Chapter 6

Custom Types

In algorithms, in addition to predefined types (integer, real, boolean, character, strings, and arrays), users can define new types (custom types).

In our course, we are primarily interested in types such as enumeration and record.

5.1. Enumerations: (Enumerated Type)

- An enumerated type, as the name suggests, allows for the exhaustive definition of possible values.
- An enumerated type is defined by an identifier name and a range of values.
- Any enumerated type must be declared (defined) before its use.

5.1.1. Declaration:

Type NewType enum = (Value1, Value2, Value3, ...)

Example:

```
Type
 Day enum = (Saturday, Sunday, Monday, Tuesday, Wednesday, Thursday, Friday)
 Month enum = (January, February, March, April, May, June, July, August, September,
                   October, November, December)
Variable
J1, J2: Day;
M: Month;
```

// Variables J1 and J2 can only take one of the values: Saturday, ..., Friday.

// Variable M can only take one of the values: January, ..., December.

Note:

- Constants in an enumeration are related by an **order** defined by the position of values in the enumeration. Thus, the order in which identifiers are listed is significant.

  Example: Saturday < Monday and December > January.

- Names assigned to different constants (values) in an enumeration cannot be reused.

  Example: Saturday: integer ; // error

5.1.2. Operations:

Functions defined on enumerated types include:
- Ord(x)`: This function returns a positive integer corresponding to the rank of x in the list.

- Succ(x)`: This function provides the constant immediately following the value of x in the enumeration. The successor of the last value is not defined.

- Pred(x)`: This function provides the constant immediately preceding the value of x in the enumeration. The predecessor of the first value is not defined.

Example:

Ord(Saturday) = 1, Ord(Sunday) = 2, ..., Ord(Friday) = 7.
Succ(Saturday) = Sunday, Succ(Sunday) = Monday, ..., Succ(Friday) = ? (not defined).
Pred(Friday) = Thursday, Pred(Thursday) = Wednesday, ..., Pred(Saturday) = ? (not defined).

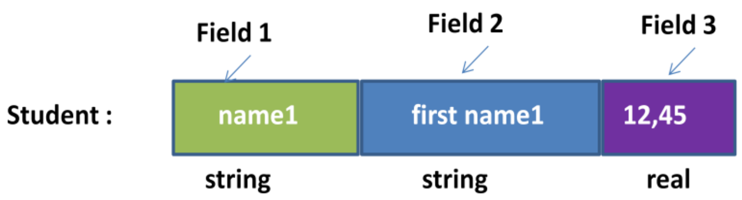
5.2 Structure type (or record)

Issues : we want to safeguard students' averages :
For each student we need to store :

- The name (string)
- First name (string)
- The average (real)

Solution :We have to use a customised type of data! We define a new type and call it **student** containing:

- the name (string)
- First name (string)
- The average (real)



5.2.1. Definition

- A record (or structure) is a type used to store several items of data, of the same or different types.
- A record is made up of components called fields, each of which corresponds to a piece of data.

5.2.2 Declaration of a Record:

```
Type

     Structure RecordTypeName
       Field1: Type1;
       Field2: Type2;
       …..
       Fieldn: Type n;
     EndStructure;
```

- The keyword "Type" is common to all new types that one wants to add to the compiler.
- The keyword "Structure" indicates the beginning of the definition of a structure.
- "RecordTypeName" is the name of the new type.
- The keyword "EndStructure" indicates the end of the record.

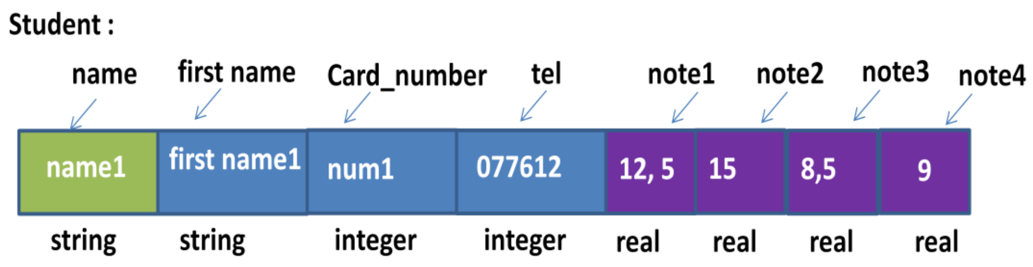
__**Example:**__ Suppose that we want to write a program that manipulates information about students, including the name, first name, card number, phone number, and the 4 grades for each student.

The following declarations are necessary for this manipulation:

```
Type
  Structure StudentIdentity
    Name: String;
    FirstName: String;
  EndStructure;
```

```
Structure Student
  CardNumber: Integer;
  Identity: StudentIdentity;
  PhoneNumber: String;
  Grades[4]: Array of Real;
EndStructure;
//variables
Student  student1, student2, student3;
```

The three variables *student1, student2, and student3* are of type *Student*, so each of these three variables has the following structure:

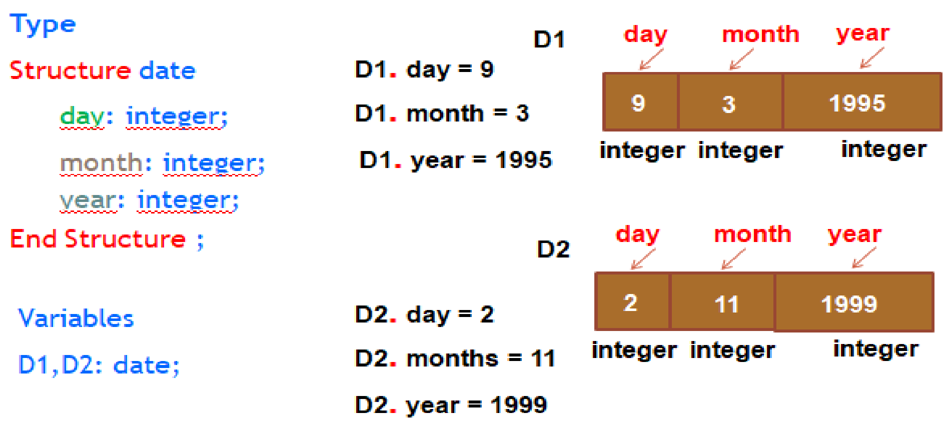


5.2.3. Handling records

a) Accessing fields in a record

Access to a field is done by specifying the name of the record type variable followed by the field identifier separated by a dot (.):

<u>Example :</u>



```
Type
Structure date
    day: integer;
    month: integer;
    year: integer;
End Structure ;

Variables
D1,D2: date;
```

```
D1. day = 9
D1. month = 3
D1. year = 1995

D2. day = 2
D2. months = 11
D2. year = 1999
```

b) Reading and writing records

By analogy with arrays, the fields of a structure are read or displayed one by one, because only the simple types defined by the compiler can be read or displayed.

Example : **reading information from a student**

```
…………………

Stud1 : student ; // Variable

Write('Please enter the card number:') ;
```

```
Read (Stud1.Card_number) ;
Write('Please enter the name:') ;
Read (Stud1.ident.Name) ;
Write('Please enter the first name:') ;
Read (Stud1.ident.Firstname) ;
Write('Please give the Telephone Number:') ;
Read (Stud1.Tel) ;
For i = 1 to 4 (step=1) do
  Ecrire(‘please enter the note N° ', i ) ;
  Read(Stud1.Note[ i ] );
End For
Write('Please give date of birth day month year' ) ;
  Read(Stud1. date_birth.day);
  Read(Stud1. date_birth.month);
  Read(Stud1. date_birth.year);
………….
```

- **Writing:**

- To display the variable "stud1" from the previous example, the procedure is as follows:

```
write (Stud1.Card_number) ;
write (Stud1.ident.Name) ;
write (Stud1.ident.Firstname) ;
write (Stud1.Tel) ;
For i = 1 to 4 (step=1) do
Write (Stud1.Note[ i ] );
End For
Write (Stud1. date_birth.day);
Write (Stud1. date_birth.month);
Write (Stud1. date_birth.year);
```

Notes :

- 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 limited (5 to 20 fields).

- A record can be the subject of an assignment (with a variable of the same type):
- Stud2 = stud1; /* This means that all the fields of stud1 are copied to the
- corresponding fields of stud2 */

5.2.4. Arrays of Records:

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

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

```
Type
Structure  name_Typerecord
  variable1 : type1;
  variable2 : type2;
  ………
  variablen : type n;
EndStructure
Name-Array[size] : Array of name-Typerecord ;
```

To select the third field of the fifth element of the array, the syntax is used as follows:

Name-array[5].variable3.

- **Reading:**

    read (Tab[2].Name);

    Tab[4].moyenne ← 10.5;

- **writing**

    write (Tab[2].Name);

- **Comparison:**

    if (Tab[2].moyenne < 10) then