REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université de M'sila
Faculté des Mathématiques et de l'Informatique
Département d'informatique

جامعة المسيلة
كلية الرياضيات والإعلام الآلي
قسم الإعلام الآلي

Chapter 6: Custom types

Algorithms and data structure 1

Presented by : Dr. Benazi Makhlouf
Academic year : 2023/2024

Contents of chapter 06:

1. Introduction
2. Enumerations
3. Records or structures
4. Other type definition possibilities

1. Introduction

- In programming, each manipulated data must have its own type.
- This allows the compiler to validate values and operations to apply.
- For the programmer, it helps to discover and avoid errors.
- There are several predefined types in the programming language, such as integers and characters.
- The programmer can also define their own types, derived from basic types, like arrays, enumerations, structures, and other types.

2. enumerations

- The **enum** type: It is an ordered list of constant values, defined by giving them names, which facilitates memorization and understanding of the program.
- The **enum** type is used to define a new type that contains only these values.

Declaration:

```
enum type_name {const_name1, ...};
enum type_name {const_name1, ...} var_list;
```

- **type_name** is the name given by the programmer to this set.
- **const_name** are constant names referring to the elements of the set.
- **var_list** is a list of variables of this type.

comment

- Constants provide easier-to-remember names during programming than numbers, and they define the set of acceptable values. This prevents the programmer from making an error during programming, as it is not possible to assign a value outside the set to an **enum** variable (C++).
- You can use **comparison** operations such as >, <, =, ≠ with this type. This type can also be used with the **switch** statement.

Examples

```
enum Gender {Male, Female };

enum Month {Janv, Feb, March, Apr, May, June, July, August, Sept,
            Oct, Nov, Dec};

enum Days {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday,
           Sunday};

Days d;

d=Tuesday ;
switch(d)
{
   case          Friday: printf("Weekend\n") ; break;
   case          Saturday: printf("de 08:00 à 12:00\n"); break;
   default :     printf("de 08:00 à 16:00\n") ;
}
```

typedef

Used to change the name of a type to increase readability

Syntax

```
typedef   typeBase   newName;
```

Example

typedef int Boolean;

Boolean b1,b2;

3. structures

- Records or structures are a composite type, representing a set of named elements, which can be accessed by their names. Each element is called a field, and these fields can be of any type.
- The structure is used to group variables into a single record.

Declaration:

```
struct struct_name {
  type field_names ;
  …
}  structure_variables ;
```

- struct_name : a name for the structure (optional) and must be a valid identifier.
- type field_names : Declares the fields that compose the structure.
- `structure_variables` : names of variables of structure type, also optional.

Example

```
struct Student {
   char name[20] ;
   float bac;
}e1, e2;
```

The declaration of variables can be deferred like this:

```
struct Student {
   char name[20] ;
   float bac;
};

struct Student e1, e2 ;
```

- The "**;**" is required after "**}**“
- "**struct**" must be mentioned before the name of the structure in C, but it is not required in C++.

Using **typedef**

- It is preferable to use "**typedef**" to declare a **struct** type before declaring variables.

```
typedef struct Student
{
  char name [20] ;
  float bac ;
} Student ;

Student e1, e2 ;
```

```
typedef struct {
  char name [20] ;
  float bac ;
} Student ;
```

```
typedef struct Student
{
  char name[20] ;
  float bac ;
} ;
```

Representation

- When defining a **struct** type, memory is not reserved.
- After declaring **variables**, memory is reserved.
- The structure is represented in memory by adjacent variables.
- For example:

e1	
nom	bac
Ahmed	13.41

e2	
nom	bac
Souad	12.50

The size of a structure is the sum of the sizes of its constituent fields.

Initialization

- In C, initial values can be specified for all elements of the structure within two braces { and } during their declaration. The values are separated by a comma « , » and these values must be of the same type, order, and number of fields.

Example

```
Student e1= { "Ahmed", 13.41} ;
```

Use

- The dot symbol « . » is used to access the elements of the structure.

Example

```
e1.moy=12.45;
scanf("%f",&e1.moy) ;
e2.moy=e1.moy+1 ;
e2=e1;
Student T[100] ;
```

```
strcpy(e1.name, "Ahmed") ;
gets(e2.name) ;
```

Example

- Write a program that defines a structure containing information about a student (student number, student name, date of birth, and high school average). Note that the date of birth is represented as a structure containing (day, month, year). Then, the program fills an array of N students and asks the user for a date to display all students born on that date.

```
#include <stdio.h>
typedef struct{
    int Day, Month , Year;
} Date ;

typedef struct{
    int num;
    char name [20] ;
    Date birthday ;
    float bac ;
} Student ;

int main(){
  Student st[100] ;
  Date d ;
  int i, N ;
printf(" enter number of students
\n") ;
scanf("%d",&N) ;

// fill in the table
for(i=0 ;i<N ;i++){
    printf(" student %d\n",i) ;
    printf("Num : ") ;
    scanf("%d",&st[i].num) ;
    getch() ;
    printf("Name : ") ;
    gets(st[i].name) ;
    printf("Date of birth (j/m/a):");
    scanf("%d%d%d",
          &st[i].birthday.Day,
          &st[i].birthday.Month,
          &st[i].birthday.Year );
    printf("Bac : ") ;
    scanf("%f",&st[i].bac) ;
}
```

```
printf(" enter a date (j/m/a) : ") ;
scanf("%d%d%d", &d.Day, &d.Month, &d.Year ) ;

// display
printf("Num\tName\tBac\n");
for(i=0 ;i<N ;i++)
    if( st[i].birthday.Day==d.Day) &&
    (st[i].birthday.Month==d.Month) &&
    (st[i].birthday.Year==d.Year) )
        printf("%d \t%s \t%.2f\n", st[i].num, st[i].name, st[i].bac) ;

return 0 ;
}
```

Other type definition possibilities

- union
- reference
- class
- …

union 1/2

A union, like a structure, is a group of elements of different types. But it can only contain a single value of one of its elements at any one time.

Declaration

```
union union_name {
 type field_names ;
 …
} union_variables ;
```

Example

```
union Result{
   char grade;
   float moy;
} r1, r2;
```

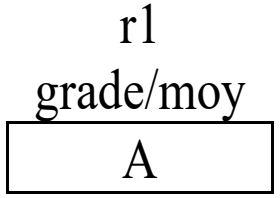
union 2/2

```
typedef union {
   char grade;
   float moy;
} Result;
Result r1, r2 ;
```

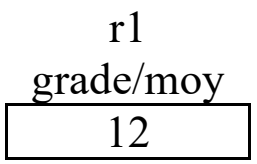
Representation

Unions are represented in memory as a single variable, which takes the size of the largest element in the union. For example, when declaring the variable r1 of type Result, if we assume that the size of the grade char is 1 byte and the size of float moy is 4 bytes, the size of the variable r1 is 4 bytes, not 5 bytes.

r1.grade='A';



r1.moy=12;



Reference (C++ only)

References allow you to manipulate a variable with a different name from the one declared.

syntax :

type **&**référence = identificateur;

Example :

int **&**ref = i;

Here, “i" and “ref” become names for the **same** variable.

End Chapter 06