The Ada Programming Language

ELTE, IK, PNYF
Zsók Viktória, Ph.D.
zsv@elte.hu

A good programming language

- Is abstract enough
- Has good syntax and it is easy to follow
- Provides good programming tools
- Has dedicated tools to handle the complexity of programs
- Has easy, understandable semantics

Why Ada?

Pros:

- Clear, logical structure
- Differs from other imperative languages
- Contains the necessary properties to introduce a Pascal-like language for students

Cons:

- It can be difficult for beginners
- Less groups are using it

History

- Originally sponsored by US defense United States Department of Defense) organizes the development of HOL (High Order Language)
- Design a language that defines requirements, is competitive and parallel

Requirements

- Establishing requirements
- evaluating the existing languages:
- FORTRAN, COBOL, PL/I, HAL/S,
- TACPOL, CMS-2, CS-4, SPL/1, J3B, Algol 60,
- Algol 68, CORAL 66, Pascal, SIMULA 67,
- LIS, LTR, RTL/2, EUCLID, PDL2, PEARL,
- MORAL, EL-1

Augusta Ada Byron

- The need for new language: none of the existing ones fulfilled the requests
- Pascal, PL/I, ALGOL 68 used as starting point
- Named after Augusta Ada Byron (1815–1852), Countess of Lovelace, daughter of the poet Lord Byron
- Assistant of Charles Babbage (1791 –1871), an English polymath. He was a mathematician, philosopher, inventor and mechanical engineer, who is best remembered now for originating the concept of a programmable computer
- She was the first programmer, worked on mechanical analytical machine

Some of the main features

- An extremely strong, static and safe type system, which allows the programmer to construct powerful abstractions
- Modularity
- Information hiding: the language separates interfaces from implementation
- Portability
- Standardisation: Ada compilers all support exactly the same language; the only dialect, SPARK

Features

- Ada was originally targeted at embedded and real-time system
- strong typing, modularity mechanisms (packages), run-time checking, parallel processing (tasks, synchronous message passing, protected objects, and nondeterministic select statements), exception handling, and generics.

Versions

- Ada 83 based on Pascal
- Feature from: Euclid, Lis, Mesa, Modula,
- Sue, Algol 68, Simula 67, Alphard, CLU
- Ada 95 included: interfaces, parallel programming features, oo classes
- Ada 2005
- Ada 2012

References

- Ada Reference Manual - Language and Standard Libraries
- The Rationale for Ada
- John Barnes: Programming in Ada 2005
- http://en.wikibooks.org/wiki/Ada_Programming
- http://www.ada2012.org/
- And many more books and web-pages

Course requirement

- 2 programming assessments (mandatory homeworks)
- 2 lab exams – 2 marks
- 1 theoretical test – 1 mark
- Final mark: average of the 3 above marks

Ada structure

- Subprograms: procedures and functions
- Hierarchically included
- Packages with interfaces and implementations
- Generics
- Tasks
- Protected objects

The structure of a program

- Including parts of the standard libraries
- Specification: variables, types
- Implementation: statements and maybe an exception handling part

Getting started

```
with Ada.Text_IO;

procedure Hello is
begin
   Ada.Text_IO.Put_Line("Hello, world!");
end Hello;
```

Hello, world! - 2

```
with Ada.Text_IO;
use Ada.Text_IO;

procedure Hello is

begin
   Put_Line("Hello, world!");
   New_Line;
   Put_Line("I am an Ada program with package use.");
end Hello;
```

Compiling, running

- gnatmake hello.adb
- ./hello
- GPS – GNAT Programming Studio
- Emacs Ada-mode

Declaration part

- Apart from statements
- Variables, constants, exceptions, program unit declarations

N: Natural;

- Initial value;

B: Boolean := True;

- More variables of the same type / value

I, J: Integer;

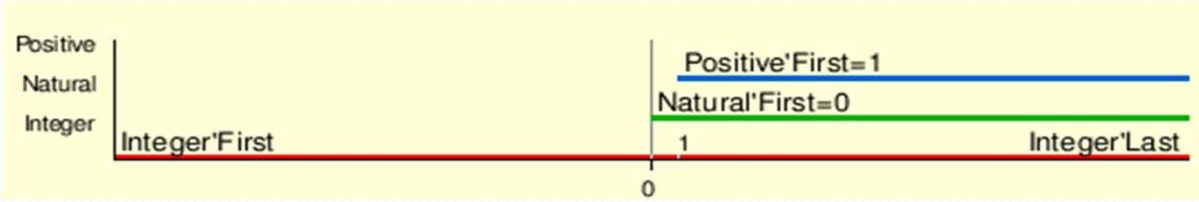
A, B: Positive := 3;

Max: constant Integer := 100;

- Integer, Natural, Positive, Boolean, Character, Float, String

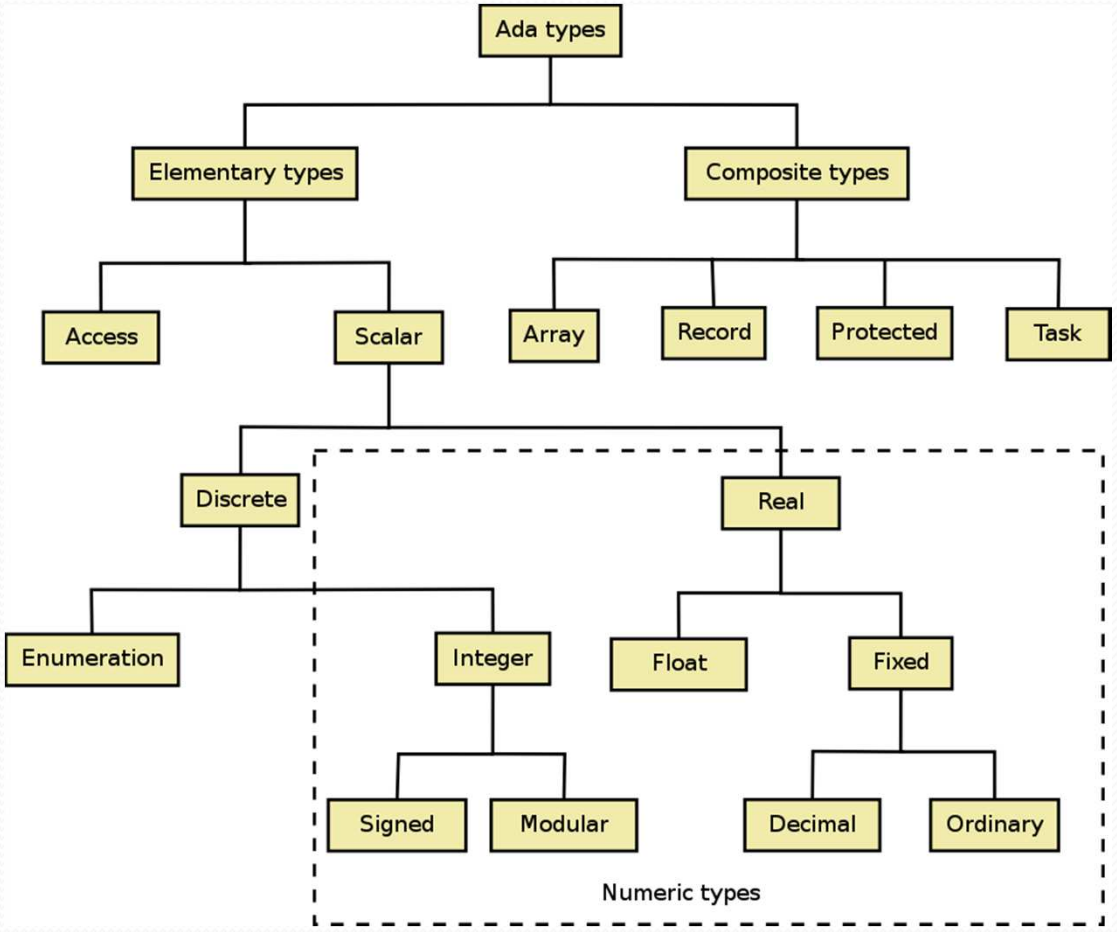
Predefined types

- Integer a value in (-2)**15+1 .. +2**15-1 interval, defines Natural and Positive as subtypes



- Float (define your own floating-point types, and specify your precision and range)
- Duration – a period of time in seconds
- Character - a special form of enumerations
- String – an array of characters
- Boolean - an enumeration of False and True

Type hierarchy



Type and subtype

- There is an important distinction between **type** and **subtype**: a type is given by a set of values and their operations.
- A subtype is given by a type, and a *constraint* that limits the set of values.
- Values are always of a type.
- Objects (constants and variables) are of a subtype

Boolean type

- Included in the Standard package (automatically can be used)
- Needed by *if* and *while*
- Comparison of elements = /= < > <= >=
- predefined operators:
- *Not and or xor and then or else*

if B = True then if B then

if B = False then if not B then

Logical operators, short circuit control forms

- Lazy evaluation: and then, or else
- ***If the value can be determined from the first argument then the evaluation stops***

```
while (I <= N) and then (i mod 2 = 0) loop
do_something;
end loop;
```

```
if A>B and then F(A,B) then
```

- Eager evaluation: and, or
- ***Evaluates both operators***

Integer

- The set of integer values:
- **…, -3, -2, -1, 0, 1, 2, 3, …**
- Predefined operators
- ***+A -A A+B A-B A*B A/B***
- ***A rem B, A mod B, abs A, A**B***
- Integer division truncates towards zero
- Exponentiation raises the first to the power of second (must be integer, positive integer or float, integer)

Differences between rem and mod

- **A B A/B A rem B A mod B**
- **12 5 2 2 2**
- **-12 5 -2 -2 3**
- **12 -5 -2 2 -3**
- **-12 -5 2 -2 -2**
- A=(A/B)*B +(A rem B) takes the sign of A
- A=B*N +(A mod B) takes the sign of B (N is integer)

Float

- *Predefined operators*
- *+X -X X+Y X-Y X*Y X/Y X**Y*
- Exponentiation: Y integer

Mixed values

- Not allowed:

I: Integer := 3;

F: Float := 3.14;

I := F + 1; F := I - 1.3; -- error

- Explicit conversion is needed:

I := Integer(F) + 1; F := Float(I) - 1.3;

Integer(F) will be truncated 1.4 – 1, 1.6 -- 2

1.5 – 2, -1.5 -- 2

Assignment

- Assigns an expression to a variable

I := 5;

- The type of the value and variable must be the same

I := True; -- error

- Subtypes are checked in runtime
- The assigment is not an operator an d it is not an expression (can not be overloaded)
- There is no simultaneous assignment

The empty statement

```
procedure Nothing is
begin
null;
end;
```

- Empty begin-end can not be written
- Used in tasks
- The keyword is multiply used

Control structures

```
if Boolean expression
then
   statements
elsif Boolean expression
then
   statements
else
   statements
end if;
```

Swap

```
if A>B then
Temp := A;
A := B;
B := Temp;
end if;

if A>B then
A := A - B;
else
B := B - A;
end if;
```

Undefined else

```
if (a>0)
    if (b>0)
        c = 1;
          else
c = 0;
```

- Not well defined in: C++, Java, Pascal etc.
- In Ada an if is closed, so we know the where is the else included

Example for if

```
with Ada.Text_IO; use  Ada.Text_IO; …
type Degrees is new Float range -273.15 .. Float'Last;
Temperature : Degrees;
…
if Temperature >= 40.0 then
    Put_Line ("It's extremely hot");
elsif Temperature >= 20.0 then
    Put_Line ("It's warm");
elsif Temperature >= 0.0 then
    Put_Line ("It's cold");
else
    Put_Line ("It's freezing");
end if;
```

Case structure

```
case  X  Is
   when 1 =>
      Walk_The_Dog;
   when 5 =>
      Eat_The_Lunch;
   when 6 | 10 =>
      Sell_All_the_Products;
   when others =>
      Do_as_usuall;
end case;
-- The subtype of X must be a discrete type, i.e. an
enumeration or integer type.
```

Case example

```
case (X mod 20) + 1 is
when 1..3 | 5 | 7 | 11 | 13 | 17 | 19 =>
    F := True;
when others =>
    F := False;
end case;
```

Loops

```
while Boolean expression loop
   statements;
end loop;
```

Example:

```
 while X <= 5 loop
      X := Calculate_Something;
 end loop;
```

example

```
with Ada.Integer_Text_IO;
procedure ten is
I: Positive := 1;
begin
while I <=10 loop
    Ada.Integer_Text_IO.Put( I );
    I := I + 1;
end loop;
end ten;
```

Loops

```
for variable in range loop
   statements;
end loop;
```

```
s := 0;
for i in 1..10 loop
     s := s + i;
end loop;
```

```
 for I in X'Range loop
      X (I) := I;
   end loop;
```

Example

```
with Ada.Integer_Text_IO;
procedure ten is
begin
for I in 1..10 loop
Ada.Integer_Text_IO.Put( I );
end loop;
end ten;
```

- The step can be only one,
- I is local variable an should not be declared, can not be changed in for

Reverse example

```
for I in reverse 1..10 loop
      Ada.Integer_Text_IO.Put( I );
end loop;
```

Endless loop

```
loop
      Do_Something;
end loop;
```

Example:

```
 loop
      X := Calculate_Something;
      exit when X > 10;
   end loop;
```

Example

```
loop
Get(Ch);
...
exit when Ch = ‘q’;
...
end loop
```