Initiation à l'algorithmique

Les structures

Mohamed MESSABIHI

mohamed.messabihi@gmail.com

Université de Tlemcen
Département d'informatique
1ère année MI

https://sites.google.com/site/informatiquemessabihi/

Introduction

Exercice :

On souhaite écrire un programme qui permet de gérer un ensemble d'étudiants. Chaque étudiant est décrit par son nom, son age et sa moyenne, écrire les fonctions suivantes :

1. Saisir les informations concernant un étudiant.
2. Afficher les informations concernant un étudiant.
3. Saisir un tableau d'étudiants.
4. afficher un tableau d'étudiants.
5. calculer la moyenne générale des étudiants.
6. Ajouter un attribut pour le numéro d'étudiant, et un autre pour son état (nouveau, répétitif ou endetté) et pour plus de précision, remplacer l'âge par la date de naissance.
7. Enfin séparer les étudiants en trois tableaux ; tab_nouveaux, tab_repetitifs et tab_endettes.

Introduction

Exercice :

On souhaite écrire un programme qui permet de gérer un ensemble d'étudiants. Chaque étudiant est décrit par son nom, son age et sa moyenne, écrire les fonctions suivantes :

1. Saisir les informations concernant un étudiant.
2. Afficher les informations concernant un étudiant.
3. Saisir un tableau d'étudiants.
4. afficher un tableau d'étudiants.
5. calculer la moyenne générale des étudiants.
6. Ajouter un attribut pour le numéro d'étudiant, et un autre pour son état (nouveau, répétitif ou endetté) et pour plus de précision, remplacer l'âge par la date de naissance.
7. Enfin séparer les étudiants en trois tableaux ; tab_nouveaux, tab_repetitifs et tab_endettes.

Introduction

Exercice :

On souhaite écrire un programme qui permet de gérer un ensemble d'étudiants. Chaque étudiant est décrit par son nom, son age et sa moyenne, écrire les fonctions suivantes :

1. Saisir les informations concernant un étudiant.
2. Afficher les informations concernant un étudiant.
3. Saisir un tableau d'étudiants.
4. afficher un tableau d'étudiants.
5. calculer la moyenne générale des étudiants.
6. Ajouter un attribut pour le numéro d'étudiant, et un autre pour son état (nouveau, répétitif ou endetté) et pour plus de précision, remplacer l'âge par la date de naissance.
7. Enfin séparer les étudiants en trois tableaux ; tab_nouveaux, tab_repetitifs et tab_endettes.

Introduction

Exercice :

On souhaite écrire un programme qui permet de gérer un ensemble d'étudiants. Chaque étudiant est décrit par son nom, son age et sa moyenne, écrire les fonctions suivantes :

1. Saisir les informations concernant un étudiant.
2. Afficher les informations concernant un étudiant.
3. Saisir un tableau d'étudiants.
4. afficher un tableau d'étudiants.
5. calculer la moyenne générale des étudiants.
6. Ajouter un attribut pour le numéro d'étudiant, et un autre pour son état (nouveau, répétitif ou endetté) et pour plus de précision, remplacer l'âge par la date de naissance.
7. Enfin séparer les étudiants en trois tableaux ; tab_nouveaux, tab_repetitifs et tab_endettes.

Introduction

Exercice :

On souhaite écrire un programme qui permet de gérer un ensemble d'étudiants. Chaque étudiant est décrit par son nom, son age et sa moyenne, écrire les fonctions suivantes :

1. Saisir les informations concernant un étudiant.
2. Afficher les informations concernant un étudiant.
3. Saisir un tableau d'étudiants.
4. afficher un tableau d'étudiants.
5. calculer la moyenne générale des étudiants.
6. Ajouter un attribut pour le numéro d'étudiant, et un autre pour son état (nouveau, répétitif ou endetté) et pour plus de précision, remplacer l'âge par la date de naissance.
7. Enfin séparer les étudiants en trois tableaux ; tab_nouveaux, tab_repetitifs et tab_endettes.

Introduction

Exercice :

On souhaite écrire un programme qui permet de gérer un ensemble d'étudiants. Chaque étudiant est décrit par son nom, son age et sa moyenne, écrire les fonctions suivantes :

1. Saisir les informations concernant un étudiant.
2. Afficher les informations concernant un étudiant.
3. Saisir un tableau d'étudiants.
4. afficher un tableau d'étudiants.
5. calculer la moyenne générale des étudiants.
6. Ajouter un attribut pour le numéro d'étudiant, et un autre pour son état (nouveau, répétitif ou endetté) et pour plus de précision, remplacer l'âge par la date de naissance.
7. Enfin séparer les étudiants en trois tableaux ; tab_nouveaux, tab_repetitifs et tab_endettes.

Introduction

Exercice :

On souhaite écrire un programme qui permet de gérer un ensemble d'étudiants. Chaque étudiant est décrit par son nom, son age et sa moyenne, écrire les fonctions suivantes :

1. Saisir les informations concernant un étudiant.
2. Afficher les informations concernant un étudiant.
3. Saisir un tableau d'étudiants.
4. afficher un tableau d'étudiants.
5. calculer la moyenne générale des étudiants.
6. Ajouter un attribut pour le numéro d'étudiant, et un autre pour son état (nouveau, répétitif ou endetté) et pour plus de précision, remplacer l'âge par la date de naissance.
7. Enfin séparer les étudiants en trois tableaux ; tab_nouveaux, tab_repetitifs et tab_endettes.

Introduction

Exercice :

On souhaite écrire un programme qui permet de gérer un ensemble d'étudiants. Chaque étudiant est décrit par son nom, son age et sa moyenne, écrire les fonctions suivantes :

1. Saisir les informations concernant un étudiant.
2. Afficher les informations concernant un étudiant.
3. Saisir un tableau d'étudiants.
4. afficher un tableau d'étudiants.
5. calculer la moyenne générale des étudiants.
6. Ajouter un attribut pour le numéro d'étudiant, et un autre pour son état (nouveau, répétitif ou endetté) et pour plus de précision, remplacer l'âge par la date de naissance.
7. Enfin séparer les étudiants en trois tableaux ; tab_nouveaux, tab_repetitifs et tab_endettes.

Les structures (enregistrements)

- Une structure (ou enregistrement) est une structure de données consistant en un nombre fixé de composants, appelés champs.
- Á la différence du tableau, ces composants ne sont pas obligatoirement du même type, et ne sont pas indexés.
- La définition du type enregistrement précise pour chaque composant un identificateur de champ, dont la portée est limitée à l'enregistrement, et le type de ce champ .

```
struct Nom_Structure
{
    int champ1;
    char champ2;
    double champ3;
    char[20] champ4;
};
```

☞Attention

Le point-virgule après l'accolade fermante est obligatoire.

Les structures (enregistrements)

- Une structure (ou enregistrement) est une structure de données consistant en un nombre fixé de composants, appelés champs.
- Á la différence du tableau, ces composants ne sont pas obligatoirement du même type, et ne sont pas indexés.
- La définition du type enregistrement précise pour chaque composant un identificateur de champ, dont la portée est limitée à l'enregistrement, et le type de ce champ .

```
struct Nom_Structure
{
    int champ1;
    char champ2;
    double champ3;
    char[20] champ4;
};
```

☞Attention

Le point-virgule après l'accolade fermante est obligatoire.

Les structures (enregistrements)

- Une structure (ou enregistrement) est une structure de données consistant en un nombre fixé de composants, appelés champs.
- Á la différence du tableau, ces composants ne sont pas obligatoirement du même type, et ne sont pas indexés.
- La définition du type enregistrement précise pour chaque composant un identificateur de champ, dont la portée est limitée à l'enregistrement, et le type de ce champ .

```
struct Nom_Structure
{
     int champ1;
     char champ2;
     double champ3;
     char[20] champ4;
};
```

☞Attention

Le point-virgule après l'accolade fermante est obligatoire.

Les structures (enregistrements)

- Une structure (ou enregistrement) est une structure de données consistant en un nombre fixé de composants, appelés champs.
- Á la différence du tableau, ces composants ne sont pas obligatoirement du même type, et ne sont pas indexés.
- La définition du type enregistrement précise pour chaque composant un identificateur de champ, dont la portée est limitée à l'enregistrement, et le type de ce champ .

```
struct Nom_Structure
{
     int champ1;
     char champ2;
     double champ3;
     char[20] champ4;
};
```

☞Attention

Le point-virgule après l'accolade fermante est obligatoire.

Les types structurés (enregistrements)

- Une fois qu'on a défini un type structuré, on peut déclarer des variables enregistrements exactement de la même façon que l'on déclare des variables d'un type primitif.

```
struct Etudiant {
    char nom[30];
    int age;
    float moyenne;
};

void main(){
    struct Etudiant e1, e2;
}
```

☞Faut-il obligatoirement écrire le mot-clé **struct** lors de la définition de la variable ?

Les types structurés (enregistrements)

- Une fois qu'on a défini un type structuré, on peut déclarer des variables enregistrements exactement de la même façon que l'on déclare des variables d'un type primitif.

```
struct Etudiant {
    char nom[30];
    int age;
    float moyenne;
};

void main(){
    struct Etudiant e1, e2;
}
```

☞Faut-il obligatoirement écrire le mot-clé **struct** lors de la définition de la variable ?

Le typedef

- L'instruction appelée **typedef** permet de créer de nouveaux noms de types (autrement dit, elle sert à créer un alias de structure) :

```
typedef float reel ;
typedef struct Etudiant Etudiant ;

struct Etudiant {
    char nom[30];
    int age;
    reel moyenne;
};

void main(){
        reel r;
    Etudiant e1, e2;
}
```

- **typedef** : permet de créer un alias de structure ;
- **struct Etudiant** : c'est le nom de la structure pour laquelle on veut créer un alias (c'est-à-dire un « équivalent ») ;
- **Etudiant** : c'est le nom de l'équivalent.

Le typedef

- L'instruction appelée **typedef** permet de créer de nouveaux noms de types (autrement dit, elle sert à créer un alias de structure) :

```
typedef float reel ;
typedef struct Etudiant Etudiant ;

struct Etudiant {
    char nom[30];
    int age;
    reel moyenne;
};

void main(){
        reel r;
    Etudiant e1, e2;
}
```

- **typedef** : permet de créer un alias de structure ;
- **struct Etudiant** : c'est le nom de la structure pour laquelle on veut créer un alias (c'est-à-dire un « équivalent ») ;
- **Etudiant** : c'est le nom de l'équivalent.

Le typedef

- L'instruction appelée **typedef** permet de créer de nouveaux noms de types (autrement dit, elle sert à créer un alias de structure) :

```
typedef float reel ;
typedef struct Etudiant Etudiant ;

struct Etudiant {
    char nom[30];
    int age;
    reel moyenne;
};

void main(){
        reel r;
    Etudiant e1, e2;
}
```

- **typedef** : permet de créer un alias de structure ;
- **struct Etudiant** : c'est le nom de la structure pour laquelle on veut créer un alias (c'est-à-dire un « équivalent ») ;
- **Etudiant** : c'est le nom de l'équivalent.

Le typedef

- L'instruction appelée **typedef** permet de créer de nouveaux noms de types (autrement dit, elle sert à créer un alias de structure) :

```
typedef float reel ;
typedef struct Etudiant Etudiant ;

struct Etudiant {
    char nom[30];
    int age;
    reel moyenne;
};

void main(){
        reel r;
    Etudiant e1, e2;
}
```

- **typedef** : permet de créer un alias de structure ;
- **struct Etudiant** : c'est le nom de la structure pour laquelle on veut créer un alias (c'est-à-dire un « équivalent ») ;
- **Etudiant** : c'est le nom de l'équivalent.

Manipulation des champs d'une structure

- La manipulation d'une structure se fait au travers de ses champs.
- Les champs d'une structure sont accessibles à travers leur nom, grâce à l'opérateur '.'

```
void main()
{
    Etudiant e1, e2;

    puts("Donnez votre nom : ");
    scanf("%s", &e1.nom);
    puts("Donnez votre age : ");
    scanf("%d", &e1.age);
    puts("Donnez votre moyenne : ");
    scanf("%f", &e1.moyenne);
    e2=e1;
}
```

- Comme pour les tableaux, il n'est pas possible de manipuler une structure globalement, sauf pour affecter une structure à un autre de même type (Par exemple : e2=e1 ;).
- Par exemple, pour afficher une structure il faut afficher tous ses champs un par un.

Manipulation des champs d'une structure

- La manipulation d'une structure se fait au travers de ses champs.
- Les champs d'une structure sont accessibles à travers leur nom, grâce à l'opérateur '.'

```
void main()
{
    Etudiant e1, e2;

    puts("Donnez votre nom : ");
    scanf("%s", &e1.nom);
    puts("Donnez votre age : ");
    scanf("%d", &e1.age);
    puts("Donnez votre moyenne : ");
    scanf("%f", &e1.moyenne);
    e2=e1;
}
```

- Comme pour les tableaux, il n'est pas possible de manipuler une structure globalement, sauf pour affecter une structure à un autre de même type (Par exemple : e2=e1 ;).
- Par exemple, pour afficher une structure il faut afficher tous ses champs un par un.

Manipulation des champs d'une structure

- La manipulation d'une structure se fait au travers de ses champs.
- Les champs d'une structure sont accessibles à travers leur nom, grâce à l'opérateur '.'

```
void main()
{
    Etudiant e1, e2;

    puts("Donnez votre nom : ");
    scanf("%s", &e1.nom);
    puts("Donnez votre age : ");
    scanf("%d", &e1.age);
    puts("Donnez votre moyenne : ");
    scanf("%f", &e1.moyenne);
    e2=e1;
}
```

- Comme pour les tableaux, il n'est pas possible de manipuler une structure globalement, sauf pour affecter une structure à un autre de même type (Par exemple : e2=e1 ;).
- Par exemple, pour afficher une structure il faut afficher tous ses champs un par un.

Initialiser une structure

- l'initialisation d'une structure ressemble un peu ressembler à celle d'un tableau.

- En effet, on peut initialiser une structure au moment de sa déclaration :

```
void main(){
    Etudiant e = {"Toto", 19, 12.65}  ;
}
```

- Sinon, on pour créer une fonction **initialiserEtudiant** qui se charge de faire l'initialisation d'une variable de type Etudiant.

- Mais pour pouvoir faire cela il faut envoyer un pointeur de la variable structure à la fonction initialiserEtudiant.

Initialiser une structure

- l'initialisation d'une structure ressemble un peu ressembler à celle d'un tableau.
- En effet, on peut initialiser une structure au moment de sa déclaration :

```
void main(){
    Etudiant e = {"Toto", 19, 12.65}  ;
}
```

- Sinon, on pour créer une fonction **initialiserEtudiant** qui se charge de faire l'initialisation d'une variable de type Etudiant.
- Mais pour pouvoir faire cela il faut envoyer un pointeur de la variable structure à la fonction initialiserEtudiant.

Initialiser une structure

- l'initialisation d'une structure ressemble un peu ressembler à celle d'un tableau.

- En effet, on peut initialiser une structure au moment de sa déclaration :

```
void main(){
    Etudiant e = {"Toto", 19, 12.65}   ;
}
```

- Sinon, on pour créer une fonction **initialiserEtudiant** qui se charge de faire l'initialisation d'une variable de type Etudiant.

- Mais pour pouvoir faire cela il faut envoyer un pointeur de la variable structure à la fonction initialiserEtudiant.

Pointeur de structure

- Un pointeur de structure se crée de la même manière qu'un pointeur de int, de double ou de n'importe quelle autre type de base

- En effet, on peut faire à la déclaration de la variable :

```
void initialiserEtudiant(Etudiant *e){
    (*e).nom = "";
    (*e).age = 0;
    (*e).moyenne = 0;
}
void main(){
    Etudiant e1, *e2=NULL;
    initialiserEtudiant(&e1);
    initialiserEtudiant(e2);
}
```

- Il faut placer des parenthèses autour de *e, car *e.age et (*e).age sont deux écritures différentes.

Pointeur de structure

- Un pointeur de structure se crée de la même manière qu'un pointeur de int, de double ou de n'importe quelle autre type de base

- En effet, on peut faire à la déclaration de la variable :

```
void initialiserEtudiant(Etudiant *e){
    (*e).nom = "";
    (*e).age = 0;
    (*e).moyenne = 0;
}
void main(){
    Etudiant e1, *e2=NULL;
    initialiserEtudiant(&e1);
    initialiserEtudiant(e2);
}
```

- Il faut placer des parenthèses autour de *e, car *e.age et (*e).age sont deux écritures différentes.

Pointeur de structure : une autre notation

- Vu qu'en programmation on manipulera très souvent des pointeurs de structures, le langage C nous offre un raccourci très pratique et très utilisé.
- Ce raccourci consiste à former une flèche avec un tiret suivi d'un chevron > (par exemple (*e).age devient e->age).

```
void initialiserEtudiant(Etudiant *e){
    strcpy((e->nom,"");
    e->age = 0;
    e->moyenne = 0;
}
void main(){
    Etudiant e1, *e2=&e1;
    e1.moyenne = 10; // une variable: on utilise le "point"
    e2->moyenne = 12; //un pointeur: on utilise la fleche
}
```

- Il ne faut surtout pas confondre la flèche avec le « point ».
- En effet, la flèche est réservée aux pointeurs, le « point » est réservé aux variables.

Pointeur de structure : une autre notation

- Vu qu'en programmation on manipulera très souvent des pointeurs de structures, le langage C nous offre un raccourci très pratique et très utilisé.
- Ce raccourci consiste à former une flèche avec un tiret suivi d'un chevron > (par exemple (*e).age devient e->age).

```
void initialiserEtudiant(Etudiant *e){
    strcpy((e->nom,"");
    e->age = 0;
    e->moyenne = 0;
}
void main(){
    Etudiant e1, *e2=&e1;
    e1.moyenne = 10; // une variable: on utilise le "point"
    e2->moyenne = 12; //un pointeur: on utilise la fleche
}
```

- Il ne faut surtout pas confondre la flèche avec le « point ».
- En effet, la flèche est réservée aux pointeurs, le « point » est réservé aux variables.

Pointeur de structure : une autre notation

- Vu qu'en programmation on manipulera très souvent des pointeurs de structures, le langage C nous offre un raccourci très pratique et très utilisé.
- Ce raccourci consiste à former une flèche avec un tiret suivi d'un chevron > (par exemple (*e).age devient e->age).

```
void initialiserEtudiant(Etudiant *e){
    strcpy((e->nom,"");
    e->age = 0;
    e->moyenne = 0;
}
void main(){
    Etudiant e1, *e2=&e1;
    e1.moyenne = 10; // une variable: on utilise le "point"
    e2->moyenne = 12; //un pointeur: on utilise la fleche
}
```

- Il ne faut surtout pas confondre la flèche avec le « point ».
- En effet, la flèche est réservée aux pointeurs, le « point » est réservé aux variables.

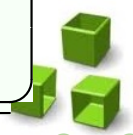
Les structures imbriquées

- Un champ dans une structure peut lui-même être **structure**.

```
typedef struct Date Date ;
struct Date {
    int jour;
    int mois;
    int annee;
};

typedef struct Etudiant Etudiant ;
struct Etudiant {
    char nom[30];
    int age;
    float moyenne;
    Date date_naissance ;
};
```

Retour à l'exercice : Saisir une variable de type Etudiant

```
void saisie_Date(Date *d){
    puts("Date de naissance");
    puts("Donnez le jour : ");
    scanf("%d", &d->jour);
    puts("Donnez le mois : ");
    scanf("%d", &d->mois);
    puts("Donnez l'annee : ");
    scanf("%d", &d->annee);

}
void saisie(Etudiant *e){
    puts("Donnez le nom : ");
    scanf("%s", &e->nom);
    puts("Donnez l'age : ");
    scanf("%d", &e->age);
    puts("Donnez la moyenne : ");
    scanf("%f", &e->moyenne);

    saisie_Date(&e->date_naissance);
}
```



Retour à l'exercice : Afficher une variable de type Etudiant

```
void dateToString(Date d, char s[12]){
   sprintf(s, "%d/%d/%d", d.jour, d.mois, d.annee);
}

void afficher(Etudiant e){
    char s[12];
    dateToString(e.date_naissance, s);
    printf("[ Nom : %s, Age : %d , Moyenne : %2.2f, Date de
        naissance : %s, Etat : %s ] \n", e.nom, e.age, e.
        moyenne, s , etat_ToString(e));

}
```

Retour à l'exercice : Afficher une variable de type Etudiant (une autre version)

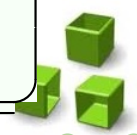
```
char* date_ToString(Date d){
    char *s;
    s = malloc (sizeof (*s) * 20);
    sprintf(s, "%d/%d/%d", d.jour, d.mois, d.annee);
    return s;
}

void afficher(Etudiant e){
    printf("[ Nom : %s, Age : %d , Moyenne : %2.2f, Date de
        naissance : %s ] \n", e.nom, e.age, e.moyenne,
        date_ToString(e.date_naissance));

}
```

Retour à l'exercice : Saisir et afficher plusieurs étudiants

```
void saisie_tab(Etudiant tab[100], int N){
    int i=0;
    for (i=0; i<N; i++)
    {
        printf("--- Etudiant N: %d ---\n", i+1);
        saisie(&tab[i]);
    }
}

void afficher_tab(Etudiant tab[100], int N){
    int i=0;
    for (i=0; i<N; i++)
    {
        afficher(tab[i]);
    }
}
```

Retour à l'exercice : Calculer la moyenne de plusieurs étudiants

```
float moyenne_Etudiants(Etudiant tab[], int N)
{
    int i=0;
    float somme=0;
    float moyenne = 0;
    for (i=0; i<N; i++)
    {
        somme = somme + tab[i].moyenne;
    }
    moyenne = somme/N;
    return moyenne;
}

void main(){
    Etudiant tab_Etudiants[100];
    saisie_tab(tab_Etudiants, 3);
    afficher_tab(tab_Etudiants, 3);
    printf("la moyenne des etudiants est : %lf",
        moyenne_Etudiants(tab_Etudiants, 3));
}
```



Types énumérés

- Au lieu de donner des valeurs conventionnelles à des données symboliques, on peut utiliser les types énumérés.

```
typedef enum Jours Jour;
typedef enum Feux Feux;
typedef enum Etat_Etudiant Etat_Etudiant;

enum Jours {DIMANCHE, LUNDI, MARDI, MERCREDI, JEUDI,
    VENDREDI, SAMEDI};
enum Feux {ROUGE, ORANGE, VERT};
enum Etat_Etudiant {Nouveau, Repetitif, Endette};
```

- Un type énuméré est un type dont les variables associées n'auront qu'un nombre très limité de valeurs (au maximum 256 différentes possibles).
- La définition d'un type énuméré consiste à déclarer une liste de valeurs possibles associées à un type.

Types énumérés

- Au lieu de donner des valeurs conventionnelles à des données symboliques, on peut utiliser les types énumérés.

```
typedef enum Jours Jour;
typedef enum Feux Feux;
typedef enum Etat_Etudiant Etat_Etudiant;

enum Jours {DIMANCHE, LUNDI, MARDI, MERCREDI, JEUDI,
    VENDREDI, SAMEDI};
enum Feux {ROUGE, ORANGE, VERT};
enum Etat_Etudiant {Nouveau, Repetitif, Endette};
```

- Un type énuméré est un type dont les variables associées n'auront qu'un nombre très limité de valeurs (au maximum 256 différentes possibles).
- La définition d'un type énuméré consiste à déclarer une liste de valeurs possibles associées à un type.

Types énumérés

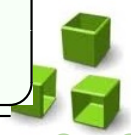
- Au lieu de donner des valeurs conventionnelles à des données symboliques, on peut utiliser les types énumérés.

```
typedef enum Jours Jour;
typedef enum Feux Feux;
typedef enum Etat_Etudiant Etat_Etudiant;

enum Jours {DIMANCHE, LUNDI, MARDI, MERCREDI, JEUDI,
    VENDREDI, SAMEDI};
enum Feux {ROUGE, ORANGE, VERT};
enum Etat_Etudiant {Nouveau, Repetitif, Endette};
```

- Un type énuméré est un type dont les variables associées n'auront qu'un nombre très limité de valeurs (au maximum 256 différentes possibles).
- La définition d'un type énuméré consiste à déclarer une liste de valeurs possibles associées à un type.

Types énumérés : exemple d'utilisation

```
typedef enum Etat_Etudiant Etat_Etudiant;
enum Etat_Etudiant  {NOUVEAU, REPETITIF, ENDETTE};

typedef struct Etudiant Etudiant ;
struct Etudiant {
    char nom[30];
    int age;
    float moyenne;
    Date date_naissance ;
    Etat_Etudiant etat ;
};
void separer_tabs(Etudiant tab[100], int N){
    int i=0, e=0, n=0, r=0;
    for (i=0; i<N; i++)
    {
        switch(tab[i].etat){
            case 0 : tab_nouveaux[n] = tab[i]; n++; break;
            case 1 : tab_repetitifs[r] = tab[i]; r++; break;
            case 2 : tab_endettes[e] = tab[i]; e++; break;
        }
    }
}
```



Résumé

- Une structure est un type de variable personnalisé qu'on peut créer et utiliser dans des programmes. C'est au programmeur de le définir, contrairement aux types de base tels que **int** et **double**,...
- Une structure est composée de « sous-variables » qui sont en général des variables de type de base comme int et double, mais aussi des tableaux ou d'autres structures.
- On accède à un des composants de la structure en séparant le nom de la variable et la composante d'un point : e.nom
- Si on manipule un pointeur de structure et qu'on veut accéder à une des composantes, on utilise une flèche à la place du point : ptrE->nom.
- Une énumération est un type de variable personnalisé qui peut seulement prendre une des valeurs prédéfinies : rouge, orange ou vert par exemple.

Résumé

- Une structure est un type de variable personnalisé qu'on peut créer et utiliser dans des programmes. C'est au programmeur de le définir, contrairement aux types de base tels que **int** et **double**,...
- Une structure est composée de « sous-variables » qui sont en général des variables de type de base comme int et double, mais aussi des tableaux ou d'autres structures.
- On accède à un des composants de la structure en séparant le nom de la variable et la composante d'un point : e.nom
- Si on manipule un pointeur de structure et qu'on veut accéder à une des composantes, on utilise une flèche à la place du point : ptrE->nom.
- Une énumération est un type de variable personnalisé qui peut seulement prendre une des valeurs prédéfinies : rouge, orange ou vert par exemple.

Résumé

- Une structure est un type de variable personnalisé qu'on peut créer et utiliser dans des programmes. C'est au programmeur de le définir, contrairement aux types de base tels que **int** et **double**,...
- Une structure est composée de « sous-variables » qui sont en général des variables de type de base comme int et double, mais aussi des tableaux ou d'autres structures.
- On accède à un des composants de la structure en séparant le nom de la variable et la composante d'un point : e.nom
- Si on manipule un pointeur de structure et qu'on veut accéder à une des composantes, on utilise une flèche à la place du point : ptrE->nom.
- Une énumération est un type de variable personnalisé qui peut seulement prendre une des valeurs prédéfinies : rouge, orange ou vert par exemple.

Résumé

- Une structure est un type de variable personnalisé qu'on peut créer et utiliser dans des programmes. C'est au programmeur de le définir, contrairement aux types de base tels que **int** et **double**,...
- Une structure est composée de « sous-variables » qui sont en général des variables de type de base comme int et double, mais aussi des tableaux ou d'autres structures.
- On accède à un des composants de la structure en séparant le nom de la variable et la composante d'un point : e.nom
- Si on manipule un pointeur de structure et qu'on veut accéder à une des composantes, on utilise une flèche à la place du point : ptrE->nom.
- Une énumération est un type de variable personnalisé qui peut seulement prendre une des valeurs prédéfinies : rouge, orange ou vert par exemple.

Résumé

- Une structure est un type de variable personnalisé qu'on peut créer et utiliser dans des programmes. C'est au programmeur de le définir, contrairement aux types de base tels que **int** et **double**,...
- Une structure est composée de « sous-variables » qui sont en général des variables de type de base comme int et double, mais aussi des tableaux ou d'autres structures.
- On accède à un des composants de la structure en séparant le nom de la variable et la composante d'un point : e.nom
- Si on manipule un pointeur de structure et qu'on veut accéder à une des composantes, on utilise une flèche à la place du point : ptrE->nom.
- Une énumération est un type de variable personnalisé qui peut seulement prendre une des valeurs prédéfinies : rouge, orange ou vert par exemple.