Chapter 6: Custom types

1. Structures (recordings):

- A recording (or structure) is a type that allows you to store several data, of the same type or of different types.
- A recording contains components called ***fields***, each field corresponds to a piece of data.

Example :

A Student record contains four fields (Last Name, First Name, Date of Birth, Average).

Name
First name
Date of birth
Average

1.1. Declaration of a registration:

Type

Structure Name_RecordType
Field1: type1;
Field2: type2;
.
.
.
Field n: type n;
End Structure;

- The **Type** keyword is common to all new types that we want to add to the compiler;
- The **Structure** keyword indicates the start of the declaration of a structure;
- **RecordType_Name** is the name of the new type;
- The **End Structure** keyword indicates the end of the recording.

Example :

We suppose that we want to write a program that manipulates student information, namely: **the name**,**first name**,**the card number**,**Phone number**and the**8 notes**of each student.

So, the data for each student is:

- **name**: string
- **first name**: string
- **card number**: integer
- **telephone number**: integer
- **note**: array of real

We declare the Student structure as follows:

Type

Structure Student

Name: string;

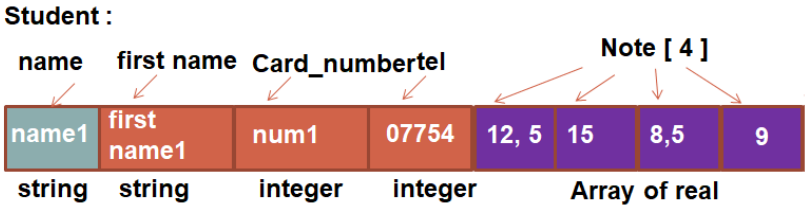
First_name: string;

NumCart: integer;

TelNumber:String;

Note[4]: array of real;

End Structure ;



Noticed :

- we can declare several structures (records), which can be related to each other.

For example, we can decompose the Student structure declared previously into two structures as follows:

Type

Structure Identity

Name: string ;

First_name: string **;**

End Structure ;

Structure Student

Ident: Identity;

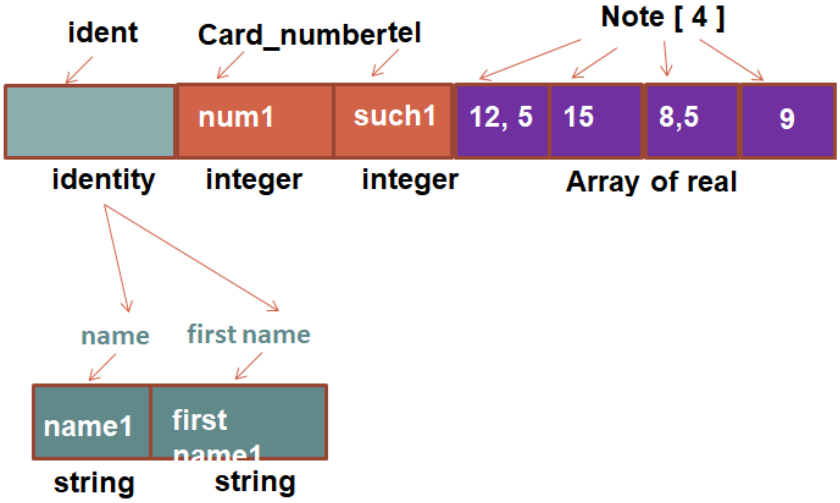
NumCart: integer;

Tel: integer;

Note[4]: array of real ;

End Structure ;

E1:



- the structure **Student** contains a field (Ident) which is of type **Identity**, which we declared **before** Student.

After declaring the types, we declare the variables that we will use:

stud1, stud2, stud3: Student;

- The three variables **stud1**,**stud2,** And **stud3** are of type **Student** therefore each one of these three variables has the following structure:

NumCart
Ident
Name
First_name

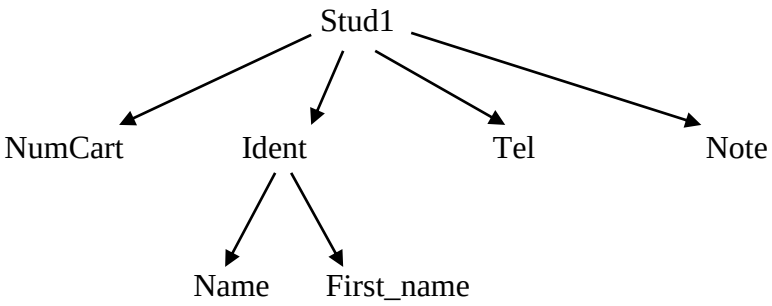
Tel
Note [4]

1.2. Handling of records:

a) Access to fields of a record:

- A record type variable (structure) can be represented in the form of a tree of fields.

Example : The Stud1 variable can be represented as follows:



- We access a field by specifying the name of the record type variable followed by the field identifier separated by a dot (.):

Example:

- To access the Stud1 student card number, write:

Stud1.NumCart

- To access the name, we write:

Stud1.Ident.Name

- To access the third element of the Note array, we write:

Stud1.Note[3]

b) Reading and writing records:

- By analogy with tables, the fields of a structure are read or displayed one by one because we can only read or display the simple types defined by the compiler.

- **Reading :**

Example :

- To fill the Stud1 variable in the previous example from standard input, we proceed as follows:

```
Write ('Please give the card number:');
Read (Stud1.NumCart);
Write ('Please give the name:');
Read (Stud1.Ident.Name);
Write ('Please give the First Name:');
```

```
Read(Stud1.Ident.First_Name);
Write ('Please give the Telephone Number:');
Read (Stud1.Tel);
For i = 1 to 4 do
        Write ('Please give the ', i, 'th note' );
        Read (Stud1.Note[i]);
End For
```

- **Writing:**

➢ To display the Stud1 variable from the previous example, proceed as follows:

Example :

```
Write ('Card number:', Stud1.CartNumber);
Write ('Name:', Stud1.Ident.Name);
Write ('First Name:', Stud1.Ident.First_Name);
Write ('Telephone Number:', Stud.Tel);
For i = 1 to 4 do
        Write ('The ', i, 'th Note is:', Stud1.Note[i])
End For
```

Remarks:

➢ Unlike arrays, there is no possibility of using a loop to manipulate all the elements of a record.

➢ In practice, the number of fields is very small (5 to 20 fields).

➢ A record can be assigned (with a variable of the same type):
Stud2←Stud1; /* This means that all the fields of Stud1 are copied to the same fields of Stud2*/

2.4. Arrays of Record:

➢ It is possible to declare an array whose elements are of record type.

➢ So, we first define the structure, and then we declare the existence of an array whose elements are of this type.

Type

```
    Structure Name_RecordType
     variable 1: type1;
     variable 2: type2;
     .....
     variable n: type n;
    End Structure
```

Array_Name [size]: Array of Name_RecordType ;

➢ To select the third field of the fourth element of the array we use the syntax:

Array_Name [4]. variable3.

➢ **Writing:**

```
Read (Tab [2].Name);
Tab [4].average ←10.5;
```

- **Reading :**

write (Tab [2].Name)
X←Tab [2].First_name ;

- **Comparison :**

If (Tab [2].average < 10) then

Exercise:

We want to record the results of the first year students (we have 300 students) to make some statistics. For this, we must know for each student their last name, first name, group, and their general average.

1) Write the algorithm that allows you to read the student information then display the **name** and the **first name** of the students from group N° **8**?

Solution :

```
Algorithm Statistical
Type
Structure Student
name: string;
first_name: string;
group: integer;
average: real
End Structure ;

E [300]: Array of Student
Begin
// reading student information
For i = 1 to 300 DO
Read (E [i].name);
Read (E [i].first_name);
Read (E [i].group);
Read (E [i].average);
End for
// displaying the first and last name of students in group 8
For i=1 to 300 DO
  If (E[i].group = 8) then
    Write (E [i].name);
    Write (E [i].first_name);
  end if
End for
END.
```

2) modify the previous algorithm so that it displays **admitted students** of the group **8?**

Answer :

```
// displaying the first and last name of admitted students in group 8
```

```
For i=1 to 300 DO
  If (E[i].group = 8) and (E[i].average >= 10) then
    Write (E [i].name);
    Write (E [i].first name);
  end if
end for
```

2) modify the previous algorithm so that it displays the best student of the promo?

```
// in the declaration part we must define a max variable of type Student
/* displaying the first and last name of the top average of the promo represented by the variable max */
Max← E[1] ;
For i=2 to 300 DO
  If (E[i].average > Max.average) then
     Max←E[i] ;
  end if
end for
Write(Max.name);
Write (Max.first_name);
```

3) modify the previous algorithm so that it displays the best student of the group 8?

```
/* First we must look for the first student of group 8 in the array for this we will use the while loop with a boolean variable b*/
i←1; b←TRUE ;
while (b=true) do
  If (E[i].group = 8) then
     Max←E[i] ;
     B←false ;
  End if
  i←i+1;
End while
// after this loop, max contains the first student of group 8 but not the best,
//now we will look for the best through another loop as follows
For i=1 to 300  DO
  If ((E[i].group = 8) and (E[i].average > Max.average)) then
     Max←E[i] ;
  end if
End for
Write (max.name);
Write (max.first_name);
END.
```