Programmation C++ Avancée

Joel Falcou Guillaume Melquiond

21 novembre 2017

1 Création, destruction, copie et élision

Créez une classe `T` possédant un constructeur sans argument, un constructeur de copie, un opérateur d'assignation par copie et un destructeur. Pour chacun de ces quatre membres, ajoutez une ligne de code qui affiche sur la sortie standard qu'il a été appelé. On pourra utiliser le pointeur `this` pour éliminer toute ambiguïté en présence de plusieurs objets :

```
T(T const &t) { std::cout << this << ": constructed from " << &t << '\n'; }
```

Testez votre code sur différents scénarios :

```
void f1(T const &t) {}
void f2(T t) {}
T f3() { return T(); }
T f4() { T t; return t; }
void f5(T &t) { t = T(); }

struct U {
  T v1, v2;
  U(T const &t): v2(t) { v1 = t; }
};

int main() {
  T a;
  f1(a);
  f2(a);
  T b = a;
  T c = f3();
  T d = f4();
  f5(d);
  U e(a);
}
```

Expliquez dans chacun des cas quand l'objet est construit, détruit, copié, etc. Soyez particulièrement attentif aux cas où le compilateur a élidé une ou plusieurs copies.

2 Accès à des fichiers

Le but est ici de fournir une encapsulation en C++ des fonctions C d'accès aux fichiers : `fopen`, `fwrite`, `fclose`, disponibles dans `<cstdio>`. On pourra lire les pages de manuel de ces fonctions pour leur description, par exemple `man fopen` en ligne de commande.

Créez une classe C++ `file` avec un champ de type `FILE *`. Définissez un constructeur prenant une chaîne de caractères et ouvrant le fichier correspondant.

Définissez une méthode `file::write` prenant une chaîne de caractère an argument et l'écrivant dans le fichier ouvert précédemment. Note : la méthode `std::string::c_str` permet d'accéder à la chaîne C qu'attend `fwrite`.

Testez votre classe sur un exemple du genre suivant. Vérifiez que les fichiers ainsi créés contiennent bien les chaînes attendues.

```
int main() {
  file f("test1.txt");
  f.write("first string for test1\n");
  file g("test2.txt");
  g.write("first string for test2\n");
  f.write("second string for test1\n");
  return 0;
}
```

Définissez un destructeur `file::~file` qui appelle `fclose` pour fermer proprement le fichier ouvert par le constructeur.

Question : quel est le comportement du constructeur de copie créé par défaut par le compilateur pour la classe `file` ? Quel est le problème avec ce comportement ?

Écrivez un programme qui plante violemment à l'exécution suite à une copie.

Appliquez l'attribut `= delete` au constructeur de copie de la classe `file` pour le désactiver. Vérifiez que le compilateur rejette maintenant votre programme incorrect.

Constatez que le même problème existe avec l'opérateur d'assignation `file::operator=`. Corrigez-le de la même manière que le constructeur de copie.

3 Sémantique de transfert

La classe `file` précédente ne permet ni construction par copie ni assignation par copie. Il est par contre facile de fournir une sémantique de transfert.

Expliquez en quoi consiste un transfert entre deux objets de la classe `file`. Cela nécessite-t-il de modifier le code du destructeur ?

Ajoutez un constructeur par transfert `file::file(file &&)`. Testez-le sur le code suivant. Note : le code doit être modifié pour être compilé sans erreur.

```
int main() {
  file f("test3.txt");
  f.write("first string for test3\n");
  file g = f;
  g.write("second string for test3\n");
  return 0;
}
```

Question : est-il possible d'écrire un code incorrect utilisant la classe `file` ?