Les Structures

DAKKAR Borhen-eddine

Lycée le Corbusier

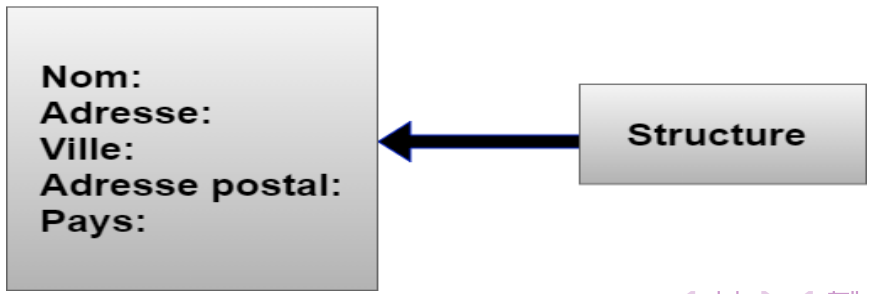
BTS SN-IR

Table des matières

Table des matières

Les structures

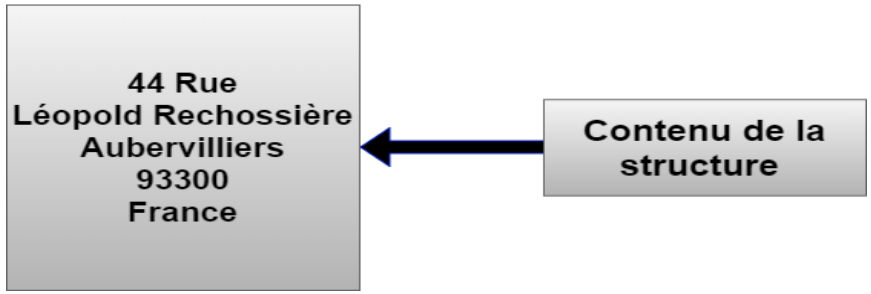
- Un tableau permet de désigner sous un seul nom un ensemble de valeurs de même type, chacune d'entre elles étant repérée par un indice.
- Les structures représentent un concept similaire aux tableaux mais plus large. En effet, une structure fait référence à la manière dont les éléments individuels d'un groupes sont organisés.
- En C, une structure est tout simplement la façon dont les éléments de données individuels sont agencés pour former une unité cohérente et associée.



Les structures

- Chacun des éléments de données individuels répertoriés dans la figure précédente est une entité par lui-même. Tous ces éléments de données forment une seule unité, représentant une adresse.
- L'adresse avec tous les éléments de données est appelée en C structure.
- Nous pouvons avoir des milliers d'adresses, la forme de chaque adresse ou sa structure est identique.
- Il est important de distinguer la forme de la structure et le contenu des données de la structure.
- La forme d'une structure se compose des noms symboliques, des types de données et de disposition des éléments individuels dans la structure.
- Le contenu d'une structure fait référence aux données réelles stockées dans les noms symboliques.

Les structures



- Le contenu de **Nom** est: 44 Rue
- Le contenu de **Adresse** est: Léopold Rechossière
- Le contenu de **Code postal** est: 93300
- Le contenu de **Pays** est: France

Déclaration des structures

- La déclaration d'une structure nécessite de lister le type de données, leur noms et leur disposition.

```
struct{
    int mois;
    int jour;
    int année;
    }naissance;
```

- La structure **naissance** réserve un stockage pour les éléments de données répertoriés à l'interieur. Elle se compse de trois éléments, appelés membres de la structure.
- L'attribution de valeurs aux éléments d'une structure est appellé **remplissage de la structure**.
- Chaque membre de la structure est accessible en donnant le nom de la structure et le nom individuel de l'élément, séparé par un point. Par exemple **naissance.mois** fait référence au mombre **mois** de la structure **naissance**.

Exemple

```c
#include <stdio.h>
main( )
{
    struct
    {
        int mois;
        int jour;
        int annee;
    }naissance;
    naissance.mois = 01;
    naissance.jour = 28;
    naissance.annee = 2021;
    printf ("Ma date de naissance est %d/%d/%d", naissance.mois,
    naissance.jour, naissance.annee);
}
```

- De plus, comme pour toutes les instructions de définition C, plusieurs variables peuvent être définies dans la même déclaration. Par exemple, l'instruction de définition:

```c
struct {int mois; int jour; int annee} naissance, actuelle;
```

crée deux structures ayant la même forme. Le type de données suit une liste de variables.

Autre mode de déclaration

- Une modification utile de la définition des structures consiste à lister la forme de la structure sans nom de variable. Dans ce cas les membres de données doivent être précédés par le nom de la structure. Par exemple, dans la déclaration.

```
struct date
{
    int month;
    int day;
    int year;
} ;
```

- La déclaration de la structure **date** fournit un modèle pour la structure sans avoir à réserver de lieux de stockage.
- Le modèle présente la forme d'une structure appelée **date** en décrivant comment les éléments de données individuels sont organisés.

Autre mode de déclaration

```c
#include <stdio.h>
struct date
    {
        int mois;
        int jour;
        int annee;
    };
main( )
{
    struct date naissance;
    naissance.mois = 29;
    naissance.jour = 28;
    naissance.annee = 2021;
     printf ("Ma date de naissance est %d/%d/%d", naissance.mois,
     naissance.jour, naissance.annee);
}
```

- L'initialisation des structures suit les mêmes règles que pour l'initialisation des tableaux.

```c
    struct date naissance = {12, 28,21};
```

Type de données d'une structure

- Tout type de données C valide peut être utilisé dans une structure.
- Considérons les données d'un étudiant suvantes composé des éléments suivants:
  - Nom
  - Prénom
  - Moyenne
- Une déclaration de cette structure peut être:

```
struct etudiant
{
char Nom[20J;
char Prenom[20];
float Moyenne;
}
```

- Une structure spécifique utilisant le modèle **etudiant** peut être définie et initialisée par:

```
struct etudiant etudiant1 = {"SAN1","1annee",12.5;}
```

Type de données d'une structure

- Les members d'une structure peuvent être n'importe quel type de données C valides, y compris des tableaux et des structures.
- L'inclusion d'une structure dans une autre structure suit les mêmes règles pour inclure tout type de données.

```c
#include <stdio.h>
struct date
{
    int mois;
    int jour;
    int annee;
};
struct etudiant
{

    char Nom[20];
    char Prenom[20];
    struct date date_naissance;
    int Moyenne;
};
main( )
{
    struct etudiant Etudiant_SN;
    Etudiant_SN.Nom = "BTSSN";
    Etudiant_SN.date_naissance.mois = 2;
}
```

Table des matières

Les tableaux de structures

```c
#include <stdio.h>
struct Etudiant
{
    long id;
    char Nom[20];
    char Prenom[20];
    float Moyenne;
}
main()
{
    int i;
    struct Etudiant liste_etudiant [5] =
    {
    { 32479, "BTS SN1","E1", 6.72 } ,
    { 33623, "BTS SN2" ,"E2", 7.54} ,
    { 34145, "BTS SN3","E3", 5.56 },
    { 35987, "BTS SN4" ,"E4", 5.43 } ,
    { 36203, "BTS SN5","E5", 8.72 }} ;
    for ( i = 0; i < 5; ++i)
    {
        printf("\n%ld %-20s %-20s %4.2f",list_etudiant[i] .id,
        liste_etudiant[i].Nom, liste_etudiant[i].Prenom,
         liste_etudiant [i].Moyenne);
    }
}
```

Pour accéder aux différents éléments de la structure, il faut donner la position de la structure souhaitée dans tableaux suivie d'un point et de l'élément de structure approprié.

Table des matières

Exercice 1

Écrire un programme qui :

- Lit au clavier des informations dans un tableau de structures du type **point** défini comme suit :

```
struct point {
        int num ;
        float x ;
        float y ;
            }
```

  Le nombre d'éléments du tableau sera fixé 5.

- Afficher l'ensemble des informations précédentes.

Exercice 1

```c
#include <stdio.h>
#define NPOINTS 5
main()
{ struct point
    {
        int num ;
        float x ;
        float y ;
    };
struct point courbe[NPOINTS] ;
int i ;
for (i=0 ; i<NPOINTS ; i++)
{ printf ("numero : ") ; scanf ("%d", &courbe[i].num) ;
printf ("x : ") ; scanf ("%f", &courbe[i].x) ;
printf ("y : ") ; scanf ("%f", &courbe[i].y) ;
}
printf (" **** structure fournie ****\n") ;
for (i=0 ; i<NPOINTS ; i++)
printf ("numero : %d x : %f y : %f\n",
courbe[i].num, courbe[i].x, courbe[i].y) ;
}
```

Table des matières

Passage de structures en paramètre

- Une structure peut être transmise à une fonction en incluant le nom de la structure comme argument de la fonction appelante. Par exemple, l'appel de la fonction:

```
calcul_salaire_net (employe) ;
```

- Où **employe** est une structure définit par :

```
struct
{
int N_id;
double paie_heure;
double Nr_heures;
} employe;
```

- L'appel de la fonction passe une copie complète de la structure **employe** à **calcul_salaire_net**.

Passage de structures en paramètre

```c
#include <stdio.h>
struct employe
{
int N_id;
double paie_heure;
double Nr_heures;
};
main( )
{
    /* declare une structure de type employe */
    struct employe emp = {6782, 8.93, 40.5};
    double salaire_net;
    double calcul_salaire_net (struct employe); /* Entête de la fonction */
    salaire_net = calcul_salaire_net(emp); /* passe une copie des
    varible de la structure */
    printf("Le salaire net de l'employee %d est %6.2f  Euros",
    emp.N_id,salaire_net);
}
double calcul_salaire_net (struct employe temp)
{ /* temp est une structure de type employe */
    return(temp.paie_heure * temp.Nr_heures);
}
```

Passage de structures en paramètre

- Le modèle de déclaration globale de la structure **employe** fourni dans le programme précédent est hautement préférable car le template global centralise la déclaration d'organisation de la structure.

- La déclaration globale assure que toute modification qui doit être apportée ultérieurement à la structure est à effectuer une suele fois sur le modèle global.

Passage par référence d'une structure

- Une alternative à la transmission d'une copie de la structure consiste à transmettre l'adresse du la structure. Ceci, bien sûr, permet à la fonction d'apporter des modifications directement sur la structure d'origine.

```
calcul_salaire_net (&employe) ;
```

- Dans cet appel, une adresse est transmise. Pour stocker correctement cette adresse la fonction **calcul_salaire_net ()** doit déclarer l'argument comme un pointeur.

```
calcul_salaire_net (struct employe *pt) ;
```

- Ici, **pt** déclare cet argument comme un pointeur vers une structure de type **employe**. La variable pointeur, **pt**, reçoit l'adresse de départ de la structure chaque fois que **calcul_salaire_net ()** est appelée.

Passage par référence d'une structure

- Ainsi, les expressions *** pt** et **(* pt). N_id**, font référence référence au membre **N_id** de la structure **employe**.
- L'expression générale **(* pointeur).element** peut être remplacée avec la notation **pointeur–> element**, où l'opérateur –> est construit du signe moins suivi d'une flèche orientée vers la droite.
- Par exemple, les expressions suivantes sont équivalentes:

```
(*pt).N_id /*peut être remplcée par*/ pt->N_id
(*pt).paie_heure /*peut être remplcée par*/ pt->paie_heure
(*pt).Nr_heures /*peut être remplcée par*/ pt->Nr_heures
```

Passage par référence d'une structure

```c
#include <stdio.h>
struct employe
{
int N_id;
double paie_heure;
double Nr_heures;
};
main( )
{
    /* declare une structure de type employe */
    struct employe emp = {6782, 8.93, 40.5};
    double salaire_net;
    double calcul_salaire_net (struct employe *); /* Entête de la fonction */
    salaire_net = calcul_salaire_net(&emp); /* passe une copie des varible de la st
    printf("Le salaire net de l'employee %d est %6.2f Euros",emp.N_id,salaire_net);
}
double calcul_salaire_net (struct employe *pt)
{ /* pt est un pointeur vers une structure * /
/ * du type d'employé */
    return(pt->paie_heure * pt->Nr_heures);
}
```

Passage par référence d'une structure

- Les opérateurs d'incrémentation et de décrémentation peuvent également être appliqués aux références de structure.

```
++pt->Nr_heures
```

- Cette instruction ajoute un au membre **Nr_heures** de la structure **employe**.
- L'opérateur −> a une priorité plus élevée que l'opérateur d'incrémentation, le membre **Nr_heures** est accédé en premier puis l'incrément est appliqué.

Retourner une structure

- Il est possible de renvoyer une structure par une fonction.
- L'exemple suivant montre cette opération:

```c
#include <stdio.h>
struct employe
{
    int N_id;
    double paie_heure;
    double Nr_heures;
};
main( )
{
    /* declare a global template */
    struct employe employe1;
    struct employe retour_val(void); /* prototype de la fonction */
    employe1 = retour_val( );
    printf ("\nLe numero d'identifant de l'employe est %d", employe1.N_id);
    printf("\nLe taux de paiement par heure est de %5.2f euros", employe1.paie_heure);
    printf("\nLe nombre d'heures de l'employe est %5.2f", employe1.Nr_heures);
}
struct employe retour_val( ) /*retour_val( ) retourne une structure employe*/
{
    struct employe nouveau;
    nouveau.N_id = 6789;
    nouveau.paie_heure = 16.25;
    nouveau.Nr_heures = 38.0;
    return (nouveau);
}
```

Structures dynamiques: listes chaînées

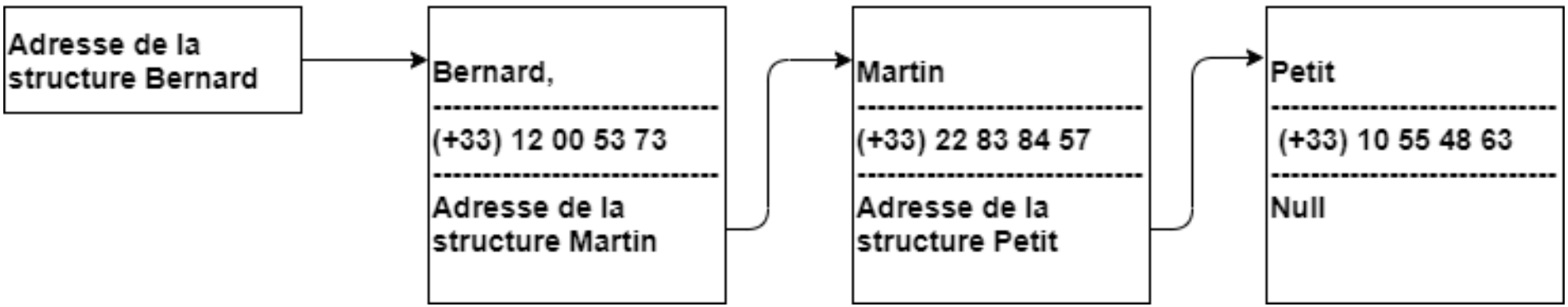
- Une liste chaînée fournit une méthode pratique pour maintenir une modification d'une liste, sans qu'il soit nécessaire de réorganiser et de restructurer continuellement la liste complète.

- Une liste chaînée est simplement un ensemble de structures dans lequel chaque structure contient au moins un membre dont la valeur est l'adresse de la ctructure suivante.

- Supposons que nous avons une liste téléphonique par ordre alphabétique comme suit:
  - Bernard, (+33) 12 00 53 73
  - Martin, (+33) 22 83 84 57
  - Petit, (+33) 10 55 48 63

Structures dynamiques: listes chaînées

- Nous cherchons à structurer notre liste comme suit:



Structures dynamiques: listes chaînées

```c
#include <stdio.h>
struct liste_tele
{
    char nom [30];
    char N_telephone[30];
    struct liste_tele *adr_suivante;
} ;
main ( )
{
    struct liste_tele t1 = {"Bernard","(+33) 12 00 53 73"};
    struct liste_tele t2 = {"Martin","(+33) 22 83 84 57"};
    struct liste_tele t3 = {"Petit","(+33) 10 55 48 63"} ;
    struct liste_tele *premiere_adr; // Un pointeur vers une structure
    premiere_adr = &t1; // Enregistre l'adresse de t1 dans premiere_adr
    t1.adr_suivante = &t2;  // Enregistre l'adresse de t2 dans t1.adr_suivante
    t2.adr_suivante = &t3; // Enregistre l'adresse de t3 dans t2.adr_suivante
    t3.adr_suivante = NULL; // Enregistre NULL dans t3.adr_suivante
    printf("\n%s \n%s \n%s",premiere_adr->nom,t1.adr_suivante->nom,
    t2.adr_suivante->nom);
}
```

Exemple d'utilisation des listes chaînées

```c
#include <stdio.h>
struct liste_tele
{
    char nom [30];
    char N_telephone[30];
    struct liste_tele *adr_suivante;
} ;
main ( )
{
    struct liste_tele t1 = {"Bernard","(+33) 12 00 53 73"};
    struct liste_tele t2 = {"Martin","(+33) 22 83 84 57"};
    struct liste_tele t3 = {"Petit","(+33) 10 55 48 63"} ;
    struct liste_tele *premiere_adr; // Un pointeur vers une structure
    void afficher(struct liste_tele *); // prototype de la fonction
    premiere_adr = &t1; // Enregistre l'adresse de t1 dans premiere_adr
    t1.adr_suivante = &t2;  // Enregistre l'adresse de t2 dans t1.adr_suivante
    t2.adr_suivante = &t3; // Enregistre l'adresse de t3 dans t2.adr_suivante
    t3.adr_suivante = NULL; // Enregistre NULL dans t3.adr_suivante
    afficher (premiere_adr) ; // Envoie l'adresse de la première structure
}

void afficher(struct liste_tele *C)// C est un pointeur vers une structure
{
    while (C != NULL) // affiche jusqu'à la fin de la liste chainée
    {
        printf("\n%-30s %-20s",C->nom, C->N_telephone);
        C = C->adr_suivante; // obtenir l'adresse suivante
    }
return;
}
```

Table des matières

La fonction malloc

- Le prototype de la fonction **malloc** (stdlib.h) est me suivant :

```
void * malloc (size_t taille)// (stdlib.h)
```

- Examinons le code suivant:

```
#include <stdlib.h>
char * adr ;
adr = malloc (50) ;
for (i=0 ; i<50 ; i++) *(adr+i) = 'x' ;
```

- alloue un emplacement de 50 octets et en fournit l'adresse en retour. Cette dernière est placée dans le pointeur **adr**.
- La boucle **for** montre un exemple d'utilisation de la zone créée.

La fonction malloc

- Dans l'exemple suivant, nous avons alloué une zone de **100 * sizeof(long)** octets.

```
long *adr ;
adr = malloc (100 * sizeof(long)) ;
for (i=0 ; i<100 ; i++) *(adr+i) = 1 ;
```

- Cela veut dire que l'espace mémoire est de 100 entiers de type long.
- **adr + i**: correspond à l'adresse contenue dans **adr**, augmentée de **sizeof(long)** fois la valeur de i (puisque adr pointe sur des objets de longueur sizeof(long)).

La fonction free

- Dans la la gestion dynamique de la mémoire, il est possible de récupérer des emplacements dont on n'a plus besoin.
- La fonction **free** nous permet de libérer un emplacement préalablement alloué.

```
#include <stdio.h>
#include <stdlib.h>
main()
{
char * adr1, * adr2, * adr3 ;
adr1 = malloc (100) ;
printf ("allocation dynamique de 100 octets en %p\n", adr1) ;
adr2 = malloc (50) ;
printf ("allocation de 50 octets en %p\n", adr2) ;
free (adr1) ;
printf ("liberation de 100 octets en %p\n", adr1) ;
adr3 = malloc (40) ;
printf ("allocation de 40 octets en %p\n", adr3) ;
}
```

Références