Dipl. Ing. Mario Blunk

Buchfinkenweg 3
99097 Erfurt / Germany

Phone + 49 (0) 361 6022 5184

Email info@blunk-electronic.de

Web www.blunk-electronic.de

Doc. Vers. 2018-11-19

About Ada

- Programming Language (procedural & object orientated)
- standards *Ada 83, Ada 95, Ada 2005, Ada 2012*
- mission/safety critical applications (aerospace, medical, transportation, financial, military, …)
- **Use it for engineering and commercial !**

- *Ada* is not old !
- *Ada* is not difficult to learn !

For Managers: Why Ada ?

https://en.wikibooks.org/wiki/Ada_Programming :

„… Consequences of these qualities are superior reliability, reusability and maintainability. For example, compared to programs written in C, programs written in Ada 83 contain "70% fewer internal fixes and 90% fewer bugs", and cost half as much to develop in the first place[3] Ada shines even more in software maintenance, which often accounts for about 80% of the total cost of development. ...“

Be courageous !

For Engineers: Why Ada ?

- ***Ada is beautiful !***
- strong and safe **type** system (avoids mixing miles with kilometers, Euros with Rubles, ...)
- **certified compiler** (gnat) detects errors before they become bugs
- reliability – reusability - maintainability
- **A complex world needs a complex language !**

Ready for something new ?

About this Course

- **THIS IS FOR BEGINNERS !**
- **We keep things simple !**
- We start with frequently used constructs.
- We learn by many **examples**.
- We focus on **small** command line applications.
- We use very simple *Makefiles.*
- exercises, Q & A

Please find full examples at https://github.com/Blunk-electronic/ada_training

Contents

1. setting up the environment (*gcc, gnat & makefile*)
2. basic constructs (output, control structures, procedures, functions, operations ...)
3. the concept of strong typing (predefined types, subtypes, derived types, user specific types)
4. conversion between types

Please find full examples at https://github.com/Blunk-electronic/ada_training

Environment #1

Our toolkit for *Ada* :

1. *GCC*, the *GNU compiler collection* (provided in all *Linux*-Distros)
2. *GNAT*, the *GNU Ada compiler* (subset of *GCC*)
3. *Ada* Run Time Libraries (RTL)
4. A simple Makefile
5. A text editor like *Kate*
6. optionally *gprbuild*

Environment #2

```
student@notebook1:~/git/BEL/training/ada/src/hello> ls
hello.adb  Makefile

student@notebook1:~/git/BEL/training/ada/src/hello> make
gcc -c -gnat2012 hello.adb -I include
gnatbind -x hello.ali; gnatlink hello.ali

student@notebook1:~/git/BEL/training/ada/src/hello> ls
hello  hello.adb  hello.ali  hello.o  Makefile

student@notebook1:~/git/BEL/training/ada/src/hello> ./hello
hello world !
```

All on the provided laptop PC !

Please find full examples at https://github.com/Blunk-electronic/ada_training

Hello World ! #1

The file hello.adb :

```
with ada.text_io;
use ada.text_io;

procedure hello is

begin
   put_line ("hello world !");
end hello;
```

Hello World ! #2

The file hello.adb :

libraries/packages to include. Use-clause optional.

This is a statement.

```
with ada.text_io;
--use ada.text_io;

procedure hello is

begin

   ada.text_io.put_line ("hello world !");

end hello;
```

Print Numbers

The file hello_numbers.adb :

```
with ada.text_io; use ada.text_io;

procedure hello_numbers is

   number_1 : integer := 2005;
   string_1 : string (1..9) := "hello Ada";

begin
   put_line (string_1 & integer'image(number_1));

   number_1 := number_1 + 7;

   put_line (string_1 & integer'image(number_1));
end hello_numbers;
```

declarative section with assignment

assignment

Control Structures #1

```
with ada.text_io; use ada.text_io;

procedure control_for is

begin
   for number in 1..4 loop
      put_line ("step:" & integer'image(number));
   end loop;

end control_for;
```

Control Structures #2

```
with ada.text_io; use ada.text_io;

procedure control_while is

   number : integer := 1;

begin

   while number <= 16 loop
      put_line ("step:" & integer'image(number));
      number := number * 2;
   end loop;

end control_while;
```

initialize „number“ with 1

Control Structures #3

```
procedure control_if is

begin
   for number in 1..6 loop
      if (number rem 2) = 0 then
         put_line ("step:" & integer'image(number));
      end if;
   end loop;

end control_if;
```

Control Structures #4

```
procedure control_case is

begin
   for n in 1..6 loop
      case n is
         when 2 =>
            put_line ("step:" & integer'image(n));
         when 5 =>
            put_line ("step:" & integer'image(n));
         when others => -- mandatory
            null;
      end case;
   end loop;
end control_case;
```

Procedures #1

```
procedure procedure_simple is

   procedure output_text is
   begin
      put_line ("hello Ada !");
   end output_text;

begin

   output_text;

end procedure_simple;
```

procedure „output_text“

Here the procedue „output_text“ is called.

Procedures #2

```
procedure procedure_in is

   procedure output_text (text : in string) is
   begin
      put_line (text);
   end output_text;

begin

   output_text ("hello Ada !");

end procedure_in;
```

This is a procedure argument.

Procedures #3

```
procedure procedure_out is

   n : integer := 1974;

   procedure modify (i : in out integer) is
   begin
      i := 1980;
   end modify;

begin
   put_line ("before :" & integer'image(n));

   modify (n);

   put_line ("after  :" & integer'image(n));
end procedure_out;
```

procedure argument

Functions #1

```
procedure function_simple is

   n : integer := 1974;

   function add (in_a, in_b : integer) return integer is
      sum : integer;
   begin
      sum := in_a + in_b;
      return sum;
   end add;

begin
   put_line ("before :" & integer'image(n));

   n := add (in_a => n, in_b => 10);

   put_line ("after  :" & integer'image(n));
end function_simple;
```

function return

function arguments

Operations #1

```
procedure operations is

   t1 : string (1..5) := "hello";
   t2 : string (1..3) := "Ada";

   n1 : integer := 10;
   n2 : integer := 20;

begin

   put_line (t1 & '_' & t2 & " !!");
   put_line ("The sum is" & integer'image(n2 + n1));
   put_line ("The difference is " & integer'image(n2 - n1));
   put_line ("The product is" & integer'image(n2 * n1));
   put_line ("The quotient is" & integer'image(n2 / n1));

end operations;
```

Operations #2

```
with ada.numerics.generic_elementary_functions;

procedure operations_math_1 is

   package functions is new
      ada.numerics.generic_elementary_functions (float);

   n : float := 16.0;

begin
   put_line ("n1 squared equals " & float'image(n ** 2));
   put_line ("The square root of n equals" &
      float'image(functions.sqrt(n)));
   put_line ("Log-base-2 of n equals" &
      float'image(functions.log(base => 2.0, x => n)));
   put_line ("Sine of n equals" &
      float'image(functions.sin(x => n, cycle => 360.0)));
end operations_math_1;
```

instantiation of package

Operations #3

```
with ada.text_io; use ada.text_io;
with interfaces;  use interfaces;

procedure operations_arithmetic_1 is

   byte : unsigned_8  := 2#00101101#; -- binary
   word : unsigned_16 := 16#1020#; -- hexadecimal
   doubleword : unsigned_32 := 16#10203040#;

begin

   put_line ("byte : " & unsigned_8'image(byte));
   byte := byte or 2#10000000#; -- set bit 7
   byte := byte and 16#FE#; -- clear bit 0
   word := word * 4; -- shift left two bits

end operations_arithmetic_1;
```

File Write #1

```
procedure file_write_1 is

   my_file : ada.text_io.file_type;

begin

   create   (
            file => my_file,
            mode => out_file,
            name => "text.txt"
         );

   put_line (my_file, "hello world !");
   close (my_file);

end file_write_1;
```

File Write #2

```
with interfaces; use interfaces;
with ada.sequential_io;

procedure file_write_2 is

   package byte_io is new ada.sequential_io(unsigned_8);
   --use byte_io;

   my_file : byte_io.file_type;
begin
   byte_io.create (    file => my_file,
                       mode => byte_io.out_file,
                       name => "some_bytes.bin");

   byte_io.write (my_file, 16#A1#);
   byte_io.write (my_file, 16#56#);
   byte_io.write (my_file, 16#F4#);
   byte_io.close (my_file);
end file_write_2;
```

instantiation of package

File Read #1

```
procedure file_read_1 is
   my_file : ada.text_io.file_type;
begin
   open  (
            file => my_file,
            mode => in_file,
            name => "text.txt"
         );
   set_input(my_file);

   while not end_of_file loop
      put_line (get_line);
   end loop;

   close (my_file);
end file_read_1;
```

File Read #2

```
procedure file_read_2 is
   package byte_io is new ada.sequential_io(unsigned_8);
   my_file : byte_io.file_type;
   byte  : unsigned_8;

begin
   byte_io.open (    file => my_file,
                     mode => byte_io.in_file,
                     name => "some_bytes.bin" );

   for b in 1..3 loop
      byte_io.read (my_file, byte);
      put_line(unsigned_8'image(byte));
   end loop;

   byte_io.close (my_file);
end file_read_2;
```

Types #1

```
procedure types_1 is

   text  : string (1..3) := "Ada";
   i     : integer   := -5;   -- -4,-1,0,1,5,107
   n     : natural   := 0;    -- 0,2,55,107
   p     : positive  := 4;    -- 1,2,55,107
   f     : float     := 4.5E-04; --4.5, -7.561
   b     : boolean   := false;

begin
   put_line (text);
   put_line (integer'image(i));
   put_line (natural'image(n));
   put_line (positive'image(p));
   put_line (float'image(f));
   put_line (boolean'image(b));
end types_1;
```

Types #2

```
procedure types_2 is

   i      : integer   := -5;
   n      : natural   := 0;
   f      : float     := 4.5;

begin

   i := i + n; -- compiles

   n := n + i; -- compiles with warning,
               -- but raises constraint error at run time

   f := f + i; -- does not compile

end types_2;
```

Types #3

```
procedure types_3 is

   c      : character := 'A';
   d      : duration  := 1.0; -- seconds

begin
   for i in 1..5 loop

      put(character'image(c) & " ");

      delay d;

   end loop;
   new_line;

end types_3;
```

Subtypes #1

```
procedure types_subtypes_1 is

   subtype my_integer_type is integer range -5..+5;
   i   : my_integer_type;

begin

   i := -5;

   i := i - 1; -- compiles with warning. raises
               -- constraint error at run time

end types_subtypes_1;
```

Derived Types #1

```
procedure types_derived_1 is
   i : integer := 2;

   -- This is a new type my_own_float.
   -- It is not compatible with type float,
   -- but inherits primitive operations defined
   -- for type float:
   type my_own_float is new float range 1.5 .. 5.0;
   mf : my_own_float := 2.0;

begin

   mf := mf * 2.0;
   put_line (my_own_float'image (mf));

    -- mf := mf + i; -- does not compile
end types_derived_1;
```

Derived Types #2

```
procedure type_conversion_1 is

   type kilometers is new float range 0.0 .. 10.0;
   type miles is new float range 0.0 .. 10.0;

   k : kilometers;
   m : miles := 2.0;

begin
   -- k := 1.61 * m; -- does not compile

   k := 1.61 * kilometers(m);

   put_line ("kilometers :" & kilometers'image (k));

end type_conversion_1;
```

User Specific Types #1a

```
procedure types_new_1 is

   -- These are new created types which are
   -- not compatible even if they have the same range:
   type parking_lots is range 1..20;
   type cars is range 1..20;

   p : parking_lots := 11;
   c : cars := 3;

begin
   p := p - 1;
   put_line ("parking lots:" & parking_lots'image (p));

   -- c := c + p; -- does not compile
   put_line ("cars:" & cars'image(c));
end types_new_1;
```

User Specific Types #1b

```
procedure type_conversion_2 is

   type parking_lots is range 1..20;
   type cars is range 1..20;

   p : parking_lots := 20;
   c : cars := 3;

begin

   -- p := p - c; -- does not compile

   p := p - parking_lots(c);

   put_line ("parking lots left:" & parking_lots'image (p));

end type_conversion_2;
```

User Specific Types #2

```
procedure types_new_1b is

   -- This is a new created types with
   -- its subtype:
   type parking_lots is range 1..20;
   subtype cars is parking_lots range 1..20;

   c : cars := 19;

begin
   c := c + 1;
   put_line ("cars:" & cars'image (c));

   c := c + 1;  -- causes a warning and
                -- constraint error at run time

end types_new_1b;
```

User Specific Types #3

```
procedure types_new_float is

   type kilogramms is digits 4 range 0.0 .. 11.0;
   mass : kilogramms := 0.2;

begin
   -- mass := 11.4;     -- Causes a warning at compile and
                        -- a constraint error at run time.

   mass := 10.035;  -- rounding takes place

   put_line ("the apple weights :" &
                kilogramms'image (mass) & " kg");

   mass := mass / 2.0; -- discards the last digit (5).
   put_line ("the apple half weights :" &
             kilogramms'image (mass) & " kg");
end types_new_float;
```

User Specific Types #3a

```
procedure types_float is

   f0 : float := 5.5;   -- accuracy machine depended
                        -- you get any accuracy
                        -- probably too much accuracy
   -- well defined accuracy
   type float_1 is digits 2;
   f1 : float_1 := 5.5;

   -- well defined range
   type float_2 is digits 2 range 0.0 .. 2.0;
   f2 : float_2 := 1.1;

begin
   put_line (float'image (f0));      -- 5.50000E+00
   put_line (float_1'image (f1));    -- 5.5E+00
   put_line (float_2'image (f2));    -- 1.1E+00

end types_float;
```

User Specific Types #4

```
procedure types_fixed_point_1 is

    step_width : constant := 0.01; -- real type
    type type_angle is delta step_width range 0.0 .. 359.99;
    for type_angle'small use step_width; -- constant accuracy

    angle : type_angle;

begin
    angle := type_angle'last; -- 359.99
    angle := type_angle'first; -- 0.00
    put_line ("step width :" & type_angle'image (step_width));

    for i in 1..10 loop

       angle := angle + type_angle'small;
       put_line ("angle :" & type_angle'image (angle));

    end loop;

end types_fixed_point_1;
```

User Specific Types #4

```
procedure types_new_2 is

   -- These are new created types which are
   -- not compatible:
   type musicians_classic is (brahms, bach, mozart);
   type musicians_techno is (marusha, vaeth, kalkbrenner);

   mc : musicians_classic := bach;
   mt : musicians_techno := marusha;

begin
   mc := brahms;
   put_line(musicians_classic'image(mc));

   mt := mozart; -- does not compile
end types_new_2;
```

User Specific Types #6

```
procedure types_new_3 is

   type colors is (red, green, blue);

begin
   put_line ( colors'image (green) );

   for c in 0..colors'pos( colors'last ) loop

      put_line( colors'image( colors'val(c) ) & " ");

   end loop;

end types_new_3;
```

Summary on Conversion

Explicit type conversion is legal between:

- any **numeric** types (natural, positive, integer, float,…)
- any **derived** types of the same type (miles, kilometers, …)
- **subtypes** of the same type (parking_lots, cars …)
- **nowhere else**

Constrained Arrays

```
procedure arrays_1 is

   -- define the array type
   type array_of_integers is array (positive range 1..5)
          of integer;

   -- instantiate an array and initalize all members
   -- with zero value
   a : array_of_integers := (others => 0);

begin
   a(2) := 4; -- assign member 2 the value 4

   -- display all members of array a
   for i in array_of_integers'first .. array_of_integers'last
      loop
         put_line ("member" & positive'image(i) & " :" &
                                          integer'image(a(i)));
      end loop;
end arrays_1;
```

Unconstrained Arrays

```
procedure arrays_2 is

   -- define the array type
   type unconstrained_array is array (natural range <>)
                                                 of integer;
   subtype constrained_array is unconstrained_array (1..5);

   -- instantiate an array and initalize all members
   -- with zero value
   a : constrained_array := (others => 0);

begin
   a(2) := 4; -- assign member 2 the value 4

   -- display all members of array a
   for i in constrained_array'first .. constrained_array'last
      loop
         put_line ("member" & positive'image(i) & " :" &
                                          integer'image(a(i)));
      end loop;
end arrays_2;
```

Records #1

```
procedure records_1 is
   type type_color is (red, green, yellow);

   type type_apple is
      record
         color    : type_color;
         weight   : float;
         rotten   : boolean := false;
      end record;

   apple : type_apple; -- instantiate an apple
begin
   apple.color := yellow;
   apple.weight := 230.4; -- gramms
   put_line ("the apple has color " &
         type_color'image (apple.color));

end records_1;
```

Records #2

```
procedure records_2 is

   type type_color is (red, green, yellow);

   -- define a limited apple
   type type_apple is limited
      record
         color     : type_color;
         weight    : float;
         rotten    : boolean := false;
      end record;

   -- instantiate two apples
   apple_a, apple_b : type_apple;

begin
   apple_a.rotten := true;

   -- make a copy of apple_a
   -- apple_b := apple_a; -- does not compile

end records_2;
```

References

(1)http://www.adaic.org

(2)https://en.wikibooks.org/wiki/Ada_Programming

(3)https://github.com/Blunk-electronic/ada_training

(4)http://www.blunk-electronic.de/ada.html

(5)Barnes, John; Programming in Ada 20xx;

Thanks for your attention !