Du C au C++ (mais vite)

François Faure, UJF-Grenoble

C++ est à l'origine une extension du C. Tout programme C compile en C++. C++ rajoute de la souplesse et des fonctionnalités. Sous Linux, le compilateur standard est g++. Sous Windows, il existe un compilateur Borland gratuit.

1 Commentaires

```
int a; // un commentaire
```

2 Déclarations à la volée

```
a = 2*b;
int c = b+2; // déclaré là où il sert, initialisé en même temps
for( int i=0; i<3; ++i ){ // dans la dernière norme, la portée de i
    ...                   // est limitée à la boucle
}
```

3 Constantes

```
const double PI = 3.14; // permet d'éviter #define
```

4 Méthodes inline

Directement remplacées dans le code, avec les vérifications syntaxiques du compilateur. Déclaration et définition doivent figurer dans le même fichier.

```
inline int max_int( int a, int b ) // permet d'éviter #define
{ return a>b ? a : b; }
```

5 Généricité

```
template <class T> inline                           // fonctionne pour tout type
T max_generique( T a, T b ){ return a>b ? a : b; }  // muni de l'operateur >
```

Déclaration et définition de méthode générique doivent figurer dans le même fichier.

6 Passage par réference

```
void permute( int& a, int& b ){  // notez les &
    int tmp=a; a=b; b=tmp;    // simplifie l'écriture
}

double volume( const Cube& c ); // méthode ne modifiant pas le paramètre
```

7 Entrées-sorties

```
int a; double b;
cin >> a >> b;                              // remplace scanf
cout<<"valeurs: "<< a <<", "<< b << endl; // remplace printf
```

8 Allocations dynamiques

```
double *b = new double;    // remplace malloc
delete b;                  // remplace free
double *v = new double[3];
delete[] v;
```

9 Classes (programmation orientée objet)

```
class A {
  public:
    A(int a, double b);       // un constructeur
    ~A();                     // le destructeur
    void set_a(int a);        // une méthode membre
    int get_a() const;        // méthode ne modifiant pas l'objet
    virtual void affiche();   // typage dynamique
    static char* className(); // méthode indépendante de l'instance
  protected:
    // données et méthodes réservées aux classes dérivées
  private:
    // données et méthodes cachées
    int le_a;
    double le_b;

    // méthodes externes ayant accès à la partie privée
    friend ostream& operator << (ostream&, const A&);
    friend istream& operator >> (istream&, A&);
};
```

```
A::A(int a, double b)                 // constructeur
  : le_a(a), le_b(b)                  // assigner des valeurs lors de la construction
{ cout<<"creation d'un A"<<endl; }  // autres opérations

void A::set_a(int a){
  le_a = a;                           // assigner des valeurs après construction
}

ostream& operator << (ostream& out, const A& a){
    out << a.le_a <<", "<< a.le_b;
    return out;
}

istream& operator >> (istream& in, A& a){
    in >> a.le_a >> a.le_b;
    return in;
}

A a(3,2.2); a.affiche();
A* b = new A(1,5.3); b->affiche();
cin >> a >> b;
cout << a <<" "<< b <<endl;
delete b;                             // toujours pas de ramasse-miettes
```

10 Héritage

```
class B: public A  { ... };          // un héritage simple
class D: public B public C { ... };  // un héritage multiple
```

11 Namespaces

Évitent les collisions de noms.

```
namespace mon_app {
   typedef int* vector;                    // vector=tableau d'entiers
   vector un_vec();
}

namespace son_app {
   typedef double vector[3];               // vector=tableau de 3 double
   vector un_vec();
}

mon_app::vector v = mon_app::un_vec();   // pas de confusion
```

```
son_app::vector vec = son_app::un_vec();
```

12 Standard Template Library (STL)

Implémente une bonne fois pour toutes des structures (tableaux, listes chaînées, listes triées, piles, chaînes de caractères, tableaux associatifs, tables de hachage,...) et algorithmes (copier, insérer, trier, trouver, compter, fusionner...) qui reviennent tout le temps.

```
std::vector<double> mon_vec(4); // tableau redimensionnable de 4 double
std::list<int> ma_liste;        // liste doublement chaînée d'entiers (vide)

template<class Container>
void met_a_zero( Container& c )          // méthode générique
{
  Container::iterator i;                 // sert à désigner un élément
  for( i=c.begin(); i!=c.end(); ++i ){  // contrôle d'itération
    *i = 0;                              // déréferencer avec *
  }                                      // applicable à tout conteneur STL
}

met_a_zero( mon_vec );
met_a_zero( ma_liste );
```

13 Références

- La page de Bjarne Stroustrup[1], créateur du C++.
- La bible : Bjarne Stroustrup, *Le langage C++*. Version française chez CampusPress, 1999; ISBN 2-7440-0609-2.
- Une introduction à la STL[2]
- La référence complète de la STL[3]
- Thinking in C++
  - Première partie[4]
  - Deuxième partie[5]
  - Compléments[6]
- un site entierement dédié au C++[7]

[1] http ://www.research.att.com/ bs/C++.html
[2] http ://www-imagis.imag.fr/Membres/Xavier.Decoret/STL_TUTORIAL/index.html
[3] http ://www.sgi.com/tech/stl/
[4] http ://ban.mansfield.ohio-state.edu/boat/books/ThinkingInCPPv1/Frames.html
[5] http ://ban.mansfield.ohio-state.edu/boat/books/ThinkingInCPPv2/Frames.html
[6] http ://www.mindview.net/Books/TICPP/Solutions/
[7] http ://www.cplusplus.com/