Université Ibn Tofail
جامعة ابن طفيل
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵉⴱⵏ ⵟⴼⵢⵍ
Ibn Tofail University

Spécifications du Langage de Programmation C++

Med. AMNAI

Filière SMI - S5

Département d'Informatique

17 novembre 2020

Plan

Plan

Plan

Plan

Transmission des arguments par valeur

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


main()
{
    void echange(int,int);
    int a=2, b=5;
    cout<<" Avant appel : " << a << " - " << b << endl;
    echange(a,b);
    cout<<" Après appel : " << a << " - " << b << endl;
}

void echange(int m,int n)
{
    int z;
    z=m;
    m=n;
    n=z;
}
```

```
Avant appel : 2 - 5
Apres appel : 2 - 5
```

Transmission des arguments par adresse

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

main(){
     void echange(int *,int *);
     int a=2, b=5;

     cout<< "Avant appel : " << a << " - " << b <<endl;
     echange(&a,&b);
     cout<< "Apres appel : " << a << " - " << b <<endl;
}

void echange(int *x,int *y)
{
     int z;
     z=*x;
     *x=*y;
     *y=z;

}
```

```
Avant appel : 2 - 5
Apres appel : 5 - 2
```

Transmission des arguments par référence

En C++, la transmission par référence est une forme simplifiée de la transmission par adresse.

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

main(){
     void echange(int &,int &);
     int a=2, b=5;

     cout<< "Avant appel : " << a << " - " << b << endl;
     echange(a,b);
     cout<< "Apres appel : " << a << " - " << b << endl;
}

void echange (int & x,int & y)
{
     int z;
     z=x;
     x=y;
     y=z;
}
```

```
Avant appel : 2 - 5
Apres appel : 5 - 2
```

Princip des arguments par défaut (pardefaut.cpp)

En C, il est indispensable que le nombre d'arguments passés correspond au nombre d'arguments déclarés. **C++ peut ne pas respecter cette règle**.

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

main(){
    int m=1, n=5;
    void f(int,int=7);

    f(3,5);
    f(4);
    // f(); //Un appel de ce type est rejeté par le compilateur.
}

void f(int a,int b)
{
    cout << "Valeur 1 : " << a << " - Valeur 2 : " << b << endl;
}
```

```
Valeur 1 : 3 - Valeur 2 : 5
Valeur 1 : 4 - Valeur 2 : 7
```

Exemple (pardefaut2.cpp)

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

main(){
    void f(int=33,int=77);
    f(99,55);
    f(44);
    f();
}

void f(int a,int b)
{
    cout << "Valeur 1 : " << a << " - Valeur 2 : " << b << endl;
}
```

```
Valeur 1 : 99 - Valeur 2 : 55
Valeur 1 : 44 - Valeur 2 : 77
Valeur 1 : 33 - Valeur 2 : 77
```

Arguments par défaut (suite)

- Lorsqu'une déclaration prévoit des valeurs par défaut, les arguments concernés doivent obligatoirement être les derniers de la liste.

- Exemple : La déclaration : **int f (int=2, int, int=7) ;** est interdite, car un appel du genre **f(3, 4) ;** peut être interprété comme : **f(2, 3, 4) ; ou f(3, 4, 7) ;**

Overloading ou surcharge

- On parle de surdéfinition ou de surcharge lorsqu'un même symbole possède **plusieurs significations** différentes ;
- Le choix de l'une des significations est en fonction du contexte.
- En C++, ce choix est **basé sur le type des arguments**.

Exemple 1 (surdefinit.cpp)

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

void f(int x){
     cout<< " fonction numero 1 : "<< x << endl;
}

void f(double x){
     cout<< " fonction numero 2 : "<< x << endl;
}

main()
{
     int a=2; double b=5.7;
     f(a);
     f(b);
     f('A');
}
```

```
fonction numero 1 : 2
fonction numero 2 : 5.7
fonction numero 1 : 65
```

Exemple 2 (cas2.cpp)

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

void f(char *){
    cout<< " fonction numero 1 "<< endl;
}

void f(void *){
    cout<< " fonction numero 2 "<< endl;
}

main(){
    char *p1;
    double *p2;

    f(p1); // Appellera la fct f1.
    f(p2); // appellera la fct f2 après conversion de p2 en void *
}
```

```
fonction numero 1
fonction numero 2
```

Exemple 3 (cas3.cpp)

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

void f(int, double){
    cout<< " fonction numero 1 "<< endl;
}

void f(double, int){
    cout<< " fonction numero 2 "<< endl;
}

main(){
    int a, b;
    double x;
    char c ;

    f(a, x); // Appellera la fct f1.
    f(c, x); // Appellera la fct f1 après conversion de c en int.
    //f(a, b); // Erreur de compilation (convertir a en double OU b en double).
}
```

```
fonction numero 1
fonction numero 1
```

Exemple 4 (cas4.cpp)

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

void f(int a=0 , double c=0){
    cout<< " fonction numero 1 "<< endl;
}

void f(double y=0 , int b=0){
    cout<< " fonction numero 2 "<< endl;
}

main(){
    int m;
    double z ;
    f(m, z); // Appellera la fonction f1.
    f(z, m); // Appellera la fonction f2.
    f(m);    // Appellera la fonction f1.
    f(z);    // Appellera la fonction f2.
    //f();   // Conduira à une erreur de compilation.
}
```

```
fonction numero 1
fonction numero 2
fonction numero 1
fonction numero 2
```

Opérateur new

new fournit comme résultat :

- Un pointeur **sur l'emplacement** correspondant, lorsque l'allocation réussie.
- Un pointeur **NULL** dans le cas contraire.

```
main(){
    int *p;
    p=new int;
    //ou
    int *p=new int; // Allocation dynamique d'un entier.
    int *p2=new int[5]; // Allocation dynamique de 5 entiers.
    char *t;
    t=new char[30];
    //ou
    char *t=new char [30];
}
```

Opérateur delete

delete possède deux syntaxes :

- delete p ;
- delete [ ]p ;

RQ : **p** est une variable devant avoir comme valeur un pointeur sur un emplacement alloué par **new**.

Utilisation des fonctions en ligne (inline)

Une fonction **en ligne** se définit et s'utilise comme une *fonction ordinaire*, avec la seule différence qu'on fait précéder son en-tête de la spécification **inline** (inline.cpp).

```
#include<iostream>
#include<math.h>
using namespace std;

inline double norme(double vec[3])
{    double s=0;
     for(int i=0; i<3; i++)
         s=s+vec[i]*vec[i];
     return sqrt(s);
}

main(){
     double V1[3], V2[3];
     for (int i=0; i<3; i++) {
         V1[i]=i; V2[i]=i*3;
     }
     cout<< " Norme de V1 est : " << norme(V1) << endl;
     cout<< " Norme de V2 est : " << norme(V2) << endl;
}
```

```
Norme de V1 est : 2.23607
Norme de V2 est : 6.7082
```

Utilisation des fonctions en ligne (suite)

- La fonction **norme** a pour but de calculer la norme d'un vecteur passé comme argument.
- La présence du mot **inline** demande au compilateur de traiter la fonction *norme* d'une manière différente d'une fonction ordinaire, à chaque appel de *norme*, il devra incorporer au sein du programme, les instructions correspondantes (en langage machine). *Le mécanisme habituel de gestion de l'appel est de retour n'existe plus*, **ce qui réalise une économie de temps**.
- **RQ** Une fonction **en ligne doit être définie dans le même fichier source que celui où on l'utilise**. Elle ne peut être compilée séparément.