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

Chapitre 6:
Les types personnalisés

Algorithmique et structure de données

Présenté par: Dr. Benazi Makhlouf
Année universitaire: 2022/2023

Contenu du chapitre 06:

1. Introduction
2. Les Enumérations
3. Les enregistrements ou structures
4. Autres possibilités de définition de type

1. Introduction

- En programmation, chaque donnée manipulée doit avoir son propre type.
  - Cela permet au **compilateur** de valider les valeurs et les opérations à appliquer.
  - Pour le **programmeur** à découvrir et à éviter les erreurs.
- Il existe plusieurs types prédéfinis dans le langage de programmation.
  - Comme les nombres entiers et les caractères.
- Le programmeur peut également **définir ses propres types**, dérivés des types de base.
  - Tels que les tableaux, les énumérations, les enregistrements et d'autres types.

2. Enumérations

- Le type **enum** : C'est une liste ordonnée de valeurs constantes, définies en leur donnant des noms. ce qui facilite la mémorisation et la compréhension du programme.
- Le type **enum** est utilisé pour définir un nouveau type qui ne contient que ces valeurs.

Déclaration

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

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

- type_name est le nom que le programmeur donne à cet ensemble.
- nom_const noms de constantes faisant référence aux éléments de l'ensemble.
- var_list est une liste de variables de ce type

2. Enumérations

• Les constantes fournissent des noms plus faciles à mémoriser lors de la programmation que des nombres, et elles définissent l'ensemble de valeurs acceptable. Cela évite au programmeur de faire une erreur lors de la programmation, car il n'est pas possible d'affecter une valeur en dehors de l'ensemble à une variable enum (C++).

• Vous pouvez utiliser des opérations de comparaison telles que >, <, =, ≠. Ce type peut également être utilisé avec l’instruction **switch**.

Exemples

- **enum** Sexe {Male, Femelle };
- **enum** Mois {Janv, Fevr, Mars, Avr, Mai, Juin, Juil, Aout, 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

Utilisée pour changer le nom d’un type pour augmenter la lisibilité

Syntaxe

```
typedef typeBase newName;
```

Exemple

```
typedef int boolean;
boolean b1,b2;
```

3. Les enregistrements ou structures

• est un type composé, représentant un ensemble d'éléments nommés, auxquels on peut accéder par leur nom. Chaque élément est appelé un **champ**. Ces champs peuvent être de n'importe quel type.

• La structure est utilisée pour regrouper les variables en un seul enregistrement.

Déclaration

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

- struct_name : un nom pour la structure (facultatif) et doit être un identifiant valide.
- type field_names : Déclare les champs qui composent la structure.
- `structure_variables` : noms des variables de type structure, également facultatives.

Exemple

```
struct Etudiant {
  char nom[20] ;
  float bac;
}e1, e2;
```

La déclaration des variables peut être retardée comme ceci

```
struct Etudiant {
  char nom[20] ;
  float bac;
};

struct Etudiant e1, e2 ;
```

Le "**;**" est requis après **}** , et **struct** doit être mentionne avant le nom de la structure en C. mais elle n'est requise en C++.

il est préférable d'utiliser **typedef** pour déclarer un type struct, avant de déclarer des variables.

```
typedef struct Etudiant {
  char nom[20] ;
  float bac ;
} Etudiant ;

Etudiant e1, e2 ;
```

```
typedef struct {
  char nom[20] ;
  float bac ;
} Etudiant ;
```

```
typedef struct Etudiant {
  char nom[20] ;
  float bac ;
} ;
```

Représentation

- Lors de la définition d’un type de **struct**, la mémoire n’est pas réservée
- Aprés la déclaration des variables la mémoire est réservée
- La structure est représentée en mémoire par des variables adjacentes.
- par exemple

e1	
nom	bac
Ahmed	13.41

e2	
nom	bac
Souad	12.50

La taille d’une structure est la somme des tailles de ses champs constitutifs.

Initialisation

• En C, des valeurs initiales peuvent être spécifiées pour tous les éléments de la structure au sein de deux accolades { et } lors de leur déclaration. Les valeurs sont séparées par la virgule « , », et ces valeurs doivent être du même type, ordre et nombre de champs.

Exemple

Etudiant e1= { "Ahmed", 13.41} ;

Utilisation

- Le symbole "." point est utilisé pour accéder aux éléments de la structure.

Exemple

- `e1.moy=12.45;`
- `strcpy(e1.nom, "Ahmed") ;`
- `scanf("%f",&e1.moy) ;`
- `gets(e2.nom) ;`
- `e2.moy=e1.moy+1 ;`
- `e2=e1;`
- `Etudiant T[100] ;`

Exemple

• Écrivez le programme qui définit une structure contenant des informations sur l'étudiant (numéro d'étudiant, nom de l'étudiant, date de naissance, la moyenne du bac). Notez que la date de naissance est une structure qui contient (jour, mois, année). Puis il remplit un tableau de N étudiants, puis demande à l'utilisateur une date, pour afficher tous les étudiants nés à cette 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("entrer le nbr des étudiants\n") ;
scanf("%d",&N) ;

//remplir le tableau
for(i=0 ;i<N ;i++){
    printf("étudiants %d\n",i) ;
    printf("Num : ") ;
    scanf("%d",&st[i].num) ;
    getch() ;
    printf("Nom : ") ;
    gets(st[i].name) ;
    printf("Date de naissance (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("entrer une date (j/m/a) : ") ;
scanf("%d%d%d", &d.Day, &d.Month, &d.Year ) ;

//affichage
printf("Num\tNom\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 ;
}
```

Autres possibilités de définition de type

union

Une union, comme une structure, est un groupe d'éléments de types différents. Mais il ne peut contenir, à chaque instant du programme, qu'une seule valeur d'un de ses éléments.

Déclaration

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

Exemple

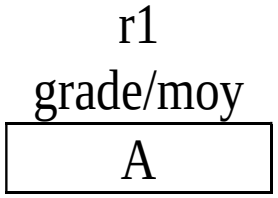
```
union Result{
  char grade;
  float moy;
}r1, r2;
```

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

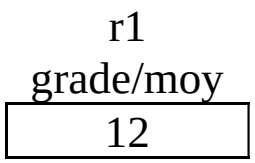
Représentation

L’union est représentée en mémoire comme une variable unique, qui prend la taille du plus grand élément de l’union. Par exemple, lors de la déclaration de la variable R1 du type Result, si nous supposons que la taille du grade char est de 1 octet et que la taille de float moy est de 4 octets, la taille de la variable r1 est de 4 octets, et non de 5 octets.

r1.grade='A';



r1.moy=12;



Référence (C++ uniquement)

Les références permettent de manipuler une variable avec un nom différent de celui qui a été déclaré.

syntaxe:

type &référence = identificateur;

Exemple:

int &ref = i;

Ici, i et ref deviennent des noms pour la même variable.

Fin Chapitre 06