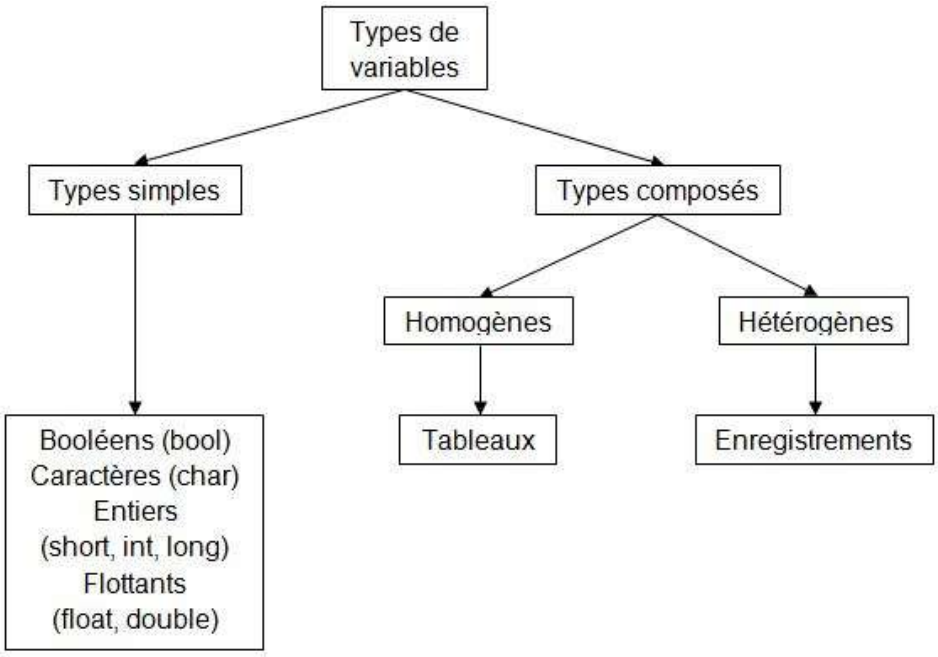
LANGAGE C – FICHE 5
Structures de données

Classification des types de variables

Les variables peuvent être de différents types, classés comme suit.



Structures de données personnalisées

Dans le langage C, il est possible de définir des structures personnalisées, adaptées à des besoins spécifiques. Elles décrivent des données composites, de types différents (qui peuvent à leur tour être constituées de structures personnalisées)

Ces données constituent les champs de la structure ; chaque champ étant défini par un type et un nom.

Une structure peut donc être interprétée comme un ensemble de variables de types différents.

Définition d'une structure

La déclaration d'une structure personnalisée doit précéder la déclaration des variables (ou fonctions) utilisant ce type. Elle se fait donc après les directives de préprocesseur (# include…), en dehors du programme principal (main()).

Elle doit se faire suivant le format :

```
struct maStructure
{
      type champ1 ;
      type champ2 ;
      …
} ;
```

Utilisation d'une variable structure

La variable doit tout d'abord être déclarée, comme n'importe quelle autre variable, en début de programme principal (main()).

```
struct maStructure nomStructure ;
```

Les valeurs peuvent ensuite être attribuées aux différents champs en utilisant la syntaxe suivante :

```
nomStructure.champ1 = valeur ;
```

Exemple

```c
/*-------------------------------------------------*/
/* IMPORTATION BIBLIOTHEQUES                       */
/*-------------------------------------------------*/

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

/*-------------------------------------------------*/
/* DEFINITION STRUCTURE                            */
/*-------------------------------------------------*/

struct prof
{
	char nom[20];
	char matiere[20];
};

/*-------------------------------------------------*/
/* PROGRAMME PRINCIPAL                            */
/*-------------------------------------------------*/

main()
{
	// Déclaration de variables
	struct prof prof1 = { "Balan", "Math" };
	struct prof prof2;

	// Utilisation des structures
	printf(" La discipline enseignee par M. %s est : %s\n", prof1.nom, prof1.matiere);

	printf("Entrer le nom du second prof : \n");
	scanf("%s", &prof2.nom);
	printf("Entrer la discipline du second prof : \n");
	scanf("%s", &prof2.matiere);
	printf("La discipline enseignee par M. %s est : %s\n", prof2.nom, prof2.matiere);
	printf("\n\n");
	return 0;

}
```

Remarque : champ de type structure

Un des champs de la structure peut lui-même être une structure personnalisée.

La définition des structures doit se faire dans un ordre cohérent : en premier lieu, les structures utilisées pour les champs, puis les structures englobantes.

L'accès à la valeur d'un champ se fait alors sous la forme :

```
nom_structure.champ1.champ2 = valeur ;
```

Exemple :

```
/*-------------------------------------------------*/
/* IMPORTATION BIBLIOTHEQUES                       */
/*-------------------------------------------------*/

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

/*-------------------------------------------------*/
/* DEFINITION STRUCTURE                            */
/*-------------------------------------------------*/

struct prof
{
	char nom[20];
	char matiere[20];
};

struct equipe
{
	char nom[100];
	struct prof prof1;
	struct prof prof2;
	struct prof prof3;
	struct prof prof4;

};

/*-------------------------------------------------*/
/* PROGRAMME PRINCIPAL                            */
/*-------------------------------------------------*/

main()
{
	// Déclaration de variables
	struct prof pr1 = { "Balan", "Math" };
	struct prof pr2 = { "Renault", "SII" };
	struct prof pr3 = { "Gregoire", "Physique" };
	struct prof pr4 = { .matiere ="Anglais", .nom = "Mauvieux" };
	struct equipe mp2i;

	printf("Entrer la classe : \n");
	scanf("%s", &mp2i.nom);
	mp2i.prof1 = pr1;
	mp2i.prof2 = pr2;
	mp2i.prof3 = pr3;
	mp2i.prof4 = pr4;

	// Affichage
	printf("Dans la classe %s, la matiere %s est enseignee par M(me) %s\n", mp2i.nom,
mp2i.prof1.matiere, mp2i.prof1.nom);

	printf("Dans la classe %s, la matiere %s est enseignee par M(me) %s\n", mp2i.nom,
mp2i.prof2.matiere, mp2i.prof2.nom);

	printf("Dans la classe %s, la matiere %s est enseignee par M(me) %s\n", mp2i.nom,
mp2i.prof3.matiere, mp2i.prof3.nom);
```

```
	printf("Dans la classe %s, la matiere %s est enseignee par M(me) %s\n", mp2i.nom,
mp2i.prof4.matiere, mp2i.prof4.nom);

	printf("\n\n");
	return 0;

}
```

La sortie obtenue est alors la suivante :

```
Entrer la classe :
theBest
Dans la classe theBest, la matiere Math est enseignee par M(me) Balan
Dans la classe theBest, la matiere SII est enseignee par M(me) Renault
Dans la classe theBest, la matiere Physique est enseignee par M(me) Gregoire
Dans la classe theBest, la matiere Anglais est enseignee par M(me) Mauvieux
```

Remarques

- Il est possible de bâtir un tableau de structure. Néanmoins, il est impossible d'initialiser les structures lors de la déclaration du tableau.
- Il est possible de passer une structure en argument à une fonction ou de retourner une structure à la fin d'une fonction.

Enumération

Un type énuméré est un type simple correspondant à une liste de symboles (mots clé), chacun étant associé à une valeur de type int.

Déclaration

Elle se fait entre les directives préprocesseurs et le programme principal, suivant le format :

```
enum nomEnumeration
{
	symbole1 ,
	symbole2 ,
	…
} ;
```

Si aucune valeur n'est précisée, la valeur associée au premier symbole vaut 0 et les suivantes sont incrémentées par rapport à la valeur précédente. Si un des symboles est associé à une valeur, le symbole suivant, si aucune valeur ne lui est attribuée explicitement, sera affecté de la valeur du symbole précédent incrémentée de 1.

Les symboles doivent être uniques.

Exemple

`enum nomEnumeration`
`{`
` haut ,`
` bas ,`
` gauche = -2 ,`
` droite`
`} ;`
	EN l'absence d'affectation, la valeur associée à haut est 0.
	La valeur de bas n'étant pas spécifiée, elle vaut 1.
	La valeur de gauche est spécifiée est vaut -2.
	La valeur de droite, non précisée, vaudra -1.

Intérêt

Le compilateur associe les valeurs entières aux symboles (et non l'inverse). Il est donc possible d'utiliser un symbole d'un type énuméré comme une valeur entière. Le contraire (utilisation d'une valeur entière en temps que symbole) n'est pas possible.

Les énumérations permettent de rendre plus lisible les codes.

Types personnalisés

Il est aussi possible de définir un identificateur pour un nouveau type. Ce procédé est principalement utilisé pour simplifier les écritures des déclarations.

La définition du nouveau type se fait entre les directives préprocesseur (# include…) et le programme principal (main()).

Définition d'un nouveau type

Dans le cas d'un type associé à une structure, les syntaxes possibles sont les suivantes :

```
typedef struct nomStructure
      {
            type champ1 ;
            type champ2 ;
            …
      } nomType ;
```

```
typedef struct
      {
            type champ1 ;
            type champ2 ;
            …
      } nomType ;
```

Si la structure a été définie auparavant, il est possible de déclarer le type de la manière suivante :

```
typedef nomStructure nomType ;
```

Il est possible (mais très fortement de déconseillé) de renommer ainsi des types existant :

```
typedef int entiers ;
```

Par contre, il peut être intéressant de définir un type associé à un tableau de dimension particulière :

```
typedef char ligne [80] ; // ici, ligne est un tableau de 80 char
```

Utilisation d'un type personnalisé

Son utilisation est identique aux types prédéfinis en C.