Chapter 6: Custom types

In addition to predefined types (standard), the programmer can define new types. In this course we are mainly interested in types: Enumeration and Records.

1. Enumerations

An enumeration is a type whose area of values is defined by the programmer.

- The months of the year (January, February ...);
- Playing cards names (AS, King...);
- Car marks (Peugeot, Renault, Fiat, ...);
- Civil status indications (single, married, divorced ...).

Syntax of declaration of a listed type:

Type Type_name = (Val1, Val2, ……., Valn);

Declaration of a variable of an enumerated type:

Var

variable_name : name_ type ;

Examples:

Type

day = (Saturday, Sunday, Monday, Tuesday, Wednesday, Thursday, Friday);

month = (January, February, March, April, May, June, July, August, September, October, November, December) ;

Var

D1, D2: Day;

M: month;

- Variables D1 and D2 can only take one of the values:
  Saturday... Friday.
- The variable M can only take one of the values: January... December

Constants of an enumeration are linked by an order-relation defined by the position of values in an enumeration. Then, the order in which identifiers are listed is significant. Examples: Saturday<Monday and December> January.

The names attributed to the various constants of an enumeration cannot be reused.

```
Var
  Saturday: integer; // error !!!
```

Some functions can be used to manipulate enumerated types:

- Ord (x): This function returns a positive integer corresponding to the rank of x element in the enumeration.
- Succ (x): This function provides the constant which immediately follows the value of X in the enumeration. The successor of the last value is not defined.
- Pred (x): This function provides the constant which immediately precedes the value of X in the list. The first value predecessor is not defined.

Examples:

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

Syntax of enumeration in C language:

To declare such a type, we start with the **Enum** keyword. As following:

Enum Name_ Type {Val1, Val2, …, Valn};

Example:

Enum Day {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday};

Declaration of a variable of an enumerated type:

Enum Name_ type Name_variable;

Note:

The C language considers the values of the types enumerated as integer constants, converting them in the order in which they were listed during the declaration from 0.

Example :

```
#include <stdio.h>

Enum day {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

int main()
{
   enum day date;
   date = Tuesday;
   Printf ("The %d day of the week is : ", date+1);
   return 0;
}
```

2. The records

1.1. Definition

A record is a data structure allowing a set of data of different types to be grouped with the same and single object.

A record (also called a structure) is made up of components called fields.

Each field is identified by a type and a name which allows direct access to it.

1.2. Declaration

```
Type  id_record   =  Record
                        Id_Field1: Type1 ;
                        Id_Field2: Type2 ;
                        …
                        Id_FieldN: TypeN ;
                     End;
```

- Id_Field1, Id_Field2 … Id_FieldN: Are the identifiers of the fields of the record.
- Type1, Type2, …, TypeN : Are the types associated with the fields.
- Once the record type is defined, you can declare variables of this type.

Syntax :

```
Var
    Id_variable : id_record ;
```

Example :

Consider the information concerning a student: name, age, email, baccalaureate note, can be represented using a record as follows:

```
Type student = Record
                 name: string [N];
                 age: int ;
                 email: string [N];
                 bac: Real;
               end;
var
    S1, S2, S3: Student;
```

A record can be represented by a set of boxes. These boxes can be different sizes, because the types of a record are not necessarily the same as for a table.

	Name	age	email	bac
S1	''Biskri Ali''	19	Biskri.ali@gmail.com	14.25

1.3. Access to a field of a record

The fields of a record are accessible using the variable identifier and the field name separated by a dot (.)

Syntax:

Id_Variable.id_field

Example 1:

S1.age
S2.email

Since the fields of a record correspond to a consecutive space of bytes, therefore they play the role of variables. They can thus be used in assignment, reading, writing, etc. actions.

Example 2:

```
S1.age ← 21 ;
read (S1.name) ;
write (S1.email) ;
```

1.4. Records in C language

Syntaxe :

```
Typedef struct {
                    Type1  Id_Field1;
                    Type2  Id_Field2;
                     ...
                    TypeN  Id_FieldN;
                }id_record;
```

Example :

```
Typedef struct {
                    Char name [10] ;
                    Int  age;
                    Char email  [10] ;
                    Float bac ;
                } Student ;
```

Example :

```
#include <stdio.h>

typedef struct {
                     char name[20];
                     int age;
                   } person;

int main()
{
   person p;
   gets(p.name);
   scanf("%d",& p.age);
   printf("\n  The name is : ");
   puts(p.name);
   printf("\n The age is : %d  ", p.age );

  return 0;
}
```

1.5. Case of nested structures

A record can be nested in a table or record types. The notation used to select fields remains the same (Use of point).

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

```
Type  id_record   =  Record
                          Id_Field1: Type1 ;
                          Id_Field2: Type2 ;
                          …
                          Id_FieldN: TypeN ;
                    End;
 Var
    id_array : array [ 1 .. N ] of id_record ;
```

1.6. Access to elements

We first access the table box, using the brackets [ ], then we access the field using the dot symbol (.)

Example:

Id_array [i] . id_Field;

1.7. Manipulating arrays with record type

- Read(Tab[2].age); // save the value in the age field of the 2nd element in the array.
- Tab[3]. bac ← 13.50; // assign a value 13.50 to the field bac in the 3rd box of the table.
- Write(Tab[1].name); // display the name in the 1st box of the table.

Exercise:

Using records and arrays structures, write an algorithm which allows:

1- Creates and fill a data base for students (name, age, email, phone number).

2- Calculate the number of students with age superior of 28