Algorithmique et structures de données

Structures de données

Une introduction

Samar MOUCHAWRAB, PhD Eng

2A Cycle Préparatoire – Semestre 1
2017/2018

Introduction

Jusqu'à présent dans le cours, nous avons utilisé des données simples: caractères, entiers, flottants, et tableaux. On n'avait pas établi un lien fonctionnel entre des données de natures différentes.

Lorsque le nombre de données augmente, il devient très problématique de les manipuler sous une forme « éclatée ». Chaque fois que l'on veut manipuler une nouvelle entité, il faut déclarer un nombre de variables important.

De plus, si l'on souhaite utiliser une fonction sur l'une de ces entités, il faut lui passer en argument toutes les variables utiles pour le traitement : ceci devient rapidement très lourd, source d'erreurs, et potentiellement source de dégradation de performance.

Introduction

C’est pour résoudre ce type de problème que les langages de haut niveau proposent un mécanisme d’agrégation, permettant de regrouper des données.

On désigne en général par structure de données un tel objet agrégé. On trouve également la terminologie d’enregistrement ou de record.

On peut aussi distinguer les structures de données statiques où la taille est définie lors de la création des objets et la mémoire est réservée et les structures de données dynamiques qui changent de taille au besoin lors de l’exécution du programme.

Les types structurés en C

Une structure rassemble des variables, qui peuvent être de types différents, sous un seul nom ce qui permet de les manipuler facilement.

Elle permet de simplifier l'écriture d'un programme en regroupant des données liées entre elles.

Un exemple type d'utilisation d'une structure est la gestion d'un répertoire. Chaque fiche d'un répertoire contient (par exemple) le nom d'une personne, son prénom, son adresse, ses numéros de téléphone, etc...

Le regroupement de toutes ces informations dans une structure permet de manipuler facilement ces fiches.

Les types structurés en C

Autre exemple: On peut représenter un nombre complexe à l'aide d'une structure.

On définit une structure à l'aide du mot-clé: *struct* suivi d'un identificateur (nom de la structure) et de la liste des variables qu'elle doit contenir (type et identificateur de chaque variable).
Cette liste est délimitée par des accolades.

Chaque variable contenue dans la structure est appelée un champ ou un membre.

Définir une structure consiste en fait à définir un nouveau type. On ne manipulera pas directement la structure de la même manière que l'on manipule un type. On pourra par contre définir des variables ayant pour type cette structure.

Les types structurés en C

Exemple:

```
struct complexe
{
double reel; double imag;
};

struct complexe x,y;
```

Dans cet exemple, on définit une structure contenant deux doubles puis on déclare deux variables ayant pour type struct complexe.

Les opérations permises sur une structure sont :

- l'affectation (en considérant la structure comme un tout),
- la récupération de son adresse (opérateur &) et
- l'accès à ses membres.

Les types structurés en C

On accède à la valeur d'un membre d'une structure en faisant suivre l'identificateur de la variable de type structure par un point suivi du nom du membre auquel on souhaite accéder.

Par exemple `x.reel` permet d'accéder au membre « reel » de la variable `x`.

On peut initialiser une structure au moment de sa déclaration, par exemple:

```
struct complexe x={10,5};
```

Les types structurés en C

On peut définir des fonctions qui renvoient un objet de type structure.

Par exemple, la fonction suivante renvoie le complexe conjugue de x:

```
struct complexe conjugue(struct complexe x);
{
    struct complexe y;
    y.reel=x.reel;
    y.imag=-x.imag;

    return y;
}
```

L'utilisation de cette fonction pourra ressembler à:

```
    struct complexe x,z;
    z=conjugue(x);
```

Les types structurés en C

L'affectation revient à affecter chaque membre de la structure, par exemple:

```
struct complexe x,z;
x=z;
```

est équivalent à:

```
struct complexe x,z;
x.reel=z.reel;
x.imag=z.imag;
```

Exemple

Un programme en C utilisant une structure pour saisir et stocker les données d'un étudiant : nom, numéro, et note

```
#include <stdio.h>
struct student {
    char name[50];
    int roll;
    float mark;
};
void main(){
    struct student s;
    printf("Enter information of student:\n\n");
    printf("Enter name: ");
    scanf("%s",s.name);
    printf("Enter roll number: ");
    scanf("%d",&s.roll);
    printf("Enter mark: ");
    scanf("%f",&s.mark);
    printf("\nDisplaying Information\n");
    printf("Name: %s\n",s.name);
    printf("Roll: %d\n",s.roll);
    printf("Mark: %.2f\n",s.mark);
}
```

Exemple

Résultat d'exécution :

```
Enter information of student:

Enter name: Adele
Enter roll number: 21
Enter mark: 334.5

Displaying Information
name: Adele
Roll: 21
Mark: 334.50
```

Nom de nouveau type

Le langage C permet de créer de nouveaux noms de types de données grâce au mot-clé typedef.

Pour un type structuré, le mot-clé struct fait partie intégrante du nom de type. En utilisant typedef (type definition), précédant une définition de structure, le programmeur peut donner un nom de type à la structure définie. Exemple :

```
typedef struct Books {
   char title[50];
   char author[50];
   char subject[100];
   int book_id;
} Book;
```

On peut alors créer une variable livre de type Book:

```
Book livre;
```

Exemple

Création d'un nouveau type Book:

```
#include <stdio.h>
#include <string.h>

typedef struct Books {
   char title[50];
   char author[50];
   char subject[100];
   int book_id;
} Book;

void main( ) {
   Book book;
   strcpy( book.title, "C Programming");
   strcpy( book.author, "Nuha Ali");
   strcpy( book.subject, "C Programming Tutorial");
   book.book_id = 6495407;
   printf( "Book title : %s\n", book.title);
   printf( "Book author : %s\n", book.author);
   printf( "Book subject : %s\n", book.subject);
   printf( "Book book_id : %d\n", book.book_id);
}
```

Exercice 1

Ecrire un programme qui calcule la somme de deux nombres complexes. Définir une structure pour les nombres complexes et une fonction add

Solution

```
#include <stdio.h>
typedef struct complex{
    float real;
    float imag;
}complex;
complex add(complex n1,complex n2);
int main(){
    complex n1,n2,temp;
    printf("For 1st complex number \n");
    printf("Enter real and imaginary respectively:\n");
    scanf("%f%f",&n1.real,&n1.imag);
    printf("\nFor 2nd complex number \n");
    printf("Enter real and imaginary respectively:\n");
    scanf("%f%f",&n2.real,&n2.imag);
    temp=add(n1,n2);
    printf("Sum=%.1f+%.1fi",temp.real,temp.imag);
    return 0;
}
complex add(complex n1,complex n2){
      complex temp;
      temp.real=n1.real+n2.real;
      temp.imag=n1.imag+n2.imag;
      return(temp);
}
```

Exercice 2

Ecrire un programme qui calcule la différence entre deux périodes de temps (heures/minutes/secondes). Définir une structure TIME et une fonction difference

Solution

```
#include <stdio.h>
struct TIME{
  int seconds;
  int minutes;
  int hours;
};

struct TIME Difference(struct TIME t1, struct TIME t2);
int main(){
    struct TIME t1,t2,diff;
    printf("Enter stop time: \n");
    printf("Enter hours, minutes and seconds respectively: ");
    scanf("%d%d%d",&t1.hours,&t1.minutes,&t1.seconds);
    printf("Enter start time: \n");
    printf("Enter hours, minutes and seconds respectively: ");
    scanf("%d%d%d",&t2.hours,&t2.minutes,&t2.seconds);
    printf("\nTIME DIFFERENCE: %d:%d:%d - ",
               t1.hours,t1.minutes,t1.seconds);
    printf("%d:%d:%d ",t2.hours,t2.minutes,t2.seconds);
       diff = Difference(t1, t2);
    printf("= %d:%d:%d\n",diff.hours,diff.minutes,diff.seconds);
    return 0;
}
```

Solution

```
struct TIME Difference(struct TIME t1, struct TIME t2){
    struct TIME differ;
    if(t2.seconds>t1.seconds){
        --t1.minutes;
        t1.seconds+=60;
    }
    differ.seconds=t1.seconds-t2.seconds;
    if(t2.minutes>t1.minutes){
        --t1.hours;
        t1.minutes+=60;
    }
    differ.minutes=t1.minutes-t2.minutes;
       if (t2.hours > t1.hours) {
              t1.hours+=24;
       }
    differ.hours=t1.hours-t2.hours;
       return differ;
}
```

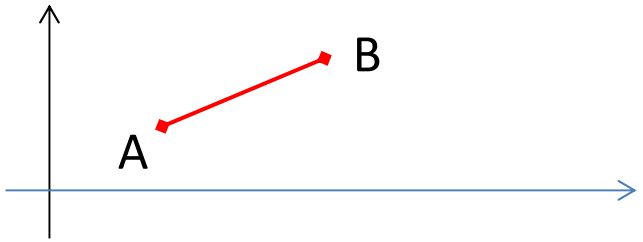
Exercice 3

Définir une structure **Point**, formée des coordonnées (abscisse et ordonnée) et une structure **Segment** formée des deux points extrémités du segment (structure de structure).

Ecrire une fonction **Milieu** qui retourne le point milieu d'un segment.

Ecrire une fonction **Longueur** qui retourne la longueur d'un segment.

A noter que la fonction *double sqrt(double x)* retourne la racine carrée de x. C'est une fonction de la librairie math.h



Solution

```
#include <stdio.h>
#include <math.h>

typedef struct Point Point;
struct Point {
        float x;
        float y;
};

typedef struct Segment Segment;
struct Segment {
        Point A;
        Point B;
};
```

Solution

```
Point Milieu (Segment S) {
        Point M;
        M.x = (S.A.x + S.B.x)/2;
        M.y = (S.B.y + S.A.y)/2;
        return M;
}

double Longueur (Segment S) {
        return sqrt((S.A.x-S.B.x)*(S.A.x-S.B.x) + (S.A.y-
S.B.y)*(S.A.y-S.B.y));
}
```

Solution

```
void main() {
        Segment S;
        printf("Entrer A et B");
        scanf("%f %f %f %f", &S.A.x, &S.A.y, &S.B.x, &S.B.y);
        printf("abscisse du milieu : %f \n", Milieu(S).x);
        printf("ordonne du milieu : %f \n", Milieu(S).y);
        printf("longueur du segment : %f \n", Longueur(S));
}
```

Exercice 4

Définir une structure Ville qui comprend les membres suivants : Nom, Nombre d'habitants, Pays.

Ecrire une fonction CompareVilles qui compare le nombre d'habitants de deux villes passées en paramètres et retourne le nom de la ville qui contient plus d'habitants. Si le nombre d'habitants est le même, retourner le mot « Egalite ».

Faire un test de la fonction CompareVilles en l'appelant de la fonction main.

Solution

```
#include <stdio.h>

typedef struct Ville {
        char Nom[20];
        int N_habitants;
    char Pays[20];
} Ville;

char* CompareVilles (Ville v1, Ville v2) {
        char* r;
        if (v1.N_habitants < v2.N_habitants )
                r = v2.Nom;
        else
                if (v2.N_habitants < v1.N_habitants )
                        r = v1.Nom;
                else r = "Egalite";
        return r;
}
```

Solution

```
void main() {
        Ville v1, v2;
        puts("saisir Nom, nb d'habitants en milliers et pays de la
                premiere ville :\n");
        scanf("%s %d %s",v1.Nom, &v1.N_habitants, v1.Pays);
        puts("saisir Nom, nb d'habitants en milliers et pays de la
                deuxieme ville :\n");
        scanf("%s %d %s",v2.Nom, &v2.N_habitants, v2.Pays);
        printf("\nResultat: %s", CompareVilles(v1, v2));
}
```