Les erreurs en programmation

Les types d'erreur

Une des tâches principales des développeurs est de gérer les “erreurs”, au sens large. Tout le monde fait des erreurs (même vous !) et il faut savoir gérer cela quand vous écrivez un programme. Il existe différents types d'erreurs : les erreurs faites par les développeurs quand ils écrivent du code, et les erreurs faites par les utilisateurs de vos programmes.

Les erreurs faites par les utilisateurs correspondent à des erreurs d'entrées et sorties : saisir du texte quand vous demander de saisir un nombre, cliquez en dehors d'une fenêtre quand vous demandez de cliquer sur un bouton, ouvrir un fichier image quand vous demander d'ouvrir un fichier texte, etc.

Un programme *fiable* est un programme qui continuera de fonctionner correctement, même si les utilisateurs font n'importe quoi. Gérer ce type d'erreur fait donc partie du **comportement attendu** d'un programme, qui doit être prévu par les développeurs. Plusieurs chapitres de ce cours seront consacrés à ce problème.

Le second type d'erreur est celles faites par les développeurs lorsqu'ils écrivent du code. Ce type d'erreur correspond a un **comportement non attendu** d'un programme. Ou pour etre plus precis, le comportement obtenu sera différent du comportement qu'avait prévu le développeur.

Ce type d'erreur est en principe simple a tester. Il “suffit” d'exécuter le programme et de comparer le comportement obtenu avec ce qui était prévu. En pratique, il est extrêmement difficile de prévoir tous les comportements, dans toutes les situations. Et il est également difficile (et cela prend beaucoup de temps) d'écrire tous ces tests.

Et même en écrivant correctement tous les tests, cela n'est pas suffisant

pour avoir un programme fiable a 100%. En particulier parce qu'il y a un concept important en C++ : les comportements indefinis (UB, pour *Undefined Behavior*). Un comportement indéfini signifie que le programme peut faire tout et n'importe quoi, une fois qu'il est entré dans un comportement indéfini. Il n'y a aucune garantie de ce que va faire le programme par le suite. En particulier, il peut avoir un comportement qui semble etre le comportement correct !

Une (mauvaise) approche souvent utilisée par les débutants (et les développeurs peu consciencieux) est de lancer le programme puis de faire quelques tests. Cette approche est mauvaise parce que cela ne permet pas de tester suffisamment le code et il est facile d'oublier de nombreux tests.

- un programme non testé ne peut pas etre considéré comme correct ;
- un programme correctement testé ne sera jamais fiable a 100%.

La question n'est donc pas de savoir comment écrire un code fiable a 100% ou comment écrire des tests. Pour écrire un programme de qualité, il faut entrer dans une démarche visant à rechercher de façon systématique les causes possibles d'erreurs.

Dans la suite de ce cours, vous verrez l'approche de développement piloté par les tests (TDD, pour*Test driven development*), consiste à écrire les tests avant le code, pour renforcer la qualité du code.

Les assertions

Dans la suite de ce cours, les codes d'exemple utilisent les assertions pour tester le comportement des codes. Une assertion est simplement un test sur un booléen, qui arrête le programme si le test est faux.

Il existe deux types d'assertion :

- `assert(CONDITION)` a l'execution ;
- `static_assert(CONDITION, MESSAGE)` a la compilation.

Pour utiliser `assert`, il faut inclure le fichier d'en-tête `<cassert>`.

Voici un exemple d'utilisation de `assert` :

main.cpp

```
#include <iostream>
#include <cassert>

int main() {
    assert(1 + 2 == 4);
}
```

affiche le message suivant lors de l'exécution :

```
a.out: main.cpp:5: int main(): Assertion '1 + 2 == 4' failed.
```

Avertissements et erreurs du compilateur

- compilateur et analyse statique/dynamique, sera vu dans la suite du cours. Oui, mais 1. faut activer les warnings. .2. faut savoir utiliser les outils d'analyse. 3. ne permet pas de tout verifier.

debug, trouver les erreurs

assert

Chapitre précédent	Sommaire principal	Chapitre suivant