FINAL TECHNICAL REPORT

ARPA Grant # MDA972-92-J-1031

(Undergraduate Curriculum & Course Development
in Software Engineering and the Use of Ada)

"Ada in Introductory Computer Science Courses"

SUBMITTED BY:

Sandra Honda Adams
Computer Science
Sacred Heart University
5151 Park Avenue
Fairfield, Connecticut 06432-1000

TO:

Defense Technical Information Center
Attn: DTIC-DFAC
Cameron Station
Alexandria, VA 22304-6145

TABLE OF CONTENTS

Introduction

Introduction

Sacred Heart University Computer Science Department which is a part of the Faculty of Science, Math, and Computer Science, has offered a computer science major since fall 1983. This program offers two options to service both the business and scientific communities. The curriculum is continually updated following American Computing Machinery (ACM) curriculum guidelines. There are currently 150 computer science majors at the University. Graduates of our program are employed in banks, financial firms, software development firms, utility companies, local and state government, manufacturing firms, and also three local defense contractors, UT/Sikorsky Aircraft, UT/Norden Systems, and Avco Lycoming Industries.

Sacred Heart University's current computer science curriculum has been modified in the 1992-1993 school year after receiving an ARPA Grant (Advanced Research Projects Agency) for "Undergraduate Curriculum and Course Development in Software Engineering and the Use of Ada", "***BAA #91-18, Category #1***. The grant entitled, "**Ada in Introductory Computer Science Course**", allowed for the modification of both introductory programming courses to use Ada as the language of choice, since it is a good software engineering tool that can best support many of the goals and principles of software engineering. The two introductory computer science courses, CS050 (Introduction to Computer Science) and CS051 (Data Structures) were developed to include Ada and software engineering principles. The developed courses allows the student to apply a methodology using Ada and some principles of software engineering to build software that is maintainable, efficient and understandable at an introductory level.

The courses utilized features of Ada to support the development of high-quality, reliable, reusable, and portable software. Learning to conform to good programming practices using Ada provided very clear ideas of software engineering principles and goals without formally teaching software engineering concepts. The students will formally learn software engineering in their junior year. Basic software engineering principles can be indirectly learned during the first two programming courses.

The students were provided with a course syllabus, program examples, handouts, programming assignments and articles to read. The students were introduced to programming with ease, and seemed to enjoy the classes. They found the Ada language simple to understand and use. Most of all, they found Ada to be highly readable and clear. Most of my efforts went into developing the first course. Covering the right material in the right order is an art. The transition into programming should be a painless one. Both scientific and

information option students enroll in both courses. It was important to make the program problems in the first course simple while students dealt with their first encounter with Ada, the editor and unix. Initial assignments were not difficult. Assignments became increasingly open-ended to give the better students an opportunity for creativity. By giving the students handouts and examples to follow, learning was made very simple. The text was not followed sequentially, but used as a supplimental resource. Students were fore warned of this and were told to follow the lecture order in their text. All material on the syllabus was covered in class and supplimented with handouts. The CS051 course followed the text much more carefully. It did not require as many handouts.

Summary

Summary

The grant received from ARPA provided the opportunity to develop new courses material for CS050 and CS051, using Ada and software engineering principles. It made work easier for the project director to develop Ada software examples at home with the purchase of a Meridian Ada PC Compiler. It provided wonderful opportunities to meet other educators and defense personnel promoting the Ada language. It provided opportunities to participate in several Ada and Software Engineering related conferences and to arrange for one to be held on our campus. I have met people from ASEET, strong Ada supporters, educators who have adopted or are about to adopt Ada, and defense contractors in our local area. And best of all, it has allowed me to discover a language that I personally like, to use in the introductory programming courses. It has been a wonderful opportunity and experience that I have thoroughly enjoyed and appreciated!

Time Table

Time Table

Summer 1992

Prepare for CS050 (Introduction to Programming)
- Review Textbooks for course
- Research Articles for use in class
- Decide on course content
- Prepare handouts

Fall 1992

Implement CS050 (Introduction to Programming)- 2 Sections

Prepare for CS051 (Data Structures)
- Research Articles for use in class
- Decide on course content
- Prepare handouts

Participate in **REUSE EDUCATION WORKSHOP** hosted by West Virginia Univ, CARDS, ASEET, & AdaNET Working Group 1:
"Software Reuse in Computer Science Courses"

Spring 1993

Evaluate Course, Revise Course

Implement CS051 (Data Structures)
- 1 Section

Implement CS050 (Introduction to Programming)
- 2 Sections

Summer 1993

Evaluate Courses

Make Revisions

Fall 1993

Implement CS050
-2 Sections (Introduction to Programming)
Implement CS051
-1 Section (Data Structures)
Evaluate and make Revisions

Spring 1994

implement CS050
1 Section (Introduction of Programming)
implement CS051
1 Section (Data Structures)
Attended **Eighth Annual ASEET Symposium**
Albuquerque, New Mexico, Jan. 10-13, 1994
Attended Introduction to Ada9X classes
by Capt. David Cook & Eugene Bingue

Present Paper, at **Twelfth Annual National Conference on Ada Technology** entitled,
"Ada, a Software Engineering Tool, in Introductory Computer Science Programming Courses at Sacred Heart University: Mutual Benefits" Williamsburg, Va Mar. 21-24
Attended Ada9X tutorial by Normam Cohen

Summer 1994

Complete report for grant.
Planning to host 3 days hands-on Ada9X
ASEET 1994 Summer Workshop on campus
for educators and those interested in Ada9X
"The New World of Ada9X", August 3-5, 1994
- Introduction to Ada9X
- Object Oriented Design in Ada9X
- Real-time in Ada9X

Coordinator - Catherine McDonald IDA/ASEET

Budget

BUDGET

Personnel

Project Director, Sandra Honda Adams; Summer 1992: 2 months X $3,000/mo	$6,000

Non-Personnel

Meridien Ada Compiler for use on personal computer (for use by Project Director on home computer)	$ 333
Duplicating Costs for Educator Packets 100 Packets @ $5/Packet =	$ 500
Postage for Dissemination of Information	$ 300
Total Direct Costs:	$ 7133
INDIRECT COSTS (@ 10% of TDC):	$ 713
TOTAL PROJECT COSTS:	$ 7846

CS050 Notes

SACRED HEART
UNIVERSITY

Notes for CS050
Introduction to Structured Programming

Textbook: **Ada Problem Solving and Program Design**
by Michael B. Feldman and Elliot B. Koffman

Course Objective: To develop expertise in writing structured programs using ADA and software engineering concepts.

(Students should read text covering material discussed in class . Specific chapters and sections will be given during each class.)

Week 1 & Week 2

- Introduce the students to Ada through a brief lecture about the history of Ada. Mention the software crisis and the need for software engineering.
- Explain **Handout #1** Example of an Ada Program ***violet.ada***
- Give a general explanation of the parts of a program (refer ot handout)
    - With and use Context Clause
    - Header
    - Declaration
    - Executable Body
- ***Homework Assignment #1*** - First Program Assignment
- Explain Instructions on Dec 5500 (refer to handout)
    - Logging on, Password Change
    - Unix , Vi Editor, Compilation, Linking, and Execution
    - Printing program and output
    - Review Ultrix Mail Facility (refer to handout)
- Review of Class Procedures and Class Package (see handouts)
  Student projects are collected in a class procedure and housed in a package. Both are mailed electronically to each student for extraction and compilation. This first introduction to subprogram procedures and packages will be an easy one for students to understand.
- Explain Software Engineering Concepts (Reuse, Abstraction, Information Hiding) as seen in the development of the class project.
- Review Program Development
    - Problem Definition
    - Specification of Domain and I/O
    - Algorithm Development - (refer to handout)

5151 PARK AVENUE, FAIRFIELD, CONNECTICUT 06432-1000 (203) 371-7999 FAX (203) 365-7609

(flowchart/pseudo code/top down design)
- Flowchart Definition; Examples (refer to handout)
- Coding, Testing, Documentation

- ***Homework Assignment #1b*** (Flowcharting problems)

Week 3 & 4

- ***Handout #2*** - **Program development:**
- Declaration Section (see handout)
  - Data Types
  - Variables
  - Constants
  - subtypes
  - generic instantiation
    Puts, Gets, Put_line; New_line I/O from Package Text_io
- Ada Executable Statements (see handout)
  - Assignment
    - Numeric, Logical, and Relational Operators
    - Hierarchy of Operators
  - Control Statements (if, loop, while, for, case)
- Review of flowcharting payroll problem example with and without loop
  Coding in Ada - (flowcharting problems)(3 sets in handout)
  - without loop
  - with while loop
  - with loop-end loop
- ***Assignment #2*** - Coding flowcharting Problems in Ada
- Ada Examples from flowcharting problem set
  - Using proper documentation and code formatting
  - Importance of naming Identifiers properly
  - Importance of code structure readability

Week 5

- **Handout #3** - String Variable and Constant declaration
- Enumeration type declaration
- Control structure - Case Statement
- If, If-Elsif, Case
- ***Homework Assignment #3*** - Develop example programs using
  - all 3 control structures and enumeration data types
  - and reading
- **Exam I**

Week 6

- **Handout #4** - Logical Expressions
- File Processing Subprogams from Text_io
  - open, create, close, get, put, end_of_file
- Example on file processing
- ***Homework Assignment #4*** - Code 2 out of 3 Problems

Week 7 & 8

- **Handout #5** - Subprogram Development
  - Procedures
  - Functions
  - Parameter Passing Modes (in, out, in out)
- For Statement - Single and nested for loops
- Examples of Subprograms and For statements (Truth Table)
- ***Homework Assignment #5*** - Change Last Assignment to utilize Subprograms. Create single and nested for loops.

Week 9

- **Exam II**
- Class participation on using files & developing subprograms
- Exception Handling
- **Handout #6** - Examples of Exceptions
- ***Homework Assignment #6*** - Use exceptions

Week 10

- One and Multi-Dimensional Arrays
  - Array type declaration
  - Name notation, Positional Notation
- **Handout #7** - Examples of array declaration and use
- ***Homework Assignment #7*** - Write program using 1 D arrays

Week 11

- Separate Compilations - Procedure and Function
- Multi-dimensional arrays
- **Handout #8** - Examples of Multi-dimensional arrays and separate Compilations
- ***Homework Assignment #8*** - Write program using 2-D arrays

Final Exam

CS051 Notes

SACRED HEART
UNIVERSITY

Notes for CS051
Data Structures
3 Credits

Textbook: **Data Structures with Abstract Data Types and Ada**
by Daniel F. Stubbs and Neil W. Webre

Course Objective: To introduce the students to the basic classical data structures of computer science, emphasizing skills in design, analysis and software engineering, through the use of packages, generics, and private types.

Week 1 & Week 2	Chapter
• Introduction to data structures	2.1
• **Handout #1** - Sorting, Subprograms, Modular Design, Menus	
• Arrays	2.2
• Dynamic and Unconstrained Arrays	2.3
• ***Homework Assignment #1*** (Arrays)	
Week 3 & 4	
• Records	2.2
• Pointers (Dynamic Memory Allocation	2.4
• Abstract Data Types (ADT)	1.6
• **Handout #2** - Examples of Stacks	
• Stack package (ADT)	1.4
• Generic Stack Package	1.3
• ***Homework Assignment #2*** - Stacks	
• **Exam I**	
Week 5 & 6	
• **Handout #3** - FIFO Queues	3.3
• Program Examples of Queues	
• Scheduling I/o Requests on a Magnetic Disk	3.5
• Queue package (ADT)	
• Generic Queue Package	
• ***Homework Assignment #3*** - Parking Garage Problem	

5151 PARK AVENUE, FAIRFIELD, CONNECTICUT 06432-1000 (203) 371-7999 FAX (203) 365-7609

Week 7 & 8

- **Handout #4** - Program Examples of Linked Lists
- Linked List Abstraction 4.3
- Double Linked and Circular Lists 4.4
- Ordered Lists 4.5
- Rings 4.6
- Linked Lists (ADT)
- Generic Linked Lists (ADT)
- ***Homework Assignment #4*** - Linked List Problem
- **Exam II**

Week 9 - 11

- **Handout #5** - Examples of Trees
- Elements and structure of trees 5.2
- Binary Trees 5.3
- Binary Tree Search 5.4
- Introduction to Recursion 2.46
- Tree Traversal and Display 5.6
- ***Homework Assignment #5*** - Tree Problem

Final Exam

Appendices

Appendix A

Bibliography

Bibliography of Text Books

Bell/Morrey/Pugh, **Software Engineering, A Programming Approach**, Prentice Hall, 1992, NJ

Booch, **Software Engineering With Ada**, Second Edition, Benjamin Cummings, 1986, CA

Caverly/Goldstein, **Introduction to Ada: A Top-Down Approach for Programmers**, Brooks Cole, 1986, CA

Cohen, **Ada As a Second Language**, McGraw Hill, 1986, NY

Cooling/N. Cooling/J. Cooling, **Introduction to Ada**, Chapman & Hall, 1993, London

Dale/Weems/McCormick, **Programming and Problem Solving with Ada**, D. C. Heath and Company, 1994, MA

Feldman/Koffman, **Ada Problem Solving and Program Design**, Addison-Wesley, 1992, MA

Gehani, **Ada: An Advanced Introduction**, Second Edition, Prentice Hall, 1989, NY

Ghezzi/Jazayeri/Mandrioli, **Fundamentals of Software Engineering**, Prentice Hall, 1991, NJ

Gilpin, **Ada A Guided Tour & Tutorial**, Prentice Hall, 1986, NJ

Hillam, **Introduction to Abstract Data Types Using Ada**, Prentice Hall, 1994, NJ

Nielsen, **Object-Oriented Design with Ada**, Bantam Books, 1992, NY

Olsen/Whitehill, **Ada for Programmers**, Reston, 1983, VA

Savitch/Petersen, **Ada, An Introduction to the Art and Science of Programming**, 1992, CA

Shumate, **Understanding Ada with Abstract Data Types**, Wiley, 1989, NY

Texel, **Introductory Ada: Packages for Programming**, Wadsworth, 1986, CA

Vliet, **Software Engineering Principles and Practice**, Wiley, 1993, NY

Volper/Katz, **Introduction To Programming Using Ada**, Prentice Hall, 1990, NJ

Watt/Wichmann/Findclay, **Ada language and Methodology**, Prentice Hall, 1987, NJ

Weiss, **Data Structures and Algorithm Analysis in Ada**, Benjamin/Cummings, 1993, CA

Bibliography of Others

Annual ASEET Symposium (4th), **Tutorials**, June, 1989

Riehle, "**Ada: A Software Engineering Tool**", Programmer's Journal, Vol. 6.5, p. 68-79, 1988

Software Productivity Consortium, Inc, "Ada Quality and Style for Professional Progammers", SPC-91061-CMC Ver 02.01.01, Dec. 1992

USAF, **Technical Training Manual On Fundamentals of Ada Programming/ Software Engineering**, June, 1989

USAF, **Technical Training Manual On Fundamentals of Ada Programming/ Software Engineering**, December, 1987

Appendix B

Syllabus

SACRED HEART
UNIVERSITY

Syllabus
CS050
Introduction to Structured Programming

3 Credits

Textbook:	**Ada Problem Solving and Program Design** by Michael B. Feldman and Elliot B. Koffman
Course Objective:	To develop expertise in writing structured programs using ADA and software engineering concepts.

(Students should read text covering material discussed in class . Specific chapters and sections will be given during each class.)

Week 1 & 2

History of Ada and Software Engineering
Example of an Ada Program
Parts of a Program (General explanation as related to handout)

- With Context Clause
- Header
- Declaration
- Executable Body

Instructions for the Dec 5500

- Logging on, Password Change
- Ultrix, Vi Editor, Compilation, Linking, and Execution
- Printing program and output

Homework Assignment #1
Review of Class Procedure and Class Package
Software Engineering Concepts - Reuse, Abstraction, Information Hiding
Problem Development

- Problem Definition
- Specification of Domain and I/O
- Algorithm Development
  (flowchart/pseudo code/top down design)
- Flowchart Definition ; Examples
- Coding, Testing, Documentation

Homework Assignment #1b (Flowcharting problems)

5151 PARK AVENUE, FAIRFIELD, CONNECTICUT 06432-1000 (203) 371-7999 FAX (203) 365-7609

Week 3 & 4

Ultrix Mail Facility
Program Development
Procedure sub program and packages
Declaration Types
- Data Types
- Variables
- Constants
- subtypes

Ada Executable Statements
- Assignment
- Numeric, Logical, and Relational Operators
- Hierarchy of Operators
- Control Statements (if, loop, while, for, case)

I/O from Package Text_io
- Puts, Gets, Put_line; New_line

Review of flowcharting problem assignments
Coding in Ada - (flowcharting problems 3 sets)
- Using proper documentation and code formatting
- Importance of naming identifiers
- Importance of code structure readability

Examples of Ada problems - 3 sets
- Wthout loop
- With while loop
- With loop end-loop

Assignment #2 - Code Flowcharting Problems in Ada
- Using proper documentation and code formatting
- Importance of naming identifiers
- Importance of code structure readability

Week 5

- String Variable and constant declaration
- Enumeration type declaration

Homework Assignment #3 - Develop example programs using all 3 control structures and enumeration data types and reading

Exam I

Week 6

- Logical Expressions
- File Processing Subprogams from Text_io
  (open, create, close, get, put, end_of_file)
- Examples of File processing

Homework Assignment #4 - Code 2 out of 3 Problems

Week 7 & 8

Subprogram Development

- Procedures
- Functions
- Parameter Passing Modes (in, out, in out)

For Statement - Single and nested loops
Examples of Subprograms and For statements(Truth Table)
Homework Assignment #5 - Change Last Assignment to utilize Subprograms. Also write program(2) using files, subprograms, enumeration types and for and case statements.

Week 9

Exam II
Class participation on using files & developing subprograms
Exceptions
Homework Assignment #6 - Use exceptions

Week 10 & 11

1 Dimensional Arrays
Homework Assignment #7 - Write program using 1 D arrays

Week 12

Multi-dimensional arrays
Separate Compitlations
Homework Assignment #8 - Write program using 2-D arrays

Final Exam

SACRED HEART
UNIVERSITY

Syllabus
CS051
Data Structures
3 Credits

Textbook: **Data Structures with Abstract Data Types and Ada**
by Daniel F. Stubbs and Neil W. Webre

Course Objective: To introduce the students to the basic classical data structures of computer science, emphasizing skills in design, analysis and software engineering, through the use of packages, generics, and private types.

Week 1 & Week 2	Chapter
• Introduction to data structures	2.1
• **Handout #1** - Sorting, Subprograms, Modular Design, Menus	
• Arrays	2.2
• Dynamic and Unconstrained Arrays	2.3
• ***Homework Assignment #1*** (Arrays)	
Week 3 & 4	
• Records	2.2
• Pointers (Dynamic Memory Allocation	2.4
• Abstract Data Types (ADT)	1.6
• **Handout #2** - Examples of Stacks	
• Stack package (ADT)	1.4
• Generic Stack Package	1.3
• ***Homework Assignment #2*** - Stacks	
• **Exam I**	
Week 5 & 6	
• **Handout #3** - FIFO Queues	3.3
• Program Examples of Queues	
• Scheduling I/o Requests on a Magnetic Disk	3.5
• Queue package (ADT)	
• Generic Queue Package	
• ***Homework Assignment #3*** - Parking Garage Problem	

5151 PARK AVENUE, FAIRFIELD, CONNECTICUT 06432-1000 (203) 371-7999 FAX (203) 365-7609

Week 7 & 8

- **Handout #4** - Program Examples of Linked Lists
- Linked List Abstraction 4.3
- Double Linked and Circular Lists 4.4
- Ordered Lists 4.5
- Rings 4.6
- Linked Lists (ADT)
- Generic Linked Lists (ADT)
- ***Homework Assignment #4*** - Linked List Problem
- **Exam II**

Week 9 - 11

- **Handout #5** - Examples of Trees
- Elements and structure of trees 5.2
- Binary Trees 5.3
- Binary Tree Search 5.4
- Introduction to Recursion 2.46
- Tree Traversal and Display 5.6
- ***Homework Assignment #5*** - Tree Problem

Final Exam

Appendix C

CS050
Handouts

SACRED HEART
UNIVERSITY

Syllabus
CS050
Introduction to Structured Programming

3 Credits

Textbook:	**Ada Problem Solving and Program Design** by Michael B. Feldman and Elliot B. Koffman
Course Objective:	To develop expertise in writing structured programs using ADA and software engineering concepts.

(Students should read text covering material discussed in class . Specific chapters and sections will be given during each class.)

Week 1 & 2

History of Ada and Software Engineering
Example of an Ada Program
Parts of a Program (General explanation as related to handout)

- Context Clause
- Header
- Declaration
- Executable Body

Instructions for the Dec 5500

- Logging on, Password Change
- Ultrix, Vi Editor, Compilation, Linking, and Execution
- Printing program and output
- Ultrix Mail Facility

Homework Assignment #1
Review of Class Procedure and Class Package
Software Engineering Concepts - Reuse, Abstraction, Information Hiding
Problem Development

- Problem Definition
- Specification of Domain and I/O
- Algorithm Development
  (flowchart/pseudo code/top down design)
- Flowchart Definition ; Examples
- Coding, Testing, Documentation

Homework Assignment #1b (Flowcharting problems)

5151 PARK AVENUE, FAIRFIELD, CONNECTICUT 06432-1000 (203) 371-7999 FAX (203) 365-7609

Week 3 & 4

Program Development

Assignment #2 - Declaration Types

- Data Types
- Variables
- Constants
- subtypes

Ada Executable Statements

- Assignment
- Numeric, Logical, and Relational Operators
- Hierarchy of Operators
- Control Statements (if, loop, while, for, case)

I/O from Package Text_io

- Puts, Gets, Put_line; New_line

Review of flowcharting problem assignments

Coding in Ada - (flowcharting problems 3 sets)

- Using proper documentation and code formatting
- Importance of naming identifiers
- Importance of code structure readability

Examples of Ada problems - 3 sets

- Wthout loop
- With while loop
- With loop end-loop

Code Flowcharting Problems in Ada

- Using proper documentation and code formatting
- Importance of naming identifiers
- Importance of code structure readability

Week 5

- String Variable and constant declaration
- Enumeration type declaration

Homework Assignment #3 - Develop example programs using all 3 control structures and enumeration data types and reading

Exam I

Week 6

- Logical Expressions
- File Processing Subprogams from Text_io
  (open, create, close, get, put, end_of_file)
- Examples of File processing

Homework Assignment #4 - Code 2 out of 3 Problems

Week 7 & 8

- Subprogram Development
  - Procedures
  - Functions
  - Parameter Passing Modes (in, out, in out)
- For Statement - Single and nested for loops
- **Handout #5** - Examples of Subprograms and For statements (Truth Table)
- ***Homework Assignment #5*** - Change Last Assignment to utilize Subprograms. Create single and nested for loops.

Week 9

- **Exam II**
- Class participation on using files & developing subprograms
- Exception Handling
- **Handout #6** - Examples of Exceptions
- ***Homework Assignment #6*** - Use exceptions

Week 10

- One and Multi-Dimensional Arrays
  - Array type declaration
  - Name notation, Positional Notation
- **Handout #7** - Examples of array declaration and use
- ***Homework Assignment #7*** - Write program using 1 D arrays

Week 11

- Separate Compilations - Procedure and Function
- Multi-dimensional arrays
- **Handout #8** - Examples of Multi-dimensional arrays and separate Compilations
- ***Homework Assignment #8*** - Write program using 2-D arrays

Final Exam —

CS050
Handout Set #1

Handout #1

Example of an Ada Program called **violet.ada** :

```
-------------------------------------------------------------------
-- Source File Name: violet.ada
-- This Ada procedure will simply output a picture
-- As directed by the put_line procedure
-- Programmed by S. Honda  on January 15, 1994
-------------------------------------------------------------------
with text_io; use text_io;
procedure violet is
      -- no declarations
begin -- violet
      new_line;
      for i in 1..2 loop
            put_line("         {>o<}          ");
            put_line("       {{{{{}}}}}       ");
            put_line("      {{{{{{{}}}}}}      ");
            put_line("      {{{  o  o  }}}     ");
            put_line("      {{{    ^   }}}     ");
            put_line("       {{    =    }}      ");
            put_line("        ~          ~      ");
      end loop;
end violet;
```

To Compile, Link and run Program :

1. ada <u>violet.ada</u> -- compile source file
   Source file name

2. ald <u>violet</u> -o <u>violet.exe</u> -- link & create executable file
   procedure name executable file name

3. <u>violet.exe</u> -- run
   <u>executable file name</u>

Parts of an Ada Program

context clause
- with *Makes available Packages*
- use *Makes resources visable and directly usable*

header - *Identifies Procedure*

declaration
- enumeration types
- array types
- subtypes
- variables
- constants
- subprograms (procedures and functions)
- generic instantiation of Package Text_io

executable body
- executable Ada instructions
- subprogram calls

context clause — with ***text_io***; use ***text_io***;

header - procedure ***ProcName*** is

executable Ada Instruction

```
for i in 1..2 loop
    statement(s)
end loop;
```

subprogram calls

```
put_line("  Hi There");
put("The ");
put_line("End");
new_line(2);
```

LOGON PROCEDURE FOR COMPUTER:

1. Turn on the terminal.
   The terminal will display :
   Enter username >

2. Type the letter a and press the Enter key.
   The terminal will display :
   Xyplex >

3. To connect to the DEC 5500, type : C SIIU, and press the Enter key

4. Login using your user-id and password.

LOGOFF PROCEDURE

1. To logoff of the Unix System, Hold Down the Control Key labeled [CTRL] and press the Letter "D" on the keyboard.

DIRECTORY INFORMATION:

NOTE: THE UNIX COMMANDS ARE CASE SENSITIVE. THEREFORE: IF THE COMMAND IS DISPLAYED IN LOWER CASE, IT MUST BE TYPED IN LOWER CASE.

$ ls -l		To list the files in your directory
$ rm -i		To erase a file in your directory:
$ mkdir	directory_name	To create a sub-directory
$ cd	directory_name	To change to the sub-directory
$ more	filename.ext	To type a file to the screen.
$ lpr -P	filename.ext	To print a listing of your file.

vax1
vax2
s107

TO CHANGE YOUR PASSWORD:

1. LOGON following the instruction given on this page.

2. Type passwd; The system will respond "New Password:"

If you have already set a password, the system will respond:

Changing password for username:
Old Password:
Type your current password and press RETURN.

3. The system will respond "New Password:"

4. Type your new password. (The password must have at least 6 characters, using no spaces, at least 2 alphabetic characters, and 1 numeric/special character.) Press RETURN.

5. The system will respond "Re-enter new password:". Type your password again. Press RETURN.

6. If there are no error messages, your password has been changed. REMEMBER YOUR NEW PASSWORD.

NOTES: Your password will not be displayed on the terminal during the password change session.

If you should forget your password, the LAB DIRECTOR will have to issue you a new password

TO CREATE/MODIFY A FILE:

$ vi filename.ext

Once you are in the VI EDITOR:

To save a file:

1. Press the Escape Key labeled ESC or [F11]

2. Type a colon ":" followed by the letter w and the letter q. Press RETURN.
   Example: :wq

To exit without saving:

1. Press the Escape Key labeled ESC [F11]

2. Type a colon ":" followed by the word quit followed by the exclamation point "!". Press RETURN.
   Example: :quit!

S. Hong

VI EDITOR

THE VI EDITOR IS A SCREEN EDITOR. IT IS AVAILABLE IN NEAR IDENTICAL FORM ON NEARLY EVERY UNIX SYSTEM.

COMPONENTS OF EDITING:

1. to insert text (insert mode)
2. to delete text (command mode)
3. to change letters or words (command mode)

VI COMMANDS ARE

1. CASE SENSITIVE
2. NOT ECHOED TO THE SCREEN
3. DO NOT REQUIRE A [RETURN] AFTER THE COMMAND

TO INVOKE THE VI EDITOR FROM THE UNIX OPERATING SYSTEM, TYPE THE FOLLOWING:

vi program.add[RETURN]
filename.extension

WHERE filename CAN BE ANY ASCII CHARACTER EXCEPT / WHICH IS RESERVED AS A SEPARATOR BETWEEN FILES AND SUB-DIRECTORY PATHS.

UPON ENTERING THE EDITOR, YOU WILL BE IN COMMAND MODE. THERE ARE THREE MODES:

1. LINE COMMAND MODE

   allows saving and exiting of files, allows for pattern replacement, etc.

2. COMMAND MODE

   allows movement in a file, allows one to perform edits, and to enter insert mode.
   (Keyboard keys have new meaning)

3. INSERT MODE

   used for inserting or appending text to your file.
   (Keyboard is used like a typewriter)

I. COMMAND MODE

A. CURSOR MOVEMENT COMMANDS:

h moves 1 space to left of cursor position
j moves down a line
k moves up a line
l moves 1 space to the right of cursor position

The ← ↓ ↑ → cursor movement keys will do the above also. However they are out of the way.

H moves cursor to the top of screen
M moves cursor to the middle of screen
L moves cursor to the last line of screen

Ctrl f moves forward by screen
Ctrl b moves backward by screen
Ctrl d moves forward 1/2 screen
Ctrl u moves backward 1/2 screen

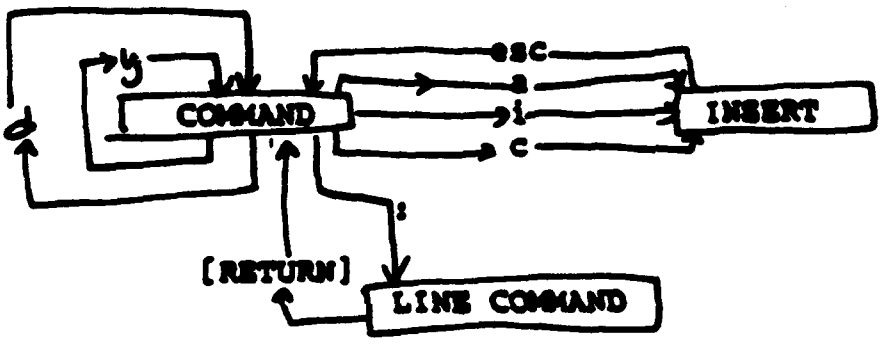
w moves cursor forward by word
W moves forward by word (ignoring punctuation)
b moves backward by word
B moves backward by word (ignoring punctuation)

e moves cursor to end of word
E moves to end of word (ignoring punct)
(moves to beginning of previous sentence
) moves to beginning of next sentence
{ moves to beginning of previous paragraph
} moves to beginning of next paragraph

SIX BASIC EDIT COMMANDS

1. i insert mode
2. a append mode
3. c change word,line
4. d delete word(s) line(s)
5. y yank line(s)

MOVEMENT BETWEEN 3 COMMAND MODES



EDIT COMMANDS IN COMMAND MODE

OBJECT	Change	Delete	Copy
1 word	cw	dw	yw
2 words, ignoring punctuation	2cW or c2W	2dW or d2W	2yW or y2W
3 words back	3cb or c3b	3db or d3b	3yb or y3b
1 line	cc	dd	yy or Y
to end of line	c$ or C	d$ or D	y$
to beginning of line	cØ	dØ	yØ
single character	r	x	yl

OTHER COMMANDS IN COMMAND MODE

place text from buffer	p or P	(stay in COMMAMD MODE)
insert mode	i	INSERT MODE
append mode	a	
append at end of current line	A	
insert at beginning of line	I	
open up line below cursor	o	
open up line above cursor	O	

II. COMMANDS IN LINE COMMAND MODE:

(note that the : gets you to line command mode from command mode)

writes (saves) the buffer to the file but does not exit.	:w
quits the file (and returns to UNIX prompt.	:q
Both writes and quits the file	:wq
quits the file (emphatic)	:q!
writes the file (emphatic)	:w!

Unix Commands

1.	ls -l	list directory of current path
2.	cp _____ _____ file1 file2	copies file1 to file2
3.	rm _____	remove file
4.	cd / cd ..	change directory to root directory change to subdirectory one level above
5.	lpr -Pp_____ _____ _____ file1 file2 vax1 vax2 s107	prints out files
6.	vi _____	invokes the vi editor
7.	more _____	types file to screen
8.	Ctrl c	breaks out of a loop
9.	Ctrl d	logoff terminal
10.	passwd	to issue password
11.	cal	gives you the calander for
	cal _____ year	gives you the calander for the year
12.	>	redirects output (used to capture output)
13.	cat _____ _____ > _____ file1 file2 file3	concatenate files 1 and 2 to file3
14.	to compile link and run ada programs:	
	ada _Source file name_ ald _Main procedure name_ a.out	compiles ada source file links object code runs executable code

Printing Source Program and Program Output

1. BEGIN LOG FILE.

   shu.sacredheart.edu> **script** **filename.lis**

2. TYPE PROGRAM TO SCREEN

   csh> **cat** **filename.ada**

3. RUN PROGRAM

   csh> **filename.exe**

4. REPEAT STEPS 2 & 3 IF REQUIRED

5. END LOG FILE

   csh> **[CTRL-d]**

6. PRINT LOG FILE

   shu.sacredheart.edu> **lpr -Pvax1** **filename.lis**

TO RECIEVE MAIL

1. AT THE SYSTEM PROMPT TYPE THE FOLLOWING:

   SHU> mail [RETURN] -- puts you into mail facility

2. YOU WILL SEE A NEW PROMPT "&" AND A LIST OF MAIL FROM OTHER USERS. TYPE THE FOLLOWING:

   ```
   & 1              [RETURN]  -- types 1st message to you
   & s main.ada     [RETURN]  -- extracts message to file
                              --     main.ada
   & x              [RETURN]  -- leaves mail facility
   ```

TO SEND MAIL

1. AT SYSTEM PROMPT TYPE :

   SHU> mail 11a01 [RETURN]

2. TYPE A ONE WORD SUBJECT IF YOU LIKE, OR JUST A RETURN

   Subject: Party [RETURN]

3. NOW TYPE YOUR MESSAGE; AS MANY LINES YOU LIKE

4. TYPE Ctrl-D WHEN DONE.

5. CC MEANS COPY TO OTHER USERS IF YOU LIKE, OR JUST RETURN

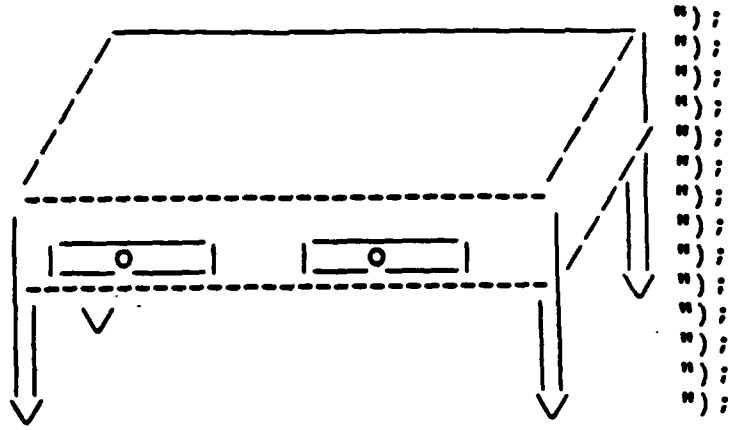
   CC: [RETURN]

WHILE IN THE MAIL FACILITY, YOU MAY WANT TO USE THE FOLLOWING COMMANDS:

```
& h              - lists the mailgrams

& d              - deletes the mailgram you have currently selected

& 2              - selects mailgram number 2 and types it to you

& s filename     - extracts mailgram to a file called filename

& x              - leaves mail facility, not saving changes
                   (deleted mailgrams will not be deleted)

& q              - leaves mail facility, saving chages
                   (deleted mailgrams will be deleted!)

& Ctrl-c Ctrl-c      abandons mailgram- does not send it!
```

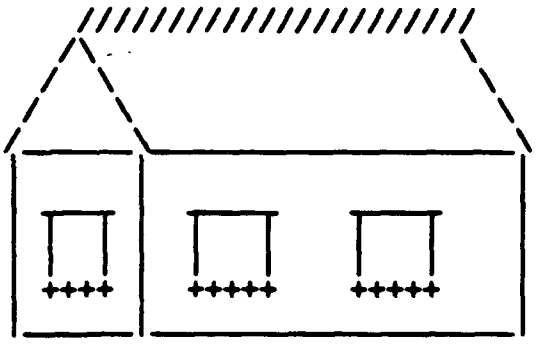
```
-------------------------------------------------------------------------
--      FILE : ART.ADA
--      This package contains a list of available student procedures.
--      Note that the implementation of the procedures are not found
--      here.  They are in the package body.
--      Fall 1993              S. Honda
-------------------------------------------------------------------------
package art is
        procedure desk;
        procedure tree;
        procedure hi;
        procedure house;
        procedure boat;
        procedure house2;
        procedure tracks;
        procedure dog;
        procedure rocket;
end art;
```

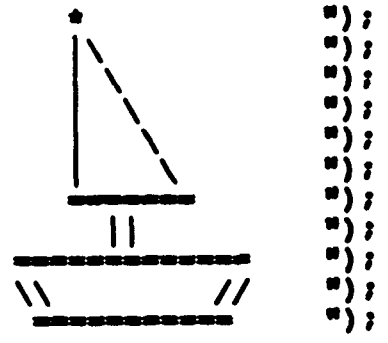
```
------------------------------------------------------------------
--      FILE : ARTBODY.ADA
--      This package body contains student programs from CS050A
--      Note that the implementation of the procedures are found here.
--      Fall 1993           S. Honda
------------------------------------------------------------------
with text_io; use text_io;
package body art is

     ------------------------------------------------------------
     ------------File:desk.ada-----------------------------------
     ------------------------------------------------------------
     --Program by: D A R S H A N   T O O L S I D A S S-----------
     --------------------09/16/93--------------------------------
     procedure desk is
        begin--desk
                put_line("                                                ");
                put_line("                                                ");
                put_line("                                                ");
                put_line("                                                ");
                put_line("                                                ");
                put_line("                                                ");
                put_line("                                                ");
                put_line("                                                ");
                put_line("                                                ");
                put_line("                                                ");
                put_line("                                                ");
                put_line("                                                ");
                put_line("                                                ");
                put_line("                                                ");
        end desk;
```



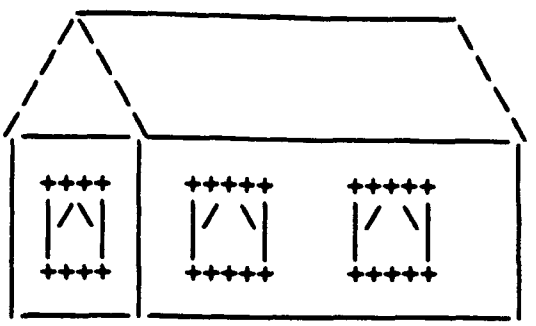
```
     ------------------------------------------------------------
     --      file : tree.ada
     --      This program creates a tree-pole
     --      It uses a package called text_io that allows for I/O
     --      Programmed by Pom Sirichantho
     ------------------------------------------------------------
     procedure tree is
         begin -- tree
                    put_line("           *****                ");
                    put_line("         ((*****))              ");
                    put_line("       ((*********))            ");
                    put_line("    (((*****(*)*****)))         ");
                    put_line("    (((****(***)****)))         ");
                    put_line("     ((((***(*)***))))          ");
                    put_line("      (((***()****)))           ");
                    put_line("        (***\|/***)             ");
                    put_line("             |.|                ");
                    put_line("             |.|                ");
                    put_line("             |.|                ");
                    put_line("            /   \               ");
                    put_line("  *   **              * *       ");
                    put_line("                  ** *          ");
                 put_line(" wecome to CS050!");
          end tree;
```

```
-------------------------------------------------------------------
--      file: Hi.ada
--      This program creates a Hi
--      It usees a package called text_io that allows for I/O
--      Programmed by Daniel DaSilva
-------------------------------------------------------------------
procedure hi is
    begin -- hi
            new_line;
            put_line("   *         *      * ");
            put_line("   *         *      * ");
            put_line("   *         *      * ");
            put_line("   **********      * ");
            Put_line("   *         *      * ");
            Put_line("   *         *      * ");
            Put_line("   *         *      * ");
            Put_line("   ");
            Put_line("   Welcome to CS05");new_line;
    end hi;

-------------------------------------------------------------------
--  file : house.ada       by Don Osvay
-------------------------------------------------------------------
procedure house is
    begin -- house
            put_line("                                              ");
            put_line("                                              ");
            put_line("                                              ");
            put_line("                                              ");
            put_line("                                              ");
            put_line("                                              ");
            put_line("                                              ");
            put_line("                                              ");
            put_line("                                              ");
            put_line("                                              ");
            put_line("                                              ");
            put_line("                                              ");
            new_line;
    end house;

 -------------------------------------------------------------------
 -- file : boat.ada
 -- Joe Peddle
 -------------------------------------------------------------------
 procedure boat is
   begin -- boat
         put_line("                                              ");
         put_line("                                              ");
         put_line("                                              ");
         put_line("                                              ");
         put_line("                                              ");
         put_line("                                              ");
         put_line("                                              ");
         put_line("                                              ");
         put_line("                                              ");
         put_line("                                              ");
         put_line("                                              ");
   end boat;
```





```
-------------------------------------------------------------
--        file : house2.ada
-------------------------------------------------------------
procedure house2 is
   begin -- house2
           put_line("                                          ");
           put_line("                                          ");
           put_line("                                          ");
           put_line("                                          ");
           put_line("                                          ");
           put_line("                                          ");
           put_line("                                          ");
           put_line("                                          ");
           put_line("                                          ");
           put_line("                                          ");
           put_line("                                          ");
           put_line("                                          ");
           put_line("                                          ");
   end house2;
```

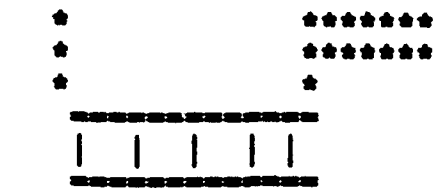


```
-------------------------------------------------------------------
-- file : planet.ada
-- This program creates railroad tracks
-- It uses a package called text_io that allows for I/O
-- Programmed by: Chauncey Wilson
-------------------------------------------------------------------
procedure tracks is
    begin -- tracks
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
                  put_line("                                      ");
   end tracks;
```

```
-------------------------------------------------------------------
-- This program draws a dog.
-- Programmer: Jason Blais
-- Date: Sept. 17, 1993
-------------------------------------------------------------------
procedure dog is
   begin --Start drawing
           put_line("      ************          ");
```

```
                put_line("      *     *        *          ");
                put_line("     *      *     O  *          ");
                put_line("     *      *          ******   ");
                put_line("     *       *              **  ");
                put_line("     *       *         |\/\/\   ");
                put_line("      ******           |        ");
                put_line("        *              |/\/\/\  ");
                put_line("        *             *******   ");
                put_line("        *             *******   ");
                put_line("        *             *         ");
                put_line("         ============           ");
                put_line("         | | | | | |            ");
                put_line("         ============           ");
                put_line(" ");
                put_line("     WOOF, WOOF!!      ");
          end dog;

      ------------------------------------------------------------
      -- file:  rocket.ada
      -- This program creates a rocket
      -- it uses a package called text_io that allows for I/O
      -- Programmed by C. Marable
      ------------------------------------------------------------
      procedure rocket is
         begin -- rocket
                        put_line("                /^\                 ");
                        put_line("               //-\\                ");
                        put_line("              /  ---  \             ");
                        put_line("             /        \            ");
                        put_line("             |        |            ");
                        put_line("             |        |            ");
                        put_line("            /|        |\           ");
                        put_line("           / |        | \          ");
                        put_line("          /  |        |  \         ");
                        put_line("         /___|________|___\        ");
                        put_line("                                   ");
                put_line("    Welcome to CS050!");
          end rocket;
end art;
```

```
-------------------------------------------------------------------
--      FILE : MAIN2.ADA
--      This program uses the art package.  It also uses the case
--      statment.
--      Fall 1993            S. Honda
-------------------------------------------------------------------
with text_io,art; use text_io,art;
procedure main2 is
     -- declaration of variables
   again, answer : character;
   begin -- main2
        loop
           put_line("         Menu Choices");
           put_line("  ============================");
           put_line("  (A) DESK       (F)  HOUSE2");
           put_line("  (B) TREE       (G)  TRACKS");
           put_line("  (C) HI         (H)  DOG   ");
           put_line("  (D) HOUSE      (I)  ROCKET");
           put_line("  (E) BOAT       (Q)  QUIT  ");new_line;
           put_line("What would you like to see ");
           put(" Type A,B,C,D,E,F,G,H,I OR Q  ==>  ");
           get(answer);new_line;
           case answer is
                when 'A' | 'a' => desk;
                when 'B' | 'b' => tree;
                when 'C' | 'c' => hi;
                when 'D' | 'd' => house;
                when 'E' | 'e' => boat;
                when 'F' | 'f' => house2;
                when 'G' | 'g' => tracks;
                when 'H' | 'h' => dog;
                when 'I' | 'i' => rocket;
                when 'Q' | 'q' => exit;      -- this will exit loop
                when others => put_line("    @#!!! wrong input !!!");
           end case;
           new_line;
        end loop;
        put_line("      That's all folks!");new_line;
end main2;
-------------------------------------------------------------------
                EXECUTION RUN OF MAIN2.ADA
-------------------------------------------------------------------
        Menu Choices
  ============================
  (A) DESK       (F)  HOUSE2
  (B) TREE       (G)  TRACKS
  (C) HI         (H)  DOG
  (D) HOUSE      (I)  ROCKET
  (E) BOAT       (Q)  QUIT

What would you like to see
 Type A,B,C,D,E,F,G,H,I OR Q  ==> h
      ************
     *    *       *
    *     *     O  *
    *     *          ******
    *      *            **
    *      *          |\/\/
     ******           |
       *              |/\/\
```

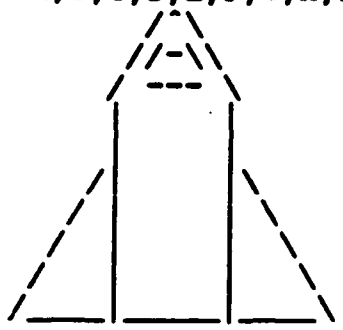


WOOF, WOOF!!

Menu Choices

(A)	DESK	(F)	HOUSE2
(B)	TREE	(G)	TRACKS
(C)	HI	(H)	DOG
(D)	HOUSE	(I)	ROCKET
(E)	BOAT	(Q)	QUIT

What would you like to see
Type A,B,C,D,E,F,G,H,I OR Q ==> I



Welcome to CS050!

Menu Choices

(A)	DESK	(F)	HOUSE2
(B)	TREE	(G)	TRACKS
(C)	HI	(H)	DOG
(D)	HOUSE	(I)	ROCKET
(E)	BOAT	(Q)	QUIT

What would you like to see
Type A,B,C,D,E,F,G,H,I OR Q ==> p
@#!!! wrong input !!!

Menu Choices

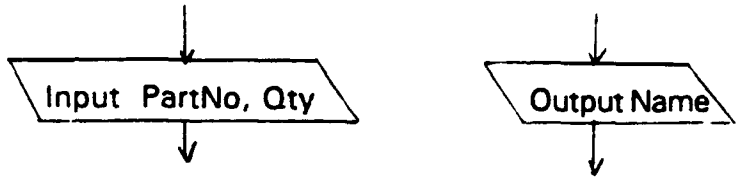
(A)	DESK	(F)	HOUSE2
(B)	TREE	(G)	TRACKS
(C)	HI	(H)	DOG
(D)	HOUSE	(I)	ROCKET
(E)	BOAT	(Q)	QUIT

What would you like to see
Type A,B,C,D,E,F,G,H,I OR Q ==> q
That's all folks!

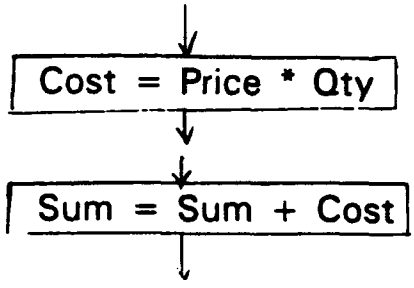
Flowcharting Symbols

1. Start/End

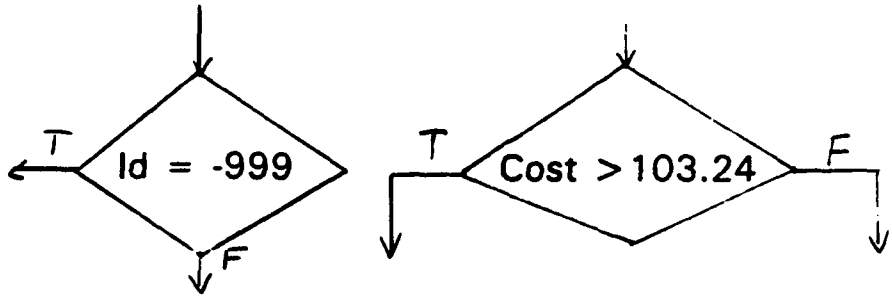
2. I/O



3. Processing



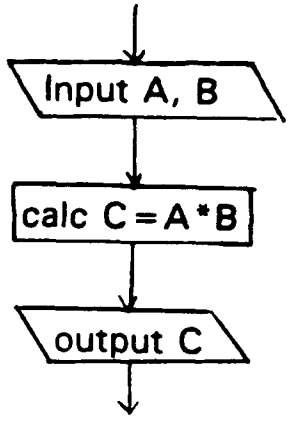
4. Decision Making



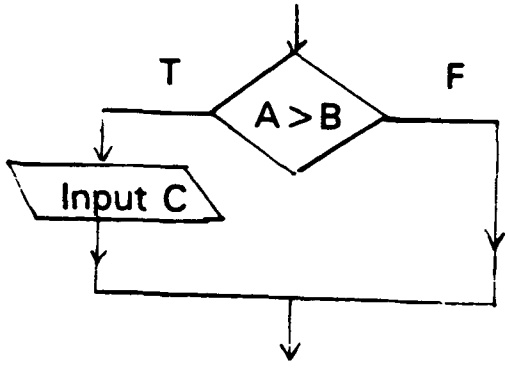
5. Connection

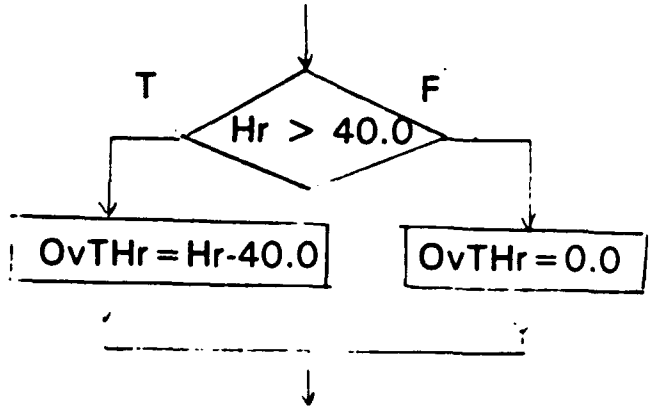
Logic Patterns

1. Simple Sequence

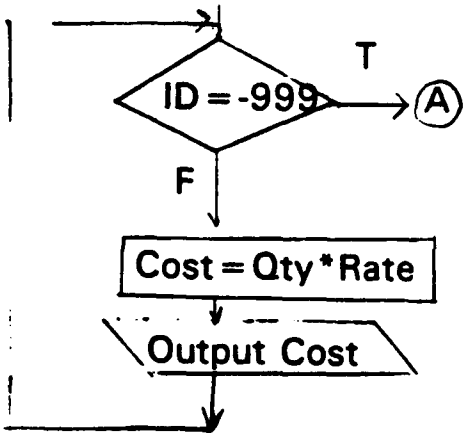


2 Decision Logic Pattern

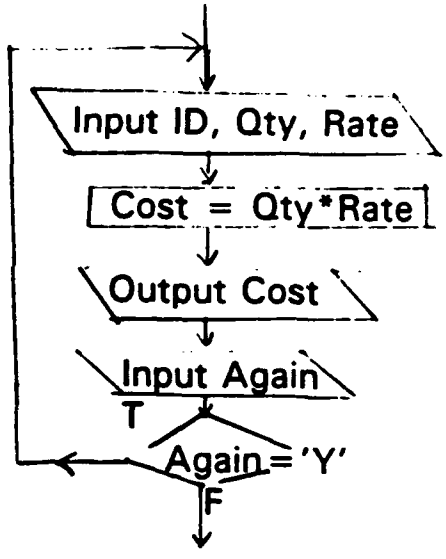




3. Repetition Logic Pattern



pre test



post test

Flow charting Problems

1. Input two integer numbers, A and B. If the first is greater than the second, print "Larger", otherwise print "Not Larger".

2. Input an ID Number, Rate of Pay, Hours Worked, and the Tax Rate. Calculate Gross Pay, Taxes owed, and the Net Pay. (Remember to pay Time and a Half for Over-Time Pay). Print out the Gross Pay, Taxes Owed, and the Net Pay.

3. You need an air conditioner. You need one between 5000 and 6000 BTU's. You want to find one with the highest Energy Efficiency Ratio (EER) because it will be the most economical to operate.

$$EER = \frac{\text{Number of BTU's}}{\text{Number of Watts}}$$

You are considering three air conditioners:

MODEL	BTU	WATTS
A	5000	820
B	6000	910
C	5500	850

Which one should you buy? Print out the EER values of all three Units and a message indicating the most efficient unit to operate, Model A, B, or C.

4. Input an ID Number, 3 Test Grades and a Final Exam. Each test is worth 20% and the Final Exam 40%. Calculate the Final Score. Print also the ID Number.

5. Suppose there are more students in the class (Referring to problem 4). I would like to process all the students in one execution run. Revise the flowchart such that many students could be processed. If the ID is equal to an END OF DATA TAG such as -999, there are no other students to process. -999 Marks the end of the data list. Modify the flowchart for Problem 4.

6. Do the same for problem 1 and 2 such that several sets of data can be processed.

7. Each salesperson earns a base salary of $185.00. Moveover, if a salesperson's total weekly sales exceeds $1000, a commission of 5.3% is earned on any amount up to $5000 and 7.8% is earned on any amount in excess of $5000. Determine the weekly Pay, for any Salesperson whose total weekly sales amount is input. Print out the Saleman's ID and his weekly sales and pay. Use an EOD TAG to

terminate the Program.

8. Input two Numbers X and Y. If the sum of X + Y is greater than 42, print out "42". If not, increase X by 10 and Y by 3. Print out the next X and Y values and check to see if the sum is greater than 42. Continue until the sum is greater than 42, at which time you need to output "42".

9. Input a list of numbers one at a time. The last value, not part of this list is 999. This marks the end of the data list (EOD). Print the smallest and the largest values of the list excluding the last value 999 which is not part of the list.

10. Input a list of Numbers whose EOD is 999. Print out the Largest and also a count of how many numbers are included in the list.

11. Several pairs (X,Y) of Numbers are to be input. Any pair with the first value equal to the second value serves as the End of Data (EOD). Determine and print out counts of how many pairs satisfy X<Y and how many pairs satisfy X>Y.

12. Follow the following algorithm:

    a. Input a value for N.
    b. If N is less than or equal to zero, goto step (a.)
    c. If N is greater than 100, output "Value is too Large" and goto step (g.)
    d. If N is greater than 90 calculate the SUM = 50+49+48+47+...+N and goto step (f.)
    e. Calculate the SUM = 1+2+3+4+...+N
    f. Output the value of SUM.
    g. Print out the message "Goodbye" and stop.

CS050
Handout Set #2

Handout #2

Predefined **Types** and **Subtypes** found in package **STANDARD**:

INTEGER
POSITIVE
NATURAL
FLOAT
DURATION
CHARACTER
STRING
BOOLEAN

(Note: Numeric values are implementation dependent)

Ada Data types

I. **Scalar Data Types**
   - A. ***Discrete Types***
     1. Integer
        - a. INTEGER
        - b. user defined
     2. Enumeration Types
        - a. CHARACTER
        - b. BOOLEAN
        - c. user defined
   - B. ***Real types***
     1. Floating point
        - a. FLOAT
        - b. user defined
     2. Fixed Point Types user defined

II. **Composite Types**
   - A. *Array*
   - B. *Record*

III. **Access Types**

IV. **Task Types**

V. **Private Types**
   - A. *Limited Private*
   - B. *Private*

Examples of Declaration Statements:

1. Examples of **Variable Declarations**: (Variables may be assign values in the program)

```
Hours : integer;
Rate, TaxRate, Taxes : float;
MiddleInitial, Again  : character;
Answer : boolean;
Weight, Age : positive;
```

2. Examples of **Constant Declarations**: (Constants do not change in the program)

```
Pi : constant float := 3.14;
BasePay : constant float := 185.00;
Greetings := constant string := "Sacred Heart University";
```

3. **Instantiation of generic packages** to allow for I/O (input/output):

```
package iio  is new integer_io(integer); use iio;          (allows for integer I/O)

package fltio  is new float _io(float); use fltio;   (allows for I/O of decimal values)

package Answer_io  is new enumeration_io( boolean ); use Answer_io;
                                          (Allows for I/O of boolean values)
```

Ada Executable Statements

1. **Assignment Statements** - Assigns a value to a variable. The value on the right is assigned to the variable on the left of :=

Syntax : variable := value, expression, or variable ;

Examples:
```
Rate := 12.43;
Hours := 40.0;
Gross := Rate * Hours;
Again := 'Y';
```

2. **Procedure Calls** - Procedures are invoked or called by using the procedure name.

Examples:
```
put("Hours Worked "); Put(Hours);
new_line; put_line("Wonderful Days!");
```

The procedures put, new_line, and put_line are invoked to output values

Examples:
```
put("Enter The Hours Worked ==> ");
get(Hours);
new_line;
```
The procedures put, get, new_line are invoked by using their names.

3. **If Statements** - Allows the computer to make decisions and take one of several paths.

```
if _____ then
        _____ ;
        _____ ;
          statement(s)
else
        _____ ;                    (optional clause)
        _____ ;
          statement(s)
end if;
```

4. **While loop** - Allows for repetition; The test to end the loop is at the top of the loop (for pre-test loops)

syntax:
```
while _____ loop
        _____ ;
        _____ ;
        _____ ;
          statement(s)
end loop;
```

5. **Loop-Endloop** - Allows for repetition; The test to end the loop is at the bottom. (for post test loops)

syntax:
```
loop
        _____ ;
        _____ ;
        _____ ;
          statement(s)
        exit when _____ ;
                  condition(s)
end loop;
```

Program I/O (Input/Output)

Generic Instantiation of data types other than character or strings.

Package Text_io allows for I/O of these DataTypes by providing some generic packages. Generic Packages are not directly usable. They must be instantiated with a particular data type before use. Instantiation creates a new package from the generic package which is like template. These new packages now contain the PUT and GET procedures that allow for I/O.

Generic Packages you may be using:

Float_io
Integer_io
Enumeration_io
Fixed_io

Syntax: type *IOTypeName* is new *genericpackage(datatype);*

examples:

```
type Int_io is new integer_io (integer): -- get,put integers
type Flt_io is new float_io (float);       -- get,put floats
subtype AgeType is integer range 1..104;
use Int_io, Flt_io;
Id    : integer;
Age   : AgeType;
Amt : float := 1.03e2;
.
.
.
get(Id);
get(Age); --Package Int_io allows I/O of integer subtypes
put(Amt);
```

or,

if the only variable requiring I/O is Age,

```
type Int_io is new integer_io (AgeType);
... get(Age); put(Age);
```

Ada Language Character Set used to build ADA words or lexical units

95 ASCII graphics character set

Basic Set:

26 upper case letters (A,B,C...Z)
10 digits (1,2,3,4,5,6,7,8,9,0)
19 special characters " # & ' ( ) * + , - . / : ; < = > _ |
and the space character

Also included in the Basic set are the following characters:

26 lower characters (a,b,c...z)
additional special characters ! $ % ? @ [\ ] ^ ` { } ~

Ada Lexical Units

are identifiers (including reserve words), numeric literals, character literals, string literals, delimiters and comments. A lexical unit must fit on one line. Embedded spaces are not permitted except in strings or comments.

Identifiers are names that are defined byt the progrrammer for such entities such as procedures, packages, tasks, variables, labels, functions, constants, etc. First characters of an identifier must be a letter followed by any number of letters or digits or an embedded underscore.

Reserved words are considered identifiers.

Numeric literals are Integer or Real literals

example of integer literals:

```
15   -23   123_423   1E3   42E2
```

examples of real literals:

```
15.0   -22.34   3.115_92   12.2E+4   41.35E-7
```

Other numeric literals in other bases:

```
2#101_111      8#57#      16#2F     -- Integer, decimal value 47
16#E#E1        2#1110_0000#       -- Integer, decimal value 224
```

(base) #Number Mantissa#optional power

examples of character literals

```
'A'   '+'   '"'   ''   '_'
```

examples of string literals

```
"Hello There"
```

DELIMITERS

& ' ( ) * , . + - / : ; < = > |

or these special compound symbols:

=> .. ** := >= <= << >> <> /=

COMMENTS start with two hyphens and extent to the end of the line.

Ada Operators

	Operators	Operation
	**	Exponentiation
	abs	absolute value
	not	logical negation
	rem	remainder
	mod	modulo
	/	division
	*	multiplication
	+	unary sign
	-	negation
	&	binary catenation
	-	subtraction
	+	addition
relational operators		
	>=	
	<	
	>	
	<=	
	=	
	/=	
logical	xor	exclusive or
	or	inclusive or
	and	conjunction

```
-------------------------------------------------------------
--    File : payroll.ada
--    This procedure does payroll for one user
--    Programmed by S. Honda      February 8, 1994
-------------------------------------------------------------
with text_io; use text_io;
procedure payroll is
      -- variable declaration
      gross,net,hours,rate,txrate,taxes,otp,regp : float;
      id : positive;
      -- generic instantiation for I/O
      package iio is new integer_io(integer);
      package fltio is new float_io(float);
      use iio,fltio;
      begin -- payroll
         put("Enter id ==> "); get(id);new_line;
         put("Enter hours worked in decimal format xx.x ==> ");
         get(hours);new_line;
         put("Enter rate per hour ==> $"); get(rate); new_line;
         put("Enter taxrate in decimal format x.xx ==> ");
         get(txrate);new_line;
         -- calculate overtime and regular pay
         if hours > 40.0 then
            regp := 40.0 * rate;
            otp  := (hours - 40.0) * rate * 1.5;
         else
            regp := hours * rate;
            otp  := 0.0;
         end if;
         -- calculate gross,taxes, and net
         gross := regp + otp;
         taxes := txrate * gross;
         net   := gross - taxes;
         -- output answers
         put_line("----------------------------");
         put("id ==> "); put(id); new_line;
         put("over time pay is ==> $");
         put(otp,5,2,0);new_line;
         put("regular pay is   ==> $");
         put(regp,5,2,0);new_line;
         put("tax amount is ==>$ "); put(taxes,4,2,0); new_line;
         put("net pay is ==> $");put(net,5,2,0); new_line;
         put_line("----------------------------");
         put_line("   END OF JOB!");
   end payroll;

-------------------------------------------------------------
--    TO COMPILE, LINK, AND RUN PROGRAM
-------------------------------------------------------------
sacredheart.shu> ada payroll.ada                    -- COMPILES
sacredheart.shu> ald -o payroll.exe payroll         -- LINKS
sacredheart.shu> payroll.exe                        -- RUNS PROGRAM
Enter id ==> 1111
Enter hours worked in decimal format xx.x ==> 10.0
Enter rate per hour ==> $ 5.00
Enter taxrate in decimal format x.xx ==> 0.50
----------------------------
id ==> 1111
over time pay is ==> $    0.00
regular pay is   ==> $   50.00
```

```
tax amount is ==>$   25.00
net pay is ==> $   25.00
-----------------------------
   END OF JOB!


sacredheart.shu> payroll.exe                    -- TO RUN PROGRAM AGAIN
Enter id ==> 2222
Enter hours worked in decimal format xx.x ==> 41.0
Enter rate per hour ==> $ 1.00
Enter taxrate in decimal format x.xx ==> 0.50
-----------------------------
id ==> 2222
over time pay is ==> $    1.50
regular pay is   ==> $   40.00
tax amount is ==>$   20.75
net pay is ==> $   20.75
-----------------------------
   END OF JOB!

-------------------------------------------------------------------
--    TO CAPTURE PROGRAM LISTING AND OUTPUT
-------------------------------------------------------------------
sacredheart.shu> payroll.exe>payroll.run
1111
10.0
5.00
0.50
sacredheart.shu> payroll.exe>payroll.run2
2222
41.0
1.00
0.50
sacredheart.shu> cat payroll.ada payroll.run payroll.run2 > payroll.lis
sacredheart.shu> lpr -Pvax1 payroll.lis
```

```
------------------------------------------------------------
--    File : payroll3.ada
--    This procedure does payroll for everyone
--    Programmed by S. Honda      February 8, 1994
------------------------------------------------------------
with text_io;  -- note that there is no use clause for this package
procedure payroll3 is

      -- variable declaration
   gross,net,hours,rate,txrate,taxes,otp,regp : float;
   id    : integer;
   again : character;

      -- generic instantiation for I/O
   package iio is new text_io.integer_io(integer);
   package fltio is new text_io.float_io(float);
       -- note no use clause for package iio and fltio

   begin -- payroll3
      loop
           -- prompt user for data
         text_io.put("Enter id ==> "); iio.get(id);
         text_io.new_line;
         text_io.put("Enter hours worked in decimal format xx.x ==> ");
         fltio.get(hours); text_io.new_line;
         text_io.put("Enter rate per hour ==> $"); fltio.get(rate);
         text_io.new_line;
         text_io.put("Enter taxrate in decimal format x.xx ==> ");
         fltio.get(txrate); text_io.new_line;

            -- calculate overtime and regular pay
         if hours > 40.0 then
            regp := 40.0 * rate;
            otp  := (hours - 40.0) * rate * 1.5;
         else
            regp := hours * rate;
            otp  := 0.0;
         end if;

            -- calculate gross,taxes, and net
         gross := regp + otp;
         taxes := txrate * gross;
         net   := gross - taxes;

             -- output answers
         text_io.put_line("-----------------------------");
         text_io.put("id ==> "); iio.put(id); text_io.new_line;
         text_io.put("over time pay is ==> $");
         fltio.put(otp,5,2,0); text_io.new_line;
         text_io.put("regular pay is   ==> $");
         fltio.put(regp,5,2,0); text_io.new_line;
         text_io.put("tax amount is ==>$ "); fltio.put(taxes,4,2,0);
         text_io.new_line;
         text_io.put("net pay is ==> $"); fltio.put(net,5,2,0);
         text_io.new_line;
         text_io.put_line("-----------------------------");
         text_io.put("Do you wish to do this again? (y/n) ==> ");
         text_io.get(again); text_io.new_line;
         exit when (again /= 'y');
      end loop;
```

```
      text_io.put_line("   END OF JOB!");
   end payroll3;

-------------------------------------------------------------------
--    TO COMPILE, LINK, AND RUN PROGRAM
-------------------------------------------------------------------
sacredheart.shu> ada payroll3.ada                    -- COMPILES
sacredheart.shu> ald -o payroll3.exe payroll3        -- LINKS
sacredheart.shu> payroll3.exe                        -- RUNS PROGRAM

Enter id ==> 1111
Enter hours worked in decimal format xx.x ==> 10.0
Enter rate per hour ==> $ 5.00
Enter taxrate in decimal format x.xx ==> 0.50
-----------------------------
id ==>   1111
over time pay is ==> $    0.00
regular pay is   ==> $   50.00
tax amount is ==>$   25.00
net pay is ==> $   25.00
-----------------------------
Do you wish to do this again? (y/n) ==> y
Enter id ==> 2222
Enter hours worked in decimal format xx.x ==> 50.0
Enter rate per hour ==> $ 1.00
Enter taxrate in decimal format x.xx ==> 0.10
-----------------------------
id ==>   2222
over time pay is ==> $   15.00
regular pay is   ==> $   40.00
tax amount is ==>$    5.50
net pay is ==> $   49.50
-----------------------------
Do you wish to do this again? (y/n) ==> n
   END OF JOB!
```

```
------------------------------------------------------------
--    File : payroll1.ada
--    This procedure does payroll for the entire company
--    Programmed by S. Honda      February 8, 1994
------------------------------------------------------------

with text_io; use text_io;
procedure payroll1 is
         -- variable declaration
      gross,net,hours,rate,txrate,taxes,otp,regp : float;
      id : integer;

          -- generic instantiation for I/O
      package iio is new integer_io(integer);
      package fltio is new float_io(float);
      use iio,fltio;

      begin -- payroll1
         -- prompt user for id
         put("Enter id (-999 to end) ==> "); get(id);new_line;

            -- create a loop to process payroll
         while id /= -999 loop
            put("Enter hours worked in decimal format xx.x ==> ");
            get(hours);new_line;
            put("Enter rate per hour ==> $"); get(rate); new_line;
            put("Enter taxrate in decimal format x.xx ==> ");
            get(txrate);new_line;

              -- calculate overtime and regular pay
            if hours > 40.0 then
               regp := 40.0 * rate;
               otp  := (hours - 40.0) * rate * 1.5;
            else
               regp := hours * rate;
               otp  := 0.0;
            end if;

               -- calculate gross,taxes, and net
            gross := regp + otp;
            taxes := txrate * gross;
            net   := gross - taxes;

               -- output answers
            put_line("-----------------------------");
            put("id ==> ");put(id);new_line;
            put("over time pay is ==> $");
            put(otp,5,2,0);new_line;
            put("regular pay is   ==> $");
            put(regp,5,2,0);new_line;
            put("tax amount is ==>$ "); put(taxes,4,2,0); new_line;
            put("net pay is ==> $");put(net,5,2,0); new_line;
            put_line("-----------------------------");
               -- prompt user for next id
            put("Enter id (-999 to end ==> "); get(id);new_line;
         end loop;

         put_line("   END OF JOB!");
      end payroll1;
```

```
-------------------------------------------------------------------
--    TO COMPILE, LINK, AND RUN PROGRAM
-------------------------------------------------------------------
sacredheart.shu> ada payroll1.ada                    -- COMPILES
sacredheart.shu> ald -o payroll1.exe payroll1        -- LINKS
sacredheart.shu> payroll1.exe                        -- RUNS PROGRAM

Enter id (-999 to end) ==> 111
Enter hours worked in decimal format xx.x ==> 10.0
Enter rate per hour ==> $ 5.00
Enter taxrate in decimal format x.xx ==> 0.50
------------------------------
id ==>   1111
over time pay is ==> $    0.00
regular pay is   ==> $   50.00
tax amount is ==>$   25.00
net pay is ==> $   25.00
------------------------------
Enter id (-999 to end ==> 2222
Enter hours worked in decimal format xx.x ==> 41.0
Enter ~~te per hour ==> $ 1.00
Enter    rate in decimal format x.xx ==> 0.50
-----    --------------------
id ==    2222
over ~    pay is ==> $    1.50
regular   ay is   ==> $   40.00
tax amou.t is ==>$   20.75
net pay is ==> $   20.75
------------------------------
Enter id (-999 to end ==>
   END OF JOB!
```

CS050
Handout Set #3

Handout #3

String Variable Declaration - Strings are declared as an array of characters. It contains multiple storage spaces for characters.

Syntax:

string variable name : string(1..10); -- allows 10 characters

examples:

```
FirstName : string(1..12);
Address   : string(1..24);
zip       : string(1..5) := 06611;  -- initialize zip
```

Enumeration Type Declaration - User may define their own data type. The data values must be enumerated in a ***type declaration statement*** in a particular order. In order to accomplish I/O for each different type, the user must instantiate each enumerated data type.

Syntax: type *EnumeratedTypeName* is (*datatype values*);
VariableName : EnumeratedTypeName;

examples:

```
type FishType is (cod, salmon, mahimahi, ahi, catfish);
Fish : FishType;
type ComputerType is (IBM, Apple, Dec, Compaq);
Computer:ComputerType := Dec;  -- initialize Computer
```

Generic Instantiation of FishType and Computer Type - To allow for I/O

Syntax: type *InstTypeName* is new *genericpackage(datatype);*

examples:

```
type Fish_io is new enumeration_io(FishType):
type Computer_io is new enumeration_io(ComputerType);
use Fish_io, Computer_io;
```

```
type day_type is (monday,tuesday,wednesday,thursday,
        friday,saturday,sunday);
day : day_type;
...
```

if statement

```
if day = saturday then
    put_line("Play");
else
    if day=sunday then
        put_line("Sleep");
    else
        if day = friday then
            put_line("Call in sick");
            put_line("Play computer games!");
        else
            put_line("Go to work!");
        end if;
    end if;
end if;
```

if's with elsif clauses

```
if day= saturday then
    put_line("Play");
elsif day=sunday then
    put_line("Sleep");
elsif day = friday then
    put_line("Call in sick");
    put_line("Play computer games!");
else
    put_line("Go to work!");
end if;
```

Case Statement

```
case day is
    when saturday=>     put_line("Play");
    when sunday  =>     put_line("Sleep");
    when friday  =>     put_line("Call in sick");
                        put_line("Play computer games!");
    when others  =>     put_line("Go to work!");
end case;
```

Case Statement

- ***The Case expression must be of a discrete type***
- ***Each of the possible values of the case expression must be covered in one and only one when clause.***
- ***If the when other clause is used, it must appear as a single choice at the end of the case statement***
- ***Choices in a when clause must be static***

Case Statement Example

```
atomic_number : integer range 1..105;
. . .
case atomic_number  is
      when 1
                  => put("Hydrogen");
      when 2| 10 | 18 | 36 | 54 | 86
                  => put("Noble Gas");
      when 3 | 11 | 19 | 37 | 55 | 87
                  => put("Alkali Metal");
      when 4 | 12 | 20 | 38 | 56 | 88
                  => put("Alkaline Earth Metal");
      when 5 | 13 | 31 | 49 | 81
                  => put("Aluminum Family");
      when 6 | 14 | 32 | 50 | 82
                  => put('Carbon Family ");
      when  7 | 15 | 33 | 51 | 83
                  => put("Nitrogen Family");
      when  8 | 16 | 34 | 52 | 84
                  => put("Chalcogen");
      when 9 | 17 | 35 | 53 | 85
                  => put("Halogen");
      when 58..71
                  => put("Rare Earth");
      when 90..103
                  => put("Actinide");
      when others
                  => put("Transition Metal");
end case;
```

Which Case Statements are Legal?

```
response : character;
...
get(response);
case response is.                    -- using selectors
      when 'Y' | 'y'   =>   put_line("You are begin positive");
      when 'N' | 'n'   =>   put_line("You are being negative");
end case;
```

```
type day_of_week_type is (monday,tuesday,wednesday,thursday,
                          friday,saturday,sunday);
day : day_of_week_type;
. . .
case day is                -- using discrete range
      when monday..thursday =>  put("Go to work!");
      when friday           => put("Take sick leave");
      when saturday..Sunday  =>  put("Watch TV");
end case;
```

```
subtype age_type is float range 0.0..14.0;
age : age_type
. . .
case age is
      when 0.0 .. 12.0  =>   put("Child");
      when others       =>   put("Teeny-Bopper");
end case;
```

```
max:integer;
get(max);
case value is
      when 1 .. max     =>   put("In range");
      when others       =>   put("Out of range");
end case;
```

```
max : integer := 5;
value : integer :=2;
. . .
case value is
      when 1 .. max     =>   put("In range");
      when others       =>   put("out of range");
end case;
```

CS050
Handout Set #4

Handout #4

Logical Expressions

Logical Expressions (sometimes called boolean expressions) are expressions that evaluate to boolean values true or false. Boolean Variables may be assign to boolean constant values of true or false.

In a declaration section where

```
Again : boolean ;
A,B   : integer ;
```

In the executable body where

```
Again := true ;

while Again loop
      statement(s) ;
      put("Do you wish to repeat this? (true or false ") ;
      get(Again) ;
end loop;

if A >500 then
      statement(s) ;
end if ;
```

In the above examples **Again := true**, **Again**, and **A>500** are examples of Logical or Boolean Expressions. Both expressions evaluate to true or false.

Sequential File Processing in Ada

The following subprograms from Text_io are generally used in sequential file processing:

Procedures	**open, close, create**
Function	**end_of_file**

1. The **open procedure** allows a physical data file to be opened and used for input.

   syntax:
   open(*LogicalFileName*, *filemode*, *filename*);

   Where *LogicalFileName* is an identifier that is declared as a **file_type**,
   filemode is **In_file** *or* **out_file**, and
   filename is the physical file found on secondary storage.
   The name of the physical file can be enclosed in quotes, or the name may be stored in a string variable.

   Examples:

```
Sam       : file_type;
. . .           . . .
open(Sam, in_file, "payroll.dat");
```

   or

```
Sam       : file_type;
FName     : string(1..11);
. . .
put("Enter filename, [exactly 11 characters please] ");
get(FName); new_line;
open(Sam, in_file, FName);
```

2. The **close procedure** closes an opened file. Remember to close all opened files in your program when not needed.

   syntax: **close**(*LogicalFileName*);

   Example: **close**(Sam);

3. The test for end of file can be done with the function **end_of_file**. This function returns a value of **true** if the end of the file is reached; otherwise it returns a value of **false**.

Example:

```
Sam, Joe   : file_type;

FileName   : string(1.11);
PayRate    : float;
. . .
open(Sam, in_file, Filename);
open(Joe, in_file, "acctpay.dat");
while not end_of_file(Sam)  loop
      get(Sam, Id);
      get(Joe,Id); get(Joe,PayRate);
      . . .
end loop;
close(Sam);
close(Joe);
```

4. The **create** procedure will create a file for output.

syntax: **create**(*LogicalFileName, filemode , filename);*

Example:

```
OutFile     : file_type;
Rate        : float ;
Hr          : float ;
. . .
-- prompt user for data
put("Enter Pay rate = = > ");
get(Rate); new_line;
put("Enter Hours worked = = > ");
get(Hr); new_line;
create(OutFile, out_file, "PayFile");
put(OutFile, "Pay file - 1993");   -- header in 1st record of file
put(OutFile, Rate);                -- writes pay rate to file
put(OutFile, Hr);                  -- writes hours worked to file
. . .
close(OutFile);
```

COMPARING

Instructions

A report of the book royalties for authors is to be prepared. A program should be designed and coded to produce the report.

Input

Input consists of sales records that contain the author's name, the title of the book, and the number of books sold. The input data is shown below.
end of file indicator.

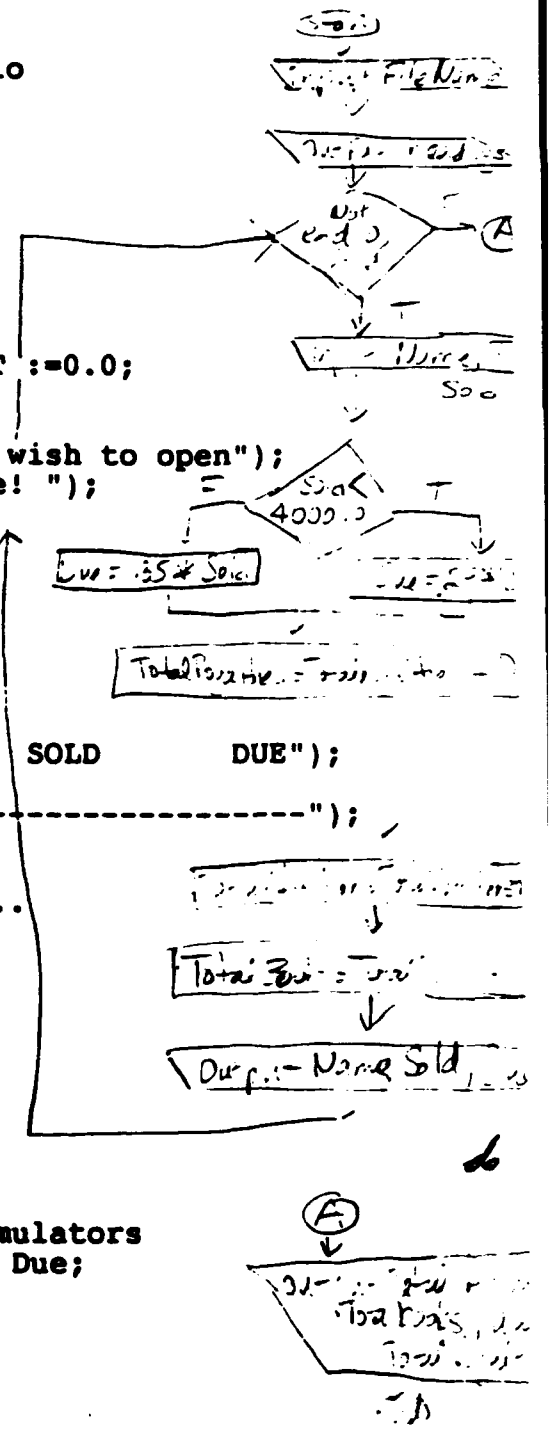
NAME	TITLE	NUMBER SOLD
BROWN	BASIC	3999
DAVIS	COBOL	5679
EVANS	PASCAL	4397
LAMB	PILOT	3867

Output

Output is a book royalty report containing the author's name, the title of the book, the number of copies sold, and the royalty. If a book sells less than 4,000 copies, a royalty of 29 cents per book is paid. If a book sells 4,000 copies or more, a royalty of 35 cents per book is paid. After all records have been processed, the total number of authors, the total number of books sold, and the total royalties paid to all authors are to be displayed. The format of the output is illustrated below.

```
                    BOOK ROYALTIES

NAME            TITLE           SOLD        DUE

BROWN           BASIC           3999        1159.71
DAVIS           COBOL           5679        1987.65
EVANS           PASCAL          4397        1538.95
LAMB            PILOT           3867        1121.43

TOTAL AUTHORS     4
TOTAL BOOKS       17942
TOTAL ROYALTIES   5807.74
```

```
--------------------------------------------------------------------
--      DATA FILE : COMUPTER.DAT
--------------------------------------------------------------------
Brown           Basic    3999.0
Davis           Cobol    5679.0
Evans           Pascal 4397.0
Lamb            Pilot   3867.0
--------------------------------------------------------------------
--      DATA FILE : MATHEMAT.DAT
--------------------------------------------------------------------
SMITH   DISCRETE MATH    1000.0
YEE     CALCULUS           3433.0
KINIK   ALGEBRA I        10000.0
KINIK   GEOMETRY           3001.0
GROUCHY MATH IS FUN      7012.0
SELLS   PROBABILITY      1234.0
--------------------------------------------------------------------
--      PROGRAM : BOOK.ADA
--------------------------------------------------------------------
with text_io; use text_io;
procedure book is
                -- generic instantiation of float_io
        package fltio is new float_io(FLOAT);
        package iio is new integer_io(NATURAL);
        use iio, fltio;
                --  declare variables
        Name                            : STRING(1..8);
        FileName                        : STRING(1..12);
        Title                           : STRING(1..14);
        TotalAuthors                    : NATURAL := 0;
        Sam                             : FILE_TYPE;
        TotalBooks,Sold,Due,TotalRoyalties  : FLOAT :=0.0;
begin -- book
                -- prompt user for input file name
        put_line("What is the name of the file you wish to open");
        put("Type no more than 12 characters please! ");
        get(FileName);
        put(FileName);
        new_line;
                -- open the input file
        open(Sam,IN_FILE,FileName);
                -- put headers
        put_line("                BOOK ROYALTIES");
        new_line;
        put_line(" NAME          TITLE                  SOLD       DUE");
        new_line;
        put_line("-----------------------------------------------");

        while not end_of_file(Sam) loop
                        -- begin reading a record...
                get(Sam,Name);
                get(Sam,Title);
                get(Sam,Sold);
                new_line;
                        -- calculate royalities
                if Sold < 4000.0 then
                        Due := 0.29 * Sold;
                else
                        Due := 0.35 * Sold;
                end if;
                        -- update counter and accumulators
                TotalRoyalties := TotalRoyalties + Due;
                TotalAuthors := TotalAuthors + 1;
                TotalBooks := TotalBooks + Sold;
                        -- output information
                put(Name);put("   ");
```



```
                put(Title);put("  ");
                put(Sold,fore=>8,aft=>2,exp=>0);
                put(Due,fore=>8,aft=>2,exp=>0);
                new_line;
                        --get name from next record
        end loop;
        close(Sam);
        new_line;
        put(" Total Authors ");
        put(TotalAuthors,width=>4);new_line;
        put(" Total Books ");
        put(TotalBooks,fore=>8,aft=>0,exp=>0);
        new_line;
        put(" Total Royalties due $");
        put(TotalRoyalties,fore=>8,aft=>2,exp=>0);
        new_line;
        put_line("     End of Job");new_line;
end book;
-------------------------------------------------------------------
--      COMPILATION,LINKING, AND EXECUTION RUN OF BOOK.ADA
-------------------------------------------------------------------
SHU> ada book.ada  [ENTER]
SHU> ald book      [ENTER]
SHU> a.out         [ENTER]

What is the name of the file you wish to open
Type no more than 12 characters please! computer.dat
             BOOK ROYALTIES

 NAME          TITLE               SOLD      DUE

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

Brown              Basic        3999.00    1159.71

Davis              Cobol        5679.00    1987.65

Evans              Pascal       4397.00    1538.95

Lamb               Pilot        3867.00    1121.43

 Total Authors    4
 Total Books    17942.0
 Total Royalties due $    5807.74
     End of Job

-------------------------------------------------------------
--      ANOTHER RUN OF BOOK.ADA USING DIFFERENT INPUT FILE
-------------------------------------------------------------
SHU> a.out   [ENTER]

What is the name of the file you wish to open
Type no more than 12 characters please! mathemat.dat
             BOOK ROYALTIES

 NAME          TITLE               SOLD       DUE

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

SMITH      DISCRETE MATH        1000.00     290.00

 YEE        CALCULUS            3433.00     995.57

 KINIK      ALGEBRA I          10000.00    3500.00

KINIK      GEOMETRY             3001.00     870.29
```

GROUCHY	MATH IS FUN	7012.00	2454.20
SELLS	PROBABILITY	1234.00	357.86

Total Authors 6
Total Books 25680.0
Total Royalties due $ 8467.92
End of Job

BASIC ARITHMETIC OPERATIONS — ACCUMULATING FINAL TOTALS

Instructions

A report of the payments for a gasoline charge card system is to be prepared. A program should be designed

Input

Input consists of sales records containing the customer number, the customer name, the previous balance, the current purchases, and the payments made. The input data is shown below

CUSTOMER NUMBER	CUSTOMER NAME	PREVIOUS BALANCE	CURRENT PURCHASES	PAYMENTS
201	DRAKE	100.25	15.75	15.75
234	HOLLY	175.75	22.25	21.25
366	LOOMS	238.75	46.75	50.25
436	RAMOS	135.75	17.25	17.75

Output

Output is a report of payments due. The report is to contain the customer name, the current balance, and the minimum payment due. The current balance is obtained by adding the previous balance to the current purchases, and subtracting the payments. The minimum payment due is obtained by taking 16% of the current balance. After all records have been processed, the total number of customers and the total current balance of all customers are to be displayed. The format of the output is illustrated below.

```
                CREDIT CARD SUMMARY

CUSTOMER          CURRENT          MINIMUM
  NAME            BALANCE          PAYMENT

DRAKE              100.25            16.04
HOLLY              174.75            28.28
LOOMS              238.25            37.64
RAMOS              135.25            21.64

TOTAL CUSTOMERS 4
TOTAL - CURRENT BALANCES $ 647.5
```

BASIC ARITHMETIC OPERATIONS — ACCUMULATING FINAL TOTALS

Instructions

A report is to be prepared for a transportation company to determine the miles per gallon obtained from each bus driven for the day. A program should be designed

Input

Input consists of records that contain the bus driver's name, identification number, the miles driven, and the gallons of gasoline used. The input data is shown below

DRIVER ID	DRIVER NAME	MILES	GALLONS
15	C DAVIS	250	25
31	R ROAMS	462	42
46	G GROLS	336	26
87	L JAMES	255	17

Output

Output is a bus mileage report that contains the driver's name, the miles traveled, the gallons of gasoline used, and the miles per gallon. The miles per gallon is obtained by dividing the gallons used into the miles traveled. After all records have been processed, the total number of drivers, the total miles traveled, the total gallons used, and the average miles per gallon for all buses are to be displayed. The format of the output is illustrated below.

```
                    BUS MILEAGE

DRIVER          MILES          GALLONS          MPG

C DAVIS         253            25               10
R ROAMS         462            42               11
G GROLS         336            26               13
L JAMES         255            17               15

TOTAL DRIVERS 4
TOTAL MILES 1306
TOTAL GALLONS 110
AVERAGE MPG 11.8636
```

A school financial assistance report is to be prepared for the veterans enrolled in a college. A program should be designed to produce the report.

INPUT

Input consists of records containing the name of the student, a code (code 1 if the student is a non-veteran; code 2 if the student is a veteran), the number of units the student is enrolled for, and the number of dependents. All records will contain either a code 1 or a code 2, except for the end of file record. The input data is shown below.

STUDENT NAME	CODE	UNITS	DEPENDENTS
LOGUE	1	15	0
JAMES	2	16	1
ORGIV	2	15	3
MANLEY	2	12	1
SAUL	2	14	5

OUTPUT

Output is a financial assistance report of all veterans. If the individual is not a veteran (code 1), that their name is not to appear on the report. If the individual is a veteran, is a part-time student (taking less than 15 units) and has less than two dependents, then the financial assistance is $20.00 per unit being taken. If the individual is a part-time student and a veteran with two or more dependents, the financial assistance is $23.00 per unit being taken. If the veteran is a full-time student (15 units or more) with two or more dependents, then the financial assistance is $30.00 per unit being taken. If the veteran is a full-time student with less than two dependents, the financial assistance is $27.00 per unit being taken. After all records have been processed, the total number of full-time veterans, the total number of part-time veteran students, and the total amount of financial assistance should be displayed. The format of the output is illustraded below.

VETERANS
FINANCIAL ASSISTANCE

NAME	STATUS	DEPS	AMOUNT
JAMES	FULL TIME	1	$ 432
ORGIV	FULL TIME	3	$ 450
MANLEY	PART TIME	1	$ 240
SAUL	PART TIME	5	$ 322

TOTAL-FULL TIME VETERAN STUDENTS 2
TOTAL-PART TIME VETERAN STUDENTS 2
TOTAL AMOUNT $ 1444

CS050
Handout Set #5

Handout #5

Ada Subprograms (Procedures and Functions)

A **subprogram** is a sequence of code, that allows programs to be structured in modular style. It also supports code reuse. Code that can be used over again should be written as subprograms and housed in packages. A complex problem can be broken down into smaller manageable pieces.

All Ada subprograms, **functions** or **procedures** have the same basic parts. (See next page). Identifiers that are used only within a subprogram should be defined locally within the subprogram. Identifiers that are used in the main procedure as well as in several subprograms should be defined globally in the main subprogram. See text to review visibility rules.

A flowchart of a problem that use macro modules can be written as **subprograms** that can be farmed to several programmers.

The major differences between **Functions** and **Procedures** are:

Functions

- return a value when invoked.
- are called within a statement or as part of another unit.
- contain at least one **return statement** in the executable body.

Procedures

- return zero, one or more values.
- are invoked by using the procedure name.
- Do not contain return statements

Parameter passing allows different values to be used when a **subprogram** is invoked at different times. The different types of parameter modes include **in, out** and **inout**. See text to review parameter modes.

Ada Procedure

Header	-- identifies the procedure
Declaration Section	types (enumerated) array types subtypes constants variables *subprogram declaration (Functions or Procedures) instances of generic packages or subprograms
Executable Body	executable ada instructions or statements subprogram calls

---

*Subprogram defined in the declaration Section

Function:

FUNCTION ***FuncName*** **(*Param1:*** in *DataT1*; ***Parm2:*** in *DataT2*)return *DataT3* is
Formal Parameter List

LOCAL DECLARATIONS
BEGIN -- FuncName
-- executable statements
return _____ ;
END FuncName;

Procedure:

PROCEDURE ProcName (*Parm1*: in *DataT1*; ***Parm2***:out *DataT2*; ***Parm3***:in out *DataT3*) is
Formal Parameter List

LOCAL DECLARATIONS
BEGIN -- ProcName
-- executable statements
END ProcName;

3 Possible MODES for Parameter Lists

in	The formal parameter is treated as a constant whose value is provided by the corresponding actual parameter. This parameter cannot be changed by the subprogram.
out	The parameter is a variable whose value is assigned to the corresponding actual parameter as a result of the procedure.
in out	The formal parameter is a variable whose corresponding actual parameter value can be referenced and updated.

Parameter Passing Modes

in

- has initial value
- read only
- cannot alter value
- default mode
- actual parameter can be an expression

out

- has no initial value
- write only
- must assign value
- actual parameter must be a variable

in out

- has initial value
- read and write
- can alter value
- actual parameter must be a variable

```
------------------------------------------------------------------
--      FILE : TAXES.ADA
------------------------------------------------------------------
--      This procedure declares a function called taxcalc that
--      calculates the tax return, if you give it 4 PARAMETERS:
--              1.  the TAXABLE IN of a person(s)
--              2.  the AMOUNT of base taxes you need to pay for that
--                      income range
--              3.  the PERCENTAGE amount in decimal format
--              4.  the Amount over value
--      It will then calculate your taxes and return that amount!
------------------------------------------------------------------
with text_io; use text_io;
procedure taxes is
        Mr,Mrs,MrTax,MrsTax,JointTax : float;
        Temp : float;
        package fltio is new float_io(float);
        use fltio;
        ------------------------------------------------------------
        function taxcalc(Income,Amt,Percent,Over:float) return float is
                taxamt : float;
                begin -- taxcalc
                        taxamt := Amt + Percent*(Income-Over);
                        return taxamt;
                end taxcalc;
        ------------------------------------------------------------
begin -- taxes
        put("Enter Mr. Doe's Taxable Income => $");
        get(Mr);put(Mr,fore=>6,aft=>2,exp=>0);new_line;
        put("Enter Mrs. Doe's Taxable Income => $");
        get(Mrs);put(Mrs,fore=>6,aft=>2,exp=>0);new_line;
                -- calc taxes invoking function taxcalc
        MrsTax := TAXCALC(Mrs,2190.0,0.32,20000.0);
        MrTax :=  TAXCALC(Mr,1630.0,0.28,18000.0);
        JointTax :=TAXCALC(Mr+Mrs,3960.0,0.29,36000.0);
                -- output returns
        put("Mr. Doe's Separate Tax return will be $");
        put(MrTax,fore=>7,aft=>2,exp=>0);new_line;
        put("Mrs. Doe's Separate Tax Return will be $");
        put(MrsTax,fore=>7,aft=>2,exp=>0);new_line;
        put("Their joint tax return will be => $");
        put(JointTax,fore=>7,aft=>2,exp=>0);new_line;
                -- decide which way to file
        if MrTax + MrsTax > JointTax then
                put_line("File Joint Return");
        else
                if MrTax + MrsTax < JointTax then
                        put_line("File Separate Returns");
                else
                        put_line("File anyway you like!");
                end if;
        end if;
        new_line;
end taxes;
------------------------------------------------------------------
--      EXECUTION RUN OF TAXES.ADA
------------------------------------------------------------------
Enter Mr. Doe's Taxable Income => $ 18750.0
Enter Mrs. Doe's Taxable Income => $ 20312.0
Mr. Doe's Separate Tax return will be $   1840.00
Mrs. Doe's Separate Tax Return will be $   2289.84
Their joint tax return will be => $   4847.98
File Separate Returns
```

```
-------------------------------------------------------------------
--    FILE : paint.ada
--    This main procedure illustrates the use of subprograms.
--    It contains a function and 3 procedures in the main declaration
--    area of the main procedure.          S. Honda     3/93
-------------------------------------------------------------------
with text_io; use text_io;
procedure paint is
      package fltio is new float_io(float);
      use fltio;
          -- global declarations
      length,width,area : float;
      costpergal,no_gal,costpaint : float;
      ---------------------------------------------------------------
      function painta(wid,len:in float) return float is
               -- local declarations
               height : constant float := 8.0;   -- height of room 8 ft.
               wall1,wall2,ceiling,totalsqft : float;
               begin -- painta
                    wall1 := len*height;
                    wall2 := wid*height;
                    ceiling := wid*len;
                    totalsqft  := 2.0*(wall2 + wall1) + ceiling;
                    return totalsqft;
               end painta;
      ---------------------------------------------------------------
      procedure getdata(l,w,costpergal : out float) is
               begin -- getdata
                     put("Enter length of room in ft. ==> ");
                     get(l); new_line;
                     put("Enter width of room in ft. ==> ");
                     get(w); new_line;
                     put("Enter cost of paint per gallon ==> ");
                     get(costpergal); new_line;
                end getdata;
      ---------------------------------------------------------------
      procedure calc(area,costpergal: in float; no_gal,costpaint
                : out float) is
                -- local declaration

               coverage : constant float :=233.0;
               temp : float;
               begin -- calc
                    temp    := area/coverage;
                    temp    := float(integer(temp + 0.5));
                    no_gal  := temp;
                    costpaint := temp*costpergal;
               end calc;
       ---------------------------------------------------------------
       procedure print(w,l,area,no_gal,costpaint:in float) is
               begin -- print
                    put_line("The paint used to paint the room covers");
                    put_line("233 sqft.  The room to be painted is a");
                    put_line("standard 8 feet high room!");  new_line;
                    put("The length of room is ");
                    put(l,2,1,0);put_line(" ft");
                    put("The width of room is ");
                    put(w,2,1,0);put_line(" ft");
                    put("The area to be painted is ");
                    put(area,4,1,0);put_line(" sqft.");
```

```
                    put("The cost of paint is $");
                    put(costpaint,4,2,0);new_line;
                    put("The no of gallons to buy : ");
                    put(no_gal,2,0,0);new_line;
                end print;
        ------------------------------------------------------------
        begin -- paint
             getdata(length,width,costpergal);
             area := painta(width,length);
             calc(area,costpergal,no_gal,costpaint);
             print(width,length,area,no_gal,costpaint);
        end paint;
-------------------------------------------------------------------------
--    EXECUTION RUN OF PROGRAM
-------------------------------------------------------------------------
shu.sacredheart.edu > ada paint.ada                 -- COMPILES
shu.sacredheart.edu > ald -o paint.exe paint        -- LINKS
shu.sacredheart.edu > paint.exe                     -- RUNS EXECUTABLE FILE

Enter length of room in ft. ==> 10.0
Enter width of room in ft. ==> 10.0
Enter cost of paint per gallon ==> 10.00
The paint used to paint the room covers
233 sqft.  The room to be painted is a
standard 8 feet high room!

The length of room is 10.0 ft
The width of room is 10.0 ft
The area to be painted is  420.0 sqft.
The cost of paint is $  20.00
The no of gallons to buy :  2.0
-------------------------------------------------------------------------
shu.sacredheart.edu > paint.exe                     -- RUNS EXECUTABLE FILE

Enter length of room in ft. ==> 1.0
Enter width of room in ft. ==> 1.0
Enter cost of paint per gallon ==> 10.00
The paint used to paint the room covers
233 sqft.  The room to be painted is a
standard 8 feet high room!

The length of room is  1.0 ft
The width of room is  1.0 ft
The area to be painted is   33.0 sqft.
The cost of paint is $  10.00
The no of gallons to buy :  1.0
```

File : COMPUTER.DAT

```
BROWNBASIC  100
WHITECOBOL  6000
GREENRPG    1000
EVANSADA    6080
```

File : BOOK.ADA

```
-------------------------------------------------------------
--  This program is the book royalty problem that has been
--  modularized.  Note the three procedures (Headers, Loops,
--  PrintTotals).
-------------------------------------------------------------
with text_io; use text_io;
procedure book is
        package iio is new integer_io(integer);
        package fltio is new float_io(float);
        use iio,fltio;
        TotalAuthors,TotalBooks : natural :=0;
        TotalRoyalties          : float   :=0.0;

        ---------------------------------------------------------
        procedure Headers is
                begin -- Headers
                        put_line("              Book Royalties");
                        new_line;
                        put(" Name      Title      ");
                        put_line(" Sold  Due");
                        new_line;
                        put("-----------------------------");
                        put_line("-----------");
                end Headers;
        ---------------------------------------------------------
        procedure Loops(TotalAuthors,TotalBooks:out natural;
                        TotalRoyalties: out float) is
                ta,tb,Sold : natural := 0;
                due,tr : float := 0.0;
                Name    : string(1..5);
                Title   : string(1..6);
                Sam     : file_type;
                begin --  Loops
                        open(Sam,in_file,"computer.dat");
                        while not end_of_file(Sam) loop
                                get(Sam,Name);
                                get(Sam,Title);
                                get(Sam,Sold);
                                new_line;
                                if Sold < 4000 then
                                        Due := 0.29 * Float(Sold);
                                else
                                        Due := 0.35 * Float(Sold);
                                end if;
                                tr := tr + Due;
                                ta := ta + 1;
                                tb := tb + Sold;
                                put(Name);put("    ");
                                put(Title);put("    ");
                                put(Sold);
                                put(Due,8,2,0);
                                new_line;
                        end loop;
```

```
                close(Sam);
                TotalAuthors := ta;
                TotalRoyalties := tr;
                TotalBooks := tb;
        end Loops;
-------------------------------------------------------------
-- Note the parameters passed have different names.
procedure PrintTotals(TA,TB: in natural; TR: in float) is
        begin --PrintTotals
                new_line;
                put(" Total Authors ");
                put(TA,width=>4);new_line;
                put(" Total Books Sold ");
                put(TB);new_line;
                put(" Total Royalties due $");
                put(TR,8,2,0);new_line;
                put_line("End of Job!");
        end PrintTotals;
-------------------------------------------------------------
begin -- book
        Headers;
        Loops(TotalAuthors,TotalBooks,TotalRoyalties);
        PrintTotals(totalAuthors,TotalBooks,TotalRoyalties);
end book;
```

Execution Run of Book.exe:

```
             Book Royalties

 Name      Title       Sold  Due

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

BROWN     BASIC             100       29.00

WHITE     COBOL            6000     2100.00

GREEN     RPG              1000      290.00

EVANS     ADA              6080     2128.00

 Total Authors     4
 Total Books Sold       13180
 Total Royalties due $    4547.00
End of Job!
```

Ada Executable Statement

6. **for loops** - Allows for repetition. It automatically increments with a starting value, to an ending value and performs a pre-test.

syntax:

```
for ForVariable in Initialvalue .. Endingvalue loop
    statement(s) ;
    . . .
end loop;
```

Note: **ForVariable** does not need to be declared. It will cease to exist after **endloop**. **ForVariable** will take on the value of **Initialvalue**. Initialvalue and Endingvalue must be discrete scalar constant values, expressions or variables. It may not be any other data type such as **real** or **composite**. **ForVariable**, **Initialvalue**, **Endingvalue** must be of the same data type.

Examples:

1.
```
-- prints message in put_line 10 times
for i in 1..10 loop
        put_line("I love to program in Ada!");
end loop;
```

2.
```
-- Prints 3 and 4 Times table
for i in 3..4 loop
        put("Set  ");
        put (i);
        put_line(" times table ");
        for j in 1..12 loop
                put(j); put(" times "); put(i); put(" = "); put(i*j); new_line;
        end loop;
        new_line;
end loop;
```

3.
```
type food_type is (hamburger, sushi, fries, saimin, hotdog);
food : food_type;
package food_io is new enumeration_io(food_type); use food_io;
. . .
for food in hamburger..fries loop
        put_line(food);
end loop;
```

```
with text_io; use text_io;
procedure ascii is
        c:integer;
begin -- ascii

        put_line("This is print some of the ASCII characters from");
        put_line("   the space character  to the ~");
        new_line;
        c:=0;
        for i in ' '..'~' loop
                put(i);
                put("  ");
                if c<=10 then
                        c:=c+1;
                else
                        c:=0;
                        new_line;
                end if;
        end loop;
        new_line;
end ascii ;
```

RUN OF PROGRAM ASCII.ADA

```
This is print some of the ASCII characters from
   the space character  to the ~

   !  "  #  $  %  &  '  (  )  *  +
,  -  .  /  0  1  2  3  4  5  6  7
8  9  :  ;  <  =  >  ?  @  A  B  C
D  E  F  G  H  I  J  K  L  M  N  O
P  Q  R  S  T  U  V  W  X  Y  Z  [
\  ]  ^  _  `  a  b  c  d  e  f  g
h  i  j  k  l  m  n  o  p  q  r  s
t  u  v  w  x  y  z  {  |  }  ~
```

```
with text_io;use text_io;
procedure truth_table is
   ------------------------------------------------------------
   -- Example of using boolean variables; I/O boolean variables
   -- and of nested loops
   ------------------------------------------------------------
      -- declaration of variables
   p,q:boolean;
      -- generic instantiation of generic package enumeration_io
   package boolio is new enumeration_io(boolean);
   use boolio;
   begin --truth_table
      put("  p     q     p or q    ");
      put("p xor q    p and q") ;
      new_line;
      put("_____________________________________________________");
      new_line(2);
      for p in boolean loop
         for q in boolean loop
        put(p,6);
        put(q,9);
        put(p or q,9);
        put(p xor q,width=>9);
        put(p and q,width=>9);
        new_line;
         end loop;
      end loop;
    end truth_table;
```

```
------------------------------------------------------------
      EXECUTION RUN OF TRUTH_TABLE.ADA
------------------------------------------------------------

  p     q     p or q    p xor q    p and q
 ____________________________________________

FALSE FALSE    FALSE     FALSE     FALSE
FALSE TRUE     TRUE      TRUE      FALSE
TRUE  FALSE    TRUE      TRUE      FALSE
TRUE  TRUE     TRUE      FALSE     TRUE
```

CS050
Handout Set #6

Handout #6
User Defined Exceptions

A user defined exception is an identifier that is declared as an exception type. It is an identifier that is used in exception handling. It is an error condition explicitly raised:

Exception Handler

- is used to handle unanticipated runtime errors
- determines the problem and takes appropriate action
- is embedded in Ada bodies
- is placed at the end of a subprogram, block or package body
- is similar to the case statement

Syntax:

```
declare
      declarations ;
begin
      . . .
exception
      when  choices  = >  statement(s) ;
      . . .
      when  choices  = >  statement(s) ;
end;
```

Example:

```
declare
      Number : integer range 0..100;
begin
      put("Enter value from 0 to 100");
      get(Number); new_line;
exception
      when constraint_error = >
            put_line("Only values 0 to 100");
end;
```

Other Location of Exception Handlers

```
procedure Main is
      declaration(s) ;
begin -- Main
      statement(s) ;
exception
      exception handler ;
end Main;
```

```
package PkgName is
      visible declaration(s) ;
end PkgName ;

package body PkgName is
      hidden declaration(s) ;
begin
      -- code
exception
      exception handler ;
end PkgName ;
```

Built-In Exceptions

- Constraint_Error
- Numeric_Error
- Storage_Error
- Program_Error
- Data_Error
- Task_Error

```
------------------------------------------------------------------------
--    EXAMPLE OF EXCEPTION HANDLING
--    Note the declaration of user defined exceptions too_old,
--    and too_young.  The user may define his own exceptions or
--    use predefined ones such as constraint_error, program_error,
--    storage_error, numeric_error, data_error. (Check reference
--    manuals to determine how error is treated your implementation.
------------------------------------------------------------------------
with text_io; use text_io;
procedure excep is
         -- declaration of a subtype called agetype
   subtype agetype is integer range 1..120;
         -- declaration of variables
   age                 : agetype;
   too_young,too_old : exception;
         -- instantiation of integer_io
   package iio is new integer_io(agetype);use iio;

begin -- excep
   loop

      begin  -- local block
         put("Enter your age => ");
         get(age); put(age); new_line;
             -- raising user defined exceptions
         if age<12 then
            raise too_young;
         elsif age >80 then
            raise too_old;
         end if;

         put_line("Just the right age...");
         put_line("Just the person I need...");
         exit;     -- exit loop

         exception
            when constraint_error =>
               put_line("This is out of range!");
            when data_error =>
               put_line("This is the wrong data type ");
            when too_young =>
               put_line("You are too young to smoke!");
            when too_old =>
               put_line("You are too old for sky diving!");
            when others =>
               put_line("I give up!");
      end;  -- local block

   end loop;
   put_line("This is the end!");
   new_line;
end excep;
------------------------------------------------------------------------
--  EXECUTION RUN OF EXCEP.ADA
------------------------------------------------------------------------
Enter your age => 0 This is the wrong data type
Enter your age => 121 This is the wrong data type
Enter your age => 95
You are too old for sky diving!
Enter your age => 5
```

```
You are too young to smoke!
Enter your age =>  28
Just the right age...
Just the person I need...
This is the end!
```

CS050
Handout Set #7

Handout #7

An **array** is a composite object consisting of components of the same subtype. Its composite value consists of the values of its components. To reference a component of a one-dimensional array object, one uses the name of the array object appended with an index value enclosed in parentheses. The index is analogous to subscript. Any discrete type can be used as an index or subscript.

Syntax :

type *ArrayType* is array(*beginrange* .. *endrange* of *datatype*;

where *beginrange* and *endrange* are discrete types.

or

ArrayName : array(*beginrange* .. *endrange* of *datatype* ;

Examples:

```
-- An one dimensional array of 5 elements
-- Vector(1), Vector(2), Vector(3), Vector(4), and Vector(5)
type VectorType is array(1..5) of integer;
Vector : VectorType;
     or
Vector : array(1..5) of integer;

-- 3000 elements of strings whose length is 25 characters and
-- Part(1000), Part(1001), Part(1002) .. Part(3999)
type Part_Id is range 1000..3999;
type Part_Type is array(Part_Id) of string(1..25);
Part : Part_type;

Two dimensional array:
--  Matrix contains 3 rows and 5 columns of data elements
-- Matrix(1,1) Matrix(1,2) Matrix(1,3) Matrix(2,1) Matrix(2,2) ..
type MatrixType array(1..3,1..5) of float;
Matrix : MatrixType ;
```

~W CS520

```
--------------------------------------------------------------------------
--              FILE : MAIN.ADA
--     This main procedure reads from a sequential file called "STUDENT.DAT".
--     The data is read into several arrays (NAME,STATUS,JOB, and WEIGHT).
--     After closing the file, the average weight is calculated and the
--     maximum weight is found.  The appropriate student is identified also.
--------------------------------------------------------------------------

with text_io; use text_io;
procedure main is
         -- variable declaration
   max : positive := 15;
         -- type declaration
   type jobtype is (Navy_Captain,Actor,Hotel_Executive,Pilot,
            Lawyer,Opera_Singer,Polo_Player);
   type statustype is (freshmen,sophomore,junior,senior);
   type jobary is array(1..max) of jobtype;
   type statusary is array(1..max) of statustype;
   type weightary is array(1..max) of float;
   type nameary is array(1..max) of string(1..8);
         -- instantiation of generic packages for I/O
   package jobio is new enumeration_io(jobtype);
   package statusio is new enumeration_io(statustype);
   package fltio is new float_io(float);
   use jobio,statusio,fltio;
         -- declaration of variables
   status               : statusary;
   job                  : jobary;
   weight               : weightary;
   name                 : nameary;
   ave,maximum, total   : float;
   i,no_rec,rec         : positive;
   joe                  : file_type;

begin -- main
   i:=1;
   put_line("   People from file:");
   put_line("-------------------------------------");
   open(joe,in_file,"student.dat");
   while not end_of_file(joe) loop
         get(joe,name(i));
         get(joe,weight(i));
         get(joe,job(i));
         get(joe,status(i));
         skip_line(joe);
         put(name(i)); put("   ");
         put(status(i),12);
         put(job(i),17);
         put(weight(i),5,0,0);
         new_line;
         no_rec:=i;
         i:=i+1;
    end loop;
    close(joe);
      -- Calculate the Average weight
      total := 0.0;
      for i in 1..no_rec loop
            total := total + weight(i);
      end loop;
      ave := total/float(no_rec);
```

```
      -- Print the average weight
      put("---  The average weight is ");
      put(ave,7,2,0);new_line(2);
      -- Find the Maximum from list
      maximum := weight(1);
      rec:=1;
      for i in 2..no_rec loop
            if weight(i) > maximum then
                  maximum := weight(i);
                  rec := i;
            end if;
      end loop;
      -- Print our a list of names and weights
      for i in 1..no_rec loop
            put(name(i));
            put(weight(i),7,0,0);new_line;
      end loop; new_line;
      -- Print out the maximum weight & name
      put("... The maximum weight is ");
      put(maximum,7,0,0); new_line;
      put("... It belongs to ");
      put(name(rec)); new_line;
end main;
```

```
--------------------------------------------------------------------------
--       DATA FILE : STUDENT.DAT
--------------------------------------------------------------------------
Louis___152.0 Actor Senior
Lucinda_162.0 Pilot Sophomore
Alfred__125.0 Opera_Singer Freshmen
Forester217.0 Polo_Player Freshmen
Mary____172.0 Lawyer Junior
Tyron___115.0 Hotel_Executive Junior
Todd____105.0 Navy_Captain Sophomore
--------------------------------------------------------------------------
--       EXECUTION OF MAIN.ADA
--------------------------------------------------------------------------
   People from file:
-----------------------------------


Louis___     SENIOR       ACTOR              152.0
Lucinda_     SOPHOMORE    PILOT              162.0
Alfred__     FRESHMEN     OPERA_SINGER       125.0
Forester     FRESHMEN     POLO_PLAYER        217.0
Mary____     JUNIOR       LAWYER             172.0
Tyron___     JUNIOR       HOTEL_EXECUTIVE    115.0
Todd____     SOPHOMORE    NAVY_CAPTAIN       105.0
---  The average weight is     149.71

Louis___      152.0
Lucinda_      162.0
Alfred__      125.0
Forester      217.0
Mary____      172.0
Tyron___      115.0
Todd____      105.0

... The maximum weight is     217.0
... It belongs to Forester
```

```
-------------------------------------------------------------------
        DATA FILE : IN.DAT
-------------------------------------------------------------------
101Ross  actor         99.9 34.6
102Van   opera_singer 65.6 89.43
103Lynn  lawyer        56.1 89.43
110Susan stunt_man     56.45 100.0
-------------------------------------------------------------
--               FILE : MAIN3.ADA
--    This is an example of a main procedure that invokes
--    several procedures and functions.  It also uses array
--    types, subtypes and enumerated types, reading data
--    from a data file.       S. Honda     10/9/93
-------------------------------------------------------------
with text_io; use text_io;
procedure main3 is
        -- declaration of enumeration type status_type
       type stat_type is (actor,lawyer,opera_singer,stunt_man);
        -- generic instantiations for I/O
       package stat_io is new enumeration_io(stat_type);
       package n_io is new integer_io(natural);
       package f_io is new float_io(float);
       use stat_io, n_io, f_io;
        -- declaration of constant max
       max : constant natural := 300;
        -- declaration of array types
       type id_ary is array(1..max) of natural;
       type name_ary is array(1..max) of string(1..5);
       type status_ary is array(1..max) of stat_type;
       type grade_ary is array(1..max) of float;
        -- variable declarations
       name       :  name_ary;
       id         :  id_ary;
       status     :  status_ary;
       art,math   :  grade_ary;
       n          :  natural;
       ------------------------------------------------------------
       procedure getdata(id: out id_ary;name: out name_ary; status :
          out status_ary; art,math : out grade_ary; n:out natural) is
             sam : file_type;
             i   : natural := 0;
         begin -- getdata
             open(sam,in_file,"in.dat");
             while not end_of_file(sam) loop
                 i:=i+1;
                 get(sam,id(i));
                 get(sam,name(i));
                 get(sam,status(i));
                 get(sam,art(i));
                 get(sam,math(i));
             end loop;
             n:=i;
             close(sam);
             put_line("closing file....");
         end getdata;
       ------------------------------------------------------------
       procedure printlist(name:in name_ary;n:in natural;
             grade:in grade_ary;status:in status_ary;id:in id_ary) is
         begin --printlist
             put_line("     GRADE LISTING ");
```

```
            put_line("ID  Name  Grade  Status");
            put_line("-------------------------");
            for j in 1..n loop
                put(id(j),width=>3);put(" ");put(name(j));
                put(grade(j),6,1,0);put("  ");
                put(status(j),width=>8);new_line;
            end loop;
            new_line(2);
      end printlist;
      ------------------------------------------------------------
      function calc_avg(grade:in grade_ary;n:in natural) return float is
            temp : float := 0.0;
         begin -- calc_avg
            for i in 1..n loop
                temp := temp + grade(i);
            end loop;
            temp := temp/float(n);
            return temp;
          end calc_avg;
      ------------------------------------------------------------
      begin -- main3
            getdata(id,name,status,art,math,n);   -- invoke getdata
            put_line("      Art Grades");
            printlist(name,n,art,status,id);      -- invoke printlist
            put_line("      Math Grades");
            printlist(name,n,math,status,id);     -- invoke printlist
            put("Average art score is ");
            put(calc_avg(art,n),5,2,0);new_line;  -- invoke calc_avg
            put("Average math score is ");
            put(calc_avg(math,n),5,2,0);new_line;  -- invoke calc_avg
      end main3;
-----------------------------------------------------------------------
      EXECUTION RUN OF MAIN3.ADA
-----------------------------------------------------------------------
closing file....
      Art Grades
     GRADE LISTING
ID  Name  Grade  Status
-------------------------
101 Ross      99.9  ACTOR
102 Van       65.6  OPERA_SINGER
103 Lynn      56.1  LAWYER
110 Susan     56.5  STUNT_MAN


      Math Grades
     GRADE LISTING
ID  Name  Grade  Status
-------------------------
101 Ross      34.6  ACTOR
102 Van       89.4  OPERA_SINGER
103 Lynn      89.4  LAWYER
110 Susan    100.0  STUNT_MAN


Average art score is    69.51
Average math score is    78.37
```

```
------------------------------------------------------------------------
--          FILE : MAIN4.ADA
------------------------------------------------------------------------
with text_io; use text_io;
procedure main4 is
      type number_ary is array(1..100) of integer;
      numbers : number_ary;
      choice  : character;
      no_rec  : positive;
      package iio is new integer_io(integer);use iio;
      ------------------------------------------------------------------
      procedure menu(choice:in out character) is
            begin -- menu
                  new_line;
                  put     ("       (I)  Input values    ");
                  put_line("       (S)  Sort values");
                  put     ("       (M)  Find the Minimum");
                  put_line("       (L)  List Values");
                  put_line("             (Q)  Quit this program");
                  new_line;
                  put("Which of the following do you wish to do?");
                  get(choice);put(choice);new_line;
             end menu;
      ------------------------------------------------------------------
      procedure Input(numbers: in out number_ary; no_rec: out positive) is
            i:positive:=1;
            begin --Input
                  put("Enter integers => ");
                  get(numbers(i));new_line;
                  while numbers(i) /= -999 loop
                        i:= i+1;
                        get(numbers(i));
                  end loop;
                  no_rec:=i-1;
             end Input;
      ------------------------------------------------------------------
      procedure sort(numbers: in out number_ary; no_rec: in positive) is
            begin -- sort
                  put_line("...in development");
             end sort;
      ------------------------------------------------------------------
      function minimum(numbers: in number_ary; no_rec: in positive)
                                          return positive is
             -- local declaration
             min : positive:=1;
             begin --minimum
                  put_line("... in development");
                  return min;
             end minimum;
      ------------------------------------------------------------------
      procedure PrintList(numbers:in number_ary; no_rec: in positive) is
             begin -- PrintList
                  put_line(" List of values");
                  put_line("---------------");
                  for i in 1..no_rec loop
                        put(numbers(i));new_line;
                  end loop;
             end PrintList;
      ------------------------------------------------------------------
      begin -- Main4
```

```
            loop
                 Menu(choice);              -- Invoke Subprogram MENU
                 case choice is
                      when 'I' | 'i' => Input(Numbers,no_rec);
                      when 'S' | 's' => Sort(Numbers,no_rec);
                      when 'M' | 'm' => put(minimum(Numbers,no_rec));
                      when 'L' | 'l' => PrintList(Numbers,no_rec);
                      when 'Q' | 'q' => put_line(".... Goodbye!");
                                        exit;    -- exit loop
                      when others    => put("... wrong choice dummy!");
                                        new_line;
                  end case;
             end loop;
        end main4;
-------------------------------------------------------------------------
--       EXECUTION RUN OF MAIN4.ADA
-------------------------------------------------------------------------

       (I)  Input values           (S)  Sort values
       (M)  Find the Minimum       (L)  List Values
             (Q)  Quit this program

Which of the following do you wish to do? i
Enter integers =>

       (I)  Input values           (S)  Sort values
       (M)  Find the Minimum       (L)  List Values
             (Q)  Quit this program

Which of the following do you wish to do? s
...in development

       (I)  Input values           (S)  Sort values
       (M)  Find the Minimum       (L)  List Values
             (Q)  Quit this program

Which of the following do you wish to do? l
 List of values
---------------
    34
    -4
     0
   100

       (I)  Input values           (S)  Sort values
       (M)  Find the Minimum       (L)  List Values
             (Q)  Quit this program

Which of the following do you wish to do? n
... wrong choice dummy!

       (I)  Input values           (S)  Sort values
       (M)  Find the Minimum       (L)  List Values
             (Q)  Quit this program

Which of the following do you wish to do? q
.... Goodbye!
```

CS050
Handout Set #8

Handout #8

Functions and procedures may be written as separate units, compiled, and used again and again by other subprograms. The following are examples of external subprograms that are called by another programs. Note context clause **with** that makes packages or subprograms visible. **Use** can only be used by packages.

Note two-dimensional array airline flight problem and is associated Ada program that uses logical two-dimensional variables.

```
------------------------------------------------------------------------
--                  FILE : MAIN.ADA
--   This is an example of an external subprogram that is called by
--   another program called MAIN.
------------------------------------------------------------------------
procedure swap(a,b:in out integer) is
            -- declaration of local variables
         temp : integer;
   begin -- swap
         temp := a;          -- assigns a to temp
         a := b;             -- assigns b tc a
         b := temp;          -- assigns temp to b
   end swap;

------------------------------------------------------------------------
--                  FILE : SWAP.ADA
--   This is an example of using an external subprogram procedure
--   called SWAP that exits on secondary storage.
------------------------------------------------------------------------
with swap,text_io; -- make procedure swap and package text_io available
use text_io;       -- Note you can only USE packages!
procedure main is
      x,y : integer;
      package iio is new integer_io(integer);
      use iio;
      begin -- main
                  -- prompt user for 2 integers
            put_line("Enter an integer for x ==> ");
            get(x);new_line;
            put_line("Enter another integer for y==> ");
            get(y);new_line;
                  -- invoke procedure swap
            swap(x,y);
            put("Value of x is ==> ");put(x);new_line;
            put("Value of y is ==> ");put(y);new_line;
      end main;
------------------------------------------------------------------------
         COMPILATION OF MAIN AND SWAP, AND EXECUTION RUN
------------------------------------------------------------------------
sacredheart.shu>ada swap.ada main.ada
sacredheart.shu> ald -o main.exe main
sacredheart.shu> main.exe

Enter an integer for x ==> -105
Enter another integer for y ==> 200
Value of x is ==>    200
Value of y is ==>   -105
```

```
with text_io; use text_io;
procedure sqrt is
        x:float;
        package fltio is new float_io(float);
        use fltio;
        ------------------------------------------------------------
        function sqr(x:in float) return float is
                root:float:=x/2.0;
        begin -- sqr
                while abs(x-root*root) > 2.0 * x * 0.000007 loop
                        root:=(root + x/root)/2.0;
                end loop;
                return root;
        end sqr;
        ------------------------------------------------------------
        begin  -- sqrt
                put("enter number to find the sqrt of ");
                get(x);put(x,5,6,0);new_line;
                put("The Square Root is ");
                put(sqr(x),5,6,0);new_line;
        end sqrt;
------------------------------------------------------------------
                EXECUTION RUN OF A.OUT
------------------------------------------------------------------
SHU> a.out
enter number to find the sqrt of     4.0
     The Square Root is    2.000000

SHU> a.out
enter number to find the sqrt of    26.459999
     The Square Root is    5.143949

SHU> a.out
enter number to find the sqrt of   145.678894
     The Square Root is    12.069752

SHU> a.out
enter number to find the sqrt of   1044.0
     The Square Root is    12.000000
```

```
-------------------------------------------------------------
--      This is an external function called FILE : SQR.ADA
--      It was created as a separate file and compiled.
-------------------------------------------------------------
        function sqr(x :in float) return float is
                root:float :=x/2.0;
        begin -- sqr
                while abs(x-root*root) > 2.0 * x * 0.000007 loop
                        root := (root + x/root)/2.0;
                end loop;
                return root;
        end sqr;
-------------------------------------------------------------
        COMPILATION OF EXTERNAL FUNCTION
-------------------------------------------------------------
shu.sacredheart.edu> ada sqr.ada


-------------------------------------------------------------
--      This is a main program that makes the function SQR.ADA
--      available with the context clause WITH.  (Note you can
--      only USE packages.)
-------------------------------------------------------------
with text_io,sqr;use text_io;
procedure mainsqr is
        x: float;
        package fltio is new float_io(float);
        use fltio;
        begin -- mainsqr
                put("enter number to find the sqrt of ");
                get(x);put(x,5,6,0);new_line;
                put("The square root is ");
                put(sqr(x),5,2,0);new_line;
        end mainsqr;

-------------------------------------------------------------
        COMPILATION AND LINKING OF MAIN PROGRAM.
-------------------------------------------------------------
shu.sacredheart.edu> mainsqr.ada                -- compiles mainsqr
shu.sacredheart.edu> ald -o mainsqr.exe mainsqr -- links mainsqr
-------------------------------------------------------------
        EXECUTION RUNS OF PROGRAM MAIINSQR.ADA
-------------------------------------------------------------
shu.sacredheart.edu>mainsqr.exe
enter number to find the sqrt of    12.540000
The square root is     3.54


shu.sacredheart.edu>mainsqr.exe
enter number to find the sqrt of     4.000000
The square root is     2.00
```

10 An airline flies between six cities. Whether or not there is a direct flight from one city to another is indicated in the following table:

		To					
		1	2	3	4	5	6
From	1	F	T	T	F	F	T
	2	T	F	T	F	F	T
	3	F	F	F	T	F	F
	4	F	T	F	F	T	F
	5	F	T	F	T	F	F
	6	F	T	F	F	T	F

On the left and across the top are the numbers of the cities. If there is a T at the intersection of a row and column, there is a direct flight from the city marked on the left to the city indicated at the top. An F indicates that there is no direct flight between the two cities.

The information for this table is recorded in the first six records of a file. Recorded in each record is the data for one row of the table. Following the table data is one record for each customer with the customer's number and his or her request for a flight pattern. The flight pattern indicates the cities between which the customer wishes to fly. For example, a pattern of 13426 indicates that the customer wishes to fly from city 1 to city 3, then from city 3 to city 4, then to city 2, and finally to city 6. The maximum number of cities in a flight pattern is five. If the customer has fewer than five cities in his or her pattern, the remaining numbers are zero. Thus a pattern of 62000 indicates that the customer wishes to fly from city 6 to city 2 and does not wish to continue beyond that.

Write an Ada program to read the data for the flight table. Print the table with appropriate headings. Then determine if each customer's requested flight pattern is possible. Print the customer's number, his or her requested flight pattern, and a statement of whether or not a ticket may be issued for th desired pattern.

To test the program use the data in the previous flight table and the following customer data:

Customer number	*Flight pattern*
10123	13426
11305	62000
13427	42?20
18211	52500
19006	34212
20831	65426
21475	32000
22138	43621
24105	13424
24216	65231
26009	34250

```
------------------------------------------------------------
--       DATA FILE CUST.DAT
------------------------------------------------------------
10123 1 3 4 2 3
13427 4 2 3 2 0
11305 5 2 0 0 0
18211 5 2 0 0 0
12342 6 2 0 0 0
------------------------------------------------------------
--       DATA FILE TABLE.DAT
------------------------------------------------------------
FALSE TRUE TRUE FALSE FALSE TRUE
TRUE FALSE TRUE FALSE FALSE TRUE
FALSE FALSE FALSE TRUE FALSE FALSE
FALSE TRUE FALSE FALSE TRUE FALSE
FALSE TRUE FALSE TRUE FALSE FALSE
FALSE TRUE FALSE FALSE TRUE FALSE

------------------------------------------------------------------
--    This Main procedure called ARRAYS.ADA uses 2 two-dimensional
--    arrays.  It reads data from two sequential files, a CUST.DAT
--    file and a TABLE.DAT file to determine if a ticket should be
--    issued, depending on whether or not the flight pattern is
--    available.                          S. Honda 11/92
------------------------------------------------------------------
with text_io; use text_io;
procedure arrays is
        -- declaration of types
      type arytype is array(1..6,1..6) of boolean;
      type custary is array(1..50) of natural;
      type fltpat is array(1..50,1..5) of natural;
         -- instantiation of generic packages
      package iio is new integer_io(natural);
      package boolio is new enumeration_io(boolean);
      use iio, boolio;
         -- declaration of variables
      cust      : custary;
      table     : arytype;
      flt       : fltpat;
      no_of_rec : positive;
     ------------------------------------------------------------------------
     --  This procedure reads data from 2 files
           procedure read(cust : out custary; flt : out fltpat;
                          table : out arytype; no_of_rec : out positive) is
              -- local declarations
              joe : file_type;
              i,j : positive:=1;
           begin -- read
              open(joe,in_file,"Cust.dat");
              while not end_of_file(joe) loop              -- get data from
                    get(joe,cust(i));                      -- file "CUST.DAT"
                    for j in 1..5 loop                     -- store cust id in
                        get(joe,flt(i,j));                 -- array CUST(i) and
                    end loop;                              -- flight pattern in
                    i:=i+1;                                -- array FLT(i,j)
              end loop;
              no_of_rec:=i-1;
              close(joe);
              open(joe,in_file,"table.dat");               -- get data from
              for r in 1..6 loop                           -- file "TABLE.DAT"
```

```
            for c in 1..6 loop                          -- and store data  i
                  get(joe,table(r,c));                  -- array TABLE(r,c)
            end loop;
      end loop;
      close(joe);
end read;
-------------------------------------------------------------------
--  This procedure prints out data read from file
procedure print(cust: in custary; flt:in fltpat;
                table:in arytype; no_of_rec:in positive) is
            begin -- print
                  new_line;
                  put_line("Customer       Flight");
                  put_line(" Number        Pattern");
                  put_line("_____________________");
                  for i in 1..no_of_rec loop
                        put(cust(i));put("          ");
                        for j in 1..5 loop
                              put(flt(i,j),width=>1);
                        end loop;
                        new_line;
                  end loop;
                  new_line(2);
                  put_line("          Flight pattern between 6 cities");
                  put_line("                              TO");
                  put("         ");
                  for i in 1..6 loop
                       put(i);
                  end loop;
                  new_line;  put("      ");
                  put_line("-----------------------------------------");
                  for i in 1.. 6 loop
                        if i=4 then
                              put("FROM ");
                        else
                              put("     ");
                        end if;
                        put(i);put("  ");
                        for j in 1..6 loop
                              put(table(i,j));put(' ');
                        end loop;
                        new_line;
                  end loop;
            end print;
-------------------------------------------------------------------
-- This procedure prints out customer requests and ticket message
procedure request(cust: in custary; flt: in fltpat;
                  table:in arytype; no_of_rec: in positive) is
            possible:boolean;
            k:integer;
            begin --  request
                  new_line(2);
                  put("Cust ID   Flight Pattern");
                  put_line("  Message");
                  put_line("-----------------------------------------"
                  for i in 1..no_of_rec loop
                        put(cust(i));put("  ");
                        possible:=true;
                        for j in 1..5 loop
                              put(flt(i,j),3);
```

```
                                    end loop;
                                    put("  ");
                                    k:=1;        -- determine if flight is possibl
                                    while (k<5) loop
                                       if flt(i,k+1)/=0 then
                                          if not table(flt(i,k),flt(i,k+1)) then
                                             possible:=false;
                                          end if;
                                       end if;
                                       k:=k+1;
                                    end loop;
                                    if possible  then
                                          put_line(" issue ticket");
                                    else
                                          put_line(" ** no ticket **");
                                    end if;
                              end loop;
                        end request;
            ------------------------------------------------------------
      begin -- arrays
            read(cust,flt,table,no_of_rec);
            print(cust,flt,table,no_of_rec);
            request(cust,flt,table,no_of_rec);
      end arrays;
-------------------------------------------------------------
--       RUN OF ARRAYS.ADA
-------------------------------------------------------------

Customer        Flight
 Number         Pattern

_____________________
 10123           13423
 13427           42320
 11305           52000
 18211           52000
 12342           62000


         Flight_pattern between 6 cities
                  —          TO
                  1     2     3     4     5     6
       ---------------------------------------
           1  FALSE TRUE TRUE FALSE FALSE TRUE
           2  TRUE FALSE TRUE FALSE FALSE TRUE
           3  FALSE FALSE FALSE TRUE FALSE FALSE
FROM       4  FALSE TRUE FALSE FALSE TRUE FALSE
           5  FALSE TRUE FALSE TRUE FALSE FALSE
           6  FALSE TRUE FALSE FALSE TRUE FALSE


Cust ID   Flight Pattern   Message
-----------------------------------------------
 10123     1  3  4  2  3   issue ticket
 13427     4  2  3  2  0   ** no ticket **
 11305     5  2  0  0  0   issue ticket
 18211     5  2  0  0  0   issue ticket
 12342     6  2  0  0  0   issue ticket
```

Appendix C

CS051
Handouts

Notes for Teaching CS051

For all data structures,

1. Introduce concept of data structure, for example, ***Stacks.*** Give examples how stacks are implemented
   - cafeteria trays
   - pile of books

2. Do drawings on board

3. Show development of pseudo code for activities performed with datatype. For example for ***Stacks,***
   - clearstack
   - emptystack
   - fullstack
   - push
   - pop

4. Explain ATD of this particular data type.
   - package code
   - generic package code

5. Software Engineering Principles
   - Information Hiding
   - Reuse
   - Abstraction
   - Modularity
   - Localization
   - Uniformity
   - Completeness
   - Confirmability

6. Ada's language features that aid in the implementation of software engineering principles.
   - package specification & body
   - subprograms
   - limited & private types
   - generics

CS051
Handout #1

Refer to old program examples and notes covering arrays from CS050

```
--ADA PROGRAM TO INPUT AN ARRAY OF 5 VALUES AND SORT THEM.
--                  D. PINTO ....SPRING 93 CS 051
------------------------------------------------------------------
with text_io; use text_io;
procedure bubsort is
        package iio is new integer_io(integer);
        use iio;
        type arytype is array(1..5) of integer;
        list,slist : arytype;
        no_rec     : integer;
------------------------------------------------------------------
procedure getdata(no : out integer; alist : out arytype) is
        n:integer;
        begin -- getdata
                put_line("ENTER 5 INTEGERS");
                for i in 1..5 loop
                        get(n);
                        alist(i) := n;
                end loop;
                no := 5;
end getdata;
------------------------------------------------------------------
procedure sort(alist : in out arytype;no : in integer) is
        temp : integer;
        done : boolean;
        begin -- sort
        done := false;
             while not done loop
                 done :=true;
                 for j in 1 ..no - 1 loop
                        if alist(j) > alist(j+1) then
                           temp    :=  alist(j+1);
                           alist(j+1) := alist(j);
                           alist(j) := temp;
                           done := false;
                        end if;
                 end loop;
              end loop;
end sort;
------------------------------------------------------------------
begin--bubsort
        getdata(no_rec,list);
        put_line("    unsorted list");
        for j in 1..no_rec loop
             put(list(j),5);
             new_line;
        end loop;
        slist := list;
        sort(slist,no_rec);
        put_line("     sorted list");
        for j in 1..no_rec loop
            put(slist(j),5);
            new_line;
        end loop;
        new_line;
end bubsort;
```

```
program bubsort(input,output);
type arytype = array[1..5] of integer;
var a,b : arytype;
        i , j  : integer;

(*************************************************************************)

procedure getdata(var  p : integer; var ary : arytype);
var n : integer;
begin
        writeln(' enter 5 integers');
        for i := 1 to 5 do
        begin
             readln(n);
             ary[i] := n;
        end;
        p := 5;
end;
(*************************************************************************)

procedure sort(var aa: arytype; var i : integer);
var j,temp : integer;
    done : boolean;
begin
        done := false;
        while not done do
          begin
            done := true;
             for j:= 1 to i-1 do
              begin
                if aa[j] > aa[j+1] then
                   begin
                        temp := aa[j+1];
                        aa[j+1] := aa[j];
                        aa[j] := temp;
                        done := false;
                   end;
              end;
          end;
end;

(*************************************************************************)

begin
        getdata(i,a);
        for j := 1 to i do
            write(a[j] :5);
        writeln;
        for j := 1 to i do
        b[j] := a[j];
        sort(b , i);
        for j := 1 to i do
            write(b[j] :5);
end.
```

CS051
Handout #2

Stack Exercises

Show what is written by the following segments of code, given that **Stack** is a stack of integer elements and X. Y, and Z are integer variables.

1.
```
ClearStack(Stack) ;
        X := 1;
        Y := 0
        Z := 4;
        Push(Stack, Y);
        Push(Stack, X);
        Push(Stack, X + Z);
        Pop(Stack, Y);
        Push(Stack, SQR(Z))
        Push(Stack, Y);
        Push(Stack, 3);
        Pop(Stack, X);
        Writeln('X = ', X);
        Writeln('Y = ', Y);
        Writeln('Z = ', Z);
        WHILE NOT EmptyStack(Stack) DO
                BEGIN
                        Pop(Stack, X);
                        Writeln(X)
                END
```

2.
```
ClearStack(Stack);
X := 4;
Y := 0;
Z := X + 1;
Push(Stack, Y);
Push(Stack, Y+1);
Push(Stack,X);
Pop(Stack, Y);
X := Y + 1;
Push(Stack, X);
Push(Stack, Z);
WHILE NOT EmptyStack(Stack) DO
        BEGIN
                Pop(Stack, Z);
                Writeln(Z);
        END;
Writeln('X = ',X);
Writeln('Y = ',Y);
Writeln('Z = ',Z)
```

```
with text_io; use text_io;
procedure stackmain is
      maxstack : constant integer := 100;
      subtype elementtype is character range ' '..'z';
      type arytype is array(1..maxstack) of elementtype;
      type stacktype is record
            elements : arytype;
            top      : integer range 0..maxstack;
      end record;
      ch : character; s : stacktype;
      ------------------------------------------------------------------
      procedure clearstack(stack : out stacktype) is
           begin  -- clearstack
               stack.top := 0;
           end clearstack;
      ------------------------------------------------------------------
      function fullstack(stack :in stacktype) return boolean is
           begin  -- fullstack
               if stack.top = maxstack then
                     return true;
               else
                     return false;
              end if;
           end fullstack;
      ------------------------------------------------------------------
      function emptystack(stack:in stacktype) return boolean is
           begin  -- emptystack
               if stack.top = 0 then
                    return true;
               else
                    return false;
               end if;
           end emptystack;
      ------------------------------------------------------------------
      procedure push(stack:in out stacktype; newelement:in elementtype) is
           begin  -- push
               stack.top := stack.top + 1;
               stack.elements(stack.top) := newelement;
           end push;
      ------------------------------------------------------------------
      procedure pop(stack:in out stacktype; poppedelement:in out
                    elementtype) is
           begin   -- pop
               poppedelement := stack.elements(stack.top);
               stack.top := stack.top - 1;
           end pop;
      ------------------------------------------------------------------
      begin -- stackmain
           clearstack(s);
           put_line("Enter Characters for stack.enter 'q' for quit");
           new_line;
           get(ch);put(ch);
           while (ch /= 'q') and not fullstack(s) loop
                  push(s,ch);
                  get(ch);put(ch);
           end loop;
           new_line;
           if fullstack(s) then
                  put_line("Stack full");
           end if;
```

```
            while not emptystack(s) loop
                  pop(s,ch);
                  put(ch);
            end loop;
            new_line;
end stackmain;
```

```
SHU> ada stackmain.ada          -- COMPILES SOURCE CODE
SHU> ald stackmain              -- LINKS OBJECT CODE
SHU> a.out                      -- EXECUTES EXECUTABLE CODE

Enter Characters for stack.enter 'q' for quit

adnoH ydnaSq
Sandy Honda
```

```
SHU> a.out                      -- EXECUTES EXECUTABLE CODE

Enter Characters for stack.enter 'q' for quit

:OW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THE WONDERFUL BEAUTIFUL
Stack full
RPU GNITICXE SUOIROLG LUFITUAEB LUFREDNOW EHT FO DIA EHT OT EMOC OT NEM DOOG LI
```

```
-------------------------------------------------------------------
-- FILE : STACKPKG.ADA  ( Package Specification )    1/93  S. Honda
-------------------------------------------------------------------
package stackpkg is
   maxstack : constant integer := 15;
   subtype elementtype is integer range integer'first..integer'last;
   type arytype is array(1..maxstack) of elementtype;
   subtype toptype is integer range 0..maxstack;
   type stacktype is record
         elements : arytype;
         top      : toptype;
    end record;
    procedure clearstack(stack:out stacktype);
    function fullstack(stack: in stacktype) return boolean;
    function emptystack(stack:in stacktype) return boolean;
    procedure push(stack:in out stacktype;newelement:in elementtype);
    procedure pop(stack:in out stacktype;
                        poppedelement:in out elementtype);
end stackpkg;
-------------------------------------------------------------------
--    FILE : STACKBPK.ADA     ( Package Body ) 1/93   S. Honda
-------------------------------------------------------------------
with text_io; use text_io;
package body stackpkg is
    ---------------------------------------------------------------
    procedure clearstack(stack:out stacktype) is
         begin -- clearstack
               stack.top := 0;
         end clearstack;
    ---------------------------------------------------------------
    function fullstack(stack: in stacktype) return boolean is
         begin -- fullstack
               if stack.top = maxstack then
                     return true;
               else
                     return false;
               end if;
         end fullstack;
    ---------------------------------------------------------------
    function emptystack(stack:in stacktype) return boolean is
         begin -- emptystack
               if stack.top = 0 then
                     return true;
               else
                     return false;
               end if;
          end emptystack;
    ---------------------------------------------------------------
    procedure push(stack:in out stacktype;newelement:in elementtype) is
         begin -- push
               stack.top := stack.top + 1;
               stack.elements(stack.top) := newelement;
         end push;
    ---------------------------------------------------------------
    procedure pop(stack:in out stacktype;
                        poppedelement:in out elementtype) is
         begin -- pop
               poppedelement := stack.elements(stack.top);
               stack.top := stack.top - 1;
         end pop;
```

```
-------------------------------------------------------------------
end stackpkg;
-------------------------------------------------------------------
--    FILE : STKMAIN.ADA   ( Project 2 )     1/93  S. Honda
-------------------------------------------------------------------
with stackpkg, text_io; use stackpkg, text_io;
procedure stkmain is
      package iio is new integer_io(integer);
      use iio;
      s    : stacktype;
      numb : integer;
      begin -- stkmain
            clearstack(s);
            for i in 1..100 loop
                  put("Enter integer(");put(i);put(") ");
                  put(" (0 to quit) => ");
                  get(numb);put(numb);new_line;
                  if numb = 0 then
                        exit;
                  elsif (numb > 0) and not fullstack(s) then
                        push(s,numb);
                        put("... ");put(numb);put_line(" pushed onto stack");
                  elsif (numb < 0) and not emptystack(s) then
                        pop(s,numb);
                        put("... ");put(numb);put_line(" popped from stack");
                  elsif (numb > 0) and fullstack(s) then
                        put("... Stack Full cannot push");
                        put(numb);put_line(" onto stack");
                  elsif (numb < 0) and emptystack(s) then
                        put("... Stack Empty cannot pop");
                        put_line(" number from stack");
                  end if;
            end loop;
            new_line;
            put_line("Values from stack are ");
            while not emptystack(s) loop
                  pop(s,numb);
                  put(numb);put_line(" popped ");
            end loop;
            put_line("     End of Job!");
       end stkmain;
-------------------------------------------------------------------
--       EXECUTION RUN OF STKMAIN.ADA
-------------------------------------------------------------------
Enter integer(     1)  (0 to quit) =>      1
...      1 pushed onto stack
Enter integer(     2)  (0 to quit) =>      2
...      2 pushed onto stack
Enter integer(     3)  (0 to quit) =>    -22
...      2 popped from stack
Enter integer(     4)  (0 to quit) =>    -11
...      1 popped from stack
Enter integer(     5)  (0 to quit) =>    -10
... Stack Empty cannot pop number from stack
Enter integer(     6)  (0 to quit) =>      1
...      1 pushed onto stack
Enter integer(     7)  (0 to quit) =>      2
...      2 pushed onto stack
Enter integer(     8)  (0 to quit) =>      3
...      3 pushed onto stack
```

```
Enter integer(     9)  (0 to quit) =>       4
...       4 pushed onto stack
Enter integer(    10)  (0 to quit) =>       5
...       5 pushed onto stack
Enter integer(    11)  (0 to quit) =>       6
...       6 pushed onto stack
Enter integer(    12)  (0 to quit) =>       7
...       7 pushed onto stack
Enter integer(    13)  (0 to quit) =>       8
...       8 pushed onto stack
Enter integer(    14)  (0 to quit) =>       9
...       9 pushed onto stack
Enter integer(    15)  (0 to quit) =>      10
...      10 pushed onto stack
Enter integer(    16)  (0 to quit) =>      11
...      11 pushed onto stack
Enter integer(    17)  (0 to quit) =>      12
...      12 pushed onto stack
Enter integer(    18)  (0 to quit) =>      13
...      13 pushed onto stack
Enter integer(    19)  (0 to quit) =>      14
...      14 pushed onto stack
Enter integer(    20)  (0 to quit) =>      15
...      15 pushed onto stack
Enter integer(    21)  (0 to quit) =>      16
... Stack Full cannot push    16 onto stack
Enter integer(    22)  (0 to quit) =>     -15
...      15 popped from stack
Enter integer(    23)  (0 to quit) =>     -14
...      14 popped from stack
Enter integer(    24)  (0 to quit) =>       0

Values from stack are
    13 popped
    12 popped
    11 popped
    10 popped
     9 popped
     8 popped
     7 popped
     6 popped
     5 popped
     4 popped
     3 popped
     2 popped
     1 popped
     End of Job!
```

CS051
Handout #3

```
------------------------------------------------------------------
--          FILE - QUEPKG.ADA
--     This is the package specification that will allow one to
--     implement a que with integers.
------------------------------------------------------------------
package quepkg is
      subtype maxqueuetype is positive range 1..1000;
      maxqueue : maxqueuetype := 25;
      subtype elementtype is integer range -30000..30000;
      type elementary is array(1..maxqueue) of elementtype;
      type queuetype is
            record
                  elements   : elementary;
                  front,rear : (0..maxqueue);
            end record;
      procedure clearqueue(queue : in out queuetype);
      function fullqueue(queue : in queuetype) return boolean;
      function emptyqueue(queue : in queuetype) return boolean;
      procedure insert(queue : in out queuetype;
                               newelement: in elementtype);
      procedure delete(queue : in out queuetype;
                               element : in out elementtype);
end quepkg;

------------------------------------------------------------------
--          FILE QUEPKGB.ADA
--     This is the package body for the package QUEPKG
------------------------------------------------------------------
package body quepkg is
      ------------------------------------------------------------
      procedure clearqueue(queue : in out queuetype) is
            begin -- clearqueue
                  queue.front := maxqueue;
                  queue.rear  := maxqueue;
            end clearqueue;
      ------------------------------------------------------------
      function fullqueue(queue : in queuetype) return boolean is
            querear : 1..maxqueue;
            begin -- fullqueue
                  if queue.rear = maxqueue then
                        querear := 1;
                  else
                        querear := queue.rear + 1;
                  end if;
                  if querear = queue.front then
                        return true;
                  else
                        return false;
                  end if;
            end fullqueue;
      ------------------------------------------------------------
      function emptyqueue(queue : in queuetype) return boolean is
            begin -- emptyqueue
                  if queue.rear = queue.front then
                        return true;
                  else
                        return false;
                  end if;
            end emptyqueue;
      ------------------------------------------------------------
```

```
      procedure delete(queue : in out queuetype;
                                 element: in out elementtype) is
            begin -- insert
                  if queue.front = maxqueue then
                        queue.front := 1;
                  else
                        queue.front := queue.front + 1;
                  end if;
                  element := queue.elements(queue.front);
            end delete;
      ------------------------------------------------------------
      procedure insert(queue : in out queuetype;
                                 newelement : in elementtype) is
            querear : elementtype;
            begin -- insert
                  if queue.rear = maxqueue then
                        queue.rear := 1;
                  else
                        queue.rear := queue.rear + 1;
                  end if;
                  queue.elements(queue.rear) := newelement;
             end insert;
end quepkg;

------------------------------------------------------------------
--          FILE : QUEMAIN.ADA
--     This main procedure will create a queue called q and allow
--     the user to place integer values onto the queue.
------------------------------------------------------------------
with quepkg,text_io; use quepkg,text_io;
procedure quemain is
            q    : queuetype;
            numb : integer;
            package iio is new integer_io(integer); use iio;
      begin -- quemain
            clearqueue(q);
            put("Enter data to put into the queue ");
            get(numb);put(numb);new_line;
            while (numb /= 0) and not fullqueue(q) loop
                  insert(q,numb);
                  get(numb);put(numb);new_line;
            end loop;
            if fullqueue(q) then
                  put_line("Full Queue! ");
            end if;
            put_line("Queue Elements are : ");
            while not emptyqueue(q) loop
                  delete(q,numb);
                  put(numb);new_line;
            end loop;
      end quemain;
------------------------------------------------------------------
--     EXECUTION RUN OF QUEMAIN
------------------------------------------------------------------
Enter data to put into the queue    100
   -50
     0
Queue Elements are :
   100
   -50
```

```
-------------------------------------------------------------------
--       FILE : QUEUE.ADA
--    This main procedure is an example of implementing a queue.
-------------------------------------------------------------------
with text_io; use text_io;
procedure queue is
      subtype maxqueuetype is positive range 1..1000;
      maxqueue : maxqueuetype := 25;
      subtype elementtype is integer range -30000..30000;
      type elementary is array(1..maxqueue) of elementtype;
      type queuetype is
            record
                  elements   : elementary;
                  front,rear : (0..maxqueue);
            end record;
            q    : queuetype;
            numb : integer;
            package iio is new integer_io(integer); use iio;
      -------------------------------------------------------------
      procedure clearqueue(queue : in out queuetype) is
            begin -- clearqueue
                  queue.front := maxqueue;
                  queue.rear  := maxqueue;
            end clearqueue;
      -------------------------------------------------------------
      function fullqueue(queue : in queuetype) return boolean is
            querear : 1..maxqueue;
            begin -- fullqueue
                  if queue.rear = maxqueue then
                        querear := 1;
                  else
                        querear := queue.rear + 1;
                  end if;
                  if querear = queue.front then
                        return true;
                  else
                        return false;
                  end if;
            end fullqueue;
      -------------------------------------------------------------
      function emptyqueue(queue : in queuetype) return boolean is
            begin -- emptyqueue
                  if queue.rear = queue.front then
                        return true;
                  else
                        return false;
                  end if;
            end emptyqueue;
      -------------------------------------------------------------
      procedure delete(queue : in out queuetype;
                                element: in out elementtype) is
            begin -- insert
                  if queue.front = maxqueue then
                        queue.front := 1;
                  else
                        queue.front := queue.front + 1;
                  end if;
                  element := queue.elements(queue.front);
            end delete;
      -------------------------------------------------------------
```

```
      procedure insert(queue : in out queuetype;
                                 newelement : in elementtype) is
            querear : elementtype;
            begin -- insert
                  if queue.rear = maxqueue then
                        queue.rear := 1;
                  else
                        queue.rear := queue.rear + 1;
                  end if;
                  queue.elements(queue.rear) := newelement;
             end insert;
      ------------------------------------------------------------
      begin -- queue
            clearqueue(q);
            put("Enter data to put into the queue ");
            get(numb);put(numb);new_line;
            while (numb /= 0) and not fullqueue(q) loop
                  insert(q,numb);
                  get(numb);put(numb);new_line;
            end loop;
            if fullqueue(q) then
                  put_line("Full Queue! ");
            end if;
            put_line("Queue Elements are : ");
            while not emptyqueue(q) loop
                  delete(q,numb);
                  put(numb);new_line;
            end loop;
      end queue;
-------------------------------------------------------------------
--    EXECUTION RUN OF QUEUE.ADA
-------------------------------------------------------------------
Enter data to put into the queue      5
   -34
   100
    10
     0
Queue Elements are :
     5
   -34
   100
    10
```

```
------------------------------------------------------------------
--    FILE - GPKG.ADA
--    This is the generic package specification that will
--    allow one to implement a queue with elementtype.
------------------------------------------------------------------
generic
       maxqueue : in positive;
       type elementtype is private;
       package gpkg is
          procedure clearqueue;
          function fullqueue return boolean;
          function emptyqueue return boolean;
          procedure insert (newelement: in elementtype);

          procedure delete (element : in out elementtype);
       end gpkg;
```

```
------------------------------------------------------------
--    FILE - GPKGB.ADA
--    This is the package body for the package   GPKG.
------------------------------------------------------------
package body gpkg is
          subtype maxqueuetype is integer range 1..maxqueue;
          type elementary is array(maxqueuetype) of elementtype;
          front,rear : maxqueuetype;
          queue : elementary;
       ------------------------------------------ -----------
        procedure clearqueue is
             begin -- clearqueue
                   front := maxqueue;
                   rear  := maxqueue;
             end clearqueue;
        ------------------------------------------------------
        function fullqueue return boolean is
             querear : maxqueuetype;
             begin -- fullqueue
                   if rear = maxqueue then
                         querear := 1;
                   else
                         querear := rear + 1;
                   end if;
                   if querear = front then
                         return true;
                   else
                         return false;
                   end if;
             end fullqueue;
        ------------------------------------------------------
        function emptyqueue return boolean is
             begin -- emptyqueue
                   if rear = front then
                         return true;
                   else
                         return false;
                   end if;
              end emptyqueue;
        ------------------------------------------------------
        procedure insert (newelement: in elementtype) is
              begin -- insert
                   if rear = maxqueue then
                         rear := 1;
                   else
                         rear := rear + 1;
                   end if;
                   queue(rear) := newelement;
              end insert;
        ------------------------------------------------------
        procedure delete (element : in out elementtype) is
              begin -- delete
                   if front = maxqueue then
                         front := 1;
                   else
                         front := front + 1;
                   end if;
                   element := queue(front);
              end delete;
end gpkg;
```

```
with text_io,gpkg; use text_io;
procedure main3 is
        -- type declarations
   type rectype is
        record
              itemf:character;
              namef:string(1..5);
        end record;
   type employeetype is
        record
              name:string(1..5);
              age :positive;
              smoker:boolean;
        end record;
        -- variable declarations
   size     : positive := 7;
   item     : rectype;
   employee : employeetype;
   noemp    : positive :=5;
        -- instantiation of generic packages
   package iio is new integer_io(integer);
   package empqueue is new gpkg(noemp,employeetype);
   package q1 is new gpkg(size,rectype);
   package q2 is new gpkg(size,rectype);
   package smokeio is new enumeration_io(boolean);
   use q1,q2,iio,empqueue,smokeio;

   begin --main3
      q1.clearqueue;q2.clearqueue;
      item.itemf := '*';
      item.namef := "*****";
        q1.insert(item);
      item.itemf := 'S';
      item.namef := "Sandy";
        q1.insert(item);
      item.itemf := 'S';
      item.namef := "Sally";
        q1.insert(item);
      item.itemf := 'H';
      item.namef := "Henry";
        q2.insert(item);
      item.itemf := 'T';
      item.namef := "Tomas";
        q2.insert(item);
      item.itemf := 'L';
      item.namef := "Larry";
        q2.insert(item);
        q1.delete(item);
        put("deleted item from q1.  It was ");put(item.itemf);new_line;
        put(".....      Name was ");put(item.namef);new_line;
      item.itemf := 'M';
      item.namef := "Moris";
        q1.insert(item);
        new_line(2);
      while not q1.emptyqueue loop
         q1.delete(item);
         put(item.namef);
      end loop;
      new_line;
      while not q2.emptyqueue loop
```

```
          q2.delete(item);
          put(item.namef);
       end loop;
       empqueue.clearqueue;
       while not empqueue.fullqueue loop
            put("Enter employee name (5 characters only!) ==> ");
            get(employee.name);put(employee.name);new_line;
            put("Does employee smoke? true or false please ==> ");new_line;
            get(employee.smoker);put(employee.smoker);new_line;
            put("How old is the employee? ==> ");
            get(employee.age);put(employee.age);new_line;
            insert(employee);
        end loop;
        while not empqueue.emptyqueue loop
             delete(employee);
             put(employee.name);put(employee.age);put(employee.smoker);
             new_line;
        end loop;
        put_line(".... End of Job!!!"); new_line;
end main3;
-------------------------------------------------------------------
--   EXECUTION RUN OF MAIN3
-------------------------------------------------------------------
deleted item from q1.  It was *
............      Name was *****


SandySallyMoris
HenryTomasLarryEnter employee name (5 characters only!) ==> Frank
Does employee smoke? true or false please ==>
TRUE
How old is the employee? ==>     39
Enter employee name (5 characters only!) ==> Carol
Does employee smoke? true or false please ==>
FALSE
How old is the employee? ==>     23
Enter employee name (5 characters only!) ==> Larry
Does employee smoke? true or false please ==>
FALSE
How old is the employee? ==>     45
Enter employee name (5 characters only!) ==> Terry
Does employee smoke? true or false please ==>
TRUE
How old is the employee? ==>     35
Frank    39TRUE
Carol    23FALSE
Larry    45FALSE
Terry    35TRUE
.... End of Job!!!
```

CS051
Handout #4

```
------------------------------------------------------------
--    FILE : pts.ada
--    This procedure creates a simple linked list of records.
--                              S. Honda      3/93
------------------------------------------------------------
with text_io; use text_io;
procedure pts is
      type rec ;
      type ptr is access rec;
      type rec is
         record
               id:character;
               link : ptr;
         end record;
      first,p : ptr;
      begin
         first := new rec'('S',null);
         first.link := new rec'('A',null);
         first.link.link := new rec'('N',null);
         first.link.link.link := new rec'('D',null);
         first.link.link.link.link := new rec'('Y',null);
         p:=first;
         while p /= null loop
               put(p.id);
               p:=p.link;
         end loop;
         new_line;
      end pts;
------------------------------------------------------------
shu.sacredheart.edu > ada pts.ada
shu.sacredheart.edu > ald -o pts.exe pts
shu.sacredheart.edu > pts.exe

SANDY
```

```
--------------------------------------------------------------------
--      FILE : linklist.ada
--      This program creates a linked list of 10 numbers.
--                              March 22, 1993  S. Honda
--------------------------------------------------------------------
with text_io; use text_io;
procedure linklist is
        type node;
        type ptr is access node;
        type node is
                record
                        info : integer;
                        next : ptr;
                end record;
        printnode, firstnode, oldnode, newnode : ptr;
        value,i : integer := 0;
        --      GE   IC INSTANTIATION
        package i    s new integer_io(integer);
        use iio;

        ------------------------------------------------------------
        --      THIS PROCEDURE PROMPTS AND GETS VALUE FROM USER
        ------------------------------------------------------------
        procedure getdata(value : in out integer) is
                begin -- getdata
                        put("Enter value : ");
                        get(value);put(value); new_line;
                end getdata;
        ------------------------------------------------------------
        begin -- linklist
                firstnode := new node;
                oldnode   := firstnode;
                GETDATA(value);
                oldnode.info := value;
                oldnode.next := null;
                for i in 2..10 loop
                        GETDATA(value);
                        newnode := new node'(value,null);
                        oldnode.next := newnode;
                        oldnode      := newnode;
                end loop;
                printnode := firstnode;
                put("List of Values :");new_line;
                while printnode /= null loop
                        put(printnode.info);
                        new_line;
                        printnode := printnode.next;
                end loop;
        end linklist;
--------------------------------------------------------------------
--      COMPILATION, LINKING, AND  EXECUTION RUN OF LINKLIST.ADA
--------------------------------------------------------------------
shu.sacredheart.edu>  ada linklist.ada            [RETURN] -- compiles
shu.sacredheart.edu>  ald -o linklist.exe linklist [RETURN] -- links,
                                                      creates executable file
shu.sacredheart.edu>  linklist.exe                [RETURN] -- run

Enter value :          53
Enter value :          -2
Enter value :          10
Enter value :           2
Enter value :          -5
Enter value :           3
Enter value :           0
Enter value :           1
Enter value :           6
```

```
Enter value :           7
List of Values :
          53
          -2
          10
           2
          -5
           3
           0
           1
           6
           7
```

```
------------------------------------------------------------------
--    PROCEDURE links.ada
--    This program creates a double link list that links all
--    records together with link, and links all smokers together
--    with smokerlink.  It uses a record within a record structure.
--    March 1993           S. Honda
------------------------------------------------------------------
with text_io; use text_io;
procedure links is
      package iio is new integer_io(integer);
      package smokeio is new enumeration_io(boolean);
      use iio,smokeio;
      type infotype is
         record
            name  : string(1..6);
            smoker: boolean;
            age   : natural;
         end record;
      type node;
      type ptr is access node;
      type node is
         record
            info       : infotype;
            link       : ptr;
            smokerlink : ptr;
         end record;
               -- variable declartions
      first            : ptr;
      ------------------------------------------------------------
      procedure getdata(first:in out ptr) is
            joe                          : file_type;
            next,last,nextsk,lastsk : ptr;
            dummy : node  := (("dummy ",false,999),null,null);
            info_rec          : infotype;
              begin -- getdata
            first := new node'(dummy);
            next  := first;
            nextsk:=first;
            open(joe,in_file,"Age.dat");
            while not end_of_file(joe) loop
                 get(joe,info_rec.name);
                 get(joe,info_rec.smoker);
                 get(joe,info_rec.age);
                 last := next;
                 next := new node'(info_rec,null,null);
                 last.link := next;
                 if next.info.smoker=true then
                       nextsk.smokerlink := next;
                       nextsk            := next;
                 end if;
            end loop;
            close(joe);
          end getdata;
      ------------------------------------------------------------
      procedure printlist(first:in out ptr) is
               next : ptr;
          begin -- printlist
               put_line(" Linked list of all...");
               put_line("--------------------");
               next := first.link;
```

```
            while next /= null loop
                  put(next.info.name);
                  put("  ");
                  put(next.info.smoker);new_line;
                  next := next.link;
            end loop;
            new_line(2);
            put_line(" Linked list of smokers...");
            put_line("--------------------------");
            next := first.smokerlink;
            while next /= null loop
                  put(next.info.name);put("  ");
                  put(next.info.smoker);
                  put(next.info.age);new_line;
                  next := next.smokerlink;
            end loop;
            put_line("   end of listing...");
        end printlist;
    ------------------------------------------------------------
    begin -- links
        getdata(first);
        printlist(first);
    end links;
-------------------------------------------------------------
--    DATA FILE : Age.Dat
-------------------------------------------------------------
Samuel false 41
Harold true  53
Edward false 55
AliceK false 75
Gerald false 99
Angela true  48
George false 40
DawnB. true  39
LizzyH false 15
JaneH. true  51
Donald false 51
Andrew true  35
-------------------------------------------------------------
--    EXECUTION RUN OF LINKS.ADA
-------------------------------------------------------------
 Linked list of all...
----------------------
Samuel   FALSE
Harold   TRUE
Edward   FALSE
AliceK   FALSE
Gerald   FALSE
Angela   TRUE
George   FALSE
DawnB.   TRUE
LizzyH   FALSE
JaneH.   TRUE
Donald   FALSE
Andrew   TRUE


 Linked list of smokers...
--------------------------
Harold   TRUE     53
```

```
Angela  TRUE    48
DawnB.  TRUE    39
JaneH.  TRUE    51
Andrew  TRUE    35
   end of listing...
```

```
--------------------------------------------------------------------------
--    FILE : LLPKG.ADA      package for link lists!
--------------------------------------------------------------------------
with text_io; use text_io;
package llpkg is
      type node;
      type ptr is access node;
      type node is
            record
                  info : integer;
                  next : ptr;
            end record;
      printnode,firstnode : ptr := null;
      newvalue : integer;
      -- for integer I/O
      package iio is new integer_io(integer);
      -- new procedures
      procedure getdata(newvalue : in out integer);
      procedure insert(firstnode: in out ptr;
                       newvalue : in integer);
      procedure printlinklist(firstnode : in ptr);
end llpkg;

--------------------------------------------------------------------------
--    FILE : LLPKGB.ADA    Package body of package LLPKG.ADA
--------------------------------------------------------------------------
package body llpkg is
      use iio;
      procedure getdata(newvalue : in out integer) is
      begin -- getdata
           put("Enter value : ");
           get(newvalue);put(newvalue);new_line;
      end getdata;
      ------------------------------------------------
      procedure insert(firstnode: in out ptr;
                       newvalue : in integer) is
          -- local declarations
          nextnode, newnode : ptr;
          found      : boolean;
          begin -- insert
            newnode := new node'(newvalue,null);
            if firstnode = null then
               firstnode := newnode;
            else
               nextnode := firstnode;
               -- check for special case inserting to front of list
               if newvalue < nextnode.info then
                   -- insert before first node
                   newnode.next := firstnode;
                   firstnode := newnode;
               else
                   -- find insertion place
                   found := false;

                   while (nextnode.next /= null) and not
                         found loop
                      if newvalue >= nextnode.next.info then
                         nextnode := nextnode.next;
                      else
                          found := true;
```

```
                    end if;
                end loop;

                -- connect pointers
                newnode.next := nextnode.next;
                nextnode.next := newnode;
            end if;    -- general case
        end if;        -- insert into nonempty list
    end insert;
    ------------------------------------------------------------
    procedure printlinklist(firstnode : in ptr) is
        printnode : ptr;
    begin -- printlinklist
        printnode := firstnode;
        new_line(2);
        put("List of Ordered Values :");new_line;
        while printnode /= null loop
            put(printnode.info,12);new_line;
            printnode := printnode.next;
        end loop;
    end printlinklist;
end llpkg;
------------------------------------------------------------------------
--    FILE : MAIN.ADA      Main procedure that uses the above package
------------------------------------------------------------------------
with llpkg,text_io; use llpkg,text_io;
procedure main is
    use iio;
    begin -- main
        getdata(newvalue);
        while newvalue /= -999 loop
            insert(firstnode,newvalue);
            getdata(newvalue);
        end loop;
        printlinklist(firstnode);
    end main;
------------------------------------------------------------------------
--    To compile package specification, package body, and the main
--    procedure; to link the main procedure; and run main:
------------------------------------------------------------------------
SHU>  ada llpkg.ada llpkgb.ada main.ada
SHU>  ald main
SHU>  a.out
------------------------------------------------------------------------
Enter value :    100
Enter value :     42
Enter value :      5
Enter value :     -3
Enter value :     38
Enter value :     97
Enter value :   -999


List of Ordered Values :
          -3
           5
          38
          42
          97
         100
```

```
------------------------------------------------------------
--    FILE : ll.ada
--    This program creates an ordered linked list of Names
--                              4/93    S. Honda
------------------------------------------------------------
with text_io; use text_io;
procedure ll is
      type node;
      type ptr is access node;
      type node is record
            info:string(1..5);
            link:ptr;
      end record;
      first : ptr;
      ------------------------------------------------
      procedure createlist (first:in out ptr) is
            old,next : ptr;
            found    : boolean := false;
            subtype nametype is string(1..5);
            name     : nametype;
            ------------------------------------------
            procedure getdata(name : out nametype) is
                  person : nametype;
                  begin -- getdata
                        put("Enter Name => ");
                        get(person); put(person); new_line;
                        name := person;
                  end getdata;
            ------------------------------------------
            begin -- createlist
                  getdata(name);
                  first := new node'(name,null);
                  old  := first;
                  getdata(name);
                  while name /= "xxxxx"  loop
                     next := new node'(name,null);
                     if old.link = null then
                           -- only one node in list
                         if old.info > name then
                              -- insert in front of head node
                              first := next;
                              next.link := old;
                         else
                              -- or insert after head node
                              old.link := next;
                         end if;
                     else  -- more than one node in link list
                         if (old = first) and (next.info < old.info)
                              then
                              first     := next;
                              next.link := old;
                         else
                              found := false;
                              while old.link /= null and not found
                                    loop
                                 if old.link.info > next.info then
                                    found := true;
                                 else
                                    old := old.link;
                                 end if;
```

```
                                end loop;
                                if found then
                                    next.link := old.link;
                                    old.link := next;
                                else
                                    old.link := next;
                                end if;

                            end if;
                        end if;
                        old := first;
                        getdata(name);
                    end loop;
            end createlist;
      ------------------------------------------------------------------
      procedure printlist (first:in ptr) is
            next : ptr;
            begin -- printlist
                  next := first;
                  put_line("People in order :");
                  put_line("-----------------");
                  while next /= null loop
                        put(next.info);
                        new_line;
                        next := next.link;
                  end loop;
            end printlist;
      ------------------------------------------------------------------
      begin -- ll
            createlist(first);
            printlist(first);
      end ll;
------------------------------------------------------------------------
--    EXECUTION RUN OF LL.ADA
------------------------------------------------------------------------
Enter Name => Larry
Enter Name => Eddie
Enter Name => Sandy
Enter Name => Andie
Enter Name => Lizzy
Enter Name => Janet
Enter Name => Allan
Enter Name => Candy
Enter Name => Gerry
Enter Name => xxxxx
People in order :
-----------------
Allan
Andie
Candy
Eddie
Gerry
Janet
Larry
Lizzy
Sandy
```

```
-------------------------------------------------------------
--    PROCEDURE links.ada
--    This program creates a double link list that links all
--    records together with link, and links all smokers together
--    with smokerlink.  It uses a record within a record structure.
--    March 1993           S. Honda
-------------------------------------------------------------
with text_io; use text_io;
procedure links is
      package iio is new integer_io(integer);
      package smokeio is new enumeration_io(boolean);
      use iio,smokeio;
      type infotype is
         record
            name  : string(1..6);
            smoker: boolean;
            age   : natural;
         end record;
      type node;
      type ptr is access node;
      type node is
         record
            info       : infotype;
            link       : ptr;
            smokerlink : ptr;
         end record;
               -- variable declartions
      first            : ptr;
      -------------------------------------------------------------
      procedure getdata(first:in out ptr) is
            joe                        : file_type;
            next,last,nextsk,lastsk : ptr;
            dummy : node  := (("dummy ",false,999),null,null);
            info_rec          : infotype;
              begin -- getdata
            first := new node'(dummy);
            next  := first;
            nextsk:= first;  lastsk:=first;
            open(joe,in_file,"Age.dat");
            while not end_of_file(joe) loop
                 get(joe,info_rec.name);
                 get(joe,info_rec.smoker);
                 get(joe,info_rec.age);
                 last := next;
                 next := new node'(info_rec,null,null);
                 last.link := next;
                 if next.info.smoker=true then
                       lastsk.smokerlink := next;
                       lastsk            := next;
                 end if;
            end loop;
            close(joe);
          end getdata;
      -------------------------------------------------------------
      procedure printlist(first:in out ptr) is
              next,last : ptr;
          begin -- printlist
              put_line(" Linked list of all...");
              put_line("---------------------");
              next := first.link;
```

```
            while next /= null loop
                  put(next.info.name);
                  put("  ");
                  put(next.info.smoker);new_line;
                  next := next.link;
            end loop;
            new_line(2);
            put_line(" Linked list of smokers...");
            put_line("-------------------------");
            next := first.smokerlink;
            while next /= null loop
                  put(next.info.name);put("  ");
                  put(next.info.smoker);
                  put(next.info.age);new_line;
                  next := next.smokerlink;
            end loop;
            put_line("   end of listing...");
        end printlist;
      ------------------------------------------------------------
      begin -- links
          getdata(first);
          printlist(first);
      end links;
-------------------------------------------------------------------
--    DATA FILE : Age.Dat
-------------------------------------------------------------------
SandyH false 41
Harold true  53
Edward false 55
AliceK false 75
GrandS false 99
Angela true  48
George false 40
DawnB. true  39
LizzyH false 15
JaneH. true  51
Donald false 51
Andrew true  35
-------------------------------------------------------------------
--    EXECUTION RUN OF LINKS.ADA
-------------------------------------------------------------------
 Linked list of all...
----------------------
SandyH   FALSE
Harold   TRUE
Edward   FALSE
AliceK   FALSE
GrandS   FALSE
Angela   TRUE
George   FALSE
DawnB.   TRUE
LizzyH   FALSE
JaneH.   TRUE
Donald   FALSE
Andrew   TRUE


 Linked list of smokers...
--------------------------
Harold   TRUE    53
```

```
Angela  TRUE    48
DawnB.  TRUE    39
JaneH.  TRUE    51
Andrew  TRUE    35
   end of listing...
```

CS051
Handout #5

Tree Traversal

This is a non-recursive psedocode for an inorder tree traversal.

```
PROCEDURE InOrder (TreeRoot : TreeType);
        (*Print the elements in the binary tree pointed to *)
        (* by TreeRoot in order from Smallest to largest.  *)

VAR
        PtrStack   : StackType;          (* stack of pointers used to  *)
                                         (* keep track of nodes until  *)
                                         (* they are printed           *)
        Ptr        : PointerType;        (* used  to traverse the tree *)

BEGIN   *( InOrder *)

     (* Start out with an empty stack *)
     ClearStack(PtrStack);

     (* Begin at the root of the tree  *)
     Ptr := TreeRoot;

  REPEAT
          (* Process until the whole tree is finished      *)
          (*  Go to the left as far as possible, pushing pointer       *)
          (*  to each node as it is passed.  Stop when Ptr falls out *)
          (*  of the tree.                                             *)

     WHILE Ptr <> NIL DO
          BEGIN
               Push(Ptr Stack, Ptr);  (* Push node pointer onto stack *)
               Ptr := Ptr^.Left;      (* Keep moving to left          *)
          END;     (* while *)

          (* If there is anything left on the stack, pop, print  *)
          (* and move to the right                                *)
          IF NOT EmptyStack (PrStack) THEN
               BEGIN
                  Pop(PtrStack, Ptr);    (* Climb back into the tree.    *)
                  PrintNode(Ptr^.Info);  (*  Print Info part of Node     *)
                  Ptr := Ptr^.Right      (*  Move once to the right.     *)
               END                       (*  If stack is not empty       *)

  UNTIL (Ptr = NIL)      and (EmptyStack(PtrStack))
END;      (*  InOrder *)
```

```
------------------------------------------------------------------
--    This procedure creates a binary tree.  The left descendant
--    of each node alphabetically precedes its parent and the
--    right descendant alphabaetically follows its parents.
--    Note the recursive procedural call.  Note also that procedures
--    are defined in other procedures.     S. Honda    4/94
------------------------------------------------------------------
with text_io; use text_io;
procedure trees is
      subtype stg is string(1..6);
      type rec;
      type ptr is access rec;
      type rec is
        record
           info : stg;
           left,right : ptr;
        end record;
      root : ptr;
      ------------------------------------------------------------
      procedure TreeCreate (root : in out ptr) is
         parent : ptr;
         name   : stg;
         found  : boolean;
         ---------------------------------------------------------
         procedure Getdata(name : in out stg) is
            begin -- Getdata
                  put("Enter a name (xx (xxxxxx to quit) ==> ");
                  get(name); put(name);new_line;
            end Getdata;
         ---------------------------------------------------------
         procedure Attach(name:in stg; parent : in out ptr) is
            begin -- Attach
                  parent := new rec'(name,null,null);
            end Attach;
         ---------------------------------------------------------
         procedure TreeSearch(parent: in out ptr; name: in stg;
                              found: in out boolean) is
            begin -- TreeSearch
                  if parent = null then
                        found := false;
                        Attach(name,parent);
                    else
                        if name = parent.info then
                              found := true;
                        else
                              if name < parent.info then
                                 TreeSearch(parent.left,name,found);
                              else
                                 TreeSearch(parent.right,name,found);
                              end if;
                        end if;
                  end if;
            end TreeSearch;
         ---------------------------------------------------------
         begin -- TreeCreate
               Getdata(name);
               -- insert first string in the root node
               if name /= "xxxxxx" then
                     Attach(name,root);
               else
```

```
                root := null;
            end if;
            Getdata(name);
            while name /= "xxxxxx" loop
               parent := root;
               TreeSearch (parent,name,found);
               if found then
                  put(name); put_line(" is already on tree! ");
               end if;
               Getdata(name);
            end loop;
      end TreeCreate;
   ------------------------------------------------
   procedure Traverse (root : in ptr) is
         -- this does an inorder traversal of a tree
         ------------------------------------------
         procedure visit (root : in ptr) is
               begin -- visit
                     put_line(root.info);
               end visit;
         ------------------------------------------
         begin -- Traverse
               if root /= null then
                     Traverse (root.left);
                     visit (root);
                     Traverse (root.right);
               end if;
         end Traverse;
    ------------------------------------------------------------
   begin -- Trees     (Main Procedure)
         root := new rec;
         TreeCreate (root);
         Traverse (root);
   end Trees;
-------------------------------------------------------------------
--    EXECUTION run of trees.ada
-------------------------------------------------------------------
Enter a name (xxxxxx to quit) ==> thrush
Enter a name (xxxxxx to quit) ==> canary
Enter a name (xxxxxx to quit) ==> osprey
Enter a name (xxxxxx to quit) ==> turkey
Enter a name (xxxxxx to quit) ==> oriole
Enter a name (xxxxxx to quit) ==> pigeon
Enter a name (xxxxxx to quit) ==> falcon
Enter a name (xxxxxx to quit) ==> canary
canary is already on tree!
Enter a name (xxxxxx to quit) ==> xxxxxx
canary
falcon
oriole
osprey
pigeon
thrush
turkey

ush
```

```
-------------------------------------------------------------------
--    This package creates a binary tree.      4/93      S. Honda
-------------------------------------------------------------------
package trepkg is
      subtype stg is string(1..6);
      type rec;
      type ptr is access rec;
      type rec is
         record
            info : stg;
            left,right : ptr;
         end record;
      root : ptr;
      procedure TreeCreate (root : in out ptr);
      procedure Traverse ( root : in ptr);
end trepkg;
```

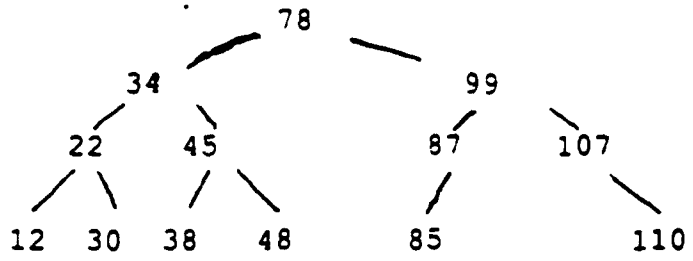
```
-------------------------------------------------------------
--    This package body creates a binary tree.  4/93      S. Honda
-------------------------------------------------------------
with text_io; use text_io;
package body trepkg is
      ---------------------------------------------------------
      procedure TreeCreate (root : in out ptr) is
         parent : ptr;
         name   : stg;
         found  : boolean;
         ---------------------------------------------------
         procedure Getdata(name : in out stg) is
            begin -- Getdata
                  put("Enter a name (xxxxxx to quit) ==> ");
                  get(name); put(name);new_line;
            end Getdata;
         ---------------------------------------------------
         procedure Attach(name:in stg; parent : in out ptr) is
            begin -- Attach
                  parent := new rec'(name,null,null);
            end Attach;
         ---------------------------------------------------
         procedure TreeSearch(parent : in out ptr; name : in stg;
                              found : in out boolean) is
            begin -- TreeSearch
                  if parent = null then
                        found := false;
                        Attach(name,parent);
                  else
                        if name = parent.info then
                              found := true;
                        else
                              if name < parent.info then
                                 TreeSearch(parent.left,name,found);
                              else
                                 TreeSearch(parent.right,name,found);
                              end if;
                        end if;
                  end if;
            end TreeSearch;
      ---------------------------------------------------------
            -- insert first string in the root node
      begin -- TreeCreate
            Getdata(name);
            if name /= "xxxxxx" then
                  Attach(name,root);
            else
                  root := null;
            end if;
            Getdata(name);
            while name /= "xxxxxx" loop
               parent := root;
               TreeSearch (parent,name,found);
               if found then
                  put(name); put_line(" is already on tree! ");
               end if;
               Getdata(name);
            end loop;
         end TreeCreate;
      ---------------------------------------------------
```

```
      procedure Traverse (root : in ptr) is
            -- this does an inorder traversal of a tree
            ------------------------------------------
            procedure visit (node : in ptr) is
                  begin -- visit
                        put_line(node.info);
                  end visit;
            ------------------------------------------
            begin -- Traverse
                  if root /= null then
                        Traverse (root.left);
                        visit (root);
                        Traverse (root.right);
                  end if;
            end Traverse;
end trepkg;
```

```
-------------------------------------------------------------------
--    MAIN PROCEDURE  tremain.ada            S. Honda    4/93
-------------------------------------------------------------------
with trepkg; use trepkg;
procedure tremain is
   begin -- tremain
       root := new rec;
       TreeCreate (root);
       Traverse (root);
   end tremain;
-------------------------------------------------------------------
--    EXECUTION RUN OF tremain.ada
-------------------------------------------------------------------
Enter a name (xxxxxx to quit) ==> osprey
Enter a name (xxxxxx to quit) ==> falcon
Enter a name (xxxxxx to quit) ==> turkey
Enter a name (xxxxxx to quit) ==> pigeon
Enter a name (xxxxxx to quit) ==> osprey
osprey is already on tree!
Enter a name (xxxxxx to quit) ==> oriole
Enter a name (xxxxxx to quit) ==> canary
Enter a name (xxxxxx to quit) ==> thrush
Enter a name (xxxxxx to quit) ==> xxxxxx
canary
falcon
oriole
osprey
pigeon
thrush
turkey
```

DELETION OF NODES FROM BINARY TREES

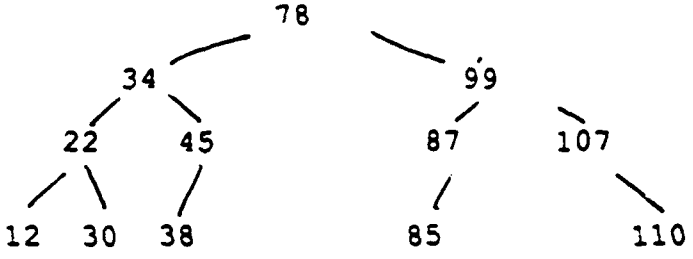
SAMPLE TREE



CASE 1 TO DELETE A NODE WHICH IS A LEAF (EASY)
JUST PLUCK IT OFF THE TREE

EXAMPLE ....DELETE 48 FROM ABOVE TREE

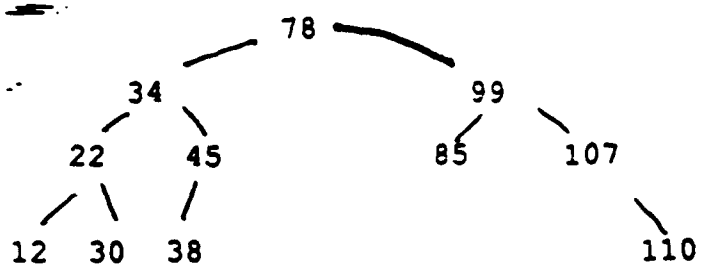
NEW TREE



CASE 2 TO DELETE A NODE WITH ONE CHILD (STILL EASY)
THE CHILD REPLACES THE PARENT

EXAMPLE....DELETE THE 87 FROM ABOVE TREE

NEW TREE

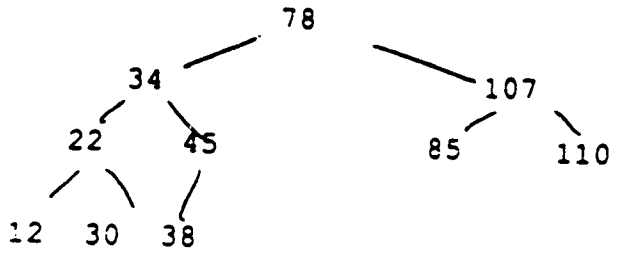


CASE 3 TO DELETE A NODE WITH TWO CHILDREN (NOT SO EASY)
RIGHT CHILD REPLACES DELETED NODE....ANY LEFT SUBTREE APPENDED
TO THE LEFT OF RIGHT SUBTREE

EXAMPLE...DELETE 99

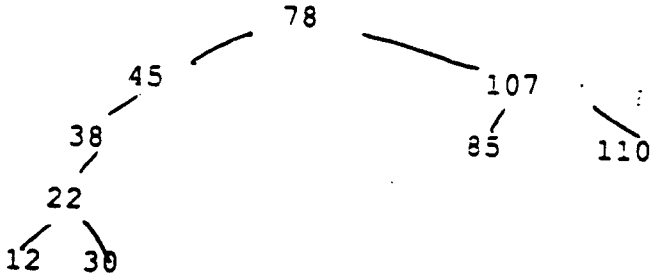
P2

NEW TREE



EXAMPLE...DELETE 34

NEW TREE



Appendix D

CS050 Projects

Homework Assignment #1

Write your own ada program that will output a design or picture. Be creative and show off your artistic talent! Your program should have a header and an executable body. It also should contain the context clause, "with text_io; use text_io;" because you need to make available the subprograms put_line and new_line. I will electronically collect your programs and create a class procedure that will contain all your files as subprograms called procedures. I will send you this file via the mail facility. I will also create a class package that will contain all your artistic work and ship this to you too, so you see how packages that contain reusable components are created.

Homework Assignment #2

Write flowcharts using only the three logic patterns discussed in class, the simple sequence, decision logic pattern, and the repetitious logic pattern. Use only the symbols discussed in class: the start/end, I/O, processing, decision making and the connection flowcharting symbols. Draw your designs as I have in class. Make certain you define the domain of the problem. Decide on the variable names you plan to use.

Homework Assignment #2b

Convert your flowcharts into Ada programs. Remember to document your programs with your name, program description, and the current date. Use good program style for readability and for later maintenance. Use proper identifier names, indentation, and comments throughout your programs. Turn in your programs along with its associated flowcharts when done.

Homework Assignment #3

1. Develop example programs using all 3 control structures.
   - IF - ELSE - END IF
   - IF - ELSIF - ELSE - END IF
   - CASE

   Include also, enumeration data types and string variables. Do some I/O with these enumeration types. Be creative! Draw flowcharts indicating logic and define domain. Remember to document your programs with your name, date program description and remember to use proper indentation and naming conventions for readability.

2. Decide which case statements are legal on handout sheet. If they are illegal, state why.

Homework Assignment #4

From the following three problems, select two to code in Ada. Remember, you need to create data files using the VI editor before you execute your Ada programs. You may want to create your own files with varying amount of records to test your programs with. Think about the problem domain and make sure you draw flowcharts before you begin to code. Remember to use good program style! What kind of documentation should be written?

Homework Assignment #5

Redo the last homework assignment using subprograms to write modular code. Explain the parameters that you pass. (Why they were passed and mode selected.)

Also create your own program(s) that uses file(s), procedure(s), function(s), enumeration type(s), and for statement(s), and a case statement.

Homework Assignment #6

Create your own program that uses exception handlers. Clearly define your problem, problem domain, remember to use good Ada style.

Homework Assignment #7

Create a file with interesting data records in it. Create an Ada program that will

- invoke procedure **GetData** to read the file into arrays.
- invoke a function that uses the array(s) to process one value (call the function whatever you like)
- invoke another procedure to compute values from the array(s) into a new array. (Name the procedure yourself)
- invoke another procedure called **PrintList** to values of the new array.

Remember to document your program well explaining what your program does.

Assignment #8

Read the following problem and write an Ada program for it, using subprograms to make it modular in design. Do part a, b, and c. Think about what could be declared local to the subprograms and what may not.

Personnel Salary Budget. The personnel office for a state government agency is in the process of developing a salary budget for the next fiscal year. The personnel file contains the following information on each employee

1. Employee name
2. Social security number
3. Current annual salary
4. Union code (1 = clerical, 2 = teachers, 3 = electrical)
5. Current step in pay schedule (1 through 5)
6. Year hired

The state agency deals with three labor unions: clerical, teachers, and electrical. Each union has negotiated a separate salary schedule which entitles each employee to an annual step increase. The salary schedules are listed in the table below. Each employee is hired at the lowest step in the salary schedule for their union, and moves up one step each year. The field "current step in pay schedule" indicates the employee's step prior to the new salary for the coming year; that is, "current annual salary" is consistent with this step. The salary for the upcoming year is to be based on the next highest step. Employees who have reached step 5 are at the maximum salary level for that job. Thus, next year's step salary is the same as their current annual salary.

In addition to the salary step increase, employees who have been employed by the state for 10 years or more are entitled to a longevity increase. A longevity increase represents a 5 percent increment added to the employee's *new* step salary.

Salary Schedules

Step	Clerical	Teachers	Electrical
1	20176	29133	32170
2	20592	30433	44260
3	20956	31833	46668
4	21320	33333	49501
5	21921	34893	52801

Personnel File

SMYTHE SMILE	032166789	10956	1	3	91
ALFRED ALFREDO	123454321	13333	2	4	88
MENDAL MICKEY	987654345	22801	3	5	87
FIELD FLORA	543297541	12170	3	1	96
CURRAN CURRENT	045811222	10176	1	1	76
HANDEL HALO	315791123	11320	1	4	90
UNKIND CORA	129834765	9133	2	1	75

a. Prepare a flowchart and write a program that prints a budget report for the personnel office. Output from the report includes employee's name, current salary, increase in salary due to step, increase in salary due to longevity, and new salary. Following the output table, print totals for the four numeric columns. Treat the salary schedules as a two-dimensional (5 × 3) array that is to be read in. Data in the personnel file and in the output table need not be treated as arrays.

b. Print a table which summarizes the salary budgets as follows:

SALARY BUDGETS

CLERICAL	$ xxxxxx
TEACHERS	$ xxxxxx
ELECTRICAL	$ xxxxxx
	$ xxxxxxx

** c. Print the table of part b *prior* to the output in part a. *Hint*. Unlike part a, now you must subscript both the variables in the personnel file and the output in the report of part a. Do you see why? Use two-dimensional arrays.

Appendix D

CS051
Projects

Cs 51 Project 2 SPRING 93
D. Pinto/S. Honda

DUE : FEB 22

USING THE SAMPLE STACK PROGRAM(S) GIVEN IN CLASS, CREATE A MAIN PROGRAM TO READ IN UP TO 100 INTEGERS. THE STACK SHOULD BE ABLE TO HOLD A MAXIMUM OF 20 NUMBERS AT ONE TIME.IF THE INTEGER INPUT IS EVEN (BUT NOT 0), THE INTEGER IS PUSHED ONTO THE STACK. IF ODD, THE STACK IS POPPED.IF 0 IS INPUT , THE PROGRAM SHOULD TERMINATE.

WHENEVER A VALUE IS POPPED FROM THE STACK , AN APPROPRIATE MESSAGE SHOULD BE DISPLAYED. IF EITHER STACK IS EMPTY OR STACK IS FULL , AN APPROPRIATE MESSAGE SHOULD ALSO OCCUR. All CASES SHOULD BE COVERED , IE. , THERE SHOULD BE AT LEAST ONE INSTANCE OF EMPTYSTACK AND ONE OF FULLSTACK.

CS 51

SPRING 93
Project 3

due : March 22

The Bashemin Parking Garage contains a single lane that holds up to fifteen cars. Cars arrive at and depart from the same end of the garage.
If a customer arrives to pick up a car that is not on the end , all cars in the way are driven out,and then restored in the same order that they were in originally.

Write a program that reads a group of input lines. Each line contains an 'a' for arrival or a 'd' for departure and a license plate number. Cars are assumed to arrive and depart in the order specified by the input. The program should print a message each time that a car arrives or departs. When a car arrives , the message should specify whether or not there is room in the garage for the car. If there is no room for a car , the car waits in a queue until there is room or until a departure line is read for the car.
When room becomes available , another message should be printed. When a car departs, the message should include the number of times the car was moved within the garage (including the departure itself but not he arrival). This number is zero if the car departs from the waiting line.

ENJOY!!

CS051 Project 4
Spring 93

Due : April 13

1. Using the linked list program given in class, alter the program to input twenty five strings, sorting them alphabetically as they are inserted into the list. Print out the sorted list.

2. Now, using the above input ten strings. If the string is present in the list, delete it, otherwise insert it in its proper place. Print out the list after each insertion and deletion.

CS 51 Project 5 Spring 93
Pinto/Honda

Due : May 10 2001

Using the tree programs enclosed , write a program to input wenty five strings into the tree. The program should print out the norder traversal for the tree. Then, compute the levels of each node n the tree and print out the maximum and minimum leaf levels. Do this or ten runs , (ie. ten different trees , and compute the average max nd average min leaf levels) ENJOY!!!!

HAPPY SUMMER!!!!!!

Appendix E

Exams

Name: Date:

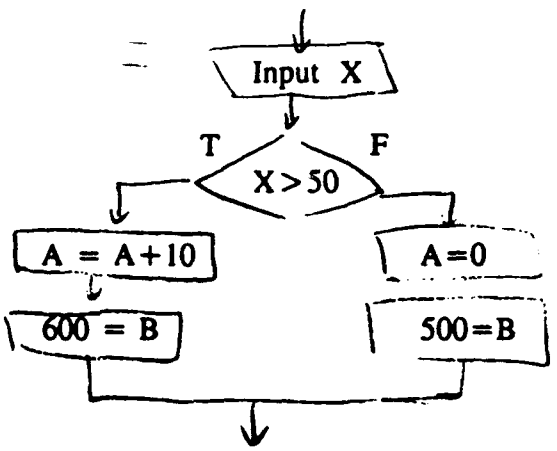
CS050 Fall 1992

Exam I

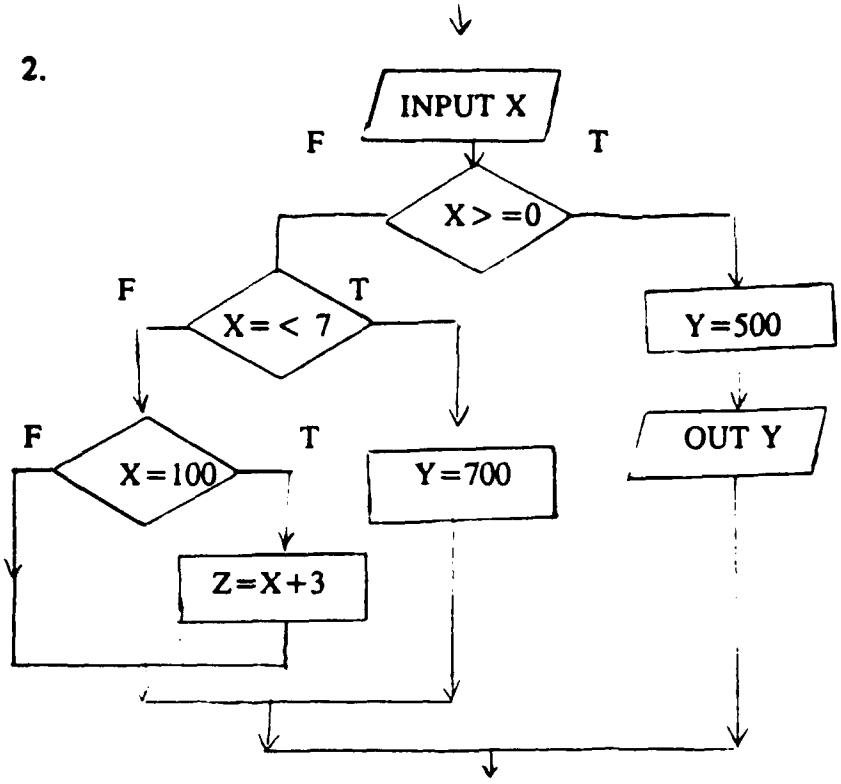
Part I Give an example of the following: (1 point each)

1. assignment statement
2. discrete data type
3. logical constant
4. numeric expression
5. logical expression
6. relational operator
7. character constant
8. logical operator
9. generic package
10. ada lexical unit

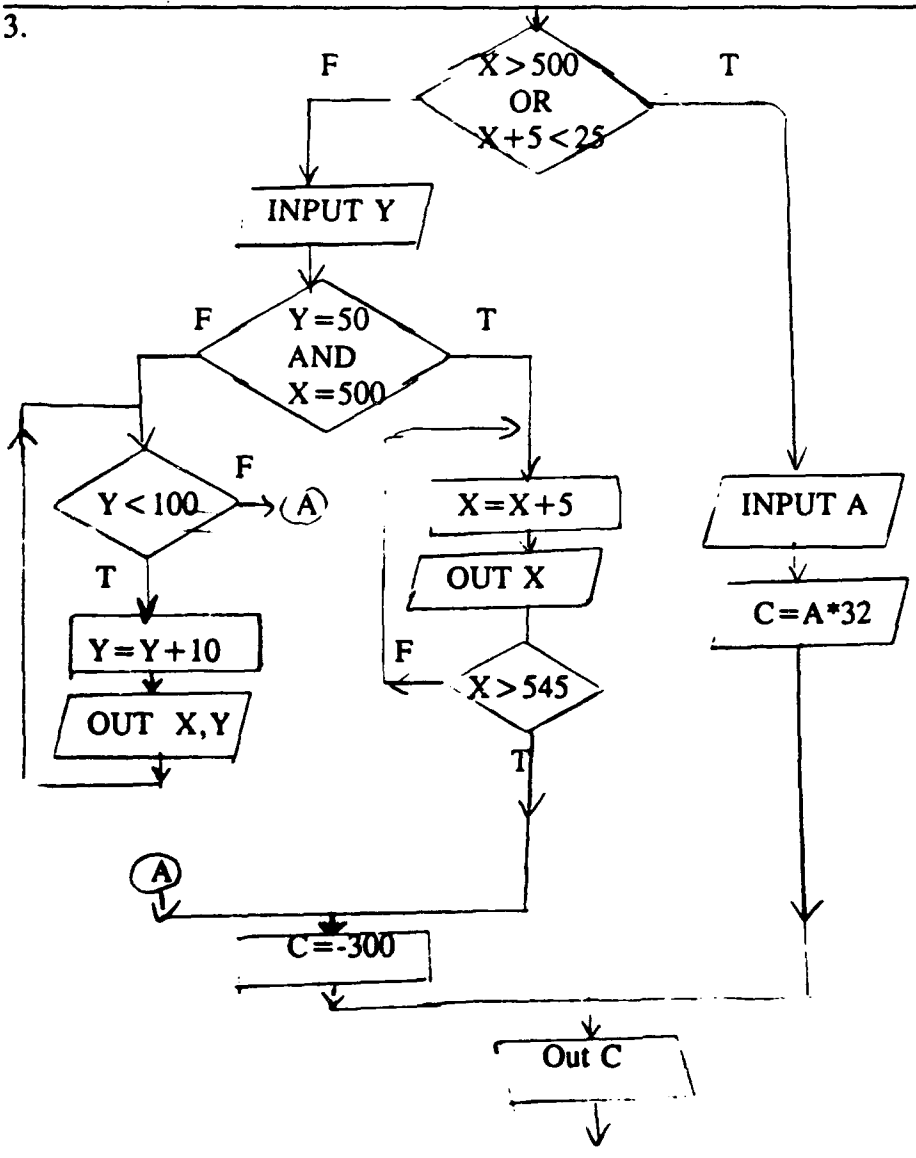
Part II. Express the following flowchart segments as appropriate Ada instructions. Assume the declarations are done. (5 Points per problem)



2.


INPUT X
F
T
X> =0
F
T
X= < 7
Y=500
OUT Y
F
T
X=100
Y=700
Z=X+3

3.


F
X>500
OR
X+5<25
T
INPUT Y
F
Y=50
AND
X=500
T
F
Y<100
A
T
Y=Y+10
OUT X,Y
X=X+5
OUT X
F
X>545
T
INPUT A
C=A*32
A
C=-300
Out C

Part III. Program the following in ADA. (Write the entire Program).
Mr. and Mrs John Doe need a program to compute their income tax and are trying to decide whether to file a **Joint** or **Separate** return. Mr. Doe's taxable income is $18,750 and Mrs. Doe's income is $20,312.

For **Separate Returns**:

Taxable Income $18,000 - $20,000 Pay IRS $1630 + 28% of the taxable amount over $18,000

Taxable Income $20,000 - $22,000 Pay IRS $2190 + 32% of the taxable amount over $20,000

For **Joint Returns**:

Taxable Income $36,000 - $40,000 Pay IRS $3960 + 29% of the taxable amount over $36,000

Print out the following:

1. How much each of the separate returns for Mr. and Mrs. Doe will be,
2. How much the joint return will be, and
3. A message indicating which way the **computer** thinks they should file: (separate or joint) Returns.

Name _____ Fall 1993

Date _____ CS050A

Exam II

1. Write an Ada program containing a function and a procedure that will do the following:

We have three sensors, Sensor1, Sensor2, and Sensor3 on our system, each of type Sensor_State which can be HIGH, MEDIUM OR LOW. If Sensor1 is HIGH, Sensor2 and Sensor3 are MEDIUM, the alarm should be set on. If Sensor 1 and Sensor2 are HIGH the alarm should be also set on. If sensor3 is HIGH, the alarm should also be set on.

Write an Ada program that will

1. Call a procedure called GET_DATA to input a record containing a system code and three sensor values from a file called DATA.DAT. (see file below) to check the state of the system at a certain time.

2. Invoke the function called ALARM_ON to return a logical value of TRUE if the alarm should be set on; FALSE otherwise. The function should determine from the sensor values whether or not the alarm should be set.

3. Say, "Run for your life! *** system" ___ "about to blow!", if the alarm is set, or another appropriate message if the alarm is not set.

DATA.DAT (field values are system code, sensor1, sensor2, and sensor3)

A	MEDIUM	LOW	LOW
U	MEDIUM	LOW	MEDIUM
D	LOW	LOW	LOW
E	LOW	MEDIUM	MEDIUM
F	MEDIUM	MEDIUM	MEDIUM
Z	MEDIUM	HIGH	MEDIUM
Y	MEDIUM	HIGH	HIGH
.	.	.	.
.	.	.	.
.	.	.	.

Part II.

1. Declare a variable **Season** as a **Season_Type** (SPRING,SUMMER, AUTUMN,WINTER):

2. Declare a variable **Forecast** as a **Forecast_Type** (RAIN,SUNSHINE, PARTLY_CLOUDY,SNOW):

3. Assuming that the following procedures have been defined:
   CLEAN_THE_HOUSE
   FLY_A_KITE
   GO_SAILING
   CARVE_A_TURKEY
   SHIVER

4. Write a case statement testing the variable Season to do the following: for each of the 4 Season_Type values, assign an appropriate forecast_type to forecast, print appropriate message(s) and select appropriate procedure(s) to run.

Name Fall 93

CS050 Ada Programming

Final Exam

Part I. Write an Ada program that will analyze stock information. INPUT consists of ID and QUANTITY_IN_STOCK for each of twenty five products.

A. Write a Main program that will call four subprograms called **INPUT, CATEGORIZE, OUTPUT** and **LIST**.

B. **INPUT** should read the identification numbers, and quantities in stock, into two arrays called **ID** and **QTY.**

C. **CATEGORIZE** should determine the number of products whose quantity in stock fall into each of the following categories.

500 or MORE
250 to 499
100 to 249
0 to 99

D. **OUTPUT** should output the number whose quantities in stock fall into each of the following categories:

500 or MORE
250 to 499
100 to 249
0 to 99

E. **LIST** should output Product ID, the QTY in stock, and a message to reorder immediately if quantities fall below 150 of the products in low quantities.(under 150).

ID	QTY	MESSAGE
ITEM103	121	*** REORDER***
ITEM114	32	***REORDER***
ITEM165	140	***REORDER***
ITEM212	99	***REORDER***

Part II.

Note the following program below. What will be the expected output?

```
with text_io; use text_io;
procedure fin is
   message:string(1..18);
   joe:file_type;
   begin
      open (joe,in_file,"file.dat");
      for i in reverse 1..18 loop
            get(joe,message(i));
      end loop;
      close(joe);
      for j in 1..16 loop
            put(message(j));
      end loop;
      new_line;put("   ");
      for i in 1..3 loop
           put(message(17));
           put(message(18));
           put(message(16));
      end loop; new_line;
    end fin;
```

FILE FILE.DAT

O
H
!
A
K
A
M
I
K
I
L
A
K

E
L
E
M

Part III. Write a program that will read a file called Numbers.dat that contains 5 records of 5 integer values to detect if it is a magic square or not. Read this into memory in a variable called Magic. Magic should hold five rows and five columns of integer values. Your program should check to see if all the sums of the rows and all of the sums of the columns, the left diagonal, and the right diagonal are of equal value. If the sums are all equal, this is a MAGIC SQUARE. Your program should print out whether or not the values input form the file make a magic square or not.

Part IV. Answer the following questions:

Some of the goals of software engineering are

- modifiability
- efficiency
- reliability
- understandability.

1. How are these goals met by the Ada language?

Some of Ada's features

- generic components
- separate compilations of modules (information hiding)
- packaging concept
- subprogramming modularity
- strong typing
- separation of specification code from implementation code

2. Now that you have been exposed to Ada language, list and describe some of Ada's features that can be used as a software engineering tool to aid in the design of software projects.

Name :

Spring 1993

CS051

Section A - Honda

Exam I

Do all work on paper provided ...Show all work!!!

I. Find the output for the following segments of code

```
CLEARSTACK (S);
Z := 10;
PUSH (S,Z);
PUSH (S,5);
POP   (S,A);
PUSH (S,2*A);
X := 4*A;
POP   (S,Y);
PUSH (S,X+Y);
Y := A + X;
POP   (S,X);
POP   (S,Y);
PUSH (S,A);
POP   (S,Z);
PUSH (S,Y*10);
WHILE NOT EMPTY (S) DO
      BEGIN
            POP (S,A);
            WRITELN (A);
      END;
WRITELN (A,X,Y,Z);
```

II. What is output by the following segment of code?

```
CLEARSTACK (STACK);
PUSH (STACK,1);
WHILE NOT EMPTY (STACK) DO
      BEGIN
            POP (STACK, N);
            WRITE(N);
            IF N<=6  THEN
                  BEGIN
                        PUSH (STACK, N+1);
                        PUSH (STACK,  3*N-1)
                  END (* IF *)
            END;  (* WHILE *)
```

III. Write the following **declaration** of a **variable** called **COST**. It should look like this in primary memory: I want to store the average prices for a 4-Door, 2-Door, and the sports coupe for 1986,1987,1988,1989,and 1990.

GM	FOUR-D	TWO-D	SPORTS-COUPE
CHRYSLER			
PONTIAC			
SABB			
HONDA			
NISSAN			

for 1986

IV. Write a complete program to input up to a maximum of 10 decimal numbers into an array. The program is to use a procedure to reverse the values in the array and also to count the number of negative values in the array.

NAME
CS 51

SPRING 93

FINAL EXAMINATION

I. GIVEN THE FOLLOWING SET OF INTEGERS , PLACE THEM IN A TREE USING THE INSERTION ROUTINE GIVEN IN CLASS. DRAW THE TREE.

60 56 87 98 34 65 23 11 90 69 45 62 53 89 14 9 100 81 33 59

WRITE OUT THE NODES

INORDER

POSTORDER

PREORDER

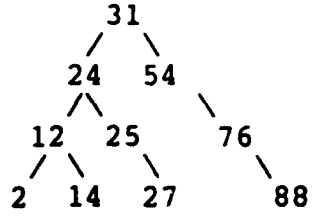
SUPPOSE YOU WANTED TO INSERT 77 AND 41 AND THEN DELETE 65 AND 9

DRAW THE TREE AFTER THESE FOUR OPERATIONS

II. WRITE A FUNCTION NUMLEAVES THAT RETURNS THE NUMBER OF LEAVES IN ANY NONEMPTY BINARY TREE WHEN GIVEN THE POINTER TO THE ROOT OF THE TREE.

III. SHOW THAT A COMPLETE BINARY TREE WITH N LEAFS CONTAINS 2N-1 NODES. DO THIS FOR A FEW EXAMPLES AND TRY TO REASON YOUR WAY THROUGH THIS MATHEMATICALLY.(NB COMPLETE MEANS LEVELS ARE AS FULL AS POSSIBLE)

IV. GIVEN THE FOLLOWING TREE , SHOW HOW THE PREORDER RECURSIVE FUNCTION WOULD OPERATE (IE . PERFORM A TRACE)



V. **GIVEN THE FOLLOWING CODE , ASSUME WE HAVE THE TREE LISTED ABOVE. TREE POINTS TO THE NODE CONTAINING 31. WHAT IS THE OUTPUT?ASSUME ALL STACK OPERATIONS.**

```
ClearStack(PtrStack);
Ptr := Tree;
REPEAT
    WHILE Ptr <> NIL DO
     BEGIN
        WriteLn(Ptr^.Info);
        Push(PtrStack,Ptr);
        Ptr := Ptr ^.LEFT;
     END;
    IF NOT EmptyStack(PtrStack) Then
     BEGIN
       Pop(PtrStack,Ptr);
       Ptr := Ptr^.Right;
     END
UNTIL (Ptr=NIL) AND (EmptyStack(PtrStack))
```

Appendix F

Articles

Daily Breeze

WEDNESDAY
September 6, 1989 ZONE A,B,C

CSC signs $134 million pact

By Laurel Sweeniste
AEROSPACE WRITER

El Segundo-based Computer Sciences Corp. has signed a $134 million contract with International Business Machines Corp. to help the Federal Aviation Administration get its computers ready to deal with an anticipated doubling of air traffic in the next decade.

IBM is the prime contractor for the work, which spans 13 years beginning in November 1988.

The prime contract for the Advanced Automation System is for $3.6 billion, part of an 18-year, $12 billion modernization program scheduled to be complete by the turn of the century, said FAA spokesman Fred Farrar.

CSC's share of the work will be to develop, test, install and integrate the software. The programs will be written in Ada, the standard government programming language.

The work will be performed at the company's System Sciences Division in Calverton, Md. No new jobs will be created in the South Bay.

Options for additional work during the base period and support through 2010 could potentially bring the value of the contract to more than $700 million, the CSC said.

The company will write more than 800,000 lines of Ada code for processing flight plans, predicting and resolving potential conflicts in aircraft flight paths, providing weather data, simulating air traffic situations for training air traffic controllers and analyzing system activities.

The software is for a new IBM computer already installed but still using old software, Farrar said.

"There will be an increased amount of automation in the system, built into the software," Farrar said. "It will not automate the controlling of airplanes, but it will automate many of the clerical or routine functions," allowing faster access to information on flights.

No reduction in the number of controllers is anticipated as a result of the increased automation, he said.

The FAA award comes on the heels of the company's announcement last Friday that it has been selected as one of four companies in the running to design a system aimed at cutting down on the paperwork in the U.S. Army's procedures for buying and maintaining weapon systems.

The Army awarded CSC a 13-month, $2.7 million contract for the initial design competition. Designing and developing the system, dubbed Computer-aided Acquisition and Logistics Support, could potentially be worth several hundred million dollars.

The company's Integrated Systems Division will lead the CALS work

CSC/BACK PAGE

CSC

FROM PAGE C1

through its Moorestown, N.J. headquarters.

The CSC team consists of McDonnell Douglas Corp., St. Louis; General Research Corp., Santa Barbara; West Coast Information Systems, Walnut Creek; and Management Systems Associates, Raleigh, N.C.

With revenues of $1.3 billion and 20,000 employees, CSC is the largest independent professional-services company in the computer industry.

Computer Sciences's clients include military bases, the state of New Jersey and NASA. Only about 300 of the company's 20,500 employees work in El Segundo, where the company was founded in 1959 and which still is home to corporate headquarters. About 12,000 work in Washington, D.C., where the company does much of its business, while others are in Boston.

Vol. 8, No. 15

WASHINGTON TECHNOLOGY

The Business Newspaper of Technology

November 4, 1993

OPINION

An Endorsement at the Top

DoD Needs to Increase the Appeal of Ada Beyond Its Own Halls

By Emmett Paige Jr.

Emmett Paige Jr. is Assistant Secretary of Defense for Command Control Communications and Intelligence. This article is adapted from a speech to a recent ADA dual-use workshop.

I'm not a supporter of Ada for the sake of Ada. If I didn't believe in it, if I didn't think it was necessary, I never would have been a supporter of it since 1979, when I moved across the highway at Fort Monmouth to find out what it was all about.

The whole reason for Ada, in the first place, still exists. The need today is probably greater than it ever was, greater than it was then. And a lot of people that didn't even believe in automated systems back in those days are now believers in automated systems.

Ada has been a standardized programming language since 1983. Since its introduction, DoD and the software engineering community have benefitted greatly from Ada. And with the features that Ada 9X promises to bring to the table, things will get even better. While we took some early heat because of lack of quality and validated compilers in the early days, the results to date show quantitatively that Ada makes sense, both technically and from a business point of view.

Needless to say, the policy official in DoD who's responsible for C^{4}I is firmly committed to the Ada strategy. Any speculation that DoD is wavering on the commitment to Ada is wrong. Based upon the results that we've seen, we have no compelling reason to rethink our Ada strategy.

This strategy uses Ada as the kingpin to bring a software engineering discipline into DoD. And in my humble view, we have held Ada too tightly in the past. As someone said, it's just not getting enough air, and we've got to loosen our grip.

Based on my experience within the Department of Defense, combined with my recent experience in the private sector, there appears to be a common misconception that Ada has not received the level of support that is necessary to ensure its acceptance within all of the Department of Defense and the commercial sectors.

I know for a fact that there are many private firms that have embraced Ada. I've read that Silicon Graphics is using Ada in some of its virtual reality simulations. I'm sure that they would not be using Ada if they did not think it was the best language to use for the particular purpose.

NASA and FAA are using Ada in their major projects. Certainly, we will not blame all of the computer problems and other disasters that NASA has suffered in the past few years to their use of the Ada language. Likewise, we cannot blame FAA's use of Ada for the program delays that they've encountered. Many firms overseas, like Ferranti (for nuclear electric applications) and Nippon Telegraph and Telephone are using Ada with positive results.

However, while I have not witnessed these examples firsthand, I nonetheless feel that Ada has not penetrated the commercial sector to the degree that we, within DoD, had hoped. The Ada market is perceived as a niche market by many of the vendors that I talk with. And I believe there is a lot of truth to this assessment.

We've got to increase the appeal of Ada outside of the Department of Defense and outside the federal government. Else, we will forever have to carry the burden of the industrial tower of language Babel on our shoulders. While we are not afraid to continue our investments, the best of all possible worlds would be one in which the market caused others to spend their own money to enhance their market share of a larger Ada market internationally. The pull needs to accompany the push. Else, our chances of success in the future will be limited.

I would like to see U.S. companies leading the world and building jobs in this country producing applications using the Ada language. In my view, we in the federal government have tried to prime the pump with Ada, but we have failed to find the answer to cause the commercial sector in the U.S. to pick up the Ada baton and use it for their own purposes.

A few years ago, I used to say that the day we see IBM adopt Ada for their commercial systems will be the day that we can say Ada has arrived.

Well, today we see Microsoft and Borland using Ada. We can say that Ada has arrived.

There are other issues as well. A lot of people still believe Ada is overkill, slow and non-responsive. Well, we've come a long way in the area of performance. The benchmarks indicate that most Ada applications run as fast and are more robust than their counterparts in other languages. With regard to

richness, this has never been a handicap to those properly trained in software engineering and Ada. As a matter of fact, they love the capabilities the language provides them.

The availability of quality compilers, tools and environments (including linkages) has also been raised as an issue in the past. Well, we've come a long way in this area in the past 10 years. Those of you old enough to remember the early compilers will recall, as I do, the inefficient beasts that consumed whole machines. Today's compil-

continued on back

Large Ada projects show productivity gains

Ware Myers, Contributing Editor

After years of development and an initial skeptical reception, many people are now using Ada and saying that they like it. At least 91 projects have been completed in Ada, 103 are under development, and 38 are in the planning stage, according to a March survey by the Defense Dept.'s Ada Joint Program Office. About 13 percent of these 241 projects were large: more than 100,000 lines of source code; more than 3 percent have more than 500,000 lines. The survey covered Defense Dept., commercial domestic, and foreign projects.

The growth in Ada's use has been helped by favorable reports from early adopters ("Ada Catches on in the Commercial Market," Soft News, *IEEE Software*, November 1986, p. 81) and by the growing number of validated compilers. As of June there were 129 validated base Ada compilers and 63 derived compilers.

But not everyone has the tools needed. "A surprise is that Ada is as far behind as it is. The support environments aren't there: tools, compilers, prototypes, bindings," said Howard Yudkin, chief executive officer of the Software Productivity Consorium, a 14-member group of defense contractors based in Reston, Va.

In Europe, the lack of Ada environments on IBM and Cray computers — the computers used for atomic energy and other large projects — has constrained use of Ada, said Annie Kuntzmann-Combelles, managing director of software engineering and applications at CISI Engineering in Rungis, France.

Large productivity gains. The largest embedded system completed to date is the US Army's Advanced Field Artillery Tactical Data System, said Allan Kopp, the Ada Joint Program Office's deputy director. The system entered formal qualification testing in July. The project results show that Ada can greatly increase productivity for large systems.

Release 4.04 of the system contains 1,175,498 noncomment lines of source code and 7,553 files. All but 3,000 lines of operating-system and communication software were written in Ada.

The line count does not include reusable software, such as math packages, not developed on the project. Moreover, reusable software developed on the project was counted only once. Roughly 13 percent of the delivered software was reusable. This reuse saved 190 man-months of effort (a 9-percent savings) and reduced the schedule by two calendar months (a 4-percent savings), said Donald G. Firesmith of Magnavox Electronic Systems, the system's contractor. Magnavox expects to increase the reuse rate to 25 percent on the next similar project and believes that a rate of 50 percent is possible, he said.

Productivity for the execution environment — including the operating system, data management, information management, communications support, and communications interface — was 550 lines per man-month, Firesmith said. Productivity for the applications software — including fire-support planning, fire-support execution, movement control, and common functions — was 704 lines per man-month, he said.

The Magnavox rates exceed the average productivity of the 1,500 systems in productivity consultant Lawrence Putnam's database: 77 lines per man-month (at the 1.2-million-line level).

Magnavox found that using Ada meant that more time went to requirements analysis and less to integration and testing than is typical on traditional projects, Firesmith said. Requirements analysis and design took 55 percent of the effort, coding took 10 percent, and testing and integration took 35 percent, he said. Ada's object orientation meant that 90 percent of the code was very small and simple programming units, which caused far fewer problems during integration and test, he said.

In one project, Ada's object orientation meant that 90 percent of the code was very small, simplifying system integration and test.

Magnavox also found that Ada-oriented development methods, such as object-oriented design, are not compatible with the functional-decomposition methods or waterfall life-cycle models that have been commonly used in Defense Dept. standards, Firesmith said. (The recently revised DoD-Std-2167-A is methodology- and language-neutral.)

Other projects. Several companies have shown similar results with Ada:

- Nokia Information Systems has completed 2 million lines in a variety of systems for the Bank of Finland. In the early 1980s, Nokia built a compiler, operating system, and environment — all in Ada. The results of the banking project have been so successful that Nokia plans to develop its next point-of-sale system in Ada.
- Boeing Aerospace decided three years ago to standardize on Ada, for both defense systems and its commercial aircraft. The new Boeing 747-400 plane now contains 500,000 lines of Federal Aviation Administration flight-certified Ada software, most of it produced by subcontractors. The cost savings may be as high as 30 percent, said Boeing's Brian Pflug.
- France's CISI Engineering saw a productivity increase of 20 percent compared to C, Fortran, and Pascal in software-engineering-tool projects, said CISI's Kuntzmann-Combelles. A team well-trained and experienced with Ada increased its productivity from 40 lines of code per man-day to 80 lines, but she cautioned that less experienced programmers would do less well.

From 1984 to 1986, the European Community had a special dissemination program for Ada, which West Germany and Britain were very active in, Kuntzmann-Combelles said. France's Defense Ministry recently mandated Ada for real-time, image-processing, and robotics systems.

The US Defense Dept. has strengthened its mandates to use Ada, and several nonmilitary agencies — including the National Aeronautics and Space Administration and the Federal Aviation Administration — are turning to Ada.

- Lockheed Missile and Space Corp. is several years into the Ada development of the Air Force's 400,000-line Milstar system for control-, command-, and telemetry-processing for a spacecraft.
- The Air Force's Advanced Tactical Fighter will require 5 million to 10 million lines of code for on-board and ground-based software. The competing prime contractor groups — led by Lockheed and Northrop — both proposed to use Ada.
- NASA has committed to Ada for the 10 million lines of code that the *Freedom* space station is expected to need. The agency is switching from its long-time language, Hal-S. The agency has done more than 150 projects in Ada in the last five years. Pilot projects total 313,000 lines and production efforts total 446,000 lines.
- The European Space Agency recently decided to use Ada for its *Columbus* space-station module and *Hermes* space shuttle.
- The FAA will use Ada as the high-order language for its Advanced Automation System, an air-traffic control system estimated to require 1.5 million lines of new code. Air-traffic control systems for Belgium and Spain are also using Ada.

IBM Forges Links to Ada Vendors To Enhance Role in Aerospace Market

BOSTON

International Business Machines has signed marketing agreements with three Ada software and hardware vendors to make it easier for aerospace companies to use IBM computers to design and run software in Ada, a high-level programming language developed by the Defense Dept.

The agreements, signed late last year with three of the leading vendors for Ada compilers and related tools, will allow IBM sales representatives to make joint calls with companies that enjoy considerable Ada business with aerospace defense contractors.

DETAILS OF PACTS

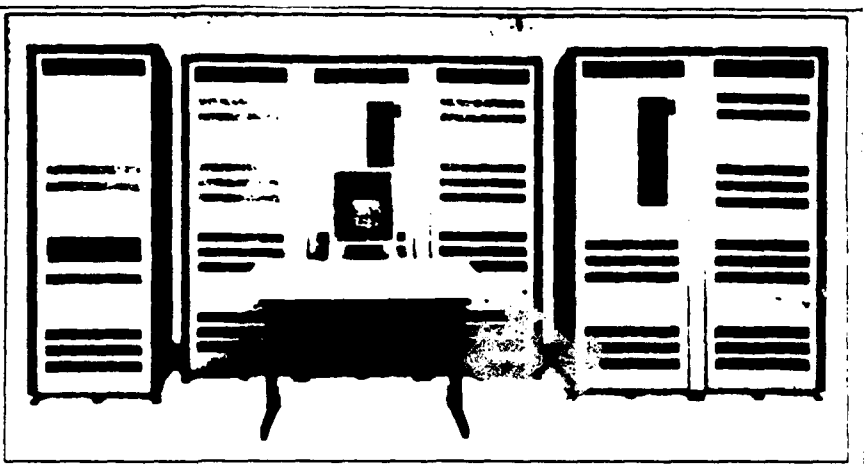
The three agreements are as follows:

- An Industry Marketing Assistance Program agreement with Rational of Mountain View, Calif., a company that specializes in Ada development tools. This will allow IBM to market the R1000 Development System, which consists of hardware and software tools needed by teams of 10-30 or more programmers. The software support system includes an Ada compiler, which allows programmers to write in Ada and to have it automatically translated into machine-level instructions.
- A Marketing Assistance Program with Alysys, Inc., of France, which allows IBM to market Alysys's Ada compiler for IBM 370-class computers running on an IBM operating system.
- An Industry Marketing Assistance Program agreement with CRI, Inc., of Santa Clara, Calif., that allows IBM to market CRI's Relate/DB relational database for IBM's 9370 minicomputer and 4281 mainframe. The database works with an IBM operating system.

Last year the Defense Dept. mandated the use of Ada for all new weapon system software development as well as for more general types of computing. The Pentagon wants to standardize on this language, rather than continue using a variety of incompatible languages, such as Pascal, Jovial, Cobol and Fortran.

Industry officials say that IBM, the world's largest computer manufacturer, has been late in recognizing the importance of Ada. Recent IBM moves are an attempt to catch up, these officials say.

"I think IBM came late in the game. Now that they are in, they are starting to be more aggressive," Kevin J. Dyer, who is project manager for Adanet, a West Virginia-based development network for Ada software applications, said. "They had their toes in the water. Now they're in up to their knees."



Rational sells three models of the R1000 Development System. Model 10 (left) can support up to 10 programmers using multiple workstations. Model 20 (right) supports up to 20 programmers. The Model 40 system (center) supports up to 40 programmers.

Whether the Defense Dept. and aerospace contractors engaged in developing billions of dollars worth of computer programs for weapon systems will decide to use IBM hardware may depend on the availability of associated Ada tools that make it possible to write hundreds of thousands and even millions of lines of software code in Ada.

The use of computer hardware and software tools to support programmers working in Ada is called computer-aided software engineering (CASE). Today, the CASE tools for Ada development, including hardware and software, are just being developed to the point needed to accomplish the massive programming tasks found in large defense systems. IBM believes that the CASE market will take off and it wants to be in a position to capitalize on this. IBM also wants to sell its computers for use as embedded systems dependent on Ada for software development.

Ada is supposed to make it easier to maintain software programs once they are in the field, an area that accounts for as much as 80% of the Pentagon's software costs. Ada is also supposed to allow programmers to create software modules that could be placed in a library and reused later, since a section of code that performs one function might fulfill the same function on the next weapon system project.

Digital Equipment Corp. meanwhile is well positioned in the Ada environment due to its early commitment to the language. Digital has one of the earliest and most successful Ada compilers.

The IBM agreement with Rational

IBM Division Wins Bid to Develop Mission Computer for Navy's ATA

BOSTON

The McDonnell Douglas/General Dynamics team for the U. S. Navy A-12 advanced tactical aircraft program has selected IBM's Federal Systems Div. to develop the aircraft's mission computer.

The dollar value of the contract awarded March 14 has not been disclosed. The ATA project is often cited as one prominent weapon system program using the Ada high-level programming language for software development. IBM will supply the hardware while McDonnell Douglas will program the software in Ada. The hardware will incorporate Very High-Speed Integrated Circuit (VHSIC) technology, according to IBM officials.

The ATA mission computer system also will incorporate common avionics modules as part of the Defense Dept.'s initiative to promote standardization among the armed services.

IBM's Federal Systems Div.'s Owego, N. Y., plant will perform the development work. A Navy official said dual supply sources will be used on the 30 highest value ATA subsystems, and a McDonnell Douglas official said a second source will be chosen for the mission computer. □

raised some eyebrows in the industry because the R1000 Development System includes Rational-developed hardware—not IBM hardware. Ada industry officials considered it unusual for IBM to help a company market non-IBM computer hardware. But the R1000 would work in tandem with IBM computers to perform software development functions.

CONTRACTS LOST

Another reason for IBM's recent moves in Ada is that it had lost out on some key U.S. government contracts, including the software support environment for the NASA space station. This contract, won last June by Lockheed using a Rational system for development, is valued at $140 million. It is expected that the task will require 10 million lines of Ada code. Lockheed has owned 10% of Rational since 1985 and has one of its top executives on Rational's board of directors.

Rational has grown to prominence in the Ada market since its formation as a startup in 1979 by two ex-Air Force officers who left the service after working on computer programs designed to track satellites.

Today, customers for Rational's R1000 Development System come predominantly from the aerospace industry, and include Lockheed, Martin Marietta, TRW, Hughes, Rockwell, General Electric, Westinghouse and the Air Force—just the sort of customers IBM wants to do more business with in an Ada environment.

Rational also has business overseas. The R1000 system is being used by Philips Elektronikindustrier to write Ada code for an electronic command and control system to be used on four new Goteborg-type coastal corvettes for the Royal Swedish Navy. Rational officials say the Philips programming work in Ada is the largest single Ada project in full-scale development in the world, with 140 software engineers working on 1 million lines of Ada code.

The R1000 has models ranging in price from $295,000 for one serving up to 12 programmers and $795,000 for one serving up to 40 programmers. Assuming 40 software engineers use one, a productivity improvement of 5-10% is needed to break even on the cost of a system. Rational claims its system can improve productivity from 25-300%, resulting in savings of up to $28 million on a project involving 800,000 source lines of Ada code.

The Rational R1000 is useful for software engineering of 100,000 to 1 million lines or more of Ada code. Lockheed Missiles and Space Co.'s Astronautics Div. will use Rational systems, for example, to automate the design and development of software for the NASA Space Station. The same arm of Lockheed and GM Hughes Electronics also will use Rational systems to develop software systems for two prototypes of the advanced tactical fighter. Martin Marietta will also use an R1000 on its National Test Bed contract with the Strategic Defense Initiative Organization.

IBM's Federal Systems Div., which bids on Defense Dept. contracts, already owns six R1000 systems for use in Ada-related programming.

IBM officials cite the activities of the Federal Systems Div. on a number of key Defense Dept. contracts as indication of the company's strong commitment to Ada. One of the largest of these contracts is the Army's Worldwide Military Command and Control System (WWMCCS). Another program won March 14 is the mission computer for the Navy's advanced tactical aircraft.

Ada industry officials considered it unusual for IBM to help a company market non-IBM hardware

However, others in the Ada community point out that the larger IBM marketing organization outside of the Federal Systems Div. has not embraced Ada yet. The agreement with Rational is expected to make Ada tools more available to this part of the organization to support sales of IBM computers to aerospace companies and even commercial users. Telephone companies, for example, are expected to find Ada helpful for software development and IBM is now in a better position to serve these needs as they emerge.

The agreement with Alsys is another example of IBM aligning itself with some of the leading companies in the Ada community. Alsys was founded in 1980 by Dr. Jean D. Ichbiah, formerly with Cii Honeywell Bull in France. He led the team that won the worldwide competition over three other groups to develop the Ada language for the U.S. Defense Dept. Alsys lists among its current customers Ball Aerospace Systems, Boeing, McDonnell Douglas, Mitre Corp., Martin Marietta, NASA, Singer-Link, TRW, Allied Bendix Aerospace, Hughes Aircraft, Lockheed Missiles & Space and Raytheon.

CRI, Inc., the third Ada vendor with which IBM is aligning itself, is also a long-term player in the Ada market. CRI customer McDonnell Douglas encouraged IBM to make CRI's Relate/DB database available on the IBM 9370. McDonnell Douglas is using the CRI database as part of a system to track pilot training data for T-45 aircraft under a U.S. Navy contract. CRI's Relate/DB is being used by NASA, Lockheed and Stanford University on a project to reduce paperwork requirements for the maintenance of the space shuttle thermal protection system.

IBM's long-range plans are to provide Ada software for all of its computers. The company will expand the number of Ada vendors involved in marketing agreements. As one IBM marketing official who focuses on the CASE market said, "Our product plans will unfold."

Estimate of Ada Market Potential In U.S. Aerospace Industry

1986	1987	1988	1989	1990	1991	1992
THOUSANDS OF PEOPLE DEVELOPING SOFTWARE						
100.0	112.0	125.0	140.0	157.0	176.0	197.0
PERCENT WORKING ON ADA PROGRAMS						
3%	6%	12%	20%	30%	40%	50%
THOUSANDS OF PEOPLE DEVELOPING SOFTWARE IN ADA						
3.0	6.7	15.0	28.0	47.1	70.4	98.5
INVESTMENT PER PERSON WORKING ON ADA PROGRAMS *(In Thousands)*						
$10.0	$10.0	$10.0	$10.0	$10.0	$10.0	$10.0
ADA SUPPORT MARKETING OPPORTUNITY *(In Millions)*						
$30.0	$67.0	$150.0	$280.0	$471.0	$704.0	$985.0

Assumption: No. of software developers grows 12% per year — *Source: Rational, Mountain View, Calif.*

IBM and other companies are interested in supporting Ada programmers in aerospace because the number of software engineers programming in Ada is expected to increase more than tenfold by 1992, when half of all software engineers in the industry are expected to be working in Ada.

Next-Generation Defense Programs Will Increase Use of Ada Language

DAVID HUGHES/BOSTON

Ada, the high-level computer programming language developed by the Defense Dept., is being accepted for use in aerospace and defense programs even though it is trying to live down a record tarnished by exaggerated performance claims.

"Ada is beset with too much hyperbole for its own good," then-Under Secretary of the Army James R. Ambrose said at the annual Ada Expo and Special Interest Group convention in Boston. Ambrose, who retired last month but still serves as a consultant to Secretary of the Army John O. Marsh, Jr., cautioned Ada vendors to be conservative in statements about the capabilities of the new language because exaggerated claims do more harm than good.

These candid remarks from a high-ranking representative of the U.S. Army—which has been one of the leading advocates for using Ada as the standard computer language for the Defense Dept.—did not mean that the Army has abandoned Ada, according to an Army official. The official compared Ambrose's remarks to a pep talk a coach might give to his team during halftime.

"I am not impressed, have not been, and I remain willing to be impressed, with quantitative measurements of the superiority or productivity or the virtues of anything such as Ada," Ambrose said while encouraging the 2,000 attendees to avoid claiming that the language can be all things to all people. "What I have seen thus far has been largely rhetoric."

The Defense Dept. began developing Ada in the late 1970s as a solution to the problems created in the armed services by the use of multiple computer languages with poor documentation. This made it difficult for anyone to fix problems with a software program once it was fielded or to port a software program from one computer to another.

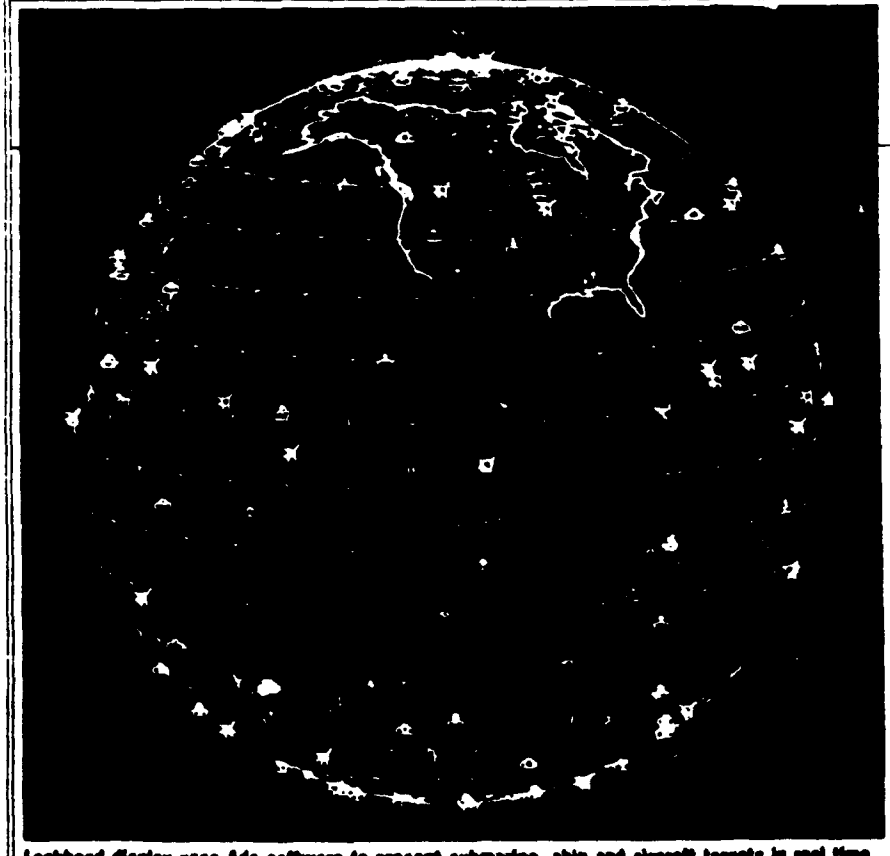
EARLY SHORTFALLS

Ada was to have solved these and other problems, and it may yet. However, the use of Ada in programs in the early 1980s, when compilers were either nonexistent or of poor quality, caused some program managers in the Pentagon to write Ada off as a failure. A compiler is a key software development tool that allows a programmer to write in a high-level language such as Ada and have it automatically translated to machine-level instructions.

"Ada's only handicap is that it claimed to be a savior for all applications," David E. Quigley, the Vax Ada product manager for Digital Equipment Corp., said. Digital is one supplier of computer hardware that dedicated some of its top programming talent to developing Ada compilers from the beginning. Soon the company will introduce its third compiler. It also offers customers a variety of Ada tools. Quigley said this may well be the year the Ada market takes off.

Now that there are many high-quality compilers available that are validated by the Defense Dept.'s stringent tests, things are changing. The Ada show here offered some signs of these improvements. Attendance here was triple that of the show three years ago in Minneapolis. Also, the attendees included representatives of many new companies and higher ranking officials of program offices in the Pentagon.

The convention was sponsored in part by Software Valley Corp., a nonprofit or-



Lockheed display uses Ada software to present submarine, ship and aircraft targets in real time.

Lockheed Using Ada In Airborne Display

BOSTON

Lockheed is developing an advanced display for antisubmarine warfare and airborne warning and control system (AWACS) applications using the Ada programming language for development.

The display, which can present airborne, surface and subsurface targets on a global scale using the CIA's World Data Base II mapping system, can also zoom in so the operator can view an area a few miles across. The display is running on a Digital Vax 11875 and will be ported over to a Motorola 68030 microprocessor.

Lockheed's Advanced Avionics System Center in the company's Aeronautical Systems Co. is using the display development as part of its own company-funded effort to improve sensors, enhance crew coordination on multiplace aircraft, and refine the interface between man and machine. The Ada program consists of about 1,000-1,500 lines of code. The project has been under development for 15 months and the display will be tested on board an aircraft late next year. The team will use its experience to provide lessons learned on Ada to the Lockheed organization.

ganization founded by U S Senate Majority Leader Robert C Byrd (D-W Va) to promote high technology industry in West Virginia Software Valley has held other conferences to promote Ada and to attract related companies to the state

Furthermore, the use of Ada is spreading into the commercial marketplace, particularly in Europe, where telecommunications firms are using it. The European defense ministries and NATO are also advocating the use of Ada, but are not moving as quickly in this area as the Pentagon

KEY TO ACCEPTANCE

Industry officials say the key to the widespread acceptance of Ada will be its use in many of the U. S. Defense Dept.'s new, high-profile weapon system programs, including the advanced tactical fighter and the advanced tactical aircraft, and other government programs.

The Pentagon mandated on Mar. 31, 1987, that, effective immediately, all software development for new weapon systems be performed in Ada as the single, common, high-order programming language. Deputy Defense Secretary William Taft, 4th, broadened that directive in April, 1987 by specifying that Ada must be used on all Defense Dept. computer resources, with a few other languages permitted in certain cases. These reaffirmations of the Defense Dept.'s commitment to Ada are expected to curb the number of waivers requested and the number granted to develop programs in a language other than Ada.

Currently there are 84 Air Force, 51 Army and 24 Navy programs using Ada for software development (see chart for examples). NASA also is using Ada for the space station and the Federal Aviation Administration is using it for the Advanced Automation System.

COMMERCIAL USES

Ada has spread from use in the military to important commercial avionics programs. The Collins division of Rockwell International, for example, developed Ada programming capability to address military markets, then used it to write software in Ada for the Boeing 747-400 and Beech Starship avionics suites. Boeing also specified Ada for the 7J7 project, which is now on hold.

The emerging Ada support industry, which has spawned many start-up companies, seems about to enjoy an upsurge in business following many lean years, when the use of Ada by the Defense Dept. was more the exception than the rule. Industry officials estimate that the rapidly growing Defense Dept. market for Ada software passed the $1.8-billion mark last year.

One difficulty in determining the full scope of Ada's use by the Defense Dept. is that it is being used on many classified programs that cannot be identified, according to industry officials. Also, the Defense Dept. keeps no statistics on how much it is spending on software development in Ada, or any other language, by program or even as an overall percentage of the total defense budget.

The real test for the language is expected to be how it performs in weapon system programs that are employing it for full-scale development. How the language performs in key programs like the ATF and the ATA will go a long way in determining its ultimate success, according to industry officials.

One worry expressed by many at the Ada conference here is that any budget cuts that trim or eliminate new weapon system programs in which Ada is being used will slow the acceptance of the language by the defense community and reduce the demand for Ada software support products □

Major Software Applications Coded in Ada

Program	Status	Lines of Code	Type of Software	Company
AIR FORCE				
Advanced Tactical Fighter	Planned	7,000,000	Embedded	
Milstar	Development	500,000	Command & Control	Lockheed
Common Ada Missile Packages	Development	30,000	Embedded	McDonnell
Small ICBM	Development	2,000	Embedded	Rockwell
ARMY				
Intermediate Forward Test Equipment	Development	500,000	Support	Grumman
Mobile Automated Field Instrumentation System	Development	70,000	Simulation	
Maneuver Control System	Complete	34,000	Command & Control	Ford Aero
Regency Net	Development	110,000	Support	
Advanced Field Artillery Tactical Data System	Development	790,000	Command & Control	Magnavox
Army Worldwide Information System	Development	8,000,000	Command & Control	TRW
NAVY				
F-4J Weapon System Trainer	Complete	150,000	Embedded	SAI
Advanced Tactical Aircraft	Planned	NA	Embedded	
SDI				
Battle Management and C³ Portions	Planned	10,000,000	Command & Control	
NASA				
Space Station Software Support Environment	Planned	750,000	Embedded	Lockheed
FAA				
Advanced Automation System	Planned	NA	Support	IBM or GMH
COMMERCIAL				
Avionics for Beech Starship	Complete	82,000	Embedded	Rockwell*
Sensor Station Simulation Facility	Development	25,000	Simulation	Allied Signal
7J7 Aircraft Avionics	Development	NA	Embedded	Boeing
OTHER				
Canadian Air Traffic Control	Development	1,000,000	Support	
F-20 Avionics	Complete	NA	Embedded	Northrop

The future of Ada will most likely be determined by its performance on high-profile weapon system programs like the advanced tactical fighter. This chart only lists the largest programs using Ada. There are many more using it in the Defense Dept.

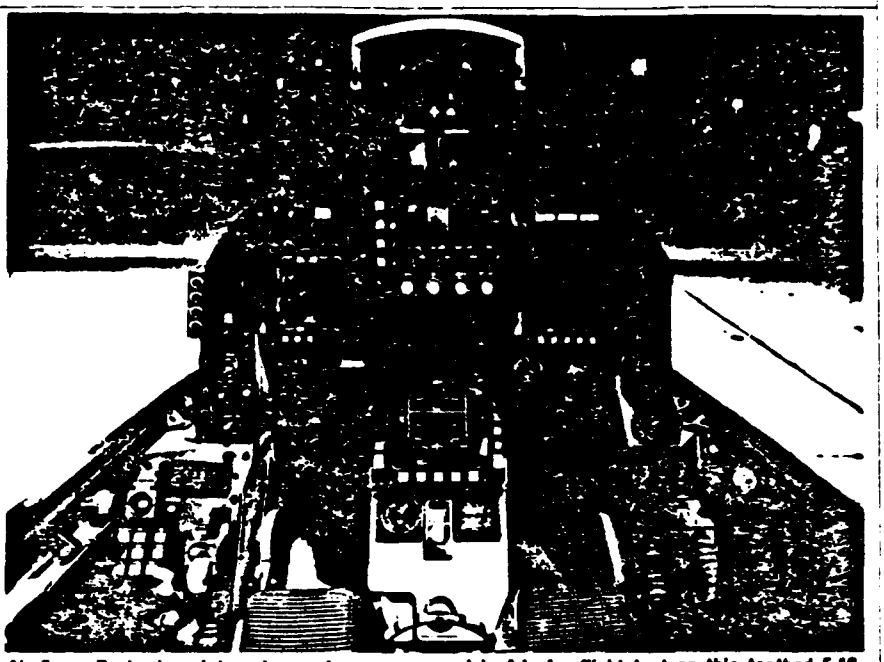
*Rockwell's Collins General Aviation Div NA Not Available Source Salomon Brothers Inc

General Dynamics Explores Ada In Extensive Flight Test Program

BOSTON

Ada, the high-level computer programming language, has some limitations running real-time embedded computer systems, as General Dynamics engineers have learned in developing a program in Ada for a series of flights on an F-16 testbed aircraft.

The U.S. Air Force Technology Integrator program uses an F-16 to flight test high-risk future technologies that the Air Force wants to evaluate. One operational flight program begun in 1986 involves 21,000 lines of source code, of which 88% is written in Ada, the language developed by the Defense Dept. starting in the mid-1970s. The rest of the program is written in assembly language specific to the Zilog Z8002 microprocessor. The Z8002 was used to run a communication, navigation and identification friend or foe (IFF) system.



Air Force Technology Integrator system programmed in Ada for flight test on this testbed F-16 includes a keypad and display (left console) and an electronics unit (not shown).

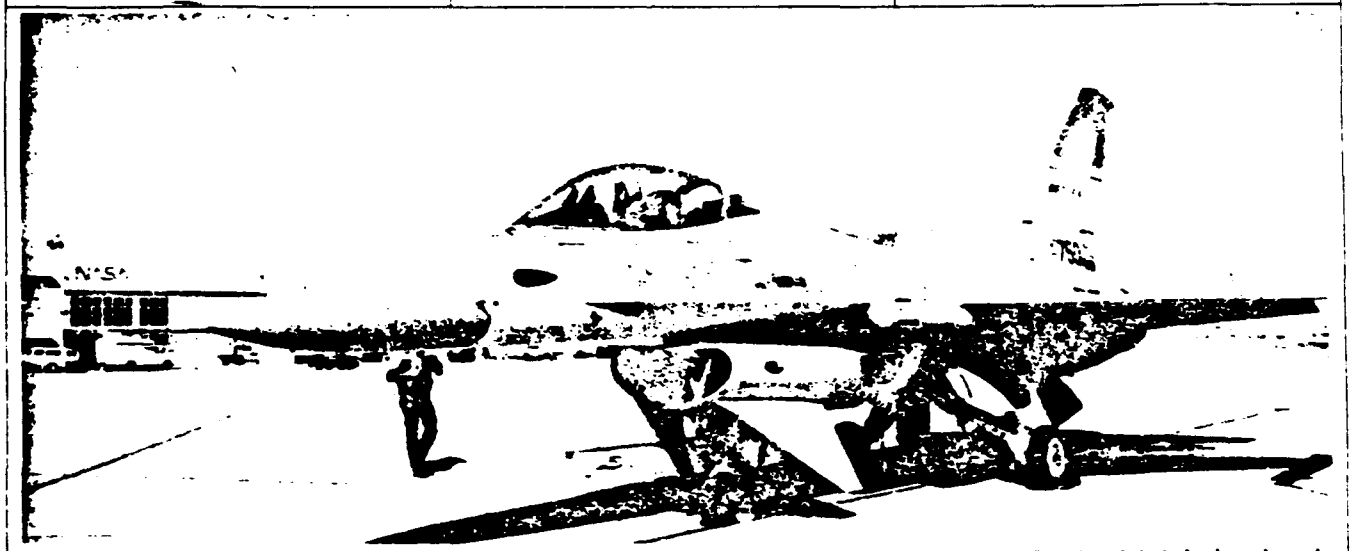
HIGH EXPECTATIONS

Though limited in scope, this use of Ada is one of the first in an embedded system that has actually flown. This is true despite the fact that the Defense Dept. is staking the future of many key weapon system programs on the use of Ada, which has yet to prove itself widely in actual use in embedded computer processors, such as the 1750 used on board the F-16. One other flight test program on an F-15 in 1984 also involved a limited use of Ada in an embedded processor.

The F-16 system using Ada is called the Data Entry/Cockpit Interface Set (DE/CIS). It was originally programmed in the Jovial language. The system consists of an integrated control panel for entering frequencies for communications and navigation radios as well as IFF, a display to show the pilot the entries made. The system also has some control stick mounted switches.

Unlike a similar navigation and communications system, which combines control of all radios and IFF in one control unit on production F-16s, the one programmed in Ada also incorporates a voice recognition feature. This allows pilots to use voice commands as well as the numeric keyboard found on current F-16s to enter frequencies and to change the aircraft's course. The voice command function is able to recognize and act upon numbers and up to 40 distinct words.

J. P. Sarkar, avionics software leader for General Dynamics on its National Aero-Space Plane (NASP) study, previously led the company's Ada efforts on the navigation and communications system for the F-16. He said the Ada-devel-



Air Force Technology Integrator F-16 returned to NASA's Ames-Dryden Flight Research Facility at Edwards AFB last month after modifications were completed by General Dynamics. The aircraft is to begin a close air support research program this spring (AWAST Nov. 2, 1987, p. 78).

oped package performed well in more than 20 flight tests. Even though extra crosschecks built into the Ada language prevent errors from cropping up when a program is run, these and other features require more computer code to be written, which slows down the speed of execution. However, Sarkar says Very High-Speed Integrated Circuits (VHSICs) may reduce the time penalties involved.

Also, Ada compilers, which are essential tools for programming with the high-level language, need to be refined further. Most compilers do not yet incorporate a number of optional features in military standards established for Ada, which are essential for developing programs to handle real-time requirements, according to General Dynamics engineers.

Programs written to run on embedded computer systems face two constraints. First, the programs cannot be too extensive because they must run on compact processors that fit in the limited space on board aircraft. Secondly, the programs must be able to execute tasks at high speed to meet the requirements of a real-time environment. On the F-16, for example, computers dealing with the physics of flight must be able to perform rapid calculations. The aircraft can roll at a rate of 360 deg./sec. Computers following this motion only update calculations at 50 times/sec., but this is quick enough to present the motion to the pilot as if it were smooth and continuous.

Sarkar and other General Dynamics engineers associated with the Ada test program said many aerospace engineers are concluding that Ada is not suited to "hard" real-time applications for aircraft command and control software. Hard real time is where such software subroutines as analog to digital conversions must occur in less than 10 microseconds, although Ada is suited to "soft" real-time applications where the less stringent requirement for subroutines is more like a few milliseconds. The General Dynamic's experience with the Ada test program on the F-16 reinforces this view. However, efforts are under way to modify the Ada language to make it more suitable for "hard" real-time applications, and Sarkar expects these efforts to be successful.

Sarkar said the true benefit of Ada will be seen in the maintenance phase. The requirements of design updates on embedded computer systems like those found in the F-16, which has 12-15 per aircraft, requires constant changes to be made to the programming. These changes will be much easier to make if the original program is developed using Ada rather than some other language.

The original F-16 flight testing of the DE/CIS system ended late last year, but flight testing is resuming in a follow-on Air Force contract awarded for the AFTI program. □

FILTER CENTER

VALUE OF THE VOLT IN THE U. S. will change by about nine parts per million on Jan 1, 1990, substantially affecting calibration of sensitive aerospace electronic test equipment. The change, to align the U. S. volt with the world standard, will affect as many as 100,000 precision instruments, such as digital multimeters in automatic test equipment, used for electronic production, test and maintenance. Military, and particularly black programs, are apt to be most affected. National Bureau of Standards is concerned about the problems and economic impact if industry has not planned adequately for extra workload of the phase-in, and will publish a pamphlet with suggestions in the fall. Contact: Norman Belecki, (301) 975-4223.

AIR FORCE AERONAUTICAL SYSTEMS DIV. is sponsoring development of hardened solar space power systems. The systems are to be hardened against natural and man-made threats, have a 10-year life, 20% conversion efficiency during periods of sunlight, and a minimum weight of 10 watts per kilogram.

ARMY HAS SELECTED THE THOMSON-CSF/HAMILTON STANDARD team to supply 56 head-up displays to the Army for the Bell OH-58 C helicopters. The HUDs are the first award in the Army's Stinger program to provide the helicopters with an air-to-air missile, gun and rocket capability, according to Thomson-CSF. First production deliveries are scheduled for 1989.

I. C. SIMS HAS DEVELOPED A RECONFIGURABLE COCKPIT system for development and training with two or three dimensional displays that simulate television, Flir, moving maps and out-the-window perspective views. The real-time computer-based system can be configured to the cockpit of a specific aircraft quickly and flown through a simulated tactical environment that includes targets and enemy threats. The low-cost, flexible system will perform air-to-air, air-to-ground and tactical support missions.

AIR FORCE WANTS TO DEVELOP A HIGH-POWER MICROWAVE (HPM) SOURCE generated by broadband video pulses switched by bulk avalanched solid state laser triggered switches. Current HPM generators include magnetrons, vircators, klystrons, gyrotrons and free-electron lasers. The first phase of the planned 26-month effort will develop a single element power module. The second phase will develop a 5×5 antenna array of power modules. High-power microwaves have potential as future tactical weapons against both electronics and personnel.

ROYAL NORWEGIAN AIR FORCE will install Collins Global Positioning System (GPS) receivers on its fleet of Falcon 20 aircraft, which will be used for airfield certification and as VIP transports. The Rockwell division is also building military GPS receivers for the U. S. Defense Dept. with a potential value of $454 million.

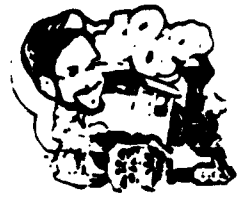
U. S. AIR FORCE has exercised an option to purchase 122 additional terrain-following radar systems from Texas Instruments for F-111 aircraft. Deliveries under the contract, the second of seven anticipated lots, will be made by the end of December. The contract, worth $88.5 million, is part of a total program for 450 radars plus spares that will run through 1991. Texas Instruments completed first-lot deliveries of 68 radars at the end of 1987.

ARMA DEI CARABINIERI, the Italian Army's Judiciary Police Branch, has awarded Datapoint Corp. a $2.7-million contract for local area network-based computer systems. The contract calls for 58 Datapoint processors and 10 terminals to be used to configure Attached Resource Computer (ARCNET) local area networks at Carabinieri installations throughout Italy. Each network site will use Datapoint's Resource Management System (RMS) network-oriented operating environment.

AIR FORCE HAS AWARDED SINGER a $77-million Joint Tactical Information Distribution System contract. Singer will provide class two terminals for the Air Force E-8A Joint Surveillance Target Attack Radar System aircraft and for the Navy's F-14 and E-2C aircraft. JTIDS is designed to use spread spectrum and fast frequency hopping techniques to transmit digitized, jam resistant, secure voice and data and to provide precise relative navigation to the U. S. military and NATO.

Do-While Jones

Ada Info



Why the Navy Doesn't Use Ada

Insiders have known for some time that the Navy has fallen behind the other services when it comes to Ada. The Navy position was obvious even to outsiders at the Sixth National Conference on Ada Technology, March 14–17. There the Air Force and Army proudly talked of their major Ada projects, especially CAMP (Common Ada Missile Packages) and RAPID (Reusable Ada Packages for Information systems Development). The Navy spent most of the conference avoiding questions. How did the Navy get so far behind?

The three services all recognized the need to develop Ada compilers and support environments years ago. The Army awarded a contract for the development of the Ada Language System (ALS), and the Air Force funded an Ada environment called AIE. The Navy saw this as a duplication of effort, and wisely decided not to develop a third environment. They chose instead to monitor the development of the ALS and AIE, with the intention of adapting the better of the two for Navy use.

The Air Force quickly decided to cancel the AIE development contract. They told vendors they would use any validated Ada compiler targeted to the Air Force standard 1750A computer architecture. Today there are validated Ada compilers for 1750A target computers, and the Air Force Ada effort is going strong.

Since AIE dropped out of the race early, the ALS won by default. The Navy announced its intention to adapt the Army's ALS for Navy use by replacing the Army compiler with a compiler targeted to the Navy standard computers. The Navy version of ALS was to be called ALS/N.

The ALS went into Beta test. It was so big it would fit on a VAX 780 only if the VAX was dedicated to a single user. It had compilation speeds reminiscent of the old NYU Ada-Ed translator of years ago. It generated code that didn't have a ghost of a chance of running in real-time embedded computer applications. If that wasn't bad enough, Motorola and Intel were selling much better microprocessors than the Army standard microprocessors. The Army was stuck with a terrible Ada environment for a family of state-of-the-past computers.

The Army came up with a brilliant solution. With great fanfare they announced the successful completion of the ALS project. They generously placed the technology (which had cost so much money to develop) in the public domain so everyone could benefit from it. At the same time they casually mentioned they would be using commercial Ada environments for rugged versions of commercial microprocessors, instead of the ALS. The Army Ada efforts are doing quite well now.

The Air Force and Army generals know there are times when you have to lose a battle to win the war. The failures of AIE and ALS were stepping stones to victory. The Navy has a different tradition. "The captain goes down with the ship."

Although studies done in Navy labs show that the ALS/N is inadequate for Navy applications, the Navy insists on a policy of requiring the ALS/N for all Navy Ada applications. In other words, the Navy has given the message to compiler vendors, "Don't bother designing Ada compilers for Navy standard computers, because we won't buy them." That's why there are no compilers for Navy computers today.

After boldly stating that the Navy would use ALS/N for Ada development in 1990, Admiral Quast hastily left the Conference before anyone could ask him any embarrassing questions. I suspect he knew someone would ask, "Hasn't ALS/N been cut from the budget?" (It has since received $4 million that was taken from other projects.) Three times I heard people ask, "How can a software contractor bid on a Navy project when the Navy insists on requiring the contractor to use ALS/N?" They never got an authoritative answer because the Navy brass wasn't there to answer the question.

I tried to give Commander Barber a chance to redeem the Navy's tarnished image. I publicly asked this carefully worded question: "If any vendor validates an Ada compiler targeted to a Navy standard computer, and that compiler is clearly superior to the ALS/N, will the Navy allow it to be used?" I hoped he would say, "Of course. The Navy always uses the best technology available." His answer was, "Maybe we will think about it." I suppose that's better than a flat "No."

So how are software contractors going to bid on Navy contracts? Well, if I was going to bid on a Navy project, I would bid it with the intention of using my favorite Ada compiler on my favorite host computer. In the years it takes to complete the project, the ALS/N might become a useful product, or the Navy might adopt a reasonable policy concerning the use of a third party Ada compiler targeted to the Navy computer. If this happens, I would recompile the Ada source code (which I developed on my favorite compiler) using the Navy cross compiler. Of course I would be careful to isolate the machine dependent code to a few IO packages (but one should do that anyway). The bodies of these packages would have to be rewritten before recompiling, but the rest of the source code would remain unchanged. Not much would be lost switching from my favorite environment to the Navy system.

We have to take into account the fact that the ALS/N may never be of any use, and independent vendors might not risk developing a compiler for a market that might not exist. Then the fall back plan is to translate the Ada design to CMS-2 for the Navy computer, or get a waiver to use another computer of my choice, which has a good Ada compiler. That wouldn't satisfy the original contract, but all contractors know to get into a position where they can propose a contract modification that the government can't refuse.

What should compiler vendors do? That depends on how they feel about taking chances. If one vendor goes out on a limb and develops a compiler for the Navy standard computers, and the Navy decides the logistics of maintaining spare parts for a limited number of computers outweighs a

Journal of Pascal, Ada, & Modula-2, Vol. 7, No. 5, pp. 40–41 (1988)

CCC 0735-1232/88/050040-02$04.00

romantic devotion to ALS/N, that vendor could corner the Navy market. On the other hand, there may never be a Navy standard computer market, and all the development cost will be lost.

Before you all start to write me letters telling me that you know of a Navy Ada project or two, let me assure you that I know about some, too. I'm not saying there aren't any Navy Ada programs. I'm just saying there are noticeably fewer Navy programs than the other services. Perhaps the key word here is "noticeably." The Navy programs I know of are grass roots programs undertaken by Navy laboratory engineers with little or no backing from Washington. Perhaps these programs are intentionally taking a low profile to avoid the risk of their sponsor telling them not to take such a risky approach and stick with an established language.

The official Navy position is that the Navy supports Ada, but actions speak louder than words. If the Navy really supports Ada, why was the Navy Ada implementation plan so late. (I use the past tense because I hope it will be done by the time this is published. Perhaps it still isn't done.)

It seems to me that sooner or later the Navy has to get with the program and follow in the path of the Air Force or the Army. They either have to open the market to third party vendors (like the Air Force did), or switch to a computer that already has good Ada support (like the Army did). Until they do, they will remain up a creek without a paddle.

---

shipped 737,500 terminals. That number rose 11% in 1986 to 833,000, but despite the jump in terminal sales, dollar volume for the year dropped to $1.42 billion from $1.48 billion in 1985.

According to analysts, IBM has remained a formidable competitor in the terminal market, although in 1985 it slipped one percentage point in market share to the plug-compatible competition, mainly Telex. In 1985, IBM had 57.6% of the 3270 market, followed by Telex with 16.1%, ITT Courier with 7.1%, AT&T-Teletype with 4.7%, Memorex with 3.4%, Lee Data with 3.2%, Harris with 2.7%, and the remaining companies with 5.3%.

Competitors Strike Back

But even in this new era where more terminal shipments mean less dollars, some plug-compatible vendors are holding their own.

Instead of going offshore for margins, Telex Corp., headquartered in Tulsa, Okla., has made a large investment in a fully automated terminal manufacturing facility in Raleigh, N.C., and an automated distribution and repair center in Tulsa.

Last week, Telex introduced its entry into the 3191 market, the Telex 191, which will initially be available only in the U.S. Like the 3191, it is built in a completely automated facility to keep costs down. The 191 attaches to either a Telex or an IBM control unit. It offers an 88-key keyboard or a recently announced 122-key keyboard. An added optional feature is the ability to attach a light pen or a message printer. The 191 lists for the same price as its IBM counterpart, with a 90-day warranty.

According to Mike Bowman, product manager for Telex's 3270 terminals, the company will be competitive with IBM on volume discounts and warranty periods, which run as long as three years.

Other vendors to enter the 3191 market include Beehive International (with an enhanced 3278 offering) and Memorex, both in San Jose. ITT Courier, also in San Jose, is readying its 3191 entry for a March announcement.

"It is an overriding situation. Plug-compatible vendors must have a 3191-compatible product because that is now the entry point for 3270 terminals," Wagner says.

As terminal prices drop, plug-compatible vendors have to look to other segments of the market to make up for slipping 3270 margins.

"Telex has expanded into new markets that complement and expand our 3270 [offerings]," says Pat Reiner, vice president of marketing and product planning for Telex. "For example," she says, "in July of last year, we announced a new series of airline [reservation] terminals. In September we introduced the C078 voice/data terminal, and in November we entered the System 3X market with nine new products."

"THE TERMINAL GIVES IBM ROOM TO CUT PRICES."

Analysts also say that specialized products like the voice/data terminal and airline reservation terminals can still command higher prices and margins for Telex. Companies such as Lee Data, Minneapolis, and Memorex are the only terminal vendors now offering a more diversified product line.

Another area in which Telex is doing well is control units. At the time of the 3191 announcement, IBM also said it would add multiple sessions and windowing capabilities to its control unit, but Big Blue has yet to offer a control unit with these features. However, analysts say IBM is readying products with these features for release sometime in the first half of this year.

Taking Advantage

In the meantime, companies such as Telex are taking advantage of the time by bringing out products that match IBM's direction.

Telex recently introduced its 274 control unit with windowing, which allows 3270 terminals to configure up to four displayable windows of four different host sessions. Data can be copied from one window to another, even if one of the windows is an ASCII host session. The control unit also supports the IBM 3179G graphics support feature, which enables an attached 3179G to use the all-points-addressable (APA) graphics support available from the host computer.

According to Telex's Bowman, when his company incorporates announced features from IBM, as in the case of the control unit, the vendor must adhere to strict compatibility.

"These vendors don't want to jeopardize their plug compatibility. They don't want to extend themselves too far away from IBM," says IDC's Goldman.

In the short term, IBM is taking a very aggressive posture in the 3270 business as it attacks its competition's strongest selling point—price. But, until IBM can deliver all it has promised in the area of control units, there is plenty of room in the market for the plug-compatible vendors. ■

LANGUAGES

What the Countess Didn't Count On

Ada continues its slow march to the dp world, but can it shake off the DOD image?

BY EDITH D. MYERS

Ada, the Department of Defense-mandated programming language named for Ada Augusta Byron, Countess of Lovelace, is hardly topping the wish lists of corporate MIS managers.

There are those who think it should and those who think it someday will (see "Ada Fans Say Now's the Time," May 15, 1984, p. 38), and there are some things happening that could propel the language into a prominent position in the world of commercial data processing. Foremost in most minds was IBM's seeming endorsement of the language when it leaked news at a SHARE meeting in August that it would be offering Ada compilers and productivity tools for 370 systems running under MVS and VM/CMS. This disclosure eventually materialized as an agreement between IBM and Telesoft, San Diego.

Despite the IBM imprimatur, Ada apparently still has neither extended its identity nor ventured beyond the military/industrial world. One of those reasons, of course, is the entrenchment of languages such as COBOL in data processing departments. Another is the relatively re-

cent incursion of so-called fourth generation languages from a number of vendors. Says Michael Ryer, director of Ada products for Intermetrics Inc., Cambridge, Mass., "There is a great inventory of billions of lines of COBOL code. You can't very well rewrite every single line, and it's hard to put a little bit of Ada on top of a lot of COBOL."

Nevertheless, Ada is moving along and is finding applications. One of the most visible areas where this is becoming true is in the aircraft industry.

Some of the lure for aircraft manufacturers to use Ada, primarily in embedded applications, is from the efforts of a working group of the Systems Architecture and Interfaces subcommittee of the Airline Electronics Engineering Committee (AEEC) of Arinc Inc., Washington, D.C., a nonprofit organization owned by a number of major airlines and manufacturing companies that provides reports and specifications for the avionics industry.

"WE'VE GOT THE BEGINNING OF THE BUILDING OF A WAVE."

Last month, the working group held a second meeting to review what is currently Arinc's proposed paper 613, which provides guidelines for using Ada in avionics design and which could become the Arinc report 613 by midyear, giving it a lot of weight with avionics designers. Paul Prisaznuk, avionics engineer with AEEC, says the instigation for, as well as much guidance in, the work came from Boeing Co., which plans to use Ada in the development of its 7J7 aircraft.

Evaluation at Boeing

Brian Pflug, manager of software engineering for avionics design, Boeing Commercial Airplane Co., says Boeing definitely plans to use Ada in the 7J7 project if "the state of the technology proves such that there will be a cost benefit. We've told all our suppliers [for the 7J7 project] that that is our direction and we're in the middle of a project right now to determine if that is correct."

He says the project involves benchmarking currently available Ada compilers to determine the efficiency of code produced. An earlier, similar evaluation for the prototype stages of the 7J7 project was inconclusive to the point that Boeing then left it up to its suppliers whether or not to use Ada.

"The quality of the code produced by the compilers was not always as efficient as it could be in terms of space or time," says Pflug. "It has nothing to do with the language but rather with the maturity of compilers. Most are fairly new." He says the current project has a targeted production decision date of August of this year.

Pflug says about 100 companies are candidates to be suppliers of avionics equipment for the 7J7 project, with 15 to 20 of them probably destined to be major suppliers. The aircraft is scheduled for first customer delivery in 1992.

On another commercial front, a big push for Ada probably will come from creation of a Commercial Ada Users' Working Group (CAUWG) under the banner of SIGADA (Special Interest Group on ADA) of the Association for Computing Machinery (ACM). Corporate members include GTE, Stamford, Conn.; Lear Siegler Inc., Los Angeles; Boeing Co., Seattle; Advanced Computer Techniques, New York; and CRI Inc., Santa Clara.

The working group first met in November in conjunction with a SIGADA Ada Expo conference in Charleston, W. Va., and now has scheduled a third meeting for March 17 in Washington, D.C.

Dave Dikel, director of Washington, D.C., operations for Addamax Corp., a Champagne, Ill.-based contract software service firm focusing on the Ada market, and chairman of CAUWG, says he was asked to find a commercial Ada users group by the Ada Joint Program Office when an investigation he was conducting into commercial uses of Ada was terminated.

He said his investigation had unearthed the facts that there was "substantial" commercial work being done in Ada, that users tended to be secretive about what they were doing, that there was no single good source of information on Ada, and that there were a lot of "ghost facts" around, a prevalence of bad information.

A Government Image

"I looked for a group but didn't find one," Dikel says. "I approached the users committee of SIGADA and found there was no group focusing on commercial uses and was asked if I wanted to start one." This he did with some financial backing from AJPO. Formation was started in March 1986. "Our aim is to influence development of efficient standards and products," he says, "and to fight the prevalent image that Ada is a government thing, is a DOD thing."

Casper Weinberger: There will be fewer DOS waivers on Ada.

But, he says, "the DOD has a key role in our group. There are a number of key executives in DOD who are determined to get the best technology for their dollars. They are buyers of commercial products for prices with lots of zeros. We welcome their support."

"We've got the beginning of the building of a wave," predicts Paul Fuller, vice president of marketing and sales at CRI. "We'll be getting fallout. Companies like Lockheed, Martin Marietta, and McDonnell Douglas will have to train thousands of people in Ada [for work on defense contracts] so why would they want to write their own in-house systems in C?"

Edward V. Berard, founder and president of EVB Software Engineering Inc., Frederick, Md., notes that "the U.S. is the only place on the globe in which Ada is thought of as being primarily military. In Europe, 85% of all Ada applications are commercial. Japan is a huge commercial consumer of Ada."

Grady Booch, director of software engineering for Rational, Mountain View, Calif., which provides software development technologies based on Ada, says language is the least important aspect of the Ada movement, but "people relate to a language." The big thing, he believes, is software engineering, a discipline he thinks is scarce in the U.S., which accounts, in part,

for the more rapid spread of Ada elsewhere. "The Europeans, as a whole, take software a lot more seriously than we do. They don't have the money to waste that we have."

Booch says two past barriers—the lack of people well-trained in Ada and the lack of good compilers—have been lowered. "We are seeing Ada taught in the universities now and there are a lot of good compilers out there," he says.

One of the first companies to come out with Ada compilers was Telesoft, and the San Diego company's agreement with IBM for both

"IT'S HARD TO PUT A LITTLE BIT OF ADA ON TOP OF A LOT OF COBOL."

compilers and programming tools is considered to have much significance by both the company and its competitors. Fuller says IBM's association with Ada will promote the language to the commercial world.

"If IBM didn't tout it [Ada], it would have the same luck as Pascal. Why is C not more popular? Because of IBM's reluctance to support it," sats Fuller.

He also believes Digital Equipment Corp.'s heavy involvement with Ada will help advance the language in commercial markets. "We have the two biggest commercial [computer] companies in it. Everyone else will want in too." DEC offers a range of Ada products, both hardware and software.

Telesoft, too, believes the maturation of the market led IBM to come knocking. "We have been working with IBM for three or four years," says Bruce Sherman, Telesoft director of marketing. "Until now, it was with the Federal Systems Division on specific government programs. As the Ada market began maturing over the last few years, the commercial side of IBM decided to talk to us."

Telesoft also has an technology exchange agreement with Prime Computer, Natick, Mass. Prime isn't offering an Ada product now, nor would product manager Wolf Metzner say when the company would. He did say that he sees a lot of potential for Ada in large systems, "which Prime is getting into with its high-end machines." He notes that Prime does half of its business outside the U.S., "where Ada interest is high."

There are other encouraging signs, too. Two relational database management systems in Ada have been announced. One was announced more than a year ago by CRI of Santa Clara, and the other, internally called Adaplex, is under development by Computer Corporation of America, Cambridge, Mass. CCA's product is due for beta site installation late this year.

Two other producers of relational database management systems, Relational Technology Inc., Alameda, Calif., and Oracle Corp., Belmont, Calif., are offering Ada hooks to their software, primarily to get their feet into the government market but with an eye toward future commercial users.

Says Toby Younis, manager of technical support, federal operations for Oracle, "Ada has a great deal of potential and, when it blossoms, we [hope to be] leading the pack." ■

BENCHMARKS

Buys Graphics Firm

Bolt Beranek & Newman Inc., Cambridge, Mass., has acquired Delta Graphics Inc., a Bellevue, Wash.-based developer of computer image generation systems for simulation and animation applications. The acquisition, in the form of a stock deal, is valued at $16.5 million. Delta Graphics markets its products primarily to government and military agencies. Last summer, Delta Graphics was awarded a $30 million contract to provide its Simnet distributed multiuser training system for military vehicle operators to the Defense Advanced Research Projects Agency and the Army. Bolt will operate Delta as BBN Delta Graphics Inc.

Lay Off at Wang

Weaker than expected sales at Wang Laboratories have prompted a new round of belt-tightening measures, including the elimination of 1,000 jobs and a 6% wage cut for all salaried workers. An estimated $35 million loss for the fiscal second quarter ended Dec. 31, 1986, was also blamed on overly optimistic sales projections, the company says. The size of the loss may be larger depending on results of an asset evaluation under way at press time, a spokesman says. Wang Labs two years ago began releasing workers after sales growth lagged behind expectations. Some 1,600 jobs were eliminated last July and a similar number were let go a year earlier in response to lower sales. The latest cutbacks will pare annual expenses by approximately $50 million and trim the employee roster to about 30,000 people.

Pulling the Plug

IBM has rid itself of two unprofitable businesses in recent weeks. The first was International MarketNet, a two-year-old joint venture with Merrill Lynch & Co. designed to serve the financial services market. Imnet, New York, began shipping its standalone micro-based System 100 in June, but was never able to complete its cornerstone product, the Series/1-based System 500. The companies say the decision was made upon reassessing the financial viability of the venture. The majority of Imnet's 250 employees have been laid off. The second consolidation was that of IBM Instruments Inc., formed in 1980 to sell chemical analysis tools to laboratory scientists. The unit employed 150 people who the company says will be reassigned. IBM has sold its interest in two small firms that manufactured some of the instruments under IBM's label. IBM says it will continue to service products it sold in the last five years.

Leaving the Fold

Unisys Corp., Lockheed Corp., and Allied-Signal Inc. have all announced plans to withdraw from Microelectronics and Computer Technology Corp. by the end of this year. The decision by these three companies to leave the fold brings the number of active members down to 18. According to a spokesperson for the Austin, Texas-based MCC, the withdrawal of these companies is unrelated to the resignation of Adm. Bobby Ray Inman. MCC's first and only chief executive. Allied's sale of its Amphenol division ended the Morristown, N.J., company's interest in the venture. A spokesperson for Lockheed, headquartered in Burbank, Calif., says that the aerospace company's departure had nothing to do with Inman's resignation or the other departures from MCC. Last year, Gould, BMC, and Mostek all left MCC, but the research consortium picked up Hewlett-Packard and Westinghouse. ■

AdaIC

Ada Information Clearinghouse
sponsored by the Ada Joint Program Office

Ada
The International Language
for Software Engineering

COMMERCIAL APPLICATIONS IN Ada

Reprinted with permission of the authors,
Ann S. Eustice and Barry Lynch

When the U.S. Department of Defense (DoD) offered to subsidize the creation of a new language for embedded systems in 1979, it was searching for a solution to the armed services' software problems. Their embedded computer systems that controlled airplanes, submarines, etc., were written in dialects, which required unique compilers and tools. Because each piece of software had a vocabulary specific to it, the DoD's laboratories could not easily and inexpensively alter software as their needs changed, or port it to new hardware platforms, or depend on the result.

The DoD's solution was to establish a competition for the creation of a powerful language that would embody modern software engineering techniques. After the department chose what it considered to be the best language, it mandated that all new embedded systems be written in MIL-STD-1815A, or Ada. It was accepted as an international standard, and other countries' defense departments, such as those of Germany, France, and Australia, also began mandating or introducing Ada to their software laboratories. As a result, a decade ago almost everyone who used Ada did so under a general's orders. Today, the language has infiltrated some commercial sectors which have the same software problems of maintaining and reusing their software, and are looking for the same solution in Ada.

This trend of certain commercial sectors accepting Ada is welcome news to vendors of Ada compilers and development tools. Since the international "outbreak of peace", brought on by the collapse of Warsaw Pact and the demise of the Soviet Union, Ada product vendors have been motivated to explore market niches outside the Pentagon in anticipation of defense budgets shrinking in the '90s.

The commercial sector can expect to receive more telephone calls not only from Ada product vendors but also from Ada programmers and trainers who will be job hunting as defense-related industries lay off staff. Wells Fargo Bank, in San Francisco, Calif., for example, cites the availability of highly experienced Ada programmers as one reason it chose the language for its new investment analysis application.

While a reduced demand for Ada products and developers in the defense market may increase the commercial use of Ada in the future, those in the private sector who use Ada now are reacting to different economic forces. Most developers of commercial applications interviewed for this article mentioned Ada software engineering features, such as packaging and information hiding, as their main reason for choosing the language. Others chose Ada because it was known to facilitate reuse and the development of large applications. Both characteristics increase the software's reliability, which aviation and space agencies and financial services companies mentioned as the deciding factor in using Ada in their new applications.

Ada IN FINANCIAL SERVICES

One of the early high-profile Ada successes was with Reuters financial services in Hauppauge, NY. Reuters is best known as a British international print and photo wire service that transmits real-time information on financial markets and news. Lesser known are its automated currency futures trading and options trading systems for the Chicago Mercantile Exchange.

Reuters' two systems enable trader-to-trader communication and automate the matching of orders. The systems respond within two seconds, handle high loads, and improve the presentation and usefulness of data to clients. Most importantly, the software must transmit and process the data absolutely correctly and on time. Because the application had to carry a heavy load of data accurately and quickly, Reuters ran a greater risk of the software failing due to its complexity. The resultant mistakes could have been extraordinarily costly. According to Alfred H. Scholldorf, manager of Advanced Projects, after studying the language and building a prototype system in 1985, Reuters decided that using Ada was "required for success".

Each system used eight Ada developers to write 250,000 lines of code. Reuters invested 25 staff-years to build each application's 10 major subsystems. They now run on several large VAX machines with multiplex inputs arriving from PCs in New York, Chicago, London, and Tokyo, which are broadcast to other PCs internationally. The applications process billions of

Belgium and Switzerland leading the way. Banksys, an organization responsible for electronic fund transfers in Belgium, also develops systems for use in other countries. The system is based on Tandem central computers, a private X25 network, terminal concentrators, and Banksys-developed terminals.

Having originally developed the system in C and assembler, Banksys decided to change to Ada because of its real-time capabilities, ease of maintenance on larger systems, and high level of portability.

The Union Bank of Switzerland has written two systems, COSY and DESY+, almost entirely in Ada. COSY (Control System) is a real-time monitoring and control system for VAX/VMS and RISC Ultrix architectures. It enables the bank to manage large computer sites with a minimum of operations staff and to maximize system up-time. COSY was first released in mid-1988. Now in its fourth version, COSY allows almost fully automated systems operations with a graphical user interface running under Motif.

Like Reuters' Ada systems, the DESY (Dealing System) supplies foreign exchange dealers with real-time data to support their transactions. The first DESY release was not written in Ada. In 1986, it was modernized with some Ada. The new version, DESY+, will be released next year, and is written almost entirely in Ada.

Ada IN COMMERCIAL AVIONICS

Nowhere is Ada more deeply entrenched in both the public and private sectors than in the international avionics market.

In the public sector, Boeing Commercial Airplanes can be credited with leading the push for Ada. On May 25, 1985, Boeing established the policy that it would use Ada in future avionics systems, related laboratory facilities, simulations, and associated tools. After the company lobbied the Airlines Electronic Engineering Committee of the Aeronautical Radio, Inc. (ARINC), the committee selected Ada as the "language of choice" in 1988. (Domestic airlines founded ARINC in the 1940s in order to regulate radio navigational frequencies. Since then, the airlines have tried to maximize standards through ARINC that could benefit the entire avionics community.)

Today, Boeing uses about 500,000 lines of Ada to fly its commercial 747-400 in subsystem components, critical certification, and human safety features. Two of the three largest systems on the 747, or 43 percent of the executable bytes, are written in Ada. The software is FAA certified. Boeing's new 777, which is costing between $4 to $5 billion to develop, will be 90 percent Ada by lines of code when it makes its maiden flight in 1994. Brian Pflug, manager of the Central Software Engineering Group in Renton, Wash., says that Ada portability saves Boeing's suppliers the most money.

Another leader in using Ada for flight is Collins Commercial Avionics in Cedar Rapids, Iowa, of Rockwell International. Collins began using Ada in late 1983 for government work. It decided to write commercial applications also in Ada in order to swap personnel, compilers, tools, and training easily between projects. For example, Collins invented and developed a Global Positioning System (GPS) satellite communications board in Ada for the US DoD in the mid-1980s. Later, the division installed the board in commercial airplanes, trains, and even a van that it uses to demonstrate state-of-the-art technology to international automobile makers. Rockwell's Ada work has since spread to its divisions in California, Texas and Florida.

Collins' first commercial applications of Ada were fiber-reinforced plastic corporate turbo-props, the Beechcraft Starship 1 and Beechjet. It started programming the Starship's 375 000 lines of Ada in 1984. Since then, Collins has written a Central Maintenance Computer and an Integrated Display System in Ada, both of which fly in the Boeing 747. Boeing's 737, 757, and 767 use Collins' Electrical Flight Instrument System equipment. In June 1991, the Collins division began marketing its Ada-run GPS modules, called NavCore V, to original equipment manufacturers for around $450. Collins' market for the 2.5" x 4" module includes manufacturers of navigational systems for airplanes, commercial fishing boats, trains, yachts, etc.

FAA'S ADVANCED AUTOMATION SYSTEM

The largest avionics effort written in Ada is the U.S. Federal Aviation Agency's (FAA) $12 billion effort to modernize its air traffic control system. IBM Federal Sector Division in Rockville, Md., won the contract in 1988 for developing 2.3 million lines of new code for the Advanced Automation System (AAS) portion, which will cost approximately $3.55 billion. About 1.8 million lines of code will be written in Ada.

The AAS portion will support requirements for takeoffs and landings, and will control departures and arrivals. It will monitor flights at 22 enroute control stations, 188 terminal radar approach control facilities, 258 air traffic control towers, and more. It also will make suggestions for efficient routing and fuel consumption.

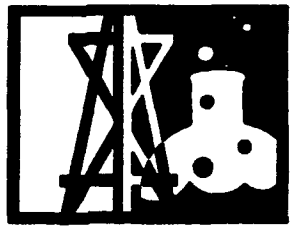
support for programming in the large especially attractive.

The first Ada code to be flown in an ESA spacecraft should take place in the Infrared Solar Observatory (ISO) satellite in May 1993. ISO is a scientific satellite going to the Sun. The Attitude and Orbit Control subsystem of the satellite is developed in Ada using a 1750 processor (MAS-281) from UK-based Marconi.

At present, ESA is investing heavily in preparing future Ada technologies, such as in developing an Ada Tasking Coprocessor (ATAC). ATAC is a VLSI chip implementing the full Ada tasking model which can be attached to any 16 or 32-bit microprocessor available in the market. It takes care of all scheduling decisions.

The agency's hard real-time system studies have led to the proposal of new methodologies for Hierarchical Object-Oriented Hard Real-time Systems (HRT-HOOD). ESA was instrumental in developing HOOD as a method to incorporate state-of-the-art scheduling techniques; i.e., deadline monotonic scheduling.

Ada IN OIL EXPLORATION



Like NASA and the ESA, Shell Oil and Dowell-Schlumberger Inc. were concerned first with writing reliable and accurate software, and chose Ada because of its reputation. The oil companies hoped to save money by using software to predict the outcome of proposed projects.

Shell initially selected Ada in 1985 because of its software engineering features — such as records, pointers, strong typing, generics, exception handling, and multi-tasking — and because of the international standard. It uses two Ada systems in testing ocean floors for oil: a seismic processing system, which is written almost entirely in Ada, and a graphical user interface, which includes C and UIL. Both systems constitute a single, larger project.

The seismic system breaks long processing sequences into small parts. Each part is programmed with an Ada task, allowing for parallel execution. The system has been ported and successfully executed on Sun3, Sun4, Convex, VAX, RS6000, and Cray machines, using several different compilers.

Seismic processing involves performing hundreds of individual steps on a great quantity of data. The graphic interface allows a user to assemble hundreds of batch jobs and to decide the sequencing among them. A multi-colored display shows their status. The system uses Ada for the background processes (which handle the job management functions) and the internal portion of the actual interface. Roughly, the user interface consists of 217,000 lines of Ada code, and 363,000 lines of other languages. The runtime system consists of 222,000 lines of Ada, and the operations is projected to be 250,000 lines of Ada. The interface is now in production use, driving an older signal processing system. Users will begin testing the Ada seismic system later this year.

Dowell-Schlumberger Inc., in Tulsa, Okla., an oilfield service company, has written between 150,000-175,000 lines of Ada for simulation software since 1985. The company's five Ada applications, which run on MicroVax IIs, predict what will happen and how much material is needed when the company provides a service for an oil producer. The tool CemCADE (Cement Computer-Aided Design and Evaluation), for example, simulates the cementing of an oil well to stop oil and gas from rising and mixing with the fresh water supply around it. PacCADE does the same for packing gravel around the oil well. The company's 200 international locations all use the tools.

Victor Ward, section head of the CADE Product Team in Tulsa, says Ada was originally chosen because of its generics and because code could be easily maintained and reused. "Ada isn't more difficult to use than any other language", he said, "once you get over the start up costs".

Ada IN THE PACIFIC

The Japanese SIGAda, with 410 members, is one of the international special interest group's largest chapters. Only about 25 of the members in Japan are from academia; the others are from Japanese corporations. The world's largest corporation, Nippon Telegraph and Telephone (NTT), was one of the first to commit to Ada by developing compilers and support tools in 1983, when the language became a standard. By 1989, NTT had developed 2.5 million lines of Ada code. "Software productivity and reliability are critical to NTT," according to Kiyoshi Tanaka, a senior research engineering supervisor. "The Ada language promised, and has proven to be in practice, a sound basis for the development of large-scale commercial software systems."

NTT has implemented several commercially available telecommunications services in Ada: a videotext communication system, a cellular telephone service, a satellite communications system, and a database management system. It has started developing a digital cellular telephone system service in Ada, using an object-oriented design.

systems. The number has grown steadily since then, until the Clearinghouse's October 1992 Ada Use Database listed over 90 commercial applications. The language has caught on with some small developers, who are using it to edit videotapes in Saratoga, Calif., and to search documents with hypertext in Houston, Texas. Some larger companies are testing their products' reliability with Ada software, such as Motorola in Illinois testing its cellular phone switching systems, Trace Inc. in California testing bare circuit boards, and Collins Avionics in Iowa testing a variety of its electronic navigational systems.

For future markets, Ada compiler vendors now have products for the hand-held computers, which shops use to read bar-coded prices and overnight delivery services use to route packages. Ada compilers are also now available for digital signal processors, which operate everything from suspension systems in automobiles to high-speed modems in PCs.

By satisfying DoD requirements, Ada was able to appeal to a much larger market than its creators first envisioned. Today, the commercial sector, which includes an estimated 24 percent of the Ada market, may not financially support Ada vendors enough to keep them afloat when the DoD begins cancelling projects. Research and development contracts are often the first in line to be cut, and many of them are being written in Ada. As the defense industry slims down, more commercial software developers will have to see Ada as a solution to their cost overruns and maintenance problems in order for the language to be viable in the next century.

ACKNOWLEDGEMENTS

We would like to thank the following people for their generous contributions to this article: Jose-Luis Fernandez of ISDEFE, Spain; Bjorn Kallberg, Ulf Olsson of Nobeltech, Sweden; and Marcus Meier of UBS, Switzerland; and John Walker of the Ada Information Clearinghouse in Arlington, Va.

THE AUTHORS

Ann S. Eustice is vice chair of the SIGAda Commercial Ada Users Working Group (CAUWG). She is a writer for ITT Research Institute, and publishes regularly in the Ada Information Clearinghouse Newsletter.

Barry Lynch is a director of Software Professionals Ireland in Dublin. He is a board member of Ada Europe and an International Representative on the Executive Committee of ACM SIGAda. His special Ada interests are in environments and public tool interfaces.

Ada—
A Software Engineering Tool

by
Richard D. Riehle

Robert Frost once said that writing free verse was like playing tennis with the net down. The game under those conditions requires great self-discipline. In computer programming we often play with the net down. Ada puts up the net, adds several referees, and installs electric-eye sensors on the fault line and boundaries.

As a response to a "software crisis," the U.S. Department of Defense (DoD) decreed in 1979, that all new software development should be performed using Ada. Then the DoD created a waiver process so that nearly anyone with a good imagination could create a rationale for avoiding the transition to Ada. The result: Ada didn't have enough launch-pad thrust to reach orbit. Another factor in Ada's slow acceptance was a reluctant IBM. Without the IBM endorsement, any new language has trouble gaining acceptance. According to *Aviation Week and Technology*, IBM finally put its full weight behind Ada after its Federal Systems Division lost several million dollars in government contracts that required Ada.

Now Ada's popularity is increasing in velocity: internationally in Europe and Japan; academically in the university software engineering community. The National Aeronautics and Space Administration (NASA) has adopted Ada. The Federal Aviation Administration has chosen Ada for the new air traffic control system. The University of Santa Clara now requires Ada instruction in its electrical engineering curriculum. Ada is even gathering a following in the commercial and MIS marketplace. CRI, Inc., of Santa Clara, California, has developed a relational database product in Ada for use in Ada systems.

Estimate of Ada Market Potential in U.S. Aerospace Industry

1986	1987	1988	1989	1990	1991	1992
Thousands of People Developing Software						
100	112	125	140	157	176	197
Percent Working on Ada Programs						
3%	6%	12%	20%	30%	40%	50%
Thousands of People Developing Software in Ada						
3.0	6.7	15.0	28.0	47.0	70.4	98.5
Investment per Person Working on Ada Programs (In Thousands)						
$10	$10	$10	$10	$10	$10	$10
Ada Support Marketing Opportunity (In Millions)						
$30	$67	$150	$280	$471	$704	$985

Assumption: Number of software developers grows 12% per year
Source: Rational, Mountain View, California

In April 1987, Undersecretary of the Army James Ambrose declared an end to waivers and mandated that all Army projects be developed in Ada, including MIS applications. Since then all branches of the DoD have elevated their commitment to Ada, and most of the defense contractors have gotten the message. Ada is now alive and well—in fact, thriving—and is a viable language alternative for any software development project.

Why do we need another programming language? And why Ada? And what is an "Ada," anyway? Then again, what is all this nonsense about "software engineering"? Is that just another fancy term for programming in the way that "sanitary engineer" is another name for janitor? In this series of articles, we hope to answer these and other related questions about Ada. We'll discuss the premises on which the language was designed and its differences from other languages, and we'll examine the issue of fulfilled and unfulfilled expectations.

Language Characterisitics

Ada is a descendant of Algol-68 via Pascal. If you know Pascal, Algol, or PL/1, you will find much that's familiar in Ada. However, Ada is a very formal language, and many words and phrases take on specific, new meanings when describing Ada concepts. Also, Ada adds new capabilities to its ancestral languages and moves in the direction of "object-oriented design" (OOD). But Ada is not a "pure" OOD language in the image of Smalltalk, Objective C, or ACTOR.

Before we proceed with more detail, let's take a look at some of the major characteristics of the language. First a definition. We use the word *type* a lot in Ada. A type defines both the permitted set of values and the legal operations for an object. Objects may be discrete data items (scalars), composite data items, or entire executable modules.

Some of Ada's more prominent features are:

- multiple levels of abstraction
- strong typing
- strict scope and visibility rules
- high modularity
- object-oriented design
- built-in exception handling
- built-in concurrent/real-time processing capability
- separation of specification code from implementation code
- separate compilation of modules
- incorporation of current concepts of software engineering

One of Ada's unique features, *generic* components, enables Ada programmers to build context-independent modules.

The Software Engineering Imperative

Without an awareness of Ada's software engineering foundations, it will be hard to understand why the code sometimes reads like the software equivalent of a Bach fugue.

During the past twenty-five years intensive scholarly research into the programming process has resulted in concepts like "formal proof of correctness" (Dijkstra), Structured Analysis and Design (DeMarco), problem/solution space models (Ledgard), and "information hiding" and "levels of abstraction" (Parnas). There have been hundreds of other contributors to this evolving discipline, and a visit to a university bookstore will now turn up plenty of titles that include the expression "software engineering."

Goals of Software Engineering

The designers of Ada adopted the goals of software engineering defined by Ross, Goodenough, and Irvine:

- Modifiability
- Efficiency
- Reliability
- Understandability

These software engineering goals are no different from the implied goals in our day-to-day programming, but now they are explicitly stated—almost codified.

An additional goal for Ada is "portability": There is only one Ada. No dialects of the language are permitted. Source code must compile in any Ada environment. The Ada Joint Programming Office (AJPO) of the DoD validates every Ada compiler. An unvalidated compiler isn't Ada. This is the first computer language subjected to such a rigorous standard. Any attempt to corrupt Ada will be rebuffed. She's ". . . just not that kind of girl."

Principles

The goals of software engineering led to a definition of underlying principles. By principles, we do not mean "methods." A method is something like "structured analysis." A principle is the foundation for the method.

Since software engineering is an emerging discipline, there is no complete agreement on all of the principles, but the principles most commonly associated with Ada are also defined in the work of Ross, Goodenough, and Irvine:

- Abstraction
- Information hiding
- Modularity
- Localization
- Uniformity
- Completeness
- Confirmability

Notice the absence in this list of Warnier-Orr, Jackson System Development, software metrics, object-oriented design. These are "methods," which would be based on the principles.

Abstraction

One of the most important principles in software engineering is "levels of abstraction." By abstraction we mean: Show only the essential properties of a program without revealing the details. There is a presumption that we can decompose an abstraction into its components.

Ada is expressly designed to enable multiple levels of abstraction. We find this principle represented in Ada by packages, generic program units, distinctions between unit specification and unit body, among abstract data types, and between private and limited-private types, and by the ability to define entirely new types.

An example software abstraction familiar to many designers is the Data Flow Diagram (DFD). The "context diagram" represents the highest level of abstraction of the DFD. Subsequent DFD levels describe subordinate levels of abstraction until we are at the most elementary (non-decomposable) level.

Information Hiding

Information hiding is closely related to abstraction. We simplify the use of program units by hiding unnecessary information. Information, here, is defined in a very broad sense and includes details about algorithm implementation, data types, and objects.

Information hiding is not a new principle. It has been available to us in one form or another from the earliest days of programming. One example is the OPEN command found in many languages. We issue the

```
OPEN (parm1, parm2)
```

statement and are spared the tedious effort of coding our own device driver interface, exception handling, etc. Ada, by its

Who Is Ada?

Ada is often called "the first computer programmer."

Augusta Ada Byron, daughter of Lord Byron, was born December 8, 1815, in England. She was always addressed by her middle name Ada. When her father left to caroose about the continent, Ada's mother Lady Byron set about educating Ada in mathematics as a moral discipline. Ada continued to study mathematics throughout her life.

Ada became friends with Charles Babbage, inventor of the Difference Engine and the Analytical Engine. An Italian engineer, Luigi Menebrea, wrote a short paper describing the Analytical Engine, and one of Babbage's friends suggested that Ada translate the paper into English.

When Ada undertook the task of translating Menebrea's paper, she also decided to make a few "notes." The notes, labeled A thru G, were three times the length of the original paper. It is in these notes that Ada inadvertandy immortalized herself in computer history.

The "notes" describe programming methods far beyond what was then possible with the Analytical Engine. Ada's mind took a leap into the future. One hundred years before Eniac, she was describing looping constructs, branching, subroutines, variables, and GIGO (garbage in, garbage out). Ada even touched lightly on Artificial Intelligence. Of course, Ada did not use our contemporary computer argot, but the notes read almost like one of the popular books of our day that describe the possibilities of electronic computers.

When in 1979, the U.S. Department of Defense High Order Language Working Group (HOLWG) finally decided to accept the language design of Jean Ichbiah's group at France's Honeywell/Bull, the language needed a name. According to Grady Booch, author of *Software Engineering with Ada*, "Jack Cooper of the Navy Material Command evolved the perfect name for this new language: Ada . . . ."

It seems appropriate that an organization as committed to feminist issues as the Department of Defense should name its new programming language in honor of a nineteenth century woman.—*RDR* ♦

very design, expands upon and actually enforces this principle. In later discussions we'll examine data types that provide elegant ways to hide implementation details.

Modularity

Ada encourages program design using small, well-defined modules. Ada programmers have standard methods for separate compilation, data and procedure encapsulation, the creation of "loosely-coupled" objects, and top-down structural design.

The phrase "loosely-coupled" is an important concept in Ada's rendition of object-oriented design. The loosely-coupled object in Ada is a *generic* unit that can be used over and over in many different programs independently of the specific contexts of those programs. In fact, building generic "reusable components" is a key feature of Ada.

An example of loose-coupling in the real world is in your automobile. You can't take the carburetor from your Chevy and put it on my Ford. The carburetor and the engine are tightly-coupled objects. On the other hand, your twelve-volt battery is a loosely-coupled object that I can steal from your Chevy and use in my Ford.

This leads us to the concept of "software ICs." Many integrated circuits are loosely-coupled. Often, an electrical engineer may select generic ICs (objects) from a catalog to create a unique hardware design. Object-oriented design strives to develop reusable software units, or components, as generic as those integrated circuits!

Localization

Localization is almost the inverse of modularity. But localization stresses the cohesiveness of the objects in a module. This tends to make individual modules small, logically concise, and easy to modify. We see here that much of Ada's design relates to the modifiability goal. Small, loosely-coupled, highly-cohesive modules are easier to modify and maintain. They are also easier to create.

Ada permits an encapsulated *type* that can be affected only by the operations defined within the scope of a package. This is achieved via *private* and *limited*-private data types. The end result is "abstract data types."

Uniformity

We could also use the word consistency. Every programming organization has its standards. Some use indentation on the line after the (while others insist on data name normalization and still others are totally laissez faire. Often, "correct" coding style is subject to argument. The principle of uniformity is related to the goals of modifiability and understandability. The idea, of course, is to keep the style the same.

Ada supports uniformity by virtue of its rigorous structure. Ada programmers build software systems as small, well-defined modules (Ada *packages*). The implementation coding style of a particular package may be peculiar, but the package user never sees the algorithmic code. All access to the package is through the package specification, and that substantially limits the variations of style available to the package developer. Even here Ada doesn't let us play with the net down.

Completeness

How can we know our solution is complete? No programming language alone can make this happen, but there are those Ada enthusiasts who insist that Ada's structure and environment can help.

Confirmability

By confirmability, we mean some method of determining that our program is correct. Edsgar Dijkstra's "formal proof of correctness" for software would be ideal here. Unfortunately, neither Ada nor any other language advances us to that level of confirmability.

Ada's modularity enables us to separately compile and test program components. In addition, strong data typing provides tools to enforce confirmability. As a stongly "typed" language, Ada encourages design that includes defining new, rigorously constrained data types. And Ada's built-in exception handling permits us to implement our own error and exception routines.

Software Engineering Beyond Ada

No language can provide, by itself, all the tools necessary for fulfilling the goals and principles of software engineering. But Ada is more than a language. Compiler developers are also supplying the Ada Programming Support Environment (APSE). In future articles we'll explore different implementations of APSE.

Other elements of software engineering are also necessary—project management tools, structured methods (analysis, design, walkthroughs), prototyping, quality assurance, Computer Assisted Software Engineering (CASE) tools, and software metrics.

The Ada Language

Now that we have reviewed some of Ada's underlying goals and principles, we can look at the language itself. In keeping with the principles of software engineering, we start at the highest level of abstraction, the *package*.

The package is unique to Ada. We can find concepts in other languages that roughly correspond to the Ada package but nothing that is as complete. By a package we mean a collection of logically related objects. This collection can be data, data types, related subprograms, and type declarations. A package consists of two parts: the *package specification* and the *package body*.

The user of a package usually has no need to see the details of the implementation (package body). The only part of a package the programmer would see is the package specification. The specification is the programmer's window into the package.

There are software companies that specialize in creating both generic and non-generic Ada packages. This is a grow-

Ada Reserved Words

abort	declare	generic	of	select
abs	delay	goto	or	separate
accept	delta		others	subtype
access	digits	if	out	
all	do	in		task
and		is	package	terminate
array	else		pragma	then
at	elsif		private	type
	end	limited	procedure	
	entry	loop		
begin	exception		raise	use
body	exit	mod	range	
			record	when
			rem	while
		new	renames	with
case	for	not	return	
constant	function	null	reverse	xor

Why Do We Need a New Language?

In his book on Software Engineering, Henry Ledgard describes what sets the professional apart from the amateur programmer: The completed work of a professional must be reliably usable and maintainable by someone other than the author. Moreover, the professional typically develops software that will interact with a large body of other software written by other professionals.

There are other differences. Professional programmers and programming managers know that the vast majority of time in any software organization is devoted to maintenance of existing code. In the Department of Defense it is estimated that 80% of the software dollar goes into maintenance. Not only is this a thankless task, but it also propagates new bugs. Additionally, because there is no "science" of programming, each coder does things differently. Attempts at discipline over the years have been dismally unsuccessful. Imagine maintaining the code for all the systems of the DoD (or any government agency) and transporting the code into a worldwide military command and control system or two hundred logistics systems.

There is also the old issue of re-inventing the wheel. Most of the code we create does the same task as some other code written by someone else. Comparatively little original software is coded. The ideal is to make code as re-usable, generic, and simple as paperclips. This is a non-trivial problem in a small organization like your hometown bank; certainly non-trivial in an organization the size of the DoD.

Now imagine the cost of all this duplication, error-prone maintenance, and managerial oversight. It's enough to make any self-respecting general want to give up nuclear weapons and return to clubs and stones.

Ada is designed for professional programming/software engineering. The typical Ada project is very large, consists of many diverse components, and must be faultlessly reliable. The system is usually constructed by a team of programmers working on independent modules, and each module (package) must work properly with all the other modules. Professor Phil Schrodt of Northwestern University has stated a rule of thumb: "No good program can be written by more than ten people. The best programs are written by one or two people." Ada presumes to repeal Professor Schrodt's rule of thumb.

It became clear in the late 1960s that the DoD was in a "software crisis." By the early 1970's the DoD was using over 400 different programming languages. Maintenance of the systems programmed with these languages was becoming a nightmare, and the very computer software that was being implemented for the defense of the country was becoming part of the threat.

When a government agency has a problem, the first thing it does is establish a committee. In this case the committee, formed in 1975, was named the High Order Language Working Group (HOLWG) and consisted of both defense and civilian members. HOLWG developed criteria for the "ideal" DoD language and surveyed the languages then available. HOLWG decided that none of the languages was consistent with DoD requirements.

In 1977, HOLWG issued a request for proposal for a new computer language. It received 15 proposals and selected four finalists labeled Red, Green, Blue, and Yellow. In the next round of refinements, the Red and Green languages were selected. Finally, in 1979, the Green language, submitted by a team at Cii-Honeywell/Bull of France, was selected. The team was headed by Dr. Jean D. Ichbiah.

Ada was designed according to current concepts of software engineering. Some say it is the first language to be developed that way. It is, however, not without its critics. Many software engineers believe that a better language that implements the principles is Modula-2, developed by the author of Pascal Niklaus Wirth. Some say that Ada is the last great language of the 1960s and point to the emergence of 4GL (Fourth Generation Language) technology. Still others dislike the "size" of the language, comparing it to the current trend towards Reduced Instruction Set Computers (RISCs) instead of Complex Instruction Set Computers (CISCs).

The jury is still out on the eventual fate of Ada. At present it seems to be healthy and gaining strength. Just a few years ago, many people had their doubts. According to a bulletin released by the Ada Joint Programming Office (AJPO) in December 1987, there are now over 120 compilers available for the language. Moreover, organizations that have adopted Ada and implement its "reusability" features report dramatic improvements in programming productivity.

There is hardly a mainframe or minicomputer environment for which there is no compiler. As you might expect, there are outstanding Ada compilers for the DEC VAX series. Strangely, though, the system best designed to take advantage of Ada's tasking feature, the Cray series, does not yet have a "validated" compiler. By "validated" we mean that the compiler has successfully compiled and executed the 1,850 programs contained in the AJPO test suite. And the rule is, until it is validated by the AJPO it isn't Ada. Recent estimates are that there will be about 150 validated Ada compilers by Summer of 1988.

There are some good implementations of Ada for the IBM/Clone PC environment. For a low-cost Ada compiler, try RR Software, Inc.'s JANUS/Ada (C-PAK priced around $100). Recently, Meridian Software introduced a pre-validated version of Ada for the Macintosh. Meridian and Alsys also have validated compilers for MS-DOS environments.—*RDR* ♦

Ada Compilers for Microcomputers

This list of Ada compilers is not comprehensive, nor should its entries be construed as specific recommendations.

		Validated	Environment
Janus/Ada	RR Software, Madison, WI	Yes	PC/Clone
AdaVantage	Meridian Software, Laguna Hills, CA	Yes Soon Yes	PC/Clone, MacIntosh (under MAC OS) Z-8000
Alsys	Alsys, Inc. Waltham, MA	Yes Yes Yes Soon	PC/Clone, Macintosh (under AUX) Motorola 68000 family PS/2 OS/2
Ada-86	Softech, Inc. Waltham, MA	Yes	Cross-compiler from Vax series targeted at the INTEL 8xx86 series
NYU Ada/Ed	New York University		

NYU Ada/Ed is a low-cost "Ada" interpreter designed for learning something about Ada. However, you can't do much with it and it's not at all suitable for production programming. Ada/Ed will execute most of the programs in the book by Clark (see the bibliography), but watch out for integer sizes.

ing business, but there is still a need (and a marketplace) for better packages.

Here is a skeleton package that incorporates other packages. Note that Ada comments are preceded by two hyphens (--):

```
with OPERATIONS_RESEARCH;              -- begin package specification
package BOX_CAR_SCHEDULER is
    type STATUS_TYPE  is (FULL, EMPTY);     -- defined  data type
    type SERIAL_NO  is new INTEGER;         -- derived  data type
    BOX_CAR_STATUS : STATUS_TYPE;           -- assign a data type
    BOX_CAR        : SERIAL_NO;             -- assign a data type
    function  BOX_CAR_IS(BOX_CAR :in SERIAL_NO) return
     STATUS_TYPE;
    procedure PLACE_BOX_CAR(PARM1:PARM2:PARM3...);
            .
            .
end BOX_CAR_SCHEDULER;                 -- end package specification

package body  BOX_CAR_SCHEDULER        -- begin implementation
                                       -- algorithms
            .                          -- are in this part
end BOX_CAR_SCHEDULER;                 -- end implementation
```

In this example we are creating a package to solve the classic problem of moving empty and full box cars in a railroad network. The package will use another package, OPERATIONS_RESEARCH, which is incorporated into this package by the Ada *with* statement; then we define two data types, STATUS_TYPE and SERIAL_NO, and create two variables and assign a type to each; then we specify the functions and procedures that will be found in the body of the package. The most important thing to observe here is that the package specification may be the only thing available to anyone who wants to use the package BOX_CAR_SCHEDULER. The package body may be hidden. With this specification, another programmer can create a new program unit (package, procedure, or function). For example:

```
with TRACK_MANAGEMENT;
with PERSONNEL;
with ENGINE_MANAGEMENT;
with BOX_CAR_SCHEDULER;
package RAILROAD_MANAGEMENT is
            .
            .
end RAILROAD_MANAGEMENT;               -- end package specification
```

The package body of BOX_CAR-SCHEDULER and other packages are hidden from the programmer writing the package, RAILROAD_MANAGEMENT. And the package body for OPERATIONS_RESEARCH was hidden from the creator of BOX_CAR_SCHEDULER.

Subprograms

Ada has two kinds of subprograms: *procedures* and *functions*. The difference between the two is quite simple. A function returns a result as part of an expression. A procedure is a simple statement; for example, a simple procedure to convert non-metric to metric:

```
with TEXT_IO;
procedure METRIC is
   INCHES, FEET, YARDS : INTEGER;
   METERS : FLOAT;
   package INT_IO is new TEXT_IO.INTEGER_IO(INTEGER);
        function METER_CONV(IN,FT,YDS:INTEGER) return FLOAT is
            INCH_TOTAL        : INTEGER;
            NEW_METRIC        : FLOAT;
            CONVERSION_FACTOR : constant FLOAT := 0.02540;
        begin
            INCH_TOTAL := (YDS * 36) + (FT * 12) + IN;
            NEW_METRIC := FLOAT(INCH_TOTAL) * CON-
               VERSION_FACTOR;
                  -- Not .a the conversion of INTEGER type to
                  -- FLOAT type using FLOAT(INCH_TOTAL)
              return NEW_METRIC;
        end METER_CONV;
begin
        INT_IO.GET(YARDS);
        INT_IO.GET(FEET);
        INT_IO.GET(INCHES);
        METERS :=  METER_CONV(INCHES,FEET,YARDS);
end METRIC;
```

Here we have a function call within a procedure. In addition, we have a procedure call, GET, from INT_IO.

Illustrations like this often elicit a ho-hum response from experienced programmers. There's nothing here that couldn't be done in some other language. The directive with is somewhat equivalent to C's **#include** or Pascal's {$I ...}, but there are some advantages in clarity. For example, the value to be returned in an Ada function is always explicitly returned. Also, we do an explicit type conversion on INCH_TOTAL in the expression that computes CONVERSION.

The first thing we see in this procedure is the invocation of another package, TEXT_IO. Ada doesn't have its own input/output procedures, so input and output are controlled by Ada packages. Several packages are provided as part of the standard language implementation: SEQUENTIAL_IO, DIRECT_IO, TEXT_IO, and LOW_LEVEL_IO. Other _IO packages are available from Ada software vendors and the AJPO. These include console, screen, and window handling packages, graphics packages, and packages for device drivers.

Since we want to enter integer numbers via a keyboard, we need to use the package TEXT_IO, which contains three generic packages named INTEGER_IO, FLOAT_IO, and FIXED_IO.

An Ada generic does not exist as an executable entity. It is often referred to as a "template." To use any generic object (package, procedure, or function), you must supply the characteristics of the objects to be processed and create a newly named version of the generic object. This is called *instantiation*, meaning to create a new "instance" of the generic object. In Ada the parameter(s) for a generic may be a type, a variable, or even another subprogram.

INTEGER_IO, as a generic package within TEXT_IO, allows us to send and receive numbers from/to a device in ASCII format. INTEGER_IO automatically converts the numbers to the proper internal INTEGER format so that we can perform calculations. The generic does not exist as a working package until we instantiate it. In this case we instantiated INTEGER-IO as a new package named INT_IO for data type INTEGER.

This may seem a little redundant, but suppose we had really wanted to constrain the GET for each object in the procedure. We might have created the following new data types:

```
type INCH_TYPE is range 1..12;
type YARD_TYPE is range 1..144;
type FEET_TYPE is range 1..3;
```

then created objects of those types,

```
INCHES : INCH_TYPE;
FEET   : FEET_TYPE;
YARDS  : YARD_TYPE;
```

and then instantiated INTEGER_IO for each type

```
package INCH_IO is new TEXT_IO.INTEGER_IO(INCH_TYPE);
package FEET_IO is new TEXT_IO.INTEGER_IO(FEET_TYPE);
package YARD_IO is new TEXT_IO.INTEGER_IO(YARD_TYPE);
```

Now our procedural coding would read:

```
INCH_IO.GET(INCHES);
FEET_IO.GET(FEET);
YARD_IO.GET(YARDS);
```

If the newly instantiated package, INCH_IO, receives input outside the range of 1 through 12, a constraint error will be raised at runtime. This is one reason Ada code sometimes seems a little baroque.

Some relief is available in the form of the *use* option. Rather than explicitly name TEXT_IO as the parent of INTEGER_IO, we could have said,

```
with TEXT_IO; use TEXT_IO;
        .
        .
package INT_IO is new INTEGER_IO(INTEGER);
use INT_IO;
        .
        .
GET(INCHES);
```

Many Ada programming shops prohibit or discourage the use option because of the need for absolute clarity and control over the visibility and scope of every element of the program. This may be of little consequence in a small software product but becomes a major issue in a large software system designed to control parallel processing in a "mission critical" environment.

Tasks

One unique feature of Ada is the *task*, a program unit devoted to concurrent processing. The task may be used in either a multiprogramming or multiprocessing environment. The task has no direct analogue in other languages. Historically, we resort to assembler language or some special purpose language we access via CALL constructs to match Ada's task.

The task becomes an especially important programming construct as supercomputing and parallel processing emerge. With Ada we can design a system in which we launch twenty (not a limit) parallel tasks, all of which can communicate with each other. Tasks can be used with great efficiency in complex simulation programs where it is often convenient to break a problem into small pieces and merge results at different stages of completion.

The overall structure of a task is similar to that of a package:

```
task  PACEMAKER is
          .                  -- Specification for a task
          .                  -- in an embedded system that
          .                  -- controls a heart pacemaker
end   PACEMAKER;

task body PACEMAKER is
          .                  -- Algorithmic implementation
          .                  -- is placed here
          .
end   PACEMAKER;
```

An important aspect of Ada is the ease with which we can create programs for "embedded systems," by which we mean software systems consisting of multiple programs and/or processors that operate independently of any human interference. Examples would be radar guidance systems, missile telemetry, automated medical monitoring and control devices, unmanned space vehicles, etc. Reliability is the most important attribute of these kinds of systems.

Ada's charter to be the language of embedded systems is one reason for the rigorous discipline it enforces on the software engineer.

We can create multiple tasks that rendezvous with each other in a variety of ways. There are methods for prioritizing, starting, stopping, and monitoring tasks. One task may start several others and wait until one of those others completes