C++

Programmation Objet en C++
Programmation générique en C++
Design Patterns et C++
Policies

Meilleur C

Le langage C++ se veut un langage C amélioré.

Il possède des fonctionnalités supplémentaires, et notamment

* la surcharge de fonctions
* le passage par référence
* l'emplacement des déclarations
* l'allocation dynamique

Les apports spécifiques de C++ sont

* l'aide à l'abstraction de données: définition de types de données, et de leur implémentation concrète.
* l'aide à la programmation objet: hiérarchie de classes et héritage.
* l'aide à la programmation générique: classes patron et algorithmes génériques.

Meilleur LOO

La programmation Générique:

- Implémentation efficace et typé des containers
- Possibilités de Méta-programmation
- Mise en oeuvre puissante des Design Patterns
- Le beurre ou l'argent du beurre de la syntaxe objet.

Incompatibilités entre C et C++

Toute fonction doit

- être définie avant utilisation
- ou être déclarée par un prototype

```
float fct (int, double, char*);
```

(En C, une fonction non déclarée est supposée de type de retour int.

Une fonction qui ne retourne pas de valeur a le type de retour void.

Le qualificatif const peut être utilisé pour une expression constante nécessaire pour la taille d'un tableau

```
const int N = 10; // remplace #define N 10
int valeurs[N];
```

Définition de Classes

Le `typedef` est implicite dans les structures la définition :

```
struct Point {
  int x, y;
};
```

Défini un type Point et un type struct Point.

La définition

```
classe Point { public: int x; int y; } ;
```

a le même effet.

Entrées-sorties

Les entrées et sorties sont gérées dans C++ à travers des objets particuliers appelées streams ou flots. Inclure <iostream.h>

Deux opérateurs sont surchargés de manière appropriée pour les flots:

* l'opérateur d' insertion << (écriture)
* l'opérateur d' extraction >> (lecture)

Trois flots prédéfinis sont

* cout attaché à la sortie standard;
* cerr attaché à la sortie erreur standard;
* cin attaché à l'entrée standard;

on retrouve les flots du C stdin, stdout, stderr.

E/S exemple 1

```
#include <iostream.h>
main() {
  cout << "Bonjour, monde !\n";
}
```

Plusieurs expressions:

```
cout << ex_1 << ex_2 << ... << ex_n ;
```

Plusieurs ``lvalues'':

```
cin >> lv_1 >> lv_2 >> ... >> lv_n ;
```

Les types écrits ou lus sont

```
char, short, int, long, float, double, char*
```

Exemple

```
#include <iostream.h>
int i;
main() {
  cout << "Un entier : ";
  cin >> i;
  cout << "Le carre de " << i  <<" est " << i*i << endl;
}
```

Commentaires

/* commentaire

// commentaire

commentaire

*/

// commentaire

Comme en JAVA

Emplacement des déclarations

Une déclaration peut apparaître en n'importe quelle position d'un corp de classe ou de fonction, mais doit précéder son utilisation.

```
int n;
n = 3;
int q = 2*n-1;
for (int i = 0; i<n ; i++) {...} // déconseillé
```

Arguments par référence

Enfin le passage par référence qui manque au C.

Un paramètre dont le nom est suivi de & est transmis par référence, donc

- pas de copie de l'argument à l'appel;
- possibilité de modifier l'argument.

Déclaration

```
void echange(float&, float&) ;
```

Définition

```
void echange(float& a, float& b) {
  float t = a;   a = b;   b = t;
}
```

Exemple

```
float x = 2, y = 3;
echange (x, y);
cout << x <<","<< y; //Affiche 3,2
```

Références Constantes

Passage par référence constante pour

ne pas copier l'argument à l'appel;

ne pas modifier l'argument :

```
void afficher(const objet&);
```

Passage par référence constante permet d'assurer l'encapsulation en évitant la copie sur la pile de structures trop grande.

Arguments par défaut

Les derniers arguments d'une fonction/méthode peuvent prendre des ``valeurs par défaut".

Déclaration

```
float f(char, int = 10, char* = "Tout");
```

Appels

```
f(c, n, "rien")
f(c, n)    // <-> f(c, n, "Tout")
f(c)       // <-> f(c,10, "Tout")
f()        // erreur
```

Seuls les derniers arguments peuvent avoir des valeurs par défaut.

```
float f(char = 'a', int, char* = "Tout"); // erreur
```

Surcharge

Un même identificateur peut désigner plusieurs fonctions, si elles diffèrent par la liste des types de leurs arguments.

```
float max(float a, float b) { return (a > b) ? a : b;}
float max(float a, float b, float c) {  return max(a, max(b, c));}
float max(int n, float t[]) {
  if (!n) return 0;
  float m = t[1];  for (int i = 2 ; i < n; i++)
    m = max(m, t[i]);
  return m;
}
void main() {
  float x, y, z;
  float T[] ={11.1, 22.2, 33.3, 44.4, 7.7, 8.8 };
  x = max (1.86, 3.14);
  y = max (1.86, 3.14, 37.2);
  z = max (6, T);
 cout << x <<" "<< y <<" "<< z;
}
```

Allocation dynamique

Deux Opérateurs Intégrés au langage.

Les opérateurs `new` et `delete` gèrent la mémoire dynamiquement.

```
new chose[n]
```

alloue la place pour éléments de type chose et retourne l'adresse du premier élément;

```
delete addresse
```

libère la place allouée par new.

```
int* a = new int; // malloc(sizeof(int))
double* d = new double[100];
                  // malloc(100*sizeof(double))
```

Fonctions/Méthodes « en ligne »

Une fonction en ligne (`inline`) est une fonction dont les instructions sont incorporées par le compilateur dans le module objet à chaque appel. Donc

* il n'y pas d'appel : gestion de contexte, gestion de pile;
* les instructions sont engendrées plusieurs fois;
* rappellent les macros.

Déclaration par qualificatif inline.

```
inline int sqr(int x) { return x*x; }
```

[les méthodes définie dans le corps de classe sont inlinées]

[les fonctions trop grandes ou récursives ne sont pas inlinées]

Classes et objets

Une classe est une structure, dont les attributs sont des données ou des méthodes. Un objet, ou une instance, est un exemplaire de cette structure.

Classe:

```
class Complexe {
  public:
    float re, im;//données
    void show(); //méthode
};
```

On déclare des objets de la classe par:

```
Complexe a, b;
```

On les manipule de manière usuelle:

```
a.re = b.im;     b.re = 7;     a.show();
```

La définition d'une méthode se fait soit en ligne, soit séparément; dans le deuxième cas, elle utilise l' opérateur de portée noté :: pour désigner la classe.

```
void Complexe::show() {
  cout << re << ' ' << im;
}
```

A l'appel a.show(), la méthode est celle de la classe de a, donc Complexe::show (), et les champs re et im sont ceux de l'objet appelant, c'est-à-dire a.re et a.im.

La surcharge d'opérateurs permet de définir une forme agréable pour des opérations sur des objets.

```
Complexe operator+(Complexe s, Complexe t) {
  Complexe w;
  w.re = s.re + t.re;
  w.im = s.im + t.im;
  return w;
}
```

On peut alors écrire :

```
Complexe a, b, c;
  ...
a = b + c;
```