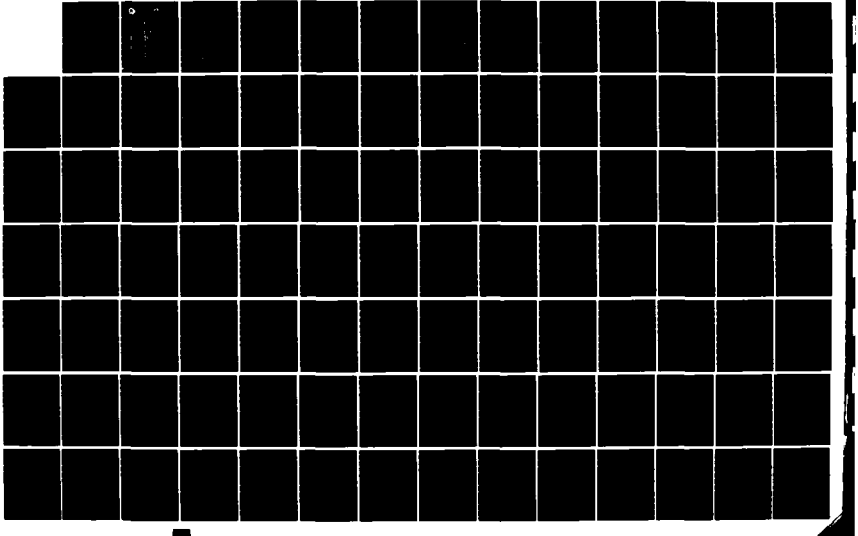
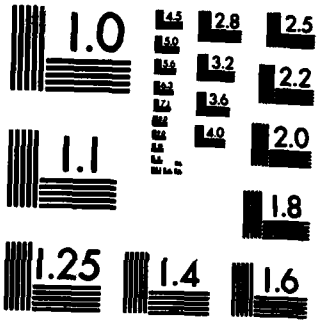
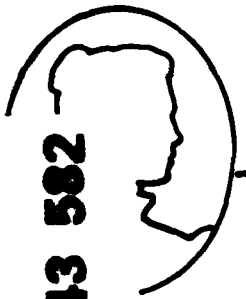
AD-A143 502 ADA (TRADEMARK) TRAINING CURRICULUM USING THE ADA LANGUAGE REFERENCE MANUAL L402 TEACHER'S GUIDE(U) SOFTECH INC WALTHAM MA JUL 84 DAA807-83-C-K514 1/5

UNCLASSIFIED F/G 9/2 NL







July 1984

Ada® Training Curriculum

Using the Ada Language Reference Manual
L402
Teacher's Guide

Center For Tactical Computer Systems
(CENTACS)

U.S. Army Communications-Electronics Command
(CECOM)

Contract DAAB07-83-C-K514

Prepared By:

SOFTECH, INC.
460 Totten Pond Road
Waltham, MA 02154

•Ada is a registered trademark of the U.S. Government, Ada Joint Program Office

*Approved For Public Release/Distribution Unlimited

(L402)

USING THE Ada LANGUAGE REFERENCE MANUAL

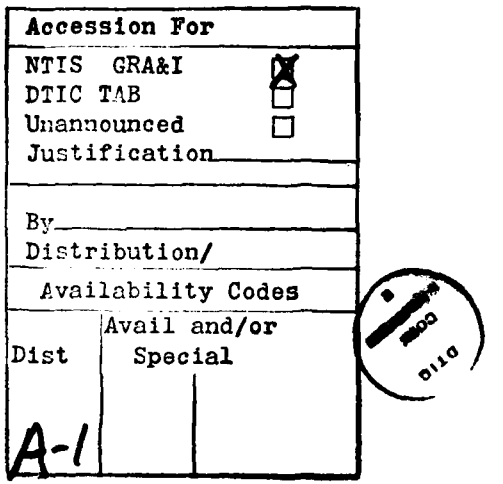
VG 703

INSTRUCTOR NOTES

- INTRODUCE YOURSELF AS THE INSTRUCTOR AND HAVE STUDENTS INTRODUCE THEMSELVES
- STATE THE NAME OF THE COURSE WITH A BRIEF DESCRIPTION
- QUICKLY REVIEW COURSE HANDOUTS TO BE SURE EACH STUDENT HAS THEIRS.
- STATE UP FRONT THAT THIS COURSE IS NOT FOR THE CASUAL USER OF Ada AND THAT ITS PURPOSE IS NOT TO TEACH THE Ada LANGUAGE
- IT IS A COURSE FOR LANGUAGE LAWYERS, THE PEOPLE WHO WILL BE ANSWERING THE QUESTIONS OF OTHERS IN AN ORGANIZATION.

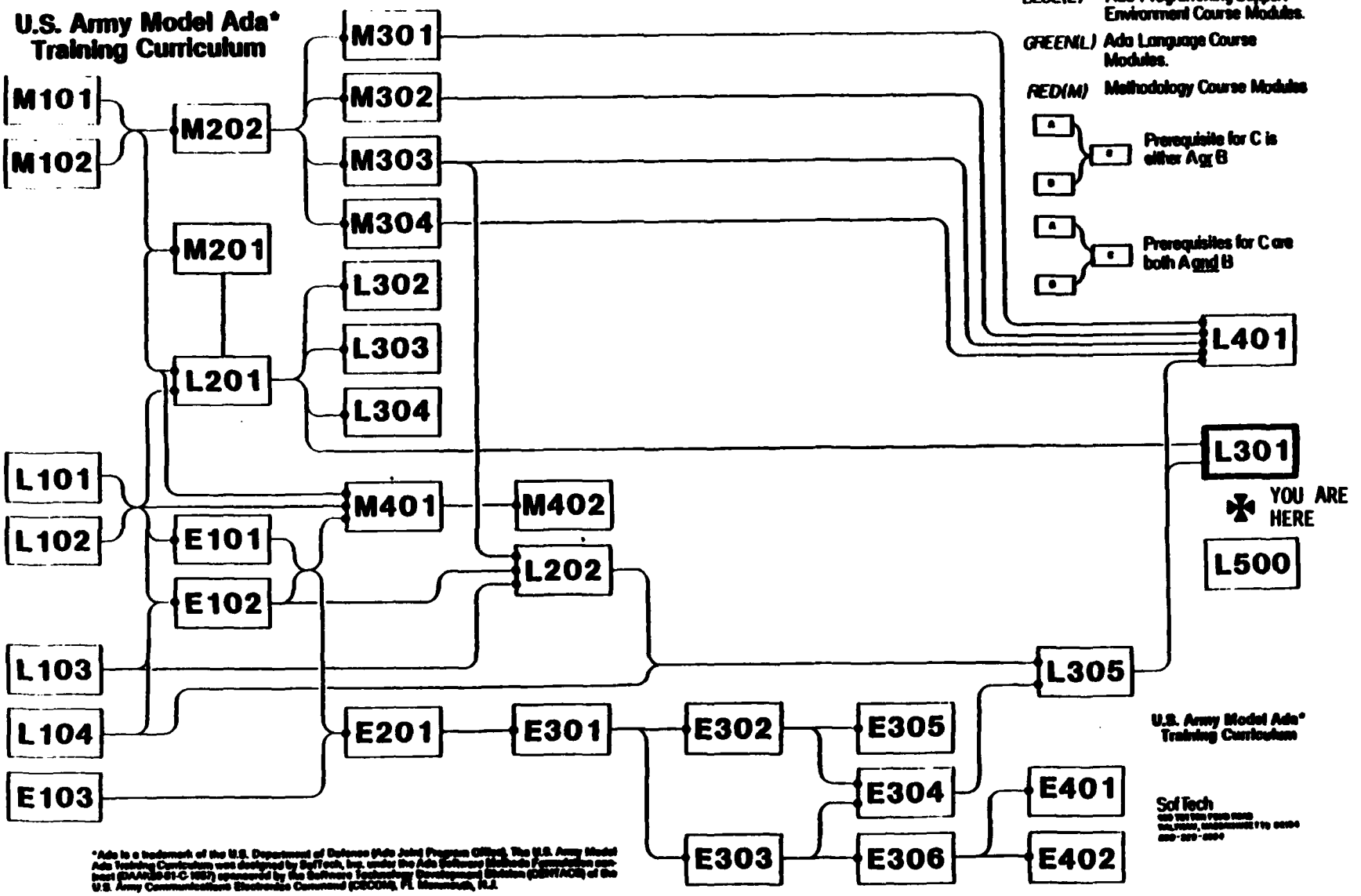
Section 1

INTRODUCTION



INSTRUCTOR NOTES

SHOW WHERE THE STUDENTS ARE IN THE CURRICULUM.


U.S. Army Model Ada* Training Curriculum
BLUE(E) Ada Programming Support Environment Course Modules.
GREEN(L) Ada Language Course Modules.
RED(M) Methodology Course Modules
Prerequisite for C is either A or B
Prerequisites for C are both A and B
M101
M102
M202
M201
L201
L101
L102
E101
E102
L103
L104
E103
M301
M302
M303
M304
L302
L303
L304
M401
M402
L202
E201
E301
E302
E303
E305
E304
E306
E401
E402
L305
L401
L301
YOU ARE HERE
L500
U.S. Army Model Ada* Training Curriculum
SofTech
*Ada is a trademark of the U.S. Department of Defense (Ada Joint Program Office). The U.S. Army Model Ada Training Curriculum was designed by SofTech, Inc. under the Ada Software Education Foundation contract (DAAB30-81-C-1057) sponsored by the Software Technology Development Division (SEMTACS) of the U.S. Army Communications Electronics Command (CECOM), Ft. Monmouth, N.J.

VG 703

INSTRUCTOR NOTES

- EXPLAIN THAT THE FIRST FIVE TOPICS WILL BE COVERED QUICKLY IN THE MORNING OF THE FIRST DAY
- THE LAST TOPIC, CHAPTER TOPICS, IS THE MEAT OF THE COURSE. THE MAJORITY OF THE TIME WILL BE SPENT ON IT. SOME CHAPTERS ARE MORE COMPLEX THAN OTHERS. THE COMPLEX ONES WILL BE GIVEN MORE TIME THAN THE "EASY" ONES. SOME "EASY" CHAPTERS, FOR EXAMPLE CHAPTER 5, MAY BE SKIPPED ENTIRELY, DEPENDING ON TIME.

INTRODUCTION	} 90 minutes
BREAK	
STRUCTURE OVERVIEW	} 60 minutes
Chapter 1	} 20 minutes
Chapter 2	} 10 minutes
LUNCH	
Chapter 8	} 90 minutes
BREAK	
Chapter 3	} 90 minutes

DAY 1

Chapter 4	} 30 minutes
Chapter 6	} 30 minutes
Chapter 7	} 30 minutes
BREAK	
Chapter 9	} 60 minutes
Chapter 12	} 30 minutes
LUNCH	
Chapter 11	} 30 minutes
Chapter 10	} 30 minutes
Chapter 13	} 30 minutes
BREAK	
Chapter 5	} 25 minutes
Chapter 14	} 55 minutes
CONCLUSION	} 10 minutes

DAY 2

COURSE TOPICS

- GOALS
- COURSE RATIONALE
- PREREQUISITE
- LRM HISTORY
- LRM STRUCTURE
- CHAPTER TOPICS

INSTRUCTOR NOTES

- A QUICK INTRODUCTION OF THE LRM STRUCTURE IS NEEDED FOR BACKGROUND

- SYNTAX STRUCTURE IS A REVIEW. IF STUDENTS SEEM UNFAMILIAR WITH BNF NOTATION EXPLAIN THAT THIS COURSE IS NOT DESIGNED TO TEACH SYNTAX. WE DON'T HAVE TIME.

LRM STRUCTURE

- SYNTAX NOTATION
- LANGUAGE TERMS
- NOTES AND EXAMPLES
- REFERENCES
- ANNEXES
- APPENDICES

INSTRUCTOR NOTES

- THESE TOPICS WILL BE REPEATED FOR EACH CHAPTER

- BRIEFLY DESCRIBE THE COURSE FORMAT
  - MIXTURE OF LECTURE AND IN CLASS EXERCISES (MOST OF THE TIME WILL BE SPENT DOING AND DISCUSSING EXERCISES)
  - NO HOMEWORK (OTHER THAN READING THE LRM AND POSSIBLY THE IMPLEMENTORS' GUIDE)
  - BECAUSE OF TIME CONSTRAINTS, CLASS EXERCISES WILL HAVE A TIME LIMIT. THIS MAY NOT BE ENOUGH TIME FOR STUDENTS TO DO ALL THE EXERCISES.

- THE EXERCISES MODEL REAL LIFE WHERE QUESTIONS CAN'T ALWAYS WAIT UNTIL THE NEXT DAY FOR ANSWERS
  - ANSWERS OR SOLUTIONS WILL BE DISCUSSED IN CLASS

CHAPTER TOPICS

- CHAPTER OUTLINE
- DISCUSS ANY KEY ISSUES DESCRIBED IN THE CHAPTER
- EXAMPLE PROBLEM AND SOLUTION
- EXERCISES
- DISCUSS SOLUTIONS

INSTRUCTOR NOTES

THE PURPOSE OF THE NEXT FEW SLIDES IS TO STATE UPFRONT WHAT THE STUDENTS SHOULD EXPECT TO GET OUT OF THE COURSE, WHAT BACKGROUND THEY SHOULD HAVE, AND WHY THEY SHOULD BOTHER STUDYING THE LRM.

COURSE GOALS

- TO ENABLE STUDENTS TO CORRECTLY INTERPRET THE ADA LANGUAGE THROUGH:
  - KNOWLEDGE OF THE LRM
  - KNOWLEDGE OF THE ADA BOARD

INSTRUCTOR NOTES

- THE MODULE IS ABOUT USING THE LRM, NOT ABOUT USING ADA.
- THE LRM WILL NOT BE WALKED THROUGH CHAPTER BY CHAPTER OR PARAGRAPH BY PARAGRAPH.
- EXPLAIN THAT KNOWING THE LRM STRUCTURE, THE MEANING OF SPECIAL LANGUAGE TERMS, AND BEING AWARE OF CERTAIN SEMANTIC RAMIFICATIONS WILL HELP THE STUDENT TO CORRECTLY ANSWER ADA RELATED QUESTIONS.
- ASK THE STUDENTS TO OPEN AND LEAF THROUGH THE LRM AS WE COVER CERTAIN SECTIONS. (I.E. TURN TO ANNEX A WHEN WE TALK ABOUT ANNEX A.)

COURSE GOALS

- FAMILIARIZE THE STUDENT WITH THE Ada LANGUAGE REFERENCE MANUAL (LRM)
  - THROUGH KNOWING THE LRM STRUCTURE
  - THE LANGUAGE TERMS
  - SOME INTERESTING SEMANTIC RAMIFICATIONS

- ENABLE THE STUDENT TO CORRECTLY INTERPRET THE LRM
  - THROUGH EXAMPLES AND SOLUTIONS

INSTRUCTOR NOTES

- COULD POINT OUT THAT THE FIRST GOAL WAS NOT COMPLETELY MET. IF THE LRM WAS CLEAR AND CONCISE, THIS COURSE WOULDN'T BE NECESSARY.

- LAST POINT, REMIND THEM AGAIN ABOUT THE ADA BOARD.

Ada LANGUAGE REFERENCE MANUAL GOALS

- PROVIDE A CLEAR, CONCISE AND COMPLETE DEFINITION OF THE LANGUAGE
  - SPECIFYING WHICH ASPECTS OF THE LANGUAGE ARE UNDEFINED
  - DEFINING WHERE IMPLEMENTATION DEPENDENT ASPECTS ARE DEFINED
- PROVIDE A MEANS OF RESOLVING LANGUAGE ISSUES
  - THE LRM INTENDED TO BE THE SOLE MEANS BY WHICH LANGUAGE ISSUES ARE RESOLVED

INSTRUCTOR NOTES

- MENTION THAT ALTHOUGH CHAPTER 1 OF THE LRM DOES INDICATE THAT THE FORMAT USED IN SPECIFYING THE SYNTAX IS A SUGGESTED FORMAT, IT IS BY NO MEANS A REQUIREMENT AND NEVER WAS A GOAL

NON-GOALS

(OF BOTH THE COURSE AND THE LRM)

- PROVIDE STYLE GUIDELINES
  - INDENTATION TO IMPROVE READABILITY
  - COMMENTS TO IMPROVE READABILITY
  - NAMING CONVENTIONS
- PROVIDE ANSWERS TO IMPLEMENTATION SPECIFIC ISSUES
  - HOW PROGRAM UNITS ARE INVOKED
  - HOW EXECUTING UNITS ARE CONTROLLED
  - REQUIRED SPEED OF EXECUTION
  - FORM OR CONTENT OF ERROR AND WARNING MESSAGES
  - EFFECT OF ERRONEOUS PROGRAMS
- SPECIFY THE INTENDED OR TYPICAL USE OF LANGUAGE FEATURES

INSTRUCTOR NOTES

- EXPLAIN THAT THERE SIMPLY ISN'T ENOUGH TIME TO TEACH EVERY DETAIL. WHAT WE ARE TRYING TO DO HERE, IS ENABLE THEM TO INTERPRET THE LRM.

COURSE NON-GOALS

- TO TEACH PROGRAMMING TECHNIQUES
- TO TEACH EVERY DETAIL IN THE LRM

INSTRUCTOR NOTES

EXPLAIN THAT HISTORICALLY THIS COURSE IS INTENDED FOR SENIOR LEVEL PEOPLE WHO ARE EXPECTED TO BE THE ADA EXPERTS. OTHERS WILL BE COMING TO THEM WITH PROBLEMS. WE ARE TRYING TO PREPARE THEM, SO THEY WILL BE ABLE TO SOLVE THESE PROBLEMS.

GIVE SOME EXAMPLE OF QUESTIONS WHICH THEY SHOULD BE PREPARED FOR.

THE ULTIMATE GOAL

TO BE ABLE TO ANSWER, "WHAT'S WRONG WITH THIS PROGRAM?"

INSTRUCTOR NOTES

- STRESS OFFICIALLY
  - IMPLEMENTERS WHEN CHALLENGING VALIDATION MAY ONLY CITE THE LRM
  - REQUESTS FOR BUG FIXES IN COMPILERS MUST CITE THE LRM

WHY STUDY THE LRM

- ONLY THE ANSI STANDARD, MIL-STD-1815 A MAY BE USED TO OFFICIALLY RESOLVE Ada LANGUAGE ISSUES.

- *TO DISCOVER INGENIOUS LANGUAGE FEATURES.*

INSTRUCTOR NOTES

- LIST REFERENCES NOT BASED ON THE LRM
  - BARNE'S BOOK "PROGRAMMING IN ADA"
  - WEGNER'S SELF ASSESSMENT
  - MANY OTHERS

WHY THE LRM OVER OTHER REFERENCES?

- OTHER REFERENCES SHOULD NOT BE USED BECAUSE THEY: MAY BE INCORRECT OR INCOMPLETE PARAPHRASES AND INTERPRETATIONS OF THE STANDARD.
- MAY NOT BE BASED ON THE STANDARD.
- HAVE NOT BEEN SUBJECTED TO THE SAME INTENSE SCRUTINY.

INSTRUCTOR NOTES

- THE IG'S GOAL OF TESTING ADHERENCE TO THE LRM CAUSED THE ANALYSIS WHICH LED TO THE SEMANTIC RAMIFICATIONS AND "ACCEPTED INTERPRETATIONS" OF VAGUELY DEFINED AREAS.

- STUDENTS WILL HAVE A COPY OF THE IG. HAVE THEM TURN TO AND READ AN INTERESTING SEMANTIC RAMIFICATION. POINT OUT THAT NOT ALL SEMANTIC RAMIFICATIONS WILL BE EXPLICITLY TALKED OF HERE, ONLY MAJOR ONES. THEY MAY READ ABOUT THE REST IN THE IG ON THEIR OWN TIME.

- NOT A STANDARD YET ANYWAY.

- IT EVEN STATES THAT THE LRM TAKES PRECEDENCE.

- HAVE THE STUDENTS OPEN THE IG. EXPLAIN ITS FORMAT. FOR THE MOST PART THE STUDENTS WILL BE INTERESTED IN THE SECTION TITLED "SEMANTIC RAMIFICATIONS".

THE IMPLEMENTERS' GUIDE

- PURPOSE IS TO DEFINE COMPILER VALIDATION PROCEDURES
- SIDE EFFECTS
  - PRESENTS SEMANTIC RAMIFICATIONS (INTENDED FOR THE IMPLEMENTER TO BE AWARE OF BUT USEFUL FOR THE ADA PROGRAMMER TO BE AWARE OF)
  - PRESENTS "ACCEPTED INTERPRETATIONS" OF LANGUAGE ISSUES
- WHEN IT DIFFERS FROM THE LRM, LRM TAKES PRECEDENCE
- IS SUBJECT TO CHALLENGE
- NOT A STANDARD
- HOWEVER, PROBABLY THE BEST REFERENCE FOR INTERPRETING THE LRM

INSTRUCTOR NOTES

EACH SECTION IN THE IG CORRESPONDS TO A LRM CHAPTER. IN EACH SECTION THERE ARE THESE SUBSECTIONS. THE FIRST, SEMANTIC RAMIFICATIONS WILL BE OF MOST USE TO THE STUDENTS.

POINT OUT THAT BECAUSE THE UPDATED IG IS STILL UNDER DEVELOPMENT, THE COPY THEY HAVE IS INCOMPLETE.

- SEMANTIC RAMIFICATIONS SUB-SECTION
  - POINTS OUT UNOBVIOUS SEMANTIC RAMIFICATION OF SPECIAL INTEREST TO THE IMPLEMENTER

- COMPILE-TIME RESTRICTIONS SUB-SECTION
  - EXPLICITLY LISTS LEGALITY RESTRICTIONS NOT EXPRESSED BY THE SYNTAX

- EXCEPTION CONDITIONS SUB-SECTION
  - EXPLICITLY LISTS THE CONDITIONS UNDER WHICH AN EXCEPTION MUST BE RAISED

- TEST OBJECTIVES AND DESIGN GUIDELINES
  - LIST TEST OBJECTIVES AND TEST GUIDELINES

- ASSUMPTIONS/GAPS
  - DOCUMENTS LANGUAGE ASPECTS THAT ARE UNCLEAR OR INCONSISTENT (RARE IN THE STANDARD)

INSTRUCTOR NOTES

- WHEN TALKING ABOUT Ada 82
  - STRESS THAT THE DIFFERENCES ARE SUBTLE
  - COULD EASILY BE MISLED BY USING THE Ada 82 LRM IN SOLVING Ada QUESTIONS FOR EXAMPLE:

```
package P is
    type Grandparent is (X, Y, Z);
    type Parent is new Grandparent;
    type Child is new Parent;  -- legal in 82
                               -- illegal in the standard
end P;
```

PAST Ada LRMs

Ada 80 LRM

- OUT OF DATE
- SYNTACTIC DIFFERENCES AS WELL AS SEMANTIC DIFFERENCES
  - PARAMETERLESS FUNCTIONS
  - QUALIFIED EXPRESSIONS

Ada 82 LRM

- SUBTLE SEMANTIC DIFFERENCES
- CLOSE ENOUGH TO THE STANDARD TO BE DANGEROUS

INSTRUCTOR NOTES

- STRESS THAT MANY ARE BASED ON Ada 80.
- SOME ARE BASED ON Ada 82.
- ALL THAT ARE BASED ON 80 OR 82 CANNOT BE TRUSTED TO BE RIGHT.
- MANY INCOMPLETE REFERENCES ARE WRONG. IN AN ATTEMPT TO SIMPLIFY, STATEMENTS ARE MADE AS IF THEY ARE FACT. OFTEN THE STATEMENTS ARE INCORRECT.

Ada REFERENCE BOOKS

- OFTEN NOT BASED ON THE STANDARD
  - BASED ON PAST REFERENCE MANUALS
    (Ada 80, 82, AND IN RARE CASES THE PRELIMINARY
    DESIGN FOR THE GREEN LANGUAGE)
    - TO GET SOMETHING ON THE MARKET

- OFTEN WRONG OR INCOMPLETE
  - DUE TO AN INCOMPLETE UNDERSTANDING OF THE LANGUAGE
    ON THE PART OF THE AUTHOR

- SOME INFORMATION IS DEEMED UNNECESSARY FOR THE INTENDED READER

- NONE GO INTO ENOUGH DEPTH OR DETAIL

INSTRUCTOR NOTES

- POINT OUT THAT:
  - THE LRM DICTATES WHAT IS LEGAL.
  - THE ADA BOARD INTERPRETS.
  - THE AVO ENFORCES.
- COMPILERS MAY CONTAIN ERRORS IN AREAS:
  - THAT ARE LIGHTLY COVERED IN THE VALIDATION SUITE.
  - THAT ARE CHANGED DUE TO CHALLENGES OF THE VALIDATION SUITE.
- ALL ERRORS FOUND IN VALIDATED COMPILERS ARE REPORTED TO THE AVO AND THE VALIDATION SUITE IS UPDATED ACCORDINGLY.
- THERE ARE MANY AREAS WHERE THE COMPILER IS FREE TO DO THINGS ANY WAY THEY WISH. THESE AREAS MAY CHANGE FROM RELEASE TO RELEASE ALSO.
- ON THE WHOLE, CODING FOR THE COMPILER WILL CAUSE A LOT OF REWRITING OF CODE WHEN THE COMPILER CHANGES, AND THE ODDS ARE FAIRLY GOOD THAT THE COMPILERS WILL CHANGE.

CODING FOR THE COMPILER

- MEANS THAT THE COMPILER IS USED TO DECIDE WHAT IS LEGAL
- A COMPILER MAY CONTAIN ERRORS
  - ADA COMPILERS ARE REGULATED BY THE Ada VALIDATION OFFICE (AVO)
  - NEXT COMPILER RELEASE MAY BE DIFFERENT
    - ERROR CORRECTIONS FOR RE-VALIDATION
    - ENHANCEMENTS AND OPTIMIZATIONS MAY CAUSE A FEATURE TO BEHAVE DIFFERENTLY
- THE LANGUAGE ALLOWS MANY AREAS TO BE IMPLEMENTATION-DEPENDENT
  - HANDLING OF ERRONEOUS PROGRAMS
  - DEFINITION OF
    - THE TYPE INTEGER
    - THE PACKAGE SYSTEM
    - IMPLEMENTATION-DEFINED ATTRIBUTES
  - THESE CAN (ARE ALLOWED) TO CHANGE FROM COMPILER TO COMPILER
  - CODING FOR A PARTICULAR COMPILER CAN BE VERY COSTLY!
  - PORTABILITY DECREASES

INSTRUCTORS NOTES

- IN SUMMARY, STRESS THAT THE RATIONALE FOR THE COURSE IS "SINCE ONLY THE LRM CAN BE USED TO RESOLVE LANGUAGE ISSUES, HAVING PEOPLE LEARN TO READ IT AND INTERPRET IT CORRECTLY IS THE BEST WAY TO INSURE HIGH QUALITY SOFTWARE AT MINIMAL COSTS."

- BY QUICKLY IDENTIFYING PROBLEM AREAS MAJOR SOFTWARE REWRITES CAN BE AVOIDED.

SUMMARY POINTS

- MOST OTHER REFERENCES ARE INCOMPLETE, OR INCORRECT, OR BOTH
- COMPILERS ARE NOT STATIC, REGARDLESS OF CORRECTNESS
  - MANY IMPLEMENTATION DEPENDENT AREAS EXIST
- ANSI STANDARD LRM IS THE ONLY COMPLETE AND (BY DEFINITION) CORRECT SPECIFICATION OF THE ADA LANGUAGE
- NEED TO UNDERSTAND THE LRM TO IDENTIFY (AND AVOID) PROBLEM AREAS IN PROGRAMS

INSTRUCTOR NOTES

- STRESS A STRONG BACKGROUND IN ADA
- RESTATE THAT THIS COURSE WILL NOT TEACH ANY OF THESE TOPICS
- QUESTIONS DEALING WITH WHAT THESE CONSTRUCTS ARE, OR HOW ONE WOULD USE THEM ARE NOT APPROPRIATE FOR THIS CLASS. WE WILL NOT SPEND TIME EXPLAINING THEM.

PREREQUISITES

STRONG BACKGROUND IN ADA!

- TRAINING AND EXPERIENCE IN THE FOLLOWING AREAS
  - Ada PROGRAMMING
    - GENERICS
    - TASKS
    - EXCEPTIONS
    - DISCRIMINATED RECORDS
    - PRIVATE TYPES
    - LIMITED TYPES
  - ENVIRONMENT TRAINING
    - LINKERS
    - LOADERS
    - DEBUGGERS
    - DATABASE

INSTRUCTOR NOTES

THESE QUESTIONS ARE ADDRESSED IN OTHER MODULES. HERE WE ARE INTERESTED IN WHAT THE LRM SAYS ABOUT THE LEGALITY OF THESE CONSTRUCTS.

INAPPROPRIATE QUESTIONS

- WHEN WOULD YOU WANT TO USE FEATURE X?
- HOW DOES THE PROGRAM WORK?
- WHY DOES THE LANGUAGE DEFINE IT THIS WAY?
- HOW DO I IMPLEMENT MY SPECIFIC PROBLEM IN ADA?

INSTRUCTOR NOTES

THIS SECTION EXPLAINS THE HISTORY OF THE LRM. REMIND THE STUDENTS THAT THEY ARE FAMILIAR WITH THE LANGUAGE HISTORY. WE AREN'T COVERING THAT. WE ARE SPECIFICALLY TALKING ABOUT THE LRM AND ITS DEVELOPMENT.

THE PURPOSE OF THIS SECTION IS TO GIVE THE STUDENTS AN IDEA OF HOW MUCH THOUGHT AND EFFORT WENT INTO ANALYZING THE LRM.

ALLOW 30 MINUTES FOR THIS SECTION.

Section 2

LANGUAGE REFERENCE MANUAL HISTORY

INSTRUCTOR NOTES

- CONTRARY TO POPULAR OPINION, ANSI DID NOT REJECT THE 1980 VERSION OF THE MANUAL. THEY CONDITIONALLY ACCEPTED IT WITH A STRONG RECOMMENDATION THAT IT BE REWRITTEN.

- THE AJPO (ADA JOINT PROGRAMMING OFFICE) DECIDED TO HAVE IT REWRITTEN DUE TO ANSI CONCERNS.

Ada LANGUAGE REFERENCE MANUAL HISTORY

- ORIGINALLY PUBLISHED IN JULY 1980
- REPRINTED NOVEMBER 1980
  - CORRECTION OF TYPOGRAPHICAL ERRORS
- CONDITIONALLY ACCEPTED AS AN ANSI STANDARD IN 1980
  - NEEDED TO IMPROVE READABILITY
  - NEEDED TO BE MORE RIGOROUS

INSTRUCTOR NOTES

- POINT OUT THAT "WITHOUT AFFECTING THE LANGUAGE" WASN'T EXACTLY SUCCESSFUL. WORDING CHANGES AND CLARIFICATIONS IN MANY CASES HAD GREAT IMPACT ON THE LANGUAGE.
- MANY OF THESE CHANGES ARE NOT NOTICEABLE TO THE AVERAGE USER.
- EXPLAIN MEANING (DIFFERENCES) BETWEEN THE REVIEWERS.
- REVIEWING MECHANISM WAS VIA THE ARPA-NET.
- REFER TO PRAGMA HANDOUT, THE NEXT FEW SLIDES REFER TO IT.
    - SUGGEST THEY READ IT AT THEIR LEISURE TO GET A FLAVOR OF THE DEBATES AND ANALYZATION THAT WENT INTO THE REWRITING.

THE LRM REWRITTEN

- TO INCORPORATE ANSI CONCERNS
- WITHOUT AFFECTING THE LANGUAGE
- APPOINTED THE LANGUAGE DESIGN TEAM (LDT)
  - HEADED BY JEAN D. ICHBIAH OF ALSYS
- REVIEWERS
  - DISTINGUISHED REVIEWERS (APPOINTED BY CONTRACT)
  - AUTHORIZED KIBITZERS (OTHER CONTRIBUTORS)
  - VOLUNTARY REVIEWERS (ADDED MUCH LATER)
- MECHANISM FOR REVIEWING
  - CHAPTER REVISIONS MADE AVAILABLE FOR REVIEW
  - COMMENTS AND SUGGESTIONS SENT TO THE LDT
    - SOME 4000 COMMENTS BY JULY 82
    - RECEIVED FROM ALL OVER THE WORLD

INSTRUCTOR NOTES

THE ENTIRE SERIES OF COMMENTS IS A HANDOUT FOR THE STUDENT. WHAT FOLLOWS IN THE NEXT 5 SLIDES IS A BRIEF SUMMARY. SUGGEST THE STUDENTS READ THE COMMENTS AT THEIR LEISURE.

EXAMPLE OF REVIEW COMMENTS

- A QUESTION WAS ASKED OF THE LDT, "CAN PRAGMAS EXTEND THE LANGUAGE"?
- THE LDT RESPONSE WAS TO REQUEST REVIEWERS TO SUPPLY SUGGESTED WORDINGS.
- WORDING SUGGESTED BY WRITERS OF THE IMPLEMENTERS' GUIDE, (ESSENTIALLY THAT THE LEGALITY AND MEANING OF A PROGRAM CANNOT BE AFFECTED BY THE PRESENCE OF A PRAGMA) CAUSED GREAT CONTROVERSY.

INSTRUCTOR NOTES

- ELABORATE VIOLATES THIS RULE BECAUSE IT COULD MAKE AN ERRONEOUS PROGRAM INTO A CORRECT PROGRAM; AND INTERFACE VIOLATES IT, BECAUSE WHEN PRESENT, A BODY SUPPLIED FOR THE SUBPROGRAM IS ILLEGAL, AND WHEN ABSENT A BODY MUST BE SUPPLIED.

- OTHER AREAS IN WHICH THE LANGUAGE COULD BE EXTENDED ARE
  - IMPLEMENTATION DEFINED ATTRIBUTES
  - ADDITIONS TO STANDARD AND/OR SYSTEM
  - ADDITIONS TO THE COMPILATION LIBRARY

VIEWPOINT 1

- ADA ITSELF VIOLATES THIS RULE WITH SOME OF ITS PREDEFINED PRAGMAS (I.E. ELABORATE AND INTERFACE)

- WHEN PRAGMA INCLUDE WAS DROPPED FROM THE LANGUAGE, PART OF THE JUSTIFICATION FOR DROPPING IT WAS THAT AN IMPLEMENTATION COULD ALWAYS ADD IT. THIS WORDING WOULD FORBID THIS PRAGMA.

- THE AREA OF PRAGMAS IS JUST ONE OF A FEW AREAS IN WHICH THE LANGUAGE COULD BE EXPANDED, WHY RESTRICT JUST PRAGMAS.

- WE HAVE TWO CONFLICTING GOALS.
  - MAXIMIZE PORTABILITY
  - EFFECTIVE USE OF ADA IN IMPLEMENTATION DEPENDENT APPLICATIONS

  THE WORDING MUST LIMIT ARBITRARY AND INESSENTIAL EXTENSIONS WHILE ALLOWING IMPLEMENTATION AND ENVIRONMENTAL NECESSARY EXTENSIONS.

- CONCLUSION: NOTHING SHOULD BE SAID IN THE LRM CONCERNING THIS ISSUE.

INSTRUCTOR NOTES

THIS IS ESSENTIALLY A RESPONSE TO VIEWPOINT 1.

IN POINT 3, MENTION THAT HAVING NO RESTRICTION WOULD ALLOW PROGRAMS FOR INLINE INSERTION OF ASSEMBLY CODE, PRAGMA CHARACTER_SET, PRAGMA TRANSLITERATE, ETC.

VIEWPOINT 2

- THE RULE APPLIES TO IMPLEMENTATION-DEFINED PRAGMAS

- INCLUDE A PREPROCESSOR FUNCTION, NOT A LANGUAGE FUNCTION

- PRAGMAS MUST BE RESTRICTED OR PROGRAMS SUCH AS

```
PRAGMA      WINK_WINK_WINK;
            DIMENSION IARR (10)
            READ (5, 100) IARR

100         FORMAT (10I5)
            DO 200 I = 1, 10
            IF (IARR(I).GE.34) GO TO 200
            WRITE (6, 100) IARR(I)
200         CONTINUE
C           WHO SAID LEARNING ADA WAS GONNA BE HARD?
```

  BECOMES A LEGAL ADA PROGRAM.

- AGREE THAT A MIDDLE GROUND MUST BE FOUND BETWEEN OUR CONFLICTING GOALS

- CONCLUSION: SOMETHING MUST BE SAID IN THE LRM

THE CONTROVERSY CONTINUED ...

THIS SUMMARIZES THE MAJOR POINT OF RON BRENDER COMMENT #3831.

NOTE THAT THE DRAFT CHAPTERS CHANGED AND SOME OF THE WORDING WAS INCORPORATED TO SAY THAT AN IMPLEMENTATION DEFINED PRAGMA MAY NOT EFFECT THE LEGALITY OF AN ADA PROGRAM.

VIEWPOINT 1

- LEGALITY SHOULD NOT BE USED TO DECIDE IF A PRAGMA IS ALLOWED.
  - CAN ARGUE THAT ALL PRAGMAS, USED IN A CERTAIN WAY, WILL AFFECT THE MEANING (RESULT) OF THE PROGRAM.

- PRAGMAS ARE SUPPOSED TO INFORM THE COMPILER ABOUT THE EXECUTION ENVIRONMENT.
  - CURRENT PRAGMAS MUST BE MODIFIED TO DO THIS OR ONE MUST BE DEFINED TO DO SO. BOTH METHODS ARE NOT ALLOWED BY THE CURRENT WORDING.

- THE UTILITY AND APPLICABILITY OF THE ADA LANGUAGE WILL BE SEVERELY CONSTRAINED.

- WITHOUT A CHANGE TO THIS WORDING, THE LANGUAGE IS NOT AN ACCEPTABLE ANSI STANDARD.

THE CONTROVERSY CONTINUED AFTER THIS UNTIL THE LDT (JEAN D. ICHBIAH) MADE A DECISION.

VIEWPOINT 2

- LEGALITY CRITERION IS NOT PERFECT BUT IT'S BETTER THAN NOTHING.
  - ORIGINALLY THE WORDING HAD BEEN "LEGALITY OR MEANING" BUT THIS WAS TOO VAGUE A CONCEPT TO SUPPORT.

- THE GOAL IS NOT TO ENFORCE PORTABILITY, BUT TO PREVENT THE LRM FROM BECOMING A MOCKERY.
  - NO RESTRICTION WOULD BE EQUIVALENT TO APPENDING "IN THE ABSENCE IMPLEMENTATION DEFINED PRAGMAS" TO EVERY RULE IN THE LRM.

- NOT ACCEPTABLE STANDARD?
  - SEMANTICALLY, THE LANGUAGE DOES NOT NEED PRAGMAS.
    - THEY ARE A MATTER OF CONVENIENCE.
    - THEY ARE NOT WORTH COMPROMISING THE INTEGRITY OF THE LANGUAGE.

INSTRUCTOR NOTES

THE FINAL WORDING OF THE STANDARD REMAINED ESSENTIALLY UNCHANGED.

REMIND THE STUDENT THAT THIS WAS JUST AN EXAMPLE OF ONE ISSUE IN WHICH COMMENTS WERE MADE.

CONCLUSION

- LDT POSITION
  - ANY COMPILER MUST BE ABLE TO CHECK THE LEGALITY OF ANY ADA PROGRAM.
  - UNCONVINCED BY THE ARGUMENTS THAT PRAGMAS ARE NECESSARY.
  - AGREE THAT WE WANT TO ALLOW PRAGMAS WHOSE EFFECT IS TO MODIFY TEXT, THE LEGALITY CHECK COULD BE APPLIED AFTER EXPANSION.
  - NO WATERTIGHT WORDING FOR THIS RELAXATION, SO LEAVE SUCH PRAGMAS AS PREPROCESSOR DIRECTIVES.

"MOSTLY ON TYPOGRAPHICAL ERROR" BUT NOT ALL. THERE ARE, AS STATED PREVIOUSLY, SOME VERY SUBTLE SEMANTIC DIFFERENCES.

- DRAFT PUBLISHED IN JULY 1982
  - RECEIVED ABOUT 1000 ADDITIONAL COMMENTS
    - MOSTLY ON TYPOGRAPHICAL ERRORS
- PRINTERS COPY, NOVEMBER 1982
  - NOT AVAILABLE TO THE PUBLIC
  - SENT TO REVIEWERS
  - SENT TO ANSI
- MADE OFFICIAL ANSI STANDARD ON FEB 17, 1983

INSTRUCTOR NOTES

THE ADA BOARD MEMBERS REPRESENT THE GOVERNMENT, INDUSTRY, ACADEMIA, AND EUROPEAN INTERESTS. THE MAJOR FUNCTION IS TO ADVISE THE AJPO DIRECTOR.

THE ADA BOARD IS MENTIONED TO MAKE STUDENTS AWARE THAT THE LANGUAGE MAINTENANCE COMMITTEE ANSWERS QUESTIONS AND SUPPLIES ACCEPTED INTERPRETATIONS OF THE LRM.

THE VALIDATION COMMITTEE ADVISES ON VALIDATION POLICY ISSUES. THE STANDARDS COMMITTEE IS INVOLVED WITH THE VARIOUS STANDARD ORGANIZATIONS SUCH AS MILITARY STANDARDS, ANSI, ISO, ETC. THE ORGANIZATION INTERFACES COMMITTEE IS INVOLVED IN PUBLIC RELATIONS.

THE ENVIRONMENTAL COMMITTEE IS INVOLVED WITH ISSUES RELATED TO THE APSE.

THE LANGUAGE MAINTENANCE COMMITTEE CAN BE REFERENCED DIRECTLY. THE LAST PAGE OF THE LRM GIVE DIRECTIONS FOR THIS.

THE ADA BOARD

- TECHNICAL ADVISORY BOARD TO THE DIRECTOR OF THE AJPO.
- CONSISTS OF VARIOUS COMMITTEES
  - VALIDATION COMMITTEE
  - LANGUAGE MAINTENANCE COMMITTEE
  - STANDARDS COMMITTEE
  - ENVIRONMENT COMMITTEE
  - ORGANIZATION INTERFACES COMMITTEE
- CAN BE CONTACTED THROUGH THE AJPO.

INSTRUCTOR NOTES

THIS SECTION EXPLAINS THE STRUCTURE OF THE LRM. ITS PURPOSE IS TO MAKE THE STUDENTS AWARE OF THE DIFFERENT PARTS OF THE LRM AND, SPECIFICALLY, THEIR USEFULNESS IN READING THE LRM.

ALLOW 60 MINUTES FOR THIS SECTION.

BREAK HERE FOR 15 MINUTES.

Section 3

LRM STRUCTURE

INSTRUCTOR NOTES

- POINT OUT THAT UNDERSTANDING THE SYNTAX IS THE FIRST STEP IN INTERPRETING THE LANGUAGE. (BUT ONLY THE FIRST STEP.)

- "THE SYNTAX IS DEFINED TO BE CONTEXT-FREE ... "

SYNTAX NOTATION

- CONTEXT-FREE SYNTAX TOGETHER WITH CONTEXT-DEPENDENT REQUIREMENTS
- VARIANT OF BACKUS-NAUR-FORM
  - LOWER CASE WORDS DENOTE SYNTACTIC CATEGORIES
    - ITALICIZED PART OF A NAME IS USED TO CONVEY SEMANTIC MEANING
  - BOLD FACE WORDS DENOTE RESERVED WORDS
  - SQUARE BRACKETS [] ENCLOSE OPTIONAL ITEMS
  - BRACES { } ENCLOSE REPEATED ITEMS
  - VERTICAL BARS
    - SEPARATE ALTERNATIVE ITEMS

      EXAMPLE : NAME|{,NAME}
    - REPRESENTS ITSELF

      EXAMPLE : CHOICE {|CHOICE}
- SPECIFIED IN SECTION 1.5 OF THE LRM
- A SYNTAX SUMMARY IS LISTED IN APPENDIX E

INSTRUCTOR NOTES

POINT OUT THAT THE EXISTENCE OF { BEFORE THE elseif AND } AFTER Sequence_Of Statements INDICATES THAT THIS SECTION OF THE STATEMENT MAY BE REPEATED ZERO OR MORE TIMES.

POINT OUT THAT THE EXISTENCE OF [BEFORE THE else AND] AFTER Sequence_Of_Statements INDICATES THAT THIS PART OF THE if STATEMENT IS OPTIONAL.

NOTE THAT else CANNOT BE REPEATED.

POINT OUT THE RESERVED WORDS.

CONDITION AND Sequence_Of_Statements ARE OTHER SYNTATIC CATEGORIES.

RESERVE WORDS ARE UNDERLINED HERE. IN THE LRM THEY WOULD BE BOLD FACED.

EXAMPLE : SYNTAX OF if STATEMENT

```
if_statement ::=

      if Condition then

            Sequence_Of_Statements

      { elsif  Condition  then

            Sequence_Of_Statements }

      [ else

            Sequence_Of_Statements]

      end if;
```

INSTRUCTOR NOTES

- ALLOW 10 MINUTES FOR THIS EXERCISE.

- THE ANSWER TO QUESTION 1 IS YES

    - THE WORD constant IS OPTIONAL
    - My_Tree IS AN IDENTIFIER, WHICH CAN BE AN Identifier_List
    - THE : IS PRESENT
    - Tree range Redwood .. Pine IS A VALID SUBTYPE INDICATION
    - THE INITIAL EXPRESSION IS OPTIONAL

- THE ANSWER TO QUESTION 2 IS NO.

    - (On, Off) IS NOT A VALID SUBTYPE INDICATION. SHOULD STUDENTS NEED A DEFINITION OF SUBTYPE INDICATION, HAVE THEM LOOK IN THE SYNTAX SUMMARY, APPENDIX E.

EXAMPLE PROBLEM

ACCORDING TO THE FOLLOWING SYNTAX RULES:

```
object_declaration ::=
    identifier_list : [constant] subtype_indication
                      [ := expression];

identifier_list ::=
    identifier {, identifier}
subtype_indication ::= Type_Mark [constraint]
```

1. IS

   ```
   My_Tree : Tree range Redwood .. Pine;
   ```

   SYNTACTICALLY LEGAL?

2. IS

   ```
   Switch : (On, Off);
   ```

   SYNTACTICALLY LEGAL?

INSTRUCTOR NOTES

ALLOW 20 MINUTES FOR STUDENTS TO ATTEMPT PROBLEMS AND SOLUTIONS TO BE DISCUSSED.

ANSWERS:

1. SYNTACTICALLY LEGAL, SPECIFICATION IN 7.1

   - basic_declarative_item CAN APPEAR ZERO OR MORE TIMES, IN THIS CASE ZERO.
   - private IS OPTIONAL, IN THIS CASE ALLOWED.
   - AGAIN basic_declarative_item CAN APPEAR ZERO OR MORE TIMES.
   - end NEED NOT BE FOLLOWED BY THE PACKAGE NAME.

2. SYNTACTICALLY ILLEGAL, SPECIFICATION IN 2.4.1

   - THE OPTIONAL UNDERLINE MUST BE FOLLOWED BY A DIGIT, . (PERIOD) IS NOT A DIGIT
   - . (PERIOD) MUST BE PRECEDED BY AN INTEGER, 3_ IS NOT A VALID INTEGER

3. SYNTACTICALLY LEGAL, SPECIFICATION IN 4.1.4

   - Integer IS AN identifier, WHICH IS A simple_name, WHICH IS A name, WHICH IS A prefix

   - Range IS A VALID attribute_designator

     NOTE THAT THIS IS ONLY SYNTACTICALLY CORRECT! 'RANGE CAN ONLY BE APPLIED TO AN ARRAY TYPE. IT IS SEMANTICALLY INCORRECT.

4. SYNTACTICALLY LEGAL, SPECIFICATION IN 3.2

   - string_object IS A LEGAL identifier_list
   - : IS PRESENT
   - String IS A LEGAL subtype_indication

   NOTE THAT THIS ALSO IS ONLY SYNTACTICALLY LEGAL. IT IS SEMANTICALLY ILLEGAL BECAUSE OBJECTS MUST BE CONSTRAINED!

**POINT OUT THAT ONE OF THE REASONS WE ARE STUDYING THE LRM IS BECAUSE SYNTAX ALONE DOES NOT SPECIFY THE LEGALITY OF A STATEMENT.

EXERCISES

USING APPENDIX E DETERMINE WHETHER OR NOT THE FOLLOWING ARE SYNTACTICALLY LEGAL.

1. `package P is private end;`

2. `3_.105`

3. `Integer'Range`

4. `String_Object : String;`

INSTRUCTOR NOTES

- BY LANGUAGE TERMS WE MEAN WORDS WHICH HAVE A SPECIAL MEANING IN Ada.
- POINT OUT THAT WHEN THE ONES NOT EXPLICITLY DEFINED ARE USED IN LANGUAGE DEFINITIONS, THE LANGUAGE DEFINITIONS BECOME OPEN TO INTERPRETATION. THIS IS OFTEN WHERE LANGUAGE ISSUES ARISE.
- APPENDIX D IS NOT PART OF THE STANDARD (LRM 1.2/4).

LANGUAGE TERMS

- WORDS WITH A VERY SPECIFIC MEANING IN Ada
- SOME LANGUAGE TERMS ARE NOT EXPLICITLY DEFINED
  - THE DEFINITION IS UNCLEAR OR AMBIGUOUS
  - THE MEANING IS OPEN TO INTERPRETATION
- MOST ARE LISTED IN THE GLOSSARY, APPENDIX D

INSTRUCTOR NOTES

- THESE ARE EXAMPLES. EACH OF THESE HAS A VERY EXPLICIT MEANING. AS WE GO THROUGH THE CHAPTERS OF THE LRM, THE MEANING OF THESE TERMS WILL BE EXPLAINED.

LANGUAGE TERM EXAMPLES

PROGRAM

ERRONEOUS

SCOPE

NAME

DENOTE

ELABORATION

APPROPRIATE FOR

INSTRUCTOR NOTES

- IN WRITING THE MANUAL, EXAMPLES AND NOTES WERE USED TO ILLUSTRATE THE INTENT OF THE RULES STATED IN THE SECTION OR CHAPTER.

- SOMETIMES THE INTENT IS CONTRADICTED ELSEWHERE FOR SPECIAL CASES. FOR THIS REASON NOTES SHOULD NOT BE RELIED ON.

NOTES AND EXAMPLES

- ARE SCATTERED THROUGHOUT THE LRM
- ARE NOT CONSIDERED TO BE PART OF THE LANGUAGE DEFINITION
- THEIR PURPOSE IS TO CLARIFY THE RULES STATED
- WHEN THEY CONFLICT WITH WHAT IS STATED ELSEWHERE IN THE LRM, THEY ARE INCORRECT
- SHOULD NOT BE RELIED ON WHEN RESOLVING LANGUAGE ISSUES

INSTRUCTOR NOTES

- STRESS THAT THE REFERENCES ARE A VERY GOOD SOURCE FOR TRACING DOWN INFORMATION
- THEY TIE THE LRM TOGETHER
- HAVE THE STUDENTS OPEN TO SECTION 3.2 AND LOOK AT THE REFERENCES. MAKE SURE THEY KNOW WHAT THE REFERENCES ARE.

REFERENCES

- LISTED AT THE END OF EVERY SECTION
- INFORM THE USER WHERE FURTHER INFORMATION ON TOPICS DISCUSSED IN THAT SECTION CAN BE FOUND
- THE PRIMARY MEANS OF TRACING THROUGH THE LRM IN SEARCH OF ANSWERS
- EXAMPLE:

  IN SECTION 3.2 ON OBJECT DECLARATIONS THE SECTION TITLED References AT THE END OF SECTION, LISTS 3.5.4 AS THE SECTION TO LOOK IN FOR MORE INFORMATION ON Universal_Integer

INSTRUCTOR NOTES

THESE ARE AT THE END OF THE LRM. STRESS THAT D, E, AND F ARE NOT PART OF THE LANGUAGE DEFINITION.

ANNEXES AND APPENDICES

ANNEX A	PREDEFINED LANGUAGE ATTRIBUTES
ANNEX B	PREDEFINED LANGUAGE PRAGMAS
ANNEX C	PREDEFINED LANGUAGE ENVIRONMENT
APPENDIX D	GLOSSARY
APPENDIX E	SYNTAX SUMMARY AND SYNTAX CROSS REFERENCE
APPENDIX F	IMPLEMENTATION-DEPENDENT CHARACTERISTICS

NOTE : APPENDICES ARE NOT PART OF THE LANGUAGE DEFINITION

INSTRUCTOR NOTES

- HAVE STUDENTS OPEN TO ANNEX A
- POINT OUT THE USE OF "DENOTES" AND "IS APPROPRIATE FOR"
- "FOR A PREFIX P THAT DENOTES" MEANS "FOR A PREFIX P THAT NAMES OR IS". (LRM SECTION 3.1)
- A PREFIX IS DEFINED TO BE "APPROPRIATE FOR" A TYPE IF
  - THE TYPE OF THE PREFIX IS THE TYPE CONSIDERED
  - THE TYPE OF THE PREFIX IS AN ACCESS TYPE WHOSE DESIGNATED TYPE IS THE TYPE CONSIDERED (LRM SECTION 4.1)

  NOTE: THE PREFIX MUST BE SOMETHING WHICH HAS A TYPE, i.e., AN OBJECT
- THEY WILL SEE THESE TERMS AGAIN, AND AGAIN ...

ANNEX A - PREDEFINED LANGUAGE ATTRIBUTES

- SUMMARIZES THE DEFINITIONS OF THE LANGUAGE DEFINED ATTRIBUTES
- SPECIFIES LEGAL PREFIXES FOR EACH ATTRIBUTE
- SPECIFIES THE RESULT VALUE AND TYPE FOR EACH ATTRIBUTE

INSTRUCTOR NOTES

HAVE STUDENTS NOW TURN TO ANNEX B.

ANNEX B - PREDEFINED LANGUAGE PRAGMAS

- SUMMARIZES THE RULES FOR THE LANGUAGE DEFINED PRAGMAS
- STATES THE OBJECTIVE OF EACH PRAGMA
- EXPLAINS THE EFFECT OF EACH PRAGMA
- SPECIFIES THE ARGUMENT, IF ANY

INSTRUCTOR NOTES

- HAVE THE STUDENTS TURN TO ANNEX C.

- ANNEX C DOES NOT CONTAIN THE SPECIFICATION OF THESE OTHER LANGUAGE DEFINED LIBRARY UNITS. THEY ARE SCATTERED THROUGHOUT THE LRM.

ANNEX C - PREDEFINED LANGUAGE ENVIRONMENT

- OUTLINES THE SPECIFICATION FOR PACKAGE STANDARD
- LISTS (IN A NOTE) OTHER LANGUAGE DEFINED LIBRARY UNITS

INSTRUCTOR NOTES

- HAVE THE STUDENTS OPEN TO THE GLOSSARY WHILE DISCUSSING THE EXAMPLE.

APPENDIX D - GLOSSARY

- NOT PART OF THE LANGUAGE DEFINITION
- DEFINES CERTAIN LANGUAGE TERMS
- NOT COMPLETE
  - FOR INSTANCE Homograph IS NOT LISTED
- OFTEN UNHELPFUL
  - Denote POINTS TO Declaration, WHICH USES Denote IN THE DEFINITION INSTEAD OF DEFINING IT. THE MEANING OF Denote IS MADE CLEARER BY THE DEFINITION OF Name.

INSTRUCTOR NOTES

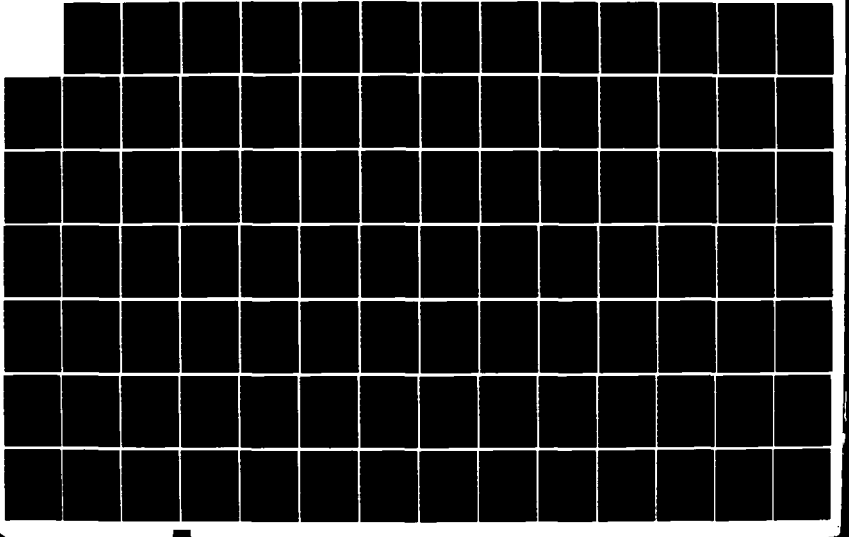
HAVE STUDENTS NOW TURN TO APPENDIX E.

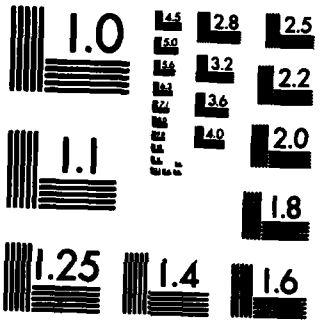
APPENDIX E - SYNTAX SUMMARY AND SYNTAX CROSS REFERENCE

- NOT PART OF THE LANGUAGE DEFINITION
- SYNTAX SUMMARY
  - LISTS CONSTRUCTS IN THE ORDER THEY APPEAR IN THE LRM
  - HARD TO USE UNTIL YOU KNOW WHICH CONSTRUCTS APPEAR IN WHICH SECTIONS
- SYNTAX CROSS REFERENCE
  - LISTS THE SECTION IN WHICH SYNTACTIC ITEMS APPEAR
  - SHOULD BE USED IN CONJUNCTION WITH THE SYNTAX SUMMARY

AD-A143 502 ADA (TRADEMARK) TRAINING CURRICULUM USING THE ADA LANGUAGE REFERENCE MANUAL L402 TEACHER'S GUIDE(U) SOFTECH INC WALTHAM MA JUL 84 DAAB07-83-C-K514 2/5

UNCLASSIFIED F/G 9/2 NL





INSTRUCTOR NOTES

- HAVE THEM TURN TO APPENDIX F.
- SUGGEST THAT THE STUDENTS GET AN APPENDIX F FOR THEIR IMPLEMENTATION.
- AS AN EXAMPLE TO THE SECOND POINT, WESTERN DIGITAL RECENTLY PUBLISHED AND DISTRIBUTED THE LRM WITH APPENDIX F FOR THEIR IMPLEMENTATION.

APPENDIX F - IMPLEMENTATION-DEPENDENT CHARACTERISTICS

- NOT PART OF THE STANDARD
- EACH IMPLEMENTATION MUST SUPPLY ONE
- CONTAINS

  IMPLEMENTATION DEFINED PRAGMAS
  IMPLEMENTATION DEFINED ATTRIBUTES
  SPECIFICATION OF PACKAGE SYSTEM
  RESTRICTIONS ON REPRESENTATION CLAUSES
  RESTRICTIONS ON UNCHECKED CONVERSION
  SPECIFICS OF I/O PACKAGES

- SHOULD BE REFERRED TO BECAUSE THE ANSWER TO MANY QUESTIONS WHICH ARISE WILL BE IMPLEMENTATION DEFINED

ALLOW 5 MINUTES FOR THIS EXERCISE.

ANSWER:

OBJECTS, PROGRAM UNITS, LABELS, AND ENTRIES CAN HAVE THE ATTRIBUTE ADDRESS APPLIED.

TYPES AND SUBTYPES CANNOT

ANSWER IS IN ANNEX A.

EXAMPLE PROBLEM

LIST THE ENTITIES THAT THE ATTRIBUTE Address CAN BE APPLIED TO?

LIST THE ENTITIES THAT IT CANNOT BE APPLIED TO?

ALLOW 20 MINUTES FOR THESE EXERCISES.

ANSWERS

1. NO, IT MUST BE AN OBJECT OF A TYPE WITH DISCRIMINANTS OR A PREFIX THAT DENOTES A PRIVATE TYPE OR SUBTYPE. ARRAYS ARE NOT ONE OF THESE.

2. YES, THE VALUE WOULD REPRESENT THE LOWER BOUND OF THE DESIGNATED INDEX (THE FIRST IF NONE IS SPECIFIED).

3. NO, THE VALUE WOULD BE MEANINGLESS. THERE IS NO FIRST VALUE OR LOWER BOUND FOR RECORDS.

4. NO, 'BASE MAY ONLY BE USED IN CONJUNCTION WITH OTHER ATTRIBUTES.

5. ADDRESS
   BASE
   CALLABLE
   COUNT
   LAST_BIT (IF THE COMPONENT IS A TASK)
   POSITION (IF THE COMPONENT IS A TASK)
   SIZE
   STORAGE_SIZE
   TERMINATED

6. CONSTRAINT_ERROR

7. NO

8. NO

9. SECTIONS 5.6, 11.1, 7.1, 8.5, 6.3, AND 9.1

EXERCISES

1. CAN THE ATTRIBUTE Constrained BE APPLIED TO A PREFIX THAT DENOTES AN ARRAY TYPE?

2. CAN THE ATTRIBUTE First BE APPLIED TO AN OBJECT OF AN ARRAY TYPE? IF SO, WHAT WOULD THE VALUE THAT IS RETURNED REPRESENT?

3. CAN THE ATTRIBUTE First BE APPLIED TO A PREFIX THAT DENOTES A RECORD TYPE? IF SO, WHAT WOULD THE VALUE THAT IS RETURNED REPRESENT?

4.
```
subtype Int is Integer range 1 .. 10;
X : Int'BASE;     -- legal?
```

5. LIST THE ATTRIBUTES WHICH APPLY TO TASKS.

6. WHAT IS THE FINAL RESULT?

```
declare
   type Week is (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
   Day : Week;
begin
   Day := Week'Pred (Week'Val (Week'Pos (Week'Last) +1));
end;
```

7. IS Main A PREDEFINED PRAGMA?

8. CAN THE PRAGMA Shared BE APPLIED TO A CONSTANT?

9. IN WHAT SECTIONS DOES THE SYNTACTIC ELEMENT exception APPEAR?

INSTRUCTOR NOTES

THIS IS THE REAL MEAT OF THE COURSE. SOME CHAPTERS, DEPENDING ON TIME CAN BE SKIPPED. REMIND THEM THAT BECAUSE OF THE TIME, WE ARE NOT GOING INTO DEPTH, BUT ARE HIGHLIGHTING IMPORTANT POINTS IN EACH CHAPTER.

WE START WITH CRITICAL CHAPTERS, ON WHICH OTHER CHAPTERS DEPEND.

Section 4

CHAPTER TOPICS

INSTRUCTOR NOTES

THESE ARE THE LRM CHAPTERS. THE TITLES ARE AS THEY APPEAR IN THE LRM. WE WILL BE COVERING THE CHAPTERS IN THIS ORDER.

IN EACH CHAPTER, WE WILL PRESENT THE TOPICS COVERED IN THAT CHAPTER, HIGHLIGHT IMPORTANT INFORMATION, PRESENT AN EXAMPLE PROBLEM AND SOLUTION (FOR CHAPTERS 2 THROUGH 14), AND ASSIGN CLASS WORK PROBLEMS.

IMPORTANT LANGUAGE TERMS WILL BE PRESENTED AS THEY ARE ENCOUNTERED.

SHOULD GET THROUGH CHAPTER 3 THE FIRST DAY AND THROUGH CHAPTER 12 BY LUNCH THE SECOND DAY.

CHAPTER TOPICS

CHAPTER 1 - INTRODUCTION

CHAPTER 2 - LEXICAL ELEMENTS

CHAPTER 8 - VISIBILITY RULES

CHAPTER 3 - DECLARATIONS AND TYPES

CHAPTER 4 - NAMES AND EXPRESSIONS

CHAPTER 7 - PACKAGES

CHAPTER 6 - SUBPROGRAMS

CHAPTER 9 - TASKS

CHAPTER 12 - GENERIC UNITS

CHAPTER 11 - EXCEPTIONS

CHAPTER 10 - PROGRAM STRUCTURE AND COMPILATION ISSUES

CHAPTER 13 - REPRESENTATION CLAUSES AND IMPLEMENTATION-DEPENDENT FEATURES

CHAPTER 5 - STATEMENTS

CHAPTER 14 - INPUT-OUTPUT

INSTRUCTOR NOTES

THIS CHAPTER IS COVERED FIRST BECAUSE IT DEFINES MANY TERMS WHICH ARE USED THROUGHOUT THE LRM. IT IS IMPORTANT TO UNDERSTAND THESE TERMS.

ALLOW 20 MINUTES FOR THIS SECTION.

SECTION 5

LRM - CHAPTER 1

INTRODUCTION

THESE ARE THE TOPICS IN CHAPTER 1.

1.6 IS THE IMPORTANT SECTION. THE ERROR CLASSIFICATIONS ARE REFERRED TO THROUGHOUT THE LRM.

SECTION TOPICS

1.1 SCOPE OF THE STANDARD

1.1.1 EXTENT OF THE STANDARD

1.1.2 CONFORMITY OF AN IMPLEMENTATION WITH THE STANDARD

1.2 STRUCTURE OF THE STANDARD

1.3 DESIGN GOALS AND SOURCES

1.4 LANGUAGE SUMMARY

1.5 METHOD OF DESCRIPTION AND SYNTAX NOTATION

1.6 CLASSIFICATION OF ERRORS

INSTRUCTOR NOTES

"A CONFORMING COMPILER MUST ..."

THESE POINTS ARE HIGHLIGHTED BECAUSE THE STUDENTS SHOULD BE MADE AWARE AGAIN THAT A COMPILER MUST CONFORM TO THE STANDARD TO BE VALIDATED. THEY ARE TO CODE TO THE STANDARD, NOT THE COMPILER.

POINT OUT THAT ON THE THIRD BULLET, THEY ARE NOT REQUIRED TO REJECT PROGRAMS WITH ERRORS THAT THEY ARE NOT REQUIRED TO DETECT.

POINT OUT THE FIFTH BULLET. THE LRM DEFINES WHAT IS PERMITTED.

DEFINITION OF A CONFORMING COMPILER

- CORRECTLY TRANSLATE AND EXECUTE LEGAL Ada PROGRAMS
- REJECT PROGRAMS WHICH EXCEED THE IMPLEMENTATION'S CAPACITY
- REJECT PROGRAMS CONTAINING ERRORS WHICH THE IMPLEMENTATION IS REQUIRED TO DETECT
- SUPPLY ALL *PREDEFINED PROGRAM UNITS*
- CONTAIN NO UNPERMITTED VARIATIONS
- SPECIFY ALL PERMITTED VARIATIONS IN THE MANNER PRESCRIBED BY THE LRM (i.e., LISTS IN APPENDIX F)

INSTRUCTOR NOTES

"THESE ARE THE PROBLEM AREAS THE LDT TRIED TO KEEP IN MIND WHILE DESIGNING THE LANGUAGE."

NOTE THAT THESE GOALS ARE LISTED IN PRIORITY ORDER

MENTION THAT ONE OF THE LDT GOALS IN WRITING THE LRM (AND THE REASON FOR MUCH OF ITS FORMAT) WAS TO MINIMIZE FORWARD REFERENCES.

DESIGN GOALS

TO DEFINE A LANGUAGE WHICH SUPPORTS

- PROGRAM RELIABILITY AND MAINTENANCE
- PROGRAMMING AS A HUMAN ACTIVITY
- EFFICIENCY

INSTRUCTOR NOTES

COMPILE TIME ERRORS - ERRORS WHICH MUST BE CAUGHT AT COMPILE TIME

EXECUTION ERRORS - ERRORS WHICH RAISE EXCEPTIONS

ERRONEOUS EXECUTION - FOR EXAMPLE, THE USE OF AN UNINITIALIZED OBJECT

THE REASON THIS IS HIGHLIGHTED IS BECAUSE "ERRONEOUS", "LEGAL", "ILLEGAL", "MUST", "IN SOME ORDER", ETC. ARE USED THROUGHOUT THE LRM. IT IS IMPORTANT TO KNOW WHAT IS MEANT BY THESE TERMS.

ERROR CATEGORIES

- COMPILE TIME ERRORS
  - VIOLATIONS OF RULES THAT USE THE WORDS must, allowed, legal AND illegal
- EXECUTION ERRORS
  - ERROR SITUATIONS THAT ARE ASSOCIATED WITH PREDEFINED EXCEPTIONS
- ERRONEOUS EXECUTION
  - VIOLATIONS OF RULES THAT THE COMPILER IS NOT REQUIRED TO DETECT. THE EXECUTION OF ERRONEOUS PROGRAMS IS UNPREDICTABLE
- INCORRECT ORDER DEPENDENCES
  - APPLY TO RULES WHICH STATE "IN SOME ORDER THAT IS NOT DEFINED BY THE LANGUAGE"
  - A CONSTRUCT IS IN ERROR IF THE MEANING DEPENDS ON THE ORDER OF EVALUATION

NOTE: THE IMPLEMENTATION IS ALLOWED (BUT NOT REQUIRED) TO DETECT ERRONEOUS EXECUTION AND INCORRECT ORDER DEPENDENCES

INSTRUCTOR NOTES

ALLOW 5 MINUTES FOR THIS EXERCISE.

IN THE EXAMPLE THE VALUE OF A DEPENDS ON THE ORDER IN WHICH THE PARAMETERS X AND Y ARE EVALUATED UPON RETURN FROM THE PROCEDURE CALL. IF X IS EVALUATED FIRST, A WILL HAVE THE VALUE 4. IF Y IS EVALUATED FIRST, A WILL HAVE THE VALUE 2.

IF A = 2 BEFORE THE CALL, THERE IS NO ORDER DEPENDENCE.

EXAMPLE PROBLEM

WHY IS THE FOLLOWING AN INCORRECT ORDER DEPENDENCY?

```
declare -- Block
    A: Integer := 0;
    procedure Proc (X: in out Integer;
                    Y: out    Integer) is
    begin -- Proc
       X := X + 2;
       Y := 4;
    end Proc;
begin -- Block
    Proc (A, A);  -- INCORRECT ORDER DEPENDENCY
end;  -- Block
```

INSTRUCTOR NOTES

NOTE THAT ILLEGAL PROGRAMS ARE ONLY PROGRAMS WHICH CONTAIN COMPILE-TIME ERRORS. SPECIFICALLY, A PROGRAM WHICH IS SURE TO RAISE AN EXCEPTION, OR THAT IS ERRONEOUS, IS NOT ILLEGAL.

IMPORTANT DEFINITIONS

LEGAL	-	PROGRAM WHICH CONTAINS NO COMPILE-TIME ERRORS
ILLEGAL	-	PROGRAM WHICH CONTAINS COMPILE-TIME ERRORS
ERRONEOUS	-	PROGRAM WHICH CONTAINS ERRORS THE COMPILER IS NOT REQUIRED TO DETECT
MUST	-	A DIRECTIVE, A LANGUAGE PRIMITIVE
ALLOWED	-	PERMISSION, A LANGUAGE PRIMITIVE

INSTRUCTOR NOTES

THIS CHAPTER IS NEXT BECAUSE THE RULES FOR LEXICAL ELEMENTS ARE BASIC AND REFERRED TO THROUGHOUT THE LRM. THIS IS AN EASY CHAPTER AND MAY BE SKIPPED IF YOU ARE BEHIND SCHEDULE.

ALLOW 10 MINUTES FOR THIS CHAPTER.

Section 6

LRM - CHAPTER 2

LEXICAL ELEMENTS

INSTRUCTOR NOTES

THESE ARE THE TOPICS IN THIS CHAPTER.

POINT OUT PRAGMAS AS A POINT OF INTEREST.

NORMALLY YOU WOULDN'T THINK OF LOOKING IN THE LEXICAL ELEMENTS CHAPTER FOR INFORMATION ON PRAGMAS.

SECTION TOPICS

INSTRUCTOR NOTES

ALLOW 5 MINUTES FOR THIS EXERCISE.

SECTION 2.6/3 STATES "IF A QUOTATION CHARACTER VALUE IS TO BE REPRESENTED IN THE SEQUENCE OF CHARACTERS VALUES, THEN A PAIR OF ADJACENT QUOTATION CHARACTERS MUST BE WRITTEN AT THE CORRESPONDING PLACE WITHIN THE STRING LITERAL."

SO THE FIRST " STARTS THE STRING, THE NEXT TWO REPRESENT THE SINGLE QUOTE IN THE STRING, AND THE LAST " CLOSES THE STRING LITERAL.

EXAMPLE PROBLEM

WHY IS "**** A STRING OF LENGTH 1?

CITE THE LRM PASSAGE WHICH SAYS THIS.

INSTRUCTOR NOTES

TELL STUDENTS, IF SHORT ON TIME, TO ATTEMPT 5 AND 6 FIRST. THEY ARE THE MORE INTERESTING OF THE PROBLEMS. THE FIRST FEW ENABLE THE STUDENTS TO FIND SUPPORT FOR THINGS THEY SHOULD ALREADY KNOW.

ANSWERS:

1. NO, IN 2.4.1/2 THE SYNTAX DOES NOT ALLOW FOR A SIGN TO PRECEDE THE DIGIT. (WE WILL SEE LATER THAT - IS A UNARY OPERATOR).

2. YES, 2.3/3 STATES "IDENTIFIERS DIFFERING IN ONLY THE USE OF CORRESPONDING UPPER AND LOWER CASE LETTERS ARE CONSIDERED AS THE SAME.

3. NO, SECTION 2.3/3 STATES "ALL CHARACTERS OF AN IDENTIFIER ARE SIGNIFICANT, INCLUDING ANY UNDERLINE CHARACTER..."

4. YES, SECTION 2.4.1/3 STATES "AN UNDERLINE CHARACTER INSERTED BETWEEN ADJACENT DIGITS OF A DECIMAL LITERAL DOES NOT AFFECT THE VALUE OF THIS NUMERIC LITERAL.

5. NO, IT MAY ISSUE A WARNING. SECTION 2.8/9 STATES "A PRAGMA THAT IS NOT LANGUAGE-DEFINED HAS NO EFFECT IF ITS IDENTIFIER IS NOT RECOGNIZED BY THE (CURRENT) IMPLEMENTATION." FURTHER MORE, SECTION 2.8/8 STATES "THE PRAGMAS DEFINED BY THE LANGUAGE ... MUST BE SUPPORTED BY EVERY IMPLEMENTATION." AN IMPLEMENTATION MUST RECOGNIZE LANGUAGE DEFINED PRAGMAS.

6. YES, SECTION 2.10/3 STATES "THE SHARP CHARACTER (#) OF A BASED LITERAL CAN BE REPLACED BY COLON (:)..."

EXERCISES

SUPPORT YOUR ANSWERS WITH RELEVANT LRM SECTION AND PARAGRAPH NUMBERS.

1. IS -3 AN INTEGER LITERAL?

2. ARE Integer, integer, AND INTEGER ALL CONSIDERED TO BE THE SAME IDENTIFIER?

3. ARE prime_number AND PRIMENUMBER CONSIDERED TO BE THE SAME IDENTIFIER?

4. DO 1_000 AND 1000 REPRESENT THE SAME NUMERIC VALUE?

5. IS AN IMPLEMENTATION ALLOWED TO REJECT PROGRAMS WHICH CONTAIN UNRECOGNIZABLE PRAGMAS?

6. IS AN IMPLEMENTATION ALLOWED TO ACCEPT A BASED NUMBER REPRESENTED AS 2:1011: ?

INSTRUCTOR NOTES

THIS CHAPTER IS VERY IMPORTANT. UNDER NO CIRCUMSTANCES IS IT TO BE SKIPPED. ALLOW 90 MINUTES FOR THIS CHAPTER.

THIS CHAPTER CONTAINS THE VISIBILITY RULES AND IS FUNDAMENTAL IN UNDERSTANDING THE REST OF THE LRM.

Section 7

LRM - CHAPTER 8

VISIBILITY RULES

INSTRUCTOR NOTES

THESE ARE THE SECTION TOPICS IN THIS CHAPTER. 8.3 IS VERY IMPORTANT.

SECTION TOPICS

8.1 DECLARATIVE REGION

8.2 SCOPE OF DECLARATIONS

8.3 VISIBILITY

8.4 USE CLAUSES

8.5 RENAMING DECLARATIONS

8.6 THE PACKAGE STANDARD

8.7 THE CONTEXT OF OVERLOAD RESOLUTION

INSTRUCTOR NOTES

THESE ENTITIES ARE SPECIFIED IN SECTION 8.2/3-8.

ENTITIES WITH AN EXTENDED SCOPE

- DECLARATION THAT OCCURS IN THE VISIBLE PART OF A PACKAGE
- ENTRY DECLARATIONS
- COMPONENT DECLARATIONS
- DISCRIMINANT SPECIFICATIONS
- PARAMETER SPECIFICATIONS
- GENERIC PARAMETER DECLARATIONS

INSTRUCTOR NOTES

THE DEFINITION OF A HOMOGRAPH IS IN SECTION 8.3/15. HAVE THE STUDENTS OPEN TO THAT PARAGRAPH AND READ IT. THEY WILL PROBABLY HAVE QUESTIONS, SUCH AS "WHAT DOES THIS MEAN?," NOW EXPLAIN THE VIEWGRAPH.

NEXT SLIDE DEFINES WHICH ENTITIES ARE OVERLOADABLE.

POINT OUT THAT HOMOGRAPHS MAY NOT BE DECLARED IN THE SAME DECLARATIVE PART.

SECTION 6.6 DEFINES PARAMETER AND TYPE PROFILE.

HOMOGRAPHS

- DECLARATIONS WITH THE SAME IDENTIFIER AND WHERE AT MOST ONE IS OVERLOADABLE

- DECLARATIONS WITH THE SAME IDENTIFIER, OPERATOR SYMBOL, OR CHARACTER LITERAL, AS WELL AS THE SAME PARAMETER AND RESULT PROFILE WHERE BOTH ARE OVERLOADABLE

INSTRUCTOR NOTES

THESE ARE THE ENTITIES WHOSE NAMES CAN BE OVERLOADED. THEY ARE SPECIFIED IN SECTION 8.7/1.

HAVE STUDENTS READ THIS PARAGRAPH. IT SHOULD BE ADEQUATELY CLEAR TO THE STUDENTS.

ENTITIES FOR WHICH OVERLOADING IS DEFINED

- SUBPROGRAMS
- ENUMERATION LITERALS
- OPERATORS
- SINGLE ENTRIES
- BASIC OPERATIONS
  - ASSIGNMENT
  - MEMBERSHIP TESTS
  - ALLOCATORS
  - THE LITERAL NULL
  - AGGREGATES
  - STRING LITERALS

INSTRUCTOR NOTES

ALLOW 5 MINUTES FOR THIS EXERCISE.

ANSWER:

SECTION 8.4/1-2 STATES THAT THE NAME FOLLOWING THE WORD use MUST DENOTE A PACKAGE.

- REFER STUDENTS TO HANDOUT FOR COMPILER'S ERROR REPORT.

EXAMPLE PROBLEM

WHY IS THE FOLLOWING ILLEGAL?

```
procedure Proc is
    type T is 1 .. 10;
    type B is new Boolean;
begin -- Proc
    null;
end Proc;

with Proc; use Proc;          -- illegal
procedure P is
begin -- P
    null;
end P;
```

ALLOW A HALF HOUR FOR THESE EXERCISES.

ANSWERS

1. SECTION 8.1/2 DEFINES A DECLARATIVE REGION TO BE "A SUBPROGRAM DECLARATION, A PACKAGE DECLARATION, A TASK DECLARATION, OR A GENERIC DECLARATION, TOGETHER WITH THE CORRESPONDING BODY, IF ANY." THE DECLARATIVE REGION FOR Obj IS THE PACKAGE P TOGETHER WITH THE BODY OF PACKAGE P.

2. SECTION 8.3/15 EXPLAINS HOMOGRAPH. BETWEEN 2 AND 1, 2 IS A HOMOGRAPH OF 1 BECAUSE THEY BOTH HAVE THE SAME IDENTIFIER AND OVERLOADING IS ALLOWED FOR A MOST ONE (IT IS ALLOWED FOR DECLARATION 1). BETWEEN 2 AND 3, 3 IS A HOMOGRAPH OF 2 FOR THE SAME REASON (OVERLOADING IS ALLOWED FOR DECLARATION 3). BETWEEN 3 AND 4, NEITHER IS A HOMOGRAPH OF THE OTHER. ALTHOUGH THEY HAVE THE SAME IDENTIFIER, BOTH ARE OVERLOADABLE. THEY WOULD ONLY BE HOMOGRAPHS IF THEY BOTH HAD THE SAME RESULT TYPE PROFILE. 3 HAS NO RESULT TYPE PROFILE AND 4'S RESULT TYPE PROFILE IS BOOLEAN. NOTE THAT IF DECLARATION 2 WERE REMOVED, THERE WOULD BE NO HOMOGRAPHS IN THE TEXT.

3. SECTION 8.5/2 STATES THAT THE TYPE OF AN OBJECT RENAMING DECLARATION MUST BE A TYPE_MARK. Integer range 1 .. 10 IS A SUBTYPE_INDICATION (SECTION 3.3.2) NOT A TYPE_MARK.

4. YES. SECTION 8.5/3 STATES THAT WHERE THE RENAMING DECLARATION IS VISIBLE, THE IDENTIFIER DENOTES THE RENAMED IDENTIFER. THE CONSTRAINTS ON THE RENAMED IDENTIFIER HAVE NOT CHANGED.

EXERCISES

1. DEFINE THE DECLARATIVE REGION FOR Obj

```
package Pack is
    package P is
       Obj : Integer range 5 .. 10;
    end P;
end Pack;

package body Pack is
    ...        -- some sequence of declarations
    package body P is
       ...
    end P;
end Pack;
```

2. IN THE FOLLOWING, WHICH F'S ARE HOMOGRAPHS?

```
package Pack is
    function F return Character;    -- 1
    package P is
       type F is (a, b, c);         -- 2
       package Q is
           procedure F;             -- 3
           function F return Boolean;      -- 4
       end Q;
    end P;
end Pack;
```

3. WHY IS THE FOLLOWING ILLEGAL?

```
Y : Integer range 1 .. 10;
X : Integer range 1 .. 10 renames Y;      -- illegal
```

4. IF THE RENAMING WERE

```
X : Integer renames Y;
```

IS X CONSTRAINED TO HAVE ONLY THE VALUES 1 THROUGH 10?

5. NO. GREEN IS AN OVERLOADED ENTITY. REFER TO THE DEFINITION OF HOMOGRAPHS. A HOMOGRAPH CAN ONLY OCCUR WHEN AT MOST ONE OF THE ENTITIES IS OVERLOADABLE.

EXERCISES

5. IS THE SECOND COMMENT CORRECT?

```
procedure R is
   package Traffic is
      type Color is (Red, Amber, Green);
      ...
   end Traffic;

package Water_Colors is
   type Color is (White, Red, Yellow, Green, Blue, Brown, Black);
   ...
end Water_Colors;

use Traffic;          -- Color, Red, Amber and Green are directly visible
use Water_Colors;     -- two homographs of Green are directly visible
...                   -- but Color is no longer directly visible
```

INSTRUCTOR NOTES

THESE ARE THE IMPORTANT LANGUAGE TERMS DEFINED IN THIS CHAPTER.

NOTICE THAT EXTENDED SCOPE IS NOT A TERM DEFINED BY THE LRM. IT IS IMPLICITLY DEFINED WHEN SECTION 8.2/2 OF THE LRM SAYS "FURTHERMORE, ... THE SCOPE OF THE DECLARATION EXTENDS BEYOND THE IMMEDIATE SCOPE."

IMPORTANT DEFINITIONS

LOCAL	-	A DECLARATION THAT OCCURS IMMEDIATELY WITHIN A DECLARATIVE REGION
GLOBAL	-	DECLARATIONS THAT OCCUR IN AN OUTER ENCLOSING DECLARATIVE REGION
IMMEDIATE SCOPE	-	FROM THE BEGINNING OF THE DECLARATION TO THE END OF THE DECLARATIVE REGION
VISIBLE	-	ABLE TO BE ACCESSED DIRECTLY BY ITS SIMPLE NAME
HIDDEN	-	AN ENTITY IS HIDDEN WITHIN AN INNER REGION WHEN THE INNER REGION CONTAINS A HOMOGRAPH OF THE ENTITY

INSTRUCTOR NOTES

THIS CHAPTER IS IMPORTANT. ALLOW 90 MINUTES FOR IT. THIS CHAPTER HAS A LOT OF SEMANTICS, NOT ALL OF WHICH WILL BE COVERED. FOR FURTHER INFORMATION ON THESE SUBJECTS, SEE THE IMPLEMENTERS' GUIDE.

Section 8

LRM - CHAPTER 3

DECLARATIONS AND TYPES

INSTRUCTOR NOTES

THESE ARE THE TOPICS COVERED IN THIS CHAPTER.

SECTION TOPICS

INSTRUCTOR NOTES

THESE ARE THE SUBSECTIONS OF SECTION 3.5.

SUBSECTIONS

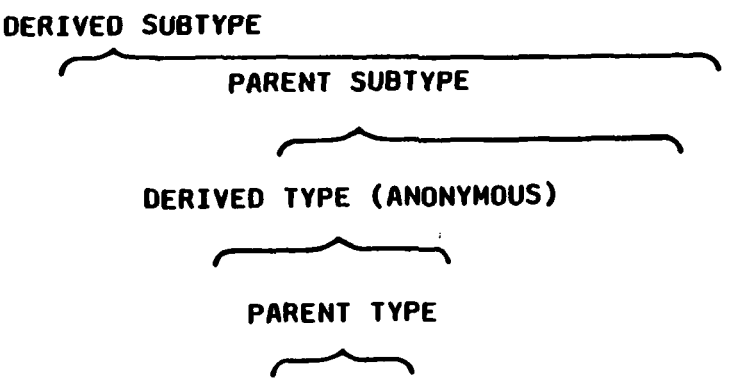
INSTRUCTOR NOTES

- STRESS THAT THE DERIVED TYPE IS ACTUALLY AN ANONYMOUS BASE TYPE AND THAT IN REALITY WHEN A DERIVED TYPE IS DEFINED, THE TYPE NAME ACTUALLY DENOTES A DERIVED SUBTYPE

TYPE DEFINITIONS

- PARENT TYPE - THE BASE TYPE FROM WHICH A TYPE IS DERIVED
- DERIVED TYPE - THE BASE TYPE OF THE DERIVED TYPE
- DERIVED SUBTYPE - A SUBTYPE OF THE DERIVED TYPE
- PARENT SUBTYPE - THE SUBTYPE INDICATION THAT OCCURS AFTER THE WORD NEW

EXAMPLE




```
type Int is new Integer range 10 .. 50;
```

INSTRUCTOR NOTES

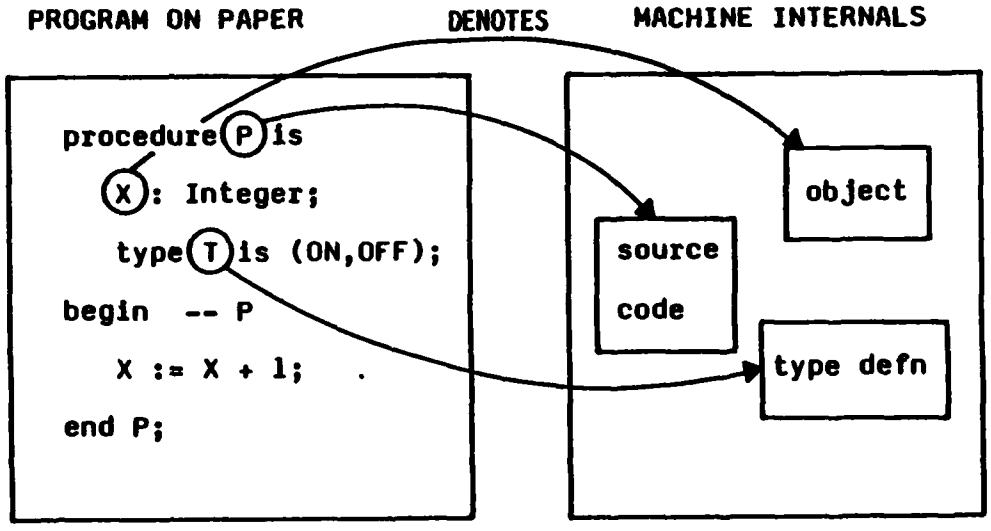
THE POINT OF THIS SLIDE IS TO EXPLAIN THAT NAME AND DENOTE ARE USED VERY DELIBERATELY IN THE LRM. WHEN NAME APPEARS IN THE LRM, IT IS REFERRING TO THE SYNTATIC CATEGORY.

DENOTE IS USED TO MEAN "NAME," "LABEL," OR "IDENTIFY" SOME ENTITY. THE WORDING CHOICE IS USED SPECIFICALLY TO AVOID CONFUSION.

- THE IDENTIFIERS P, T, AND X ARE NAMES WHICH DENOTE ENTITIES.
- THE NAME THAT APPEARS ON PAPER IN THE PROGRAM, "DENOTE" ENTITIES THAT ARE INTERNAL TO THE MACHINE.
- THE ARROW SHOWS THE "DENOTE" RELATIONSHIP.

NAME AND DENOTE

- NAME - SYNTATIC CATEGORY
- DENOTE - "THE NAME IS SAID TO DENOTE THE ASSOCIATED ENTITY"



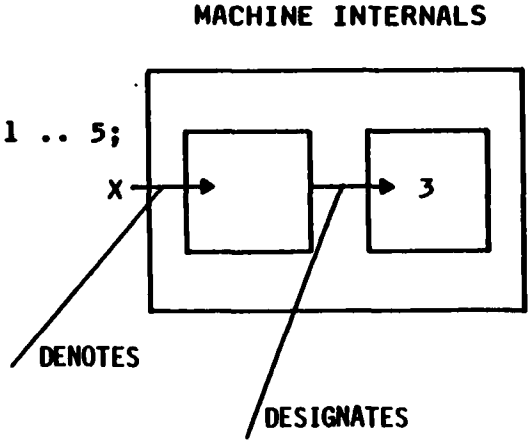
INSTRUCTOR NOTES

- X DENOTES A MEMORY LOCATION WHICH CONTAINS A VALUE WHICH DESIGNATES ANOTHER MEMORY LOCATION WHICH CONTAINS THE INTEGER VALUE 3.

DESIGNATED ENTITY

- DESIGNATE - POINTS TO
- DESIGNATED TYPE - BASE TYPE OF THE DESIGNATED OBJECT
- DESIGNATED SUBTYPE - SUBTYPE OF THE DESIGNATED OBJECT

```
Procedure Example is
  type acc_int is access integer range 1 .. 5;
  x : acc_int := new Integer'(3);
begin  -- Example
  null;
end Example;
```



ALLOW 5 MINUTES FOR THIS EXERCISE.

ANSWER:

SECTION 3.2.1/16 STATES "THE INITIALIZATION OF AN OBJECT CHECKS THAT THE INITIAL VALUE BELONGS TO THE SUBTYPE OF THE OBJECT; ... THE EXCEPTION CONSTRAINT_ERROR IS RAISED IF THIS CHECK FAILS." SO EVEN THOUGH THE COMPILER COULD CHECK THIS AT COMPILE TIME, IT IS ONLY ALLOWED TO ISSUE A WARNING AND MUST COMPILE THE PROGRAM.

- REFER STUDENTS TO HANDOUT FOR COMPILER'S ERROR REPORT.

EXAMPLE PROBLEM

WHY IS THE FOLLOWING

```
subtype Int is Integer range 1 .. 10;
X : Int := 30;          -- error
```

A RUNTIME ERROR?

INSTRUCTOR NOTES

ALLOW 45 MINUTES FOR THESE EXERCISES.
ANSWERS

1. YES, SECTION 3.2/8 STATES "A NUMBER DECLARATION IS A SPECIAL FORM OF OBJECT DECLARATION..."

2. SECTION 3.2/10 STATES "A MULTIPLE OBJECT DECLARATION IS EQUIVALENT TO A SEQUENCE OF THE CORRESPONDING NUMBER OF SINGLE OBJECT DECLARATIONS." THEREFORE, THE DECLARATION IS EQUIVALENT TO

```
X: array (1 .. 20) of Boolean;
Y: array (1 .. 20) of Boolean;
Z: array (1 .. 20) of Boolean;
```

   SECTION 3.6/6 STATES "A CONSTRAINED ARRAY DEFINITION DEFINES BOTH AN ARRAY TYPE AND A SUBTYPE OF THIS TYPE." EACH OF THE ABOVE IS A CONSTRAINED ARRAY DEFINITION, SO EACH DEFINES A NEW TYPE.

3. CONSTRAINT_ERROR BECAUSE SECTION 3.4/1 STATES THAT "A DERIVED TYPE DEFINITION DEFINES A NEW (BASE) TYPE WHOSE CHARACTERISTICS ARE DERIVED FROM THOSE OF A PARENT TYPE; THE NEW TYPE IS CALLED A DERIVED TYPE. A DERIVED TYPE DEFINITION FURTHER DEFINES A DERIVED SUBTYPE, WHICH IS A SUBTYPE OF THE DERIVED TYPE."

   THEREFORE, MY_INT IS A SUBTYPE OF SOME ANONYMOUS BASE TYPE WHICH HAS THE SAME SET OF VALUES AS INTEGER. 101 REPRESENTS A VALUE OF THAT TYPE. THEREFORE IT IS NOT A TYPE MISMATCH.

4. SECTION 3.4/15 STATES "IF A DERIVED OR PRIVATE TYPE IS DECLARED IMMEDIATELY WITHIN THE VISIBLE PART OF A PACKAGE, THEN WITHIN THIS VISIBLE PART, THIS TYPE MUST NOT BE USED AS THE PARENT TYPE OF A DERIVED TYPE DEFINITION."

5. F(3) IS NOT STATIC (SECTION 4.9 DEFINES STATIC), SECTION 3.5.4/3 STATES "IF A RANGE CONSTRAINT IS USED AS AN INTEGER TYPE DEFINITION, EACH BOUND OF THE RANGE MUST BE A STATIC EXPRESSION ..."

- REFER STUDENTS TO HANDOUT FOR COMPILER'S ERROR REPORT.

EXERCISES

1. IS A NAMED NUMBER CONSIDERED TO BE AN OBJECT?

2. WHY ARE

```
X, Y, Z : array (1 .. 20) of Boolean;
```

CONSIDERED TO BE OF THREE DIFFERENT TYPES

3. WHEN YOU HAVE

```
type My_Int is new Integer range 1 .. 100;
X : My_Int := 101;          -- error
```

DOES THIS RAISE CONSTRAINT_ERROR OR IS IT A TYPE MISMATCH CAUGHT AT COMPILE TIME?

4. WHY IS THE FOLLOWING ILLEGAL?

```
package Pack is
    type Color is (Red, Blue, Yellow, Green, Orange, Purple);
    type Room_Color is new Color range Red .. Orange;
    type Prime_Color is new Room_Color
       range Red .. Yellow;          -- illegal
end Pack;
```

5. WHY IS THE FOLLOWING ILLEGAL?

```
function F (I : Integer) return Integer is
   return I;
end F;
type Int is range 1 .. F(3);   -- illegal
```

ANSWERS (CONTINUED):

6. YES BECAUSE, THE BOUNDS OF A SUBTYPE INDICATION DO NOT NEED TO BE STATIC. NO REFERENCE REQUIRES IT TO BE, THEREFORE IT MAY BE Non_Static.

7. SECTION 3.5.9/15 STATES "MULTIPLICATION AND DIVISION OF FIXED POINT VALUES DELIVER RESULTS OF AN ANONYMOUS PREDEFINED FIXED POINT TYPE ... THE VALUES OF THIS TYPE MUST BE CONVERTED EXPLICITLY TO SOME NUMERIC TYPE."

8. SECTION 3.6/2 STATES THAT AN ARRAY TYPE DEFINITION MUST BE CONSTRAINED OR UNCONSTRAINED. SYNTACTICALLY, UNCONSTRAINED INDICES MAY NOT BE MIXED WITH CONSTRAINED INDICES IN THE SAME TYPE DECLARATION.

9. SECTION 3.7.1/9 STATES "DIRECT ASSIGNMENT TO A DISCRIMINANT OF AN OBJECT IS NOT ALLOWED..."

- REFER STUDENTS TO HANDOUT FOR COMPILER'S ERROR REPORT.

6. WHY DOES CHANGING THE DECLARATION TO

```
subtype Int is Integer range 1 .. F(3);
```

MAKE THE DECLARATION LEGAL?

7. WHY IS THE FOLLOWING A TYPE MISMATCH?

```
declare
    type Fixed is delta 0.0 range 0.0 .. 100.0;
    X,Y,Z : Fixed;
begin
    X := 3.0;
    Y := 7.0;
    Z := X * Y;                          -- type mismatch
end;
```

8. WHY IS THE FOLLOWING ILLEGAL?

```
type Arr is array (Integer range  <> , Boolean) of Character;
```

9. WHY IS THE FOLLOWING ILLEGAL?

```
declare
    type Rec (D : Integer := 6) is record
        Compl : Boolean;
        Comp2 : Character;
        Comp3 : Integer;
    end record;
    Rc : Rec;
begin
    Rc.D := 6;                           -- illegal
end;
```

INSTRUCTOR NOTES

THESE ARE THE IMPORTANT LANGUAGE TERMS DEFINED IN THIS CHAPTER. THEY ARE USED THROUGHOUT THE LRM. THESE DEFINITIONS ARE BRIEF INTUITIVE DEFINITIONS, NOT FORMAL ONES.

END OF DAY 1 - ANSWER QUESTIONS UNTIL CLASS IS DONE, IF TIME ALLOWS.

IMPORTANT DEFINITIONS

- BASE_TYPE - UNDERLYING ANONYMOUS "TYPE" OF ANY TYPE.
- UNIVERSAL_REAL - ANONYMOUS PREDEFINED REAL TYPE
- UNIVERSAL_INTEGER - ANONYMOUS PREDEFINED INTEGER TYPE
- UNIVERSAL_FIXED - ANONYMOUS PREDEFINED FIXED TYPE
- SCOPE - REGION OF TEXT IN WHICH AN ENTITY EXISTS
- ELABORATION - THE EXECUTION OF A DECLARATION
- ELABORATED - THE STATE OF A DECLARATION AFTER ELABORATION
- SATISFY - NOT VIOLATE, A LANGUAGE PRIMITIVE
- BELONGS TO - IS A POSSIBLE VALUE OF THE SUBTYPE
- COMPATIBLE - SATISFIES SOME CONDITION

INSTRUCTOR NOTES

THIS CHAPTER CONTAINS THE RULES FOR NAMES AND EXPRESSIONS. IT IS A FAIRLY IMPORTANT CHAPTER, BUT IT IS NOT VERY COMPLEX. ALLOW 30 MINUTES FOR THIS SECTION.

THE MAIN POINT TO MAKE IN THIS CHAPTER IS THAT MANY RULES EXIST TO MAKE THE LANGUAGE CONSISTENT.

Section 9

LRM - CHAPTER 4

NAMES AND EXPRESSIONS

INSTRUCTOR NOTES

THESE ARE THE TOPIC SECTIONS IN THIS CHAPTER.

SECTION TOPICS

INSTRUCTOR NOTES

THESE ARE THE TOPICS IN SECTION 4.5.

SUBSECTIONS

4.5 OPERATORS AND EXPRESSION EVALUATION

4.5.1 LOGICAL OPERATORS AND SHORT-CIRCUIT CONTROL FORMS

4.5.2 RELATIONAL OPERATORS AND MEMBERSHIP TESTS

4.5.3 BINARY ADDING OPERATORS

4.5.4 UNARY ADDING OPERATORS

4.5.5 MULTIPLYING OPERATORS

4.5.6 HIGHEST PRECEDENCE OPERATORS

4.5.7 ACCURACY OF OPERATIONS WITH REAL OPERANDS

INSTRUCTOR NOTES

- NOTE, THIS WAS COVERED WHEN WE TALKED ABOUT ATTRIBUTES. ITS IMPORTANT THAT THEY UNDERSTAND IT.

IS APPROPRIATE FOR

- DEFINES "A PREFIX THAT IS APPROPRIATE FOR A TYPE" TO BE
  - WHEN THE TYPE OF THE PREFIX IS THE TYPE CONSIDERED
  - WHEN THE TYPE OF THE PREFIX IS AN ACCESS TYPE WHOSE DESIGNATED TYPE IS THE TYPE CONSIDERED

EXAMPLE:

```
procedure Example is
   type Ar is array (1 .. 3) of Integer;
   type Ac_Ar is access Ar;
   Acc : Ac_Ar := new Ar'(3,5,7);
   Arr : Ar := (2,4,6);
   X : Integer := 1;
begin  -- Example
   Acc (1);          -- appropriate for component selection
   Arr (2);          -- appropriate for component selection
   X := X (3);       -- not appropriate for component selection
end Example;
```

INSTRUCTOR NOTES

THROUGHOUT THE LRM, THE RULES STATE THAT AN ENTITY MUST BE STATIC. SECTION 4.9 EXPLAINS WHAT IT MEANS TO BE (AND WHAT ENTITIES ARE) STATIC.

DEFINITION OF STATIC

- ESSENTIALLY "KNOWN AT COMPILE TIME"
- STATIC ENTITIES ARE
  - ENUMERATION LITERALS (INCLUDING CHARACTER LITERALS)
  - NUMERIC LITERALS
  - NAMED NUMBERS
  - CONSTANTS WITH A STATIC SUBTYPE AND A STATIC VALUE
  - A FUNCTION CALL OF A PREDEFINED OPERATOR WITH STATIC PARAMETERS
  - A LANGUAGE DEFINED ATTRIBUTE OF A STATIC SUBTYPE, WITH STATIC PARAMETERS (IF ANY)
  - A QUALIFIED EXPRESSION OF A STATIC SUBTYPE WITH A STATIC EXPRESSION
  - A STATIC EXPRESSION ENCLOSED IN PARENTHESES

INSTRUCTOR NOTES

ALLOW 5 MINUTES FOR THIS EXERCISE.

ANSWER:

IN SECTION 4.4/2, THE SYNTAX FOR AN EXPRESSION SAYS THAT THE EXPRESSION EVALUATES AS FOLLOWS. HAVE STUDENTS TURN TO THIS SECTION.

A EVALUATES TO A NAME, TO A PRIMARY, TO A FACTOR, TO A TERM, TO A SIMPLE EXPRESSION, TO A RELATION. B DOES ALSO.

RELATION and RELATION EVALUATE TO AN EXPRESSION.

THERE IS NO RULE FOR EXPRESSION or RELATION (WHICH IS WHAT C EVALUATES TO). SO THE EXPRESSION IS SYNTACTICALLY INCORRECT.

THE CORRECT EXPRESSION WOULD BE (A and B) or C OR A and (B or C) BECAUSE (EXPRESSION) EVALUATES TO A PRIMARY AND THERE IS A RULE FOR PRIMARY or PRIMARY.

EXAMPLE PROBLEM

WHY IS THE FOLLOWING EXPRESSION A SYNTAX ERROR?

```
if A and B or C then null; end if;
```

INSTRUCTOR NOTES

ALLOW 15 MINUTES FOR THESE EXERCISES.

ANSWERS:

1. SECTION 4.1.3/18 STATES "A NAME DECLARED BY A RENAMING DECLARATION IS NOT ALLOWED AS THE PREFIX" OF AN EXPANDED NAME.

2. SECTION 4.3/7 STATES "THE TYPE OF AN AGGREGATE MUST BE DETERMINABLE SOLELY FROM THE CONTEXT ..." SECTION 8.7/4 DEFINES A DECLARATION TO BE A COMPLETE CONTEXT. SECTION 3.2.1/1 SAYS THAT THE TYPE OF THE EXPRESSION IN AN OBJECT DECLARATION MUST BE THAT OF THE OBJECT. SECTION 8.7/8 SAYS YOU CAN USE INFORMATION IN A RULE THAT REQUIRES AN EXPRESSION TO HAVE THE SAME TYPE AS A NAME. THEREFORE, THE COMPILER CAN DETECT THE TYPE OF THE AGGREGATES.

3. SECTION 4.3/4 STATES "AGGREGATES CONTAINING A SINGLE COMPONENT ASSOCIATION MUST ALWAYS BE GIVEN IN NAMED NOTATION."

EXERCISES

1. WHY IS THE FOLLOWING EXPANDED NAME ILLEGAL?

```
package P is
    type T is 1 .. 10;
    package N_P renames P;
    X : N_P.T;                -- illegal
end P;
```

2. WHY ARE THE FOLLOWING AGGREGATES NOT AMBIGUOUS?

```
type Rec is record
    Comp1 : Boolean;
    Comp2 : Boolean;
end record;
type ARR is array (1 .. 2) of Boolean;
R : Rec := (True, False);
A : Arr := (True, False);
```

3. WHY IS THE FOLLOWING ILLEGAL?

```
type Rec is record
    Comp : Character;
end record;
R : Rec := ('a');             -- illegal
```

INSTRUCTOR NOTES

ALLOW 30 MINUTES FOR THIS SECTION. IF TIME IS SHORT THIS SECTION CAN BE CONDENSED TO ONLY COVER THE TOPIC OF SECTION 7.4.4, LIMITED TYPES.

Section 10

LRM - CHAPTER 7

PACKAGES

INSTRUCTOR NOTES

- POINT OUT THAT SECTION 7.5 AND 7.6 ARE NOT CONSIDERED TO BE PART OF THE STANDARD.

SECTION TOPICS

INSTRUCTOR NOTES

ALLOW 10 MINUTES FOR THIS EXERCISE.

ANSWER:

SECTION 7.4.4/4 STATES "FOR A FORMAL PARAMETER WHOSE TYPE IS LIMITED AND WHOSE DECLARATION OCCURS IN AN EXPLICIT SUBPROGRAM DECLARATION THE MODE out IS ONLY ALLOWED IF THIS TYPE IS PRIVATE AND THE SUBPROGRAM DECLARATION OCCURS WITHIN THE VISIBLE PART OF THE PACKAGE THAT DECLARES THE PRIVATE TYPE."

- REFER STUDENTS TO HANDOUT FOR COMPILER'S ERROR REPORT.

EXAMPLE PROBLEM

WHY IS THE FOLLOWING PROCEDURE DECLARATION ILLEGAL?

```
package Pack is
    type Lp is limited private;
    X : Lp;
private
    type Lp is 1 .. 10;
end Pack;

with Pack;
procedure Proc is
    Procedure P (L : out Pack.Lp) is      -- illegal
    begin -- P
       null;
    end P;
begin -- Proc
    P (Pack.X);
end Proc;
```

INSTRUCTOR NOTES

ALLOW 20 MINUTES FOR THESE EXERCISES.

ANSWERS:

1. SECTION 7.4/4 STATES "THE TYPE MARK OF A DEFERRED CONSTANT DECLARATION MUST DENOTE A PRIVATE TYPE OR A SUBTYPE OF A PRIVATE TYPE ..."

2. SECTION 7.4/3 STATES "A PRIVATE TYPE DECLARATION IS ONLY ALLOWED AS A DECLARATIVE ITEM OF THE VISIBLE PART OF A PACKAGE OR AS THE GENERIC PARAMETER DECLARATION FOR A GENERIC FORMAL TYPE IN A GENERIC FORMAL PART."

EXERCISES

1. WHY IS THE FOLLOWING DEFERRED CONSTANT ILLEGAL?

```
package P is
    Anon : Constant Integer;        -- illegal
private
    Anon : Constant Integer := 3;
end P;
```

2. WHY ARE THE PRIVATE TYPE DEFINITIONS ILLEGAL?

```
package P is
private
    type Pv is private;             -- illegal
end P;

procedure Pr is
    type Pv is private;             -- illegal
begin
    null;
end Pr;
```

3. SECTION 7.4.1/1 STATES "IF A PRIVATE TYPE DECLARATION IS GIVEN IN THE VISIBLE PART OF A PACKAGE, THEN A CORRESPONDING DECLARATION OF A TYPE WITH THE SAME IDENTIFIER MUST APPEAR AS A DECLARATIVE ITEM OF THE PRIVATE PART OF THE PACKAGE.

4. SECTION 7.4.1/4 FORBIDS SUCH USE OF A PRIVATE TYPE BEFORE ITS FULL DECLARATION.

EXERCISES (CONTINUED)

3. WHY IS THE FOLLOWING ILLEGAL?

```
package Pack is
    type Pv is private;
    type Lp is limited private;
private
    type Lp is new character;
end Pack;

package body Pack is
    type Pv is new Integer;
end Pack;
```

4. WHY IS THE FOLLOWING ILLEGAL?

```
type Pv (A, B, C : Integer) is private;
type Apv is access Pv;
Obj : Apv := new Pv (3,4,5);        -- illegal
```

INSTRUCTOR NOTES

THESE ARE IMPORTANT DEFINITIONS, BUT INTUITIVELY OBVIOUS. THE POINT TO NOTE IS THAT THIS CHAPTER DEALS WITH THESE TOPICS.

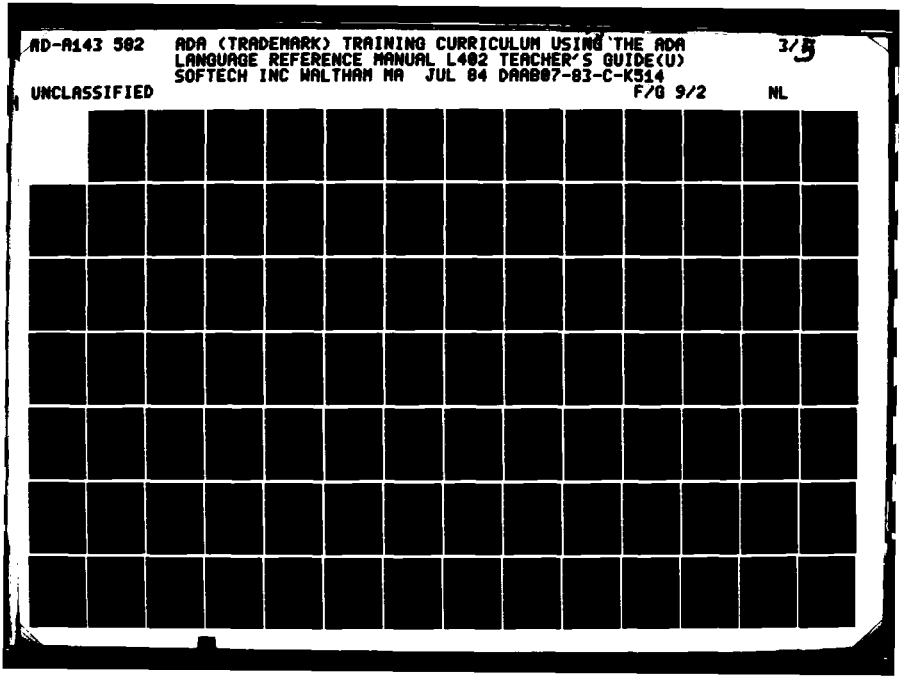
IMPORTANT DEFINITIONS

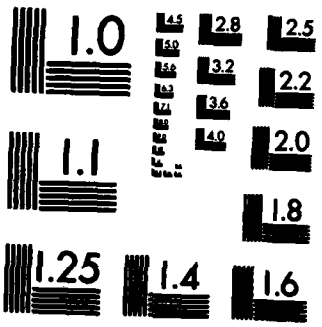
VISIBLE PART - PART ACCESSIBLE OUTSIDE THE PACKAGE

PRIVATE PART - PART INACCESSIBLE OUTSIDE THE PACKAGE

AD-A143 502 ADA (TRADEMARK) TRAINING CURRICULUM USING THE ADA LANGUAGE REFERENCE MANUAL L402 TEACHER'S GUIDE(U) SOFTECH INC WALTHAM MA JUL 84 DAAB07-83-C-K514 3/5

UNCLASSIFIED F/G 9/2 NL





INSTRUCTOR NOTES

ALLOW 30 MINUTES FOR THIS CHAPTER. SHOULD TIME BE SHORT, CONCENTRATE ON CONFORMANCE RULES, TYPE PROFILE, AND PARAMETER PASSING.

TAKE A 15 MINUTE BREAK AFTER THIS.

Section 11

LRM - CHAPTER 6

SUBPROGRAMS

INSTRUCTOR NOTES

THESE ARE THE TOPICS IN THIS CHAPTER. STRESS 6.3.1, 6.6, and 6.2.

SECTION TOPICS

INSTRUCTOR NOTES

THIS IS A QUICK SUMMARY OF THE CONFORMANCE RULES. IT IS STRESSED BECAUSE SCATTERED THROUGHOUT THE LRM, IT SAYS THAT THINGS MUST CONFORM. THIS IS THE ONLY PLACE IN THE LRM THAT STATES WHAT THAT MEANS.

CONFORMANCE RULES

DEFINITION

CONFORMING ENTITIES ARE ENTITIES WHICH ARE FORMED BY THE SAME SEQUENCE OF LEXICAL ELEMENTS.

VARIATIONS

- A NUMERIC LITERAL MAY BE REPLACED BY A DIFFERENT NUMERIC LITERAL WITH THE SAME VALUE AND STILL CONFORM.

- SIMPLE NAMES MAY BE REPLACED BY AN EXPANDED NAME OF THE SIMPLE NAME IF BOTH ARE DEFINED IN THE SAME DECLARATION AND STILL CONFORM.

- A STRING LITERAL GIVEN AS AN OPERATOR SYMBOL CAN BE REPLACED BY ANOTHER REPRESENTATION OF THE SAME OPERATOR AND STILL CONFORM.

INSTRUCTOR NOTES

ALLOW 5 MINUTES FOR THIS EXERCISE.

ANSWERS:

SECTION 6.4.1 STATES "AN ACTUAL PARAMETER ASSOCIATED WITH A FORMAL PARAMETER OF MODE in out OR out MUST BE EITHER THE NAME OF A VARIABLE OR OF THE FORM OF A TYPE CONVERSION WHOSE ARGUMENT IS THE NAME OF A VARIABLE."

(A (I)) IS NEITHER OF THESE. IT EVALUATES TO AN EXPRESSION WHICH IS ILLEGAL FOR out PARAMETERS (SEE SECTION 4.4/2).

- REFER STUDENTS TO HANDOUT FOR COMPILER'S ERROR REPORT.

EXAMPLE PROBLEM

WHY IS THE FOLLOWING ILLEGAL?

```
declare
    type Arr is array (1 .. 10) of Integer;
    procedure Proc (X: out Integer) is
    begin
        X := Integer'Last;
    end Proc;
    A : Arr;
begin
    for I in 1 .. 10 loop
        Proc ((A(I)));              -- illegal
    end loop;
end;
```

INSTRUCTOR NOTES

ALLOW 20 MINUTES FOR THESE EXERCISES.

ANSWERS:

1. SECTION 6.2/10 STATES "IF THE ACTUAL PARAMETER OF A SUBPROGRAM CALL IS A SUBCOMPONENT THAT DEPENDS ON DISCRIMINANTS OF AN UNCONSTRAINED RECORD VARIABLE, THEN THE EXECUTION OF THE CALL IS ERRONEOUS IF THE VALUE OF ANY OF THE DISCRIMINANTS OF THE VARIABLE IS CHANGED BY THIS EXECUTION ..."

2. THE VALUE OF C1 WILL CHANGE DEPENDING ON HOW THE RECORD PARAMETER IS PASSED (BY COPY OR BY REFERENCE). SECTION 6.2/7 STATES THAT AN IMPLEMENTATION IS ALLOWED TO USE EITHER METHOD FOR RECORD PARAMETERS. IT GOES ON TO SAY "THE EXECUTION OF A PROGRAM IS ERRONEOUS IF ITS EFFECT DEPENDS ON WHICH MECHANISM IS SELECTED BY THE IMPLEMENTATION".

EXERCISES

1. WHY IS THE FOLLOWING ERRONEOUS?

```
declare
    type Rec (D : Boolean := True) is record
        case D is
              when True =>  C1 : Character;
              when False =>  C2 : Integer;
        end case;
    end record;
    Rc : Rec;
    procedure P (Ch : out Character) is
    begin
          Rc := Rec'(False,3);
          Ch := 'b';
    end;
begin
    P (Rc.C1);                          -- erroneous
end;
```

2. GIVEN THE FOLLOWING

```
declare
    type Rec is record
        C1 : Boolean;
        C2 : String (1 .. 3);
    end record;
    A : Rec := (True, "BCD");
    procedure P (Y: in out Rec) is
    begin
        A.C1 := False;
        if Y.C1 then
            Y.C2 := "ABC";
        else
            Y.C2 := "DEF";
        end if;
        Y.C1 := False;
    end P;
begin
    P(A);
end;
```

WHY CAN THE VALUE OF A.C2 BE EITHER "ABC" OR "DEF". IS THE PROGRAM ERRONEOUS?

ANSWERS (CONTINUED):

3. NO IN BOTH CASES. SECTION 6.3.1/5 STATES "TWO SUBPROGRAM SPECIFICATIONS ARE SAID TO CONFORM IF ... BOTH SPECIFICATIONS ARE FORMED BY THE SAME SEQUENCE OF LEXICAL ELEMENTS ... " SECTION 6.3.1/6 STATES "CONFORMANCE IS LIKEWISE DEFINED FOR FORMAL PARTS, DISCRIMINANT PARTS, AND TYPE MARKS ..."

4. WHEN THE PROCEDURE Proc IS CALLED WITHOUT AN ACTUAL PARAMETER. SECTION 6.4.2/2 STATES "FOR ANY OMITTED PARAMETER ASSOCIATION, THE DEFAULT EXPRESSION IS EVALUATED BEFORE THE CALL AND THE RESULTING VALUE IS USED AS AN IMPLICIT ACTUAL PARAMETER."

EXERCISES (CONTINUED)

3. DO THE TYPE DECLARATIONS OF Pv CONFORM?

```
package Pack is
    type Pv (C,D : Integer) is private;
    procedure Proc (X: Integer);
private
    type Pv (C: Integer; D: Integer) is record
        null;
    end record;
end Pack;
```

DO THE PROCEDURE SPECIFICATIONS CONFORM?

```
package body Pack is
    procedure Proc (X : in Integer) is
    begin
        null;
    end Proc;
end Pack;
```

4. IN THE FOLLOWING

```
type Int is 1 .. 10;
procedure Proc (X : Int := 11) is
begin
    null;
end;
```

WHEN, IF EVER, WILL Constraint_Error BE RAISED?

INSTRUCTOR NOTES

THESE TERMS ARE USED IN SECTION 6.2 WHEN TALKING ABOUT PARAMETER MODES, AND SPECIFICALLY WHAT MAY BE DONE INSIDE THE SUBPROGRAMS WITH PARAMETERS OF CERTAIN MODES.

IMPORTANT DEFINITIONS

- READ - WHEN AN OBJECT'S VALUE IS EVALUATED
- UPDATED - WHEN AN ASSIGNMENT IS PERFORMED ON A VARIABLE

INSTRUCTOR NOTES

THIS SECTION IS VERY IMPORTANT. ALLOW 60 MINUTES FOR THIS SECTION. DO NOT SKIP. THE MAJOR POINT HERE IS THAT MUCH OF THE CHAPTER IS UNDEFINED.

Section 12

LRM - CHAPTER 9

TASKS

INSTRUCTOR NOTES

- POINT OUT THAT SECTION 9.12 IS NOT CONSIDERED TO BE PART OF THE STANDARD.

SECTION TOPICS

INSTRUCTOR NOTES

DIRECT DEPENDENCE IS DISCUSSED IN SECTION 9.4/1-4. IT IS IMPORTANT TO UNDERSTAND DEPENDENCIES IN ORDER TO AVOID TASKING PROBLEMS, SUCH AS DEADLOCK.

DIRECT DEPENDENCE

- A TASK DESIGNATED BY A TASK OBJECT OR SUBCOMPONENT CREATED BY THE EVALUATION OF AN ALLOCATOR DEPENDS ON THE MASTER THAT ELABORATES THE CORRESPONDING ACCESS TYPE DEFINITION

- A TASK DESIGNATED BY ANY OTHER TASK OBJECT DEPENDS ON THE MASTER WHOSE EXECUTION CREATES THE TASK OBJECT

INSTRUCTOR NOTES

TASK TERMINATION IS DEFINED IN SECTION 9.4/5-10. POINT TO NOTE, SECTION 9.4 IS IMPORTANT. IF TIME PERMITS,,ALLOW STUDENTS TIME TO READ THIS SECTION AND DISCUSS ANY QUESTIONS.

TASK TERMINATION

- WHEN NO DEPENDENT TASKS, TERMINATION TAKES PLACE WHEN EXECUTION HAS COMPLETED

- WHEN DEPENDENT TASKS, TERMINATION TAKES PLACE WHEN ITS EXECUTION IS COMPLETE AND ALL DEPENDENT TASKS HAVE TERMINATED

INSTRUCTOR NOTES

SHARED VARIABLES ARE DISCUSSED IN SECTION 9.11. THE SECOND POINT IS THE IMPORTANT POINT HERE.

SHARED VARIABLES

- VARIABLES WHICH ARE READ OR UPDATED BY MULTIPLE TASKS
- NEITHER MAY ASSUME ANYTHING ABOUT THE ORDER OPERATIONS ARE PERFORMED EXCEPT WHERE THEY SYNCHRONIZE
- THEY MAY ASSUME
  - THAT IF BETWEEN TWO SYNCHRONIZATION POINTS A TASK READS A SCALAR OR ACCESS SHARED VARIABLE, THEN THE VARIABLE IS NOT UPDATED BY ANY OTHER TASK BETWEEN THESE TWO POINTS
  - THAT IF BETWEEN TWO SYNCHRONIZATION POINTS, A TASK UPDATES A SHARED VARIABLE WHOSE TYPE IS SCALAR OR ACCESS, THEN THE VARIABLE IS NEITHER READ OR UPDATED BY ANY OTHER TASK AT ANY TIME BETWEEN THESE TWO POINTS

INSTRUCTOR NOTES

THE SYNCHRONIZATION POINTS ARE DEFINED IN SECTION 9.5.

SYNCHRONIZATION POINTS

- TWO TASKS ARE SYNCHRONIZED AT THE
  - START AND END OF A RENDEZVOUS
  - START AND END OF ACTIVATION
- A COMPLETED TASK IS SYNCHRONIZED WITH ANY OTHER TASK

INSTRUCTOR NOTES

ALLOW 10 MINUTES FOR THIS EXERCISE.

ANSWER:

SECTION 9.6/3 STATES "THE SIMPLE EXPRESSION MUST BE OF THE PREDEFINED FIXED POINT TYPE DURATION ..." THE TYPE OF X IS FLOAT.

EXAMPLE PROBLEM

WHY IS THE DELAY STATEMENT ILLEGAL?

```
task T is
    entry E1;
end T;

task body T is
    X : constant Float := 20.0;
begin
    select
        accept E1;
    or
        delay X;      -- illegal
    end select;
end T;
```

INSTRUCTOR NOTES

ALLOW 30 MINUTES FOR THESE EXERCISES.

ANSWERS:

1. TASK A STARTS EXECUTION JUST AFTER PASSING THE begin IN Proc. (SECTION 9.3/2). TASK B AND C START EXECUTION WHEN THEY ARE ASSIGNED new T, SECTION 9.3/6 STATES "A TASK OBJECT THAT IS THE OBJECT ... CREATED BY THE EVALUATION OF AN ALLOCATOR IS ACTIVATED BY THIS EVALUATION." NOTE THIS HOLDS TRUE FOR TASK C EVEN THOUGH IT IS IN A DECLARATIVE PART.

EXERCISES

1. WHEN DOES THE EXECUTION OF EACH TASK START?

```
procedure Proc is
    task type T is
    end T;
    type Act is access T;
    A : T;
    B : Act;
    task body T is
    begin
        null;
    end T;
begin -- Proc
    B := new T;
    declare
        C : Act := new T;
    begin
        null;
    end;
end Proc;
```

ANSWERS (CONTINUED):

2. SECTION 9.5/11 - 14 STATES THAT WHEN ENTRY T.Outer GETS CALLED THE EXECUTION OF THE CALLER IS SUSPENDED WHILE THE ACCEPT STATEMENT IN THE TASK IS EXECUTED. (9.5/14). INSIDE THE ACCEPT, THE ACCEPT FOR Inner IS ENCOUNTERED, EXECUTION OF THE TASK IS SUSPENDED (9.5/12) UNTIL ITS ENTRY IS CALLED. NOW BOTH ARE SUSPENDED, EACH WAITING FOR THE OTHER. THIS IS CALLED A DEADLOCK.

EXERCISES (CONTINUED)

2. WHY DOES EXECUTION NEVER REACH POINT 1?

```
declare
    task T is
        entry Outer;
        entry Middle;
        entry Inner;
    end T;

    task body T is
    begin
        accept Outer do
            accept Middle do
                accept Inner do
                    null;
                end Inner;
            end Middle;
        end Outer;
    end T;
begin
    T.Outer;
    T.Middle;          -- Point 1
    T.Inner;
end;
```

INSTRUCTOR NOTES

THESE ARE THE LANGUAGE TERMS DEFINED IN THIS CHAPTER.

IMPORTANT DEFINITIONS

TASK	-	ENTITIES WHOSE EXECUTION PROCEEDS IN PARALLEL (THIS INCLUDES THE MAIN PROGRAM)
RENDEZVOUS	-	SYNCHRONIZATION OR INTERACTION BETWEEN TASKS
DEPENDENT TASK	-	ONE WHICH HAS A MASTER
MASTER	-	A TASK, CURRENTLY EXECUTING BLOCK STATEMENT OR SUBPROGRAM, OR A LIBRARY PACKAGE
COMPLETED	-	FINISHED EXECUTION
ABNORMAL	-	STATE OF AN ABORTED TASK
SHARED	-	ACCESSED BY MULTIPLE PROGRAM UNITS

INSTRUCTOR NOTES

ALLOW 30 MINUTES FOR THIS SECTION.

Section 13

LRM - CHAPTER 12

GENERIC UNITS

INSTRUCTOR NOTES

- POINT OUT THAT SECTION 12.4 IS NOT CONSIDERED TO BE PART OF THE STANDARD.

SECTION TOPICS

INSTRUCTOR NOTES

ALLOW 5 MINUTES FOR THIS EXERCISE.

ANSWER:

THE INSTANTIATION IS ILLEGAL BECAUSE OF HOW THE PRIVATE TYPE Pv IS USED WITHIN THE GENERIC PACKAGE. IT IS USED TO DECLARE AN OBJECT. THIS PLACES THE RESTRICTION ON THE PRIVATE TYPE THAT THE ACTUAL TYPE MUST BE CONSTRAINED (WHICH String ISN'T). SEE SECTION 12.3.2/4.

- REFER STUDENTS TO HANDOUT FOR COMPILER'S ERROR REPORT.

EXAMPLE PROBLEM

WHY IS THE FOLLOWING INSTANTIATION ILLEGAL?

```
generic
    type Pv is private;
package Pk is
    Obj : Pv;
end Pk;

package Npk is new Pk (String);     -- illegal
```

INSTRUCTOR NOTES

ALLOW 20 MINUTES FOR THESE EXERCISES.

ANSWERS:

1. THIS ONE IS TRICKY. ALTHOUGH SECTION 12.1/5 STATES "... WITHIN THE DECLARATIVE REGION ... THE NAME OF THIS PROGRAM UNIT DENOTES THE SUBPROGRAM OBTAINED BY THE CURRENT INSTANTIATION OF THE GENERIC UNIT." THE REAL REASON IT IS ILLEGAL IS BECAUSE THE OUTER F IS HIDDEN (SECTION 8.3) AND THE GENERIC F CANNOT BE REFERRED TO UNTIL ITS BODY.

2. SECTION 12.1.2/2 STATES "THE ONLY FORM OF DISCRETE RANGE THAT IS ALLOWED WITHIN THE DECLARATION OF A GENERIC FORMAL (CONSTRAINED) ARRAY TYPE IS A TYPE MARK." 1 .. 10 IS NOT A TYPE MARK (3.3.2/2)

   SECTION 12.1.2/3 STATES "THE DISCRIMINANT PART OF A GENERIC FORMAL PRIVATE TYPE MUST NOT INCLUDE A DEFAULT EXPRESSION ..."

3. SECTION 12.3.1/1 STATES "A GENERIC FORMAL PARAMETER OF MODE in out OF A GIVEN TYPE IS MATCHED BY THE NAME OF A VARIABLE OF THE SAME TYPE." ((3*6)**2) IS NOT THE NAME OF A VARIABLE.

   SECTION 12.3.3/1 STATES "A GENERIC FORMAL TYPE DEFINED BY (<>) IS MATCHED BY ANY DISCRETE SUBTYPE." FLOAT IS NOT A DISCRETE SUBTYPE.

   THIS SAME SECTION GOES ON TO SAY "A GENERIC FORMAL TYPE DEFINED BY range <> IS MATCHED BY ANY INTEGER SUBTYPE". BOOLEAN IS NOT AN INTEGER SUBTYPE.

- REFER STUDENTS TO HANDOUT FOR COMPILER'S ERROR REPORT.

EXERCISES

1. GIVEN

```
function F return Integer;

with F;
package P is
    X : Integer := F;                    -- legal
    generic
       Y : Integer := F;                 -- illegal
    function F (A : Integer) return Integer;
end P;
```

WHY IS THE SECOND DEFAULT EXPRESSION ILLEGAL?

2. WHY IS THE FOLLOWING ILLEGAL?

```
generic
    type Arr is array (1 .. 10) of Boolean;        -- illegal
    type Pv (D : Boolean := True) is private;      -- illegal
package Gp is
end Gp;
```

3. WHAT IS WRONG WITH THE FOLLOWING INSTANTIATION?

```
generic
    Z : in out Integer;
    type T1 is (<>);
    type T2 is range <>;
package P is end P;

package Np is new P (((3*6)**2), Float, Boolean);
```

INSTRUCTOR NOTES

THESE ARE THE IMPORTANT LANGUAGE TERMS DEFINED IN THIS CHAPTER.

BREAK FOR LUNCH HERE.

IMPORTANT DEFINITIONS

- TEMPLATE - CONSTRUCTION OR BUILDING TOOL
- INSTANCES - OCCURRENCE OR COPY
- MATCH - STRUCTURALLY THE SAME

INSTRUCTOR NOTES

ALLOW 30 MINUTES FOR THIS SECTION.

Section 14

LRM - CHAPTER 11

EXCEPTIONS

INSTRUCTOR NOTES

THESE ARE THE TOPICS COVERED IN THIS CHAPTER.

SECTION TOPICS

INSTRUCTOR NOTES

ALLOW 5 MINUTES FOR THIS EXERCISE.

ANSWER:

SECTION 11.1/8 STATES "THIS EXCEPTION IS RAISED .. IF THE SPACE AVAILABLE FOR THE COLLECTION OF ALLOCATED OBJECTS IS EXHAUSTED."

HOW MUCH SPACE IS NEEDED FOR R? R HAS TWO COMPONENTS, D, THE DISCRIMINANT, AND St, A STRING WHOSE LENGTH DEPENDS ON THE DISCRIMINANT. WHEN R WAS DECLARED, NO EXPLICIT VALUE WAS GIVEN FOR THE DISCRIMINANT. THIS ESTABLISHED TWO THINGS. FIRST, IT TOLD THE COMPILER TO TAKE THE DEFAULT VALUE IN TYPE REC FOR THE INITIAL VALUE OF D, AND SECOND, IT TOLD THE COMPILER THAT THIS DISCRIMINANT VALUE CAN CHANGE AND THE STRING SIZE CAN CHANGE. DUE TO THIS SECOND FACT, THE AMOUNT OF STORAGE NEEDED FOR THIS OBJECT IS ENOUGH STORAGE TO HOLD THE BIGGEST R POSSIBLE, THAT IS WHEN D IS Integer'Last.

EASILY THIS COULD RAISE Storage_Error.

EXAMPLE PROBLEM

THE FOLLOWING RAISES Storage_Error. WHY?

```
type Rec (D: Integer := 0) is record
    St : String (1 .. D);
end record;

R : Rec;                    -- Storage_Error
```

INSTRUCTOR NOTES

ALLOW 20 MINUTES FOR THESE PROBLEMS.

ANSWERS:

1. EXCEPTION IS A RESERVED WORD, NOT THE NAME OF A TYPE. 3.6/2 - WHAT FOLLOWS "OF" IN THE SYNTAX MUST BE THE NAME OF A TYPE. 2.9/3 - A RESERVE WORD MUST NOT BE USED AS A DECLARED IDENTIFIER.

2. SECTION 11.1/7 STATES "THIS EXCEPTION IS ALSO RAISED IF THE END OF A FUNCTION IS REACHED ..."

3. BECAUSE SECTION 11.7/1 STATES "THE PRESENCE OF A SUPPRESS PRAGMA GIVES PERMISSION ..." THE KEYWORD HERE IS PERMISSION. IT IS NOT A DIRECTIVE OR AN ORDER THAT THE COMPILER MUST OBEY. AN IMPLEMENTATION IS FREE TO CONTINUE MAKING CHECKS.

EXERCISES

1. WHY IS THE FOLLOWING ILLEGAL?

```
type T is array (1 .. 10) of exception;
```

2. WHY IS Program_Error RAISED HERE?

```
declare
    C : Character;
    function F (B: Boolean) return Character is
    begin -- F
        if B then
           return 'a';
        end if;
    end F;
begin
    C := F (False);          -- Program_Error
end;
```

3. IN THE FOLLOWING,

```
procedure Main is
    pragma Suppress (Range_Check);
    subtype Int is Integer range 1 .. 20;
    X : Int := 0;
begin -- Main
    for I in 1 .. 10
    loop
        X := X + I;
        exit when X > 20;
    end loop;
end Main;
```

WHY CAN THE PROGRAM LEGALLY RAISE Constraint_Error WHEN X IS ASSIGNED THE VALUE 21?

INSTRUCTOR NOTES

THESE ARE THE IMPORTANT LANGUAGE TERMS DEFINED IN THIS CHAPTER. THESE SHOULD BE WELL KNOWN TO THE STUDENTS.

IMPORTANT DEFINITIONS

EXCEPTION - ERRORS OR OTHER EXCEPTIONAL SITUATIONS THAT ARISE DURING PROGRAM EXECUTION

RAISE - ABANDON NORMAL PROGRAM EXECUTION TO DRAW ATTENTION TO THE CORRESPONDING ERROR SITUATION

HANDLING - EXECUTING SOME ACTIONS IN RESPONSE TO THE ARISING OF AN EXCEPTION

INSTRUCTOR NOTES

ALLOW 30 MINUTES FOR THIS SECTION. IF SHORT ON TIME THIS SECTION CAN BE SKIPPED.

Section 15

LRM - CHAPTER 10

PROGRAM STRUCTURE AND COMPILATION ISSUES

INSTRUCTOR NOTES

THESE ARE THE TOPICS COVERED IN CHAPTER 10.

SECTION TOPICS

INSTRUCTOR NOTES

ALLOW 5 MINUTES FOR THIS EXERCISE.

ANSWER:

SECTION 10.2/5 STATES "EACH SUBUNIT MENTIONS THE NAME OF ITS PARENT UNIT, THAT IS, THE COMPILATION UNIT WHERE THE CORRESPONDING BODY STUB IS GIVEN ... IF THE PARENT UNIT IS ITSELF A SUBUNIT, THE PARENT UNIT NAME MUST BE GIVEN IN FULL AS AN EXPANDED NAME, STARTING WITH THE SIMPLE NAME OF THE ANCESTOR LIBRARY UNIT."

EXAMPLE PROBLEM

WHY IS THE SECOND SEPARATE CLAUSE ILLEGAL?

```
procedure Main is
    ...
    procedure Proc is separate;
    ...
begin
    ...  -- some sequence of statements
end Main;

separate (Main)
procedure Proc is
    ...
    function Fn return Boolean is separate;
begin
    ...  -- some sequence of statements
end Proc;

separate (Proc)   -- illegal
function Fn return Boolean is
begin
    ...  -- some sequence of statements
    return True;
end Fn;
```

INSTRUCTOR NOTES

ALLOW 20 MINUTES FOR THESE EXERCISES.

ANSWERS:

1. IT IS IMPLEMENTATION-DEPENDENT. SECTION 10.1/8 STATES "AN IMPLEMENTATION MAY IMPOSE CERTAIN REQUIREMENTS ON THE PARAMETERS AND ON THE RESULT, IF ANY, OF A MAIN PROGRAM ... IN ANY CASE, EVERY IMPLEMENTATION IS REQUIRED TO ALLOW, AT LEAST, MAIN PROGRAMS THAT ARE PARAMETERLESS PROCEDURES ..."

2. Put_line IS AN UNDECLARED IDENTIFIER BECAUSE THE ENTITIES KNOWN TO Proc ARE ONLY THOSE THAT ARE DECLARED IN Proc OR IN PACKAGE A. SECTION 10.1.1/4 STATES "THE with CLAUSES AND use CLAUSES OF THE CONTEXT CLAUSE OF A LIBRARY UNIT APPLY TO THIS UNIT AND ALSO TO THE SECONDARY UNIT THAT DEFINES THE CORRESPONDING BODY ..." Text_IO IS ONLY APPLIED TO PACKAGE A.

EXERCISES

1. CAN A MAIN PROGRAM BE A PROCEDURE WITH FORMAL PARAMETERS? CAN A MAIN PROGRAM BE A FUNCTION?

2. WHY IS THE FOLLOWING ILLEGAL?

```
with Text_IO; use Text_IO;
package A is
    type Int is range 1 .. 100;
    package Int_IO is new Integer_IO (Int);
    use Int_IO;
end A;

with A;
procedure Proc is
    Num : A.Int;
begin -- Proc
    Put_Line ("ENTER A NUMBER LESS THAN 10"); -- illegal
    A.Get (Num);
    Num := Num * Num;
    A.Put (Num);
end Proc;
```

INSTRUCTOR NOTES

THESE ARE THE LANGUAGE TERMS DEFINED BY THIS CHAPTER.

IMPORTANT DEFINITIONS

- PROGRAM LIBRARY - THE COMPILATION UNITS OF A PROGRAM
- MAIN PROGRAM - PROGRAM UNIT INVOKED BY THE ENVIRONMENT

INSTRUCTOR NOTES

ALLOW 30 MINUTES FOR THIS SECTION.

Section 16

LRM - CHAPTER 13

REPRESENTATION CLAUSES AND IMPLEMENTATION-DEPENDENT FEATURES

INSTRUCTOR NOTES

- POINT OUT THAT THIS CHAPTER IS ALMOST ENTIRELY DEVOTED TO IMPLEMENTATION-DEPENDENT ASPECTS OF THE LANGUAGE.

SECTION TOPICS

ALLOW 5 MINUTES FOR THIS EXERCISE.

ANSWER:

SECTION 13.9/4 STATES "THIS CAPABILITY NEED NOT BE PROVIDED BY ALL IMPLEMENTATIONS." THEREFORE, THE PRAGMA MAY BE IGNORED AND Sqrt NO LONGER HAS A BODY. SECTION 6.3/3 STATES "FOR EACH SUBPROGRAM DECLARATION THERE MUST BE A CORRESPONDING BODY (EXCEPT FOR A SUBPROGRAM WRITTEN IN ANOTHER LANGUAGE, AS EXPLAINED IN SECTION 13.9)."

EXAMPLE PROBLEM

WHY IS AN IMPLEMENTATION ALLOWED TO REJECT THE FOLLOWING? IT GIVES AN ERROR MESSAGE WHICH SAYS THE FUNCTION Sqrt NEEDS A BODY.

```
declare
   function Sqrt (X : Float) return Float;
   pragma Interface (Fortran, Sqrt);
begin
   -- some sequence of statements
end;
```

INSTRUCTOR NOTES

ALLOW 20 MINUTES FOR THESE EXERCISES.

BREAK AFTER THIS.

ANSWERS:

1. SECTION 13.3/4 STATES "THE INTEGER CODES SPECIFIED FOR THE ENUMERATION TYPE MUST SATISFY THE PREDEFINED ORDERING RELATION OF THE TYPE."

   SECTION 13.3/2, WHAT FOLLOWS THE use IS AN AGGREGATE. AGGREGATES MUST BE COMPLETE (SECTION 4.3/6).

2. SECTION 13.8/3 STATES "IF A PROCEDURE BODY CONTAINS CODE STATEMENTS, THEN WITHIN THIS PROCEDURE BODY THE ONLY ALLOWED FORM OF STATEMENT IS A CODE STATEMENT..., THE ONLY ALLOWED DECLARATIVE ITEMS ARE USE CLAUSES ..."

3. A - 2
   B - WHITE
   C - 1

   SECTION 13.3/4 STATES "THE INTEGER CODES ... MUST SATISFY THE PREDEFINED ORDERING." THEY DO NOT REPLACE OR SUPERSEDE THE PREDEFINED ORDERING. THE ATTRIBUTES ARE DEFINED ACCORDING TO THE PREDEFINED ORDERING.

EXERCISES

1. IN THE FOLLOWING, WHY IS THE REPRESENTATION CLAUSE ILLEGAL?

```
declare
    type Stat is (On, Off, Pending);
    for Stat use (1, -1, 0);        -- illegal
begin
    null;
end;

declare
    type Color is (Red, Blue, Yellow);
    for Color use (3, 4);    -- illegal
begin
    null;
end;
```

2. WHY IS THE FOLLOWING ILLEGAL?

```
procedure Code_Something is
    use Machine_Code;
    Stat : array (1 .. 10) of Boolean;
begin
    ... -- some code statement
end Code_Something;
```

3. IF YOU HAVE AN ENUMERATION TYPE

```
type Color is (Red, White, Blue);
```

AND THE REPRESENTATION CLAUSE

```
for Color use (1, 6, 9);
```

WHAT ARE THE VALUES OF

A. Color'Pos (Blue)
B. Color'Val (1)
C. Color'Pos (White)

INSTRUCTOR NOTES

THIS CHAPTER IS FAIRLY STRAIGHT FORWARD. IT MAY BE SKIPPED IF TIME IS SHORT.

ALLOW 25 MINUTES FOR THIS SECTION.

Section 17

LRM - CHAPTER 5

STATEMENTS

INSTRUCTOR NOTES

THESE ARE THE TOPICS COVERED IN THIS CHAPTER. POINT OUT THAT SECTION 5.2.1 HAS ABOUT THE ONLY INTERESTING SEMANTIC ISSUE, ARRAY SLIDING.

SECTION TOPICS

INSTRUCTOR NOTES

ALLOW 5 MINUTES FOR THIS EXERCISE.

ANSWER:

SECTION 5.4/4 STATES "... EACH VALUE OF THE SUBTYPE MUST BE REPRESENTED ONCE AND ONLY ONCE IN THE SET OF CHOICES ... "

IN THE INNER CASE STATEMENT THE SUBTYPE OF THE CASE EXPRESSION I IS INTEGER 1 .. 100, NOT INTEGER 1 .. 10. THEREFORE, ALL VALUES OF THE SUBTYPE MUST BE COVERED, IE. AN others CLAUSE IS NEEDED.

EXAMPLE PROBLEM

WHY IS THE INNER CASE STATEMENT ILLEGAL?

```
for I in 1 .. 100 loop
    case I is
        when 1 .. 10 =>
            case I is
                when 1 .. 5 =>  Do_Something;
                when 6 .. 10 =>  Do_Something_Else;
            end case;           -- illegal
        when others =>  null;
    end case;
end loop;
```

INSTRUCTOR NOTES

ALLOW 20 MINUTES FOR THESE EXERCISES.

ANSWERS:

1. SECTION 5.2/1 STATES "THE NAMED VARIABLE AND THE RIGHT HAND SIDE EXPRESSION MUST BE OF THE SAME TYPE; THE TYPE MUST NOT BE A LIMITED TYPE."

2. AN EXPRESSION WHICH APPEARS IN A CHOICE MUST BE A SIMPLE EXPRESSION (3.7.3/2). A SIMPLE EXPRESSION (SECTION 4.4/2) DOES NOT INCLUDE RELATIONAL OPERATORS UNLESS THE EXPRESSION IS ENCLOSED IN PARENTHESES.

- REFER STUDENTS TO HANDOUT FOR COMPILER'S ERROR REPORT.

EXERCISES

1. WHY IS THE FOLLOWING ASSIGNMENT STATEMENT ILLEGAL?

```
package Pack is
    type Lp is limited private;
    function Flp return Lp;
    Lp_Obj : Lp := Flp;              -- illegal
private
    type Lp is new Boolean;
end Pack;
```

2. WHY IS THE FOLLOWING ILLEGAL?

```
declare
    Answer : Boolean;
begin
...     -- some sequence of statements where Answer receives
        -- a value
    case Answer is
        when 1 > 2 =>                -- ILLEGAL
            Put_Line ("Strange occurrence here");
        when others =>  null;
    end case;
end;
```

ANSWERS (CONTINUED):

3. SECTION 5.5/2 STATES I in Integer range 1 .. 3 MUST BE A CONDITION. A CONDITION IS A BOOLEAN_EXPRESSION (SECTION 5.3/2). SO THE ABOVE MUST BE A LEGAL EXPRESSION OF THE FORM Simple_expression in Range. (SECTION 4.4/2)

   I IS CERTAINLY A SIMPLE EXPRESSION. Integer range 1 .. 3, HOWEVER, IS NOT A VALID RANGE ACCORDING TO SECTION 3.5/2. THEREFORE, I in Integer range 1 .. 3 IS AN ILLEGAL CONDITION.

4. SECTION 5.9/3 STATES "THE INNERMOST SEQUENCE OF STATEMENTS THAT ENCLOSES THE TARGET STATEMENT MUST ALSO ENCLOSE THE GOTO STATEMENT." IN SECTION 5.6/2 THE INNERMOST SEQUENCE OF STATEMENTS IN THE BLOCK STATEMENT ARE THOSE STATEMENTS AFTER THE WORD begin AND BEFORE THE WORD exception. THE GOTO IS NOT IN THAT INNERMOST SEQUENCE OF STATEMENTS.

- REFER STUDENTS TO HANDOUT FOR COMPILER'S ERROR REPORT.

EXERCISES (CONTINUED)

3. WHY IS THE FOLLOWING ILLEGAL?

```
I := 1;
while I in Integer range 1 .. 3 loop
    I := I + 1;
end loop;
```

4. WHY IS THE FOLLOWING GOTO STATEMENT ILLEGAL?

```
begin
    Get (Num);
    Ans := 30.0 / Num;
<<1>> Put (Ans);
exception
    when Numeric_error =>
       Ans := Some_Very_Large_Number;
       goto 1;                    -- illegal
    when others =>
       Put_line ("Error");
end;
```

INSTRUCTOR NOTES

THESE ARE THE LANGUAGE TERMS DEFINED IN THIS CHAPTER.

IMPORTANT DEFINITIONS

- STATEMENT - AN ACTION TO BE PERFORMED
- EXECUTION - THE PROCESS BY WHICH A STATEMENT ACHIEVES ITS ACTION

INSTRUCTOR NOTES

THIS CHAPTER IS IMPORTANT. ALLOCATE 55 MINUTES FOR THIS CHAPTER. DO NOT SKIP.

Section 18

CHAPTER 14

INPUT-OUTPUT

INSTRUCTOR NOTES

- POINT OUT THAT SECTION 14.7 IS NOT CONSIDERED TO BE PART OF THE STANDARD.

SECTION TOPICS

INSTRUCTOR NOTES

THESE ARE FURTHER SUBTOPICS IN SECTION 14.3.

SUBSECTIONS

14.3 TEXT INPUT-OUTPUT

- 14.3.1 FILE MANAGEMENT
- 14.3.2 DEFAULT INPUT AND OUTPUT FILES
- 14.3.3 SPECIFICATION OF LINE AND PAGE LENGTHS
- 14.3.4 OPERATIONS ON COLUMNS, LINES, AND PAGES
- 14.3.5 GET AND PUT PROCEDURES
- 14.3.6 INPUT-OUTPUT OF CHARACTERS AND STRINGS
- 14.3.7 INPUT-OUTPUT FOR INTEGER TYPES
- 14.3.8 INPUT-OUTPUT FOR REAL TYPES
- 14.3.9 INPUT-OUTPUT FOR ENUMERATION TYPES
- 14.3.10 SPECIFICATION OF THE PACKAGE TEXT_IO

ALLOW 15 MINUTES FOR THIS EXERCISE.

ANSWER:

SECTION 14.3.4/24 STATES END_OF_FILE "RETURNS True IF A FILE TERMINATOR IS NEXT, OR IF THE COMBINATION OF A LINE, A PAGE, AND A FILE TERMINATOR IS NEXT; OTHERWISE RETURNS False."

SECTION 14.3.2/4 STATES Reset FOR FILES OF MODE Out_File "HAS THE EFFECT OF CALLING New_Page, UNLESS THE CURRENT PAGE IS ALREADY TERMINATED; THEN OUTPUTS A FILE TERMINATOR."

CONSEQUENTLY, File CONTAINS THE FOLLOWING SEQUENCE:

"LF, LF, LF, NP, LF, LF, NP, EOF"

WHERE LF IS A LINE TERMINATOR, NP IS A PAGE TERMINATOR, AND EOF IS A FILE TERMINATOR.

THIS SEQUENCE WILL CAUSE End_of_File TO RETURN False. THE LOOP IS ENTERED AND A Get IS ATTEMPTED. SECTION 14.3.6/3 STATES "AFTER SKIPPING ANY LINE TERMINATORS, AND ANY PAGE TERMINATORS, (Get) READS THE NEXT CHARACTER..." SECTION 14.3.6/4 STATES "THE EXCEPTION End_Error IS RAISED IF AN ATTEMPT IS MADE TO SKIP A FILE TERMINATOR."

EXAMPLE PROBLEM

IN THE FOLLOWING,

```
Create (File, Out_File);
New_Line (File, 3);
New_Page (File);
New_Line (File, 2);
Reset (File, In_File);
if not End_Of_File (File) then
    Get (X);          -- End_Error
end if;
```

WHY IS End_Error RAISED?

INSTRUCTOR NOTES

ALLOW 40 MINUTES FOR THESE EXERCISES.

ANSWERS:

1. NO, SEQUENTIAL_IO IS A GENERIC PACKAGE WITH ONE FORMAL PARAMETER, A PRIVATE TYPE. (14.1/4) ACCORDING TO THE MATCHING RULES FOR A GENERIC PRIVATE TYPE (12.3.2/2-3) TYPE String MATCHES A GENERIC PRIVATE TYPE.

2. THE Get RAISES Status_Error. SECTION 14.3.5/9 STATES "Status_Error IS RAISED BY ANY OF THE PROCEDURES Get, Get_Line, Put, and Put_Line IF THE FILE TO BE USED IS NOT OPEN."

3. NO. Standard_Output IS A FUNCTION WHICH RETURNS File_Type (SECTION 14.3.2/10). Reset'S FORMAL PARAMETER File IS OF MODE in out (SECTION 14.2.1/14). ACTUAL PARAMETERS ASSOCIATED WITH FORMAL PARAMETERS OF MODE in out MUST BE THE NAME OF A VARIABLE (SECTION 6.4.1/3).

4. BOTH Num AND L HAVE UNDEFINED VALUES. SECTION 14.3.6/9 STATES THAT Get FOR A STRING "DETERMINES THE LENGTH OF THE GIVEN STRING AND ATTEMPTS THAT NUMBER OF Get OPERATIONS FOR SUCCESSIVE CHARACTERS OF THE STRING (IN PARTICULAR, NO OPERATION IS PERFORMED IF THE STRING IS NULL.)"

EXERCISES

1. IS THE FOLLOWING ILLEGAL?

```
package String_IO is new Sequential_IO (String);
```

2. WHAT HAPPENS IN THE FOLLOWING SITUATION?

```
with Text_IO; use Text_IO;
procedure P is
    File : File_Type;
    X : Character;
begin -- P
    Create (File, In_File);
    Set_Input (File);
    Close (File);
    Get (X);
end P;
```

3. IS THE FOLLOWING ALLOWED?

```
Reset (Standard_Output, In_File);
```

4. IN THE FOLLOWING

```
declare
    St : String (1 .. 9) := "123456789";
    L : Positive;
    Num : Integer;
    package Int_IO is new Integer_IO (Integer);
    use Int_IO;
begin
    get (St (3 .. 2), Num, L);
end;
```

WHAT IS THE RESULTING VALUE IN L AND IN Num?

INSTRUCTOR NOTES

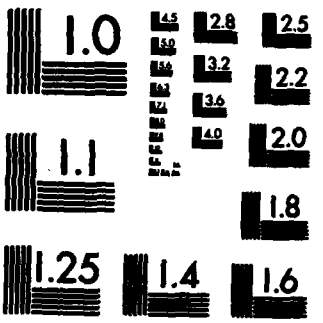
THESE ARE THE IMPORTANT LANGUAGE TERMS DEFINED IN THIS CHAPTER. REMEMBER THAT IN THE LRM, WHEN IT SAYS FILE, THE LRM IS REFERRING TO THE INTERNAL FILE. THE LRM WILL EXPLICITLY STATE "EXTERNAL FILE" WHEN THAT IS WHAT IT IS REFERRING TO.

IMPORTANT DEFINITIONS

EXTERNAL FILE	-	ANYTHING EXTERNAL TO THE PROGRAM THAT CAN PRODUCE A VALUE TO BE READ OR RECEIVE A VALUE TO BE WRITTEN
FILE	-	INTERNAL FILE OBJECT
OPEN	-	WHEN A FILE IS ASSOCIATED WITH AN EXTERNAL FILE
CLOSED	-	WHEN NOT OPEN

AD-A143 502 ADA (TRADEMARK) TRAINING CURRICULUM USING THE ADA LANGUAGE REFERENCE MANUAL L402 TEACHER'S GUIDE(U) SOFTECH INC WALTHAM MA JUL 84 DAAB07-83-C-K514 F/G 9/2 NL

UNCLASSIFIED



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

INSTRUCTOR NOTES

ALLOW 10 MINUTES FOR THIS SECTION. THIS IS THE END OF THE COURSE. THANK THE STUDENTS FOR THEIR TIME AND ATTENTION. PASS OUT COURSE EVALUATION FORMS.

Section 19

CONCLUSION

INSTRUCTOR NOTES

BULLET 1

- THE LRM IS THE FINAL ARBITER! IT DICTATES WHAT IS LEGAL. SHOULD A QUESTION ARISE, THE ADA BOARD INTERPRETS THE ISSUE.

- BY LEARNING TO READ THE LRM, UNDERSTANDING ITS ORGANIZATION AND FORMAL LANGUAGE, YOU CAN RESOLVE A DISAGREEMENT BETWEEN YOUR PROGRAM AND COMPILER!

- OLDER REFERENCES DON'T COUNT! (E.G. ADA 80, ADA 82).

BULLET 2

- FOR EXAMPLE,
  - HANDLING OF ERRONEOUS PROGRAMS
  - DEFINITION OF
    - TYPE INTEGER
    - PACKAGE SYSTEM
    - IMPLEMENTATION-DEFINED ATTRIBUTES
    - APPENDIX F

SUMMARY

- ANSI STANDARD LRM IS THE ONLY COMPLETE AND (BY DEFINITION) CORRECT SPECIFICATION OF THE ADA LANGUAGE

- LRM ALLOWS SOME AREAS OF THE LANGUAGE TO BE IMPLEMENTATION DEPENDENT

INSTRUCTOR NOTES

BULLET 1

- LOOK IT UP BECAUSE THE WORDING IS RELEVANT

BULLET 2

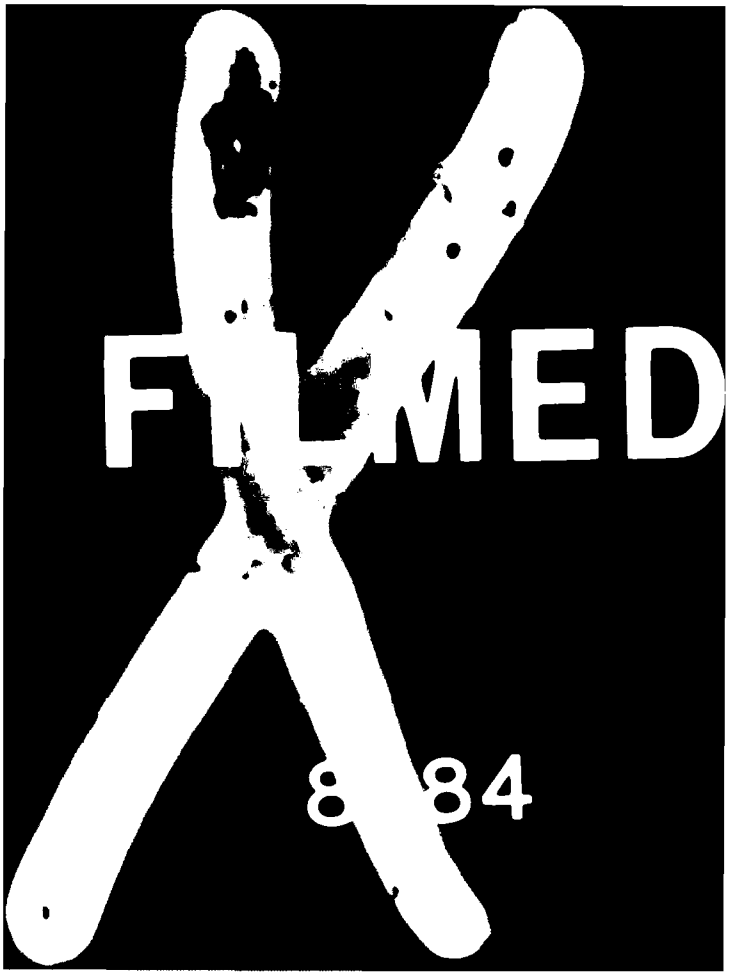
- REASSURE THE STUDENTS

BULLET 3

- THE VALIDATION SUITE THE COMPILER WAS TESTED WITH MAY NOT HAVE CONTAINED THE TEST CASE IN QUESTION. ALSO, THE VALIDATION SUITE CHANGES WITH THE ADDITION OF NEW TESTS AND CHALLENGES TO EXISTING TESTS. ALSO, IT IS NOT POSSIBLE TO CATCH ALL ERRORS BY TESTING.

REMEMBER

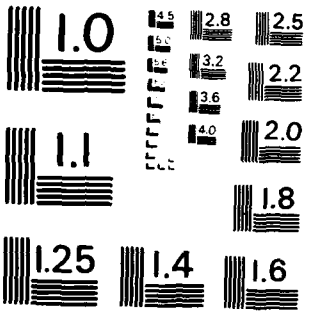
- WHEN ANSWERING A DETAILED QUESTION, ALWAYS LOOK IT UP
- FINDING YOUR WAY AROUND THE LRM GETS EASIER WITH PRACTICE
- VALIDATED COMPILERS CAN CONTAIN ERRORS


FILMED
8 84

AD-A143 582 ADA (TRADEMARK) TRAINING CURRICULUM USING THE ADA LANGUAGE REFERENCE MANUAL L402 TEACHER'S GUIDE(U) SOFTECH INC WALTHAM MA JUL 84 DAAB07-83-C-K514 F/G 9/2 NL

UNCLASSIFIED

END
DATE FILMED
11-84
DTIC



SUPPLEMENTARY

INFORMATION

DEPARTMENT OF THE ARMY
HEADQUARTERS US ARMY COMMUNICATIONS-ELECTRONICS COMMAND
AND FORT MONMOUTH
FORT MONMOUTH, NEW JERSEY 07703

REPLY TO
ATTENTION OF:

15 OCT 1984

Center for Tactical Computer Systems

Ms. Madeline Crumbacker
Defense Tactical Information Center
Cameron Station
Alexandria, Virginia 22314

Dear Ms. Crumbacker:

As per phone conversation with Ms. Andrea Cappellini, CENTACS on 11 October 1984, a copyright statement has been omitted on documents sent to DTIC and NTIS. Enclosed please find the copyright statement (Encl 1) that must appear in the enclosed list of document (Encl 2). If you have any questions, please contact Ms. Cappellini at 201-544-4280.

Sincerely,

for JAMES E. SCHELL
Director, CENTACS