

10-A198 365

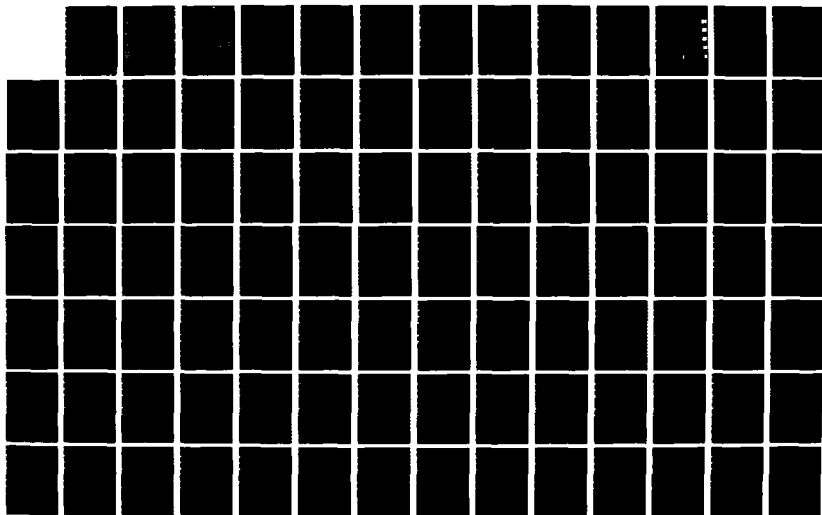
TUTORIAL TRACK I INTRODUCTION TO ADA(U) INFORMATION
SYSTEMS AND TECHNOLOGY CENTER W-P AFB OH ADA VALIDATION
FACILITY C ENGLE ET AL 89 JUN 87

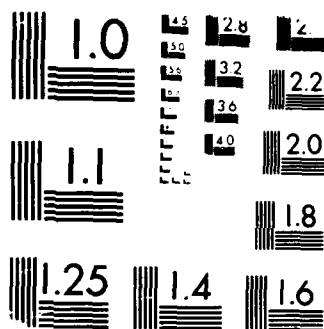
1/2

UNCLASSIFIED

F/G 12/5

NL





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

AD-A-190 365

TUTORIAL

TRACK I

INTRODUCTION TO ADA

By

Major Charles Engle, U.S. Military Academy

and

Lieutenant Tony Dominice, Keesler Air Force Base

12 22 34 2

UNCLASSIFIED

DTIC FILE COPY

(When Data Entered)

AD-A190 365

ACTION PAGE

READ INSTRUCTIONS
BEFORE COMPLETING FORM

12. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

2

Tutorial Track I. Introduction to Ada

5. TYPE OF REPORT & PERIOD COVERED
Tutorial, 9 June, 1987

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

MAJ Charles Engle, and LT Tony Dominice

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS

Ada Software Education and Training Team
Ada Joint Program Office, 3E114, The
Pentagon, Washington, D.C. 20301-308110. PROGRAM ELEMENT, PROJECT, TASK
AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Ada Joint Program Office
3E 114, The Pentagon
Washington, DC 20301-3081

12. REPORT DATE

June 9, 1987

13. NUMBER OF PAGES

124

14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office)

Ada Joint Program Office

15. SECURITY CLASS (of this report)

UNCLASSIFIED

15a. SECURITY CLASSIFICATION DOWNGRADING
SCHEDULE

N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20. If different from Report)

UNCLASSIFIED

DTIC

ELECTE

JAN 06 1988

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Training, Education, Training, Computer
Programs; Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document contains prints of viewgraphs presented at the
Introduction to Ada Tutorial, Track I June 9, 1987. Topics covered
were The Software Crisis, Technical Background, Basic Constructs,
Subprograms, Generics and Tasks.

DD FORM 1473

1 JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Introduction to Ada[®]

MAJOR CHUCK ENGLE

UNITED STATES MILITARY ACADEMY

DEPARTMENT OF GEOGRAPHY AND

COMPUTER SCIENCE

WEST POINT, N.Y. 10996

LT TONY DOMINICE

3390 TECH TRAINING GROUP

STARS TRAINING SECTION

KEESLER AFB, MS. 39534



Ada is a registered Trademark of the U.S. Government (AJPO)

A-1

OVERVIEW

- I. The Software Crisis
- II. Program Units
- III. Types
- IV. Control Statements
- V. Exceptions
- VI. Generics
- VII. Tasks
- VIII. Application Example

Types

Records

```
A_DRIVER : INSURANCE (GOOD);  
ANOTHER : INSURANCE (BAD);
```

```
begin
```

```
  A_DRIVER.NORMAL_RATE := 25;  
  A_DRIVER.DISCOUNT_RATE := 0.15;
```

```
  ANOTHER.NORMAL_RATE := 25;  
  ANOTHER.ADDITIONAL := 10;
```

Types

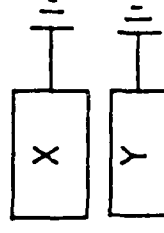
Access

- Pointer variables
- Allow for dynamic allocation of memory
- Objects created via an allocator

type POINTER is access INTEGER;

X, Y : POINTER; -- initialized to
 -- null

begin



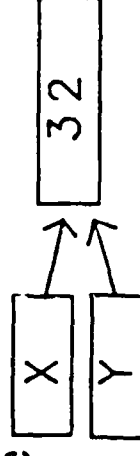
X := new INTEGER; -- allocate
 -- memory to X



X.all := 32; -- place 32 in the
 -- location pointed to
 -- by X



Y := X; -- X and Y point to the same
 -- location



Software Crisis

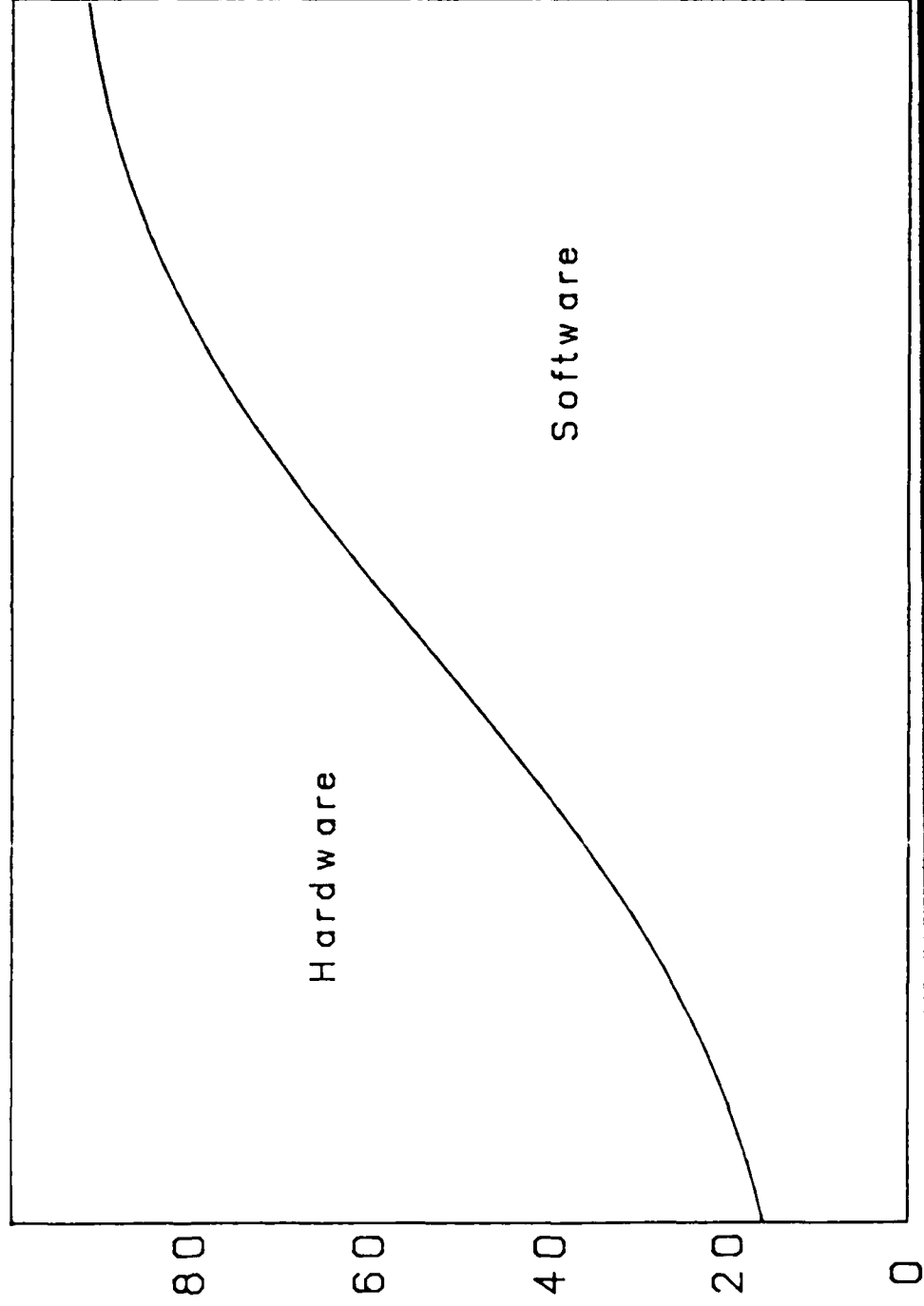
- Rising costs of software
- Unreliable
- Late
- Not maintainable
- Inefficient
- Not transportable

WHY??

- Too many languages
- Poor tools
- Changing technology
- Not enough trained people

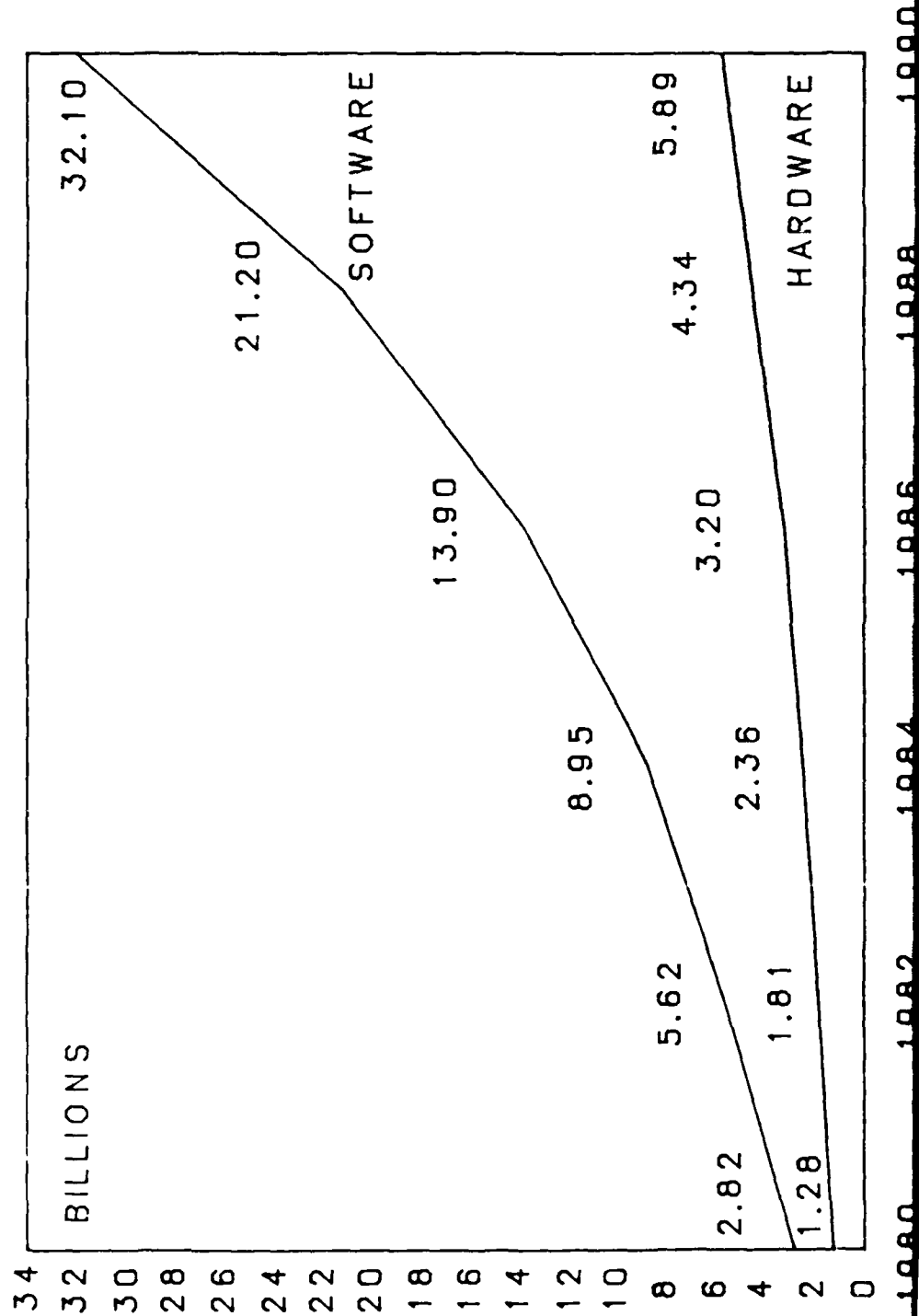
INABILITY TO MANAGE COMPLEX PROBLEMS

Software Crisis



Software Crisis

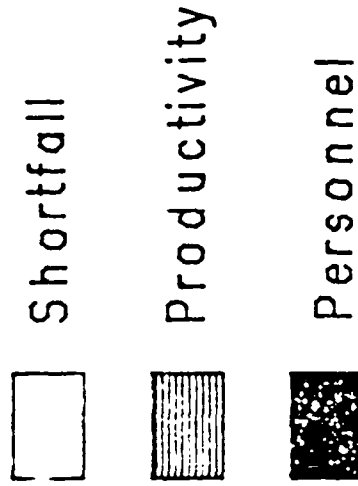
DoD Embedded Hardware/Software Costs



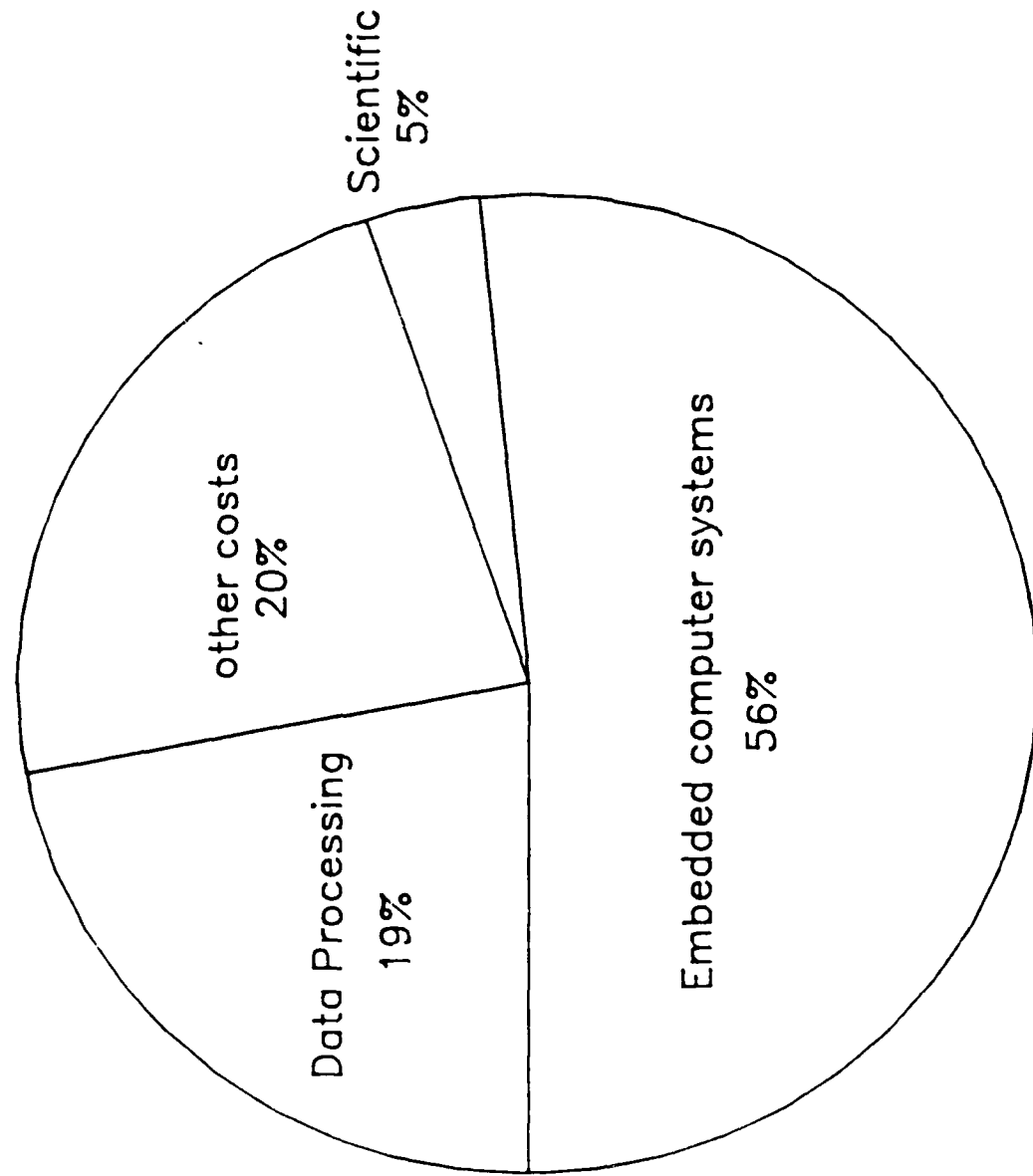
DEMAND FOR NEW SOFTWARE

50
40
30
20
10

ANNUAL PERCENTAGE INCREASES
(USING 1980 AS A BASELINE)



Software Crisis



Software Crisis

EMBEDDED SYSTEMS

- Large
- Long lived
- Continous change
- Physical constraints
- High reliability

EMBEDDED SYSTEMS SOFTWARE

- Severe reliability requirements
- Time and size constraints
- Parallel processing
- Real time control
- Exception handling
- Unique I/O

Software Crisis

SOLUTIONS

<u>Single Language</u>	<u>Improved Tools</u>	<u>Improved Methodologies</u>
------------------------	-----------------------	-------------------------------

Ada	Ada	Methodman
-----	-----	-----------

	APSE	??
--	------	----

(Ada		
-------	--	--

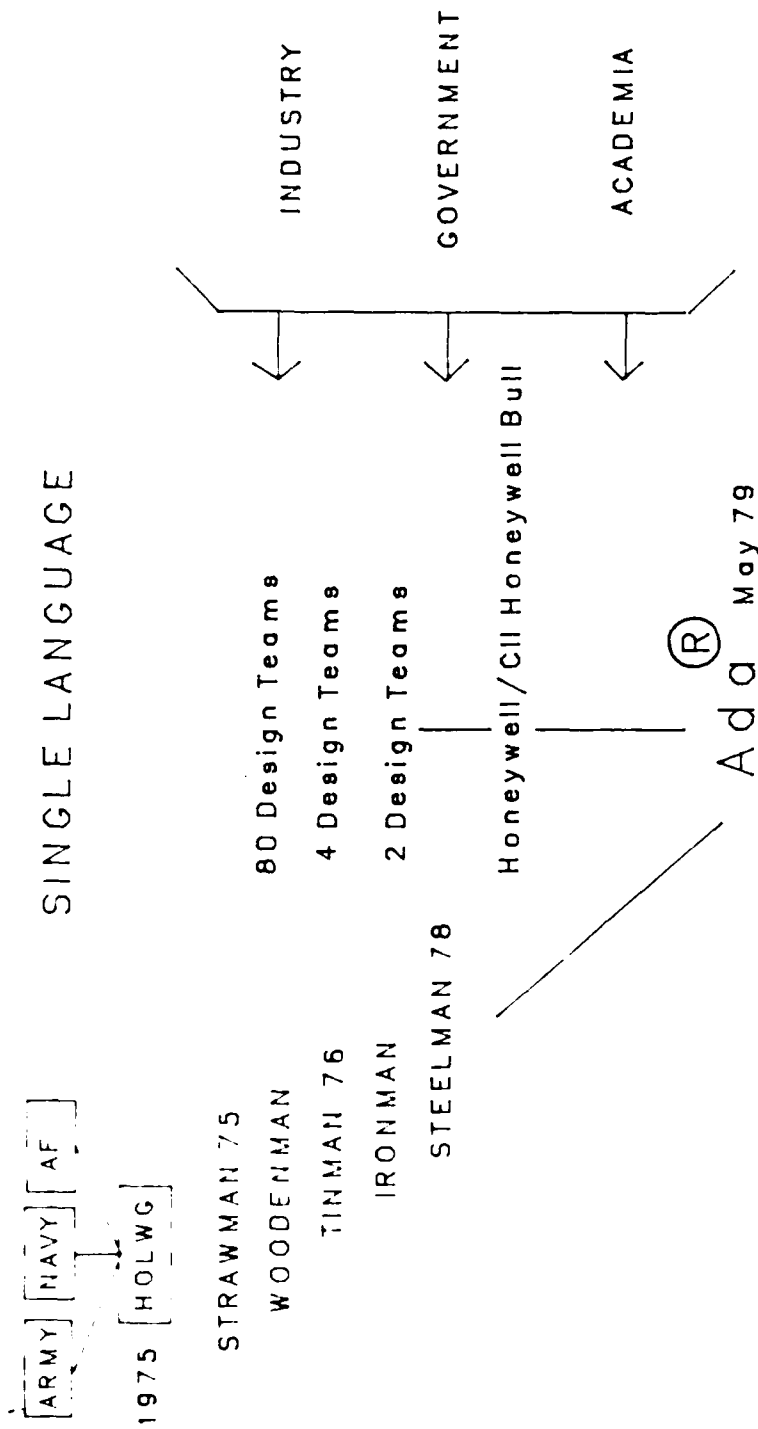
Program ming		
--------------	--	--

Support		
---------	--	--

Environment)		
---------------	--	--

SOFTWARE ENGINEERING		
----------------------	--	--

Software Crisis



Ada Joint Program Office

ANSI/MIL STD 1815A FEB 83

First Translator APR 83

Software Crisis

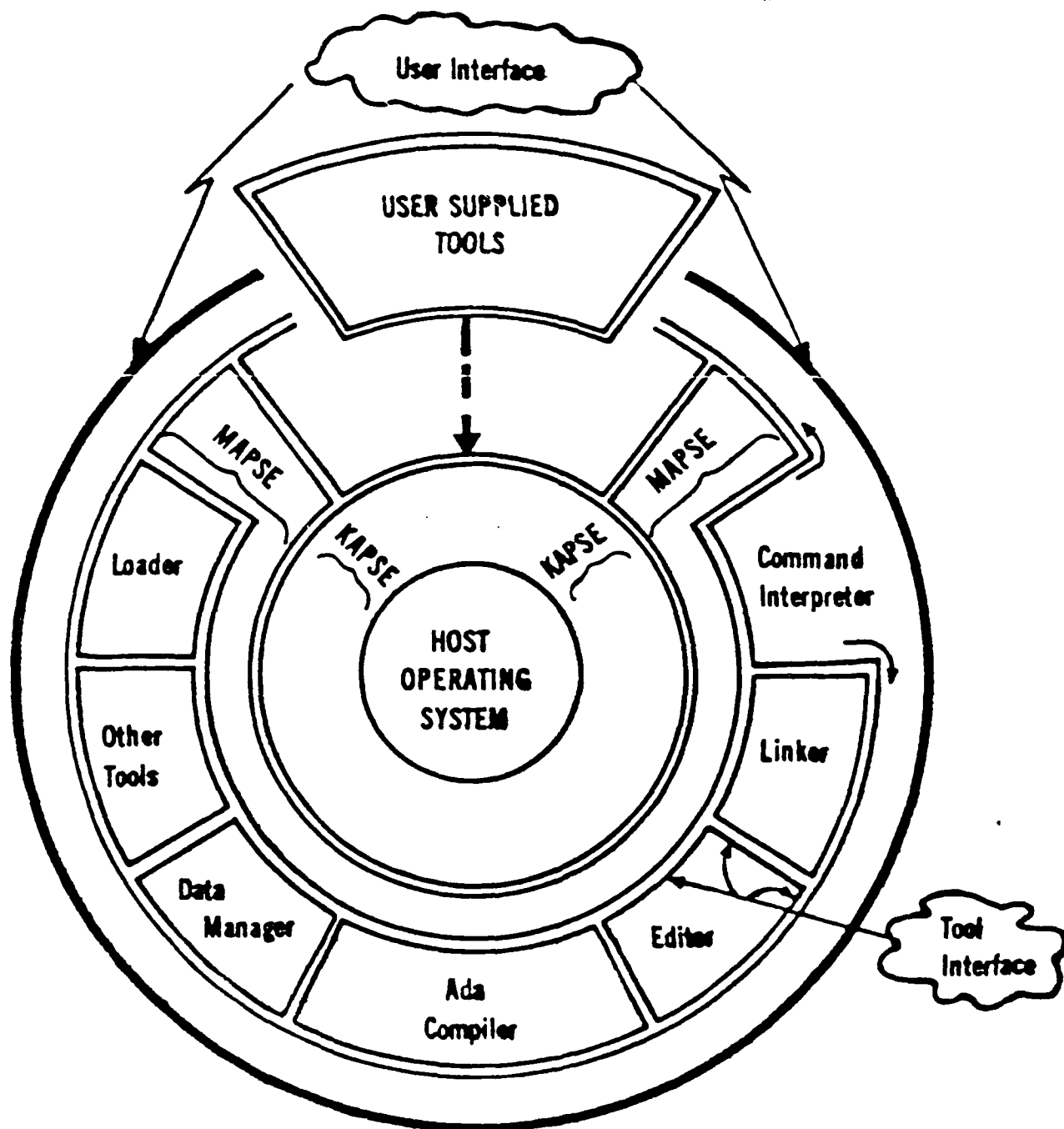
Ada Programming Support Environment

1978 SANDMAN

PEBBLEMAN

1980 STONEMAN

- Software developer productivity
- Retraining costs
- Lack of tools
- Lack of standardization



Software Crisis

"The basic problem is not our mismanagement of technology, but rather our inability to manage the complexity of our systems."

--- E.G. Booch

SOFTWARE ENGINEERING

GOALS

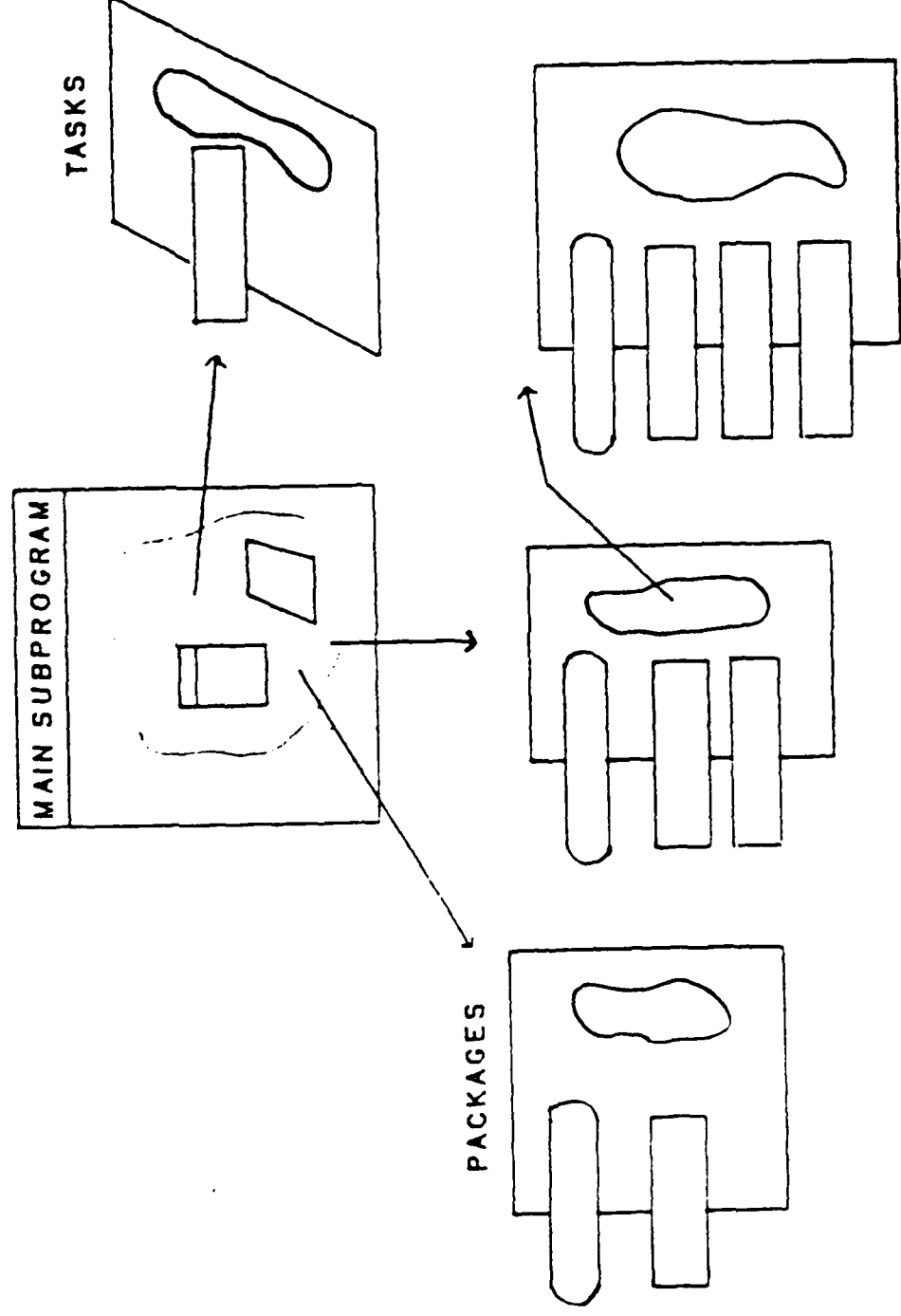
- Understandability
- Modifiability
- Reliability
- Efficiency

PRINCIPLES

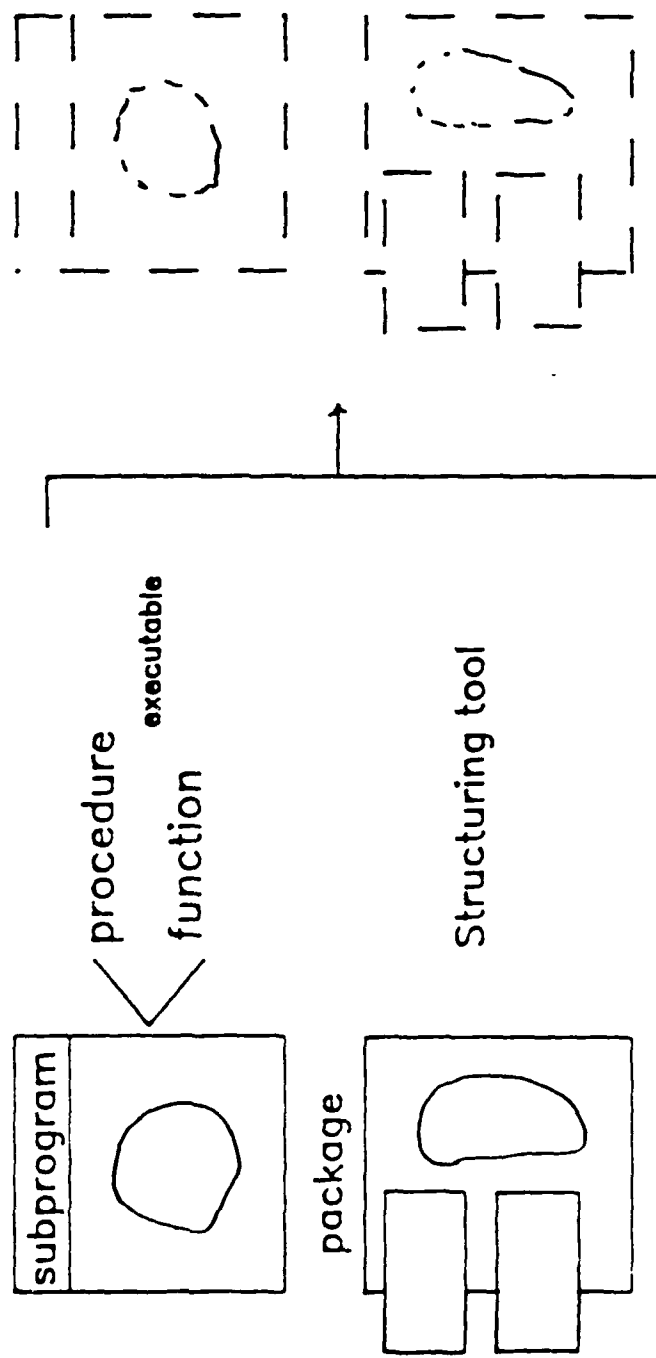
- Abstraction
- Information Hiding
- Modularity
- Localization
- Completeness
- Confirmability
- Consistency

Program Units

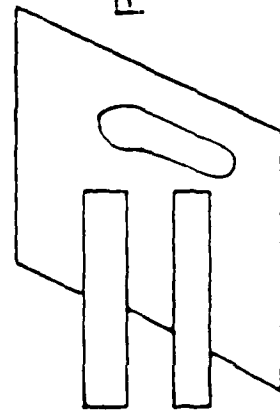
-- Ada software systems consist of one or more program units



Program Units



task

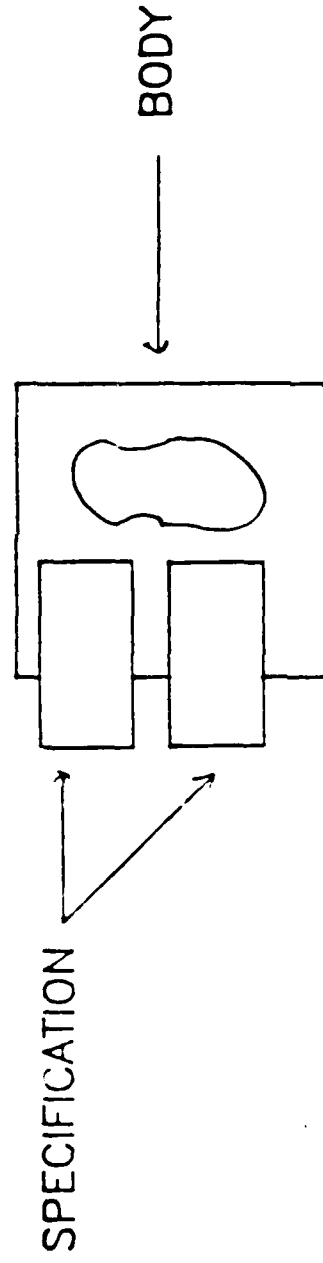


Parallel processing

Generic

Program unit
template

Program Units



"what" the program unit does $\longleftrightarrow$ ABSTRACTION $\Longrightarrow$ "how" the program unit does what it does

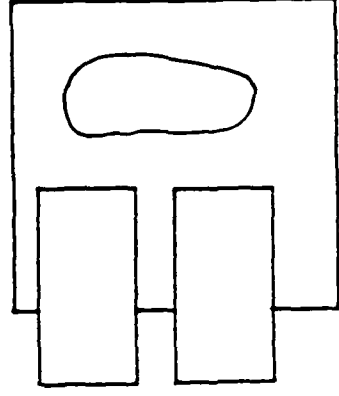
all the user of the program unit needs to know $\longleftrightarrow$ INFORMATION HIDING $\Longrightarrow$ the details of implementation are inaccessible to the user

Program Units

By separating the "what" from the "how"...



we decrease the complexity of the system...

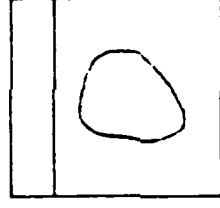


and increase: UNDERSTANDABILITY
MODIFIABILITY

Program Units

Subprograms

- Executable routines
- Main program
- Recursive



PROCEDURE

-- Defines an action to be performed

```
procedure GET_NAME ( NAME : out STRING );
```

```
    GET_NAME ( PERSONS_NAME );
```

FUNCTION

-- Returns a value

```
function SIN ( ANGLE : in RADIANS ) return FLOAT;
```

```
    ANGLE_SIN := SIN ( 2 );
```

Program Units

Procedures

SPECIFICATION

- Defines name
- Defines parameters to be passed

```
procedure ADD ( FIRST : in INTEGER;  
                SECOND : in INTEGER;  
                RESULT : out INTEGER );
```

FIRST	:	in	INTEGER
formal parameter name		parameter mode	parameter type

Program Units

Parameter modes

in — The value passed to the subprogram acts as a constant inside and may only be read. Value remains unchanged after completion.

in out — The variable passed to the procedure may be read and updated. Value may change after completion.

out — The variable passed to the procedure may only be updated. Value may change after completion

Program Units

procedures

BODY

- Defines the action to be performed
- Contains a local declarative part
- Contains a sequence of statements

```
procedure ADD ( FIRST : in INTEGER;  
               SECOND : in INTEGER;  
               RESULT : out INTEGER ) is  
    -- local declarations go here  
begin  
    RESULT := FIRST + SECOND;  
end ADD;
```

with ADD;
procedure SIMPLE_MATH is

VALUE_1, VALUE_2, VALUE_3 : INTEGER := 5;

begin

ADD (VALUE_1, 5, VALUE_2);
ADD (10, 20, VALUE_3);
ADD (VALUE_1, VALUE_2, VALUE_3);

end SIMPLE_MATH;

with TEXT_IO;
procedure SAY_HI is

```
    MAX_NAME_LENGTH : constant := 80;  
    subtype NAME_TYPE is STRING(1..MAX_NAME_LENGTH);  
    YOUR_NAME : NAME_TYPE := (others => ' ');  
    NAME_LENGTH : NATURAL := 0;
```

begin

```
    TEXT_IO.PUT_LINE("What is your name? ");  
    TEXT_IO.GET_LINE( YOUR_NAME, NAME_LENGTH );  
    TEXT_IO.PUT( "Hi " );  
    TEXT_IO.PUT_LINE( YOUR_NAME(1..NAME_LENGTH) );  
    TEXT_IO.PUT_LINE( "Have a nice day!!");
```

end SAY_HI;

Program Units

procedure AN_EXAMPLE is

MY_INTEGER : INTEGER := 10;

TEMP : INTEGER := 0;

procedure NEXT (AN_INTEGER : in INTEGER;
VALUE : out INTEGER) is

begin

VALUE := AN_INTEGER + 1;

end NEXT;

begin

while MY_INTEGER <= 100 loop

NEXT(MY_INTEGER,TEMP);

MY_INTEGER := TEMP;

end loop;

end AN_EXAMPLE;

Program Units

Functions

SPECIFICATION

- Defines name
- Defines parameters to be passed
- Defines result type

```
function ADD ( FIRST, SECOND : in INTEGER )  
    return INTEGER;
```

- parameter mode can only be "in"
- called as an expression

Program Units

Functions

BODY

- Defines the action to be performed
- Contains a declarative part
- Contains a sequence of statements
- Result returned in a "return" statement

```
function ADD ( FIRST, SECOND : INTEGER )
```

```
    return INTEGER is
```

```
begin
```

```
    return FIRST + SECOND;
```

```
end ADD;
```

Program Units

Functions

procedure CALCULATIONS is

VALUE : INTEGER := 1;

function ADD_PREVIOUS (NUMBER : in INTEGER)
return INTEGER is

begin

return NUMBER + (NUMBER - 1);

end ADD_PREVIOUS;

begin

VALUE := ADD_PREVIOUS (5);

-- value equals 9

end CALCULATIONS;

procedure ADD_THEN is

type INDEX_TYPE is range 1 .. 3;

type REAL is digits 9;

type MATRIX_TYPE is array(INDEX_TYPE, INDEX_TYPE)
of REAL;

function "+" (LEFT, RIGHT : in MATRIX_TYPE)
return MATRIX_TYPE is separate;

FIRST, SECOND,

RESULT : MATRIX_TYPE := (others => 0.0);

begin

RESULT := FIRST + SECOND;

end ADD_THEN;

```

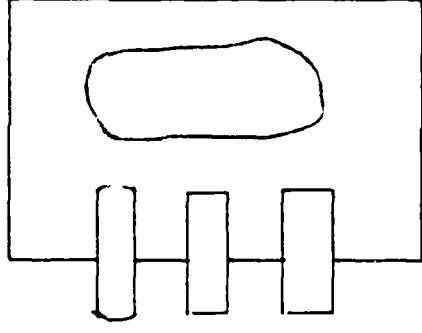
separate ( ADD_I THEM )
function "+" ( LEFT, RIGHT : in MATRIX_TYPE ) return
    MATRIX_TYPE is
    TEMP_MATRIX : MATRIX_TYPE := ( others => 0.0 );

begin
    for FIRST_INDEX in MATRIX_TYPE'RANGE(1) loop
        for SECOND_INDEX in MATRIX_TYPE'RANGE(2) loop
            TEMP_MATRIX(FIRST_INDEX, SECOND_INDEX ) :=
                LEFT(FIRST_INDEX,SECOND_INDEX) +
                RIGHT(FIRST_INDEX,SECOND_INDEX);
        end loop;
    end loop;
    return TEMP_MATRIX;
end "+";

```

Program Units

Packages



- Defines groups of logically related items
- Structuring tool
- Contains a visible part (specification) and a hidden part (private part and body)
- Primary means for extending the language

Program Units

Package specification

-- Define items available to user of package (export)

package CONSTANTS is

PI : constant := 3.14159;

e : constant := 2.71828;

WARP : constant := 3.00E+08;

-- meters/second

end CONSTANTS;

with CONSTANTS;
procedure SOME_PROGRAM is

 MY_VALUE : FLOAT := 2 * CONSTANTS.PI;
begin
 null;
end SOME_PROGRAM;

with CONSTANTS;
procedure ANOTHER_PROGRAM is

 ANOTHER_VALUE : FLOAT := 2 * CONSTANTS.PI;
begin
 null;
end ANOTHER_PROGRAM;

Program Units

```
package ROBOT_CONTROL is
    type SPEED is range 0..100;
    type DISTANCE is range 0..500;
    type DEGREES is range 0..359;
    procedure GO_FORWARD ( HOW_FAST : in SPEED;
                           HOW_FAR : in DISTANCE );
    procedure REVERSE ( HOW_FAST : in SPEED;
                       HOW_FAR : in DISTANCE );
    procedure TURN ( HOW_MUCH : in DEGREES );
end ROBOT_CONTROL;
```

with ROBOT_CONTROL;

procedure DO_A_SQUARE is
begin

ROBOT_CONTROL.GO_FORWARD(HOW_FAST => 100,
HOW_FAR => 20);

ROBOT_CONTROL.TURN(90);

ROBOT_CONTROL.GO_FORWARD(100, 20);

ROBOT_CONTROL.TURN(90);

ROBOT_CONTROL.GO_FORWARD(100, 20);

ROBOT_CONTROL.TURN(90);

ROBOT_CONTROL.GO_FORWARD(100, 20);

ROBOT_CONTROL.TURN (90);

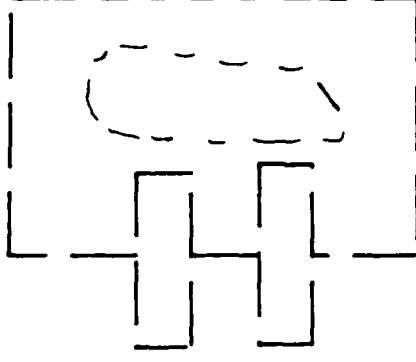
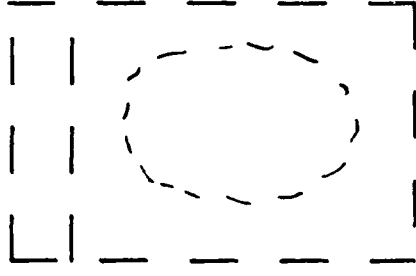
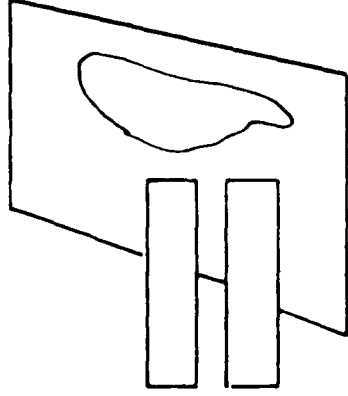
end DO_A_SQUARE;

Program Units

Package bodies

```
--Define local declarations
--Define implementation of subprograms
--- defined in specification
package body ROBOT_CONTROL is
    --local declarations
    procedure RESET_SYSTEM is
begin
    --implementation
    end RESET_SYSTEM;
    procedure GO_FORWARD...is...
    procedure REVERSE...is...
    procedure TURN...is...
end ROBOT_CONTROL;
```

Program Units



TASK

GENERIC

A program unit that operates in parallel with other program units

Template of a subprogram or package

Types

--A type consists of a set of values that
objects of the type may take on, and a
set of operations applicable to those values

--Ada is a strongly typed language!

- *Every object must be declared of some type name
- *Different type names may not be implicitly mixed
- *Operations on a type must preserve the type

```
AN_INTEGER : INTEGER;  
A_FLOAT_NUMBER : FLOAT;  
ANOTHER_FLOAT : FLOAT;
```

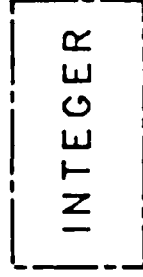
```
A_FLOAT_NUMBER := ANOTHER_FLOAT + AN_INTEGER;  
--illegal
```

Types

Types and Objects

TYPES

Define a template
for objects



OBJECTS

Variables or constants
that are instances
of a type

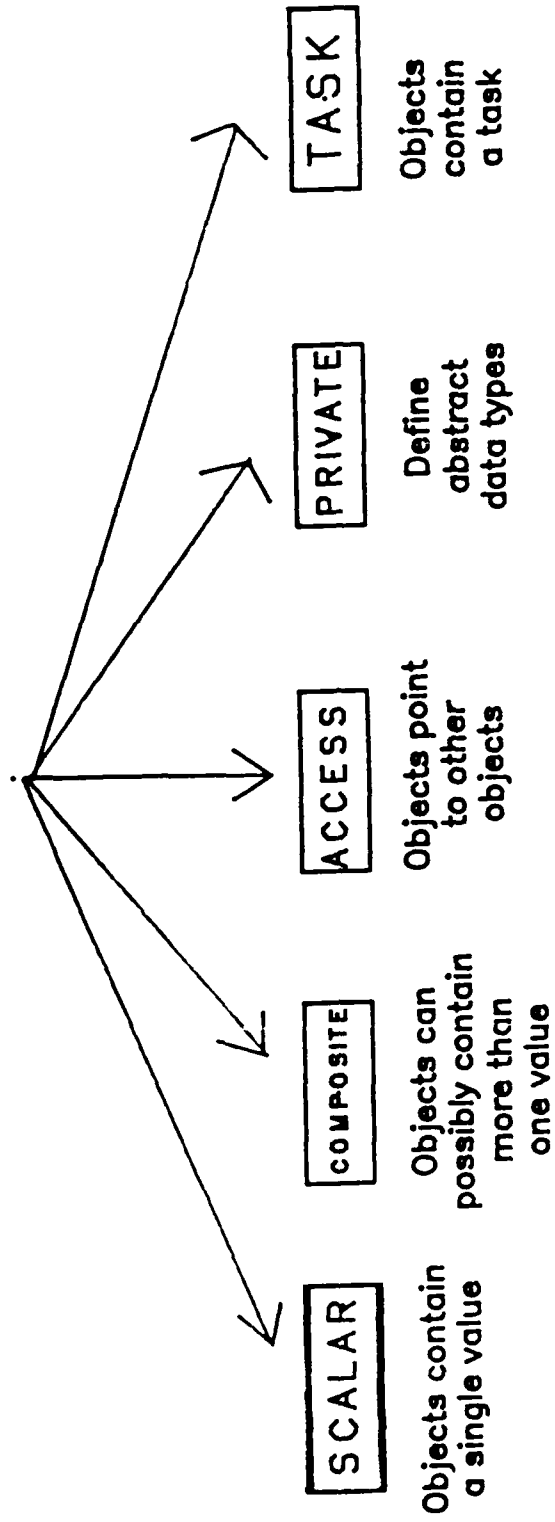


OBJECT DECLARATION

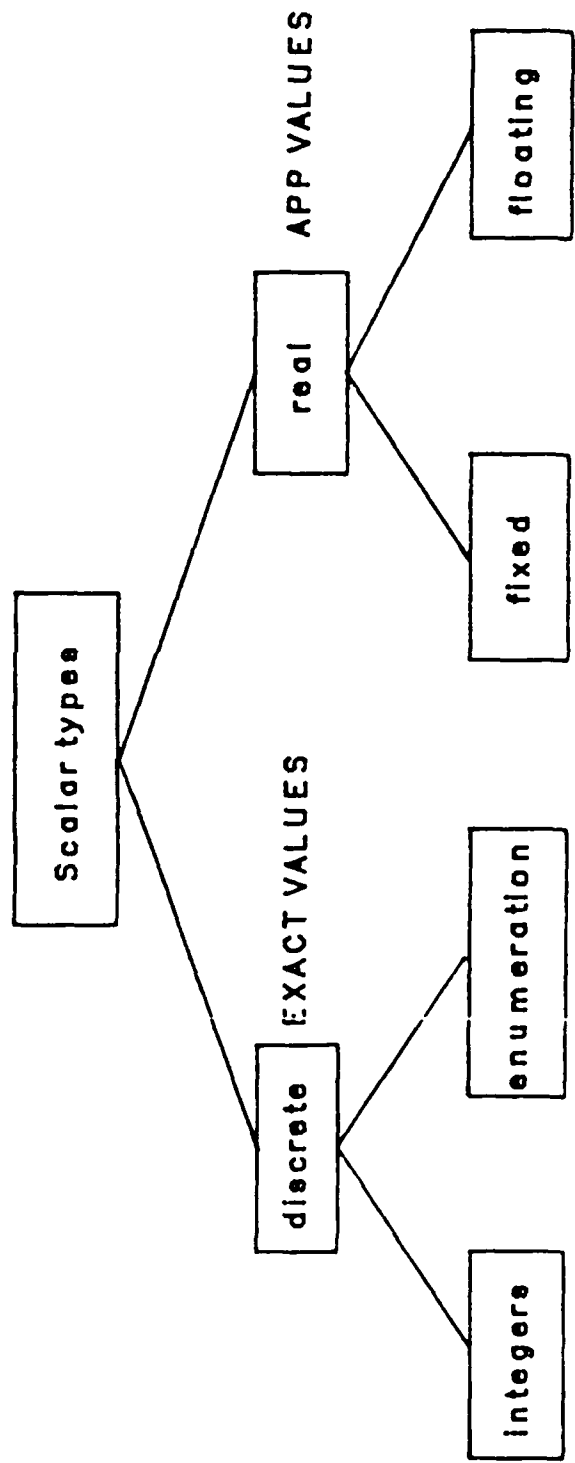
```
MY_INTEGER : INTEGER;
```

```
YOUR_INTEGER : INTEGER := 10;
```

Ada Types



Types



← USER DEFINED →

← PREDEFINED →

Types

Integers

--Define a set of exact, consecutive values

USER DEFINED

type ALTITUDE is range 0..100_000;

type DEPTH is range 0..20_000;

PLANES_HEIGHT : ALTITUDE;

DIVER_DEPTH : DEPTH;

begin

PLANES_HEIGHT := 10_000;

PLANES_HEIGHT := 200_000; -- error

PLANES_HEIGHT := DIVER_DEPTH; -- error

end;

Types

Predefined integer types

INTEGER----->(usually -32,768..32767)

"subtypes" of INTEGER

NATURAL(0..INTEGER'LAST)

POSITIVE(1..INTEGER'LAST)

LONG_INTEGER----->(usually double word)

SHORT_INTEGER----->(usually half word)

Types

Subtypes

- Constrain a range of values or accuracy on a type
- Does not define a new type ,i.e., compatible with base type

```
type ALTITUDE is range 0..200_000;  
subtype HIGH is ALTITUDE range 40_000 .. 200_000;  
subtype MEDIUM is ALTITUDE range 10_000 .. 100_000;  
subtype LOW is ALTITUDE range 0 .. 10_000;
```

Types

Enumeration

- Define a set of ordered enumeration values
- Used in array indexing, case statements,
- and looping

USER DEFINED

```
type SUIT is (CLUBS, HEARTS, DIAMONDS, SPADES);  
type COLOR is (RED, WHITE, BLUE);  
type SWITCH is (OFF, ON);  
type EVEN DIGITS is ('2','4','6','8');  
type MIXED is (ONE,'2',THREE,'*',!,'more');
```

where CLUBS < HEARTS < DIAMONDS < SPADES

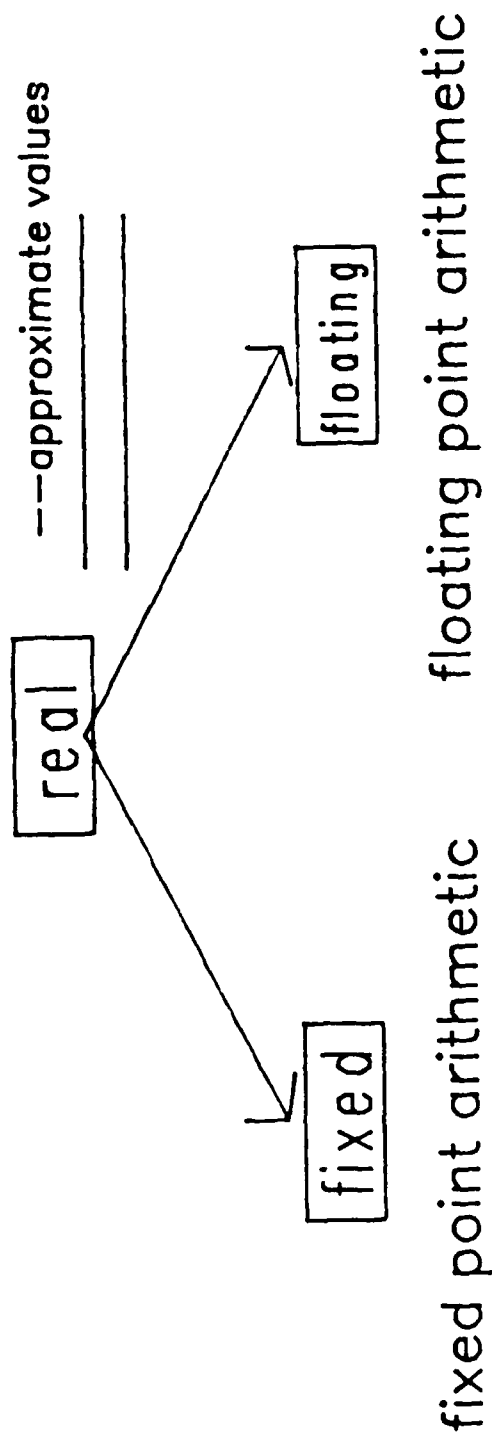
Types

Pre-defined enumeration types

BOOLEAN -----> (FALSE, TRUE)

CHARACTER

Types



Types

Fixed point types

- Absolute bound on error
- Larger error for smaller numbers (around zero)

USER DEFINED

type INCREMENT is delta 1.0/8 range 0.0 .. 1.0;

0, 1*2e-3, 2*2e-3, 4*2e-3, 5*2e-3,...

PREDEFINED

DURATION ---> (Used for "delay" statements)

Types

Floating point types

- Relative bound of error
- Defined in terms of significant digits
- More accurate at smaller numbers, less at larger

USER DEFINED

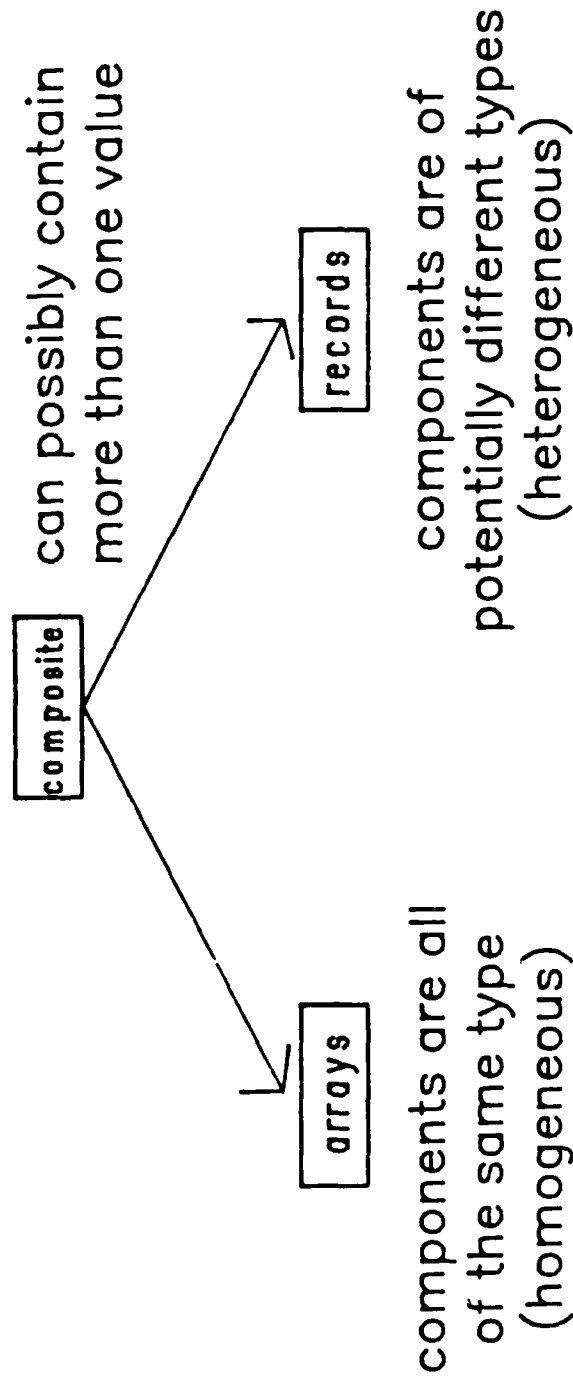
type NUMBERS is digits 3 range 0.0 .. 20_000;

0.001, 0.002, 0.003...9999.0, 1000.0, 1001.0..., 10000.0, 10100.0

PREDEFINED

FLOAT

Types



Types

Arrays

constrained

unconstrained

CONSTRAINED

```
-- Indices are static for all objects of that type  
  
type HOURS is range 0..40;  
type DAYS is ( SUN,MON,TUE,WED,THU,FRI,SAT );  
type WORK_HOURS is array( DAYS ) of HOURS;  
  
MY_HOURS : WORK_HOURS := ( 0,8,8,7,6,1,0 );
```

MY_HOURS(SUN)	MY_HOURS(MON)	MY_HOURS(TUE)	MY_HOURS(SAT)
0	8	8	0

Types

Arrays

UNCONSTRAINED

- Indices are known at elaboration (run) time
- Indices may be different for different objects

```
type HOURS is range 0..40;
```

```
type DAYS is (SUN, MON, TUE, WED, THU, FRI, SAT);
```

```
type WORK_HOURS is array (DAYS range <>) of HOURS;
```

```
HOLIDAY_WEEK : WORK_HOURS (TUE..SAT) :=(others =>0);
```

```
FULL_WEEK : WORK_HOURS (DAYS'FIRST..DAYS'LAST);
```

Types

procedure DAYS_WORKED (FIRST,SECOND: in DAYS) is

 A_WEEK : WORK_HOURS (FIRST..SECOND);

begin

 .

 DAYS_WORKED(WED,FRI);



 DAYS_WORKED(FRI,SAT);



Types

Multi--dimensional arrays

```
type VALUES is digits 6 range -10.0 .. 100.0;
type INDEX is range 1..3;
type TWO_D_MATRIX is array (INDEX,INDEX) of VALUES;

MY_MATRIX : TWO_D_MATRIX := ( others => 0.0 );
IDENTITY_MATRIX : constant TWO_D_MATRIX := ( (1.0,0.0,0.0),
                                              (0.0,1.0,0.0),
                                              (0.0,0.0,1.0));
```

begin

```
MY_MATRIX := IDENTITY_MATRIX;
MY_MATRIX (3,3) := 2.0;
```

.
. .
.

Types

Array

PREDEFINED

type STRING is array (POSITIVE range <>) of CHARACTER;

USE OF THE PREDEFINED STRING TYPE

YOUR_STRING : STRING (1..10);

MY_STRING : STRING (1..20);

THEIR_STRING : STRING; -- illegal

STRING SLICING

YOUR_STRING := MY_STRING(1..10);

MY_STRING(11..15) := YOUR_STRING(2..6);

MY_STRING(3..4) := MY_STRING(4..5);

MY_STRING(2) := 'G';

MY_STRING(2) := "G"; -- illegal

Types

Records

undiscriminated
discriminated
variant

UNDISCRIMINATED

type DAYS is (MON,TUE,WED,THU,FRI,SAT,SUN);
type DAY is range 1..31;
type MONTH is (JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,
SEP,OCT,NOV,DEC);

type YEAR is range 0..2085;

type DATE is record

DAY_OF_WEEK : DAYS;

DAY_NUMBER : DAY;

MONTH_NAME : MONTH;

YEAR_NUMBER : YEAR;

end record;

TODAY : DATE;

begin

TODAY.DAY_OF_WEEK := TUE;

TODAY.DAY_NUMBER := 26;

TODAY.MONTH_NAME := NOV;

TODAY

DAY_OF_WEEK	TUE
DAY_NUMBER	26
MONTH_NAME	NOV
YEAR_NUMBER	1985

Types

Records

```
type A_MONTH is array (DAY range <>) of DATE;  
NOVEMBER: A_MONTH(1..30);
```

```
begin
```

```
    NOVEMBER(26).DAY_OF_WEEK := TUE;  
    NOVEMBER(27) := (WED,27,NOV,1985);
```

Types

Records

DISCRIMINATED

```
type BUFFER(SIZE:POSITIVE := 10) is record
    ITEMS : STRING(1..SIZE);
end record;
```

```
MY_BUFFER : BUFFER; -- size is 10;
YOUR_BUFFER : BUFFER (20);
THEIR_BUFFER : BUFFER (SIZE => 15);
```

```
begin
    MY_BUFFER.ITEMS := "Hi There!!";
```

Types

Records

VARIANT

```
type DRIVER is (GOOD,BAD);
type INSURANCE_RATE is range 1..50;
type DISCOUNT is delta 0.01 range 0.0..1.0;
type INSURANCE (KIND:DRIVER) is record
    NORMAL_RATE : INSURANCE_RATE;
    case KIND is
        when GOOD => DISCOUNT_RATE : DISCOUNT ;
        when BAD => ADDITIONAL : INSURANCE_RATE;
    end case;
end record;
```

Types

Records

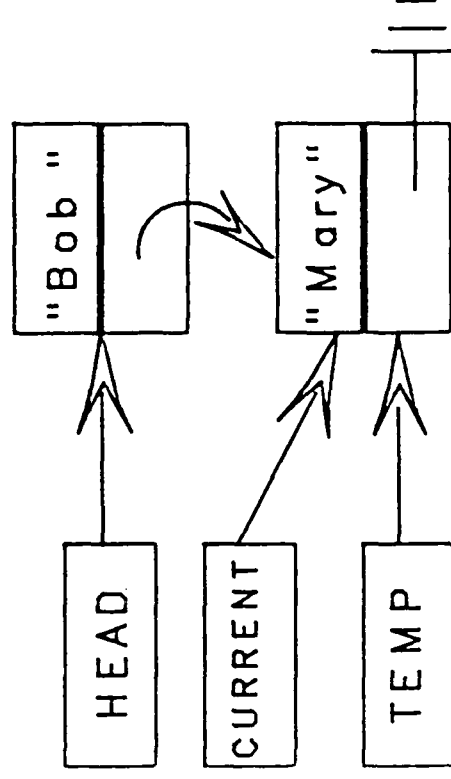
VARIANT

```
type DRIVER is (GOOD,BAD);
type INSURANCE_RATE is range 1..50;
type DISCOUNT is delta 0.01 range 0.0..1.0;
type INSURANCE (KIND:DRIVER) is record
    NORMAL_RATE : INSURANCE_RATE;
    case KIND is
        when GOOD => DISCOUNT_RATE : DISCOUNT ;
        when BAD => ADDITIONAL : INSURANCE_RATE;
    end case;
end record;
```

Types

Access types— Linked list

--Move current pointer
CURRENT := TEMP;

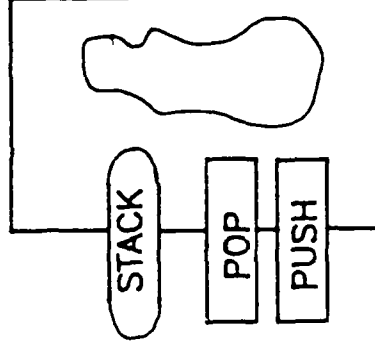


Types

Private types

- Defined in a package
- Used to create abstract data types
- Used to extend the language
- Directly supports abstraction and
- Information hiding

INTEGER_STACK



PRIVATE

$:=$ $=$ $/=$

subprograms defined in
package specification

LIMITED PRIVATE

only subprograms
defined in
package specification

Types

Access types – Linked list

procedure LINKED LIST is

- type ITEM; — incomplete type declaration

```
type POINTER is access ITEM;
```

type ITEM is record

NAME: STRING(1..20):=(others => '');

NEXT : POINTER;

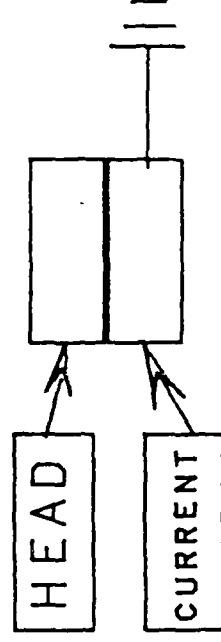
end record;

HEAD,CURRENT,TEMP:POINTER; --initialized to null

begin

```
HEAD:=new ITEM;
```

CURRENT:=HEAD;



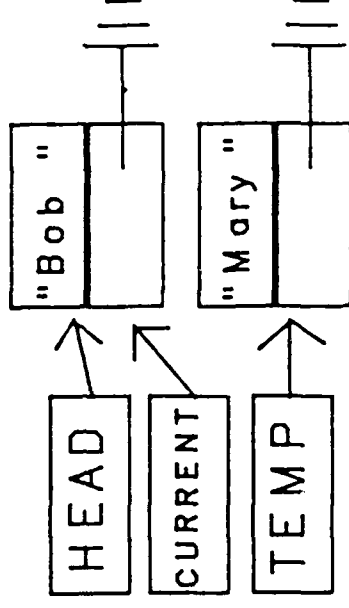
```
CURRENT.NAME(1..3):= "Bob";
```

Types

Access types – Linked list

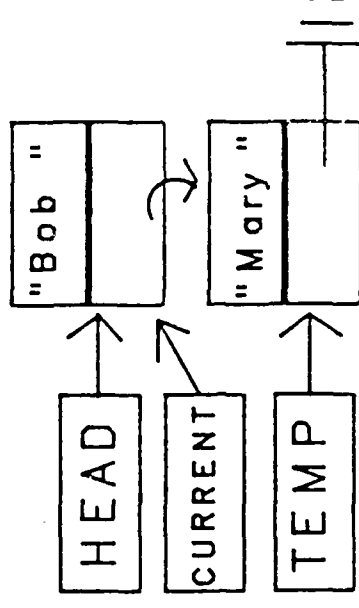
Create a New Item

```
TEMP := new ITEM;  
TEMP.NAME(1..4) := "MARY";
```



Add to List

```
CURRENT.NEXT := TEMP;
```



package BASKIN_ROBBINS is

type NUMBERS is range 0 .. 99;

procedure TAKE(A_NUMBER : out NUMBERS);

procedure NOW_SERVING return NUMBERS;

procedure SERVE(A_NUMBER : in NUMBERS);

end BASKIN_ROBBINS;

with BASKIN_ROBBINS;

procedure GET_ICE_CREAM is

YOUR_NUMBER : BASKIN_ROBBINS.NUMBERS;

begin

BASKIN_ROBBINS.TAKE(YOUR_NUMBER);

loop

if BASKIN_ROBBINS."="(BASKIN_ROBBINS.NOW_SERVING,
YOUR_NUMBER);

BASKIN_ROBBINS.SERVE(YOUR_NUMBER);

exit;

end if;

end loop;

end GET_ICE_CREAM;

with BASKIN_ROBBINS; use BASKIN_ROBBINS;
procedure GET_ICE_CREAM is

YOUR_NUMBER : BASKIN_ROBBINS.NUMBERS;

begin

BASKIN_ROBBINS.TAKE(YOUR_NUMBER);

loop

if BASKIN_ROBBINS.NOW_SERVING = YOUR_NUMBER then
 BASKIN_ROBBINS.SERVE(YOUR_NUMBER);
 exit;

else

YOUR_NUMBER := YOUR_NUMBER - 1;

end if;

end loop;

end GET_ICE_CREAM;

package BASKIN_ROBBINS is

type NUMBERS is private;

procedure TAKE(A_NUMBER : out NUMBERS);

procedure NOW_SERVING return NUMBERS;

procedure SERVE(A_NUMBER : in NUMBERS);

private

type NUMBERS is range 0 .. 99;

end BASKIN_ROBBINS;

with BASKIN_ROBBINS; use BASKIN_ROBBINS;
procedure GET_ICE_CREAM is

YOUR_NUMBER : BASKIN_ROBBINS.NUMBERS;

begin

BASKIN_ROBBINS.TAKE(YOUR_NUMBER);

loop

if BASKIN_ROBBINS.NOW_SERVING = YOUR_NUMBER then
 BASKIN_ROBBINS.SERVE(YOUR_NUMBER);
 exit;

else

YOUR_NUMBER := BASKIN_ROBBINS.NOW_SERVING;

end if;

end loop;

end GET_ICE_CREAM;

```
package BASKIN_ROBBINS is

    type NUMBERS is limited private;

    procedure TAKE( A_NUMBER : out NUMBERS );

    procedure NOW_SERVING return NUMBERS;

    procedure SERVE( A_NUMBER : in NUMBERS );

    function "="( LEFT, RIGHT : NUMBERS) return BOOLEAN;

private

    type NUMBERS is range 0 .. 99;

end BASKIN_ROBBINS;
```

```
with BASKIN_ROBBINS; use BASKIN_ROBBINS;  
procedure GET_ICE_CREAM is  
  YOUR_NUMBER : BASKIN_ROBBINS.NUMBERS;  
  procedure GO_TO_DAIRY_QUEEN is separate;  
begin  
  BASKIN_ROBBINS.TAKE( YOUR_NUMBER );  
  loop  
    if BASKIN_ROBBINS.NOW_SERVING = YOUR_NUMBER then  
      BASKIN_ROBBINS.SERVE( YOUR_NUMBER );  
      exit;  
    else  
      GO_TO_DAIRY_QUEEN;  
      exit;  
    end if;  
  end loop;  
end GET_ICE_CREAM;
```

Types

Private types

```
package INTEGER_STACK is
  type STACK is limited private;
  procedure POP ( ITEM : out INTEGER;
                 OFF_OF:in out STACK);
  procedure PUSH (ITEM: in INTEGER;
                 ON: in out STACK);
private
  --Define what a stack looks like
end INTEGER_STACK;
```

Types

Private types

```
with INTEGER_STACK;  
use INTEGER_STACK;  
procedure STACK_THEM is  
    MY_STACK, YOUR_STACK: STACK;  
    AN_ITEM: INTEGER  
begin  
    PUSH (ITEM=>20, ON=>MY_STACK);  
    PUSH (ITEM=>30, ON=>YOUR_STACK);  
    PUSH (40, ON=>MY_STACK);  
    .  
    .  
    . POP (AN_ITEM, OFF_OF=>MY_STACK);  
    --AN_ITEM = 40  
end STACK_THEM;
```

Control Statements

SEQUENTIAL -----	CONDITIONAL -----	ITERATIVE -----
ASSIGNMENT	IF	LOOP
PROCEDURE CALL	CASE	
RETURN		
NULL		
BLOCK		
TASKING -----	OTHERS -----	
ENTRY CALL	GOTO	
DELAY	RAISE	
ABORT	CODE	
ACCEPT		
SELECT		

Control Statements

Sequential

ASSIGNMENT

-- Replaces variable on left with expression on right
AN_INTEGER := (5*2) + 34;

PROCEDURE CALL

-- Executes a procedure
POP (AN_INTEGER, OFF_OF => MY_STACK);

NULL

-- Explicitly does nothing
null;

Control Statements

Sequential

RETURN

-- Causes control to be passed back to the caller
of a subprogram

For a procedure...

procedure A_PROCEDURE is

 AN_INTEGER : INTEGER;

begin

 AN_INTEGER := 5;

 return;

 null; -- never gets executed

end A_PROCEDURE;

Control Statements

Sequential

RETURN

-- For a function, returns a value

```
function IS_GREATER ( FIRST, SECOND : in INTEGER )  
    return BOOLEAN;
```

begin

```
    return ( FIRST > SECOND );  
end IS_GREATER;
```

-- Every function must have at least one
return statement

Control Statements

Sequential

BLOCK

-- Used to localize declarations and/or effects

procedure MAIN_PROGRAM is

VARIABLE : FLOAT;

begin

-- some statements

declare

LOCAL_VARIABLE : FLOAT;

begin

LOCAL_VARIABLE := 4.0;

VARIABLE := 70.0;

end;

VARIABLE := 10.0;

end MAIN_PROGRAM;

Control Statements

Conditional

IF

```
if MY_VALUE = 27 then  
    HIS_VALUE := 21;  
    THEIR_VALUE := 22;  
end if;
```

```
if MACHINE_IS_RUNNING then  
    SET_NEW_SPEED ( 47 );  
else  
    COUNT_TIME_DOWN ( CURRENT_TIME );  
end if;
```

Control Statements

Conditional

IF

```
if MACHINE_IS_RUNNING then
    SET_NEW_SPEED ( 47 );
elseif MACHINE_IS_IDLE then
    START_MACHINE_UP;
else
    COUNT_TIME_DOWN ( CURRENT_TIME );
end if;
```

Control Statements

Conditional

```
type DAY_TIMES is ( EARLY_AM,MID_AM,LUNCH,AFTERNOON,  
                    LATE_AFTERNOON,DINNER,EVENING,NIGHT );
```

```
TIME : DAY_TIMES := AFTERNOON;
```

```
begin
```

```
  if TIME = EARLY_AM then
```

```
    DRINK_COFFEE;
```

```
  elsif TIME = MID_AM then
```

```
    DRINK_COFFEE;
```

```
  elsif TIME = LUNCH then
```

```
    GO_EAT;
```

```
  elsif TIME = AFTERNOON then
```

```
    STAY_AWAKE;
```

```
  elsif TIME = LATE_AFTERNOON then
```

```
    GET_READY_TO_GO_HOME;
```

```
  else
```

```
    GET_READY_FOR_TOMMORROW;
```

```
  end if;
```

```
end;
```

Control Statements

Conditional

CASE

```
case TIME is
  when EARLY_AM | MID_AM => DRINK_COFFEE;
  when LUNCH => GO_EAT;
  when AFTERNOON => STAY_AWAKE;
  when LATE_AFTERNOON => GET_READY_TO_GO_HOME;
  when others => GET_READY_FOR_TOMMORROW;

end case;
```

Control Statements

Iterative

BASIC LOOP

```
loop
  -- statements
end loop;
```

EXIT STATEMENT

```
loop
  if X = 20 then
    exit;
  end if;
end loop;
```

```
loop
  if X = 20 then
    exit;
  end if;
end loop;
end loop;
end loop;
```

Control Statements

Iterative

OUTER:
loop

INNER:
loop
if $X = 20$ then
 exit OUTER;
end if;
exit INNER when $X = 21$;
 $X := X + 2$;
end loop INNER;
end loop OUTER;

Control Statements

Iterative

FOR LOOP ITERATION SCHEME

```
with TEXT_IO; use TEXT_IO;
procedure PRINT_ALL_VALUES is
  type COLORS is ( RED, WHITE, BLUE );
  package COLOR_IO is new ENUMERATION_IO ( COLORS );
  use COLOR_IO;

begin
  for INDEX in 1..5 loop
    null;
  end loop;

  for A_COLOR in COLORS loop
    PUT ( A_COLOR );
    NEW_LINE;
  end loop;
end PRINT_ALL_VALUES;
```

Control Statements

Iterative

```
for MY_INDEX in 20..40 loop
  -- some statements
end loop;
```

```
for YOUR_INDEX in reverse 20..40 loop
  -- some statements
end loop;
```

Control Statements

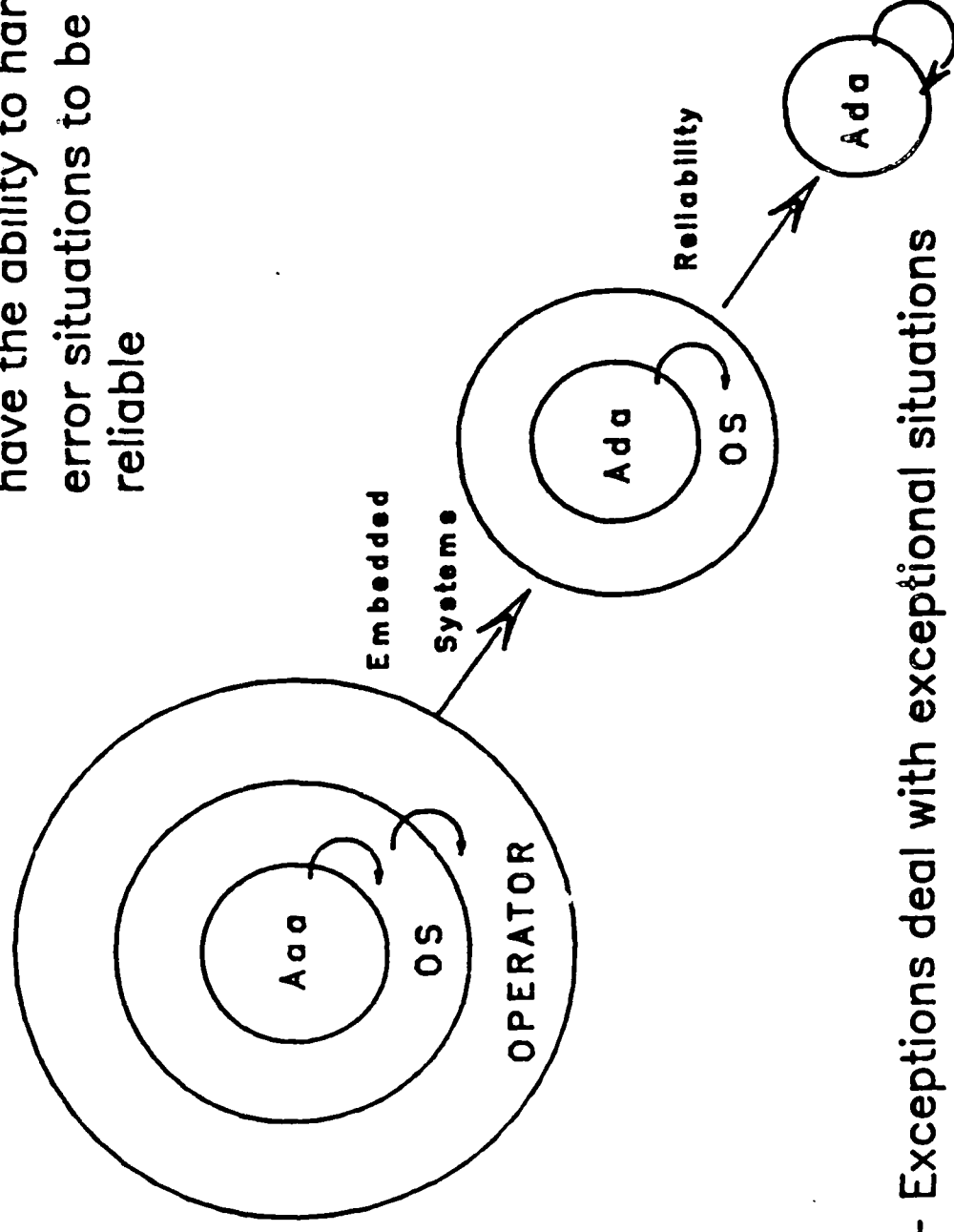
Iterative

WHILE LOOP ITERATION SCHEME

```
while NOT_DARK loop
  PLAY_TENNIS;
end loop;
TURN_ON_LIGHTS;
```

Exceptions

- Real time systems must have the ability to handle error situations to be reliable



- Exceptions deal with exceptional situations

Exceptions

```
with TEXTJO; use TEXTJO;
procedure GET_NUMBERS is
type NUMBERS is range 1..100;
package NUMJO is new INTEGERJO ( NUMBERS );
use NUMJO;
A_NUMBER : NUMBERS;
begin
loop
GET ( A_NUMBER );
NEW_LINE;
PUT("The number is ");
PUT ( A_NUMBER );
NEW_LINE;
end loop;
exception
when DATA_ERROR => PUT_LINE("That was a bad number");
end GET_NUMBERS;
```

Exceptions

- When an exception situation occurs, the exception is said to be "raised"
- What happens then, depends on the presence or absence of an exception handler

```
begin
  loop
    GET ( A_NUMBER );
    NEW_LINE;
    PUT("The number is");
    PUT ( A_NUMBER );
    NEW_LINE;
  end loop;
end GET_NUMBERS;
```

Exceptions

```
begin
loop
begin
  GET ( A_NUMBER );
  NEW_LINE;
  PUT ( "The number is " );
  PUT ( A_NUMBER );
  NEW_LINE;
exception
  when DATA_ERROR => PUT_LINE("Bad number, try again");
end;
end loop;
end GET_NUMBERS;
```

Exceptions

USER DEFINED

PREDEFINED

STACK_OVERFLOW : exception;

BAD_INPUT : exception;

DEAD_SENSOR : exception;

CONSTRAINT_ERROR

NUMERIC_ERROR

PROGRAM_ERROR

STORAGE_ERROR

TASKING_ERROR

I/O EXCEPTIONS

STATUS_ERROR

MODE_ERROR

NAME_ERROR

USE_ERROR

DEVICE_ERROR

END_ERROR

DATA_ERROR

package SIMPLE_STACK is

type STACK_TYPE is limited private;
subtype ELEMENT_TYPE is CHARACTER;

procedure PUSH (AVALUE : in ELEMENT_TYPE;
 STACK : in out STACK_TYPE);

procedure POP (AVALUE : out ELEMENT_TYPE;
 STACK : in out STACK_TYPE);

STACK_OVERFLOW, STACK_UNDERFLOW : exception;

private

type STACK_ITEM;
type STACK_TYPE is access STACK_ITEM;
type STACK_ITEM is record
 VALUE : ELEMENT_TYPE;
 NEXT : STACK_TYPE;
end record;

end SIMPLE_STACK;

NO-A190 365

TUTORIAL TRACK I INTRODUCTION TO ADA(U) INFORMATION
SYSTEMS AND TECHNOLOGY CENTER W-P AFB OH ADA VALIDATION
FACILITY C ENGLE ET AL 09 JUN 87

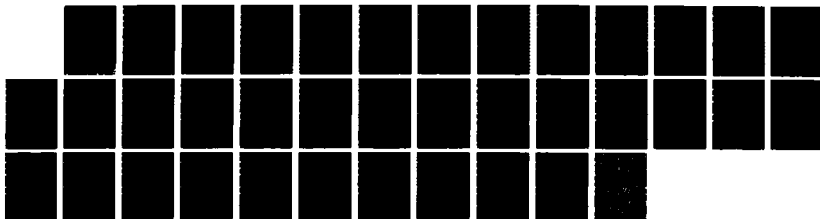
2/2

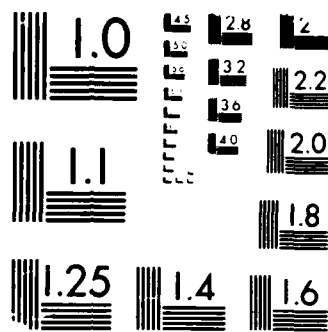
UNCLASSIFIED

F/G 12/5

NL

4





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

separate (SIMPLE_STACK)
procedure POP (A_VALUE : out ELEMENT_TYPE;
 A_STACK : in out STACK_TYPE) is

begin

 A_VALUE := A_STACK.VALUE;
 A_STACK := A_STACK.NEXT;

exception

 when CONSTRAINT_ERROR =>
 raise STACK_UNDERFLOW;

end POP;

separate (SIMPLE_STACK)
procedure PUSH (A_VALUE : in ELEMENT_TYPE;
 A_STACK : in out STACK_TYPE) is

 TEMP_ITEM : STACK_TYPE;

begin

 TEMP_ITEM := new STACK_TYPE;
 TEMP_ITEM.NEXT := A_STACK;
 TEMP_ITEM.VALUE := A_VALUE;
 A_STACK := TEMP_ITEM;

exception

 when STORAGE_ERROR =>
 raise STACK_OVERFLOW;

end PUSH;

with TEXT_IO, SIMPLE_STACK;
procedure STACK_USER is

package COUNT_IO is new TEXT_IO.INTEGER_IO(LONG_INTEGER);

MY_STACK : SIMPLE_STACK.STACK_TYPE;
COUNTER : LONG_INTEGER := 0;

begin

loop

SIMPLE_STACK.PUSH('a', MY_STACK);
COUNTER := COUNTER + 1;

end loop;

exception

when SIMPLE_STACK.STACK_OVERFLOW =>
TEXT_IO.PUT("Pushed");
COUNT_IO.PUT(COUNTER);
TEXT_IO.PUT_LINE(" times");

end STACK_USER;

Generics

Parameterized Program Unit
subprograms
packages

Cannot be called

Must be instantiated

Generics

Data Objects

- To define the template: use type declaration
- To define an instance: use object declaration

Generic program units

- To define the template: use generic declaration
- To define an instance: use generic instantiation

Generics

Generics Provide:

- factorization
- reduction in size of program text
- more compact code
- no unnecessary duplication of source
- maintainability
- readability
- efficiency

Generics

procedure INTEGER_SWAP (FIRST_INTEGER, SECOND_INTEGER:
in out INTEGER) is

TEMP : INTEGER;

begin

```
TEMP      := FIRST_INTEGER;  
FIRST_INTEGER := SECOND_INTEGER;  
SECOND_INTEGER := TEMP;
```

end INTEGER_SWAP;

Generics

generic

type ELEMENT is private;

procedure SWAP (ITEM_1,ITEM_2:in out ELEMENT);

procedure SWAP(ITEM_1,ITEM_2:in out ELEMENT) is

TEMP:ELEMENT;

begin

TEMP := ITEM_1;

ITEM_1 := ITEM_2;

ITEM_2 := TEMP;

end SWAP;

Generics

with SWAP;

procedure EXAMPLE is

 procedure INTEGER_SWAP is new SWAP(INTEGER);

 procedure CHARACTER_SWAP is new SWAP(CHARACTER);

 NUM_1, NUM_2 : INTEGER;

 CHAR_1, CHAR_2 : CHARACTER;

begin

 NUM_1 := 10;

 NUM_2 := 25;

 INTEGER_SWAP(NUM_1, NUM_2);

 CHAR_1 := 'A';

 CHAR_2 := 'S';

 CHARACTER_SWAP(CHAR_1, CHAR_2);

end EXAMPLE;

Generics

```
generic
  type DISCRETE_TYPE is (<>);
function NEXT(VALUE : in DISCRETE_TYPE)
  return DISCRETE_TYPE;
function NEXT(VALUE : in DISCRETE_TYPE)
  return DISCRETE_TYPE is
begin
  if VALUE = DISCRETE_TYPE'LAST then
    return DISCRETE_TYPE'FIRST
  else
    return DISCRETE_TYPE'SUCC(VALUE);
  end if;
end NEXT;
```

Generics

```
with NEXT;
with TEXT_IO; use TEXT_IO;
procedure MAIN_DRIVER is

  type DAYS is (MON, TUE, WED, THUR, FRI, SAT, SUN);
  TODAY, TOMORROW : DAYS;
  package DAYS_IO is new ENUMERATION_IO (DAYS);
  function DAY_AFTER is new NEXT (DAYS);

begin
  PUT ("Enter the day: ");
  DAYS_IO.GET (TODAY);
  TOMORROW := DAY_AFTER (TODAY);
  PUT ("Tomorrow is: ");
  DAYS_IO.PUT (TOMORROW);

end MAIN_DRIVER;
```

Generics

```
with NEXT;
with TEXT_IO; use TEXT_IO;
procedure MAIN_DRIVER_2 is

    type HOUR is range 1..12;
    THIS_HOUR, NEXT_HOUR : HOUR;
    package HOUR_JO is new ENUMERATION_JO (HOUR);
    function HOUR_AFTER is new NEXT (HOUR);

begin

    PUT ("The current hour is: ");
    HOUR_JO.GET (THIS_HOUR);
    NEXT_HOUR := HOUR_AFTER(THIS_HOUR);
    PUT ("Next hour is: ");
    HOUR_JO.PUT (NEXT_HOUR);

end MAIN_DRIVER_2;
```

Generics

generic

SIZE: in POSITIVE;
type ELEMENT is private;

package STACK is

STACK_UNDERFLOW,
STACK_OVERFLOW : exception;
procedure PUSH (ITEM:in ELEMENT);
procedure POP (ITEM:in out ELEMENT);

end STACK;

Generics

package body STACK is

SPACE: array (1..SIZE) of ELEMENT;

TOP: INTEGER range 0..SIZE:= 0;

procedure PUSH(ITEM:in ELEMENT)is

begin

if TOP = SIZE then

raise STACK_OVERFLOW;

end if;

TOP := TOP +1;

SPACE(TOP) := ITEM;

end PUSH;

procedure POP(ITEM:in out ELEMENT) is

begin

if TOP = 0 then

raise STACK_UNDERFLOW;

end if;

ITEM := SPACE(TOP);

TOP := TOP -1;

end POP;

end STACK;

Generics

```
with STACK;
with TEXT_IO; use TEXT_IO;
procedure STACK_OPS is

    package INT_IO is new INTEGER_IO (POSITIVE);
    use INT_IO;
    INT_ELEMENT : POSITIVE;
    STACK_SIZE : POSITIVE := 50;
    package INTEGER_STACK is new STACK
        (STACK_SIZE, POSITIVE);
    use INTEGER_STACK;
begin
    PUT ("Enter an element to push on the stack: ");
    GET (INT_ELEMENT);
    PUSH (INT_ELEMENT);
    POP (INT_ELEMENT);
    PUT ("The element popped off the stack was: ");
    PUT (INT_ELEMENT);
```

Generics

```
with STACK, TEXT_IJO; use TEXT_IJO;
procedure STACK_OPS_2 is

    STACK_SIZE : POSITIVE := 50;
    INT_ELEMENT : POSITIVE;
    FLOAT_ELEMENT : FLOAT;

    package INT_IJO is new INTEGER_IJO (POSITIVE);
    package REAL_IJO is new FLOAT_IJO (FLOAT);
    package INT_STACK is new STACK (STACK_SIZE, POSITIVE);
    package FLOAT_STACK is new STACK (100, FLOAT);

    use INT_IJO, REAL_IJO, INT_STACK, FLOAT_STACK;

begin

    PUT ("Enter a positive element to push on the stack: ");
    GET (INT_ELEMENT);
    PUSH (INT_ELEMENT);

    PUT ("Enter a FLOAT element to push on the stack: ");
    GET (FLOAT_ELEMENT);
    PUSH (FLOAT_ELEMENT);

end STACK_OPS_2;
```

Generics

generic

type ELEM is private;

with function "*" (LEFT, RIGHT : ELEM)

return ELEM is < >;

function SQUARING (X : ELEM) return ELEM;

function SQUARING (X : ELEM) return ELEM is

begin

return X * X;

end SQUARING;

Generics

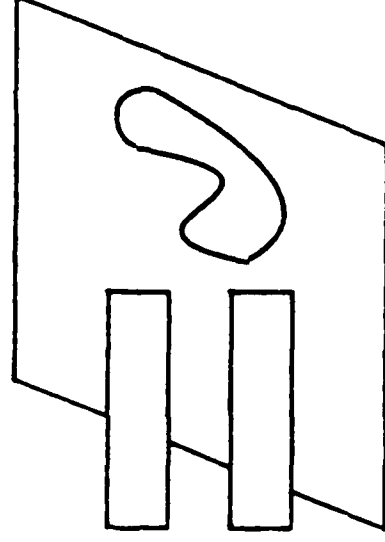
```
with SQUARING;  
procedure MATH_PROGRAM is  
  
    function SQUARE is new SQUARING (INTEGER);  
    X : INTEGER := 8;  
  
begin  
  
    X := SQUARE (X);  
  
end MATH_PROGRAM;
```

Generics

```
with SQUARING;  
procedure MATH_PROGRAM_2 is  
  type MATRIX is array (1..3, 1..3) of INTEGER;  
  A_MATRIX : MATRIX :=  
    (others => (others => 2));  
  function MULT (LEFT, RIGHT : MATRIX) return  
    MATRIX is separate;  
  function SQUARE_A_MATRIX is new SQUARING  
    (MATRIX, MULT);  
begin  
  A_MATRIX := SQUARE_A_MATRIX (A_MATRIX);  
end MATH_PROGRAM_2;
```

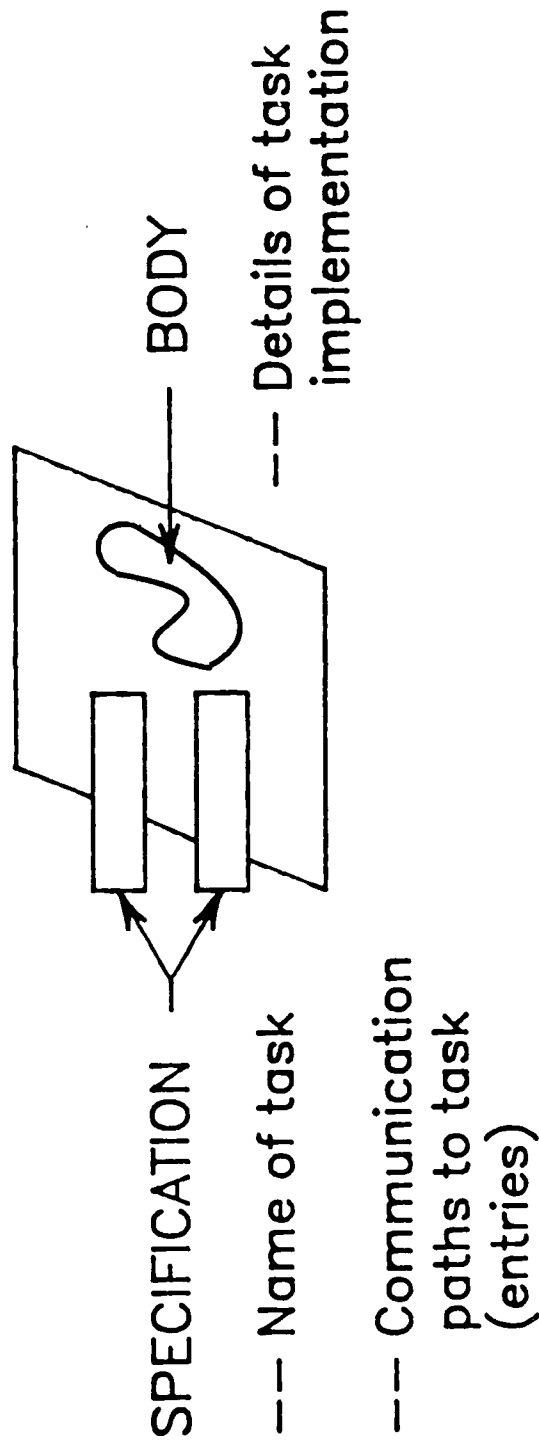
```
generic
    type ELEMENT_TYPE is private;
    procedure SWAP ( LEFT, RIGHT : in out ELEMENT_TYPE );
    procedure SWAP ( LEFT, RIGHT : in out ELEMENT_TYPE ) is
        TEMP_ELEMENT : ELEMENT_TYPE := LEFT;
    begin
        LEFT := RIGHT;
        RIGHT := TEMP_ELEMENT;
    end SWAP;
```

Tasks



- A task is an entity that operates in parallel with other entities
- Tasking may be implemented on
 - Single Processors
 - Multi-processors
 - Multi-computers

Tasks



Tasks

```
procedure SENSOR_CONTROLLER is
```

```
function OUT_OF_LIMITS return BOOLEAN;  
procedure SOUND_ALARM;
```

```
task MONITOR_SENSOR; --- specification  
task body MONITOR_SENSOR is --- body  
begin
```

```
    loop
```

```
        if OUT_OF_LIMITS then
```

```
            SOUND_ALARM;
```

```
        end if;
```

```
    end loop;
```

```
end MONITOR_SENSOR;
```

```
function OUT_OF_LIMITS return BOOLEAN is separate;  
procedure SOUND_ALARM is separate;
```

```
begin
```

```
    null; --- Task is activated here
```

```
end SENSOR_CONTROLLER;
```

Tasks

```
-- a basic task with no communication
with TEXTJO; use TEXTJO;
procedure COUNT_NUMBERS is
package INTJO is new INTEGERJO (INTEGER);
use INTJO;
task COUNT_SMALL;
task COUNT_LARGE;

task body COUNT_SMALL is
begin
  for INDEX in -100..0 loop
    PUT(INDEX);
    NEW_LINE;
  end loop;
end COUNT_SMALL;

task body COUNT_LARGE is
begin
  for INDEX in 0..100 loop
    PUT(INDEX);
    NEW_LINE;
  end loop;
end COUNT_LARGE;

begin
  null; --tasks are started here
end COUNT_NUMBERS;
```

Tasks

```
--Tasks can communicate with each other  
-- via parameters defined in entries
```

```
task CHANNEL is  
    entry PRINT(JOB:in JOB_NUMBER);  
end CHANNEL;
```

```
--To communicate use an "entry" call  
CHANNEL.PRINT(24);
```

```
--When two tasks are synchronized in time  
-- and are communicating, we say that the  
-- two tasks are in "rendezvous"
```

Tasks

--Inside a task, rendezvous occurs when
-- a task's entry has been called and
-- an accept statement is reached

task body CHANNEL is

 LOCAL_NUMBER : JOB_NUMBER;

begin

 loop

 accept PRINT(JOB:in JOB_NUMBER)do

 LOCAL_NUMBER := JOB;

 end;

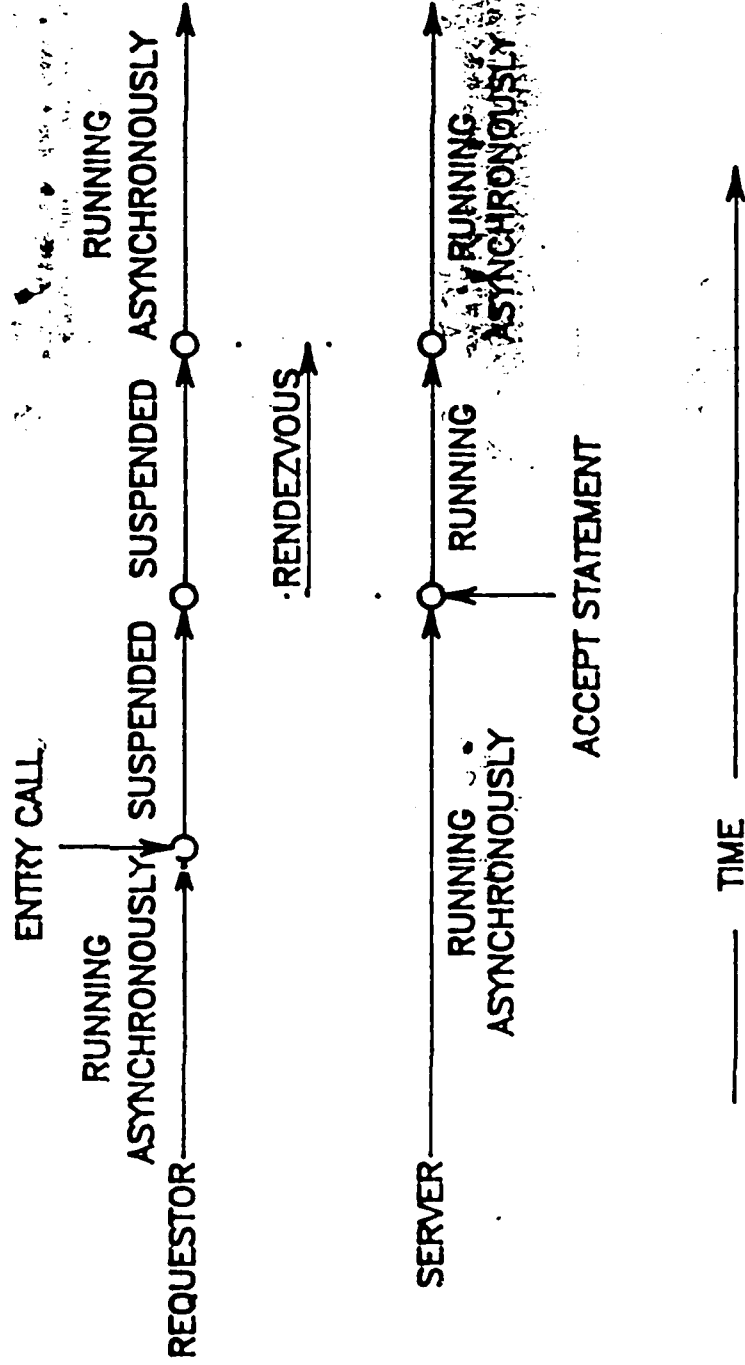
 CALL_PRINTER (LOCAL_NUMBER);

 end loop;

end CHANNEL;

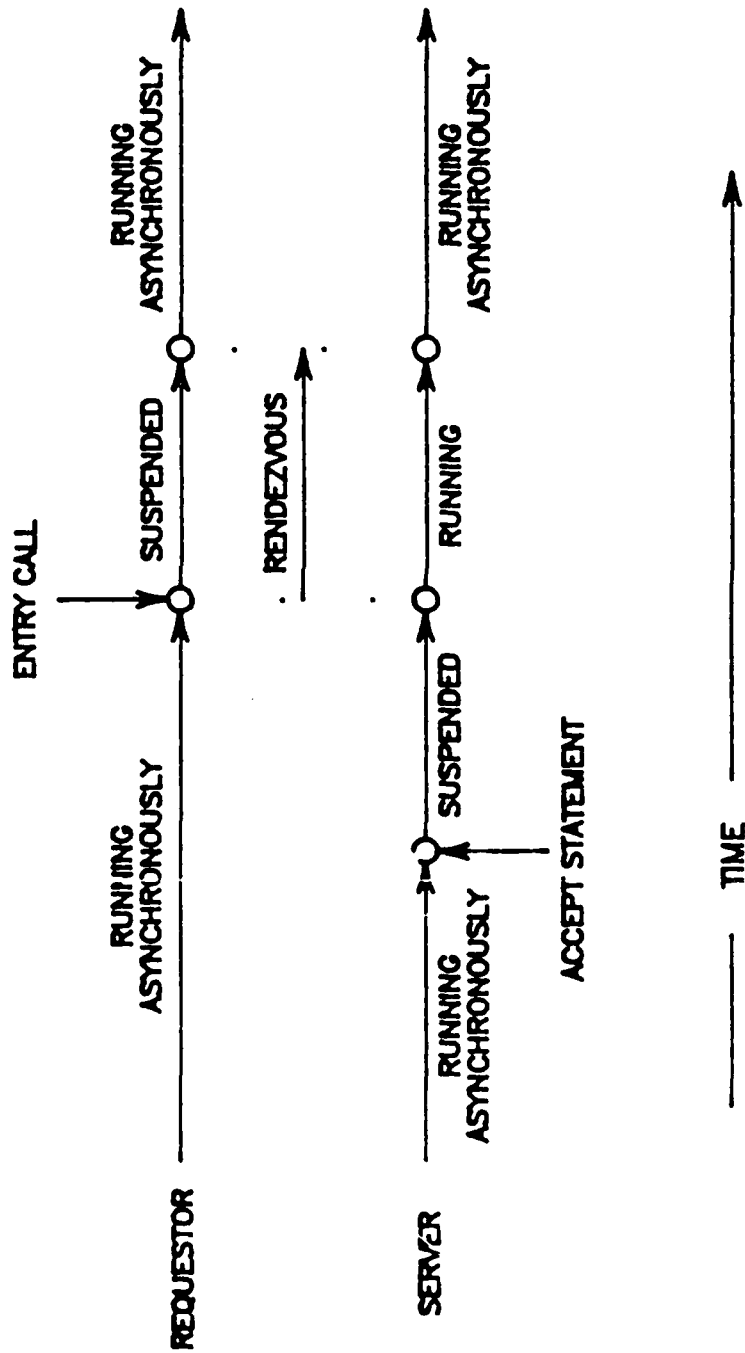
Tasks

STAGES OF A RENDEZVOUS (ENTRY CALL FIRST)



Tasks

STAGES OF A RENDEZVOUS (ACCEPT FIRST)



Tasks

Tasking statements

ENTRY CALL
DELAY
ABORT
ACCEPT
SELECT

Tasks

DELAY -----

- Used to suspend execution for at least
- the time interval specified
- delay 30.0;

ABORT -----

- Used to unconditionally terminate a task
- Only used in extreme circumstances
- abort CHANNEL;

Tasks

SELECT

```
--Used to choose between entries in a task
task DRIVE_CONTROL is
    entry READ(DATA: out DATA_TYPE);
    entry WRITE(DATA: in DATA_TYPE);
end DRIVE_CONTROL;

task body DRIVE_CONTROL is
begin
    loop
        select
            accept READ(DATA:out DATA_TYPE)do
                .
            end;
        or
            accept WRITE(DATA:in DATA_TYPE)do
                .
            end;
        end select;
    end loop;
```

with LIST_PACKAGE, TEXTJO;
use LIST_PACKAGE, TEXTJO;
procedure ORDERLIST is

UNSORTED_FILE : FILE_TYPE;
SORTED_FILE : FILE_TYPE;

MAX_ITEMS : constant := 20;

THE_LIST : ALIST(1..MAX_ITEMS);
LIST_INDEX : POSITIVE := 1;

LAST : NATURAL;
FILE_NAME : STRING(1..40);

```

begin
    PUT_LINE ("This program sorts a list of names, addresses and ");
    PUT_LINE ("phone numbers and puts that sorted list in a file.");
    NEW_LINE (2);
    PUT_LINE ("What is the name of the file to sort?");
    GET_LINE (FILE_NAME, LAST);
    OPEN (UNSORTED_FILE, IN_FILE, FILE_NAME (1..LAST));
    while not END_OF_FILE (UNSORTED_FILE) loop
        GET_LINE (UNSORTED_FILE, THE_LIST (LIST_INDEX).NAME, LAST);
        GET_LINE (UNSORTED_FILE, THE_LIST (LIST_INDEX).ADDRESS, LAST);
        GET_LINE (UNSORTED_FILE, THE_LIST (LIST_INDEX).PHONE_NUMBER, LAST);
        LIST_INDEX := LIST_INDEX + 1;
    end loop;
    SORT (THE_LIST (1..LIST_INDEX - 1));
    CLOSE (UNSORTED_FILE);

```

package LIST_PACKAGE is

MAX_LINE_LENGTH : constant := 80;

subtype ALINE is STRING(1..MAX_LINE_LENGTH);

type ITEMS is record

NAME : ALINE := (others => '');

ADDRESS : ALINE := (others => '');

PHONE_NUMBER := (others => '');

end record;

type ALLIST is array(POSITIVE range <>) of ITEMS;

procedure SORT (ANY_LIST : in out ALLIST);

end LIST_PACKAGE;

```

with SWAP;
package body LIST_PACKAGE is

  procedure SWAP_ITEMS is new SWAP ( ELEMENT_TYPE => ITEMS );

  procedure SORT ( ANY_LIST : in out A_LIST ) is
  -- implements a selection sort
    SMALLEST_INDEX, TEMP_INDEX : POSITIVE;
    SMALLEST_NAME : A_LINE := ( others => ' ' );

  begin
    for SORTED_INDEX in ANY_LIST'RANGE loop
      SMALLEST_INDEX := SORTED_INDEX;
      for CHECK_INDEX in (SORTED_INDEX+1)..ANY_LIST'LAST loop
        if ANY_LIST ( CHECK_INDEX).NAME <
           ANY_LIST (SMALLEST_INDEX).NAME then
          SMALLEST_INDEX := CHECK_INDEX;
        end if;
      end loop;
      SWAP_ITEMS ( ANY_LIST(SMALLEST_INDEX),
                   ANY_LIST(SORTED_INDEX) );
    end loop;
  end SORT;

end LIST_PACKAGE;

```

```
PUT_LINE("What is the name of the file to output to?");
GET_LINE( FILE_NAME, LAST );

CREATE ( SORTED_FILE, OUT_FILE, FILE_NAME(1..LAST) );

for FILE_ITEM in 1 .. LIST_INDEX - 1 loop

    PUT_LINE( SORTED_FILE,THE_LIST(FILE_ITEM).NAME );
    PUT_LINE(SORTED_FILE,THE_LIST(FILE_ITEM).ADDRESS );
    PUT_LINE(SORTED_FILE,THE_LIST(FILE_ITEM).PHONE_NUMBER);

    NEW_LINE(SORTED_FILE);

end loop;

CLOSE ( SORTED_FILE );

end ORDER_LIST;
```

END

DATE

FILMED

5-88

DTIC