PART 3 : LES STRUCTURES

1. Structure - C’est quoi?
2. Définition d'une structure
3. Déclarations de variables structures
4. Structures imbriquées
5. Tableaux de structures
6. Manipulation des structures
7. Structures et pointeurs
8. Structures et fonctions

Structures - C’est quoi?

- Une structure permet de rassembler sous un même nom des informations de types différents.

- Une structure peut contenir des donnés entières, flottantes, tableaux , caractères, pointeurs, etc... Ces données sont appelés les membres de la structure.

- **Exemple** : fiche d'indentification d'un personne
  - nom, prénom, âge, liste des diplômes, etc...

Définition d'une structure

- Déclaration d'une structure : syntaxe

```
struct nomdelastructure
{
        typemembre1 nommembre1 ;
        typemembre2 nommembre2 ;
        …
        typemembren nommembren ;
}
```

- Exemple : compte bancaire (compte, etat, nom et solde)

```
struct compte
{
        int no_compte ;
        char etat ;
        char nom[80];
        float solde;
};
struct compte a,b,c;
/*déclaration de 3 variables de ce type*/
```

Déclarations de variables structures (1)

- Autres façons de déclarer des variables structure

1
```
struct compte {
        int no_compte ;
        char etat ;
        char nom[80];
        float solde;
        } a, b; /*déclaration de 2 variables de ce type*/

struct compte c; /*déclaration de 1 variable de ce type*/
```

2
```
struct  { /* le nom de la structure est facultatif */
        int no_compte ;
        char etat ;
        char nom[80];
        float solde;
        } a,b,c;/*déclaration de variables de ce type ici */
/* mais plus de possibilité de déclarer d'autres variables de ce
type*/
```

Déconseillé

Déclarations de variables (2)

3

```
typedef struct compte {
        int no_compte ;
        char etat ;
        char nom[80];
        float solde;
        } cpt ;
/* cpt est alors un type équivalent à struct  compte*/

cpt a,b,c;   /*déclaration de variables de ce type*/
```

Recommandé

4

```
typedef struct {
        int no_compte ;
        char etat ;
        char nom[80];
        float solde;
        } cpt ;
```

Structures imbriquées

- Une structure peut être membre d'une autre structure

```
struct date {
              int jour;
              int mois;
              int annee;
            };

struct compte {
            int no_compte ;
            char etat ;
            char nom[80];
            float solde;
            struct date dernier_versement;
                 };
```

- **Remarque : L'ordre de déclaration des structures est important**

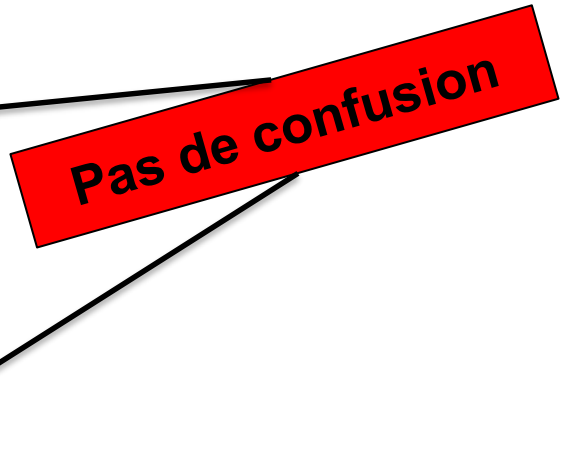
Tableaux de structures

```
struct compte client[100];
```

- La portée du nom d'un membre est limité à la structure dans laquelle il est défini.

- On peut avoir des membres homonymes dans des structures distinctes.

```
struct s1 {
        float x;
        int y ;
        };

struct s2 {
        char x;
        float y;
        };
```



Manipulation des structures (1)

- Initialisation à la compilation:

```
struct compte {
        int no_compte ;
        char etat ;
        char nom[80];
        float solde;
        struct date dernier_versement;
        };

struct compte c1 =
{12345,'i',"Dupond",2000.45,01,11,2009};
```

Manipulation des structures (2)

- Accés aux membres : opérateur .

- Syntaxe : variable.membre
  - `c1.solde = 3834.56;`
  - `struct compte c[100];`
    `y=c[33].solde;`
  - `c1.dernier_versement.jour = 15;`
  - `c[12].dernier_versement.mois = 11;`

- Sur les structures elles-mêmes:
  - Affectation : `c[4] = c1`
  - Pas de comparaison => comparer chaque membre

Exercice d'application

Write a C program to add **two fraction**s and display the result fraction.

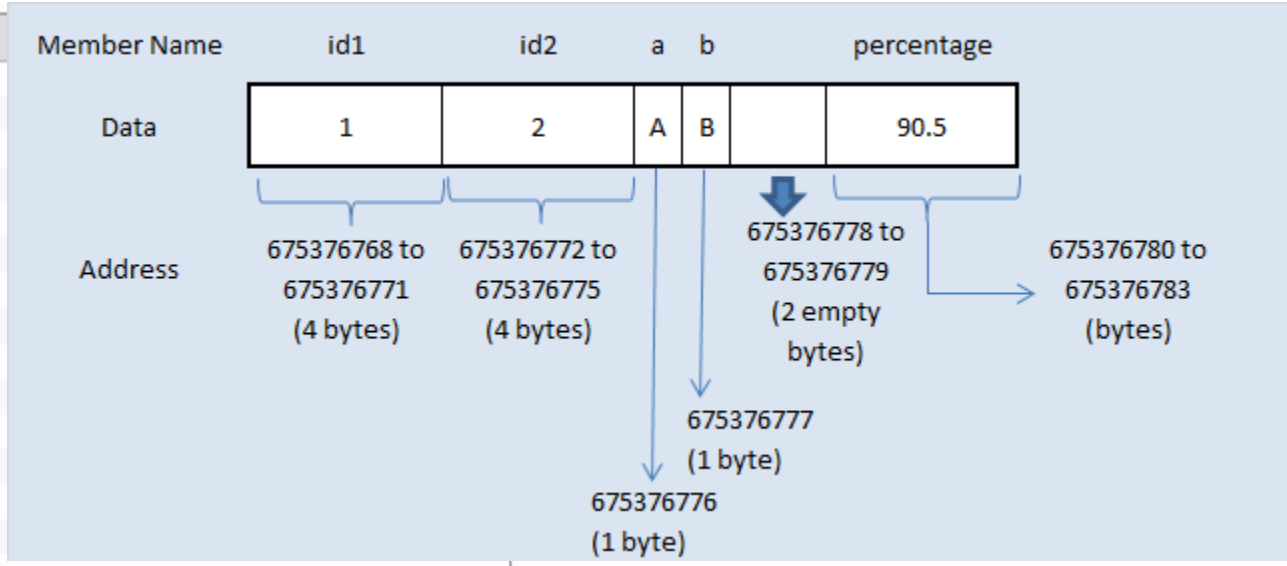
Your program will prompt the user to input fraction 1 and fraction 2. The numerator and denominator of each fraction are input separately by space.

See the example output below:

Enter fraction 1(numerator denominator): 1 2
Enter fraction 2(numerator denominator): 2 5
Result: 9/10

Structures & Allocation de mémoire

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

struct student
{
        int id1;
        int id2;
        char a;
        char b;
        float percentage;
};

int main()
{
    int i;
    struct student record1 = {1, 2, 'A', 'B', 90.5};

    printf("size of structure in bytes : %d\n",
                           sizeof(record1));

    printf("\nAddress of id1        = %u", &record1.id1 );
    printf("\nAddress of id2        = %u", &record1.id2 );
    printf("\nAddress of a          = %u", &record1.a );
    printf("\nAddress of b          = %u", &record1.b );
    printf("\nAddress of percentage = %u",&record1.percentage);

    return 0;
}
```



Structures et pointeurs (1)

- L'adresse de début d'une structure s'obtient à l'aide de l'opérateur &:

```
typedef struct {
                int no_compte ;
                char etat ;
                char nom[80];
                float solde;
                struct date dernier_versement;
                } cpt ;
cpt c1 , * pc;
```

- c1 est de type cpt, pc est un pointeur sur une variable de type cpt:

```
pc = &c1;
```

Structures et pointeurs (2)

Accès aux membres d'une structure:

```
*pc.no_compte = ...
```

Incorrect . est plus prioritaire que *

```
(*pc).no_compte =
```

Opérateur ->:

```
pc->no_compte = ...
```

Structures et fonctions (1)

- Les membres d'une structure peuvent être passés comme paramètres à des fonctions avec ou sans modification

- Exemple 1 (passage par valeur):

```
float ajoute_au_compte(float solde1, float somme1) {
        solde1 = solde1+somme1;
        return solde1;
}

void main () {
.......
cpt c1;
c1.solde = 0.;
ajoute_au_compte(c1.solde,1000);
printf("%f\n",c1.solde);
c1.solde=ajoute_au_compte(c1.solde,1000);
printf("%f\n",c1.solde);
```

Structures et fonctions (2)

- Exemple 2 (passage par adresse):

```
void ajoute_au_compte(float * solde1, float somme1) {
        *solde1 = *solde1+somme1;
}

void main () {
......
cpt c1;
c1.solde = 0.;
ajoute_au_compte(&(c1.solde),1000);  /* ou &c1.solde */
printf("%f\n",c1.solde);
```

Structures et fonctions (3)

- Un argument de fonction peut-être de type structure:

```
float ajoute_au_compte(cpt c, float somme1) {
        return(c.solde+somme1);
}
void main () {
cpt c1;
c1.solde = ajoute_au_compte(c1,1000.0);
printf("%f\n",c1.solde);
```

- Ou pointeur sur structure:

```
void ajoute_au_compte (cpt * c, float
somme1) {
        c->solde = c->solde + somme1;
}
void main () {
cpt c1;
ajoute_au_compte(&c1 ,1000.0);
printf("%f\n",c1.solde);
```

Structures et fonctions (4)

- La valeur de retour d'une fonction peut être une structure:

```
cpt ajoute_au_compte(cpt c, float somme1) {
        cpt c2;
        c2=c;
        c2.solde=c.solde+somme1;
        return(c2);
}

void main () {
.......
cpt c1;
c1.solde = 0.;
c1=ajoute_au_compte(c1,1000.0);  /* ou &c1.solde */
printf("%f\n",c1.solde);
```

Exercice d’application - Enoncé

Définir une structure de données Heure permettant de représenter une heure au format **hh/mm/ss**, puis écrire les fonctions suivantes :

1. Conversion d'un élément de type Heure en nombre de secondes (entier)

2. conversion d'un nombre de secondes (entier) en un élément de type Heure

3. addition de deux éléments de type Heure

Exercice d'application – Solution (1)

Définir une structure de données Heure permettant de représenter une heure au format **hh/mm/ss**, puis écrire les fonctions suivantes :

```
typedef struct
{
        int hh;
        int mm;
        int ss;
} Heure;
```

1. Conversion d'un élément de type Heure en nombre de secondes (entier)

```
int HeureEnSecondes(Heure h)
{
        return (h.hh*3600 + h.mm*60 + h.ss);
}
```

Exercice d'application – Solution (2)

2. Conversion d'un nombre de secondes (entier) en un élément de type Heure

```
Heure SecondesEnHeure(int sec)
{
        Heure h;
        h.hh = sec/3600;
        sec = sec%3600;
        h.mm = sec/60;
        sec = sec%60;
        h.ss = sec;
        return h;
}
```

3. Addition de deux éléments de type Heure

```
Heure AddHeures(Heure h1, Heure h2)
{     Heure h3;
       return
      (SecondesEnHeure(HeureEnSecondes(h1)+HeureEnSecondes(h2)));
}
```

QCM – Les Structures (Q1)

1. Soient les déclarations suivantes :

```
struct LIVRE {
int nb_pages ;
char langue[20], auteur[20], titre[20] ;
float prix; } L ;
```

Comment accéder à un champ d'un enregistrement ?

- auteur.L
- L.auteur
- L : auteur
- auteur.livre
- livre.auteur

QCM – Les Structures (Q2)

2. Quelle est la différence entre un tableau et un enregistrement ?

- Un tableau peut contenir des données de types différents, tandis qu'un enregistrement ne le peut pas.
- Un enregistrement peut contenir des données de types différents, tandis qu'un tableau ne le peut pas.
- Tous deux peuvent contenir des données de types différents, mais l'enregistrement occupe moins de place mémoire.
- Tous deux peuvent contenir des données de types différents, mais l'enregistrement permet un accès mémoire plus rapide.

QCM – Les Structures (Q3)

3. On considère la déclaration suivante :

```
#define MAX maxint
struct TIMBRE{
int prix ;
char origine[20] ;
int annee ;
char image[20] ;
char couleur[20];
} ;
struct TIMBRE COLLECTION [MAX];
// Une collection est un tableau de timbres
```

Comment accède-t-on à l'année du 3ème timbre de la collection?

- COLLECTION [2, 2]
- COLLECTION [2]. annee
- COLLECTION [2, annee]
- COLLECTION. annee [2]
- COLLECTION. annee

QCM – Les Structures (Q4)

4. Quelle est la déclaration correcte d'un enregistrement ?

- struct LIVRE (int nb_pages ;
  char langue[20], auteur[20], titre[20] ;
  float prix;)
- struct LIVRE { int nb_pages ;
  char langue[20], auteur[20], titre[20] ;
  float prix; }
- struct LIVRE { int nb_pages ;
  char langue[20], auteur[20], titre[20] ;
  float prix; } ;