POLYCOPIE DE C++

1. Introduction

Ce document est actuellement la compilation de notes de cours C++ donné en MIAIF, DESS, MST2.

Plan du cours

2. présentation générale
3. minimum sur les entrées sorties
4. rappel sur les références et les pointeurs
5. spécificités de C++ par rapport à C
6. classes et objets
7. propriétés des fonctions membres
8. construction destruction initialisation d'objets
9. généralisation, héritage
10. fonctions virtuelles
11. templates
12. La STL, par l'exemple de la classe `vector`
13. surdéfinition d'opérateurs =, *, <<, >>, ()
14. fonctions amies
15. entrées sorties «élaborées»

Références

Delannoy, « Le langage C++ », Eyrolles
Très bon livre pour commencer le C++, livre très pédagogique. A servi de base pour ces notes de cours.

Stroustrup, « Le langage C++ », Addison Wesley, Pearson Education
Le livre de référence par l'auteur du C++. Complet mais beaucoup trop riche pour débuter.

Scott Meyers, « Le C++ efficace », Addison Wesley
Des règles pour programmer proprement et efficacement en C++.

Schaum, C++ cours et exercices corrigés
C++ détaillé avec exemples, exercices corrigés. A servi de base pour une partie des notes de ce cours. Par contre, les classes, l'héritage et les fonctions virtuelles ne sont pas assez développés.

Johannes Weidl, « The Standard Template Library Tutorial ».
Un tutorial sur la STL, accessible sur le Web. Pour programmeur confirmé.

2. Présentation générale

Le C++ a été conçu par Bjarne Stroustrup en 1982 aux ATT Bell Laboratories. L'idée était d'ajouter au C des possibilités sur l'orienté objet et de palier aux inconvénients du C.

Le but de ces notes de cours est de donner un premier aperçu sur C++ en ne présentant que ce qu'il est nécessaire de connaître pour bien programmer objet avec C++. Ce document suppose connu le langage C et l'orienté objet sans que cela soit bloquant pour le lire.

3. Le minimum vital sur les entrées sorties

Bien sûr, C++ dispose des routines offertes par la bibliothèque standard du C ANSI <stdio.h>. Mais il comporte aussi des possibilités d'entrées sorties propres. Le but de ce paragraphe n'est pas de les présenter complètement (cf paragraphe plus loin) mais *de présenter le minimum sur les entrées sorties utilisées pour commencer* à programmer en C++.

Ecrire sur la sortie standard

Le premier programme C++ à écrire est le suivant.

```
#include <iostream.h>
main () { cout << " bonjour le monde ! " << endl ; }
```

<iostream.h> est la bibliothèque du C++ permettant de faire des entrées sorties.

Pour écrire, sur la sortie standard on utilise le " stream " de la sortie standard **cout**.

L'opérateur **<<** prend un stream en premier opérande et une chaîne de caractères en deuxième opérande et retourne le premier opérande après avoir écrit la chaîne de caractères sur le stream.

On peut chaîner l'utilisation de << pour écrire plusieurs chaînes de caractères à la suite les unes des autres.

endl signifie le saut de ligne.

Ainsi, la sortie de ce premier programme C++ est :

```
bonjour le monde !
```

On peut afficher la valeur d'une variable.

```
#include <iostream.h>
main () {
    int n = 25 ;
    cout << " la valeur de n est   " << n << endl ;
}
```

Plus généralement on peut écrire :

```
cout << … << … << … ;
```

Le type des variables que l'on peut envoyer sur << est presque quelconque.

Lire sur l'entrée standard

On peut également lire des informations sur l 'entrée standard du programme.

Le programme suivant demande d'entrer une valeur sur l'entrée standard, puis il la lit avec l'opérateur **>>** et le stream **cin** de l'entrée standard ; enfin il affiche la valeur entrée par l'utilisateur.

```
#include <iostream.h>
main () {
    int n;
    cout << " entrer une valeur :   " ;
    cin >> n ;
    cout << " la valeur entrée est   " << n << endl ;
}
```

De même que pour <<, l'opérateur >> accepte presque tous les types et il est associatif.

On peut donc écrire un programme qui demande des valeurs de types différents de la manière suivante :

```
#include <iostream.h>
main () {
    int n; float x ; char t[64] ;
    cout << " entrer un entier, un flottant et une chaîne de caractères:
  " ;
    cin >> n >> x >> t ;
    cout << " l'entier vaut   " << n ;
    cout << " le flottant vaut   " << x ;
    cout << " la chaîne vaut   " << t << endl ;
}
```

séparateurs

Les valeurs entrées doivent être séparées par un caractère espace ou n'importe quel autre caractère spécial (tabulation, saut de ligne, fin de fichier, etc.).

4. Rappel sur les pointeurs et références

Ce paragraphe rappelle le strict minimum sur ce que sont un pointeur ou une référence. Pour des détails, se reporter par exemple à [Schaum].

Quand on déclare une variable avec un nom et un type, un emplacement mémoire du type de la variable est créé à une certaine adresse avec son nom pour y accéder. L'emplacement mémoire recevra la valeur de la variable lors d'une affectation.

```
int x ;           // une déclaration
x = 3 ;           // une affectation
```

Le C permet de manipuler dans le programme, les adresses des emplacements mémoire des variables. `&x` désigne l'adresse de la variable `x`. On peut déclarer des *pointeurs* d'un type qui sont des variables contenant des adresses de variables de ce type avec le symbole `*`.

```
int * p ;         // un pointeur sur un entier
p = &x ;          // p vaut l'adresse de x
```

L'opérateur `*` s'appelle l'opérateur de *déréférencement* et `*p` désigne la valeur contenue dans l'emplacement mémoire dont l'adresse et `p`.

```
int y = *p ;      // y vaut 3
```

On peut aussi déclarer une *référence* à une variable existante. Une référence se déclare avec l'opérateur `&`. Une référence est un synonyme - un alias (un nom supplémentaire) - pour désigner l'emplacement mémoire d'une variable.

```
int & z = x ;     // z est une référence a x, z vaut 3
```

Si on change la valeur de `x`, la valeur de `z` est aussi changée et inversement. Idem avec `*p`.

```
x = 4;       // x et z valent 4, *p aussi
z = 5;       // x et z valent 5, *p aussi
*p = 6;      // *p, x et z valent 6
```

Noter que les opérateurs `&` et `*` sont inverses l'un de l'autre. On a toujours :
`*(&x) = x` et `&(*p) = p`

Attention, on ne peut pas déréférencer un pointeur qui ne contient pas une adresse valide.

```
int * q ;
*q = 7 ;    // plantage
```

Il faut initialiser le pointeur avant :

```
int * q = &x;
*q = 7 ;    // ok
```

NB : Il est très important de très bien connaître le fonctionnement des pointeurs et des références car l'intérêt de la programmation objet en C++ repose sur leur utilisation intensive.

5. Spécificités de C++ surlangage de C

C++ dispose d'un certain nombre de spécificités par rapport a C qui ne sont pas axées sur l'orienté objet :

- le commentaire
- l'emplacement libre des déclarations
- les arguments par défaut
- la surdéfinition de fonction
- les opérateurs new et delete
- les fonctions en ligne

Commentaire

Pour écrire des commentaires dans un programme, le programmeur dispose toujours des commentaires C avec /* et */ et aussi de commentaires en fin de ligne avec //

```
cout << « bonjour » ;       // ceci est une formule de politesse
```

On peut mêler les deux techniques :

```
/* ceci est un commentaire en C // donc ce double slash ne sert a rien */
```

ou encore

```
// ceci est un commentaire en C++ /* donc ce slash etoile ne sert à rien */
```

mais pas :

```
 // ceci est un commentaire en C++ /* donc ce slash etoile ne sert à rien
et cela donne une erreur de compilation si le etoile slash n'est pas sur la
même ligne */
```

Emplacement des déclarations de variables

L'emplacement des déclarations est *libre* en C++ ; le programmeur n'est pas obligé de les mettre au début d'une fonction. Par contre, on ne peut utiliser la variable déclarée que dans les instructions du bloc où est effectuée la déclaration et postérieures à la déclaration. L'avantage de cette spécificité du C++ est de pouvoir déclarer une variable juste au moment où l'on en a besoin et cela clarifie le programme. Mais cet avantage est bien maigre.

```
void f() {
    ...
    i = 4 ;       // incorrect
    ...
    int i ;       // i est déclaré ici
    ...
    i = 4 ;       // correct
    ...
    {
      ...
      float f ;   // f est declare ici
      ...
      f = 4.0 ;   // correct
      ...
    }             // fin de la portee de f
    ...
    f = 4.0 ;     // incorrect
    ...
    i = 5 ;       // correct
    ...
}                 // fin de la portee de i
```

Passage de paramètres par référence

En C, les arguments sont passés par valeur. Ce qui signifie que les paramètres d'une fonction C sont toujours en entrée de la fonction et pas en sortie de la fonction ; autrement dit les paramètres d'une fonction ne peuvent pas être modifiés par la fonction.

Ci-dessous la fonction `echange` est censée échanger les valeurs des deux paramètres `n` et `p`.

```
void echange(int a, int b) {
      int c = a ;
      a = b ;
      b = c ;
}
main() {
      int n = 10 ; int p = 20 ;
      cout << « avant appel : «  << n << «  «  << p << endl ;
      echange(n, p) ;
      cout << « après appel : «  << n << «  «  << p << endl ;
}
```

Malheureusement, pour la raison invoquée plus haut, la sortie de ce programme est :

```
      avant appel : 10 20
      apres appel : 10 20
```

Les programmeurs C ont l'habitude de palier à cet inconvénient du C en passant l'adresse des paramètres en sortie d'une fonction au lieu de passer la valeur. Ainsi, notre exemple sera écrit :

```
void echange(int * a, int * b) {
      int c = *a ;
      *a = *b ;
      *b = c ;
}
main() {
      int n = 10 ; int p = 20 ;
      cout << « avant appel : «  << n << «  «  << p << endl ;
      echange(&n, &p) ;
      cout << « apres appel : «  << n << «  «  << p << endl ;
```

Ceci est un exemple, cliquez sur le lien de téléchargement pour obtenir le cours complet.