

Chapitre V

Exceptions

- 1 Définition et syntaxe
- 2 Exemple
- 3 Détails

Comment gérer les erreurs ?

- En C, les fonctions retournent habituellement un `int` qui est égal à 0 si la fonction a été exécutée correctement ou un **code d'erreur** si une erreur s'est produite. . . mais pas toujours : `fopen`, `malloc`, `read` retournent une valeur qui a un sens, et non un code d'erreur.
- Pour écrire un programme **robuste** il faut (faudrait) toujours tester le code de retour des fonctions appelées et agir en conséquence. ⇒ Lourd à écrire ⇒ On ne le fait pas toujours.

Pour gérer les erreurs en C++, on utilise le mécanisme des **exceptions**, nettement plus agréable à utiliser.

Les fonctions peuvent retourner une valeur, et les erreurs sont « remontées » par un canal différent de la valeur retournée par une fonction.

Chapitre V

Exceptions

- 1 Définition et syntaxe
- 2 Exemple
- 3 Détails

Définition

Une exception est un objet qui est créé (« **lever** une exception ») et qui :

- **Interrompt** l'exécution du bloc d'instructions qui a levé l'exception.
- **Remonte la pile d'appels** des fonctions/méthodes. . .
- jusqu'à trouver un **bloc de gestion** de l'exception.
- Si aucun bloc de gestion n'est trouvé, l'exception « sort » du `main` et le programme se termine.

Usage

- Si une fonction se termine sans erreur...
 - Elle peut retourner une valeur.
- Si une fonction provoque une erreur...
 - Elle ne se termine pas.
Les instructions qui suivent la levée de l'exception ne sont pas exécutées.
 - Elle ne retourne **aucune valeur**.
 - Elle lève une exception.
 - Cette exception peut être gérée dans un bloc de gestion d'exceptions
 - Qui peut gérer différents types d'exceptions
 - Un seul bloc peut gérer les erreurs provoquées par **plusieurs** appels de fonctions.
 - Ce bloc n'est **pas forcément** dans le bloc qui a appelé la fonction

Syntaxe

Syntaxe

- Levée d'exception
`throw ObjetException ;`
- Bloc de gestion d'exceptions

```

try
{
  Instructions à protéger
}
catch (TypeException1 e1)
{
  Gestion de l'exception de type TypeException1
}
[catch (TypeException2 e2)
{
  Gestion de l'exception de type TypeException2
}]*
```

Chapitre V

Exceptions

- 1 Définition et syntaxe
- 2 Exemple
- 3 Détails

Exemple

Exemple

```
float division(int a, int b)
{ if (b == 0) throw 1;
  else return static_cast<float>(a) / b; }

void test()
{ int a, b;
  cin >> a; cin >> b;
  cout << division(a, b) << endl;
  cout << "Calcul fini" << endl; }

int main()
{ try
  { test();
    cout << "Test exécuté" << endl; }
  catch (int i)
  { cout << "Erreur détectée " << i << endl; }
  return 0; }
```

Exemple

Exemples d'exécution :

- Sans exception


```
2 3
0.666667
Calcul fini
Test exécuté
```
- Avec exception


```
2 0
Erreur détectée 1
```
- Sans le bloc try du main


```
2 0
terminate called after throwing an instance of 'int'
Abandon
```

Remarques et conseils

- En C++, **tout type** (primitif et classe) peut être utilisé comme exception.
- Il est conseillé de ne pas lever d'exceptions dans un constructeur ou un destructeur.
- Il est conseillé de lever une exception par **valeur**. Par contre, on peut gérer l'exception dans un catch par référence (constante).
- Dans la plupart des cas, on n'utilisera pas de types primitifs pour les exceptions mais des types spécialement créés à cet effet.

Chapitre V

Exceptions

- 1 Définition et syntaxe
- 2 Exemple
- 3 Détails

Ordre des catch

S'il y a plusieurs blocs `catch`, le **premier** capable de gérer l'exception est exécuté.

⇒ Dans le cas de classes exceptions avec relation d'héritage, l'ordre des `catch` est important.

⇒ Écrire les blocs `catch` de l'exception la plus spécifique à la plus générique.

Ordre des catch

Exemple

Exemple

```
class MonException
{
public:
int m_gravite;
public:
MonException(int g)
:m_gravite(g) {};
};

class MonExceptionFichier: public MonException
{
public:
string m_nomfichier;
public:
MonExceptionFichier(int g, string const & nf)
:MonException(g), m_nomfichier(nf) {};
};
```

Ordre des catch - Exemple

Exemple

```
int main()
{
try
{ throw MonExceptionFichier(3, "f.txt"); }
catch (MonException const & e)
{ cout << "MonException" << endl; }
catch (MonExceptionFichier const & e)
{ cout << "MonExceptionFichier" << endl; }
return 0;
}
```

warning: exception of type MonExceptionFichier will be caught by earlier handler for MonException
 ⇒ MonException

throw

- Dans un bloc `catch`, il est possible de lever une exception.
- Il est possible aussi de « lever à nouveau » l'exception qui a entraîné l'exécution du bloc `catch` par un simple `throw` sans argument.

Ceci permet d'effectuer des traitements particuliers en cas d'erreur... sans gérer l'erreur.

Ou de tester en fonction de la **valeur** de l'exception si celle-ci peut être traitée... si elle ne peut pas l'être, elle est levée à nouveau.

Exemple

```
catch (MonExceptionFichier const & e)
{ if (e.m_gravite < 5)
  cerr << "Attention à " << e.m_nomfichier;
  else throw; }
```

Bloc de gestion par défaut

`catch(...)` permet de gérer **tous les types d'exceptions** qui n'ont pas été gérés par les blocs `catch` précédents.

- Il est **optionnel**.
- Il est toujours utilisé comme **dernier** `catch` associé à un `try`.

Levée d'exception dans une fonction/méthode

Il n'est pas **nécessaire** (comme en Java avec `throws`) de déclarer les exceptions pouvant être levées dans une fonction/méthode.

- La signature d'une fonction méthode **peut** déclarer les exceptions susceptibles d'être levées.
 - `throw()`
Aucune exception ne peut être levée.
 - `throw(type1, type2, ...)`
Des exceptions de ces types-là peuvent être levées
 - aucun `throw()`
Des exceptions de tous types peuvent être levées.
- `throw()` est assez rarement utilisé en C++, car aucun contrôle n'est fait à la compilation.

Classes d'exceptions std

La bibliothèque standard définit quelques classes d'exceptions.

`std::exception` racine des classes d'exceptions.

Il est *conseillé* de définir les nouvelles classes exceptions comme sous-classes de `std::exception`. Pour cela, on redéfinira (au moins) la méthode `char const * what() const throw()`. Cette méthode est appelée dans le cas d'une exception qui sort du `main` pour afficher le message d'erreur.

`std::bad_alloc` levée lors d'une erreur rencontrée par `new`.

`std::bad_cast` levée lors d'un `dynamic_cast` impossible vers une référence d'une sous-classe.

...