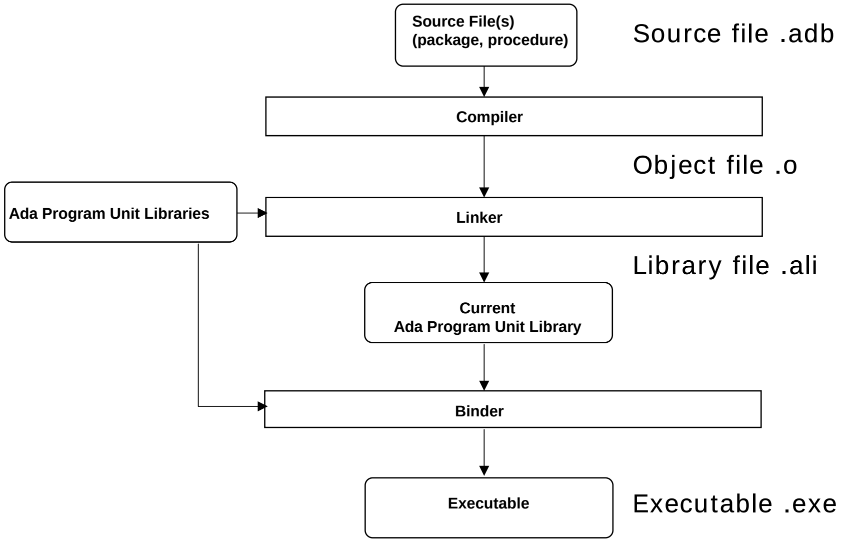
Unified Ada Tutorial

Jayakanth Srinivasan

Lean Aerospace Initiative/ ESL

Compilation Process



Source File(s) (package, procedure) — Source file .adb

Compiler

Object file .o

Ada Program Unit Libraries

Linker

Library file .ali

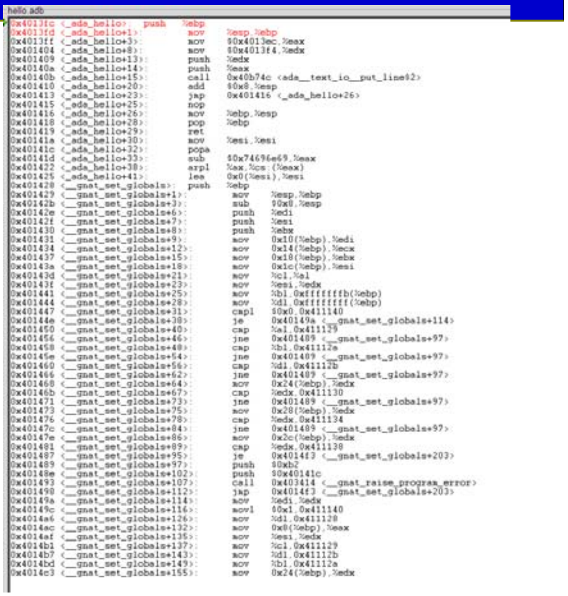
Current Ada Program Unit Library

Binder

Executable — Executable .exe

A Simple Ada Program

```
with Ada.Text_Io;
use Ada.Text_Io ;

procedure Hello is
begin
   Put("Hello");
end Hello;
```



hello.adb

```
0x4013fc <_ada_hello>:    push   %ebp
0x4013fd <_ada_hello+1>:         mov    %esp,%ebp
0x4013ff <_ada_hello+3>:         mov    $0x4013ec,%eax
0x401404 <_ada_hello+8>:         mov    $0x4013f4,%edx
0x401409 <_ada_hello+13>:        push   %edx
0x40140a <_ada_hello+14>:        push   %eax
0x40140b <_ada_hello+15>:        call   0x40b74c <ada__text_io__put_line$2>
0x401410 <_ada_hello+20>:        add    $0x8,%esp
0x401413 <_ada_hello+23>:        jmp    0x401416 <_ada_hello+26>
0x401415 <_ada_hello+25>:        nop
0x401416 <_ada_hello+26>:        mov    %ebp,%esp
0x401418 <_ada_hello+28>:        pop    %ebp
0x401419 <_ada_hello+29>:        ret
0x40141a <_ada_hello+30>:        mov    %esi,%esi
0x40141c <_ada_hello+32>:        popa
0x40141d <_ada_hello+33>:        sub    $0x74696e69,%eax
0x401422 <_ada_hello+38>:        arpl   %ax,%cs:(%eax)
0x401425 <_ada_hello+41>:        lea    0x0(%esi),%esi
0x401428 <__gnat_set_globals>:   push   %ebp
0x401429 <__gnat_set_globals+1>:          mov    %esp,%ebp
0x40142b <__gnat_set_globals+3>:          sub    $0x8,%esp
0x40142e <__gnat_set_globals+6>:          push   %edi
0x40142f <__gnat_set_globals+7>:          push   %esi
0x401430 <__gnat_set_globals+8>:          push   %ebx
0x401431 <__gnat_set_globals+9>:          mov    0x10(%ebp),%edi
0x401434 <__gnat_set_globals+12>:         mov    0x14(%ebp),%ecx
0x401437 <__gnat_set_globals+15>:         mov    0x18(%ebp),%ebx
0x40143a <__gnat_set_globals+18>:         mov    0x1c(%ebp),%esi
0x40143d <__gnat_set_globals+21>:         mov    %cl,%al
0x40143f <__gnat_set_globals+23>:         mov    %esi,%edx
0x401441 <__gnat_set_globals+25>:         mov    %bl,0xfffffffb(%ebp)
0x401444 <__gnat_set_globals+28>:         mov    %dl,0xffffffff(%ebp)
0x401447 <__gnat_set_globals+31>:         cmpl   $0x0,0x411140
0x40144e <__gnat_set_globals+38>:         je     0x40149a <__gnat_set_globals+114>
0x401450 <__gnat_set_globals+40>:         cmp    %al,0x411129
0x401456 <__gnat_set_globals+46>:         jne    0x401489 <__gnat_set_globals+97>
0x401458 <__gnat_set_globals+48>:         cmp    %bl,0x41112a
0x40145e <__gnat_set_globals+54>:         jne    0x401489 <__gnat_set_globals+97>
0x401460 <__gnat_set_globals+56>:         cmp    %dl,0x41112b
0x401466 <__gnat_set_globals+62>:         jne    0x401489 <__gnat_set_globals+97>
0x401468 <__gnat_set_globals+64>:         mov    0x24(%ebp),%edx
0x40146b <__gnat_set_globals+67>:         cmp    %edx,0x411130
0x401471 <__gnat_set_globals+73>:         jne    0x401489 <__gnat_set_globals+97>
0x401473 <__gnat_set_globals+75>:         mov    0x28(%ebp),%edx
0x401476 <__gnat_set_globals+78>:         cmp    %edx,0x411134
0x40147c <__gnat_set_globals+84>:         jne    0x401489 <__gnat_set_globals+97>
0x40147e <__gnat_set_globals+86>:         mov    0x2c(%ebp),%edx
0x401481 <__gnat_set_globals+89>:         cmp    %edx,0x411138
0x401487 <__gnat_set_globals+95>:         je     0x4014f3 <__gnat_set_globals+203>
0x401489 <__gnat_set_globals+97>:         push   $0xb2
0x40148e <__gnat_set_globals+102>:        push   $0x40141c
0x401493 <__gnat_set_globals+107>:        call   0x403414 <__gnat_raise_program_error>
0x401498 <__gnat_set_globals+112>:        jmp    0x4014f3 <__gnat_set_globals+203>
0x40149a <__gnat_set_globals+114>:        mov    %edi,%edx
0x40149c <__gnat_set_globals+116>:        movl   $0x1,0x411140
0x4014a6 <__gnat_set_globals+126>:        mov    %dl,0x411128
0x4014ac <__gnat_set_globals+132>:        mov    0x8(%ebp),%eax
0x4014af <__gnat_set_globals+135>:        mov    %esi,%edx
0x4014b1 <__gnat_set_globals+137>:        mov    %cl,0x411129
0x4014b7 <__gnat_set_globals+143>:        mov    %dl,0x41112b
0x4014bd <__gnat_set_globals+149>:        mov    %bl,0x41112a
0x4014c3 <__gnat_set_globals+155>:        mov    0x24(%ebp),%edx
```

Operator Precedence

Precedence *Operators* *Notes*

Highest `**` `not` `abs`

`*` `/` `mod` `rem` Multiply

operators

`+` `-` Unary operators

`+` `-` `&` Binary operators

`=` `/=` `<` `<=` `>` `>=`

Relational operators

`in` `not in` Membership operators

`and` `or` `xor` Logical

operators

Lowest `and then` `or else` Short-circuit

operators

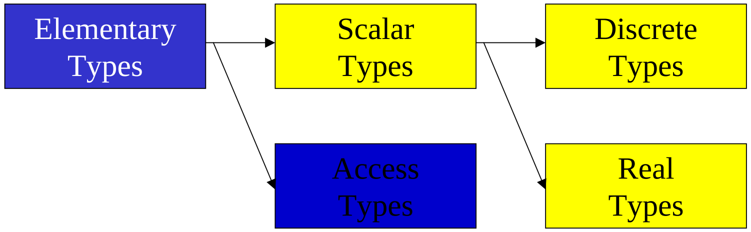
Sequential Control Statements

- assignment
- null
- block
- return
- procedure calls

```
declare  -- vars local to block
  Local_1 : Integer;
begin  -- code of the block
  Local_1 := 2;
  Value := Value / Local_1;
end;  -- end of the block
```

Type Classification

- **Elementary** Types : Values are logically indivisible
- **Composite** Types : Values composed from components



Elementary Types

Scalar Types

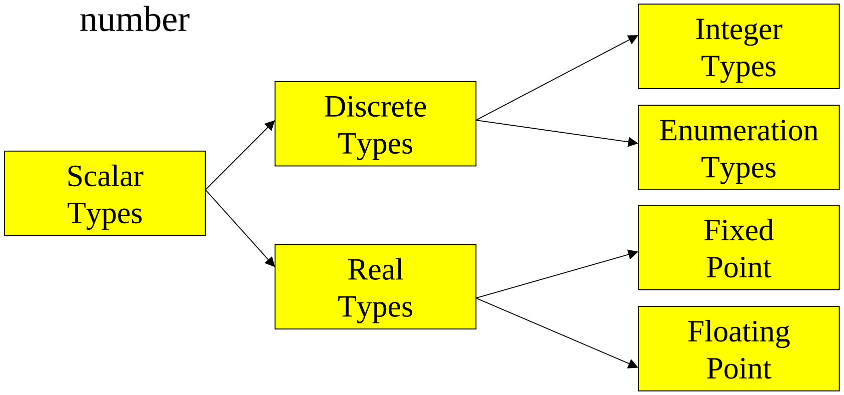
Discrete Types

Access Types

Real Types

Scalar Types

- Ordered → relational operators are defined
- Each value of a discrete type has a position number



Scalar Types

Discrete Types

Real Types

Integer Types

Enumeration Types

Fixed Point

Floating Point

Attributes of Scalar Types

- S’First denotes the lower bound of the range of S.
- S’Last denotes the upper bound of the range of S
- S’Range is equivalent to the range S’First .. S’Last
- S’Min returns lower of two elements
- S’Max returns higher of two elements
- S’Value accepts a string and returns the value in the type
- S’Image converts the value into a string
- S’Pred and S’Succ

Operations on Scalar Types

- S’Min returns lower of two elements
- S’Max returns higher of two elements
- S’Value accepts a string and returns the value in the type
- S’Image converts the value into a string
- S’Pred and S’Succ – behavior depends on the scalar type
  - S’Pred (Integer) : returns (Integer -1)
  - S’Succ (Integer) : returns (Integer + 1)

Subtypes

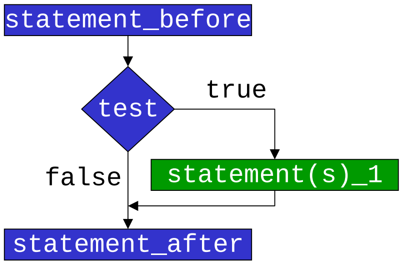
```
subtype Natural is Integer range 0..Integer’Last;

subtype Positive is Integer range 1..Integer’Last;

subtype NonNegativeFloat is Float range 0.0 .. Float’Last;


subtype SmallInt is Integer range -50..50;


subtype CapitalLetter is Character range ’A’..’Z’;
X, Y, Z      : SmallInt;
NextChar     : CapitalLetter;
Hours_Worked : NonNegFloat;


X := 25;
Y := 26;
Z := X + Y;
```

Enumeration IO

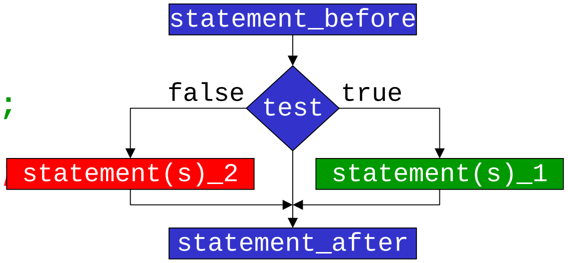
```
type Days is
  (Monday, Tuesday, Wednesday, Thursday, Friday,
   Saturday, Sunday);

package Day_IO is new Ada.Text_IO.Enumeration_IO(Enum=>Days);

if this_day in weekend_days then
   put(”Holliday!”);
end if;

Day_IO.Get(Item => Today);
Day_IO.Put(Item => Today, Width => 10);
```

Conditional Control Statements : if – then - else

```
if test then
     statement(s)_1;
end if;
```



```
if test then
     statement(s)_1;
else
     statement(s)_2
end if;
```



Conditional Control Statements : elsif

```
if test_1 then
    statement(s)_1;
elsif test_2 then
    statement(s)_2;
else
    statement(s)_3;
end if;
```

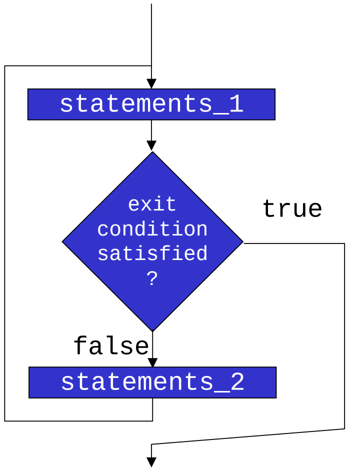
What does the flow chart look like?

Conditional Control Statements : case

```
case selector is
     when value_list_1 =>
        statement(s)_1;
     when value_list_2 =>
        statement(s)_2;
        …
     when others =>
        statement(s)_n;
  end case;
```

Iterative Control Statements: loop

```
▪ loop
    statements_1;
    exit when test;
    statements_2;
  end loop;
```


statements_1

exit condition satisfied ?

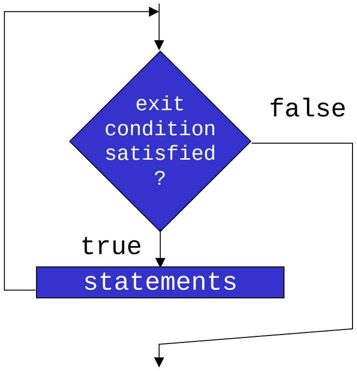
true

false

statements_2

Iterative Control Statements: while

```
▪ while condition loop
     statements;
  end loop;
```


exit condition satisfied ?

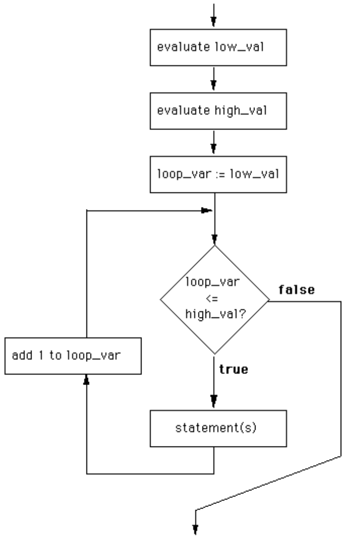
false

true

statements

Iterative Control Statements: for

```
for loop_var in
  low_val .. high_val
loop
     statement(s);
end loop;
```



Functions

```
<function_header>
   <local_variables_and_constants>
begin
   <function_body>
end <function_name>;
```

- <function_header>
  - contains the function name and parameters.
- <local_variables>
  - variables used in the function (but nowhere else).
- <function_body>
  - the code the function executes.
- <function_name>
  - the name of the function.

Function Header

```
function <function name> (
 <formal parameter name> : <data type>;
 <formal parameter name> : <data type>;
. . . ) return <data type> is
```

```
function Fact (N : Integer) return Integer is
begin
     if N <= 1 then
            return 1;
     else
            return N * Fact (N-1);
     end if;
end Fact;
```

Procedures

```
<procedure_header>
   <local_variables_and_constants>
begin
   <procedure_body>
end <procedure_name>;
```

- <procedure_header>
  - contains the procedure name and parameters.
- <local_variables>
  - variables used in the procedure (but nowhere else).
- <procedure_body>
  - the code the procedure executes.
- <procedure_name>
  - the name of the procedure.

Procedure Header

No Information Flow (No Parameters)

```
procedure <procedure name> is
```

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Hello is
begin
     Put_Line (“Hello”);
end Hello;
```

With Information Flow (With Parameters)

```
procedure <procedure name> (
   <formal parameter name> : <mode> <data type>;
   <formal parameter name> : <mode> <data type>;
   . . . ) is
```

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Increment (X : in out Integer;
                     Y : in out float) is
begin
   x:= x + 1; y := y + 1.4;
end Hello;
```

Procedure Calls

No Parameters

```
<procedure name>;
```

```
with Hello;
procedure Main is
begin
   Hello;
end Main;
```

With Parameters

```
<procedure name> (
   <formal parameter name> => <actual parameter name>;
   <formal parameter name> => <actual parameter name>,
  . . . );
```

```
with Increment;
procedure Main is
my_x : integer := 1;
my_y : float   := 2.0;
begin
   Increment(my_x, my_y);
end Main;
```

Arrays

```
type int_8_array  is array (1 .. 8) of Integer;
type CUBE6 is array (1..6, 1..6, 1..6) of Integer;
```

- Access elements using Indices
  - Single Dimension arrays A(I)
  - Two dimensional arrays A(I,J)
  - N dimensional array A(i_1, i_2,..,i_n)
- Loops can be used to access elements.

```
for I in 1 .. N loop
   for J in 1 .. N loop
      Put (B(I,J));
   end loop;
end loop;
```

Records

```
type My_Type_Record is
record
   my_boolean : Boolean;
   my_integer : Integer;
   my_real    : Float;
end record;
```

```
type My_Other_Type_Record is
record
   my_integer : Integer;
   my_real    : Float;
   my_boolean : Boolean;
end record;
```

```
Rec1 : my_type_record;
Rec2 : my_other_type_record;
```

- **`Rec2 := Rec1;`**
- **`Rec1.my_boolean := Rec2.my_boolean;`**
  `Rec1.my_integer := Rec2.my_integer;`
  `Rec1.my_Real    := Rec2.my_real;`

Arrays ADT

- Storage
- Retrieval
- Insertion
- Deletion
- Search
- Sort

```
 My_Array_Max : constant Integer:=10;
My_Array_Min : constant Integer:=1;
type My_Integer_Array is array
 (My_Array_Min .. My_Array_Max) of
 Integer
```

Arrays: Linear Search

```
Procedure Linear_Search (  Input_Array,
                           Number_to_Search)
Begin
     Set Return_Index to -1
     For I:= 1 to size_of_array do
if (Input_Array(I) = Number_to_Search)
Return_Index := I;
Exit Loop
            endif
     end loop
     return Return_Index;
End Linear_Search;
```

Arrays: Binary Search

```
Procedure Binary_Search (    Input_Array,
                             Number_To_Search, Return_Index;
Begin
       Set Return_Index to –1;
       Set Current_Index to (Upper_Bound - Lower_Bound + 1) /2.
       Loop
                 if the lower_bound > upper_bound
                          Exit;
                 end if
if ( Input_Array(Current_Index) = Number_to_Search) then
                          Return_Index = Current_Index)
                          Exit;
                 end if
if ( Input_Array(Current_Index) > Number_to_Search) then
                          Lower_Bound = Current_Index +1
                 else
                          Upper_Bound = Current_Index – 1
                 end if
       end loop
end Binary_Search;
```

Arrays: Bubble Sort

```
  Procedure Bubble_Sort(Input_Output_Array)
  Begin
  for I in 1 .. My_Array_Max loop
                       for J in I+1 .. My_Array_Max loop
                       if (Input_Output_Array(I) <=
Input_Output_Array(J)) then
                              Temp := Input_Output_Array(I);
                              Input_Output_Array(I) :=
Input_Output_Array(J);
                              Input_Output__Array(J) := Temp;
                       end if;
                       end loop;
        end loop;
  end Bubble_Sort;
```

Package Specification

-
  ```
  package package_name is
     declarations

  private
     type definitions
  end package_name;
  ```

  public portion

  private portion

- Public:
  - What you need to know to use the package
- Private:
  - Implementation of data types

Package Body

- Implementation of the resources provided by the package
- The package is a "black box" to the user of the package.
- The package body is not visible to a package user.

```
package body package_name is

    declarations

end package_name;
```