Centre Informatique pour les Lettres et les Sciences Humaines

Apprendre C++ avec QtCreator
Etape 5 : Mise en ordre

Le mécanisme de passage de paramètres que nous avons utilisé lors de l'étape 4 permet à la fonction appelante de donner des précisions à la fonction appelée sur ce que celle-ci doit faire.

La fonction appelée, pour sa part, peut renvoyer une valeur lorsque son exécution s'achève. Si la fonction appelante le souhaite, elle peut donc utiliser une valeur qui lui sera communiquée par la fonction appelée.

Ces deux mécanismes ne suffisent toutefois pas pour couvrir les besoins de communication entre des fonctions destinées à collaborer pour traiter un même problème.

1 - Création de références

La création d'une référence n'est pas la création d'une nouvelle variable, mais simplement l'introduction d'un nouveau nom désignant une variable qui existe déjà. Dans le cas le plus simple, la création d'une référence introduit une synonymie (deux noms désignant le même objet) :

```
1 int uneVariable(5);               //création d'une variable
2 int & uneReference(uneVariable); //création d'une référence désignant la variable
3 uneVariable += 1;
4 uneReference += 1;
  //uneVariable contient maintenant 7
```

Visuellement, la création d'une référence *(2)* ressemble beaucoup à la création d'une variable, mais :
- le signe `&`, placé entre le type et le nom indique qu'il ne s'agit pas d'une variable ;
- la référence doit être "initialisée" en indiquant quelle variable elle désignera.

Cette variable doit exister avant que la référence ne soit créée, et elle doit être du type annoncé pour la référence (`int`, dans l'exemple ci-dessus).

2 - Utilisation de références

La création de références ne sert habituellement pas à introduire dans un programme des synonymies qui ne feraient que le rendre plus difficile à lire et à comprendre. Dans la plupart des cas, les références sont utilisées en tant que paramètres de fonctions.

Retour sur le passage de paramètres

Imaginons la situation suivante :

```
1  int main(int argc, char *argv[])
2  {
3  int valeur(5);
4  fonction(valeur);
5  std::cout << "valeur = " << valeur;
6  }
7  //****************************************
8  void fonction(int truc)
9  {
10 ++truc;
11 std::cout << "truc = " << truc << "\n";
12 }
```

Lorsque `main()` est exécutée, la ligne *5* provoque l'appel de la `fonction()`, qui se trouve face à une valeur (5, dans cet exemple) qui, de son point de vue, n'est pas manipulable, puisqu'elle n'est désignée par aucun nom.

Fort heureusement, le mécanisme des paramètres fait qu'une variable `truc` est créée et est initialisée avec cette valeur. Par la suite, la `fonction()` peut donc faire ce que bon lui semble avec cette valeur : elle peut lui ajouter un *(10)*, l'afficher *(11)*, etc.

La variable `truc` est toutefois une variable locale à la `fonction()` : lorsque l'exécution de celle-ci s'achève, `truc` est détruite. Aucune des variables de `main()` n'est concernée par les agissements de la `fonction()`, et c'est donc la `valeur` initale (5, en l'occurrence) qui sera affichée par la ligne *5*.

Une référence comme paramètre

Au lieu de créer une variable locale pour y stocker la valeur qui lui est passée, la fonction appelée peut associer une référence à la variable spécifiée par la fonction appelante :

```
1  int main(int argc, char *argv[])
2  {
3  int valeur(5);
4  fonction(valeur);
5  std::cout << "valeur = " << valeur;
6  }
7  //****************************************
8  void fonction(int & truc)
9  {
10 ++truc;
11 std::cout << "truc = " << truc << "\n";
12 }
```

Dans cette version du programme, le mécanisme des paramètres ne crée aucune variable, mais se contente d'associer la référence `truc` à la variable `valeur`.

C'est donc bel et bien cette variable dont le contenu est modifié à la ligne *10*, et l'exécution de la ligne *5* se traduira par l'affichage du nombre 6.

> Lorsque l'exécution de la `fonction()` s'achève, c'est la référence qui est détruite, et non l'objet auquel elle était associée. C'est bien logique, puisque cet objet n'appartient pas à la `fonction()`.

Lorsqu'une fonction dispose d'un paramètre qui est une référence, elle lie cette référence à l'objet qui lui est désigné par la fonction appelante. Celle-ci peut donc ainsi confier provisoirement une de ses variables à la fonction appelée.

3 - Et si on mettait un peu d'ordre ?

Imaginons que nous ayons à écrire un programme qui saisit quatre nombres et les affiche dans l'ordre croissant. Le tri d'une liste de plus de deux valeurs est un problème moins simple qu'il en a l'air.

Plutôt que d'essayer de procéder en une seule fois à toutes les comparaisons et permutations nécessaires, nous pouvons remarquer que, lorsqu'elle est ordonnée, la liste ne présente que des couples de valeurs successives eux mêmes ordonnés. Nous pouvons donc nous contenter de traiter ces couples un à un, en rétablissant l'ordre correct lorsque c'est nécessaire. Bien entendu, cette approche peut exiger que ce traitement des couples de valeurs successives soit répété plusieurs fois. En effet, si, par exemple, la plus petite valeur a été saisie en dernier par l'utilisateur, sa permutation avec sa voisine ne la placera qu'à l'avant dernière place, et non à la première. Il faut donc répéter l'opération tant qu'il reste des couples de valeurs successives non correctement ordonnés.

Nous pourrions écrire :

```
1  int main(int argc, char *argv[])
2  {
3  QCoreApplication a(argc, argv);
4  int n1, n2, n3, n4;
5  std::cout << "Veuillez entre quatre nombres : ";
6  std::cin >> n1 >> n2 >> n3 >> n4;
7  bool triTermine(false);
8  do {
9     triTermine = ordonneCouples(n1, n2, n3, n4);
10    } while (!triTermine);
11 std::cout << "Vos nombres en ordre croissant : " ;
12 std::cout << n1 << " " << n2 << " " << n3 << " " << n4;
13 return a.exec();
14 }
```

1 La ligne *4* utilise une façon abrégée de déclarer plusieurs variables du même type.
La ligne *6* procède, elle aussi, à plusieurs opérations à la fois. Le premier nombre saisi par l'utilisateur ira dans la première variable (`n1`), le second dans la seconde variable, et ainsi de suite. L'utilisateur doit simplement séparer ses valeurs les unes des autres à l'aide d'espaces, de tabulations ou de passages à la ligne.

Il reste évidemment à écrire une fonction `ordonneCouples()` dont les appels répétés finiront par se traduire par le tri complet des valeurs. Comme une liste de quatre valeurs comporte trois couples de valeurs successives, cette fonction devra procéder à trois comparaisons suivies chacune éventuellement d'une permutation.

Un dernier détail doit être noté : pour permuter les valeurs de deux variables, il est nécessaire de disposer d'une troisième variable du même type.

Une séquence du type

```
a=b;
b=a;
```

ne convient évidemment pas car la première instruction a pour effet d'écraser la valeur initiale de a, qui ne pourra donc pas être utilisée par la seconde instruction.

Nous écrirons donc :

```
1  bool ordonneCouples(int &a, int & b, int & c, int & d)
2  {
3  bool caYest = true;
4  if (a > b) {
5    int temporaire(a);
6    a = b;
7    b = temporaire;
8    caYest = false;
9  }
10 if (b > c) {
11   int temporaire(b);
12   b = c;
13   c = temporaire;
14   caYest = false;
15 }
16 if (c > d) {
17   int temporaire(c);
18   c = d;
19   d = temporaire;
20   caYest = false;
21 }
22 return caYest;
23 }
```

4 - Exercice

Créez un projet de type <Application Qt4 en console> et vérifiez que le code suggéré ci-dessus produit bien l'effet recherché □.

La fonction `ordonneCouples()` répète trois fois le même traitement (une fois pour chacun des couples qu'elle doit considérer).

Si vous avez bien compris le fonctionnement du programme, vous devriez être capable de définir une fonction `ordonneUnCouple()` telle que les séquences *4-9*, *10-15* et *16-22* puissent être remplacées chacune par un appel à cette nouvelle fonction □.

Il ne s'agit pas d'essayer de remplacer la fonction `ordonneCouples()`, mais de la simplifier en faisant appel à une troisième fonction au lieu de répéter trois fois la même séquence d'instructions.

Question subsidiaire : à quoi ce programme ressemblerait-il s'il devait être écrit sans utiliser de références ?