

AD-A165 315

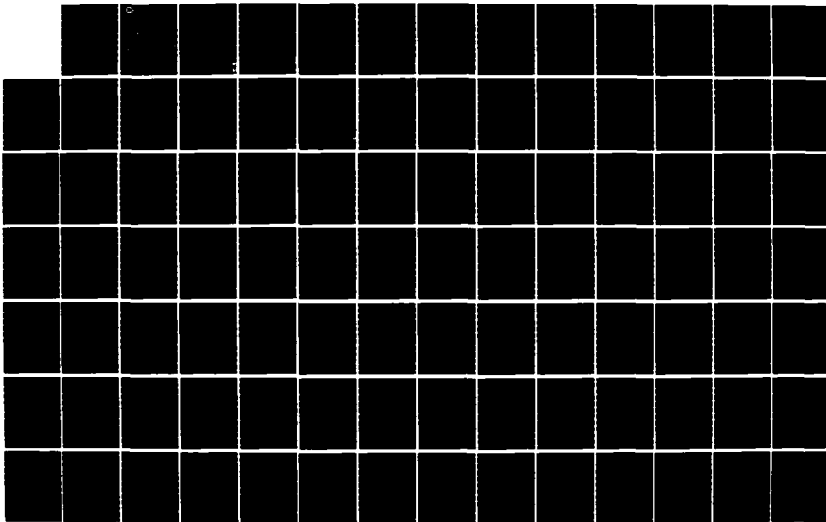
ADA (TRADEMARK) TRAINING CURRICULUM ADA (REGISTERED
TRADEMARK) FOR SOFTWARE MANAGERS L201 TEACHER'S GUIDE
VOLUME 2(U) SOFTECH INC WALTHAM MA 1986
DADB07-83-C-K506

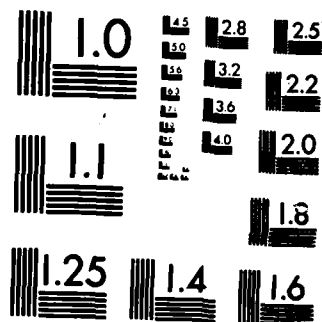
1/5

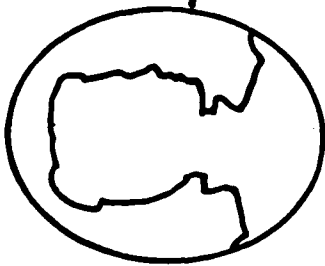
UNCLASSIFIED

F/G 5/9

NL







Ada® Training Curriculum

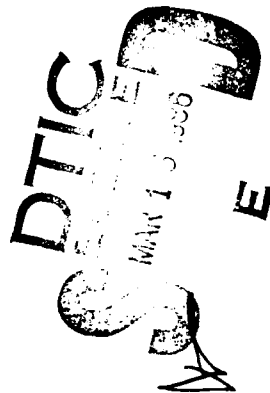
DTIC FILE COPY

AD-A165 315

Ada® For Software Managers

L201

Teacher's Guide Volume II



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

Contract DAAB07-83-C-K506

Prepared By:

SOFTECH, INC.

460 Totten Pond Road
Waltham, MA 02154

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

*Approved For Public Release/Distribution Unlimited

86 3 12 059

TASKS

VG 823.1

Accession For

☒ ☐ ☐

A-1



INSTRUCTOR NOTES

THE OTHER PROGRAM UNITS WERE PACKAGES (WHICH PHYSICALLY GROUP RESOURCES) AND SUBPROGRAMS (WHICH ARE EXECUTABLE). TASKS ARE A SECOND FORM OF EXECUTABLE PROGRAM UNIT.

TASKS

- ADA PROGRAM UNIT THAT PROVIDES PARALLEL THREADS OF CONTROL
- CONCURRENCY REAL WITH MULTIPROCESSORS
- CONCURRENCY APPARENT WITH SINGLE PROCESSOR
- MECHANISM FOR SYNCHRONIZATION AND DATA TRANSMISSION IS CALLED "RENDEZVOUS"
- RUNTIME SYSTEM HANDLES TASK SCHEDULING

INSTRUCTOR NOTES

TASKS ARE AN IMPORTANT DESIGN ISSUE. POINT OUT THAT SYSTEM DESIGN SHOULD BE APPROACHED WITH THE IDEA OF USING TASKS WHERE APPROPRIATE. OPTIMIZATION TECHNIQUES EXIST TO REDUCE THE OVERHEAD OF TASK SYNCHRONIZATION. OPTIMIZATION SHOULD NOT BE DONE PRIOR TO IDENTIFYING BOTTLENECKS IN THE CODE USING BENCHMARKS AND OTHER ANALYSIS.

TASKS

CAN BE USED FOR

- CONCURRENT ACTIONS
- CONTROLLING RESOURCES
- INTERRUPTS
- BUFFERS

INSTRUCTOR NOTES

(TASKS MUST BE INSIDE OTHER PROGRAM UNITS)

POINT OUT THE DIFFERENCE BETWEEN TASK SPECIFICATIONS AND OTHER SPECIFICATIONS: TASKS
HAVE ENTRY DECLARATIONS ONLY.

VG 823.1

7-31

TASK SYNTAX

```
task Task_Name is
    entry Entry_Name [(Formal_Parameters)];
end [Task_Name];
```

SPECIFICATION

- Visible Part
- Defines entry point(s) to the task


```
task body Task_Name is
begin
    accept Entry_Name [(Formal_Parameters)] do
        -- statement(s);
    end Entry_Name;
end [Task_Name];
```

BODY

- Hidden
- Additional Declarations
- Implements the task

-- statements
-- accept entries

• TASKS ARE DECLARED IN SUBPROGRAMS OR PACKAGES

INSTRUCTOR NOTES

VG 823.1

7-41

TASKS -- RENDEZVOUS

WHEN EITHER A CALLED TASK OR ITS CALLER ARRIVE AT A RENDEZVOUS POINT, THE ONE THAT ARRIVES FIRST WAITS FOR THE OTHER ONE

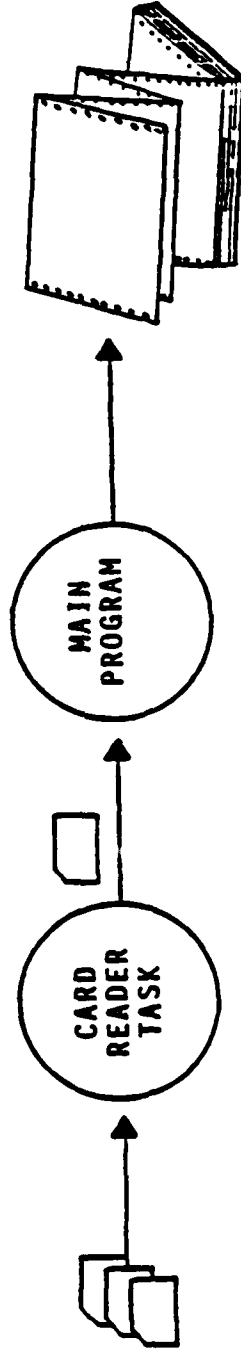
DURING THE RENDEZVOUS (THE EXECUTION OF THE STATEMENTS IN THE ACCEPT) THE CALLER IS SUSPENDED. THE CALLER RESUMES EXECUTION AFTER THE RENDEZVOUS IS COMPLETED.

INSTRUCTOR NOTES

THE CODE FOR THIS EXAMPLE APPEARS ON THE NEXT SLIDE AND IS TO BE DISCUSSED IN DETAIL.
MAKE SURE THE STATEMENT OF THE PROBLEM IS WELL UNDERSTOOD BY THE CLASS.

TASKS TO OVERLAP I/O

COPY CARD IMAGES FROM "STANDARD INPUT" TO "STANDARD OUTPUT" --
READING IS IMPLEMENTED AS A TASK EXECUTING CONCURRENTLY WITH THE
MAIN PROGRAM



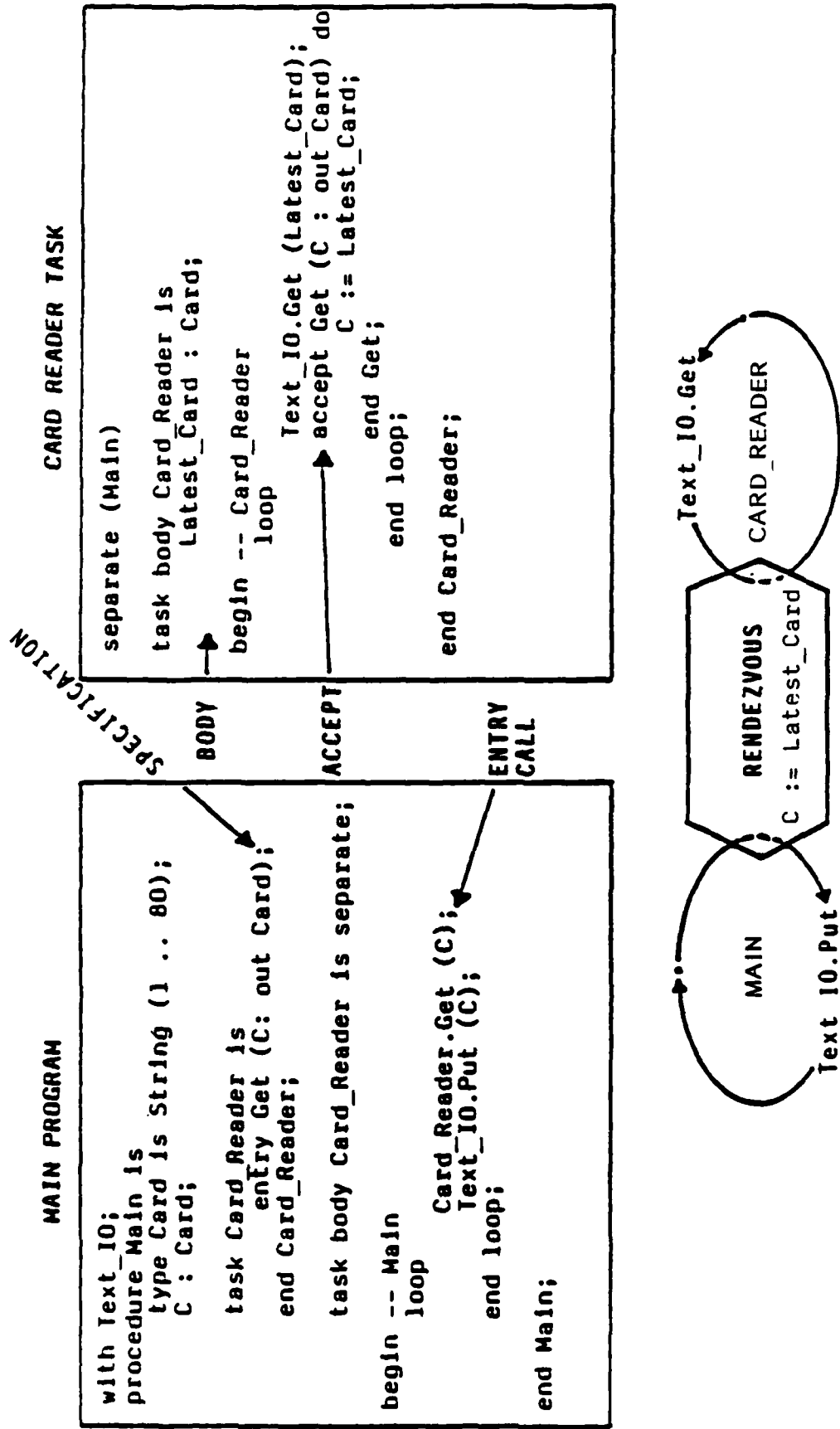
INSTRUCTOR NOTES

WALKTHROUGH THE EXAMPLE IN DETAIL.

VG 823.1

7-61

BASIC SYNCHRONIZATION



INSTRUCTOR NOTES

AN EXAMPLE OF A SELECT STATEMENT WITH GUARDS OCCURS IN SEVERAL PAGES IN THE RESOURCE
CONTROLLER EXAMPLE.

OTHER SYNCHRONIZATION ALTERNATIVES

ADA PROVIDES ADDITIONAL SYNCHRONIZATION FOR THE FOLLOWING SITUATIONS:

CALLED TASK

- ACCEPTS ONE OF SEVERAL ENTRY CALLS

CALLING TASK

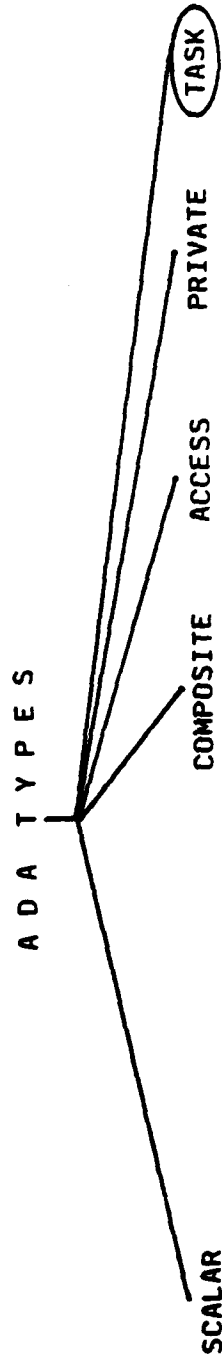
- SUSPENDS EXECUTION FOR SOME SPECIFIED TIME
- CALLER NEEDS TO INITIATE RENDEZVOUS WITHIN A GIVEN TIME
- CALLER MUST RENDEZVOUS IMMEDIATELY

INSTRUCTOR NOTES

VG 823.1

7-81

TASK TYPES



INSTRUCTOR NOTES

RELATE BACK TO THE DISCUSSION OF TYPES WHERE A TYPE IS A DEFINITION. TO GET A VARIABLE
YOU DECLARED AN OBJECT. SAME THING HERE. WHAT WE GET IS A PROCESS.

TASK TYPES -- BASIC IDEA

- ALLOWS THE USER TO CREATE A TASK TEMPLATE
- TASK OBJECTS CAN THEN BE CREATED FROM THIS TEMPLATE - OBJECTS ARE LIKE ANY OTHER TASK

INSTRUCTOR NOTES

VG 823.1

7-101

17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

TASK TYPES -- BASIC IDEA

EXAMPLE:

```
task type Simple_Channel is
    entry Send (M : in Message);
    entry Receive (M : out Message);
end Simple_Channel;

task body Simple_Channel is
    -- LIKE A TASK DEFINITION
end Simple_Channel;

TO USE:

Channel_1, Channel_2 : Simple_Channel;
.
.
.
Channel_1.Receive (Msg); -- GET A MESSAGE
Channel_2.Send (Msg);    -- AND SEND ON OTHER CHANNEL
```

INSTRUCTOR NOTES

USE TASKS WHEN LOGICALLY SEVERAL OPERATIONS CAN BE OCCURRING INDEPENDENTLY OF EACH OTHER.

THE ALTERNATIVE WOULD BE TO HAVE CHECKS FOR THE TIME EVERY FEW STATEMENTS WITHIN EDITOR (I.E., A CLUTTERED MESS). THIS SOLUTION LETS THE EDITOR BE AN EDITOR AND THE TICKER, A TICKER. IN THIS WAY THE ABSTRACTION OF THE PROBLEM IS PRESERVED.

POINT OUT THAT Calendar IS A PREDEFINED PACKAGE WHICH PROVIDES DATE AND TIME UTILITIES.

VG 823.1

7-111

TASKS AS CONCURRENT ACTIONS

PROBLEM: AN INTERACTIVE PROGRAM (E.G., AN EDITOR) MUST DISPLAY THE TIME OF DAY IN A CORNER OF THE SCREEN AT ALL TIMES.

```
with Calendar;
procedure Editor is
  procedure Display_Time (T : in Calendar.Time) is separate;

  task Ticker;

  task body Ticker is
  begin -- Ticker
    loop
      Display_Time (Calendar.Clock);
      delay 0.1;
    end loop;
  end Ticker;

  begin -- Editor
    ...
  end Editor;
```

INSTRUCTOR NOTES

IN SYSTEM STARTUPS COULD BE PERFORMING BACKGROUND DIAGNOSTICS (E.G. WHAT RESOURCE DEVICES ARE CURRENTLY AVAILABLE) WHILE THE MAN MACHINE INTERFACES ARE PERFORMED (E.G. PROMPTING THE OPERATOR FOR THE DATE, TIME).

TASKS AS CONCURRENT ACTIONS

OTHER REAL-TIME SYSTEM EXAMPLES:

- MAN-MACHINE INTERFACES
- BACKGROUND DIAGNOSTICS

INSTRUCTOR NOTES

ADA EXAMPLE SPANS 2 SLIDES.

THIS EXAMPLE SHOWS A FAMILY OF ENTRIES INDEXED BY URGENCY IN THE TASK Request_Function.

WHY IS THIS NECESSARY? WHEN TWO OR MORE TASKS COMPETE FOR A RESOURCE (E.G., THE PRINTER, OR A COMMON TABLE), SOME FORM OF ARBITRATION IS NEEDED. OTHERWISE THE PRINTOUTS WILL BE INTERMIXED OR A TASK MIGHT MODIFY A SHARED TABLE WHILE ANOTHER TASK IS READING IT.

TASKS USED TO CONTROL RESOURCES

```
package Resources is
  type Resource_Type is (Reader, Printer, Terminal_1, Terminal_2);
  type Urgency_Is (High, Medium, Low);
  procedure Some_Actions (Resource : in Resource_Type);
  task Resource_Controller is
    entry Request_Function (Urgency) (Resource : Resource_Type);
  end Resource_Controller;
end Resources;
```

INSTRUCTOR NOTES

A LOOK AT THE TASK BODY. NOTE THE RENDEZVOUS POINTS AND THE GUARDS.

THE EXAMPLE IS OVERSIMPLIFIED; OTHER PARAMETERS WILL BE NEEDED IN PRACTICE, SUCH AS THE TEST TO BE PRINTED.

TASK CONTROLLING RESOURCES (Continued)

```
package body Resources is

  procedure Some_Actions (Resource : in Resource_Type) is separate;
  task body Resource_Controller is
  begin -- Resource_Controller
    loop
      select
        accept Request_Function (Urgency => High)(Resource : Resource_Type) do
          Some_Actions (Resource);
        end Request_Function (High);
      or
        when Request_Function (High)'Count = 0 =>
          accept Request_Function (Urgency => Medium)(Resource : Resource_Type) do
            Some_Actions (Resource);
          end Request_Function (Medium);
      or
        when Request_Function (High)'Count = 0 and
          Request_Function (Medium)'Count = 0 =>
          accept Request_Function (Urgency => Low)(Resource: Resource_Type) do
            Some_Actions (Resource);
          end Request_Function (Low);
        end select;
      end loop;
    end Resource_Controller;

  end Resources;
```

INSTRUCTOR NOTES

WHEN Character_Reader, Character_Ready IS CALLED, CONTROL BRANCHES TO THE HARDWARE
ADDRESS A1₁₆.

TASKS AS INTERRUPT HANDLERS

```
task Character_Reader is
  entry Character_Ready (C : in Character);
  for Character_Ready use at 16#00A1#;
  -- PHYSICAL ADDRESS OF THE
  -- INTERRUPT ROUTINE

end Character_Reader;

with Buffer;
task body Character_Reader is
begin
  loop
    accept Character_Ready (C : in Character) do
      Buffer.Add (C);
    end Character_Ready;
  end loop;

end Character_Reader;
```

INSTRUCTOR NOTES

IN I/O EXAMPLE, CAN BE ADDING TO OR DELETING FROM A_File SIMULTANEOUSLY, THUS THERE IS NO WAY TO ENSURE THE ACCURACY OF DATA IN A_File.

WHILE TASKS CAN BE USED TO CODE A CYCLIC EXECUTIVE, THEY ARE BETTER USED TO DEVELOP A MAINTAINABLE, DATA-DRIVEN DESIGN. AS AN EXAMPLE, CONSIDER A TARGET TRACKING SYSTEM WITH THESE ALTERNATIVES: A TASK PER TRACK (THERE IS AN UPPER LIMIT OF HOW MANY TRACKS A SYSTEM CAN FOLLOW) VERSUS ONE TASK PER RADAR SECTOR. TASK PER TRACK IS MORE ELEGANT AND A BETTER ABSTRACT MODEL.

SOME POTENTIAL PITFALLS - TASKS

- STATE OF DEADLOCK WHERE TWO TASKS EACH ENDLESSLY WAIT FOR THE OTHER

- UNCONTROLLED ACCESS TO DATA

procedure IO is

```
task Card_Reader;
task Printer;
```

```
task body Card_Reader is
```

```
begin
```

```
  Get_Char (A_Char);
```

```
  Put_Char (A_File);
```

```
end Card_Reader;
```

```
task body Printer is
```

```
begin
```

```
  Get_File (A_File);
```

```
  Put_Char (A_File);
```

```
end Printer;
```

```
begin -- IO
```

```
  null;
```

```
end IO;
```

```
-- A_Char, A_File PREVIOUSLY DECLARED
```

- GETTING STAFF TO USE TASKS INSTEAD OF CODING SEQUENTIALLY

INSTRUCTOR NOTES

VG 823.1

8-1

Section 8

GENERIC

VG 823.1

INSTRUCTOR NOTES

VG 823.1

8-1i

GENERIC UNITS - MOTIVATION

LET'S SUPPOSE AN ADA SYSTEM NEEDS A STACK OF INTEGERS:

```
package Int_Stack is
  type Stack_Type is private;
  procedure Push (X : in Integer);
  function Pop return Integer;
private
  Stack_Size : constant Integer := 100;
  type Stack_Type is array (1 .. Stack_Size) of Integer;
end Int_Stack;
```

AN ADA SYSTEM MAY ALSO NEED A STACK FOR REALS OR ARRAYS OR ENUMERATION TYPES OF VARYING STACK SIZE. THE ALGORITHM REMAINS THE SAME. RATHER THAN CODE A STACK PACKAGE FOR EACH TYPE OR SIZE, WE CAN CODE ONE GENERIC PACKAGE.

INSTRUCTOR NOTES

STRESS THAT GENERICS ARE A DESIGN ISSUE. DESIGNERS MUST LOOK FOR POTENTIAL REUSES OF CODE.

8-21

VG 823.1

GENERIC PROGRAM UNITS - BASIC IDEA

- PROVIDE EFFECTIVE REUSE OF COMMON CODE
- ARE PARAMETERIZED TEMPLATES OF A PROGRAM UNIT
- ENHANCE READABILITY
- BUILD LIBRARIES
- ARE TEMPLATES LIKE TYPES, TO CREATE AN 'OBJECT' AN INSTANCE OF EXECUTABLE CODE IS PRODUCED

INSTRUCTOR NOTES

SOME MAJOR USES OF GENERICS ARE ...

GENERIC

CAN BE USED TO

- WRITE ONCE MODULES WHOSE ALGORITHMS DON'T DIFFER
- PASS TYPES OR SUBPROGRAMS AS PARAMETERS TO PROGRAM UNITS
- DETAIL DESIGN OPTIONS WHILE DEFERRING DESIGN DECISIONS

INSTRUCTOR NOTES

VG 823.1

8-41

GENERIC UNITS - BASIC FORM

TEMPLATE:

```
generic
    -- GENERIC PARAMETERS TO PASS
    -- THESE ARE THE PARTS THAT DIFFER
    Generic_Package_Or_Subprogram_Specification;
    Generic_Package_Or_Subprogram_Body is
    -- THIS IS THE COMMON ALGORITHM
end;
```

INSTRUCTOR NOTES

REMAND STUDENT THAT THEY DON'T GET THIS CODE. IT'S JUST LIKE A TEMPLATE OR A MACRO.

GENERIC PARAMETERS SAY: I WILL PASS YOU THE TYPE OF THE Stack_Item TO PUT ON THE STACK
AND ITS EXACT SIZE.

GENERIC SPECIFICATION SAYS: YOU WILL BE ABLE TO PUSH AND POP Stack_Items OF THE STACK.

USE THIS SLIDE WITH NEXT SLIDE.

GENERIC UNIT - EXAMPLE

GENERIC PARAMETERS	{	<pre> generic Stack_Size : Natural; type Stack_Item is private; package Stack is type Stack_Type is private; procedure Push (X : in Stack_Item; On_To : in out Stack_Type); function Pop (Off_Of : Stack_Type) return Stack_Item; private type Stack_Type is array (1 .. Stack_Size) of Stack_Item; end Stack; package body Stack is ... procedure Push (X : in Stack_Item; On_To : in out Stack_Type) is ... end Push; function Pop (Off_Of : Stack_Type) return Stack_Item is ... end Pop; end Stack; </pre>
GENERIC PROGRAM UNIT SPECIFICATION	{	
GENERIC PROGRAM UNIT BODY	{	

INSTRUCTOR NOTES

RELATE THE GENERIC SPECIFICATION FROM THE PREVIOUS SLIDE TO THE EXAMPLES.

VG 823.1

8-61

GENERIC UNITS - USE

A GENERIC UNIT DESCRIBES A TEMPLATE. TO CREATE EXECUTABLE CODE FOR A SPECIFIC USE, WE DECLARE AN INSTANCE OF THE TEMPLATE CALLED AN INSTANTIATION.

SYNTAX:

```
package Package_Name is new Generic_Package_Name (Parameter_List);
```

```
procedure Procedure_Name is new Generic_Procedure_Name (Parameter_List);  
function Function_Name is new Generic_Function_Name (Parameter_List);
```

EXAMPLE:

```
type Real is digits 5;  
package Int_Stack is new Stack (100, Integer);  
package Real_Stack is new Stack (50, Real);  
  
subtype Small_Integer_Type is Integer range 1 .. 15;  
package Small_Int_Stack is new Stack (100, Small_Integer_Type);
```

INSTRUCTOR NOTES

GENERIC FORMAL PARAMETERS ARE WHAT WE CAN USE TO VARY THE PACKAGE OR SUBPROGRAM.

WE WILL LOOK AT EXAMPLES OF EACH AS WE LOOK AT EXAMPLES OF GENERIC USES.

GENERIC UNITS - FORMAL PARAMETERS

GENERIC PARAMETERS CAN BE

- GENERIC FORMAL OBJECTS
- GENERIC SUBPROGRAMS
- GENERIC TYPES

ACTUAL PARAMETERS IN A GENERIC INSTANTIATION ARE MATCHED TO THE FORMAL PARAMETERS OF THE GENERIC DEFINITION.

INSTRUCTOR NOTES

FOR LIMITED PRIVATE WE CAN PASS ANY TYPE BECAUSE YOU CAN ONLY DO THE OPERATIONS
SPECIFIED IN THE GENERIC SPECIFICATION.

VG 823.1

8-8i

GENERIC FORMAL PARAMETERS - TYPES

- SIX WAYS OF EXPRESSING A GENERIC FORMAL TYPE

FORMAL

type Item is limited private;

type Item is private;

type Item is (<>);

type Item is range <>;

type Item is delta <>;

type Item is digits <>;

ACTUAL

-- ANY TYPE

-- ANY TYPE FOR WHICH EQUALITY

-- AND INEQUALITY ARE DEFINED

-- ANY DISCRETE TYPE

-- ANY INTEGER TYPE

-- ANY FIXED POINT TYPE

-- ANY FLOATING POINT TYPE

INSTRUCTOR NOTES

POINT OUT:

1. WHERE THE GENERIC PARAMETERS, GENERIC SPECIFICATION ARE
2. WHAT PARAMETERS CAN BE PASSED

VG 823.1

8-91

GENERICIS FOR REUSABLE MODULES

```
generic
  Stack_Size : Natural;
  type Stack_Item is private;
  package Stack is
    procedure Push (E : in Stack_Item);
    function Pop return Stack_Item;
    ...
  end Stack;
```

-- GENERIC FORMAL OBJECT PARAMETER
-- GENERAL TYPE PARAMETER

INSTRUCTOR NOTES

EXAMPLES OF HOW WE COULD USE THE GENERIC STACK DEFINITION OF PREVIOUS SLIDE. RELATE THE ACTUAL TO FORMAL PARAMETERS. ONCE WE INSTANTIATE -- I.E. CREATE AN EXECUTABLE COPY -- WE CAN USE THE STACK.

TO USE THE GENERIC STACK

- A STACK OF INTEGERS

```
with Stack;
procedure Int_Stack_Example is
  N : Integer;
  package Integer_Stack is new Stack (20, Integer);  -- INSTANTIATION
begin -- Int_Stack_Example
  ...
  N := Integer_Stack.Pop;
end Int_Stack_Example;

-- TO USE
```

- A STACK OF RECORDS

```
with Stack;
procedure Rec_Stack_Example is
  type CPU_Resource_Record_Type is ...;
  CPU_Resource_Record : CPU_Resource_Record_Type;
  package Resource_Stack is new Stack (100, CPU_Resource_Record_Type);
begin
  ...
  Resource_Stack.Push (CPU_Resource_Record);
end Rec_Stack_Example;
```

INSTRUCTOR NOTES

8-111

VG 823.1

GENERICIS FOR REUSABLE MODULES

ADDITIONAL EXAMPLES:

- QUEUES (BUFFERS, MONITORS)
- SEARCH AND SORT ALGORITHMS
- MAN-MACHINE INTERFACES (E.G. MENU DRIVEN SYSTEMS)

INSTRUCTOR NOTES

POINT OUT:

1. FORMAL $\longleftrightarrow$ ACTUAL PARAMETERS
2. THIS EXAMPLE ILLUSTRATES THE USE OF SUBPROGRAMS AS PARAMETERS
3. THIS IS HOW WE CAN PASS ARRAYS AS GENERIC PARAMETERS
4. "N" IS THE < DEFINED FOR THE ARRAY INDEXED BY Index_Type OF COMPONENTS OF
TYPE ITEM

GENERIC TO PASS SUBPROGRAMS TO PROGRAM UNITS

generic

```
type Index_Type is (< >);  
type Item is private;  
type List_Type is array (Index_Type) of Item;  
with function "<" (X, Y : Item) return Boolean is "<";
```

```
procedure Sort (List : in out List_Type);
```

INSTRUCTOR NOTES

Integer_Sort USES THE DEFAULT "<" FUNCTION SPECIFIED IN THE TEMPLATE.

VG 823.1

8-131

TO USE THE GENERIC SORT - INSTANTIATION

- AN ARRAY OF INTEGERS

-- WITH THE FOLLOWING DECLARATIONS

type Index is range 1 .. 10;

type Integer_Array_Type is array (Index) of Integer;

-- THE INSTANTIATION

```
procedure Integer_Sort is new Sort (Index_Type => Index,
                                     Item      => Integer,
                                     List_Type => Integer_Array_Type);
```

INSTRUCTOR NOTES

THIS EXAMPLE IS AN IMPORTANT ONE. IT ILLUSTRATES NOT ONLY THE USE OF GENERICS BUT MORE IMPORTANTLY THE USE OF ADA TO STATE REQUIREMENTS. RESEARCH IN THIS AREA.

KEEP IN MIND THAT WE ARE GOING TO MODEL HARDWARE IN ADA. THE REQUIREMENTS ARE FIRMLY SPECIFIED. AT A LATER POINT WE CAN PARTITION FOR H/W OR S/W.

ESSENTIALLY A UART CONVERTS A BIT STREAM TO CHARACTERS AND VICE VERSA.

THIS EXAMPLE SPANS THREE SLIDES. INSTRUCTOR SHOULD BE FAMILIAR WITH THIS EXAMPLE BEFORE TRYING TO PRESENT IT.

GENERIC TO DETAIL DESIGN OPTIONS WHILE DEFERRING DESIGN DECISIONS

A UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER (UART) IS A CHIP THAT CONVERTS SERIAL OR PARALLEL INPUT INTO THE OPPOSITE, PARALLEL OR SERIAL OUTPUT. THE UART EXPECTS A CERTAIN OPERATING ENVIRONMENT WHICH LENDS ITSELF TO GENERIC PARAMETERS. A UART CAN BE SET FOR VARYING OPTIONS OF PARITY, BIT FORMAT, TRANSMIT AND RECEIVE BAUD RATES, VOLTAGES, SERIAL LINE REPRESENTATION. A UART COULD BE PART OF A COMMUNICATION SYSTEM CONSISTING OF HARDWARE AND SOFTWARE.

INSTRUCTOR NOTES

GO THROUGH CAREFULLY. THE COMMENTS EXPLAIN THE ACTUAL CODE.

TYPE Duration IS A PREDEFINED FIXED POINT TYPE (PACKAGE Standard) WHOSE VALUE IS EXPRESSED IN SECONDS.

VG 823.1

8-151

THE DESIGN COULD BE REPRESENTED AS

generic

```
-- Level is the number of bits in the character, indicating the
-- type of code, such as Ascii, Baudot, etc.
-- Receive_Baud_Rate indicates the incoming baud rate
-- Transmit_Baud_Rate indicates the outgoing baud rate
-- With_Parity indicates whether parity will be transmitted or
-- received. Either parity is present both for receive and for
-- transmit or it is absent.
-- Even_Parity indicates whether the parity will be even or odd.
-- The Type_Bit models the voltage levels used for logical 0 and
-- logical 1.
-- Async_Input_Bit_Stream models the hardwired line from which the
-- incoming bits will be read.
-- Async_Output_Bit_Stream models the hardwired line on which the
-- outgoing bits will be sent.
-- SPACE represents a logical 0.
-- MARK represents a logical 1. SPACE and MARK are the names
-- traditionally used in asynchronous communication to represent
-- these logical values.

Level : Natural range 5 .. 8;
Receive_Baud_Rate : in Duration;
Transmit_Baud_Rate : in Duration;
With_Parity : in Boolean;
Even_Parity : in Boolean;
Number_of_Stop_Bits : Natural range 1 .. 2;

type Bit is (<>);
with function Async_Input_Bit_Stream return Bit;
with procedure Async_Output_Bit_Stream (Xmit_Bit: in Bit);

Space : in Bit;
Mark : in Bit;

package UART is
-- package specifications
end UART;
```


INSTRUCTOR NOTES

IF YOU NEED A UART WITH DIFFERENT OPTIONS, JUST INSTANTIATE ANOTHER VERSION OF UART.
THE ESSENTIAL NATURE OR CHARACTERISTICS OF A UART HAVE BEEN CAPTURED IN THE GENERIC
DEFINITION.

NOTE THE "WITH UART" IS NEEDED TO HAVE ACCESS TO THE IDENTIFIER UART IN THE
INSTANTIATION.

UART INSTALLATION

WHEN THE UART DEVICE IS INSTALLED, THE PINS ARE SOLDERED WHICH ROUGHLY PARALLELS
A GENERIC INSTANTIATION, SUCH AS:

```
package My_UART_Specs is
  type Bit_Voltage is (Plus_12_V, Minus_12_V);
  function Pin_20_Serial_Input_Return Bit_Voltage;
  procedure Pin_25_Serial_Output (Xmit: Bit_Voltage);
end My_UART_Specs;

with UART, My_UART_Specs; use My_UART_Specs;
package My_UART is new UART (Level
  Receive_Baud_Rate
  Transmit_Baud_Rate
  With_Parity
  Even_Parity
  Number_of_Stop_Bits
  Bit
  Asynch_Input_Bit_Stream
  Asynch_Output_Bit_Stream
  Space
  Mark
  => 7,
  => 300.0,
  => 110.0,
  => True,
  => True,
  => 1,
  => Bit_Voltage,
  => Pin_20_Serial_Input,
  => Pin_25_Serial_Output,
  => Plus_12_V,
  => Minus_12_V);
```

INSTRUCTOR NOTES

VG 823.1

8-171

IN CONCLUSION

IN THIS WAY THE ACTUAL HARDWARE/SOFTWARE PARTITIONING CAN BE POSTPONED UNTIL THE DESIGN IS MORE COMPLETE.

THE DESIGN CAN BE USED FOR MANY CONFIGURATIONS OF THE ACTUAL COMMUNICATION SYSTEM AND THE DESIGN OPTIONS ARE EXPLICITLY REFLECTED IN THE ADA CODE.

INSTRUCTOR NOTES

PEOPLE TRANSITIONING FROM A LANGUAGE (SAY FORTRAN, PASCAL) THAT PROHIBITS THE PASSING OF DATA TYPES TO PROCEDURE MIGHT TRY TO IMPLEMENT IT VIA A VARIANT RECORD AS A PARAMETER RATHER THEN WITH THE DIRECT ADA FACILITY -- GENERIC SUBPROGRAMS.

SOME POTENTIAL PITFALLS - GENERICS

- USE OF ONE SUBPROGRAM WITH A CONTROL PARAMETER INDICATING THE TYPE OF THE ARGUMENTS, RATHER THAN A GENERIC SUBPROGRAM.

FOR EXAMPLE,

```
type Arg_Type is (Integer_Type, Real_Type, ...);
type Phony_Rec (Tag : Arg_Type) is
record
  case Tag is
    when Integer_Type => Integer_Stuff : Integer;
    when Real_Type => Real_Stuff : Real;
  ...
end case;
end record;
procedure Put (What_Type : Phony_Rec);
```

RATHER THAN,

```
generic
  type Arg_Type is private;
...
procedure Put (What_Type : Arg_Type);
```

INSTRUCTOR NOTES

ALLOW 5-10 MINUTES.

SOLUTION:

```
generic
type Index_Type is (<>); -- ANY DISCRETE TYPE
type Integer_Type is range (<>); -- Integer'Last .. Integer'First
type Array_Type is (Index_Type range <>) of Integer_Type;
function Sum (A : Array_Type) return Integer_Type;

function Sum (A : Array_Type) return Integer_Type is
    Result : Integer_Type := 0;
begin -- Sum
    for I in A'Range loop
        Result := Result + A(I);
    end loop;
    return Result;
end Sum;
```

PURPOSE:

1. WITHOUT GENERICS, A SEPARATE FUNCTION WOULD BE NEEDED FOR EACH ARRAY TYPE
2. ILLUSTRATE POINT MADE AT BEGINNING OF THIS SECTION, EFFECTIVE USE OF COMMON CODE.
3. GENERICS MAY BE MORE READABLE AND REFLECT THE REAL-WORLD SITUATION MORE CLOSELY.

VG 823.1

8-191

EXERCISE

WRITE A GENERIC FUNCTION THAT SUMS THE ELEMENTS OF A ONE
DIMENSIONAL ARRAY HAVING ANY INTEGER COMPONENT TYPE AND
ANY INDEX TYPE.

INSTRUCTOR NOTES

VG 823.1

9-1

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

Section 9

INPUT/OUTPUT

VG 823.1

INSTRUCTOR NOTES

THIS ALLOWS THE PROGRAMMER TO WRITE HIS/HER OWN I/O PACKAGES, FOR EXAMPLE A FORMATTING PACKAGE, OR FOR NON-STANDARD PERIPHERALS.

INPUT/OUTPUT

- ACCESSED THROUGH PACKAGES (PREDEFINED AND USER-DEFINED)
- USER HAS COMPLETE CONTROL OF I/O
- I/O IS NOT LIMITED TO COMPUTER PERIPHERALS

INSTRUCTOR NOTES

EXAMPLES OF I/O OPERATIONS: READ, WRITE, GET, PUT.

INPUT/OUTPUT - BASIC ASPECTS

- I/O IS BY NATURE MACHINE DEPENDENT
- PACKAGE SPECIFICATIONS PROVIDE THE INTERFACE FOR COMMON I/O OPERATIONS
- PACKAGE BODIES IMPLEMENT THE I/O OPERATION IN WAYS APPROPRIATE TO INDIVIDUAL COMPUTERS

INSTRUCTOR NOTES

I/O PACKAGES SUPPLIED WITH THE LANGUAGE ARE VERY PRIMITIVE.

BY TEXT FILES WE MEAN CHARACTERS - ASCII I/O.

I/O PACKAGES SUPPLIED BY LANGUAGE

- Text_IO - FOR I/O ON TEXTUAL (I.E. READABLE) FILES
- Sequential_IO - FOR I/O ON SEQUENTIAL ACCESS FILES (EXAMPLE: TAPES)
- Direct_IO - FOR I/O ON DIRECT ACCESS FILES (EXAMPLE: ISAM FILES)

INSTRUCTOR NOTES

SINCE WE CAN CREATE OUR OWN TYPES HOW CAN WE HAVE A PROCEDURE ALREADY AVAILABLE --
GENERIC'S ALLOWS US TO PASS THE EXACT TYPE AND CREATE ACTUAL I/O PACKAGES AS NEEDED.

VG 823.1

9-4i

TEXT_IO

- PERFORMS SIMPLE I/O OPERATIONS (Get, Put)
- TO USE TEXT_IO FACILITIES, MUST HAVE ACCESS TO THE PACKAGE FOR TEXT_IO
 - AT TOP OF COMPILATION UNIT:

with Text_IO; use Text_IO;
- PACKAGE TEXT_IO PROVIDES
 - I/O FOR Character AND String TYPES
 - GENERIC I/O TEMPLATES FOR Integer, Enumeration, Floating, and Fixed Point TYPES.

INSTRUCTOR NOTES

STEP THROUGH QUICKLY HOW I/O FOR NUMERIC TYPES WORKS.

THE CONCEPT HERE IS THAT I/O IS TYPE DEPENDENT.

USING TEXT_IO FOR NUMERIC TYPES

EXAMPLE:

```
with Text_IO; use Text_IO;      -- PROVIDES ACCESS TO GENERIC I/O TEMPLATES
procedure Do_Sum is
    type My_Integer is range 10 .. 50;
    Value_1, Value_2 : My_Integer;

    package My_Int_IO is new Integer_IO (My_Integer); -- INSTANTIATE A COPY TO
    use My_Int_IO;      -- PROVIDE I/O FOR OBJECTS OF TYPE
                        -- My_Integer

begin -- Do_Sum
    Get (Value_1);
    Get (Value_2);
    Put (Value_1 + Value_2);

end Do_Sum;
```

NOTE:

- FOR FLOATING POINT TYPES THE GENERIC PACKAGE IS Float_IO
- FOR FIXED POINT TYPES THE GENERIC PACKAGE IS Fixed_IO

INSTRUCTOR NOTES

GO THROUGH THIS EXAMPLE. MANAGERS NEED A FEEL FOR HOW THIS WORKS FOR ENUMERATION TYPES.

USING TEXT_IO FOR ENUMERATION TYPES

```
with Text_IO; use Text_IO;
procedure Name_Next_Month is
```

```
type Month_Type is (January, February, March, April, May, June, July, August,
September, October, November, December);
```

```
package Month_IO is new Enumeration_IO (Month_Type);
use Month_IO;
```

```
This_Month, Next_Month: Month_Type;
```

```
begin -- Name_Next_Month
```

```
Get (This_Month);
```

```
if This_Month = December then
```

```
    Next_Month := January;
```

```
else
```

```
    Next_Month := Month_Type'Succ (This_Month);
```

```
end if;
```

```
Put (Next_Month);
```

```
end Name_Next_Month;
```

```
-- Get IS AVAILABLE
```

```
-- Put IS AVAILABLE
```

INSTRUCTOR NOTES

VG 823.1

9-71

USING TEXT_IO FOR ARRAY AND RECORD TYPES

- NONE IS PREDEFINED (EXCEPT FOR String)
- MUST DO COMPONENT BY COMPONENT

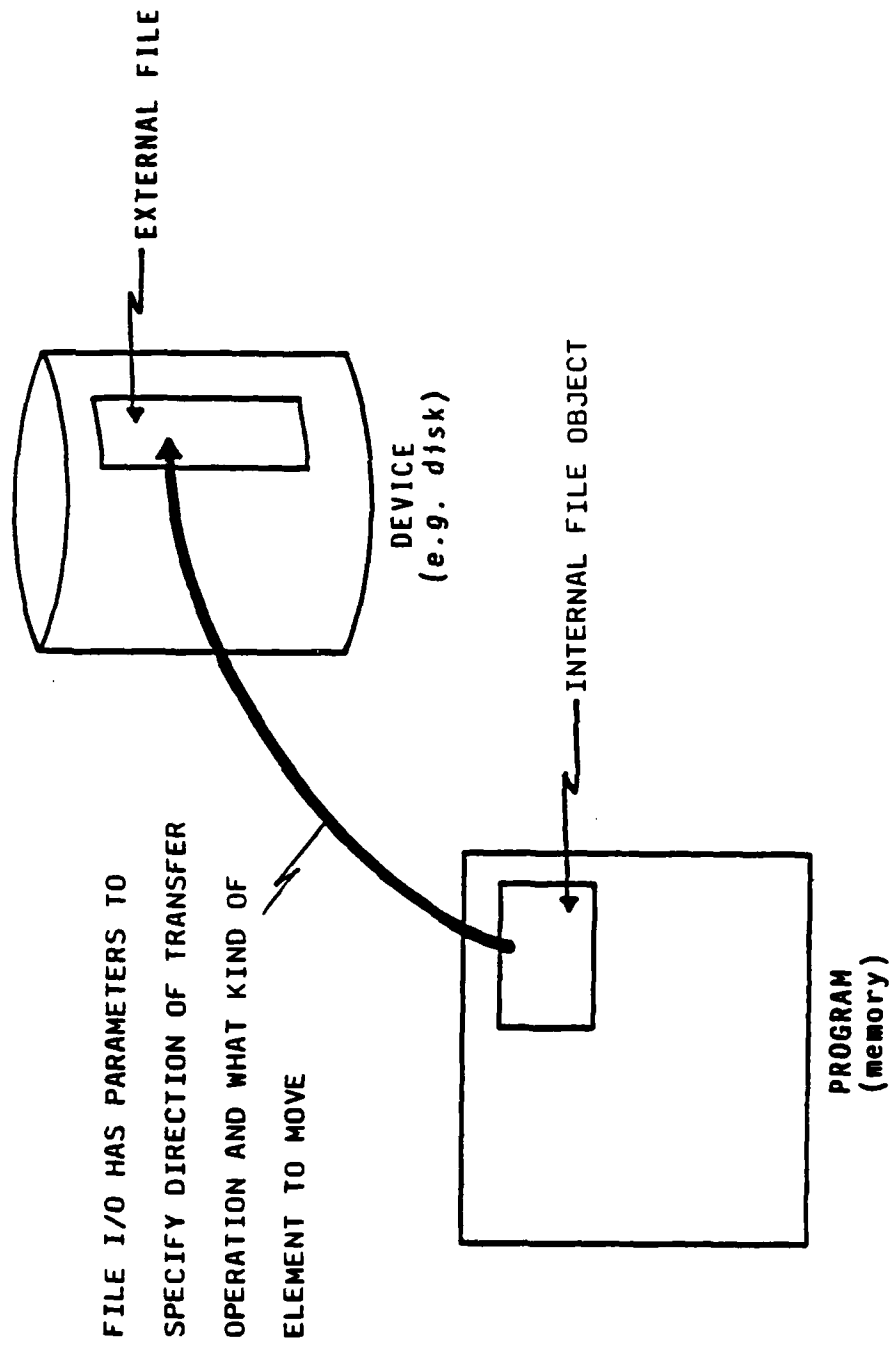
(STYLE HINT: DEFINE YOUR OWN PROCEDURES CALLED Put AND Get.)

INSTRUCTOR NOTES

THIS IS A SNAP-SHOT OF HOW FILE I/O IS VIEWED IN ADA. DON'T DWELL ON SLIDE. THE FOLLOWING IS INFORMATION PRIMARILY FOR THE INSTRUCTOR:

- EXTERNAL FILE - ANYTHING OUTSIDE THE PROGRAM THAT CAN PRODUCE A VALUE FOR INPUT OR ACCEPT A VALUE FOR OUTPUT.
- INTERNAL FILE - OBJECT WITHIN THE PROGRAM ABLE TO BE ASSOCIATED WITH AN EXTERNAL FILE.
- FILE - REFERS TO THE INTERNAL FILE
- OPEN FILE - INTERNAL FILE ASSOCIATED WITH AN EXTERNAL FILE
- CLOSED FILE - INTERNAL FILE NOT ASSOCIATED WITH AN EXTERNAL FILE

WHAT IS A FILE



INSTRUCTOR NOTES

CREATE

Create ESTABLISHES A NEW EXTERNAL FILE AND ASSOCIATES IT WITH AN INTERNAL FILE

OPEN

Open OPENS AN INTERNAL FILE BY ASSOCIATING IT WITH AN EXISTING EXTERNAL FILE

CLOSE

Close SEVERES THE ASSOCIATION BETWEEN INTERNAL AND EXTERNAL FILES

DELETE

Delete DELETES THE EXTERNAL FILE ASSOCIATED WITH THE GIVEN INTERNAL FILE AND
CLOSES THE INTERNAL FILE

RESET

Reset RESTARTS READING/WRITING AT THE BEGINNING OF THE FILE

BASIC FILE COMMANDS

- VARIOUS COMMANDS ARE AVAILABLE FOR BASIC FILE OPERATIONS. FOR EXAMPLE:

CREATE

OPEN

CLOSE

DELETE

RESET

INSTRUCTOR NOTES

VG 823.1

9-101

COMMANDS FOR FILE INFORMATION

- VARIOUS COMMANDS ARE AVAILABLE TO OBTAIN FURTHER FILE INFORMATION.

FOR EXAMPLE:

MODE OF FILE

NAME OF A FILE

IS A FILE OPEN?

END OF FILE

AD-A165 315

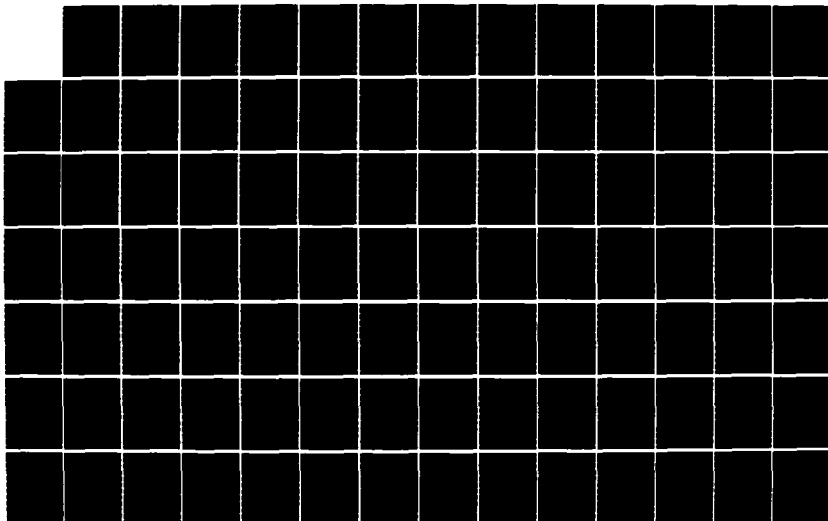
ADA (TRADEMARK) TRAINING CURRICULUM ADA (REGISTERED
TRADEMARK) FOR SOFTWARE MANAGERS L201 TEACHER'S GUIDE
VOLUME 2(U) SOFTECH INC WALTHAM MA 1986
DADB07-83-C-K506

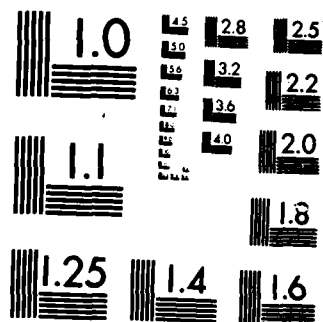
2/3

UNCLASSIFIED

F/G 5/9

NL





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

INSTRUCTOR NOTES

PROVIDE I/O FOR ARRAY AND RECORDS AS A WHOLE (I.E. NOT COMPONENT BY COMPONENT AS WITH
Text_IO) WHEN YOU DON'T NEED TO READ IT.

SEQUENTIAL_IO

- ALLOWS BINARY INPUT/OUTPUT ON SEQUENTIAL ACCESS FILES
- HAS A "CURRENT INDEX" WHICH IS SEQUENTIALLY INCREMENTED WHICH CANNOT BE INCREMENTED BY THE USER
- MAJOR OPERATIONS ARE Read AND Write
- IS A LIBRARY UNIT AND MUST BE IMPORTED VIA A with CLAUSE
EXAMPLE: with Sequential_IO;
- IS A GENERIC PACKAGE AND MUST BE INSTANTIATED WITH A type

EXAMPLE:

```
type Array_Type is array (1 .. 10) of Boolean;  
package Arr_IO is new Sequential_IO (Array_Type);  
use Arr_IO;
```

INSTRUCTOR NOTES

ALSO USE FOR COMPOSITE TYPE (ARRAYS, RECORDS) I/O.

VG 823.1

9-121

DIRECT_IO

- ALLOWS BINARY INPUT/OUTPUT ON DIRECT ACCESS FILES
- HAS A CURRENT INDEX WHICH CAN BE DIRECTLY CHANGED BY THE USER
- MAJOR OPERATIONS ARE Read AND Write
- IS A LIBRARY UNIT WHICH MUST BE IMPORTED VIA A with CLAUSE
- IS A GENERIC PACKAGE WHICH MUST BE INSTANTIATED WITH A TYPE

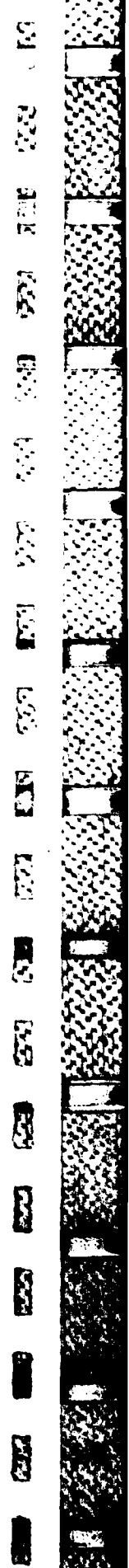
EXAMPLE: with Direct_IO;

EXAMPLE: type Array_Type is array (1 .. 10) of Boolean;
package Arr_IO is new Direct_IO (Array_Type);
use Arr_IO;

INSTRUCTOR NOTES

VG 823.1

9-131



I/O SUMMARY

- I/O PROVIDED AS PACKAGES
- THREE KINDS ARE PROVIDED WITH THE LANGUAGE
 - Text_IO,
 - Sequential_IO
 - Direct_IO
- USER CAN CREATE OWN I/O AS NEEDED

INSTRUCTOR NOTES

VG 823.1

9-141



SOME POTENTIAL PITFALLS - I/O

- I/O IN ADA IS VERY PRIMITIVE

FOR EXAMPLE:

IT WILL BE VERY DIFFICULT TO DEBUG REPORT GENERATION PROGRAMS

NOTE: MOST MANAGERS WILL MOST LIKELY HAVE THEIR OWN I/O PACKAGES DEFINED

INSTRUCTOR NOTES

VG 823.1

10-1

Section 10

EXCEPTIONS

VG 823.1

INSTRUCTOR NOTES

VG 823.1

10-11

EXCEPTIONS -- BASIC IDEA

- AN EXCEPTION STOPS SEQUENTIAL EXECUTION WHEN A PARTICULAR CONDITION IS REACHED, AND TRANSFERS CONTROL TO SOME KNOWN LOCATION WHERE THE CONDITION MAY BE HANDLED
- A MECHANISM FOR FAULT-TOLERANT PROGRAMMING
 - ALTERNATIVE TO EXPLICIT ERROR CODE PARAMETERS
 - LANGUAGE ALLOWS DIRECT REPRESENTATION OF REAL WORLD ALGORITHM
- PREDEFINED AND USER-DEFINED EXCEPTIONS
- SUPPORTS RELIABILITY

INSTRUCTOR NOTES

VG 823.1

10-21

THE
NEW
YORK
PUBLIC
LIBRARY
ASTEN LENOX TILDEN FOUNDATIONS
100 N. YERGES ST.

EXCEPTIONS

CAN BE USED TO

- DETECT AND RECOVER FROM EXCEPTIONAL OR ERROR CONDITIONS
- PERFORM "CLEANUP" ACTIONS FOR HARDWARE MALFUNCTIONS

VG 823.1

10-2

INSTRUCTOR NOTES

MOTIVATION FOR EXCEPTION HANDLING ALTERNATIVES.

EXCEPTION EXAMPLE

- WHEN DIVISOR IS ZERO AN EXCEPTION (CALLED Numeric_Error) WILL OCCUR

```
procedure Example (Dividend, Divisor      : in Float;  
                  Partial_Result, Final_Result : out Float) is
```

```
    . . .
```

```
begin -- Example
```

```
    . . .
```

```
    Partial_Result := Dividend/Divisor;
```

```
    . . .
```

```
    Final_Result := Partial_Result + 1.0;
```

```
end Example;
```

```
with Example;
```

```
procedure Main is
```

```
    A, B, C, D : Float;
```

```
    . . .
```

```
begin -- Main
```

```
    . . .
```

```
    Example (A, B, C, D);
```

```
    . . .
```

```
end Main;
```


INSTRUCTOR NOTES

WHEN THE ERROR IN THE EXAMPLE OCCURS, WHAT CAN WE DO ...

VG 823.1

10-41

WHAT COULD BE DONE?

HANDLE THE EXCEPTIONS LOCALLY IN Example: SET Partial_Result AND
Final_Result TO Float'Last AND RETURN TO PROCEDURE Main AS IF NOTHING
UNUSUAL HAD HAPPENED

OR

TERMINATE EXECUTION OF PROCEDURE Example AND MAKE SITUATION KNOWN TO
Main SO THAT IT CAN TAKE CORRECTIVE ACTION

OR

PASS THE PROBLEM TO THE OPERATING SYSTEM AND LET IT DO SOMETHING LIKE
TERMINATE THE PROGRAM WITH AN ERROR MESSAGE

- NONE OF THE ABOVE CHOICES IS RIGHT FOR EVERY APPLICATION

SO

- ADA ALLOWS THE PROGRAMMER TO CONTROL THE HANDLING OF EXCEPTIONS

INSTRUCTOR NOTES

VG 823.1

10-5i

IN GENERAL

- TO CONTROL THE HANDLING OF EXCEPTIONS WE'D LIKE TO
 - 1) NAME EXCEPTIONAL SITUATIONS
 - 2) CALL ATTENTION SOMEWHERE TO THE FACT THAT AN EXCEPTIONAL SITUATION OCCURRED
 - 3) ATTEMPT TO CORRECT THE SITUATION

- THESE ACTIONS ARE MAPPED TO ADA EXCEPTION CONSTRUCTS AS FOLLOWS:

- 1) EXCEPTION DECLARATION
- 2) raise STATEMENT
- 3) EXCEPTION HANDLER

INSTRUCTOR NOTES

PREDEFINED EXCEPTIONS MEAN THE NAME OF THE EXCEPTIONAL EVENT AND HOW IT IS MADE KNOWN (I.E. RAISED) ARE DETERMINED BY THE LANGUAGE.

BE AWARE THAT NOT ALL THE PREDEFINED EXCEPTIONS MUST BE RAISED (E.G. NUMERIC ERROR) BY AN IMPLEMENTATION.

ONLY EXPLAIN THE PREDEFINED EXCEPTIONS FOR CONSTRAINT, NUMERIC, PROGRAM AND DATA.
EXAMPLES:

- CONSTRAINT: TRYING TO ASSIGN 10 TO A VARIABLE DECLARED AS
 X : Integer range 0 .. 5;

- PROGRAM: TRY TO EXIT A FUNCTION OTHER THAN BY A RETURN OR
 ANOTHER EXCEPTION

```
begin
  if ... then
    return ...;
  else
    ...      -- no return
  end;
```

PREDEFINED LANGUAGE EXCEPTIONS

THERE EXIST FIVE PREDEFINED LANGUAGE EXCEPTIONS:

- Constraint_Error - OBJECT VALUES OUTSIDE RANGE OR INDEX CONSTRAINT
- Numeric_Error - NUMERIC OPERATION GIVES AN INCORRECT MATHEMATICAL RESULT
- Program_Error - TRYING TO ACTIVATE OR USE A PROGRAM UNIT THAT IS NOT ACTIVATED
- Storage_Error - INSUFFICIENT MEMORY TO ALLOCATE AN OBJECT OR PROGRAM UNIT
- Tasking_Error - TASK RELATED ERROR

EXCEPTIONS FOR I/O EXIST. FOR EXAMPLE:

- Data_Error - DATA OF THE INCORRECT TYPE IS ENTERED

AN EXCEPTION HANDLER IS AN ATTEMPT TO TAKE SOME CORRECTIVE ACTION FOR AN EXCEPTIONAL EVENT.

10-7i

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
84

EXCEPTION HANDLER

AN EXCEPTION HANDLER OCCURS IN A "FRAME"

```
begin
    -- sequence of statements
exception
    -- exception handlers
end;
```

FRAME

INSTRUCTOR NOTES

FIRST SHOW FRAME SYNTAX THEN STEP THROUGH WHAT HAPPENS IF AN EXCEPTION IS RAISED.

BUT LET'S SAY WE WANT TO EXECUTE THE STATEMENTS INDICATED BY THE '...' EVEN IF THE EXCEPTION IS RAISED. WHAT CAN WE DO? (NEXT SLIDE).

EXCEPTION HANDLING EXAMPLE

```
with Text_IO; use Text_IO;
procedure Expand_Command is
...
begin -- Expand_Command
    Get (Input_Character);
    ...
    exception
        when Data_Error => Put_Line ("Invalid Entry");
end Expand_Command;

-- THESE STATEMENTS NOT EXECUTED IF
-- Data_Error IS RAISED
```

INSTRUCTOR NOTES

VG 823.1

10-91

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

BLOCK STATEMENT

- BLOCKS PROVIDE A MECHANISM FOR LOCALIZING EXCEPTION HANDLER(S)
TO A STATEMENT OR SEQUENCE OF STATEMENTS

SYNTAX:

```
[Block_Name:]  
[declare  
    local declarations;]  
begin -- Block_Name  
    sequence_of_statements;  
[exception  
    exception_handler;  
    {exception_handler;}]  
end [Block_Name];
```

GO THROUGH WHAT HAPPENS WITH THIS VERSION.

VG 823.1

10-10i

EXAMPLE WITH BLOCK

```
with Text_IO; use Text_IO;
procedure Expand_Command is
...
begin -- Expand_Command

    loop -- additional data entry tries allowed for incorrectly entered data

        begin

            Get (Input_Character);

            exit; -- when all goes well.

        exception

            when Data_Error => Put_Line ("Invalid Entry");

                               Put_Line ("Try Again");

        end;

    end loop;

    ... -- only executed when valid data is entered

end Expand_Command;
```

INSTRUCTOR NOTES

VG 823.1

10-111

USER-DEFINED EXCEPTIONS

EXCEPTION DECLARATION

SYNTAX:

```
Exception_Name {,Exception_Name} : exception;
```

EXAMPLES:

```
Division_By_Zero      : exception;  
CRC_Failure, Sensor_Off : exception;
```


INSTRUCTOR NOTES

VG 823.1

10-121

RAISE STATEMENT

- PREDEFINED EXCEPTIONS ARE AUTOMATICALLY RAISED BY THE LANGUAGE (OR Text_IO)
- TO RAISE USER DEFINED EXCEPTIONS USE THE raise STATEMENT

SYNTAX:

```
raise [Exception_Name];
```

EXAMPLES:

```
raise Division_By_Zero;  
raise;           -- only in a handler  
raise Sensor_Off;
```

INSTRUCTOR NOTES

DON'T GO THROUGH EXAMPLE, JUST SHOW THE USE AND MEANING OF THE BAR IN AN EXCEPTION
HANDLER SO THEY WOULD UNDERSTAND IT IF THEY SAW IT IN AN EXAMPLE. DO THE SAME FOR THE
'others' OPTION.

VG 823.1

10-131

ALTERNATE NOTATION CHOICES IN HANDLERS

SYNTAX:

```
when exception_choice ||exception_choice| =>  
    -- sequence of statements
```

EXAMPLE:

```
begin  
    ...  
exception  
    when Division_By_Zero =>  
        -- sequence of statements  
        -- to be executed when Divisor = 0.0  
    when CRC_Failure | Sensor_Off =>  
        -- sequence of statements to be executed  
        -- whenever the exceptions CRC_Failure or  
        -- Sensor_Off are raised  
    when others =>  
        -- sequence of statements to be executed when an  
        -- exception other than one of those listed above  
        -- is raised  
end;
```

INSTRUCTOR NOTES

THIS IS THE TYPE OF EXCEPTION HANDLING WE'VE LOOKED AT SO FAR SO STEP THROUGH QUICKLY.

VG 823.1

10-141

EXCEPTION HANDLING CONTROL FLOW

LOCAL HANDLING

```
procedure Example ( Dividend, Divisor      : in Float;
                    Partial_Result, Final_Result : out Float) is
    Division_By_Zero : exception;      -- exception declaration
begin -- Example
    . . .
    if Divisor = 0.0
    then
        raise Division_By_Zero;      -- raising the exception
    else
        Partial_Result := Dividend/Divisor;
    end if;
    . . .
    Final_Result := Partial_Result + 1.0;
exception
    when Division_By_Zero =>
        Partial_Result := Float'Last;
        Final_Result   := Float'Last;
end Example;
```

INSTRUCTOR NOTES

WHEN NEEDED, WE CAN MAKE THE EXCEPTIONAL SITUATION KNOWN TO A 'hyphen' PROGRAM LEVEL (IN
ADA JARGON, PROPAGATED).

STEP THROUGH THE CONTROL FLOW CAREFULLY.

EXCEPTION HANDLING CONTROL FLOW

PROPAGATED TO CALLER

```
procedure Example ( Dividend, Divisor      : in Float;  
                   Partial_Result, Final_Result : out Float) is  
    Division_By_Zero : exception;          -- exception declaration  
begin -- Example  
    if Divisor = 0.0  
    then  
        raise Division_By_Zero;           -- raising the exception  
    else  
        Partial_Result := Dividend/Divisor;  
    end if;  
    . . . -- calculations  
end Example;                             -- no exception handler so  
                                           -- exception propagated
```

```
with Example;  
procedure Main is  
    A, B, C, D : Float;  
begin -- Main  
    . . .  
    Example (A, B, C, D);                -- these will not be executed if  
    . . .                                -- Division_By_Zero is raised in Example.  
exception  
    when others =>  
        . . . -- whatever                -- exception handler  
end Main;
```


INSTRUCTOR NOTES

THIS SUMMARIZES THE FLOW OF CONTROL WHEN HANDLING EXCEPTIONS. DON'T GO THROUGH. IT'S
HERE FOR FUTURE REFERENCE.

VG 823.1

10-16i

EXCEPTION HANDLING CONTROL FLOW

SUMMARY

- WHEN AN EXCEPTION IS RAISED, NORMAL PROGRAM EXECUTION IS SUSPENDED
- IF THERE IS A HANDLER FOR THE EXCEPTION IN THE INNERMOST FRAME, CONTROL IS TRANSFERRED TO IT
- IF THERE IS NO HANDLER, THE EXCEPTION IS PROPAGATED UP TO THE FRAME THAT CAUSED THE FRAME INCURRING THE EXCEPTION TO BE INVOKED -- THE CALLER IN THE CASE OF A SUBPROGRAM OR THE SURROUNDING FRAME IN THE CASE OF A BLOCK
- IF A HANDLER IS FOUND THERE IT IS EXECUTED. IF NOT, THE EXCEPTION IS PROPAGATED UP ONE MORE LEVEL
- IF NO HANDLER IS FOUND IN THE PROGRAM, THE UNDERLYING OPERATING SYSTEM TERMINATES THE PROGRAM (AND CAN TAKE OTHER ACTIONS SUCH AS ISSUING ERROR MESSAGES)
- WHEN A HANDLER IS FOUND, THE EXECUTION OF THE HANDLER REPLACES EXECUTION OF THE FRAME CONTAINING THE HANDLER (AND ANY LOWER LEVEL FRAMES)

INSTRUCTOR NOTES

HERE IS A REAL-TIME EXAMPLE OF THE USE OF EXCEPTIONS. EXAMPLE IS FROM A COMMUNICATION SYSTEM WHICH RECEIVES, PROCESSES, AND THEN ROUTES TO THE APPROPRIATE DESTINATIONS THE MESSAGE. ALGORITHM: A MESSAGE NODE IS SEIZED AND LOCKED DURING MESSAGE TRANSMISSION.

NOTE: WE ASSUME THAT Message_Error CAN ONLY BE RAISED BY Send_Message.

STEP THROUGH EXAMPLE AS THIS IS THE BASIS OF THE EXERCISE.

EXCEPTIONS TO DETECT AND RECOVER FROM EXCEPTIONAL OR ERROR CONDITIONS

```
begin
  loop
    if Receiving_A_Message then
      Process_Message;
    elsif Sending_A_Message then
      Seize (Message_Node);
      Send_Message (Message);
      Free (Message_Node);
    end if;
  end loop;
exception
  when Message_Error =>
    Free (Message_Node);
    Notify_Operator;
end;
```

-- Message_Error raised here

INSTRUCTOR NOTE

SOLUTION

```
loop
  if ... then ...
  elsif ... then
    Seize (Message_Node);
  begin
    Send_Message (Message_Node);
    Free_(Message_Node);
  exception
    when others = Free (Message_Node);
    Notify_Operator;
  end;
end if;
end loop;
```

NOTE: BULLET ONE IMPLIES THAT SEIZE COULD RAISE AN EXCEPTION. WE MIGHT NOT WANT TO FREE A NODE THAT HAD NOT REALLY BEEN SEIZED. THIS IS WHY WE WOULD NOT WANT SEIZE TO BE WITHIN THE BLOCK WITH THE EXCEPTION HANDLER. THE OTHERS IS USED IN THE HANDLER RATHER THAN Message_Error TO SATISFY BULLET TWO. Send_Message COULD RAISE OTHER EXCEPTIONS. BUT WHATEVER THE ERROR WE ALWAYS WANT TO FREE THE NODE.

EXERCISE

ASSUMING THAT THE SOURCE OF ALL POSSIBLE EXCEPTIONS IS KNOWN IS NOT A SAFE PRACTICE.

MODIFY THE PREVIOUS EXAMPLE SO THAT

- Free IS ONLY CALLED WHEN THE NODE HAS BEEN SEIZED
- Free IS CALLED NO MATTER WHAT EXCEPTION IS RAISED BY Send_Message

INSTRUCTOR NOTES

VG 823.1

10-191

EXCEPTIONS TO PERFORM "CLEANUP" AFTER HARDWARE MALFUNCTIONS

```
procedure Tape_Read (Tape : in Tape_File_Type;  
                    Disk : out Disk_File_Type) is  
    ...  
    Read_Error : exception;  
    ...  
begin -- Tape_Read  
    -- Read_Error raised on a H/W misread  
    ...  
exception  
    when Read_Error => Backspace (Tape);  
    Re_Read (Tape);  
end Tape_Read;
```


INSTRUCTOR NOTES

THIS SECTION IS VERY IMPORTANT). EXCEPTIONS CAN BE USED/MISUSED IN SO MANY WAYS. THEIR USE SHOULD AGAIN BE WITH DELIBERATE INTENT AND PLANNED INTO THE SYSTEM DESIGN (NOT ADDED AT THE LAST MINUTE WHEN ERRORS START OCCURRING).

GENERAL ISSUES WITH EXCEPTION HANDLERS

- AN "EASY" FIX WHICH AFFECTS ONLY LOCAL OBJECTS SHOULD BE HANDLED LOCALLY (E.G.: DATA_ERROR)

- A CATASTROPHIC FAILURE MIGHT NEED TO BE PROPAGATED TO AND HANDLED AT A HIGHER LEVEL

EXAMPLE:

A TAPE READ ERROR MAY WISH TO BACKSPACE AND TRY TO REREAD THE TAPE (LOCAL FIX), OR IF THAT FAILS, MAY WANT TO INITIATE A RECONFIGURATION (GLOBAL FIX)

- COMPLETELY UNEXPECTED EXCEPTIONS REVEAL A SOFTWARE ERROR AND SHOULD BE TREATED AT A HIGHER LEVEL

10-21i

GENERAL ISSUES WITH EXCEPTION DECLARATIONS

- TYPICALLY, A PACKAGE SPECIFICATION DECLARES EXCEPTIONAL SITUATIONS
THE PACKAGE OPERATIONS IN THE BODY RAISE THE EXCEPTIONS

- ONLY DECLARE LOCALLY A CONDITION THAT COULD BE HANDLED LOCALLY
(RARELY IS THIS THE CASE.)

- BY USING USER DEFINED EXCEPTIONS CAN DETERMINE WHAT CONDITION
TRIGGERED IT

INSTRUCTOR NOTE

VC 823.1

10-22i

GENERAL ISSUES (Continued)

- EXCEPTIONS ARE A HIGH-LEVEL DESIGN ISSUE
- APPLICATION SPECIFIC OR ORGANIZATION SPECIFIC POLICIES FOR DESIGNER AND PROGRAMMERS ARE NECESSARY
- RULE OF THUMB: EXCEPTIONS EXIST TO PROVIDE AN ADDED LEVEL OF PRECAUTION, NOT AN OPPORTUNITY FOR PLAYING GAMES WITH CONTROL FLOW.
 - AT MODULE INTERFACES, THE CALLED MODULE SHOULD "RETURN" AN EXCEPTION FOR ALL INVALID CALLS.
 - EXCEPTION HANDLERS SHOULD BE USED TO PREVENT UNEXPECTED MODULE EXIT AT CRITICAL TIMES. E.G., A PROGRAM UNIT THAT OPENS A FILE SHOULD ALWAYS HAVE AN EXCEPTION HANDLER THAT CLOSES THE FILE ... JUST IN CASE.

INSTRUCTOR NOTES

THESE TECHNICAL MANAGERS ARE IN THE POSITION TO SET THESE POLICIES -- AS SUCH THEY NEED TO BE AWARE OF THE PROBLEMS. THERE ARE NO SET ANSWERS FOR A GIVEN APPLICATION, UNFORTUNATELY.

VG 823.1

10-231

SOME POTENTIAL PITFALLS - EXCEPTIONS

- ASSUMING THAT AN EXCEPTION COMES FROM ONLY A KNOWN PLACE

```
procedure Search (S: String; C: Character) is
  Position : Integer := S'First;
begin -- Search
  while S(Position) /= C loop
    Position := Position + 1;
  end loop;
  Found (Where => Position);
exception
  when Constraint_Error => Not_Found;
end Search;
```

NOTE: THE PROGRAMMER ASSUMES THAT Constraint_Error WILL BE RAISED BY Position INDEXING OUT OF THE BOUNDS OF S. IN FACT, IT COULD BE UNEXPECTEDLY RAISED BY Found AND PROPAGATED BACK TO Search (IF THERE IS NO HANDLER IN Found).

INSTRUCTOR NOTES

VG 823.1

10-241

PITFALLS (Continued)

- USING EXCEPTIONS TO SIMULATE (HIDE) A GOTO. (SYMPTOM: EXCEPTION RAISED AND HANDLED IN SAME UNIT)
- NOT PERFORMING ADEQUATE ERROR CHECKING AT MODULE INTERFACES
- NOT PROVIDING EXCEPTION HANDLERS FOR MODULES THAT ALLOCATE SYSTEM-WIDE RESOURCES
- USING others IN AN EXCEPTION HANDLER AS A CATCH-ALL

INSTRUCTOR NOTE

THIS IS A CONTROVERSIAL USE OF EXCEPTIONS. MANAGER MAY HAVE TO DECIDE ON THE APPROPRIATE USE FOR THEIR ORGANIZATION OR PROJECT. THIS COULD BE CODED AS

```
begin -- Tomorrow
  if D /= Days'Last then
    return Days'Succ (D);
  else
    return Days'First;
  end Tomorrow;
```

THE ISSUES ARE:

1. WHICH IS CLEARER CODE (FOR UNDERSTANDING ITS FUNCTIONING) FOR THE MAINTAINER
2. ARE EXCEPTIONS ONLY FOR EXCEPTIONAL SITUATIONS

PITFALLS (Continued)

- USING EXCEPTIONS TO DETECT AND RECOVER FROM EXPECTED CONDITIONS

```
... type Days is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
... function Tomorrow (D : Days) return Days is
begin -- Tomorrow
    return Days'Succ (D);
exception
    when Constraint_Error =>
        return Days'First; -- Constraint_Error Raised
end Tomorrow;
```

INSTRUCTOR NOTES

VG 823.1

11-1

Section 11

STUBBING

VG 823.1

INSTRUCTOR NOTES

VG 823.1

11-14

3-11 0261 0259 0258 0257 0256 0255 0254 0253 0252 0251 0250 0249 0248 0247 0246 0245 0244 0243 0242 0241 0240 0239 0238 0237 0236 0235 0234 0233 0232 0231 0230 0229 0228 0227 0226 0225 0224 0223 0222 0221 0220 0219 0218 0217 0216 0215 0214 0213 0212 0211 0210 0209 0208 0207 0206 0205 0204 0203 0202 0201 0200 0199 0198 0197 0196 0195 0194 0193 0192 0191 0190 0189 0188 0187 0186 0185 0184 0183 0182 0181 0180 0179 0178 0177 0176 0175 0174 0173 0172 0171 0170 0169 0168 0167 0166 0165 0164 0163 0162 0161 0160 0159 0158 0157 0156 0155 0154 0153 0152 0151 0150 0149 0148 0147 0146 0145 0144 0143 0142 0141 0140 0139 0138 0137 0136 0135 0134 0133 0132 0131 0130 0129 0128 0127 0126 0125 0124 0123 0122 0121 0120 0119 0118 0117 0116 0115 0114 0113 0112 0111 0110 0109 0108 0107 0106 0105 0104 0103 0102 0101 0100 0099 0098 0097 0096 0095 0094 0093 0092 0091 0090 0089 0088 0087 0086 0085 0084 0083 0082 0081 0080 0079 0078 0077 0076 0075 0074 0073 0072 0071 0070 0069 0068 0067 0066 0065 0064 0063 0062 0061 0060 0059 0058 0057 0056 0055 0054 0053 0052 0051 0050 0049 0048 0047 0046 0045 0044 0043 0042 0041 0040 0039 0038 0037 0036 0035 0034 0033 0032 0031 0030 0029 0028 0027 0026 0025 0024 0023 0022 0021 0020 0019 0018 0017 0016 0015 0014 0013 0012 0011 0010 0009 0008 0007 0006 0005 0004 0003 0002 0001 0000

STUBBING

CAN BE USED TO

- INCREASE UNDERSTANDABILITY OF COMPLEX CODE AND DESIGN
- LOCALIZE LIBRARY UNIT INFORMATION
- DECREASE RECOMPILATION COSTS

11-2i

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1

STUBBING BY EXAMPLE

NESTED

```
procedure Calculate_Median (...) is
--
-- local declarations
--
procedure Sort (...) is
--
-- local declarations
--
begin -- Sort
...
end Sort;
begin -- Calculate_Median
...
end Calculate_Median;
```

STUB

```
procedure Calculate_Median (...) is
--
-- local declarations
--
procedure Sort (...) is separate;
begin -- Calculate_Median
...
end Calculate_Median;
```

SUBUNIT

```
separate (Calculate_Median)
procedure Sort (...) is
...
begin -- Sort
...
end Sort;
```

INSTRUCTOR NOTES

VG 823.1

11-3i

STUBS AND SUBUNITS

- AT THE POINT WHERE A SUBPROGRAM BODY OR PACKAGE BODY WOULD NORMALLY APPEAR IN A COMPILATION, A BODY STUB MAY BE USED INSTEAD

procedure Subprogram_Name is separate;

THIS IMPLIES THAT THE ACTUAL BODY WILL BE SUPPLIED IN A SEPARATE SUBUNIT.

- THE SUBUNIT IS SUPPLIED WITH A PREFIX NAMING THE COMPILATION UNIT WHERE THE CORRESPONDING BODY STUB APPEARED

separate (Parent_Unit) -- note no semicolon
procedure Subprogram_Name is -- body

- ALTHOUGH THE SUBUNIT IS SEPARATELY COMPILED THE EFFECT IS EXACTLY AS IF THE ACTUAL BODY WERE GIVEN AT THE POINT OF THE BODY STUB

ADA PROVIDES TWO MECHANISMS FOR PROGRAM DEVELOPMENT

- TOP-DOWN
 - LARGE COHERENT PROGRAM BROKEN DOWN INTO SEPARATELY-COMPILED SUBUNITS
 - SUBUNITS COMPILED AFTER UNIT ON WHICH THEY DEPEND
 - MECHANISM IMPLEMENTED USING "...is separate" AND "separate (...)" NOTATION
- BOTTOM-UP
 - TYPICAL APPLICATION IS A "LIBRARY" OF SUBPROGRAMS WRITTEN FOR GENERAL USE (PACKAGES)
 - THESE ARE WRITTEN AND COMPILED BEFORE UNITS THAT USE THEM
 - UNITS THAT DEPEND ON THEM GET ACCESS VIA with AND use CLAUSES

INSTRUCTOR NOTES

VG 823.1

11-5i

PROGRAM LIBRARY

- EVERY ADA PROGRAM HAS AN ASSOCIATED PROGRAM LIBRARY
- CERTAIN COMPILATION UNITS (PACKAGE DECLARATIONS, SUBPROGRAM DECLARATIONS, AND SUBPROGRAM BODIES WHEN THERE IS NO CORRESPONDING SUBPROGRAM DECLARATION) ARE LIBRARY UNITS
- AS LIBRARY UNITS ARE COMPILED, THEIR DECLARATIONS (NOT OBJECT CODE) GO INTO THE PROGRAM LIBRARY
- LIBRARY UNITS PROVIDE INTERFACE INFORMATION AND OTHER SPECIFICATION DATA NEEDED BY THE COMPILER TO PROCESS OTHER UNITS.

INSTRUCTOR NOTES

IF UNIT A MUST BE COMPILED BEFORE UNIT B, THEN IF A IS RECOMPILED B MUST BE RECOMPILED
(UNLESS A SMART COMPILER CAN DETERMINE THAT ANY CHANGES MADE TO A DO NOT AFFECT B).

COMPILATION ORDER

AS A GENERAL RULE:

A GIVEN UNIT MUST BE COMPILED AFTER ANY UNITS CONTAINING INFORMATION UPON WHICH IT DEPENDS.

FOR EXAMPLE:

- A COMPILATION UNIT MUST BE COMPILED AFTER ALL LIBRARY UNITS NAMED BY ITS WITH CLAUSE
- A SUBPROGRAM OR PACKAGE BODY MUST BE COMPILED AFTER THE CORRESPONDING SUBPROGRAM OR PACKAGE SPECIFICATION
- A SUBUNIT MUST BE COMPILED AFTER ITS PARENT COMPILATION UNIT

INSTRUCTOR NOTES

UNDERLINE THE KEY PROGRAM UNITS OF THE EXAMPLE FOR SIMPLIFICATION.

VG 823.1

11-7i

SEPARATE COMPILE EXAMPLE --

SUBUNITS

```
• SINGLE COMPILE UNIT
with Text_IO;
procedure Top is
  type Real is digits 10;
  R, S : Real := 1.0;
  package Facility is
    PI : constant := 3.14159 26536;
    function F (X : Real) return Real;
    procedure G (V, Z : Real);
  end Facility;
  package body Facility is
    -- some local declarations followed by
    function F (X : Real) return Real is
    begin -- F
      -- sequence of statements of F
      ...
    end F;
    procedure G (V, Z : Real) is
      -- local procedures using Text_IO
      ...
    begin -- G
      -- sequence of statements of G
      ...
    end G;
  end Facility;
  procedure Transform (U : in out Real) is
  use Facility;
  begin -- Transform
    U := F(U);
    ...
  end Transform;
begin -- Top
  Transform (R);
  ...
  Facility.G (R, S);
end Top;
```

INSTRUCTOR NOTES

VG 823.1

11-81

SEPARATE COMPILATION EXAMPLE --

SUBUNITS (Continued)

```
• FOUR COMPILATION UNITS

  procedure Top is
    type Real is digits 10;
    R, S : Real := 1.0;
  package Facility is
    Pi : constant := 3.14159 26536;
    function F (X : Real) return Real;
    procedure G (Y, Z : Real);
  end Facility;
  package body Facility is separate;
  procedure Transform (U : in out Real) is separate;

1  begin -- Top
    Transform (R);
    ...
    Facility.G (R, S);
  end Top;
-----

  separate (Top)
  procedure Transform (U : in out Real) is
    use Facility;
2  begin -- Transform
    U := F (U);
    ...
  end Transform;
-----
```

INSTRUCTOR NOTES

NOTE THAT THE LIBRARY UNIT Text_IO IS USED BY PROCEDURE 9 ONLY. BY USING STUBS WE CAN LOCALIZE THE AMOUNT INFORMATION IMPORTED.

VG 823.1

11-9i

SEPARATE COMPILATION EXAMPLE

FOUR COMPILATION UNITS (Continued)

```
3      separate (Top)
        package body Facility is
            -- some local declarations followed by
            function F (X : Real) return Real is
            begin -- F
                -- sequence of statements of F
                ...
            end F;
            procedure G (Y, Z : Real) is separate;
        end Facility;
```

```
-----
4      with Text_IO;
        separate (Top.Facility) -- full name of Facility
        procedure G (Y, Z : Real) is
            -- local procedures using Text_IO
            ...
        begin -- G
            -- sequence of statements of G
        end G;
```


INSTRUCTOR NOTES

ARROWS INDICATE DIRECTION OF COMPILE UNIT DEPENDENCIES.

KEY POINTS:

1. SUBUNITS B, C CAN ONLY BE COMPILED AFTER THEIR PARENT UNIT A.
2. SINCE SUBUNIT B DOES NOT DEPEND ON C, SUBUNIT E CAN BE COMPILED BEFORE OR AFTER B.
3. IF Text_10 WERE NOT PRE-COMPILED, IT COULD BE COMPILED AT ANY TIME BEFORE E (SINCE E IS THE ONLY UNIT USING THE RESOURCE).

(B)



(A)	(B)	(C)	(D)	(E)
(A)	(C)	(B)	(D)	(E)
(A)	(C)	(D)	(B)	(E)
(D)	(A)	(B)	(C)	(E)
(D)	(A)	(C)	(B)	(E)
(D)	(A)	(C)	(E)	(B)

VG 823.1

11-10

SEPARATE COMPILATION EXAMPLE --

LIBRARY UNITS

```
• SINGLE COMPILATION UNIT
  procedure Processor is
    Small : constant := 20;
    Total : Integer := 0;
  package Stock is
    Limit : constant := 1000;
    Table : array (1.. Limit) of Integer := (1 .. Limit => 0);
    procedure Restart;
  end Stock;
  package body Stock is
    procedure Restart is
    begin -- Restart
      for N in Table'Range
        loop
          Table (N) := 0;
        end loop;
      end Restart;
    end Stock;
    procedure Update (X : Integer) is
    use Stock;
    begin -- Update
      ...
      Table (X) := Table (X) + Small;
      ...
    end Update;
  begin -- Processor
    ...
    Update (10);
    ...
    Stock.Restart; -- reinitializes Table
    ...
  end Processor;
```

INSTRUCTOR NOTES

DON'T GO INTO GREAT DEPTH - AN EXERCISE ON COMPILATION ORDER FOLLOWS.

VG 823.1

11-12i

SEPARATE COMPILATION EXAMPLE --

LIBRARY UNITS (Continued)

• THREE COMPILATION UNITS

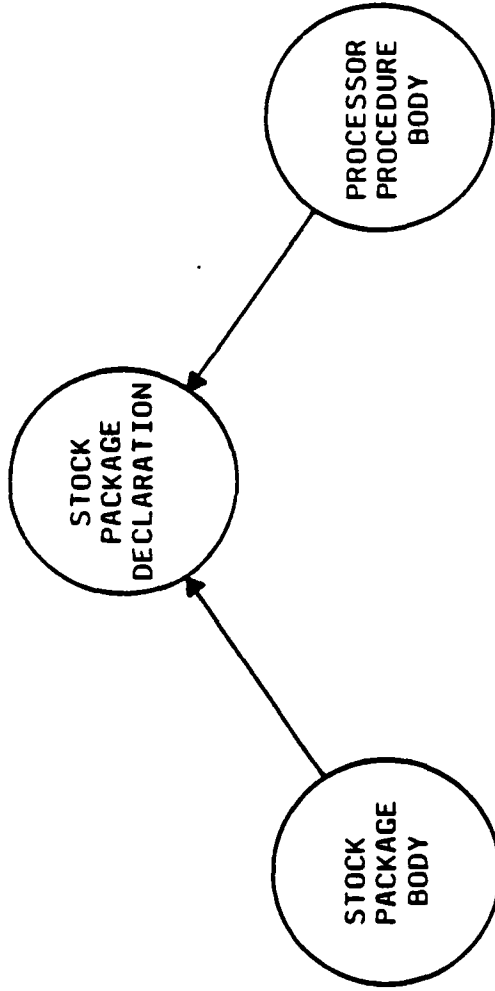
```
package Stock is
  Limit : constant := 1000;
  Table : array (1 .. Limit) of Integer := (1 .. Limit = 0);
  procedure Restart;
end Stock;
-----
package body Stock is
  procedure Restart is
  begin -- Restart
    for N in Table'Range
      loop
        Table(N) := 0;
      end loop;
    end Restart;
  end Stock;
-----
with Stock;
procedure Processor is
  Small : constant := 20;
  Total : Integer := 0;
  procedure Update (X : Integer) is
  use Stock;
  begin -- Update
    ...
    Table (X) := Table (X) + Small;
    ...
  end Update;
begin -- Processor
  ...
  Update (10);
  ...
  Stock.Restart; -- reinitializes Table
  ...
end Processor;
```

INSTRUCTOR NOTES

COMPILATION ORDER DEPENDENCIES

KEY POINTS:

1. STOCK PACKAGE DECLARATION MUST BE COMPILED BEFORE EITHER ITS BODY OR PROCESSOR.
2. PROCESSOR CAN BE COMPILED BEFORE STOCK PACKAGE BODY.



(with Stock;)

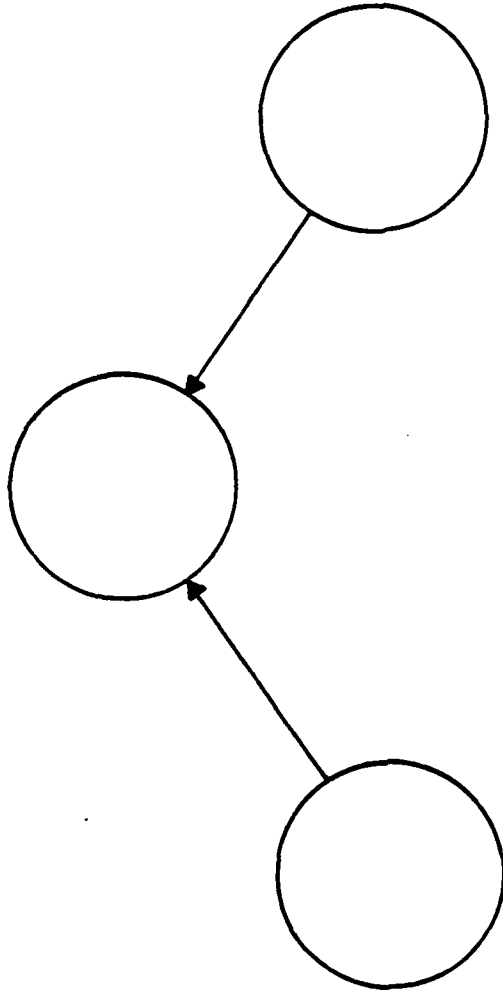
POSSIBLE COMPILATION ORDERS:

1. STOCK PACKAGE DECLARATION	1. STOCK PACKAGE DECLARATION
2. STOCK PACKAGE BODY	2. PROCESSOR PROCEDURE BODY
3. PROCESSOR PROCEDURE BODY	3. STOCK PACKAGE BODY

VG 823.1

11-131

EXERCISE



POSSIBLE COMPILATION ORDERS:

- 1.
- 2.
- 3.

VG 823.1

11-13

INSTRUCTOR NOTES

RESUME DETAILED PRESENTATION HERE.

VG 823.1

11-14i

STUBBING TO INCREASE DESIGN UNDERSTANDABILITY, LOCALIZE LIBRARY UNIT (DATA) INFORMATION, AND DECREASE RECOMPILATION COSTS

ASSUME A PROCESSING PACKAGE WITH OTHER PROGRAM UNITS, WRITTEN BY DIFFERENT PROGRAMMERS,
SPANNING 50 PAGES OF CODE.

MAINTENANCE PROBLEMS:

INTERACTIONS BETWEEN PROGRAM UNITS ARE DIFFICULT TO UNDERSTAND AND GLOBAL DATA
REFERENCES MAY BE HARD TO TRACE.

A ONE LINE CHANGE IN THE PACKAGE WOULD REQUIRE THE ENTIRE PACKAGE TO BE
RECOMPILED (TIME, MONEY, RESOURCES).

DURING TESTING AND INTEGRATION, PROGRAMMERS COMPETE FOR ACCESS TO THE SOURCE CODE
TO INCORPORATE NECESSARY REVISIONS. IF EACH HAS HIS OWN VERSION AND MAKES
REVISIONS, CONFIGURATION MANAGEMENT PROBLEMS INCREASE.

VG 823.1

11-14

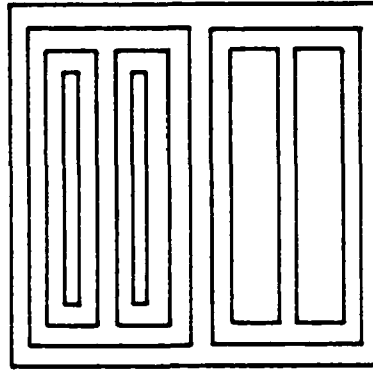
INSTRUCTOR NOTES

VG 823.1

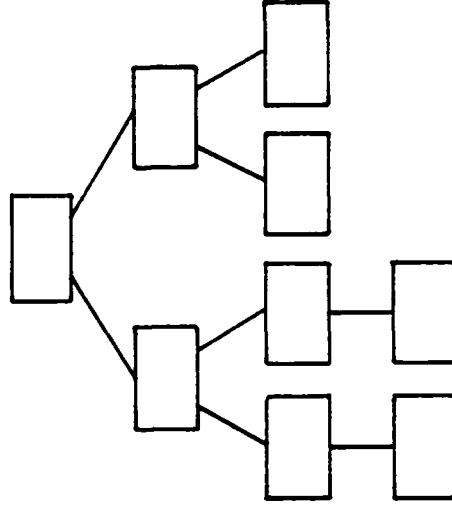
11-15i

MODULE STRUCTURE

IF WE GRAPHICALLY REPRESENT A STRUCTURE, WHICH IS EASIER TO UNDERSTAND?



VS.



INSTRUCTOR NOTES

SIMILARITY,

WHICH IS EASIER TO UNDERSTAND IN A CODED SOLUTION ...

DON'T GO THROUGH EXAMPLE. JUST ASK THE CLASS, IF AT A QUICK GLANCE THEY UNDERSTAND ITS STRUCTURE. THEN PROCEED QUICKLY TO NEXT SLIDE.

THIS?

IN OUTLINE FORM, A PACKAGE BODY

```
with L1, L2;
package body P (written by programmer 1)
task body T1 (written by programmer 1)
variable declarations for T1
function F1 (written by programmer 2)
type and variable declarations for F1
procedure P1 (written by programmers 2 and 3)
variable declarations for P1
statements for P1
function F2 (written by programmer 2)
subtype and variable declarations for F2
statements for F2
statements for F1
statements for T1
task body T2 (written by programmer 1)
subtype declarations for T2
statements for T2
function F4 (written by programmer 1)
subtype and object declarations for F4
statements for F4
task body T3 (written by programmers 2, 3 and 4)
type and variable declarations for T3
end P;
```

INSTRUCTOR NOTES

THIS ILLUSTRATES THE INCREASED UNDERSTANDABILITY THROUGH USE OF STUBS. THE ESSENTIAL STRUCTURE IS EASILY SEEN HERE BUT WAS LOST IN THE PREVIOUS EXAMPLE.

OR THIS?

OUR PACKAGE USING STUBS ...

package body P is

```
task body T1 is separate ;
task body T2 is separate ;
function F4 ( ... ) return ... is separate ;
task body T3 is separate ;
```

end P ;

NO-A165 315

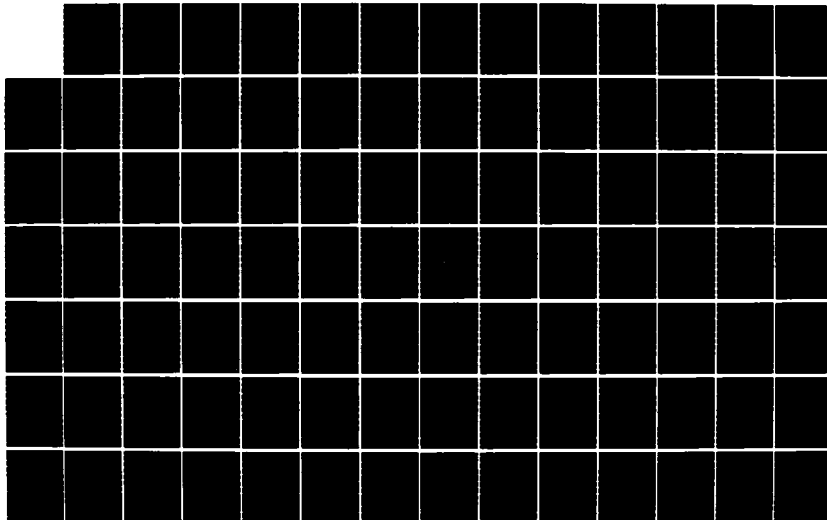
ADA (TRADEMARK) TRAINING CURRICULUM ADA (REGISTERED
TRADEMARK) FOR SOFTWARE MANAGERS L201 TEACHER'S GUIDE
VOLUME 2(U) SOFTECH INC WALTHAM MA 1986
DRAB07-83-C-K506

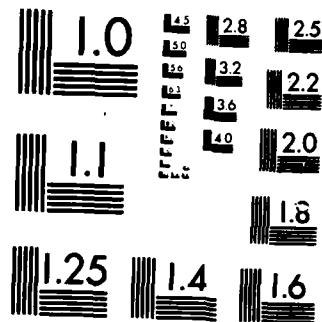
3/3

UNCLASSIFIED

F/G 5/9

NL





INSTRUCTOR NOTES

NOTE: THESE ARE JUST SOME OF THE SUBUNITS THAT WOULD RESULT -- JUST TO GIVE THE FLAVOR OF THE PROCESS.

IF A CHANGE IS NECESSARY TO ONE OF THE SUBUNITS, FOR EXAMPLE F1, ONLY F1 AND THE UNITS THAT ARE DEPENDENT ON F1, P1 AND F2 HERE, WOULD HAVE TO BE RECOMPILED -- NOT THE ENTIRE PACKAGE BODY.

"NECESSARY" LIBRARY UNITS WERE SELECTED ARBITRARILY.

... AND A LOOK AT SOME OF THE RESULTING SUBUNITS

```

separate (P)
task body T1 is
  [variable declaration for T1]
  function F1 ( ... ) return ... is separate ;
begin -- T1
  [statements for T1]
end T1;

```

```

-----
with L2; -- only the necessary library unit(s) are imported
separate (P.T1)
function F1 ( ... ) return ... is
  [type and variable declarations for F1]
  procedure P1 ( ... ) is separate;
  function F2 ( ... ) return ... is separate ;
begin -- F1
  [statements for F1]
end F1 ;

```

```

-----
with L1;
separate (P.T1.F1)
procedure P1 ( ... ) is
  [variable declarations for P1]
begin -- P1
  [statements for P1]
end P1 ;

```

INSTRUCTOR NOTES

BOTH POINTS IMPORTANT TO STRESS: STUBBING IS NOT A CURE-ALL.

VC 823.1

11-19i

A FINAL NOTE

STUBBING IS A TOOL FOR MANAGING PHYSICAL COMPLEXITY; IF THE SUBUNITS HAVE INTRICATE MUTUAL INTERDEPENDENCIES, STUBBING DOES NOT, BY ITSELF, ELIMINATE THE COMPLEXITY.

INSTRUCTOR NOTE

MODULE LENGTH QUOTAS MAY BE IN TERMS OF PAGE LENGTH OR LINES OF CODE.

VG 823.1

11-20i

SOME POTENTIAL PITFALLS - STUBBING

- USING STUBBING TO SATISFY MODULE LENGTH QUOTAS MAY RESULT IN ILLOGICAL UNITS
- TOO TIGHT AN INTERCONNECTION WITH THE PARENT UNIT (PARENT AND STUB CANNOT BE UNDERSTOOD WITHOUT EACH OTHER)

INSTRUCTOR NOTES

VG 823.1

12-i

Section 12

VISIBILITY AND SCOPE

VG 823.1

INSTRUCTOR NOTES

VG 823.1

12-11

MOTIVATION

- ALLOW DELIBERATE SHARING OF ENTITIES

SMITH WRITES AN INITIALIZATION ROUTINE. BROWN WRITES A MEMORY DIAGNOSTIC ROUTINE. SMITH'S ROUTINE NEEDS TO CALL BROWN'S. THE NAME OF BROWN'S ROUTINE MUST BE "VISIBLE" TO SMITH'S ROUTINE.

INSTRUCTOR NOTES

VG 823.1

12-21

MOTIVATION

- EVERY PROGRAMMER SHOULD BE ALLOWED TO INVENT NAMES FOR LOCAL ENTITIES WITHOUT ACCIDENTAL INTERFERENCE WITH ANOTHER PROGRAMMER'S WORK.

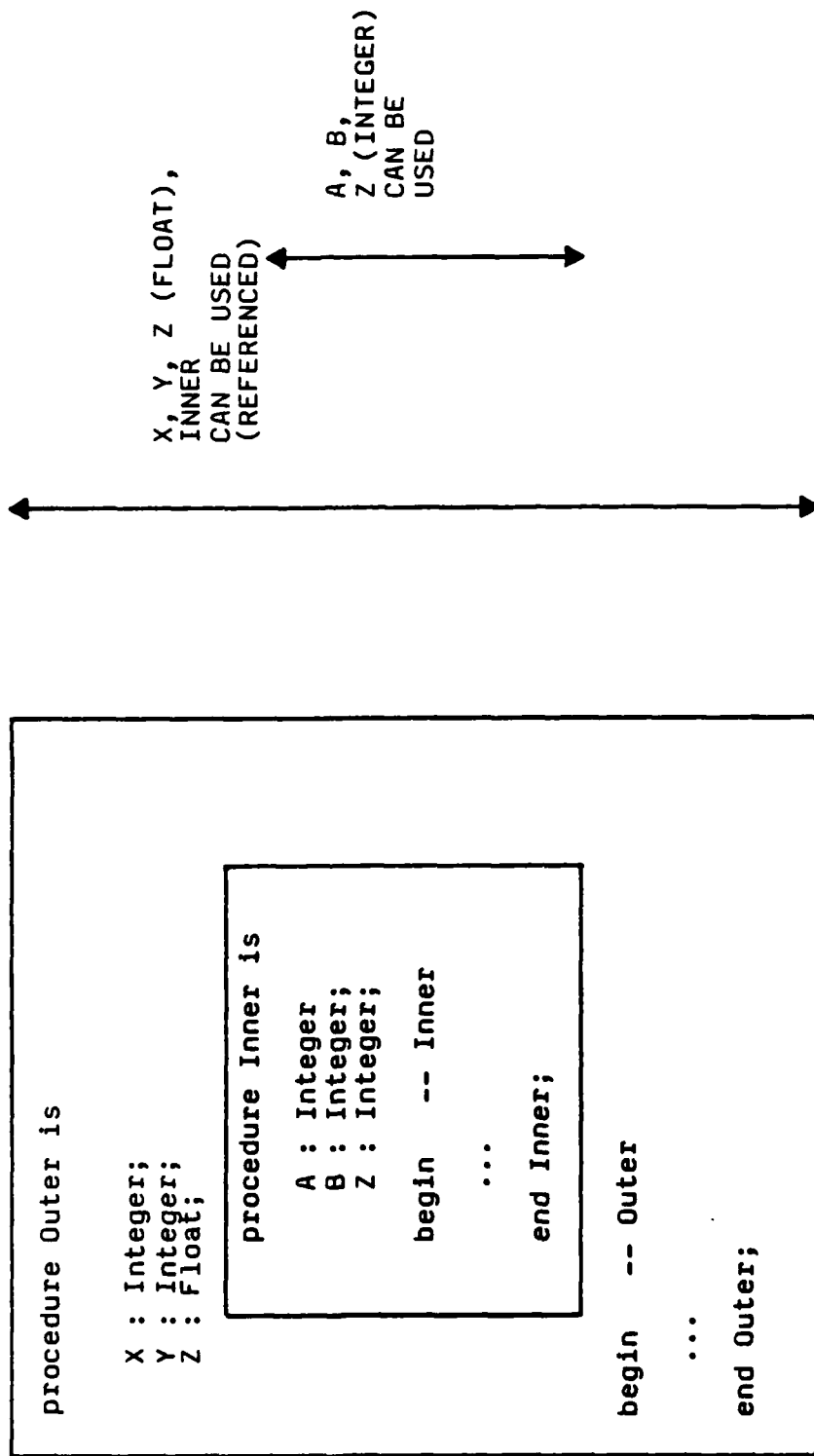
BROWN CAN HAVE A LOCAL VARIABLE CALLED TEMP. SMITH CAN HAVE A LOCAL VARIABLE CALLED TEMP.

INSTRUCTOR NOTES

SCOPE = WHERE AN IDENTIFIER IS KNOWN AND CAN BE USED.

SCOPE

- AREA OF PROGRAM TEXT IN WHICH AN ENTITY APPLIES



INSTRUCTOR NOTES

VISIBILITY = WHERE AN IDENTIFIER IS KNOWN AND WHERE IT REFERS TO THAT ENTITY
AND NOT TO SOMETHING ELSE.

WHEN WRITING/MODIFYING A PROGRAM UNIT P ...

... IF SOME ENTITY (E.G., A VARIABLE) X IS NOT VISIBLE ...

... THEN

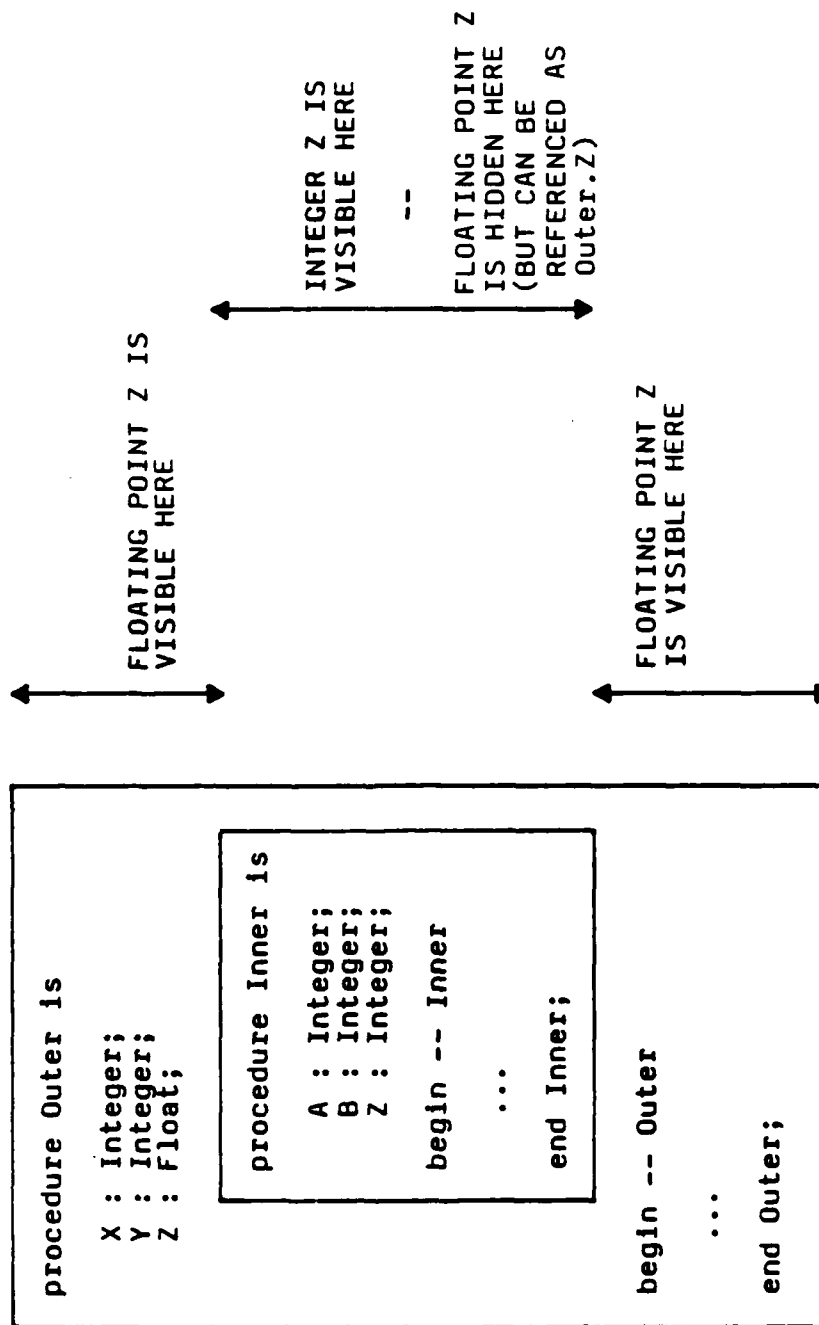
1. YOU CANNOT USE THAT VARIABLE IN ANY WAY IN P
2. YOU CAN FREELY USE THE NAME X FOR YOUR OWN DATA (OR
SUBPROGRAM, ETC.)

... IN OTHER WORDS ...

WHERE SOMETHING IS NOT VISIBLE, FOR ALL PURPOSES IT DOESN'T EXIST.

VISIBILITY

- AREA OF PROGRAM TEXT IN WHICH AN ENTITY HAS A UNIQUE NAME



INSTRUCTOR NOTES

1. MEANS B DOESN'T SEE A AND VICE VERSA.
2. MEANS B CAN'T MENTION A BUT A CAN MENTION B.

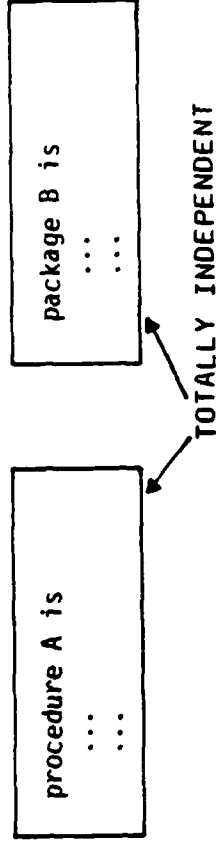
VG 823.1

12-51

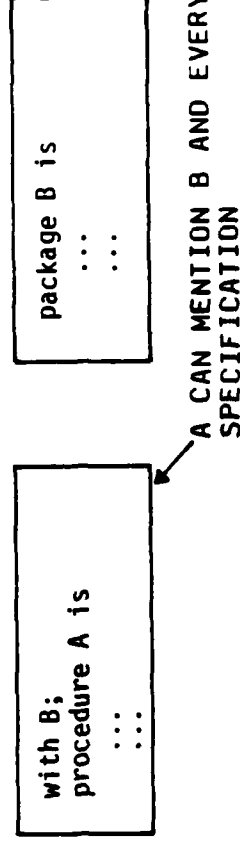
GENERAL RULES (INFORMALLY)

- LIBRARY UNITS

1. NOTHING IS MUTUALLY VISIBLE UNLESS EXPLICITLY REQUESTED



2. THE with CLAUSE MAKES THE INTERFACE VISIBLE



3. ENTITIES DECLARED IN THE BODIES CAN NEVER BE MADE VISIBLE OUTSIDE OF THE BODY.

RATIONALE:

- MODULARITY WHILE AVOIDING ACCIDENTAL NAME CONFLICTS

INSTRUCTOR NOTES

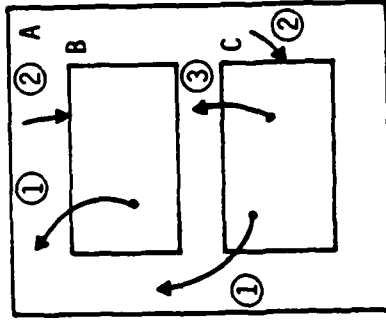
POINT OUT THAT THE OUTERMOST UNIT IS EITHER A SUBPROGRAM, PACKAGE BODY OR TASK BODY.

IF THE UNITS A, B AND C ARE ALL PACKAGE SPECS, THE RULES WOULD BE DIFFERENT:

1. STAYS SAME
2. WHAT'S DECLARED IN B IS VISIBLE BY SELECTION (I.E. B.entity) INSIDE C AND THE REST OF A BUT NOT VICE VERSA (C DECLARED AFTER B).

GENERAL VISIBILITY RULES (Continued)

- NESTED UNITS:



1. EVERYTHING DECLARED IN A IS VISIBLE FROM B AND C.
2. NOTHING DECLARED IN B OR C IS VISIBLE FROM A.
3. NOTHING DECLARED IN B IS VISIBLE FROM C, AND VICE-VERSA.

AN EASY RULE: YOU CAN LOOK OUT, BUT YOU CAN'T LOOK IN.

INSTRUCTOR NOTES

REMEMBER STUBS JUST TEXTUALLY REMOVE THE CODE TO ANOTHER AREA, BUT NOT LOGICALLY.

GENERAL VISIBILITY RULES (Continued)

- STUBS:

AS FOR NESTED UNITS (STUBBING DOESN'T CHANGE VISIBILITY)

- IN ALL CASES:

A LOCAL DECLARATION "HIDES" AN ENTITY WITH THE SAME NAME WHICH WAS VISIBLE
FOR ANY OTHER REASON.

INSTRUCTOR NOTES

THESE HAVE BEEN VERY GENERAL GUIDELINES. IN REALITY ADA HAS VERY ELABORATE SCOPE AND VISIBILITY RULES.

VG 823.1

12-81

SUMMARY

ADA HAS ELABORATE VISIBILITY RULES ...

... TO ALLOW PROGRAMMERS TO WORK INDEPENDENTLY ...

... EXCEPT WHEN THEY NEED TO COMMUNICATE.

INSTRUCTOR NOTES

VG 823.1

12-91

SOME POTENTIAL PITFALLS - VISIBILITY

- ADA WAS DESIGNED SO THAT IF A PROGRAM IS LEGAL, IT MEANS WHAT YOU THINK IT DOES. BUT BECAUSE THESE RULES ARE COMPLEX, IT IS OCCASIONALLY DIFFICULT TO DIAGNOSE WHY A PROGRAM IS ILLEGAL.
- UNRESTRICTED USE OF THE use CLAUSE FOR PACKAGES MAKES PROGRAMS LESS SELF-DOCUMENTING AND HARDER TO UNDERSTAND. THE use CLAUSE MAKES ALL IDENTIFIERS DIRECTLY VISIBLE WITHOUT PREFIX (DOT) NOTATION. IF SEVERAL PACKAGES WITH MANY IDENTIFIERS ARE USED, IT BECOMES DIFFICULT TO KNOW WHERE TO FIND A GIVEN IDENTIFIER IF CHANGES ARE NECESSARY.

RECOMMENDED USAGE:

- TO IMPORT STANDARD PACKAGES SUCH AS I/O
- TO BE ABLE TO USE IMPORTED OPERATORS IN "NATURAL" INFIX FORM
 $X + Y$ RATHER THAN P. "+" (X, Y)

INSTRUCTOR NOTES

AS WE WILL SEE IN A MOMENT, THE USER CAN DEFINE THE MEANING OF ANY OPERATION FOR
USER-DEFINED TYPES.

Section 13

OVERLOADING

VG 823.1

INSTRUCTOR NOTES

EXAMPLES OF NAMES, SUBPROGRAMS OVERLOADED.

THE FIRST PUT TAKES AN ARGUMENT OF TYPE STRING WHERE THE SECOND TAKES AN ARGUMENT OF Scores_Type. EACH Put IS A DISTINCT PROCEDURE, THEY JUST SHARE AN IDENTIFIER NAME.

OVERLOADING

- CONCEPT OF ONE ENTITY NAME REPRESENTING TWO OR MORE ENTITIES

Put ("Median of Scores is: ");

Put (Median);

- ENUMERATION LITERALS, SUBPROGRAM NAMES, OPERATORS CAN BE OVERLOADED
- ALLOWS PROGRAMMERS TO CHOOSE NAMES APPROPRIATE TO THEIR USE (THE ABSTRACTION) AS LONG AS AMBIGUITY CAN BE RESOLVED BY CONTEXT OR QUALIFICATION

INSTRUCTOR NOTES

USE OVERLOADING PURPOSEFULLY.

OVERLOADING

CAN BE USED

- TO SHOW SIMILARITY OF FUNCTION
- TO ALLOW THE USE OF "NATURAL" TERMINOLOGY WITHOUT NAME CONFLICTS

SHOULD BE USED WITH DELIBERATE INTENT

INSTRUCTOR NOTES

VG 823.1

13-31

SUBPROGRAM OVERLOADING

- SUBPROGRAMS MAY BE OVERLOADED IF THEY HAVE DIFFERENT NUMBERS OR TYPES OF OPERANDS
- THE SUBPROGRAMS CAN THEN BE DISTINGUISHED BY THE PATTERN OF THEIR OPERANDS

- CONTEXT:
 with Text_IO; use Text_IO;
 -- defines Put and Get for Strings
package Int_IO is new Integer_IO (Integer); use Int_IO;
 -- defines Put and Get for Integer

EXAMPLES:

```
Put ("NAME:"); -- uses Text_IO.Put  
Put (3); -- uses Int_IO.Put
```

INSTRUCTOR NOTES

GO THROUGH EXAMPLES, THEN ASK ...

IF WE HAD A STATEMENT LIKE

if Yes < No then ...

THE COMPILER DOESN'T KNOW WHICH TYPE THESE BELONG TO.

OVERLOADING - ENUMERATION LITERALS

SAME ENUMERATION LITERAL MAY BE USED IN MORE THAN ONE TYPE DECLARATION

```
type Counting_Format is (Binary, BCD);  
type Tape_Format is (ASCII, EBCDIC, Binary);  
type Interrupt_Status is (Yes, No);  
type Answer is (Yes, No, Maybe);
```

INSTRUCTOR NOTES

ONE QUALIFIER NEEDED.

VG 823.1

13-51

OVERLOADING: TYPE AMBIGUITY AND QUALIFICATION

USUALLY THE COMPILER CAN SORT THINGS OUT. WHEN IN TROUBLE, ADA GIVES YOU A WAY OF UNIQUELY IDENTIFYING THE OVERLOADED LITERALS BY MEANS OF TYPE QUALIFICATION

Example:

```
if Interrupt_Status'(Yes) < No then ...
```

IDENTIFYING NOTATION PRECEDES THE OVERLOADED LITERAL WITH ITS TYPE NAME AND AN APOSTROPHE.

SYNTAX:

Type_Name'(Overloaded_Literal)

INSTRUCTOR NOTES

VG 823.1

13-61

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

OPERATOR OVERLOADING

- OPERATORS ARE IMPLEMENTED AS FUNCTIONS IN ADA. FOR EXAMPLE:

function "+" (Left, Right : Integer) return Integer;

function "+" (Left, Right : Float) return Float;

- OPERATORS ARE ALREADY OVERLOADED. IT IS POSSIBLE TO EXTEND OPERATORS TO OTHER DATA TYPES THROUGH OVERLOADING. FOR EXAMPLE:

function "+" (Left, Right : Matrix_Type) return Matrix_Type;

- FOR NOTATIONAL CONVENIENCE, INFIX NOTATION CAN BE USED. FOR EXAMPLE:

X := 1 + 2; -- where X is type Integer

Y := 1.0 + 2.0; -- where Y is type Float

A := M + N; -- where M, N, A are of Matrix_Type

INSTRUCTOR NOTES

IF TWO SUBPROGRAMS PERFORM THE SAME ABSTRACT FUNCTION, WE CAN REFLECT THIS DIRECTLY IN
ADA CODE.

VG 823.1

13-71

OVERLOADING TO SHOW FUNCTION SIMILARITY

```
procedure Get (Item : out Character);  
procedure Get (Item : out Integer);  
  
function "+" (A, B : in Matrix) return Matrix;  
function "+" (A, B : in Vector) return Vector;
```

INSTRUCTOR NOTES

VG 823.1

13-8i

OVERLOADING TO ALLOW NATURAL NAMES

type Switch_Status Is (On, Off);

type Lamp_Control_Command Is (Off, Flash);

OVERLOADING

INSTRUCTOR NOTES

WE CAN DETERMINE THE SPECIFIC Sqrt FUNCTION THROUGH CONTEXT (I.E. THE TYPES OF THE PARAMETERS).

SOME POTENTIAL PITFALLS - OVERLOADING

- USING CONTRIVED IDENTIFIERS RATHER THAN OVERLOADING

FOR EXAMPLE:

```
function Float_Sqrt (X : Float) return Float;  
function Integer_Sqrt (X : Integer) return Float;
```

BOTH FUNCTIONS SHOULD BE CALLED Sqrt.

- USING OPERATORS FOR FUNCTIONS NOT RELATED TO THEIR STANDARD MATHEMATICAL MEANINGS (E.G., USING UNARY "-" TO DENOTE MATRIX INVERSION)

INSTRUCTOR NOTES

BREEZE THROUGH THIS SECTION.

Section 14

PRAGMAS

VG 823.1

INSTRUCTOR NOTES

VG 823.1

14-11

PRAGMAS -- BASIC IDEA

- CONVEY INFORMATION TO THE COMPILER
- SOME ARE LANGUAGE-DEFINED
- OTHERS ARE IMPLEMENTATION-DEFINED
- MAY AFFECT PERFORMANCE OF PROGRAM
- DO NOT CHANGE MEANING OF PROGRAM (I.E.,
NO RESULTS PRODUCED)

INSTRUCTOR NOTES

EXAMPLE OF WHAT TO PRESENT IN SLIDE:

A. SYNTAX

B. FOR EXAMPLE:

1. PRAGMA INTERFACE ALLOWS MODULES WRITTEN IN OTHER LANGUAGES TO BE USED.
2. PRAGMA PACK IS A COMMAND FOR THE OPTIMAL SPACE ASSIGNMENT AND COMPRESSION OF DATA OBJECTS.
3. PRAGMA PAGE IS A COMMAND TO HAVE THE COMPILER LISTINGS FOR WHAT FOLLOWS TO START ON A NEW PAGE.

PREDEFINED PRAGMAS

SYNTAX:

```
pragma Predefined_Pragma_Name [(Parameters)];
```

WHERE Predefined_Pragma_Name CAN BE:

--	Controlled	--	Optimize
--	Elaborate	--	Pack
--	Inline	--	Page
--	Interface	--	Priority
--	List	--	Shared
--	Memory_Size	--	Storage_Unit
--	System_Name	--	Suppress

SUMMARIZED IN APPENDIX B OF LRM

INSTRUCTOR NOTES

VG 823.1

14-31

EXAMPLES

```
pragma Page;  
separate (Vector_Services)  
procedure Sqrt (X : Float) is  
    Epsilon : constant := 0.000001;  
    Root : Float;  
begin -- Sqrt  
    ...  
end Sqrt;
```


INSTRUCTOR NOTES

VG 823.1

14-41

SOME POTENTIAL MISUSES - PRAGMAS

- POTENTIAL PORTABILITY PROBLEMS CAUSED BY IMPLEMENTATION-DEFINED PRAGMAS
- INLINE, OPTIMIZE, PACK: SHOULD BE USED WITH SOME CARE. THEY MAY ACTUALLY PRODUCE SLOW OR INEFFICIENT CODE.
- SHARED: REVEALS UNSYNCHRONIZED ACCESS TO DATA, WHICH IS EXTREMELY ERROR-PRONE.
- SUPPRESS: VERY DANGEROUS. SHOULD BE USED ONLY WHEN
 - IT CAN BE PROVED THAT THE EXCEPTION WILL NEVER BE RAISED
 - THE BENEFIT IN PERFORMANCE CAN BE DEMONSTRATED BY MEASUREMENTS

INSTRUCTOR NOTES

BREEZE THROUGH THIS SECTION. MANAGERS SHOULD KNOW THAT THESE FEATURES EXIST, THEIR USE SHOULD BE RESTRICTED TO ACTUAL NEED, SHOULD GROUP THESE FEATURES FOR EASE OF MAINTENANCE AND PORTING.

Section 15

LOW-LEVEL FEATURES

VG 823.1

INSTRUCTOR NOTES

VG 823.1

15-11

LOW-LEVEL FEATURES

NORMALLY IT IS SUFFICIENT TO ALLOW THE COMPILER TO MAKE THE CHOICES ON THE ACTUAL REPRESENTATION OF THE BIT PATTERNS AND HARDWARE ADDRESSING FOR PROGRAM ENTITIES (OBJECTS, PROGRAM UNITS, FOR EXAMPLE). WITH EMBEDDED SYSTEMS, IT BECOMES NECESSARY AT TIMES TO DEAL WITH THE ACTUAL HARDWARE ...

INSTRUCTOR NOTES

NOT EVERY PROGRAMMER WILL NEED TO USE THESE FEATURES. THEIR USE SHOULD BE THE EXCEPTION, NOT THE RULE.

VG 823.1

15-21

LOW-LEVEL FEATURES

- FACILITIES TO INTERFACE WITH UNDERLYING HARDWARE OR PERIPHERALS WITHOUT LEAVING AN HOL
- MACHINE REPRESENTATION SPECIFICATION
- MACHINE CODE INSERTION
- LOW-LEVEL I/O
- UNCHECKED CONVERSION
- CAN BE USED FOR
- MEMORY-MAPPED I/O
- EXPLOITING UNDERLYING HARDWARE FEATURES

INSTRUCTOR NOTES

VG 823.1

15-3i

REPRESENTATION SPECIFICATIONS -- BASIC IDEA

- MAP ABSTRACT DATA TO PHYSICAL HARDWARE
- PHYSICAL REPRESENTATION OF VALUES
 - SPECIFY INTERNAL CODES FOR LITERALS OF AN ENUMERATION TYPE
 - SPECIFY RECORD LAYOUT
- CONTROL STORAGE
 - SPECIFY AMOUNT OF STORAGE ASSOCIATED WITH A TYPE
- CONTROL LOCATION
 - SPECIFY REQUIRED STORAGE ADDRESS FOR AN ENTITY

INSTRUCTOR NOTES

BASED NUMERIC LITERALS CAN BE USED ANYWHERE NUMERIC LITERALS OCCUR.

SYNTAX: BASE DECIMAL VALUE # NUMBER IN BASE NOTATION # ANY EXPONENTS

VG 823.1

15-4i

ENUMERATION TYPE REPRESENTATION CLAUSES

- MAPPING FROM ELEMENTS OF A TYPE TO SPECIFIC INTERNAL CODES

type Op_Codes is (LOD, STO, ADD, SUB, JNT);

for Op_Codes use (8#01#, 8#12#, 8#13#, 8#14#, 8#17#);

INSTRUCTOR NOTES

VG 823.1

15-51

RECORD TYPE REPRESENTATION CLAUSES

- MAPS

- ORDER OF RECORD COMPONENTS
- COMPONENT POSITION
- COMPONENT SIZE
- GLOBAL ALIGNMENT OF RECORD

INSTRUCTOR NOTES

- MENTION THAT THE REPRESENTATION CLAUSE DOES NOT HAVE TO SPECIFY THE REPRESENTATION OF EACH COMPONENT. NOR DO THEY NEED TO SPECIFY THEM IN THE ORDER THEY APPEAR IN THE RECORD SPECIFICATION.
- POINT OUT THAT THE USE OF LOW LEVEL FEATURES IN PROGRAMS IS A MANAGEMENT ISSUE.
- at CLAUSE
 - LOCATES COMPONENT RELATIVE TO START OF RECORD
 - MEASURED IN STORAGE UNITS
- RANGE IN BITS
 - LOCATION AND EXTENT OF A COMPONENT RELATIVE TO A STORAGE_UNIT
- ALIGNMENT CLAUSE
 - STORAGE FJUNDARY FOR BEGINNING OF AN OBJECT

RECORD TYPE REPRESENTATION - EXAMPLE

```
type Position_Delta is range 0 .. 255;
type Gantry_Position_Reading is
```

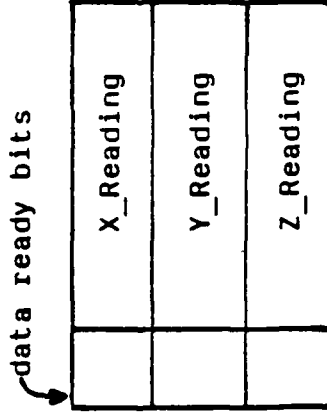
```
record
```

```
  X_Data_Ready : Boolean;
  X_Reading    : Position_Delta;
  Y_Data_Ready : Boolean;
  Y_Reading    : Position_Delta;
  Z_Data_Ready : Boolean;
  Z_Reading    : Position_Delta;
```

```
end record;
```

```
for Gantry_Position_Reading use
```

```
record at mod 2;    -- double byte boundary
  X_Data_Ready at 0 range 0..0;    -- word 0, bit 0
  X_Reading at 0 range 1..15;    -- word 0, bits 1-15
  Y_Data_Ready at 1 range 0..0;    -- etc.
  Y_Reading at 1 range 1..15;
  Z_Data_Ready at 2 range 0..0;
  Z_Reading at 2 range 1..15;
end record;
```



INSTRUCTOR NOTES

VG 823.1

15-7i

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

LENGTH SPECIFICATION

- USED TO OVERRIDE THE AMOUNT OF STORAGE NORMALLY ASSOCIATED WITH AN ENTITY
- ATTRIBUTES
 - DATA TYPE
 - T'Size : NUMBER OF BITS FOR OBJECTS OF TYPE T
 - ACCESS TYPE
 - T'Storage_Size : NUMBER OF STORAGE UNITS TO BE RESERVED
FOR ALLOCATING OBJECTS
 - TASK OR TASK TYPE
 - T'Storage_Size : NUMBER OF STORAGE UNITS TO BE RESERVED
FOR EACH ACTIVATION OF THE TASK OBJECT
- for Attribute use Integer_Expression;
for Vehicle_Record'Size use 1000;

INSTRUCTOR NOTES

VG 823.1

15-81

ADDRESS SPECIFICATION

- USED TO SPECIFY THE LOCATION OF AN OBJECT IN STORAGE
- USED TO SPECIFY THE STARTING ADDRESS OF A PROGRAM UNIT
for Boot_Program use at 8#40#;
- CAN BE USED TO ASSIGN AN ENTRY TO A HARDWARE INTERRUPT
task Fire_Control_Interrupt_Handler is
entry Fired;
for Fired use at 8#26#;
end Fire_Control_Interrupt_Handler;

INSTRUCTOR NOTES

VG 823.1

15-91

1000 9000 8000 7000 6000 5000 4000 3000 2000 1000 0

MACHINE CODE INSERTIONS -- BASIC IDEA

- ADA PERMITS INSERTION OF MACHINE CODE INSTRUCTIONS, USING THE CODE STATEMENTS
- SPECIFICS ARE IMPLEMENTATION-DEPENDENT
- SHOULD BE USED ONLY FOR VERY SHORT SEQUENCES OF MACHINE LANGUAGE INSTRUCTIONS (1-5 INSTRUCTIONS). FOR LONGER SEQUENCES, WRITE AN ASSEMBLY LANGUAGE MODULE AND USE THE INTERFACE PRAGMA
- IN GENERAL, A VERY CLUMSY WAY OF WRITING MACHINE INSTRUCTIONS IN ADA

INSTRUCTOR NOTES

DON'T DWELL ON SLIDE.

VG 823.1

15-10i

LOW-LEVEL I/O

- INTERFACES TO A PHYSICAL DEVICE

- PREDEFINED PROCEDURES

```
-- Send_Control      : SENDS CONTROL INFORMATION TO A PHYSICAL DEVICE
-- Receive_Control   : MONITORS EXECUTION OF I/O BY REQUESTING INFORMATION
                      FROM PHYSICAL DEVICE
```

- EACH PROCEDURE HAS TWO PARAMETERS

```
-- Device           : IDENTIFIES THE DEVICE
-- Data              : IDENTIFIES THE DATA
```

TYPES OF PARAMETERS ARE IMPLEMENTATION DEFINED

- BODIES FOR THE PROCEDURES MAY BE WRITTEN IN CODE STATEMENTS

INSTRUCTOR NOTES

SUSPENDS ADA'S NORMAL TYPE CHECKING. AGAIN, FEATURE SHOULD BE USED SPARINGLY.

VG 823.1

15-111

UNCHECKED CONVERSION

- USED FOR UNCHECKED TYPE CONVERSIONS (I.E. PERMITS CONVERSIONS BETWEEN ANY

TWO TYPES)

- PROVIDED BY INSTANTIATION OF A GENERIC FUNCTION

generic

type Source is limited private;

type Target is limited private;

function Unchecked_Conversions (S : Source) return Target;

- RETURNS THE BIT PATTERN OF THE INPUT SOURCE

INSTRUCTOR NOTES

COVERAGE OF THE FOLLOWING (5 SLIDES) USING LOW-LEVEL FEATURES FOR MEMORY-MAPPED I/O IS OPTIONAL. IF COVERED, INSTRUCTOR SHOULD UNDERSTAND THE CODE, THE KEY POINTS STATED AS THE ADVANTAGES/DISADVANTAGES, AND THE CLASS SHOULD BE INTERESTED IN THE TOPIC. IF NOT COVERED, TELL THE CLASS FOR THOSE INTERESTED IN THIS PARTICULAR TOPIC, EXAMPLES ARE PROVIDED FOR FUTURE REFERENCE.

LOW-LEVEL FEATURES FOR MEMORY-MAPPED I/O

IN A COMPUTER SYSTEM, MEMORY_MAPPED I/O USES REGULAR MEMORY REFERENCE INSTRUCTIONS WHICH ARE MAPPED TO A DEVICE-SPECIFIC MEMORY ADDRESS OR I/O PORT. IN EMBEDDED COMPUTER SYSTEMS, THE ABILITY TO HAVE SPECIFIC DEVICE CONTROL IS VITALLY IMPORTANT.

AN EXAMPLE:

TO BE ABLE TO READ AN 8-BIT VALUE FROM SOME MEMORY-MAPPED I/O DEVICE
AND THEN TO WRITE AN 8-BIT VALUE TO THE DEVICE.

SEVERAL POSSIBLE SOLUTIONS WILL BE SHOWN.

INSTRUCTOR NOTES

VG 823.1

15-131

THE MAIN PROGRAM FOR EACH SOLUTION

```
procedure Main is
  subtype Byte is Integer range 0 .. 255;
  Vall, Val2 : Byte;

  procedure Device_IO (Outval : in Byte;
    Inval : out Byte) is separate;

begin -- Main

  Vall := 41;
  Device_IO (Vall, Val2);

end Main;
```

INSTRUCTOR NOTES

VG 823.1

15-141

THE OBVIOUS SOLUTION

USE OF MACHINE REPRESENTATION FOR THE ADDRESS OF THE I/O PORT

```
separate (Main)
procedure Device_IO (Outval : in Byte;
                    Inval  : out Byte) is
    IO_Port      : Byte;
    IO_Port_Loc : constant := 16#C0F0#;
    for IO_Port use at IO_Port_Loc;
begin -- Device_IO

    IO_Port := Outval; -- write outval to the IO_Port
    Inval   := IO_Port; -- read inval from the IO_Port

end Device_IO;
```

DISADVANTAGES:

IT WILL NOT WORK! FOR ORDINARY VARIABLES, THE SEQUENCE OF ASSIGNMENT STATEMENTS WOULD BE EQUIVALENT TO INVAL := OUTVAL. THE COMPILER WILL ALMOST CERTAINLY DO THE SIMPLIFICATION. THE VARIABLE FOR THE I/O PORT IS NOT AN ORDINARY VARIABLE AS THE VALUE RECEIVED FROM THE DEVICE WILL USUALLY BE CHANGED BY THE SYSTEM BEFORE IT IS EVER SENT BACK TO THE DEVICE THROUGH THE I/O PORT.

INSTRUCTOR NOTES

VG 823.1

15-151

A WORKABLE SOLUTION

USING THE PRAGMA Shared INSURES THAT VARIABLES CONTAIN THEIR MOST CURRENT VALUE.

```
separate (Main)
procedure Device_IO (Outval : in Byte;
                    Inval  : out Byte) is
    IO_Port      : Byte;
    IO_Port_Loc : constant := 16#C0F0#;
    for IO_Port use at IO_Port_Loc;
    pragma Shared (IO_Port);
begin -- Device IO
    IO_Port := Outval;    -- write to IO_Port
    Inval := IO_Port;     -- read from IO_Port
end Device_IO;
```

DISADVANTAGES:

IT WILL BE HARD TO MAINTAIN WITHOUT EXPLICIT DOCUMENTATION TO DESCRIBE EXACTLY WHAT IS BEING DONE. FOR EXAMPLE, THE ASSIGNMENT STATEMENTS (IO_Port := Outval) ARE REALLY I/O OPERATIONS. THE CODE DOES NOT SHOW THIS ABSTRACTION.

INSTRUCTOR NOTES

Memory_Address_Type, Device_Type, AND Memory_Device ARE IMPLEMENTATION-DEFINED, NOT PART OF THE LRM SPECIFICATION FOR ADA.

VG 823.1

15-16i

A BETTER SOLUTION

USE THE STANDARD PACKAGE Low_Level_IO WHICH IS IMPLEMENTATION DEPENDENT

```
with Low_Level_IO;
separate (Main)
procedure Device_IO (Outval : in out Byte;
                    Inval  : in out Byte) is
    IO_Port_Loc : constant Low_Level_IO.Memory_Address_Type
                  := 16#C0F0#;
    IO_Port     : constant Low_Level_IO.Device_Type
                  := (Low_Level_IO.Memory_Device, IO_Port_Loc);
begin -- Device IO
    Low_Level_IO.Send_Control (IO_Port, Outval); -- write outval
    Low_Level_IO.Receive_Control (IO_Port, Inval); -- read inval
end Device_IO;
```

ADVANTAGES:

THE CODE MORE ACCURATELY STATES WHAT IS TO BE DONE.

INSTRUCTOR NOTES

RESUME COVERAGE HERE IF PREVIOUS EXAMPLE SKIPPED.

VG 823.1

15-171

LOW-LEVEL FEATURES TO EXPLOIT BASIC HARDWARE

FOR EXAMPLE, REPRESENTATION SPECS TO ALIGN RECORDS ON WORD BOUNDARIES WHICH THEN ALLOW MORE EFFICIENT USE OF THE UNDERLYING MACHINE CODE.

IT SHOULD BE NOTED THAT USE OF THIS ADA FACILITY MAKES THE DESIGN HARDWARE SPECIFIC AND CONSEQUENTLY LIMITS PORTABILITY.

15-18i

SOME POTENTIAL MISUSES - LOW-LEVEL FEATURES

- USED BECAUSE OF SIMILARITY TO ASSEMBLE LANGUAGE
- USED TO GAIN EFFICIENCY (RATHER THAN MORE THOUGHT, PLANNING, OR "TWIDDLING" WITH THE DESIGN)
- USED TO CIRCUMVENT ADA'S CONTROLLED TYPE CHECKING RULES
- LACK OF ENCAPSULATION

HINT: LOW-LEVEL FEATURES SHOULD BE USED TO IMPLEMENT SOME ABSTRACTION. HARDWARE CHARACTERISTICS (E.G., THE SPECIFIC FORMAT OF A CONTROL WORD) SHOULD BE HIDDEN INSIDE A PACKAGE THAT EXPORTS A LOGICAL VIEW (E.G., A RECORD TYPE DEFINITION)

INSTRUCTOR NOTES

THIS SECTION IS FOR STUDENT REFERENCE ONLY, NOT TO BE COVERED IN CLASS.

VG 823.1

16-1

Section 16

SUMMARY OF USES OF ADA FEATURES

VG 823.1

AD-A165 315

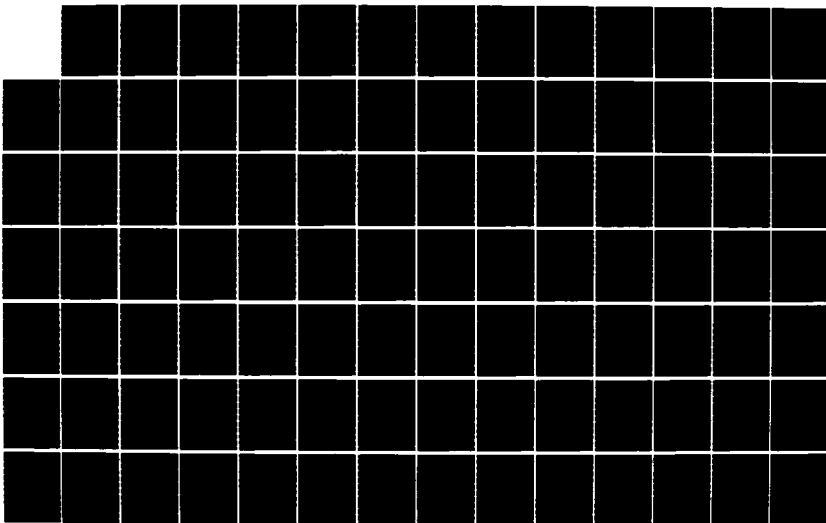
ADA (TRADEMARK) TRAINING CURRICULUM ADA (REGISTERED
TRADEMARK) FOR SOFTWARE MANAGERS L201 TEACHER'S GUIDE
VOLUME 2(U) SOFTECH INC WALTHAM MA 1986
DADB07-83-C-K506

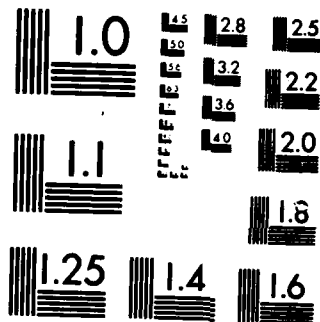
4/5

UNCLASSIFIED

F/G 5/9

NL





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

INSTRUCTOR NOTES

VG 823.1

16-11

SUMMARY OF ADA FEATURES USES*

ADA FEATURE

USES

LEXICAL RULES

BUILDING BLOCKS OF ADA CODE
FREE FORMAT FOR READABILITY/UNDERSTANDABILITY
"NATURAL" IDENTIFIER NAME

ADA TYPING SYSTEM

CORRECT CLASS OF ERRORS BEFORE INTEGRATION
DOCUMENT DESIGN
EXPRESS DESIGN CONSTRAINTS

STATEMENTS

CONTROL FOR ALGORITHMS

PACKAGES

GROUP RELATED PROGRAM UNITS
NAME COLLECTIONS OF DECLARATIONS
EXPRESS ABSTRACT DATA OBJECTS
EXPRESS FINITE STATE MACHINES

PRIVATE TYPES

CONTROL (BY USER) ABSTRACT DATA TYPES
PASS GENERIC PARAMETERS

*USES LISTED ARE A REPRESENTATIVE SAMPLING OF APPROPRIATE USES. COMBINATIONS AND MODIFICATIONS WILL OCCUR.

INSTRUCTOR NOTES

VG 823.1

16-21

SUMMARY (Continued)

<u>ADA FEATURE</u>	<u>USES</u>
SUBPROGRAMS	ARE MAIN PROGRAM UNITS DEFINITION OF ALGORITHM DEFINITION OF OPERATIONS ON ABSTRACT TYPE
TASKS	EXPRESS CONCURRENT ACTIONS CONTROL RESOURCES HANDLE INTERRUPTS ACT AS BUFFERS
GENERIC	WRITE TEMPLATE FOR MODULES WITH SIMILAR ALGORITHMS PASS TYPES OR SUBPROGRAMS TO PROGRAM UNITS DETAIL DESIGN DECISIONS WHILE DEFERRING DESIGN DECISIONS
EXCEPTIONS	DETECT AND RECOVER FROM EXCEPTIONAL OR ERROR CONDITIONS DO "CLEANUP" ACTIONS FOR HARDWARE MALFUNCTIONS
STUBS	COMPLEX CODE UNDERSTANDABILITY LOCALIZE LIBRARY UNIT INFORMATION DECREASE RECOMPILATION

INSTRUCTOR NOTES

VG 823.1

16-3i

SUMMARY (Continued)

ADA FEATURE

USES

VISIBILITY

PREVENTS ACCIDENTAL NAME CONFLICTS
ALLOWS DELIBERATE SHARING OF NAMES
ALLOWS PROGRAMMERS TO WORK INDEPENDENTLY

OVERLOADING

SHOW SIMILARITY OF FUNCTION
ALLOW NATURAL NAMES

INPUT/OUTPUT

ALLOWS USER COMPLETE CONTROL OF I/O
ALLOWS CREATION OF INTERFACES WITH
NON-STANDARD I/O DEVICES

PRAGMAS

PERMIT COMMANDS (SPECIAL REQUESTS)
TO THE COMPILER

LOW LEVEL FEATURES

DO MEMORY-MAPPED I/O
EXPLOIT UNDERLYING HARDWARE

INSTRUCTOR NOTES

VG 823.1

17-1

Section 17

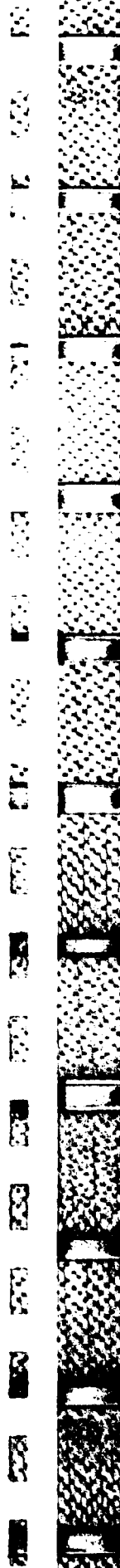
INTRODUCTION TO ADA DESIGN/CODE ASSESSMENT

VG 823.1

INSTRUCTOR NOTES

VG 823.1

17-11



ADA FOR SOFTWARE MANAGERS

INTRODUCTION
(Section 1)

ADA FEATURES
(Section 2-16)

INTRODUCTION TO ADA DESIGN/CODE ASSESSMENT
(Section 17)

CHARACTERISTICS OF GOOD ADA DESIGNS
(Section 18)

ADA IN PERSPECTIVE: REUSABILITY AND PORTABILITY
(SECTION 19)

INSTRUCTOR NOTES

ALLOW 2-2 1/2 HOURS FOR ENTIRE EXERCISE. DON'T RUSH IT TO MAKE UP TIME.

BREAK CLASS UP INTO GROUPS OF 3 OR 4. IN GROUPING THE STUDENTS, BE SURE ITS GROUP HAS AT LEAST ONE SHARP STUDENT (TO BE SURE EACH GROUP IS ABLE TO DO THE EXERCISE).

PASS OUT THE SEPARATE HANDOUT WITH SOLUTION 1. TELL THE GROUPS THAT THE HANDOUT IS THE SAME CODE AS IN THE COURSE NOTES BUT IS A MORE MANAGEABLE FORM. THEY CAN MAKE ANY NOTES THE WISH ON IT. INSTRUCT THE GROUPS THAT THEY ARE TO LOOK AT THE CODE AND AS A GROUP NOTE THEIR COMMENTS. (THE SOLUTION DOES SOLVE THE PROBLEM. WE WANT TO LOOK AT HOW IT DOES IT). TELL THE STUDENTS THEY HAVE 45-60 MINUTES FOR PART 1.

WHEN THE CLASS REASSEMBLES, HAVE EACH GROUP INFORMALLY PRESENT THEIR COMMENTS. INSTRUCTOR SHOULD MAKE A LIST OF THESE. IF MOST OF THE KEY POINTS HAVE NOT BEEN ADDRESSED BY THE CLASS, ADD THEM TO THE LIST (BUT PRESENT THEM IN THE FORM OF A QUESTION). ALLOW 15-30 MINUTES FOR DISCUSSION OF SOLUTION 1. THEN HAND OUT THE ALTERNATE SOLUTION. HAVE THE CLASS GO BACK TO THEIR GROUPS AND SPEND 30-45 MINUTES LOOKING AT THIS VERSION, KEEPING IN MIND THE ISSUES RAISED FROM SOLUTION 1. REASSEMBLE, HAVE GROUPS PRESENT, KEEPING A LIST OF THEIR COMMENTS. SUMMARIZE BY SHOWING HOW THE ALTERNATE SOLUTION COMMENTS ADDRESSED THE DEFICIENCIES NOTED FOR SOLUTION 1. ALLOW 10-20 MINUTES.

STUDENT PROBLEM

ASSESS A DESIGN.

THE TELEPHONE DIRECTORY SYSTEM ALLOWS THE USER TO LOOK UP NAMES, ADD NEW NAMES AND PHONE NUMBERS TO THE DIRECTORY, DELETE NAMES FROM THE DIRECTORY, AND EXIT THE SYSTEM.

INSTRUCTOR NOTES

WHAT HAPPENS WHEN THE SYSTEM USER ENTERS AN INCORRECT COMMAND? FROM OUR MAIN PROGRAM,
HOW IS THE DIRECTORY INITIALIZED?

CAN WE TELL FROM THE MAIN PROGRAM HOW THE SYSTEM IS ORGANIZED AND OPERATES?

HOW DOES THE USER TERMINATE AND EXIT THE SYSTEM (I.E. HOW DOES MAIN PROCEDURE END)?

WHAT HAPPENS IF THE USER DOESN'T ENTER A VALID COMMAND?

SOLUTION 1

```
with Text_IO, Dir_Mgr;
use Text_IO;
procedure Directory is

    Command : Character;
    Done : Boolean := False;
    procedure Process (Command : in Character;
                       Done : in out Boolean) is separate;

begin
    -- Directory

    New_Page; -- clear screen
    Put_Line ("Telephone Directory System");

    loop
        Put ("Enter Command (L: Lookup, " &
            "A: Add, D: Delete, Q: Quit): ");
        Get (Command);
        New_Line;
        Process (Command, Done);

        exit when Done; -- when Q is entered
    end loop;
end Directory;
```

INSTRUCTOR NOTES

IF WE NEED TO CHANGE THE DATABASE FORMAT, WHAT HAPPENS TO PROCESS?

HOW EASY IS IT TO ADD ANOTHER COMMAND OR TAKE ONE AWAY? WHAT WILL HAVE TO BE CHANGED?
(HINT: WE WOULD HAVE TO CHANGE EXECUTABLE CODE).

HOW EASY WOULD IT BE TO CHANGE OUR DATABASE?

THROUGHOUT HOW WELL DOES AN ENTITY'S NAME REFLECT WHAT IT DOES OR WHAT IT REPRESENTS?

SOLUTION 1 (Continued)

```
separate (Directory)
procedure Process (Command : in Character;
                  Done : in out Boolean) is
    Name      : Dir_Mgr.Name_String;
    Telno     : Dir_Mgr.Telno_String;
    Present, Room : Boolean := False;

    procedure Get_String (Str : out String) is separate;

begin
    -- Process

    case Command is
        when 'L' | 'l' => -- look up
            New_Line;
            Put ("Name : ");
            Get_String (Name); -- Get name to look up
            Dir_Mgr.Lookup (Name, Telno, Present);
            if not Present
            then
                Put_Line ("Name not found.");
            else
                Put ("Number is"); -- print the telephone number
                Put_Line (Telno);
            end if;
```

INSTRUCTOR NOTES

IF WE MUST CHANGE THE USER I/O INTERFACE, WHAT AND WHERE ARE CHANGES NECESSARY?

IF WE CHANGE THE SUBPROGRAM PARAMETERS, WHAT AND WHERE WILL CHANGES BE NECESSARY?

IS THIS HOW WE MIGHT SOLVE THE PROBLEM IF WE WERE CODING IN FORTRAN OR PASCAL? (YES!)

SOLUTION 1 (Continued)

```
when 'A' | 'a' => -- Add
  Put ("Name : ");
  Get_String (Name);      -- Get Name
  New_Line;
  Put ("Number : ");      -- And Number to Add
  Get_String (Telno);
  Dir_Mgr.Insert (Name, Telno, Present, Room);
  if Present
  then
    Put_Line ("Name already present.");
  elsif not Room
  then
    Put_Line ("Directory full.");
  else
    Put_Line ("Name added.");
  end if;
when 'D' | 'd' => -- Delete
  Put ("Name :");
  Get_String (Name);      -- Get Name to Delete
  Dir_Mgr.Delete (Name, Present);
  if not Present
  then
    Put_Line ("Name not present.");
  else
    Put_Line ("Name deleted.");
  end if;
when 'Q' | 'q' =>
  Done := True;
when others =>
  Put_Line ("Invalid command.");
end case;
end Process;
```


SOLUTION 1 (Continued)

package Dir_Mgr is

```
    Name_String_Size : constant := 20;  
    Telno_String_Size : constant := 12;
```

```
    subtype Name_String is String (1 .. Name_String_Size);  
    subtype Telno_String is String (1 .. Telno_String_Size);
```

```
    procedure Lookup (Name      : in Name_String;  
                     Telno     : out Telno_String;  
                     Present   : in out Boolean);
```

```
    procedure Insert (Name      : in Name_String;  
                     Telno     : in Telno_String;  
                     Present   : in out Boolean;  
                     Room      : in out Boolean);
```

```
    procedure Delete (Name      : in Name_String;  
                     Present   : in out Boolean);  
  
end Dir_Mgr;
```


17-71

SOLUTION 1 (Continued)

```
package body Dir_Mgr is

  type Dir_Entry is
    record
      Name : Name_String;
      Telno : Telno_String;
    end record;

  Size : constant := 100;
  subtype Index_Type is Integer range 0 .. Size;
  type Directory_Type is array (1 .. Index_Type'Last) of Dir_Entry;
  Directory : Directory_Type;
  Length: Index_Type;
  -- the code for the two procedures following are not included

  procedure Get_Directory (Directory : in out Directory_Type;
    Index : out Index_Type) is separate;

  procedure Save_Directory is separate;
```

17-81

SOLUTION 1 (Continued)

```
procedure Find (Name_To_Find : in Name_String;
               Entry_Location : out Index_Type;
               Present : in out Boolean) is
    Found: Boolean := false;
begin -- Find
    Entry_Location := 0;
    for Position in 1 .. Length
    loop -- thru Directory looking for Name_To_Find
        Present := Directory (Position).Name =
            Name_To_Find;
        if Present
        then
            Entry_Location := Position;    -- return INDEX Location
            exit;
        end if;
    end loop;
end Find;

procedure Lookup (Name : in Name_String;
                 Telno : out Telno_String;
                 Present : in out Boolean) is
    Entry_Location : Index_Type;
begin -- Lookup
    Find (Name, Entry_Location, Present);
    if Present
    then
        Telno := Directory (Entry_Location).Telno;
        -- return telephone
        -- number associated
        -- name
    end if;
end Lookup;
```

INSTRUCTOR NOTES

WHAT HAPPENS IF DIRECTORY IS EMPTY?

VG 823.1

17-91

SOLUTION 1 (Continued)

```

procedure Insert (Name      : in Name_String;
                  Telno     : in Telno_String;
                  Present   : in out Boolean;
                  Room      : in out Boolean) is
    Entry_Location : Index_Type;
begin
    -- Insert
    Room := False;
    Find (Name, Entry_Location, Present);
    if not Present
    then
        Room := Length < Size;  -- check to see if room exists for
                                -- additional entry
        if Room
        then
            -- add entry
            Length := Length + 1;
            Directory(Length).Name := Name;
            Directory(Length).Telno := Telno;
            end if;
        end if;
        Save_directory;
    end Insert;

procedure Delete (Name      : in Name_String;
                  Present   : in out Boolean) is
    Entry_Location : Index_Type;
begin
    -- Delete
    Find (Name, Entry_Location, Present);
    if Present
    then
        -- Delete entry and close gap
        for Ent_Num in Entry_Location .. Length - 1
        loop
            Directory (Ent_Num) := Directory (Ent_Num + 1);
        end loop;
        Length := Length - 1;
    end if;
    Save_Directory;
    Delete;
end Delete;

begin
    -- Dir_Mgr
    Get_Directory (Directory, Length);  -- Initialize Directory
    end Dir_Mgr;

```

INSTRUCTOR NOTES

OVERALL, DID OBJECT NAMES (INCLUDING PROGRAM UNITS) ACCURATELY REFLECT THEIR FUNCTION?

VG 823.1

17-101

SOLUTION 1 (Continued)

```
separate (Directory.Process)
procedure Get_String (Str : out String) is
  Last : Natural;
begin
  -- Get String
  Get_Line (Str, Last);
  Str (Last+1 .. Str'Last) := (Last+1 .. Str'Last => ' '); -- blank fill
  if not End_Of_Line
  then
    Skip_Line;
    end if;
  end Get_String;
```


INSTRUCTOR NOTES

PROGRAM UNITS HAVE BEEN RENAMED TO REFLECT THEIR FUNCTIONS. MAIN PROGRAM SHOWS THE ENTIRE SYSTEM AT A GLANCE; FOR THE DETAILS WE CAN LOOK AT THE RESPECTIVE PACKAGES. THE MAIN PROGRAM COORDINATES I/O INTERFACE AND ROUTES REQUEST FOR NECESSARY PROCESSING.

GETTING AND SAVING THE DIRECTORY ARE DONE WITHIN THE MAIN PROGRAM FOR SYMMETRY. (WE CAN "GET" THE DIRECTORY FROM WITHIN THE PACKAGE, BUT WE CANNOT "SAVE" THE DIRECTORY IN THE SAME PLACE. ADA REQUIRES EXPLICIT TESTING AND CLOSING OF FILES.)

AN ALTERNATE VERSION OF THE DIRECTORY SYSTEM

```
with Text_IO; use Text_IO;
with Directory_Services; use Directory_Services;

procedure Directory_Manager is
type Request_State is (Lookup, Add, Delete, Quit);
Name : Name_Type;
Telephone_Number : Telephone_Number_Type;
Request : Request_State;

procedure Get_Input_Request (Name      : out Name_Type;
                             Telephone_Number : out Telephone_Number_Type;
                             Request      : out Request_State)
is separate;

begin -- Directory_Manager
  Get_Directory;
loop
  begin
    Get_Input_Request (Name, Telephone_Number, Request);
    case Request is
      when Lookup => Lookup_Entry (Name, Telephone_Number);
                    Put ("Telephone number is ");
                    Put_Line (Telephone_Number);

      when Add    => Insert_Entry (Name, Telephone_Number);
                    Put_Line ("Name added");

      when Delete => Delete_Entry (Name);
                    Put_Line ("Name deleted");

      when Quit  => exit;
    end case;
  exception
    when Name_Not_In_Database => Put_Line ("Name not found");
    when Directory_Full      => Put_Line ("Directory full");
  end;
end loop;
Save_Directory;
exception
  when Read_Error => Put_Line ("Cannot read directory file");
  when Write_Error => Put_Line ("Cannot save directory file");
end Directory_Manager;
```

INSTRUCTOR NOTES

ALL USER INTERFACE IS LOCALIZED.

IF INCORRECT COMMANDS ARE ENTERED, THE USER IS PROMPTED AND ALLOWED TO TRY AGAIN
(SELF-CHECKING I/O).

THIS SAME SEQUENCE OF CODE CAN BE USED FOR ANY COMMAND; THUS WE DON'T HAVE TO REPEAT THE
SAME I/O COMMAND FOR EACH REQUEST FUNCTION.

VG 823.1

17-121

ALTERNATE (Continued)

```
separate (Directory_Manager)
procedure Get_Input_Request (Name
                             Telephone_Number : out Telephone_Number_Type;
                             Request           : out Request_State) is

package Command_IO is new Text_IO.Enumeration_IO (Request_State);
use Command_IO;
procedure Get_Name (Name : out Name_Type) is separate;
procedure Get_Number (Telephone_Number : out Telephone_Number_Type)
is separate;

begin -- Get_Input_Request
loop
begin
-- Enter prompt and read command
New_Line;
Put ("Enter Command (Lookup, Add, Delete, Quit):");
Get (Request);
New_Line;

-- Get input from user
case Request is
when Add => Get_Name(Name); Get_Number (Telephone_Number);
when Lookup | Delete => Get_Name (Name);
when Quit => exit;
end case;
exception
-- To handle user keying errors
when Data_Error => Put ("Invalid command, try again");
end;

end loop;

end Get_Input_Request;
```

INSTRUCTOR NOTES

ALL DIRECTORY SERVICES ARE COLLECTED TOGETHER WITH NO EXTRANEIOUS DECLARATIONS.

NAME CHANGE OF PROGRAM UNIT TO REFLECT ITS PURPOSE. NAMES OF ENTITIES, IN GENERAL ARE BETTER.

VG 823.1

17-131

ALTERNATE (Continued)

```
package Directory_Services is

-- Types used in user interfaces
subtype Name_Type is String (1 .. 20);
subtype Telephone_Number_Type is String (1 .. 15);

-- Primary operations allowed
procedure Lookup_Entry (Name
                        : in Name_Type;
                        Telephone_Number : out Telephone_Number_Type);
procedure Insert_Entry (Name
                       : in Name_Type;
                       Telephone_Number : in Telephone_Number_Type);
procedure Delete_Entry (Name
                       : in Name_Type);

-- Database service operations
procedure Get_Directory;
procedure Save_Directory;

-- Exceptional conditions
Name_Not_In_Database : exception;
Directory_Full       : exception;
Read_Error           : exception;
Write_Error           : exception;

end Directory_Services;
```

INSTRUCTOR NOTES

ALL DECLARATIONS RELATED TO THE DATABASE ARE FOUND IN ONE PACKAGE. THE DATABASE IS IMPLEMENTED AS AN ABSTRACT OBJECT. THE OPERATIONS ON THE TYPE MUST BE PROVIDED AT THE SAME TIME.

THE REST OF THE SYSTEM IS NOT AS DEPENDENT ON OUR ENTRIES BEING OF STRING FORMAT.

IF WE NEED TO CHANGE THE LENGTH OF THE ENTRIES OF THE DATABASE WE KNOW EXACTLY WHERE TO GO AND THEN THE REST OF THE SYSTEM WILL NOT NEED MODIFICATION.

ALTERNATE (Continued)

```
package body Directory_Services is

type Directory_Entry is
record
    Name           : Name_Type;
    Telephone_Number : Telephone_Number_Type;
end record;

subtype Index is Integer range 0 .. 100;
type Directory_Type is array (1 .. Index'Last) of Directory_Entry;
Directory : Directory_Type;
Current_Database_Length : Index := 0;
```

```
procedure Find_Entry (Name_To_Find : in Name_Type;
                     Entry_Location : out Index;
                     Present_In_Database : out Boolean) is

begin -- Find Entry
    Present_In_Database := False;
    for Position in 1 .. Current_Database_Length loop
        if Directory(Position).Name = Name_To_Find then
            Present_In_Database := True;
            Entry_Location := Position;
            exit;
        end if;
    end loop;
end Find_Entry;
```

INSTRUCTOR NOTES

SINCE "FIND" IS USED BY THREE OF THE SERVICES IT SEEMED REASONABLE TO DEFINE IT WITHIN THE SAME PHYSICAL STRUCTURE BUT THAT IT WOULD BE CLEARER IF THE SERVICES WERE SUBUNITS.

CODE FOR FIND IS CLEANER, MORE ADA-LIKE IN STYLE.

ALTERNATE (Continued)

```
procedure Lookup_Entry (Name
    Telephone Number : in Name_Type;
    is separate;
    Telephone Number : out Telephone_Number_Type)

procedure Insert_Entry (Name
    Telephone number : in Name_Type;
    is separate;
    Telephone number : in Telephone_Number_Type)

procedure Delete_Entry (Name
    Telephone number : in Name_Type;
    is separate;
    Telephone number : in Telephone_Number_Type)

procedure Get_Directory is separate;
procedure Save_Directory is separate;

end Directory_Services;
```

INSTRUCTOR NOTES

EACH OF THE REQUEST FUNCTION PROCEDURES BECOMES VERY SIMPLE AND STRAIGHTFORWARD.

IF CHANGES NEED TO BE MADE TO ANY OF THEM, IT IS EASIER TO FIND THAN BEFORE WITH A LARGE CASE STATEMENT. (IT IS TOO EASY NOT TO SEE PART OF THE CODE RELEVANT TO A PARTICULAR CASE OPTION OR TO MODIFY CODE THAT REALLY BELONGS TO ANOTHER OPTION.)

VG 823.1

17-161

ALTERNATE (Continued)

```
separate (Directory_Services)
procedure Lookup_Entry (Name           : in Name_Type);
                        Telephone_Number : out Telephone_Number_Type) is
    Entry_Location : Index;
    Entry_Found    : Boolean;
begin -- Lookup_Entry
    Find_Entry (Name, Entry_Location, Entry_Found);
    if Entry_Found then
        Telephone_Number := Directory(Entry_Location).Telephone_Number;
    else
        raise Name_Not_In_Database;
    end if;
end Lookup_Entry;
```

INSTRUCTOR NOTES

NOTE EASE OF ASSIGNMENT STATEMENT (I.E., REFLECTS A NATURAL SOLUTION STYLE).

NESTED IF STRUCTURE IS COMPLEX. AN ELSIF IS APPROPRIATE.

INSERT IS BEING USED TO PERFORM MORE THAN ONE FUNCTION. WE MIGHT WANT ANOTHER COMMAND TO CHANGE AN EXISTING ENTRY.

ALTERNATE (Continued)

```
separate (Directory_Services)

procedure Insert_Entry (Name           : in Name_Type);
                        Telephone_Number : in Telephone_Number_Type) is

    Entry_Location : Index;
    Entry_Found    : Boolean;

begin -- Insert_Entry

    Find_Entry (Name, Entry_Location, Entry_Found);
    -- "Insert is used to insert a new entry or to
    -- change the number of an existing entry
    if Entry_Found then

        Directory(Entry_Location).Telephone_Number := Telephone_Number;

    elsif Current_Database_Length < Directory'Last then
        Current_Database_Length := Current_Database_Length + 1;
        Directory (Current_Database_Length).Name := Name;
        Directory (Current_Database_Length).Telephone_Number := Telephone_Number;
    else
        raise Directory_Full;
    end_if;

end Insert_Entry;
```

INSTRUCTOR NOTES

SAME COMMENTS AS PREVIOUS SLIDE.

NOTE USE OF SLICE ASSIGNMENT (RATHER THAN A LOOP).

VG 823.1

17-181

ALTERNATE (Continued)

separate (Directory_Services)

procedure Delete_Entry (Name : in Name_Type) is

 Entry_Location : Index;
 Entry_Found : Boolean;

begin -- Delete_Entry

 Find_Entry (Name, Entry_Location, Entry_Found);

 if Entry_Found then

 Directory (Entry_Location .. Current_Database_Length - 1) :=

 Directory (Entry_Location + 1 .. Current_Database_Length);

 Current_Database_Length := Current_Database_Length - 1;

 else

 raise Name_Not_In_Database;

 end if;

end Delete_Entry;

INSTRUCTOR NOTES

YOU CAN MENTION, OR EVEN DO, THE FOLLOWING EXERCISES:

- 1) CHANGE THE DATABASE REPRESENTATION. FOR EXAMPLE, KEEP THE LIST SORTED BY NAME, SO THAT LOOK-UP IS FASTER
- 2) ADD A COMMAND TO QUERY THE NAME GIVEN A PHONE NUMBER
- 3) ALLOW THE USER TO:
 - READ A DIRECTORY FROM A FILE OF HIS OWN CHOICE
 - SAVE THE DIRECTORY INTO A FILE OF HIS OWN CHOICE
 - REINITIALIZE THE DIRECTORY IN MAIN MEMORY (PRESUMABLY TO START CREATING A NEW DIRECTORY)
 - PRINT THE WHOLE DIRECTORY

THE IMPORTANT POINT IS THAT IT'S EASY TO SEE EXACTLY WHAT NEEDS TO BE CHANGED.

ALTERNATE (Continued)

separate (Directory_Manager.Get_Input_Request)

procedure Get_Name (Name : out Name_Type) is
 Last : Natural;

begin -- Get_Name

 Put ("Please enter person's name (20 characters):");

 Get_Line (Name, Last);

 Name (Last+1 .. Name'Last) := (Last+1 .. Name'Last = ' ');

 if not End_Of_Line then

 Skip_Line;

 end if;

end Get_Name;

separate (Directory_Manager.Get_Input_Request)

procedure Get_Number (Telephone_Number : out Telephone_Number_Type) is

 Last : Natural;

begin -- Get_Number

 Put ("Please enter telephone number (4 digits):");

 Get_Line (Telephone_Number, Last);

 Telephone_Number (Last+1 .. Telephone_Number'Last) := (Last+1 .. Telephone_Number'Last => ' ');

 if not End_Of_Line then

 Skip_Line;

 end if;

end Get_Number;

INSTRUCTOR NOTES

THIS SECTION FORMALIZES MOST OF THE IDEAS AND CONCEPTS DISCUSSED AS A RESULT OF THE EXERCISE.

VG 823.1

18-1

18-1 18-2 18-3 18-4 18-5 18-6 18-7 18-8 18-9 18-10 18-11 18-12 18-13 18-14 18-15 18-16 18-17 18-18 18-19 18-20 18-21 18-22 18-23 18-24 18-25 18-26 18-27 18-28 18-29 18-30 18-31 18-32 18-33 18-34 18-35 18-36 18-37 18-38 18-39 18-40 18-41 18-42 18-43 18-44 18-45 18-46 18-47 18-48 18-49 18-50 18-51 18-52 18-53 18-54 18-55 18-56 18-57 18-58 18-59 18-60 18-61 18-62 18-63 18-64 18-65 18-66 18-67 18-68 18-69 18-70 18-71 18-72 18-73 18-74 18-75 18-76 18-77 18-78 18-79 18-80 18-81 18-82 18-83 18-84 18-85 18-86 18-87 18-88 18-89 18-90 18-91 18-92 18-93 18-94 18-95 18-96 18-97 18-98 18-99 18-100

Section 18

CHARACTERISTICS OF GOOD ADA DESIGNS

VG 823.1

18-1i

ADA FOR SOFTWARE MANAGERS

INTRODUCTION
(Section 1)

ADA FEATURES
(Section 2-16)

INTRODUCTION TO ADA DESIGN/CODE ASSESSMENT
(Section 17)

CHARACTERISTICS OF GOOD ADA DESIGNS
(Section 18)

ADA IN PERSPECTIVE: REUSABILITY AND PORTABILITY
(SECTION 19)

INSTRUCTOR NOTES

FOR ANY PROGRAM, WHAT DO WE WANT TO SEE IN A DESIGN. USE THE EXERCISE TO GIVE EXAMPLE.
FOR EXAMPLE:

UNDERSTANDABILITY: MAIN PROCEDURE OF VSN 2 IS EASIER TO HAVE A TOP LEVEL VIEW OF THE
SYSTEM WORKINGS.

RELIABILITY: WITH VSN 1 ON INCORRECT COMMAND WOULD CRASH THE SYSTEM.

QUALITIES OF A GOOD DESIGN

- MODIFIABILITY
 - THE REGION OF EFFECT FROM MODIFICATIONS IS KEPT TO A MINIMUM
- UNDERSTANDABILITY
 - THE EASE OF FUNCTIONAL UNDERSTANDING OF A SYSTEM AT THE TOP LEVEL
- RELIABILITY
 - THE ROBUSTNESS OR THE EASE WITH WHICH A SYSTEM RECOVERS OR PREVENTS SYSTEM FAILURES
- EFFICIENCY
 - PERFORMANCE ENGINEERED INTO A SYSTEM AFTER FUNCTIONS ESTABLISHED
- REUSABILITY
 - FUNCTIONS DESIGNED FOR THE USE OF SOFTWARE COMPONENTS

INSTRUCTOR NOTES

SOME GUIDELINES FOR A TECHNICAL MANAGER TO USE IN EVALUATING ADA DESIGN. THE LIST SUMMARIZES WHAT WILL BE NEXT DISCUSSED IN FURTHER DETAIL

NON-CENTRALIZED DATABASE = PACKAGING THE DATA SO IT IS KNOWN AND AVAILABLE ONLY WHERE NEEDED. THIS LIMITS THE USE OF GLOBAL DATA PREVALENT IN FORTRAN COMMON, JOVIAL COMPOOL.

CHARACTERISTICS OF GOOD ADA DESIGN

- QUALITIES OF A GOOD DESIGN (OF PREVIOUS SLIDE)
- READABLE
- DELIBERATE ABSTRACTION RATHER THAN VAGUENESS
- COMPLETE FUNCTIONAL DECOMPOSITION BEFORE PERFORMANCE
- NON-CENTRALIZED DATABASE ORGANIZATION
- SENSIBLE MODULARITY
- CONSERVATIVE USE OF LOW-LEVEL FEATURES
- CONSERVATIVE USE OF GOTO'S
- KNOWLEDGEABLE USE OF ADA CONSTRUCTS
- DESIGN STYLE IS NOT FORTRAN, PASCAL, ASSEMBLY LANGUAGE (NEW "MIND SET")

INSTRUCTOR NOTES

VERSION 2 OF THE EXERCISE READ MORE 'NATURALLY' AND REFLECTS WHAT WE CALL A CHANGE IN MIND SET (LOOK AT SYSTEM FROM USER'S VIEW RATHER THAN H/W).

VG 823.1

18-4i

READABLE

IN A GOOD ADA DESIGN

- CODE READS "NATURALLY" (I.E. CODE DIRECTLY REFLECTS THE REAL WORLD SOLUTION)
- WITH LITTLE EFFORT, THE SYSTEM STRUCTURE AND FUNCTIONS CAN BE UNDERSTOOD
- THE DESIGN REFLECTS THE SYSTEM USER'S VIEWPOINT, NOT THE HARDWARE CONSTRAINTS
- NAMES OF ENTITIES ACCURATELY REFLECT THEIR FUNCTION
- COMMENTS USED TO AUGMENT CODE NOT TO REPLACE POOR CHOICES OF NAMES AND LOGIC STRUCTURE

INSTRUCTOR NOTES

IN USING ABSTRACTION TO HIDE THE IMPLEMENTATION, WE DEFINE SAY A FUNCTION BUT NOT THE IMPLEMENTATION DETAILS. FOR EXAMPLE, `Lookup_Entry`, `Find_Entry`.

GOOD DESIGNERS USE THE EXPRESSION "I DON'T CARE HOW THIS IS IMPLEMENTED" TO MEAN "THERE ARE SEVERAL WORKABLE IMPLEMENTATIONS; I HAVE DEFINED THE INTERFACE SO THAT THE IMPLEMENTATION CHOICE CAN BE POSTPONED OR EVEN CHANGED LATER."

ABSTRACTION VS. VAGUENESS

IN A GOOD ADA DESIGN

- AN ABSTRACTION HIDES OR DEFERS IMPLEMENTATION POSSIBILITIES
- THE DESIGNER IS ABLE TO DESCRIBE IN SIMPLE ENGLISH THE INTENDED IMPLEMENTATION OR, BETTER, SEVERAL POSSIBLE IMPLEMENTATIONS
- THE INTERFACE IS USER-ORIENTED SO THE ABSTRACTION WILL BE EASY TO USE

IN A POOR ADA DESIGN

- THE ABSTRACTION IS USED TO HIDE A DESIGNER'S LACK OF THOROUGH PLANNING AND/OR A LACK OF UNDERSTANDING
- THE INTERFACE IS POORLY-PLANNED; A SIMPLE EXERCISE TO USE IT USUALLY REVEALS THAT THE ABSTRACTION IS INADEQUATE OR CLUMSY TO USE
- THE DESIGNER APPEARS TO BE GUESSING AT ONE IMPLEMENTATION WHEN ASKED TO DESCRIBE THE ABSTRACTION

INSTRUCTOR NOTES

POINTS OF GOOD ADA DESIGN VERY IMPORTANT. GENERALLY IN INDUSTRY PERFORMANCE IS
CONSIDERED FIRST AND FOREMOST.

VG 823.1

18-61

PERFORMANCE AND DESIGN

IN A GOOD ADA DESIGN

- THE FUNCTIONS ARE FIRMLY DEFINED FIRST. SYSTEM PERFORMANCE ESTIMATES AND DESIGN ADJUSTMENTS COME SECOND
- THE DESIGN DELIBERATELY ALLOWS FOR LATER TUNING

IN A POOR ADA DESIGN

- THE DESIGN IS CONCERNED PRIMARILY WITH PERFORMANCE (HOW CAN PERFORMANCE ESTIMATES BE ACCURATE IF THE FUNCTIONS ARE INCOMPLETELY DEFINED). IT IS VERY EASY TO SEE THAT THE SOFTWARE MEETS ITS PERFORMANCE CONSTRAINTS, BUT TOO HARD TO FIGURE OUT WHAT IT DOES.
- PERFORMANCE IS NOT CONSIDERED AT ALL

INSTRUCTOR NOTES

POOR ADA DESIGN - SOME EXAMPLES.

FORTRAN'S "COMMON"

JOVIAL'S "COMPOOL"

NOT USING GLOBAL DATA IS A NEW CONCEPT AND TAKES TRAINING AND KNOWLEDGEABLE MANAGERS AND QA TO MONITOR.

ESSENTIALLY, STUFF USED AS GLOBAL BEFORE SHOULD NOW BE PASSED AS PARAMETERS.

DATABASE ORGANIZATION

IN A GOOD ADA DESIGN

- THERE IS NO NEED TO USE GLOBAL DATA IN ADA
 - DATA SHOULD BE DECLARED WHERE IT IS USED
 - AS A RESULT, LESS "HUNTING" FOR AN OBJECTS DECLARATION REQUIRED
- GLOBAL DATA SHOULD BE THE EXCEPTION NOT THE RULE
 - THIS STYLE REQUIRES MUCH MORE DESIGN AND PLANNING
 - NATURAL RESISTANCE SINCE PEOPLE DON'T LIKE TO PLAN AND WOULD RATHER START PROGRAMMING

IN A POOR ADA DESIGN

- LARGE QUANTITIES OF DATA COLLECTED INTO SINGLE PROGRAM UNITS

INSTRUCTOR NOTES

BY MODULARITY WHEN THE STRUCTURING OF THE SYSTEM HAS A DEFINITE PURPOSE IN MIND.

THIS IS WHERE THE LIFE-CYCLE COST BENEFITS CAN COME. BY APPROPRIATE GROUPING, THE EFFECT OF MODIFICATIONS IS LOCALIZED. FOR EXAMPLE, IN VERSION 2, Directory_Manager ONLY MANAGES THE DIRECTORY SYSTEM IT DOESN'T INTERFACE WITH THE USES FOR INPUT REQUESTS OR DATA.

VG 823.1

18-81

MODULARITY

IN A GOOD ADA DESIGN

- RELATED OBJECTS AND OPERATIONS ARE GROUPED TOGETHER
- THERE EXISTS A LOGICAL, SENSIBLE ORDER TO MODULARITY OF THE SYSTEM

IN A POOR ADA DESIGN

- HAPHAZARD OR UNCLEAR RELATIONSHIP OF ENTITIES IN A MODULE
- MODULARITY BASED ON PHYSICAL OR PERFORMANCE CONSIDERATIONS

INSTRUCTOR NOTES

IMPORTANT TO NOTE LOW-LEVEL FEATURES SHOULD BE USED FOR EXTERNAL HARDWARE INTERFACE ONLY
NOT TO INCREASE EFFICIENCY. FOR DESIGNERS OR PROGRAMMERS WITH ASSEMBLY LANGUAGE
BACKGROUNDS, IT COULD BE ESPECIALLY TEMPTING TO USE THESE FEATURES AS THEY WILL FEEL
MORE COMFORTABLE WITH THEM THAN WITH THE HIGH ORDER LANGUAGE FEATURES.

LOW-LEVEL FEATURES

IN A GOOD ADA DESIGN

- USED SPARINGLY
- USED TO INTERFACE WITH THE EXTERNAL HARDWARE

IN A POOR ADA DESIGN

- USED FREQUENTLY
- USED TO INCREASE PERFORMANCE (RATHER THAN FIND A BETTER DESIGN SOLUTION)
- USED TO CIRCUMVENT A PURPORTED DEFICIENCY OF THE LANGUAGE. USUALLY A SIGN THAT THE PROGRAMMER/DESIGNER IS THINKING IN SOME OTHER LANGUAGE.

INSTRUCTOR NOTES

VG 823.1

18-101

GOTO'S

IN A GOOD ADA DESIGN

- SPARINGLY USED
- POSSIBLE USE IS IN FINITE STATE MACHINES (GOTO NEXT STATE)

IN A POOR ADA DESIGN

- USED INSTEAD OF STRUCTURED PROGRAMMING CONSTRUCTS, EXCEPTIONS, EXITS

INSTRUCTOR NOTES

USE OF ADA FEATURES IS A PLANNED ACTIVITY.

EACH ADA FEATURE IS IN THE LANGUAGE FOR SPECIFIC REASONS AND SHOULD BE USED ACCORDINGLY. FOR EXAMPLE:

COMMON ALGORITHM, DIFFERENT DATA => GENERICS
INFORMATION HIDING => SPECIFICATION/BODY OF PROGRAM UNITS

VG 823.1

18-111

"KNOWLEDGEABLE" USE OF ADA FEATURES

IN A GOOD ADA DESIGN

- USED WITH DELIBERATE INTENT

FOR EXAMPLE:

- TO INCREASE MODULARITY
- TO INCREASE READABILITY

IN A POOR ADA DESIGN

- USED BECAUSE THE CONSTRUCT WAS "HANDY"

FOR EXAMPLE:

- PRIVATE TYPES USED TO AVOID CAREFUL PLANNING OF OBJECT ACCESS NEEDS
- LOW-LEVEL FEATURES USED BECAUSE OF GREATER FAMILIARITY
- GOTO'S AND EXCEPTIONS USED TO AVOID STRUCTURED PROGRAMMING LOGIC

INSTRUCTOR NOTES

POSSIBLY BEFORE PRESENTING THE EXAMPLE OF ARRAYS, QUICKLY REFER BACK TO THE DIRECTORY SYSTEM. THE FIRST VERSION IS BASICALLY FORTRAN OR PASCAL THINKING IN ADA SYNTAX.

"MIND SET" CHANGE

ADA HAS FEATURES OR CONSTRUCTS THAT ALLOW US TO SOLVE PROBLEMS IN A NEW, MORE "NATURAL", STYLE. OUR DESIGN AND CODE CAN MORE ACCURATELY MATCH THE REAL WORLD. TO EFFECTIVELY USE THE POWER OF ADA, THIS CHANGE OF PERSPECTIVE MAY BE NECESSARY.

IN A POOR ADA DESIGN

- THE CONSTRUCTS AND METHODS USED IN DESIGNING FOR FORTRAN, PASCAL, JOVIAL,
OR ASSEMBLY LANGUAGE APPEAR IN ADA SYNTAX

FOR EXAMPLE:

NON-ADA STYLE

-- MULTIDIMENSIONAL ARRAY
-- CRYPTIC IDENTIFIERS
-- LITTLE USE OF ATTRIBUTES
-- RECODE ALGORITHM FOR EACH

TYPE OF DATA

ADA STYLE

-- ARRAY OF ARRAYS
-- EXPANDED IDENTIFIERS
-- ATTRIBUTES USED
-- GENERICS

INSTRUCTOR NOTES

WE CAN SHOW WAYS TO USE FEATURES, GIVE SOME DESIGN CHARACTERISTIC GUIDELINES, BUT THE
MANAGER NEEDS TO TEMPER THESE WITH HIS/HER OWN GUT FEELINGS.

VG 823.1

18-131

MANAGER'S GENERAL ADA GUIDE

DON'T COUNT OCCURRENCES OF (OR LACK OF) GOOD ADA DESIGN FEATURES, RATHER ASK:

- "IS THE DESIGN OR CODE EASY TO UNDERSTAND?"
- "IS IT EASY TO GET AN OVERALL 'BIG PICTURE'?"
- "DOES IT FEEL NATURAL?"
- "DOES IT SPEAK THE USER'S LANGUAGE?"
- "DOES IT MAKE SENSE?"

ASK DESIGNERS OR PROGRAMMERS TO EXPLAIN WHY THEY USED OR DID NOT USE A PARTICULAR ADA
CONSTRUCT OR DESIGN FEATURE

INSTRUCTOR NOTES

ONE OF ADA'S PRIMARY GOALS WAS TO INCREASE THE REUSABILITY AND PORTABILITY OF SYSTEMS OR PARTS TO PRODUCE A SOFTWARE COMPONENTS INDUSTRY (SIMILAR TO H/W COMPONENTS). THIS SECTION PROVIDES SOME GUIDELINES AND A GRASP OF THE ISSUES INVOLVED. THIS SECTION THUS SUMMARIZES THE ENTIRE COURSE.

ADA FOR SOFTWARE MANAGERS

INTRODUCTION
(Section 1)

ADA FEATURES
(Section 2-16)

INTRODUCTION TO ADA DESIGN/CODE ASSESSMENT
(Section 17)

CHARACTERISTICS OF GOOD ADA DESIGNS
(Section 18)

ADA IN PERSPECTIVE: REUSABILITY AND PORTABILITY
(SECTION 19)

INSTRUCTOR NOTES

VG 823.1

19-21

REUSABILITY

- IMPLIES THAT A SECTION OF SOFTWARE DEVELOPED FOR ONE APPLICATION CAN BE REUSED FOR ANOTHER APPLICATION
- APPLIES WITHIN A PROJECT AND ACROSS PROJECTS
- WILL NOT HAPPEN BY CHANCE
- REQUIRES ADVANCE PLANNING, DESIGN, AND EXPERIENCE
- REQUIRES CHANGE IN VIEWING OF PROBLEM SOLUTIONS AS COMBINATIONS OF SOFTWARE COMPONENT OR CONSTRUCTS LIKE HARDWARE CHIPS

INSTRUCTOR NOTES

VG 823.1

19-31

CANDIDATES FOR REUSABLE COMPONENTS

GENERAL PURPOSE ALGORITHMS OR DATA CONSTRUCTS

FOR EXAMPLE:

- MAN/MACHINE INTERFACES
- DEVICE DRIVERS
- SORT ROUTINES
- SEARCH ROUTINES
- STACKS AND QUEUES
- TABLES

INSTRUCTOR NOTES

VG 823.1 .

19-41

ADA FEATURES AND REUSABILITY

- PACKAGES TO ENCAPSULATE THE SOFTWARE COMPONENT
- LIBRARY UNITS TO BUILD A DESIGNER'S "TOOL KIT"
- GENERICS WERE SPECIFICALLY DESIGNED TO AID IN THE PRODUCTION OF REUSABLE SOFTWARE COMPONENTS
- USE OF ATTRIBUTES CAN INCREASE REUSABILITY

CONTEXT:

```
N : Integer;  
type Arr_Type is array (1 .. N) of Integer;  
V : Arr_Type;
```

EXAMPLE:

```
for Index in V'Range ... RATHER THAN  
for Index in 1 .. 60 ... WHERE V CONSTRAINED TO VALUES 1 .. 60  
IN ONE APPLICATION, BUT 1 .. 35 IN ANOTHER
```

INSTRUCTOR NOTES

VG 823.1

19-5i

PORTABILITY

IMPLIES A SECTION OF SOFTWARE DEVELOPED ON ONE COMPUTER SYSTEM CAN BE RUN ON ANY OTHER SYSTEM WITH THE SOFTWARE FUNCTIONING IN THE SAME WAY AS ON THE ORIGINAL SYSTEM.

THE AMOUNT OF MODIFICATION TO THE ORIGINAL CODE NECESSARY TO ACCOMPLISH THIS SAMENESS OF FUNCTION IS THE DEGREE OF PORTABILITY.

PERFECT PORTABILITY (NO CHANGES REQUIRED) IS SELDOM ACHIEVABLE. REALISTICALLY, A PROGRAM IS CONSIDERED PORTABLE IF:

- THE PARTS THAT WILL NEED CHANGES ARE PRE-IDENTIFIED (NO SURPRISES)
- NO REDESIGN IS REQUIRED (ONLY RE-CODING)
- THE AMOUNT OF RE-CODING IS NEGLIGIBLE COMPARED TO THE TOTAL IMPLEMENTATION EFFORT

INSTRUCTOR NOTES

TECHNICAL MANAGERS SHOULD BE AWARE OF THIS.

VG 823.1

19-61

IMPORTANCE OF PORTABILITY

RELIABILITY

- EMBEDDED COMPUTER SYSTEMS GENERALLY DEVELOPED ON A HOST COMPUTER BUT THE EVENTUAL TARGET COMPUTER IS DIFFERENT THAN THE HOST
- PROGRAM NEEDS TO PERFORM THE SAME WHETHER RUN ON THE HOST OR TARGET SYSTEMS

DECREASE IN SOFTWARE COSTS

- HARDWARE TECHNOLOGICAL DEVELOPMENT IS INCREASING AS WELL AS SOFTWARE COSTS
- AS TARGET MACHINES ARE UPGRADED, SOFTWARE THAT HAS BEEN DEVELOPED FOR PORTABILITY CAN BE REUSED

INSTRUCTOR NOTES

THIS IS TO PROVIDE THE STUDENT WITH A FEEL FOR THE SCOPE OF THE ISSUES INVOLVED. NOT ALL ISSUES ARE PRESENTED AS IT IS FELT THIS SHOULD BE LEFT TO EXPERTS TO ADVISE IN THIS AREA. THE MANAGER NEEDS TO KNOW ENOUGH OF THE ISSUES TO BE ABLE TO ASK AND SEE IF DESIGNERS ARE PLANNING WITH PORTABILITY IN MIND.

SOME CHARACTERISTICS OF PORTABLE SOFTWARE

- DESIGN AND CODE ARE STRAIGHTFORWARD AND EASY TO UNDERSTAND
- DOCUMENTATION STATES THE IMPLEMENTATION CHARACTERISTICS REQUIRED OF THE PROGRAM AND THE PARTICULAR PORTABILITY CHARACTERISTICS OF EACH PROGRAM UNIT
- LOGICALLY RELATED OBJECTS ARE PHYSICALLY GROUPED TOGETHER TO LOCALIZE THE EFFECT OF CHANGES
- PACKAGES USED TO ENCAPSULATE EXPECTED AREAS OF MODIFICATION, THUS ONLY THE PACKAGE BODY NEEDS REWRITING BUT NONE OF THE PROGRAM UNITS USING THE PACKAGE

INSTRUCTOR NOTES

VG 823.1

19-81

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

CHARACTERISTICS (Continued)

CONSERVATIVE USE OF IMPLEMENTATION DEPENDENT FEATURES, FOR EXAMPLE:

- PRAGMAS (COMPILER DIRECTIVES)
- ATTRIBUTES FIRST AND LAST
- THE PREDEFINED EXCEPTIONS Numeric_Error, Constraint_Error

INSTRUCTOR NOTES

VG 823.1

19-91

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

CHARACTERISTICS (Continued)

REPRESENTATION SPECIFICATIONS AND MACHINE CODE ARE LOCALIZED IN PACKAGES

RESTRICTED USE OF Low_Level_IO. IF NEEDED, ENCAPSULATE IN PACKAGES

SYSTEM PERFORMANCE (EITHER SPACE OR TIME) IS NOT DEPENDENT ON MACHINE OR RUN-TIME
SPECIFIC OPTIMIZATIONS

INSTRUCTOR NOTES

VG 823.1

19-101

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

IN CONCLUSION

- REUSABILITY AND PORTABILITY REQUIRES CAREFUL PLANNING IN DESIGN
- REUSABILITY AND PORTABILITY CONCERNS ARE COMPLEX, TRADE-OFFS MAY BE NECESSARY
- DEGREE OF PORTABILITY IS HARD TO ASSESS BY INSPECTION OR BY EXPERIMENT. EXPLICIT TRAINING IS NECESSARY

INSTRUCTOR NOTES

SIMPLICITY OF DESIGN AND CODE IS THE KEY TO MANAGING THE COMPLEXITY OF SYSTEMS.

**A FINAL NOTE ...
THE KEY TO COMPLEXITY
IS
SIMPLICITY**

NO-A165 315

ADA (TRADEMARK) TRAINING CURRICULUM ADA (REGISTERED
TRADEMARK) FOR SOFTWARE MANAGERS L201 TEACHER'S GUIDE
VOLUME 2(U) SOFTECH INC WALTHAM MA 1986

5/5

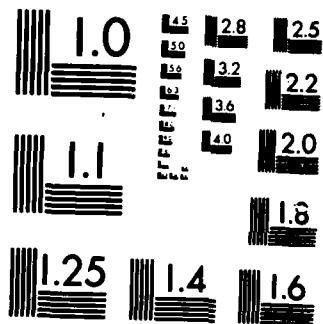
UNCLASSIFIED

DAB07-83-C-K506

F/G 5/9

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1063-A

Material: Ada for Software Managers (L201), Volume II A142431

We would appreciate your comments on this material and would like you to complete this brief questionnaire. The completed questionnaire should be forwarded to the address on the back of this page. Thank you in advance for your time and effort.

1. Your name, company or affiliation, address and phone number.

2. Was the material accurate and technically correct?

Yes ☐

No ☐

Comments:

3. Were there any typographical errors?

Yes ☐

No ☐

If yes, on what pages?

4. Was the material organized and presented appropriately for your applications?

Yes ☐

No ☐

Comments:

5. General Comments:

place
stamp
here

COMMANDER
US ARMY MATERIEL COMMAND
ATTN: AMCDE-SB (OGLESBY)
5001 EISENHOWER AVENUE
ALEXANDRIA, VIRGINIA 22233

END

FILMED

4-86

DTIC