988.

-, and P. Jackowitz. "On Defining Consistent Generics." **SIGPLAN Bulletin**, v.21 no.4, April 1986. pp. 32-41.

-, and P. Jackowitz, **Modula-2**, PWS Publishers, 1986.

-, R. Austing, and L. Cassel, "Computing Programs in Small Colleges," Available through the ACM -- Summary appears in CACM, June 1985.

-, **Data Structures**, Allyn-Bacon, Waltham, MA. 1980.

Brender, Ronald. **Character Set Issues for Ada 9X**. SEI. Pittsburgh, PA. 1989.

Booch, Grady, **Software Components with Ada**, Benjamin Cummings, Menlo Park, CA. 1987.

Cohen, Sholom. **Ada Support for Software Reuse**. SEI. Pittburgh, PA. 1990.

Doerr, A. and K. Levasseur, **Applied Discrete Structures for Computer Science**, 2nd Ed., Dellen/MacMillian

Lewis, H, and C. Papadimitriou, **Elements of the Theory of Computation**, Prentice-Hall, 1981.

Liskov, Barbara, and John Guttag, **Abstraction and Specification in Program Development**, McGraw Hill, New York, 1986.

Piff, M., **Discrete Mathematics: An Introduction for Software Engineers**, Cambridge University Press, Cambridge, 1991.

Polya, George., "Probabilities in Proofreading". **American Mathematical Monthly**. v.83 n.1 January 1976, p.42.

Shaw, Mary, W. Wulf, and R. London, "Abstraction and Verification in Alphard: Iteration and Generators", **Alphard: Form and Content**. Springer-Verlag, New York. 1981.

Stubbs, D and N. Webre. **Data Structures with Abstract Data Types and Modula-2**. Brooks/Cole. Monterey, CA. 1987.