UPPSALA
UNIVERSITET

Lab 1

Real-Time Programming using Ada

Jakaria Abdullah

11 September 2018

UPPSALA UNIVERSITET

Lab Overview

Lab goals

- Basic understanding of Ada concepts
- Handling of time
- Inter-task communication

Lab preparation

- Check lab web page

http://www.it.uu.se/edu/course/homepage/realtid/2017-368/labs/lab1

- Go through **Ada quick start**

http://www.it.uu.se/edu/course/homepage/realtid/2017-368/labs/lab1/start_ada

UPPSALA UNIVERSITET

Ada: Some Facts

- Designed by/for US military, first compiler 1983
- Versions: 83, **95 (core)**, 2005, 2012
- Standard programming language for safety-critical systems
- Suitable for applications that require certification
- Most widely used in avionics, space, defense, railway
- Common applications: fly-by-wire system, railway traffic controller, emergency system, etc.
- Check out who uses Ada:

https://www.seas.gwu.edu/~mfeldman/ada-project-summary.html

UPPSALA UNIVERSITET

Ada: Language

Features:

- Imperative, object-oriented language
- Statically typed
- Case insensitive
- Concepts of task and time as language feature

Compiler:

- Commercial and free version available
- GNAT-Pro (commercial) from AdaCore
- We will use GNAT (part of GNU compiler collection)

UPPSALA UNIVERSITET

Ada: Hello World!

```
-- File : hello_world . adb
with Ada. Text_IO ; -- Use package Ada. Text_IO
use Ada . Text_IO ; -- Integrate its namespace

procedure hello_world is
      Message : constant String := " Hello World ";
begin
     Put_Line ( Message );
end hello_world;
```

Compile: **gnatmake hello_world**
Run: **./hello_world**
Output: Helloworld (only once!)

UPPSALA UNIVERSITET

Ada: Hello World!

UPPSALA
UNIVERSITET

Ada: Code Structure

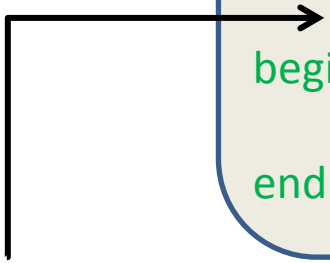
- Two types of files: specification (.ads) and implementation (.adb)
- Six types of program units: subprograms, packages, tasks, generics, protected units and protected entries
- A library unit is a separately compiled program unit, and is always a package, subprogram or generic unit
- Program units can be connected using three kinds of relationships: with clause, Parent/Child or Nesting
- A compilation unit is a library unit with or without context clauses
- Elaboration: before execution, all library units needed by the main program should be initialized

UPPSALA UNIVERSITET

Ada: Subunit Structure

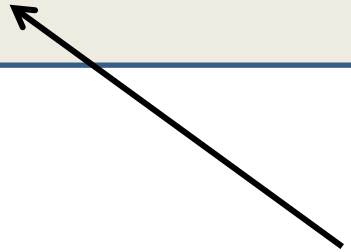
```
-- File : hello_world . adb
with Ada. Text_IO ; -- Use package Ada. Text_IO
use Ada . Text_IO ; -- Integrate its namespace

procedure hello_world is
    Message : constant String := " Hello World ";
begin
    Put_Line ( Message );
end hello_world;
```



Declaration or interface part

Start of body



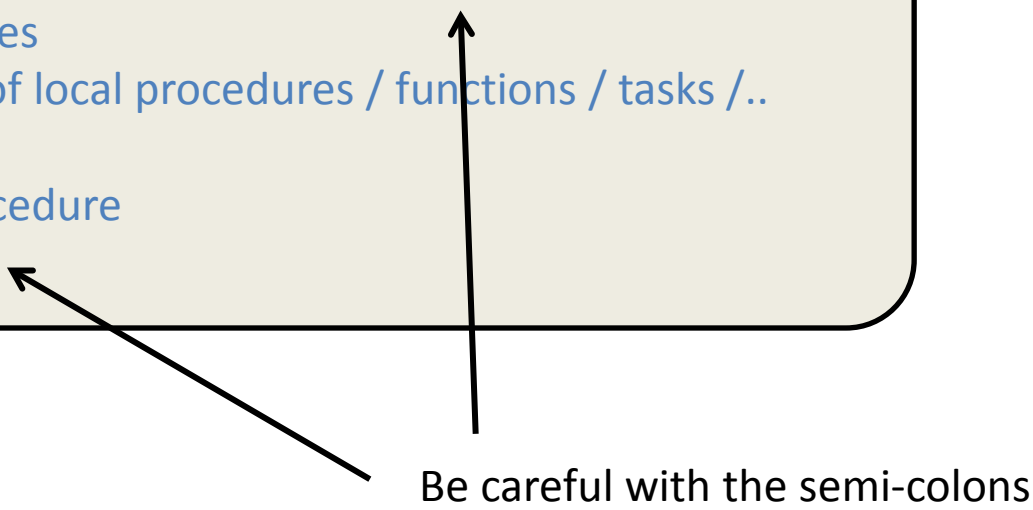
End of body

UPPSALA UNIVERSITET

Ada: Procedure

Procedure call is a statement and does not contain return statement

```
procedure Name_of_Proc ( var1 : in Integer ; var2 : out String )  is
        -- local variables
        -- definitions of local procedures / functions / tasks /..
begin
        -- code of procedure
end Name_of_Proc;
```



UPPSALA UNIVERSITET

Ada: Parameter modes

- Constant must be initialized, variable not
- Parameters have three modes indicating flow of data:
  in: constant/expression and constant within call (default)
  out: variable and modified value replace old after finish
  in out: variable, value expected before call

```
procedure Put_Line(F: in File_Type; Item: in String);
Put_Line(Error_File, "Execution error has occurred");
```

UPPSALA UNIVERSITET

Ada: Function

Function call is part of an expression and must return value

Only allows in parameters

```
function Name_of_Func (x1 , x2: Float ) return Float is
      -- local variables
      -- definitions of local procedures / functions / tasks /..
begin
      return x1 * x2;
end Name_of_Func;
```

UPPSALA UNIVERSITET

Ada: Statement

```
x := y;   -- Assign y to x
if (x = y) then
      null ;
else
      z := 0;
      x := 42;    -- Note : Several statements !
end if;
loop
      z := z + 1;
      exit when z > 100;   -- Works also with only " exit "
end loop;   -- Loops may be nested and while /for annotated
```

UPPSALA UNIVERSITET

Ada: Types

- Basic types: Boolean, Integer, Float, Character, String
- New types via type, mod, array, record
- A subtype renames a subset of values of a previously declared type, typically having a smaller range of values
- A new type can be derived from its parent type (only blueprint!)
- subtype values are legal values of the base type but derived types are not
- modular types uses "wrap-around" semantics

UPPSALA UNIVERSITET

Ada: Types

```
subtype Die is Integer range 1 .. 6;          -- One die
type Dice is array (0 .. 22) of Die;          -- 23 dies
type miniDie is new Integer range 1 .. 6;     -- derived type

type Complex is
    record
        r : Float ;    -- real component
        i : Float ;    -- imaginary component
    end record ;
d: Dice ;      -- Array access : d(1) := 3;

type Altitude_mod is mod 1000;    -- a modular type
Bar : Altitude_mod := 999;
Bar := Bar + 1;      -- Bar is now 0
```

UPPSALA UNIVERSITET

Ada: Type Conversion

- Explicit type conversions are allowed between
  - any two numeric types
  - any two subtypes of the same type
  - any two types derived from the same type
  - array types under conditions
- Types are checked at compile-time, while subtypes are checked in run-time
- Exception *Constraint Error* will be raised at run-time in case of range mismatch
- Run-time checking can be suppressed by pragma *Suppress*

UPPSALA
UNIVERSITET

Ada: Type Conversion

```
type Altitude is range 0 .. 100000;
subtype Low_Altitude is Altitude range Altitude'First .. 35000;
subtype High_Altitude is
Altitude range Low_Altitude'Last +1 .. Altitude'Last ;

Num : Integer := 1000;
Low : Low_Altitude := 1000;
High : High_Altitude := 50000;

High := High_Altitude ( Num + Integer ( Low));
-- run-time exception in conversion
```

UPPSALA UNIVERSITET

Ada: Exceptions

- Exception handlers are associated with a body, generally of a subprogram or a block
- The *try* implicitly covers all the statements in the body
- Four predefined exceptions:

  Constraint Error is raised for any run-time violation of the type system

  Program Error, which is raised in unusual program executions

  Storage Error, is raised if the program runs out of memory.

  Tasking Error, is raised for errors in inter-task communication.
- User defined exception can be raised using *raise* instruction

Ada: Exceptions

```
procedure Get_And_Process is
   procedure Process is
        I: Integer := Get(Q'Access );
   begin
        Put_Line ( Integer'Image (I));
   end Process ;

begin                    -- Get_And_Process body ;
    Process ;
exception
          when Underflow => Put(" Underflow handled in Get_And_Process ");
end Get_And_Process ;
```

UPPSALA UNIVERSITET

Ada: Text Output

- Output with line break: Ada.Text_IO.Put_Line("Hej!")
- Output without linebreak: Ada.Text_IO.Put("Hej!")
- String representation of Y of type X: X'Image(Y)
- Uses Image attribute of type X

```
use Ada.Text_IO;
i: Integer;   d: Duration;
begin
   i := 1 + 2;
   Put_Line ( Integer'Image (i));
   --print non - string type
   d := 10.0;
   Put(" Waiting time : ");
   Put_Line ( Duration'Image (d));
end Foo;
```

UPPSALA UNIVERSITET

Ada: Time

- Package: Ada.Calendar
- Type for *relative* time: Duration
- Type for *absolute* time: Time
- For getting present time: Clock (of type Time)

```
-- Assume : x: Duration ; t, t1 , t2: Time ;

delay 10.0;          -- Delays execution by 10s ( Float literal !)
delay x;             -- Use Duration type here

t := Clock ;                    -- Read present time
delay until t + 10.0;           -- Delays until t + 10 seconds
delay (t2 - t1);                -- Time difference is Duration
delay until t + x;              -- Time + Duration gives Time
```

UPPSALA UNIVERSITET

Ada: Time

- High resolution time: Ada.Real_Time

	Calender	Real_Time
Range of time	500 years	50 years
Range of interval	1 day	+- 1 hour
Precision	20 milisec	20 microsec

UPPSALA
UNIVERSITET

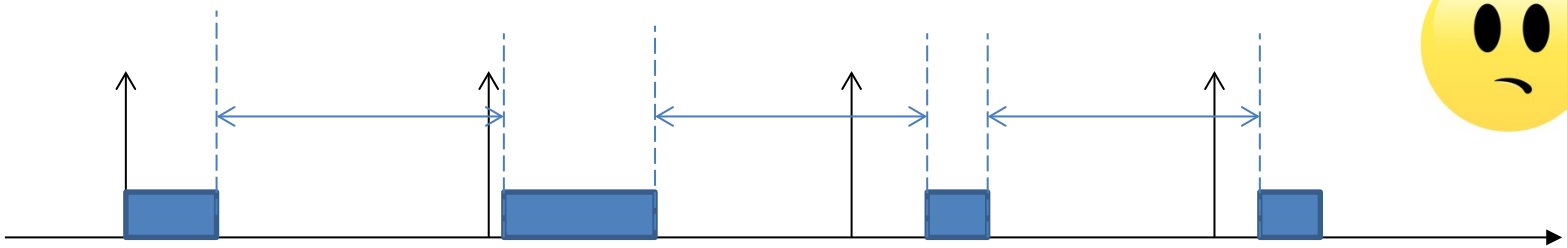
Ada: Delay

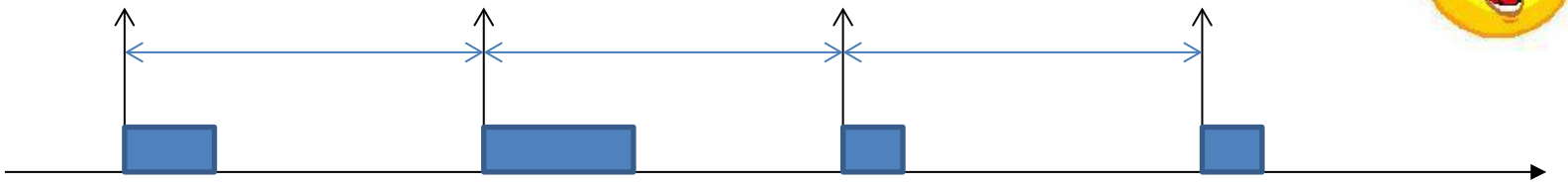
```
-- Assume : Use package Ada. Text_IO and Ada. Calendar
procedure drift is
        Message : constant String := "The time is";
        Period : Duration := 1.0;
        Start_Time : Time := Clock ;
begin
        loop
                Put( Message );
                delay Period ;
                Put_Line ( Duration ' Image ( Clock - Start_Time ));
        end loop ;
end drift ;
```

Delay = relative delay

UPPSALA UNIVERSITET

Ada: Issue with Delay





UPPSALA UNIVERSITET

Ada: Delay Until

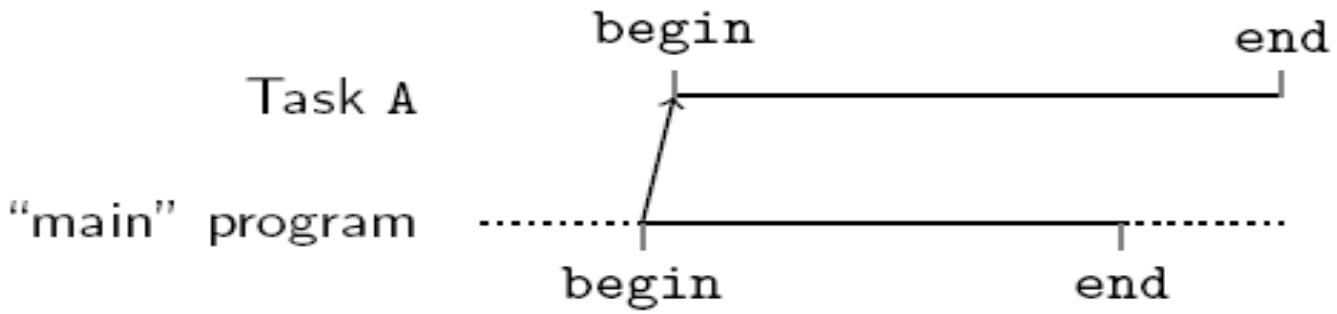
```
-- Assume : Use package Ada. Text_IO and Ada. Calendar
procedure avoid_drift is
        Message : constant String := "The time is";
        Period : Duration := 1.0;
        Start_Time : Time := Clock ;
        Next_Time : Time := Start_Time + Period ;
begin
       loop
            Put( Message );
            delay until Next_Time ;
            Next_Time := Next_Time + period ;
            Put_Line ( Duration ' Image ( Clock - Start_Time ));
       end loop ;
end avoid_drift ;
```

Delay Until = absolute delay

UPPSALA UNIVERSITET

Ada: Task

- Starts running as soon as they are created
  - In scope of main procedure: At program start!
  - Otherwise: When scope is called
- Ends when control flow reaches end
- Program does not end before all tasks are done



UPPSALA
UNIVERSITET

Ada: Task

```
-- Declaring a task
task My_Task is
        entry Init ;
        entry Get( Item : out DataItem );
        entry Put( Item : in DataItem );
end My_Task ;

 -- Defining a task body
task body My_Task is
-- Declarations needed by task code go here
begin
-- Task code goes here ( see next slide !)
end My_Task ;
```

UPPSALA
UNIVERSITET

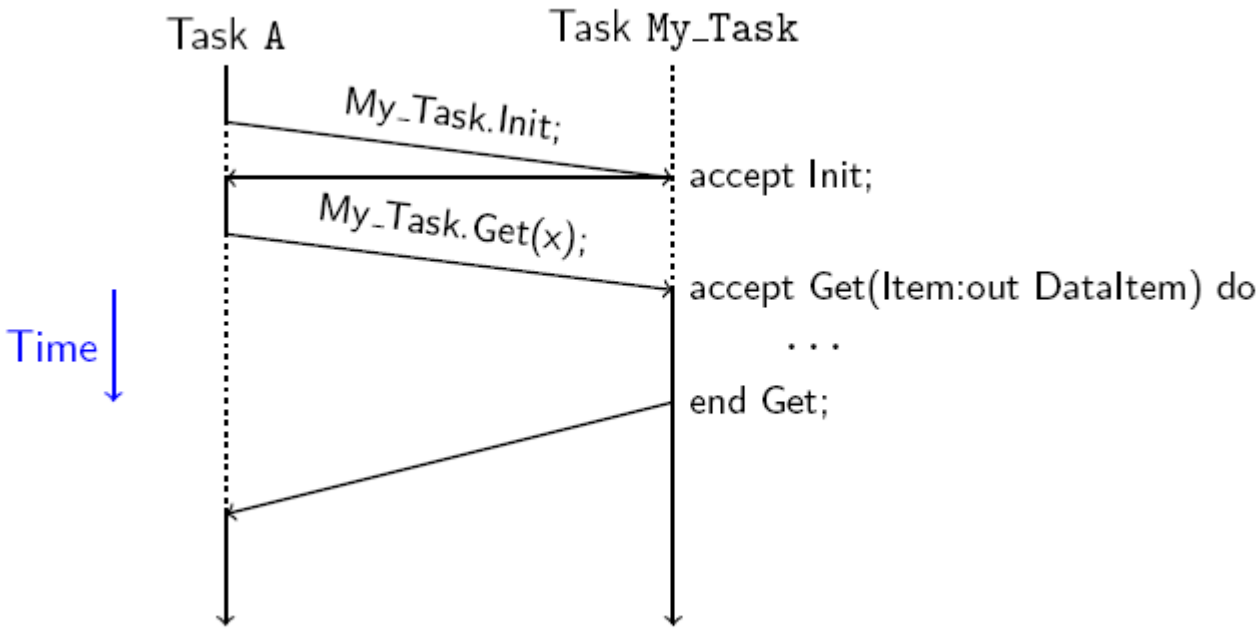
Ada: Task

```
accept Init ;      -- Wait for a call to 'Init ' ( using My_Task . Init )

accept Get( Item : out DataItem ) do      -- same , with parameters
. . .
end Get ;
```


UPPSALA
UNIVERSITET

Ada: Task


Task A
Task My_Task
My_Task.Init;
accept Init;
My_Task.Get(x);
accept Get(Item:out DataItem) do
...
end Get;
Time

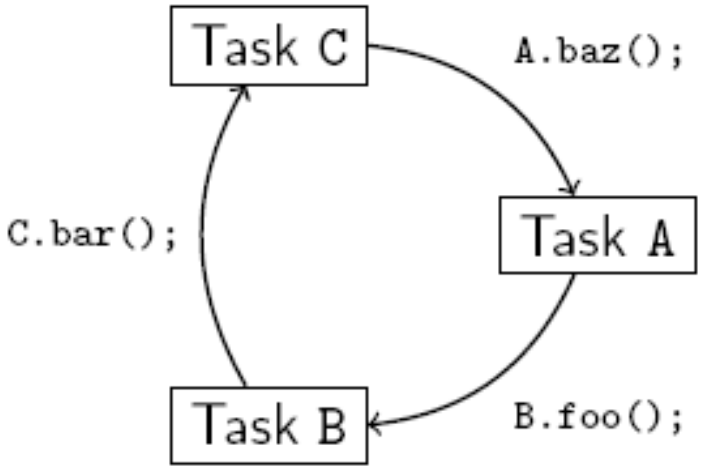
Ada: Selective Wait

```
--wait for one of the calls
select
      accept Init ;
      -- Statements after Init call
or
      accept Get (...) do
      -- Statements for Get call ( Caller blocked !)
      end Get;
end select;
```

One can add timeouts using delay!

UPPSALA
UNIVERSITET

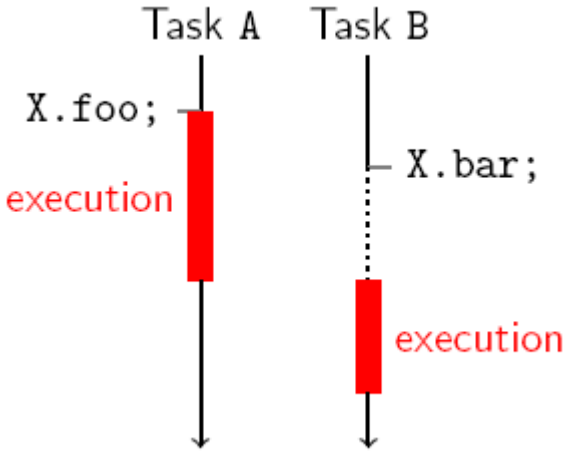
Ada: Deadlocks?



UPPSALA UNIVERSITET

Ada: Protected Types

- Only one procedure/entry of instance can run at any time
- Can implement mutually exclusive data access to an object X



UPPSALA UNIVERSITET

Ada: Protected Types

```
protected Integer_Buffer is
        entry Insert (i : in Integer );
        entry Remove (i : out Integer );
        private
                Buffer : Integer ;
                Empty : Boolean := True ;
end Integer_Buffer ; -- Note : could also be defined as a type

protected body Integer_Buffer is
        entry Insert (i : in Integer ) when Empty is
        begin
                Buffer := i; Empty := False ;
        end Insert ;
        entry Remove (i : out Integer ) when not Empty is
        begin
                i := Buffer ; Empty := True ;
        end Remove ;
end Integer_Buffer ;
```

UPPSALA UNIVERSITET

Ada: Random Number Generation

One of the ways!

```
-- package inclusion
with Ada.Numerics.Float_Random;
use Ada.Numerics.Float_Random;

-- Declaration of seed
G: Generator;
-- Reset the seed
Reset (G);
-- Get a random value
X:= Random(G);
```

UPPSALA UNIVERSITET

Ada: Lab Assignment

- **Part 1: Cyclic Scheduler**
  Implement a cyclic schedule of 3 procedures
  Avoid \drift" of the schedule
- **Part 2: Cyclic Scheduler with Watchdog**
  Add watchdog task to Part 1
  select with delay may be useful
- **Part 3: Process Communication**
  Implement Producer, Consumer and FIFO Buffer tasks and communication between them
- **Part 4: Data Driven Synchronization**
  Same as Part 3 but with Protected Type