16.070
Introduction to Computers & Programming

Ada II
Introduction to Straight-line programs

Prof. I. Kristina Lundqvist
Dept. of Aero/Astro, MIT
WITH package1;
WITH package2;
...
WITH package3;
PROCEDURE general_form IS

  declarations (variables, constants, etc.)

BEGIN -- general_form

  program statement;
  ...
  program statement;

END general_form;
WITH Ada.Text_IO;
WITH Ada.Integer_Text_IO;
PROCEDURE Distance IS

-- Finds distance traveled, given travel time and average speed
-- Author: Michael B. Feldman, The George Washington University
-- Last Modified: June 1998

HowLong : Natural;
HowFast : Natural;
HowFar : Natural;

BEGIN -- Distance

-- prompt user for hours and average speed
Ada.Text_IO.Put
(Item => "How many hours will you be driving (integer) ? ");
Ada.Integer_Text_IO.Get (Item => HowLong); Ada.Text_IO.Skip_Line;
Ada.Text_IO.Put
(Item => "At what average speed (miles per hour, integer) ? ");
Ada.Integer_Text_IO.Get (Item => HowFast); Ada.Text_IO.Skip_Line;

-- compute distance driven

-- display results
Ada.Text_IO.Put (Item => "You will travel about ");
Ada.Integer_Text_IO.Put (Item => HowFar);
Ada.Text_IO.Put (Item => " miles");
Ada.Text_IO.New_Line;

END Distance;
Constants

- Data values that do not change their value
  - CMPerInch: CONSTANT Float := 2.54;
  - Pi : CONSTANT Float := 3.1415926536;
  - Answer : CONSTANT Integer := 42;
  - MyName : CONSTANT String := “kristina”;
  - Name : CONSTANT Type := value; -- Form

- Benefits of using constants:
  - Easier to read
  - Only one line has to be modified
  - Ada statement that tries to change value of constant \(\rightarrow\) compilation error
Variables

- Storing input data and computational results
  - HowLong : Natural;
  - Initial1, Initial2 : Character;
  - Inches : Float;
  - Name : DataType;  --Form
- Main pre-declared data types in Ada: integer, float, character, string, and boolean
- **Integer (Natural, Positive)**
  - positive or negative number with no decimal part
  - 123  -456  +789
  - `Put ("The lowest integer value is: ");
    Put (Integer'First); New_line;
    Put ("The highest integer value is: ");
    Put (Integer'Last); New_line;`

- **Float**
  - Positive or negative number with decimal part
  - 123.4456  1.234e-4

- **Character**
  - Single character: Y/N answer

- **String**
  - Represent a sequence of characters as a single unit of data

- **Boolean**
  - Logical test
Strong Typing

- Cannot mix data types
  - 1 + 6
  - 4.2 / 7.0
  - 16 > 0.70
  - 12 * 3.0

- Mixed arithmetic
  - Float (16) > 0.70
  - 12 * Integer (3.0)
procedure Roundoff is   -- demonstrate rounding errors
  Zero        : constant       := 0.0;
  One         : constant Float := 1.0;
  Ten_Thousand : constant       := 10000;
  One_Ten_Thousandth : constant Float := One / Float (Ten_Thousand);
  The_Result :          Float := Zero;

begin -- roundoff

  -- the_result is initially equal to zero

  for Counter in Integer range 1 .. Ten_Thousand loop
    The_Result := The_Result + One_Ten_Thousandth;
  end loop;

  -- mathematically, the_result is now equal to
  -- 1/10_000 * 10_000 = 1.0

  if (The_Result = One) then
    Put_Line("Isn't this what you expected!" );
  else
    Put_Line("Isn't this a surprise!");
    Put("One is "); Put(One, Exp => 0); New_Line;
    Put("The result is "); Put(The_Result, Aft => 8, Exp=>0);
    New_Line;
  end if;
end Roundoff;
Isn’t this a surprise!
One is 1.00000
The result is 1.00005352
Identifiers

1. P1
2. This_will_always_be_first_item_in_list
3. Car_1
4. First_car
5. i
6. N
7. Lecture_1
8. Lecture_2

Identifiers should be as meaningful as possible, without being verbose
Assignment

- Used to perform computations and save result in a variable
  - \( \text{SquareYards} := \text{MetersToYards} \times \text{SquareMeters} \);
  - \( \text{NewX} := \text{X} \);
  - \( \text{NewX} := -\text{X} \);
  - \( \text{Sum} := \text{Sum} + \text{Item} \);
Input/Output Statements

- Get Procedure (Character)
  - Ada.Text_IO.Get (Item => Initial);

- Get Procedure (String)
  - Ada.Text_IO.Get (Item => First_Name);

- Get Procedure (Integer)
  - Ada.Integer_Text_IO.Get (Item => How_Long);

- Get Procedure (Floating Point)
  - Ada.Float_Text_IO.Get (Item => Inches);

When prompting for values from a user, always follow Get with Skip_Line
Input/Output Statements

- **Put Procedure (Character)**
  - Ada.Text_IO.Put (Item => Initial);

- **Put Procedure (String)**
  - Ada.Text_IO.Put (Item => First_Name);

- **New_Line Procedure**
  - Ada.Text_IO.New_Line (Spacing => 3);

- **Put Procedure (Integer)**
  - Ada.Integer_Text_IO.Put (Item => How_Long, Width => 5);

<table>
<thead>
<tr>
<th>Value</th>
<th>Width</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>234</td>
<td>4</td>
<td>234</td>
</tr>
<tr>
<td>234</td>
<td>6</td>
<td>234</td>
</tr>
<tr>
<td>-234</td>
<td>6</td>
<td>-234</td>
</tr>
<tr>
<td>234</td>
<td>Len</td>
<td>234</td>
</tr>
<tr>
<td>234</td>
<td>1</td>
<td>234</td>
</tr>
</tbody>
</table>
### Put Procedure (Floating Point)

- **Ada.Float_Text_IO.Put**
  
  
  ```
  (Item => Inches, Fore => 5, Exp => 0);
  ```

<table>
<thead>
<tr>
<th>Value</th>
<th>Fore</th>
<th>Aft</th>
<th>Exp</th>
<th>Displayed value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.14159</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>• 3.14</td>
</tr>
<tr>
<td>3.14159</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3.14</td>
</tr>
<tr>
<td>3.14159</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>• 3.14159</td>
</tr>
<tr>
<td>3.14159</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3.142E+00</td>
</tr>
<tr>
<td>0.1234</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0.12</td>
</tr>
<tr>
<td>-0.006</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>-6.00E-3</td>
</tr>
<tr>
<td>-0.006</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>•• -0.006</td>
</tr>
</tbody>
</table>
Expressions with several operators

- W, X, Y, Z are Integers. Let X=3, Y=4, Z=7
  - \( W := X \times Y + Z; \) \( \Rightarrow W = 19 \)
  - \( W := X - Y + Z; \) \( \Rightarrow W = 6 \)
  - \( W := X - Y - Z; \) \( \Rightarrow W = -8 \)
  - \( W := X - (Y - Z); \) \( \Rightarrow W = 6 \)

- \( \pi \times R \times R \) is equivalent to \( \pi \times (R \times R) \)
  and not \( \pi \times R \times R \)

- Keep it simple
- Use a lot of parentheses
Errors

- Three main categories of programming errors:
  - Compilation errors
    - Syntax errors
    - Semantic errors
  - Run-time errors (exceptions)
    - Constraint error
    - Data error
  - Logic or algorithmic errors
Fred Donovan: Save all your files in your catalogue
What/ why pseudocode?

- **Pseudocode**: a kind of programming language. Its syntax and semantics are in general less strict, so that algorithms can be formulated at a higher, more abstract level.

- **Ex: Sorting algorithm**
  - while not at end of list
    - compare adjacent elements
    - if second is greater than first
    - switch them
    - get next two elements
    - if elements were switched
    - repeat for entire list

- Pseudocode cannot be compiled/executed, has no real formatting or syntax rules. It is one step towards producing the final code.

- **Benefit of pseudocode**
  - Enables programmer to concentrate on the algorithms without worrying about all the syntactic details of a particular programming language.