Introduction to Computers and Programming

Prof. I. K. Lundqvist

Reading: B pp. 217-228; FK pp. 65-111
Sept 9 2003

---

General structure of Ada programs

with ...;

----------------------------------
-- header
----------------------------------

procedure program_name is

    declare constants & variables used

begin -- program_name

    statements

end program_name;
General structure of Ada programs

with Ada.Text_Io;

procedure Hello_Name is

-- Requests, then displays, user's name
-- Author: Michael Feldman, The George Washington University
-- Last Modified: June 1998
-- object to hold user's name

FirstName: String(1..10);

begin

-- Prompt for (request user to enter) user's name
Ada.Text_IO.Put(Item => "Enter your first name, exactly 10 letters.");
Ada.Text_IO.New_Line;
Ada.Text_IO.Put(Item => "Add spaces at the end if it's shorter.> ");
Ada.Text_IO.Get(Item => FirstName);

-- Display the entered name, with a greeting
Ada.Text_IO.Put(Item => "Hello ");
Ada.Text_IO.Put(Item => FirstName);
Ada.Text_IO.Put(Item => ". Enjoy studying Ada!");
Ada.Text_IO.New_Line;

end Hello_Name;

Modules

- Procedure
  - Abstracts an operation

- Package
  - Collects related operations and data types

Advantages of modules

- Procedures
  - Functional abstraction
  - Top-down development
  - Reduced complexity
  - Parallel development
  - Avoid duplication

- Packages
  - Shared resources
  - Improved productivity
  - Improved quality
Procedure

- First we see how a program to write "ADA" in giant letters would be written as a monolithic program. Then we look at it when it is broken into procedures. You can see that a procedure only needs to be written once, and can then be invoked as many times as necessary. The resultant shortening of the program is one of the benefits of procedures.
- Giant_ada_1.adb, giant_ada_2.adb

Programs and packages

- Package
  - Collection of resources
  - Encapsulated in one unit
  - Ex: Text_IO, Calendar, user-defined packages
    - Used for:
      - Collection of types and constants
      - Group of related subprograms
      - User defined types and allowable operation
Reserved words and identifiers

- **Reserved words**
  - abort abs accept access all and array at `begin`
  - body case constant declare delay delta digits else elsif end entry exception exit for function generic goto if in `is` limited loop mod new not null of or others out `package` pragma private
  - `procedure` raise range record rem renames return reverse select separate subtype task terminate then type use when while `with` xor

- **Pre-defined words**
  - Boolean Character Close Create Delete False Float Get Integer Natural `New_Line` Open Put Put_Line Positive Read Reset `Skip_Line` String Text_Io True Write

Layout conventions

- **Common layout convention** makes programs easier for others to read, understand (and mark!)

- **Basic conventions**
  - One statement (one thought) per line
  - Break long lines into readable segments
  - Indent lines to show different parts of program
  - Blank lines separate parts of the program
  - Comments help readers understand program
-- Comments

- Good comments:
  - are always correct and up to date
  - conform to usual conventions of prose
  - provide information not immediately obvious
  - describe the intended effect of (part of) the program

- Minimum comments in any program:
  - the name of the program
  - who wrote it and when
  - description of what the program does
  - description of any constants or variables
  - description of purpose of each segment of code
  - assumptions made (precondition / postcondition)

Types of statements
Input/Output, Assignment, Control statements

- Input/Output libraries
  - Text: Ada.Text_Io
  - Integer: Ada.Integer_Text_Io
  - Float: Ada.Float_Text_Io
  - Own type: define new library

  type Colors is (white, black, red, purple);
  package Color_Io is
    new Ada.Text_Io Enumeration_Io (Enum => Colors);

  One_Color : Colors;

  begin -- procedure_name
    Color_Io.Get (Item => One_Color);
Types of statements
Input/Output, Assignment, Control statements

Input

• **Get** (argument)
  - Argument is a variable that receives input values
  - Value must be same type (e.g., integer) as variable

  ```
  Put (Item => "Please enter the first number: ");
  Get (Item => Number1);
  ```

• **Skip_Line**
  - Advance to next line, ignoring unused input

  ```
  Put (Item =>
       "Please enter the first number ");
  Get (Item => Number1); Skip_Line;
  ```

  ```
  Put (Item =>
       "Please enter the second number ");
  Get (Item => Number2); Skip_Line;
  ```

Please enter the first number 42 10
Please enter the second number 23
Types of statements
Input/Output, Assignment, Control statements

Output

- Put (argument)
  - Print argument
  - Leave the cursor on the same line

- Put(Item =>
  "Please enter the first number: ");
Get(Item => Number1); Skip_Line;

Please enter the first number: 42

The sum of the numbers is: 14
The product of the numbers is: 48

- Formatted output
  
  Put(int_val, Width => positive_integer);
  Put ("The sum of the numbers is:");
  Put (Number1+Number2, Width=>7); New_Line;
  Put ("The product of the numbers is:");
  Put (Number1*Number2, Width=>3); New_Line;
  Put ("The sum of the numbers is:");
  Put (Number1+Number2, Width=>1); New_Line;

  The sum of the numbers is: 14
  The product of the numbers is: 48
  The sum of the numbers is:14
Types of statements
Input/Output, Assignment, Control statements

Put(real_val, Fore => positive_integer,
    Aft  => positive_integer,
    Exp  => positive_integer);

Put (23.456);
Put (23.456, Exp=>0);
Put (23.456, Aft=>3, Exp=>0);
Put (23.456, Aft=>2, Exp=>0);
Put (23.456, Fore=>3, Aft=>3, Exp=>0);

' 2.34560000000000E+01'
'23.45600000000000'
'23.456'
'23.46'
' 23.456'

Types of statements
Input/Output, Assignment, Control statements

• Assignment
  – Perform calculation and save result in a variable

    Total_Num := Number1 + Number2;
Data types
Storing data values

• A **variable** has a
  - Name
    • An Identifier
    • What does the variable **represent**?
  - Data type
    • What **values** can the variable have?
    • What **operations** can be performed on it?
  - Main pre-declared data types in Ada
    • Integer
    • Float
    • Character
    • String
    • Boolean

Data types
Storing data values

• **Constants** are data values that does not change
  - Name : **constant** Type := Value;

Answer : **constant** String := "forty two";
Medicare_Rate : **constant** Float := 1.4;
Pi : **constant** Float := 3.1415926536;

English_Drink := Metric_Drink * 0.568;
Liters_To_Pints : **constant** Float := 0.568;
English_Drink := Metric_Drink * Liters_To_Pints;
Data types
Storing data values

- Ada has **strong typing**
  - 3 + 4
  - 3.0 / 4.0
  - 1.0 > 0
  - 3 * 4.0

Mixed arithmetic: must convert one type to another

- 1.0 > FLOAT(0)
- FLOAT(3) * 4.0
- 3 * INTEGER(4.0)

Data types
Integer type

- Positive or negative number with **no** decimal part
  - 354 -52689 +4432
- Range of integers
  - `Integer'First`: smallest integer on given system
  - `Integer'Last`: largest integer on given system

- Put ("The lowest integer value is: ");
  - Put (Integer'First); New_Line;
Data types
Integer type

• arithmetic
  - unary minus (negation) \(-\text{int}_\text{val}\)
  - absolute value \(\text{abs } \text{int}_\text{val}\)
  - division \(23 / 4 = 5\)
  - remainder \(23 \text{ rem } 4 = 3\), \(-23 \text{ rem } 4 = -3\)
  - modulus \(-23 \text{ mod } 4 = 1\), \(23 \text{ mod } -4 = -1\)
  - exponentiation \(2 ** 4 = 16\)

• relational:
  = /= < > <= >=