

# Programming in Ada

## course

Review  
Block structure  
Functions, procedures  
Packages  
Examples



# The block structure

declare

    some declarations;

begin

    statements;

exception

    handlers;

end;



# Hierarchy

...

declare

...

begin

...

    declare ...

        begin ...

        end;

...

end;



# Module structures

```
procedure P(parameters: in out type) is  
    declarations;  
begin  
    statements;  
exception  
    handlers;  
end P;
```





# Functions

```
function F (parameters : in TypeIn) return TypeOut is  
    declarations;  
begin  
    statements;  
exception  
    handlers;  
end F;
```

# Subprograms and return

- We call the subprograms by their name

```
Ada.Text_IO.Put_Line("Some text");
```

```
Ada.Text_IO.Get(Ch);
```

- Ch a variable of type Character

```
Text_IO.New_Line;
```

- After return the subprogram is ended
- At functions, we must write the result after a return statement !

```
return X+Y;
```

# Function

function Factorial ( N: Natural ) return Positive is

Fact: Positive := 1;

begin

for I in 1..N loop

Fact := Fact \* I;

end loop;

return Fact;

end Factorial;



# Function – GCD example

```
function GCD ( A, B : Positive ) return Positive is
  X: Positive := A;
  Y: Natural := B;
  Tmp: Natural;
begin
  while Y /= 0 loop
    Tmp := X mod Y;
    X := Y;
    Y := Tmp;
  end loop;
  return X;
end GCD;
```



# Procedure

```
procedure Swap ( A, B: in out Integer ) is
    Temp: Integer := A;
begin
    A := B;
    B := Temp;
end Swap;
```

# Parameters

- **in** : transfers information from the caller

Caller -> called

- **out** : transfers the information computed in a procedure to the caller

Called -> caller

- **in out** :

- – transfers information to the procedure from the caller

- – transfers information from the procedure to the caller

Caller <--> called

Default is **in** !!!

**Out** or **in out** can be a left value

# Example

```
procedure Ex ( Vi: in Integer; Vo: out Integer;  
              Vio: in out Integer) is
```

```
begin
```

```
  Vio := Vi + Vo; -- error Vo can not be read
```

```
  Vi := Vio; -- error, Vi can not be written
```

```
  Vo := Vi; -- good
```

```
  Vo := 2*Vo+Vio; -- good, Vo already has a value
```

```
end Ex;
```



# Examples

- procedure Put ( Item: in Integer );
- procedure GCD ( A, B: in Positive; gd: out Positive);
- procedure Swap ( A, B: in out Integer );
  
- **in** can be only read, we can not assign values
- **out** parameter can be written
- **in out** parameter can be read and written

# Example

```
procedure eucl( A, B: in Positive; GCD, LCM: out Positive ) is
  X: Positive := A;
  Y: Positive := B;
begin
  while X /= Y loop
    if X > Y then X := X - Y; else Y := Y - X; end if;
  end loop;
  GCD := X;
  LCM := A * B / GCD ;
end eucl;
```

# Recursive function

```
function Factorial ( N: Natural ) return Positive is
begin
    if N > 1 then return N * Factorial (N-1);
    else return 1;
    end if;
end Factorial;
```



# Packages

- Logical entities that are connected
- Defines the set of variables, types
- More complex than a subprogram (function or procedure)
- Subprograms are executed
- Packages have components that can be used



# The structure of a package

package A is

...

end A;

package body A is

...

end A;

In a demo program has to be imported:

with A; use A;

# Package specification

- Can not contain implementation details
- Contains declarations of subprograms together with types and variables
- .ads

package Ada.Text\_IO is

...

procedure Put\_Line( Item: in String ) ;

...

end Ada.Text\_IO;



# Package implementation

- Contains the bodies of the subprograms
- .adb

package body Ada.Text\_IO is

...

procedure Put\_Line( Item: in String ) is

...

end Put\_Line;

...

end Ada.Text\_IO;

# Compiling

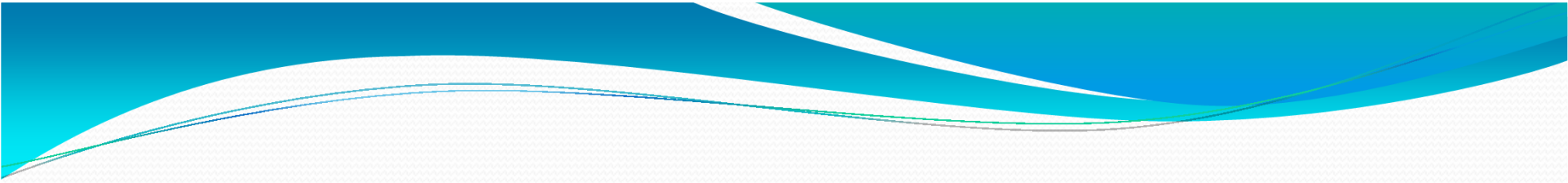
- The specification and the implementation can be compiled separately (2 units)
- A subprogram is only one compilation unit
- Why to write packages?
- Specifying interfaces
- Encouraging team work
- Parallel software development



# Packages in other languages

- C++: class, namespace
- Java: class/interface, package
- Modula-2, Clean: module
- The use statement: includes the predefined package
- It can be applied only with packages
- C++: using namespace
- Java: import





```
package math is
  function gcd ( A, B : Positive ) return Positive;
  function factorial( N: Natural ) return Positive
end math ; -- this is the math.ads file
```

```
package math is
  function gcd ( A, B : Positive ) return Positive is
  begin
    ...
  end gcd;

  function factorial( N: Natural ) return Positive is
  begin
    ....
  end factorial;
end math ; -- this is the math.adb file
-- in a main.adb program the package has to be imported:
-- with math; use math;
```