# Ada Pointers – Access type

# Access type

- Pointers in Ada are called access type
- They point to objects located at certain addresses
- Objects of access types are implicitly initialized with null - pointing to nothing when not explicitly initialized
- used rarely in Ada, there are other ways without pointers

```
type Person is record
First_Name : String (1..30);
Last_Name : String (1..20);
end record;
type Person_Access is access Person;
```

# Access type

```
Father: Person_Access := new Person;
-- uninitialized
Mother: Person_Access := new
Person'(Mothers_First_Name, Mothers_Last_Name);
-- initialized
```

access the object in the storage pool by appending .all

`Mother.all` is the complete record;
components are denoted as usual with the dot notation

```
Mother.all.First_Name
```

When accessing components, *implicit dereferencing* (i.e. omitting .all) is more convenient

```
Mother.all := (Last_Name => Father.Last_Name,
First_Name => Mother.First_Name); -- marriage
```

# Access type

```
type Vector is array (1 .. 3) of
Complex;
type Vector_Access is access Vector;
VA: Vector_Access := new Vector;
VB: array (1 .. 3) of Vector_Access :=
(others => new Vector);
C1: Complex := VA (3); -- a shorter
equivalent for VA .all (3)
C2: Complex := VB (3)(1); -- a shorter
equivalent for VB(3).all (1)
```

# Access type

- deep and shallow copies when copying with access objects:

```
Obj1.all := Obj2.all; -- Deep copy:
Obj1 still refers to an object
different from Obj2, but it has the
same content

Obj1 := Obj2; -- Shallow copy: Obj1
now refers to the same object as Obj2
```

# Access type

- Ada standard mentions a garbage collector, which would automatically remove all unneeded objects that have been created on the heap

```
with Ada.Unchecked_Deallocation;
procedure Deallocation_Sample is
type Vector is array (Integer range <>) of Float; type
Vector_Ref is access Vector;
procedure Free_Vector is new Ada.Unchecked_Deallocation
(Object => Vector, Name => Vector_Ref);
VA, VB: Vector_Ref;
 V : Vector;
begin
VA := new Vector (1 .. 10); VB := VA; -- points to the
same location as VA
VA.all := (others => 0.0); -- ...
Free_Vector (VA); -- The memory is deallocated and VA is
now null
V := VB.all; -- VB is not null, access to a dangling
pointer is erroneous
end Deallocation_Sample;
```

# Access type

- When the keyword **all** is used in their definition, they grant read-write access

```
type Day_Of_Month is range 1 .. 31; type
Day_Of_Month_Access is access all
Day_Of_Month;
```

- General access types granting read-only access to the referenced object use the keyword constant
- The referenced object may be a constant or a variable.

```
type Day_Of_Month is range 1 .. 31; type
Day_Of_Month_Access is access constant
Day_Of_Month;
```

# Access type

```
type General_Pointer is access all Integer;
type Constant_Pointer is access constant Integer;
I1: aliased constant Integer := 10; I2: aliased
Integer;
P1: General_Pointer := I1'Access; -- illegal
P2: Constant_Pointer := I1'Access; -- OK, read only
P3: General_Pointer := I2'Access; -- OK, read and
write
P4: Constant_Pointer := I2'Access; -- OK, read only
P5: constant General_Pointer := I2'Access; -- read
and write only to I2
```

# Access type

- *Anonymous access types* are in two versions
- general access types, granting either read-write access or read-only access depending on whether the keyword constant appears
- anonymous access can be used as a parameter to a subprogram or as a discriminant

```
procedure Modify (Some_Day: access
Day_Of_Month);
procedure Test (Some_Day: access constant
Day_Of_Month);
type Day_Data (Store_For_Day: access
Day_Of_Month) is record -- components
end record;
```

# Access type

- An access to subprogram allows the caller to call a subprogram without knowing its name nor its declaration location

```
type Callback_Procedure is access procedure
(Id : Integer; Text: String);
type Callback_Function is access function
(The_Alarm: Alarm) return Natural;
```

```
procedure Process_Event (Id : Integer; Text:
String);
My_Callback: Callback_Procedure :=
Process_Event'Access;
```