Sommaire

Prérequis

1. Comprendre ce qu'est une variable, et savoir les manipuler.
2. Savoir déclarer un tableau et l'utiliser.
3. Savoir concevoir et manipuler des fonctions.
4. Être à l'aise avec la portée des variables.
5. Avoir compris la notion de pointeur.

Manipulation des variables

- Types de base du langage C.
- Restriction très forte.
- Explosion du nombre d'arguments dans les fonctions.
- Programmes difficiles à réaliser. Modélisation complexe.

Phase de modélisation. Avant de réaliser un programme, il est nécessaire de passer par une phase dite de modélisation. Une phase cruciale qui permet de décrire les objets que nous allons manipuler, et comment nous allons les manipuler.

Exemple. Un carnet d'adresses.

- On souhaite manipuler des contacts
- Chaque contact nécessite un certain nombre d'informations:
  - Nom
  - Prénom
  - Mail
  - Date de naissance
- Il est donc naturel de vouloir manipuler un contact plutôt que les diverses informations.

Le langage C permet de définir des nouveaux types de données (que l'on dit structurés, composés) via les structures. C'est grâce à ce mécanismes que nous pourrons manipuler des variables qui correspondent à la réalité.

Sommaire

Définition d'un type de données structurées.

Structure

En langage C, une structure est un type composé de plusieurs variables. Les différentes variables qui composent la structure sont appelées « champs ». Définir une structure revient à définir un nouveau type de données.

```
1  struct nomDeLaStructure
2  {
3      typeDuChamp_1 nomDuChamp_1;
4      typeDuChamp_2 nomDuChamp_2;
5      typeDuChamp_3 nomDuChamp_3;
6      typeDuChamp_4 nomDuChamp_4;
7      ...
8  };
```

Définition d'un type de données structurées.

1. La dernière accolade doit être suivie d'un point-virgule.
2. L'usage veut qu'une telle définition soit placée juste après les directives du préprocesseur.
3. Le nom des champs respecte les critères des noms de variable.
4. Deux champs d'une même structure ne peuvent avoir le même nom.
5. Les champs peuvent être de n'importe quel type hormis le type de la structure dans laquelle elles se trouvent.

Exemple de définition.

Dans cet exemple, nous allons définit un type structuré permettant de manipuler des nombres rationnels. Nous avons donc besoin :

1. D'un numérateur (un entier).
2. D'un dénominateur (un entier).

Ce qui nous donne :

```
1  struct fraction
2  {
3      int num;
4      int den;
5  };
```

Important : Le type que nous venons de définir est 'struct fraction' et non pas 'fraction'.

Sommaire

Déclaration d'une variable.

Le type que nous venons de définir est 'struct fraction'. Donc pour déclarer une variable de ce type, il suffit de faire comme ceci :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct fraction
5  {
6      int num;
7      int den;
8  };
9
10 int main(void)
11 {
12     struct fraction toto;
13     // Ici le reste du code
14
15     return EXIT_SUCCESS;
16 }
```

modification des champs d'une variable.

- Une variable structurée peut être manipulée comme n'importe quelle autre variable.
- Reste encore le problème de l'accès aux différents champs de la variable.

Accès à un champs d'une variable structurée

Pour accéder à un champs d'une variable structurée, il suffit de faire suivre le nom de la variable par un point puis par le nom du champs concerné.

modification des champs d'une variable.

Exemple.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   struct fraction
5   {
6       int num;
7       int den;
8   };
9
10  int main(void)
11  {
12      struct fraction toto;
13
14      toto.num=2;
15      toto.den=5;
16
17      return EXIT_SUCCESS;
18  }
```

Chaque champs peut être manipulé comme n'importe quelle variable.

modification des champs d'une variable.

Exemple 2.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   struct fraction
5   {
6       int num;
7       int den;
8   };
9
10  int main(void)
11  {
12      struct fraction toto, autre;
13
14      toto.num=2;
15      toto.den=5;
16      autre.num=3;
17      autre.den=1;
18
19      printf("Fraction 1 : %d/%d\nSeconde fraction : %d/%d",toto.num,toto.den,
            autre.num,autre.den);
20
21      return EXIT_SUCCESS;
22  }
```

Initialisation d'une variable.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct fraction
5  {
6      int num;
7      int den;
8  };
9
10 int main(void)
11 {
12     struct fraction toto={2,5}, autre={3,1};
13
14     printf("Fraction 1 : %d/%d\nSeconde fraction : %d/%d",toto.num,toto.den,
           autre.num,autre.den);
15
16     return EXIT_SUCCESS;
17 }
```

Sommaire

Opérations globales sur des variables structurées.

Les opérations suivantes sont possibles sur les variable structurées :

1. L'affectation d'un variable structurée par une autre variable du même type.
2. Le test d'égalité ou d'inégalité entre deux variables structurées du même type.
3. Le retour d'une variable structurée par une fonction.
4. Le passage en argument d'une variable structurée à une fonction.

Affectation.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct fraction
5  {
6      int num;
7      int den;
8  };
9
10 int main(void)
11 {
12     struct fraction toto={2,5}, autre;
13     autre = toto;
14     printf("Fraction 1 : %d/%d\nSeconde fraction : %d/%d",toto.num,toto.den,
           autre.num,autre.den);
15
16     return EXIT_SUCCESS;
17 }
```

Test d'égalité.

```c
#include <stdio.h>
#include <stdlib.h>

struct fraction
{
    int num;
    int den;
};

int main(void)
{
    struct fraction toto={2,5}, autre={1,2};
    if(autre == toto)
    {
        printf("Egaux.");
    }
    else
    {
        printf("Differents.");
    }

    return EXIT_SUCCESS;
}
```

Exemple de retour de fonction.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct fraction
5  {
6      int num;
7      int den;
8  };
9
10 struct fraction creation(int a, int b)
11 {
12     struct fraction tmp;
13     tmp.num = a;
14     tmp.den = b;
15     return tmp;
16 }
17 int main(void)
18 {
19     struct fraction toto;
20     toto = creation(3,6);
21     printf("Fraction 1 : %d/%d",toto.num,toto.den);
22
23     return EXIT_SUCCESS;
24 }
```

Exemple de passage de paramètre de fonction.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct fraction
5  {
6      int num;
7      int den;
8  };
9
10 struct fraction change(struct fraction f)
11 {
12     f.num = f.num + 1;
13     f.den = f.den + 1;
14     return f;
15 }
16 int main(void)
17 {
18     struct fraction toto={3,6};
19     toto = change(toto);
20     printf("Fraction : %d/%d",toto.num,toto.den);
21
22     return EXIT_SUCCESS;
23 }
```

Sommaire

Tableau de variables structurées

Tableau de variables structurées

Il est tout à fait possible de déclarer un tableau dont le type des données serait structuré. Ainsi, chaque case du tableau sera une variable structurée.

Tableau de variables structurées

```c
#include <stdio.h>
#include <stdlib.h>

struct fraction
{
    int num;
    int den;
};

struct fraction creation(int a, int b)
{
    struct fraction tmp={a,b};
    return tmp;
}
int main(void)
{
    int i;
    struct fraction tableau[10];
    for(i=0;i<10;i++)
    {
        tableau[i] = creation(i,3);
        printf("Fraction %d : %d/%d\n",i,tableau[i].num,tableau[i].den);
    }
    return EXIT_SUCCESS;
}
```

Sommaire

Les codeurs sont des fainéants

En définissant un type structuré, le nom du type contient toujours 'struct '. C'est vraiment très long à taper.

Le mot clé typedef

typedef permet de créer un alias pour un type de données. L'usage est le suivant :

```
1 typedef type nouveau_type;
```

Il sera ensuite possible de faire appel à chacun des deux types indifféremment !

Exemple d'utilisation de typedef

```c
#include <stdio.h>
#include <stdlib.h>

struct fraction
{
    int num;
    int den;
};
typedef struct fraction frac;

frac creation(int a, int b)
{
    frac tmp={a,b};
    return tmp;
}
int main(void)
{
    int i;
    frac tableau[10];
    for(i=0;i<10;i++)
    {
        tableau[i] = creation(i,3);
        printf("Fraction %d : %d/%d\n",i,tableau[i].num,tableau[i].den);
    }
    return EXIT_SUCCESS;
}
```

Sommaire

Les structures et les pointeurs

Les variables structurées peuvent être manipulées comme bon nous semble. En particulier, nous pouvons définir des pointeurs de ce type.

Exemple de pointeur de structure

```c
#include<stdio.h>
#include<stdlib.h>

struct fraction
{
    int num;
    int den;
};
typedef struct fraction frac;

int main(void)
{
    frac f1={2,3};
    frac f2={1,2};
    frac *pt=NULL;
    pt =&f1;
    return 0;
}
```

Opérateur d'indirection

L'opérateur d'indirection, appliqué sur un pointeur permet d'accéder au contenu situé à l'adresse pointé. Ainsi, dans le cas d'une donnée structurée, il ne reste plus qu'à accéder au champs souhaité.

Exemple d'utilisation de l'opérateur d'indirection

```c
#include<stdio.h>
#include<stdlib.h>

struct fraction
{
    int num;
    int den;
};
typedef struct fraction frac;

int main(void)
{
    frac f1={2,3};
    frac f2={1,2};
    frac *pt=NULL;
    pt =&f1;
    printf("Voici la fraction f1 : %d/%d",f1.num,f1.den);
    (*pt).num=9;
    (*pt).den=(*pt).num;
    printf("\nVoici la fraction f1 : %d/%d",f1.num,f1.den);

    return EXIT_SUCCESS;
}
```

Du sucre syntaxique. Encore.

Pour accéder au champs d'une variable structurée par un pointeur, il faut utiliser l'opérateur d'indirection :

```
1 (*pt).num=9;
```

Cependant, cela implique 4 caractères :

1. les deux parenthèses
2. L'opérateur d'indirection
3. L'opérateur . pour accéder au champs

Il existe une notation permettant de n'utiliser que 2 caractères en utilisant - et > :

```
1 pt->num=9;
```

Ces deux notations sont strictement équivalentes.

```c
#include<stdio.h>
#include<stdlib.h>

struct fraction
{
    int num;
    int den;
};
typedef struct fraction frac;

int main(void)
{
    frac f1={2,3};
    frac f2={1,2};
    frac *pt=NULL;
    frac *pt2=NULL;
    pt =&f1;
    pt2=&f2;
    printf("Voici la fraction f1 : %d/%d",f1.num,f1.den);
    printf("\nVoici la fraction f2 : %d/%d",f2.num,f2.den);
    (*pt).num=9;
    (*pt).den=(*pt).num;
    pt2->num=5;
    pt2->den=1;
    printf("\nVoici la fraction f1 : %d/%d",f1.num,f1.den);
    printf("\nVoici la fraction f2 : %d/%d",f2.num,f2.den);

    return EXIT_SUCCESS;
}
```