16.070

Introduction to Computers & Programming

Ada IV
Introduction to packages, decision statements, writing functions

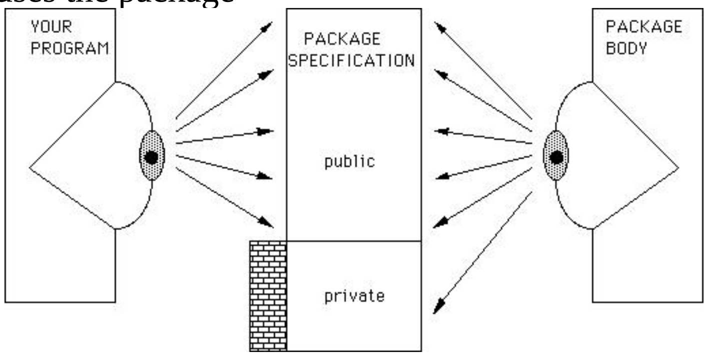
Guest Lecturer: Tony Bogner
Charles Stark Draper Laboratory

Packages

- Collection of resources
- Resources could include types, functions, procedures, object (data) declarations, even other packages
- Encapsulated in one unit
- Compiled on its own
  - Compilation order:
    - Library unit
    - Procedures that use it

Package Organization

- Package specification show "what" it provides
- Package body defines "how" it is implemented
- Both are separate from the user's program that uses the package



ADT Packages

- Different kinds of resources provided by a package
  - Types and subtypes
  - Procedures, functions

```
package Ada.Calendar is
  -- standard Ada package, must be supplied with compilers
  -- provides useful services for dates and times
  type Time is private;
  subtype Year_Number  is Integer range 1901 .. 2099;
  subtype Month_Number is Integer range 1 .. 12;
  subtype Day_Number   is Integer range 1 .. 31;
  function Clock return Time;
  function Year  (Date : Time)  return Year_Number;
  function Month (Date : Time)  return Month_Number;
  function Day   (Date : Time)  return Day_Number;
end Ada.Calendar;
```

SEPTEMBER 30, 2002

Problem specification

Display today's date in the form MONTH dd, yyyy

```
WITH Ada.Text_IO;
WITH Ada.Integer_Text_IO;
WITH Ada.Calendar;
PROCEDURE Todays_Date IS
  TYPE Months IS (January, February, March, April, May,
                  June, July, August, September, October,
                  November, December);
  PACKAGE Months_IO IS
    NEW Ada.Text_IO.Enumeration_IO(Enum => Months);
  RightNow  : Ada.Calendar.Time;            -- current time
  ThisYear  : Ada.Calendar.Year_Number;     -- current year
  ThisMonth : Ada.Calendar.Month_Number;    -- current month
  ThisDay   : Ada.Calendar.Day_Number;      -- current day
  MonthName : Months;
```


Today's Date

```
BEGIN -- Todays_Date
  -- Get the current time value from the computer's clock
  RightNow := Ada.Calendar.Clock;

  -- Extract current month, day, and year from the time value
  ThisMonth := Ada.Calendar.Month(Date => RightNow);
  ThisDay   := Ada.Calendar.Day  (Date => RightNow);
  ThisYear  := Ada.Calendar.Year (Date => RightNow);

  -- Format and display the date
  MonthName := Months'Val(ThisMonth - 1);

  Ada.Text_IO.Put (Item => "Today's date is ");
  Months_IO.Put (Item => MonthName, Set => Ada.Text_IO.Upper_Case);
  Ada.Text_IO.Put (Item => ' ');
  Ada.Integer_Text_IO.Put (Item => ThisDay, Width => 1);
  Ada.Text_IO.Put (Item => ',');
  Ada.Integer_Text_IO.Put (Item => ThisYear, Width => 5);
  Ada.Text_IO.New_Line;
END Todays_Date;
```

Sample Run:
Today's date is FEBRUARY 18, 2003

16.070 — February 26/2003 — Tony Bogner — Draper Laboratory

Ada's Math Library

```
WITH Ada.Text_IO;
WITH Ada.Float_Text_IO;
WITH Ada.Numerics.Elementary_Functions;

PROCEDURE Square_Roots IS

  SUBTYPE NonNegFloat IS Float RANGE 0.0 .. Float'Last;

  First : NonNegFloat;
  Second: NonNegFloat;
  Answer: NonNegFloat;

BEGIN  -- Square_Roots
  Ada.Text_IO.Put (Item => "Please enter first number > ");
  Ada.Float_Text_IO.Get(Item => First);
  Answer := Ada.Numerics.Elementary_Functions.Sqrt(X => First);
  Ada.Text_IO.Put (Item => "The first number's square root is ");
  Ada.Float_Text_IO.Put
    (Item => Answer, Fore => 1, Aft => 5, Exp => 0);
  Ada.Text_IO.New_Line;

  Ada.Text_IO.Put (Item => "Please enter second number > ");
  Ada.Float_Text_IO.Get(Item => Second);
  Ada.Text_IO.Put (Item => "The second number's square root is ");
  Ada.Float_Text_IO.Put
    (Item => Ada.Numerics.Elementary_Functions.Sqrt (X => Second),
     Fore => 1, Aft => 5, Exp => 0);
  Ada.Text_IO.New_Line;

  Answer := Ada.Numerics.Elementary_Functions.Sqrt(X => First + Second);
  Ada.Text_IO.Put
    (Item => "The square root of the sum of the numbers is ");
  Ada.Float_Text_IO.Put
    (Item => Answer, Fore => 1, Aft => 5, Exp => 0);
  Ada.Text_IO.New_Line;
END Square_Roots;
```

Sample Run

Please enter first number > 9

The first number's square root is 3.00000

Please enter second number > 16

The second number's square root is 4.00000

The square root of the sum of the numbers is 5.00000

16.070 — February 26/2003 — Tony Bogner — Draper Laboratory

Decision Statements

- Making a decision in your program typically involves an `IF` statement
- IF statement structure:

```
IF condition THEN
    Do something special if boolean condition is TRUE
END IF;
```

- Condition must be a Boolean expression—that is, an expression that evaluates to `TRUE` or `FALSE`
- Simple Example:

```
…
IF Hungry THEN
    Eat;
END IF;
…
```


Boolean Expressions

- Boolean expressions typically have the form
  - Variable relational operator variable
  - Variable relational operator constant
- Typical relational operators

Ada Symbol	Operation Implied by Symbol
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
=	Equal to
/=	Not equal to

Example Boolean Expressions

- Example Boolean Expressions:

Given this type and declaration:

```
TYPE Gps_Frequencies IS (L1, L2, L5);
Current_Freq : Gps_Frequencies := L1;
```

Following boolean expressions evaluate as indicated:

```
Current_Freq = L1      -- will evaluate to TRUE

Current_Freq = L2      -- will evaluate to FALSE

Current_Freq < L2      -- will evaluate to TRUE
```

- Other examples:

```
      3 < 6     -- Integer comparison will evaluate to TRUE
3.14159 >= 3.1 -- Floating point comparison will evaluate to TRUE
```


Example IF Statement

- More complex `IF` statements

```
IF Current_Freq = L2 THEN
   measure ionospheric delay
ELSE
   model ionospheric delay
END IF;
```

IF statement with two alternatives

```
IF Current_Freq = L1 THEN
   Put_Line("Tracking on L1");
ELSIF Current_Freq = L2 THEN
   Put_Line("Tracking on L2");
ELSE
   Put_Line("Not tracking on L1 or L2");
END IF;
```

IF statement with multiple alternatives

Functions and Specifications

- Function implements an algorithm that may accept input arguments and will always return a result
- Function specification tells you about the interface to the function
  - Function starts with reserved word `FUNCTION`
  - Function name must be an identifier or an operator
  - Function name is followed by a list of expected parameters, if any
  - Parameters are followed by reserved word `RETURN` and then the type of the result returned from the function
- Example function specifications:

```
FUNCTION Factorial (N: Positive) RETURN Positive;
FUNCTION Minimum (Value1, Value2: Integer) RETURN Integer;
FUNCTION Get_Current_Freq RETURN Gps_Frequencies;
```


Calling A Function

- Invocation of the function requires that any required parameters be provided and that return result is in appropriate context
- Example function calls:

```
Three_Factorial : Positive := Factorial(N=> 3);

Min_Value : Float := Minimum(Value1 => 1, Value2 => 3);

IF Get_Current_Freq = L2 THEN
  measure ionospheric delay
ELSE
  model ionospheric delay
END IF;
```


Function Body

- Function body implements the algorithm of the function
- Examples:

```
FUNCTION Factorial (N: Positive) RETURN Positive IS
BEGIN
  IF N = 1 THEN
     RETURN 1;
  ELSE
     RETURN N * Factorial(N-1);
  END IF;
END Factorial;
```

```
FUNCTION Minimum (Value1, Value2: Integer) RETURN Integer IS
  Result: Integer;  -- Local variable in function
 BEGIN
  IF Value1 < Value2 THEN
    Result := Value1;
  ELSE
    Result := Value2;
  END IF;
  RETURN Result;
END Minimum;
```


16.070

Scalar Types Supplement

Ada III (Supplement)
What you “get” from Ada for integer and floating point types

Guest Lecturer: Tony Bogner
Charles Stark Draper Laboratory

What You Get With An Integer Type

- Predefined operators, membership tests, attributes and conversions that come with integer types

Operator	Operand(s)		Result
ABS	integer type		Same integer type
**	integer type	NATURAL	Same integer type
REM, MOD	integer type	same integer type	Same integer type
*, /,	integer type	same integer type	Same integer type
+, - (unary)	integer type		Same integer type
+, - (binary)	integer type	same integer type	Same integer type
<, <=, =, /=, >=, >	integer type	same integer type	BOOLEAN

Other	Operand(s)		Result
In, not in	integer type	same integer range	BOOLEAN
ATTRIBUTES	(see Annex K of Ada LRM '95)		
Type conversion	numeric		integer type

Type conversion always rounds away from 0:

1.4 becomes 1
1.6 becomes 2
1.5 becomes 2
-1.5 becomes -2

What You Get With A Floating Point Type

- Predefined operators, membership tests, attributes and conversions that come with floating point types

Operator	Operand(s)		Result
ABS	float type		Same float type
**	float type	INTEGER	Same float type
*, /,	float type	same float type	Same float type
+, - (unary)	float type		Same float type
+, - (binary)	float type	same float type	Same float type
<, <=, =, /=, >=, >	float type	same float type	BOOLEAN

Other	Operand(s)		Result
In, not in	float type	same float range	BOOLEAN
ATTRIBUTES	(see Annex K of Ada LRM '95)		
Type conversion	numeric		float type