Les types de Passage et les Structures

Le passage par valeur :

Le passage par valeur consiste à fournir à la fonction appelée une copie d'une variable de la fonction appelante. Il s'agit de deux variables différentes, l'une étant simplement initialisée avec la valeur de l'autre.
L'exemple suivant est celui d'une fonction recevant ses arguments par un passage par valeur.

```
#include <iostream >
using namespace std ;
void PassageParValeur(int i) // Déclaration et définition
{
 i =10;
 }
 int main ()
 {
 int i=0;
PassageParValeur(i);
 cout <<" Valeur de i :" <<i;
 return 0;
 }
```

```
Résultat:
Valeur de i : 0
```

La variable i n'est pas modifiée par l'appel de la fonction **PassageParValeur**, car les variables i dans **PassageParValeur** et main sont différentes même si elles portent le même nom.

Le passage par pointeur :

Appelé aussi passage par adresse, le passage par pointeur permet de transmettre à une fonction l'adresse d'une variable. Celle-ci est stockée dans un argument de type pointeur qui peut être par la suite utilisée pour modifier la variable pointée. Les changements effectués à partir de cette adresse affectent aussi la variable de la fonction appelante. L'exemple qui suit illustre le passage d'arguments par pointeur.

```
#include <iostream >

using namespace std ;

void PassageParPointeur (int *var ) // Déclaration et définition
 {
 * var =10; // modification du contenu pointé
 }
 int main ()
 {
 int i=0;
 PassageParPointeur (&i);
```

```
 cout <<" Valeur de i :" <<i;
 return 0;
}
```

```
Résultat :
Valeur de i : 10
```

La variable i est modifiée suite à l'appel de la fonction **PassageParAdresse**

Le passage par référence :

Les références permettent de désigner des variables déclarées dans une autre fonction. Dans d'autres langages, les références sont nommées alias, c'est à dire un nom de variable qui désigne une autre variable. Les références fournissent un concept proche de celui des pointeurs et peuvent s'utiliser dans le même contexte.

La déclaration d'une référence se fait en plaçant l'opérateur **&** entre le type de la référence et son nom. Les références doivent être initialisées au moment de leur déclaration. Le passage par référence est illustré par cet exemple

```
# include <iostream>
using namespace std ;

int main ()
 {
 int a;
 int& ref =a; // declaration d'une reference vers a
 ref =10; // equivalent `a a=10
 cout <<"a = " <<a<<endl ;
cout <<" ref =" <<ref ;
 return 0;
 }
```

Les structures :

Une structure est un ensemble d'éléments de types différents réunis sous un même nom.

```
struct NomStructure { type1 champ1 ; type2 champ2 } ;
```

La déclaration d'une structure correspond à la déclaration d'un nouveau type de données ; on définit ainsi un modèle. Les constituants d'une structure sont appelés champs (ou membres). Après la définition du modèle de la structure, il est possible de déclarer des variables de ce nouveau type par la même syntaxe utilisée pour les types prédéfinis. Dans l'exemple suivant, on déclare une structure article et deux variables de cette structure.

```
struct article
{
int numero ;
int qte ;
float prix ;
} ; // D´efinition de la structure article
```

```
article art1 , art2 ; // d´eclaration de 2 variables structre
```

Manipulation d'une structure :

L'accès à un champ d'une variable structure se fait en suivant d'un point ('.') le nom de la variable structure, suivi du nom du champ. Ce champ peut alors être manipulé comme n'importe quelle variable du type correspondant.

```
art1. numero = 15 ;
cin >> art2.prix ;
art1. numero ++ ;
```

Initialisation d'une structure :

Comme pour un tableau, une structure peut être initialisée au moment de sa création en énumérant ses champs :

```
article art1 = { 100, 285, 200 } ;
```