Programmation C++ Avancée

Session 4 – Gestions des Erreurs

Joel Falcou Guillaume Melquiond

Laboratoire de Recherche en Informatique

Gestion des Erreurs

Quoi et Pourquoi

- Valider l'état d'un objet avant l'appel d'une fonction
- Rapporter des erreurs inattendues
- Rapporter des événements indépendant du code

Mise en œuvre

- Assertion
- Exception

Assertion

Principe

- Permet de valider les pré-conditions d'une fonction
- Fournie par <cassert>
- Termine le programme en Debug
- Disparaît en Release

```
#include <cassert>

float f(int x)
{
  assert(x != 0 && "x must be non-null");
  return 1.f/x;
}
```

Assertion

Principe

- Permet de valider les pré-conditions d'une fonction
- Fournie par <cassert>
- Termine le programme en Debug
- Disparaît en Release

```
int main()
{
  auto x = f(78);
  auto y = f(0);
}
```

Exception

Exceptions définies par les utilisateurs

- Effectue une sortie anticipée de portée
- Une exception = un type

```
#include <stdexcept>

void f(int)
{
  throw std::runtime_exception(”SEVEN!!!”);
}
```

Exception

Exceptions définies par les utilisateurs

- Effectue une sortie anticipée de portée
- Une exception = un type

```
int main()
{
  try
  {
    f(0);
  }
  catch( std::exception& e )
  { std::cout << e.what() << "\n"; }
}
```

Garantie d'Exception

Principe

- Valide les post-conditions exceptionnelles d'une fonction
- Propriété composable
- Limite l'usage du bloc `try`

Quelles garanties ?

- NOEXCEPT
- STRONG
- BASIC

Garantie d'Exception

Niveau NOEXCEPT

- Comportement normal garanti dans tous les cas
- Aucune exception ne sera émise
- Les exceptions internes ne sont pas observables
- Mot-clé noexcept

```
void swap(int& a, int& b)
{
  int t{a};
  a = b;
  b = t;
}
```

Garantie d'Exception

Niveau NOEXCEPT

- Comportement normal garanti dans tous les cas
- Aucune exception ne sera émise
- Les exceptions internes ne sont pas observables
- Mot-clé noexcept

```
void swap(int& a, int& b) noexcept
{
  int t{a};
  a = b;
  b = t;
}
```

Garantie d'Exception

Niveau NOEXCEPT

- Comportement normal garanti dans tous les cas
- Aucune exception ne sera émise
- Les exceptions internes ne sont pas observables
- Mot-clé noexcept

```
template<typename T> T f(T x)  // noexcept ?
{
  return g(g(x));
}
```

Garantie d'Exception

Niveau NOEXCEPT

- Comportement normal garanti dans tous les cas
- Aucune exception ne sera émise
- Les exceptions internes ne sont pas observables
- Mot-clé noexcept

```
template<typename T> T f(T x) noexcept( noexcept(g(x)) )
{
  return g(g(x));
}
```

Garantie d'Exception

Niveau STRONG

- Comportement exceptionnel permis
- En cas d'exception, les données originelles ne sont pas modifiées
- L'objet reste valide et cohérent

Garantie d'Exception

Niveau BASIC

- Une exécution partielle de la fonction est possible
- Des effets de bords peuvent avoir lieu
- Aucune fuite de ressources
- L'état de l'objet est valide mais pas forcément cohérent

Mise en pratique

```
struct A
{
   A& operator=( A const& a )
   {
      // STRONG ...


      return *this;
   }
};
```

Mise en pratique

```
struct A
{
   A& operator=( A const& a )
   {
      A tmp(a);
      this->swap(tmp);

      return *this;
   }
};
```

Mise en pratique

```
vector::vector( vector const& src )
{
   size_ = src.size();
   double* tmp = new double[src.size()];
   std::copy(tmp, tmp+src.size(), src.data());
   data_ = tmp;
}
```

Mise en pratique

```
vector::vector( vector const& src )
{
  unique_ptr<double[]> tmp = new double[src.size()];
  std::copy(tmp.get(), tmp.get()+src.size(), src.data());

  data_ = tmp.release();
  size_ = src.size();
}
```

RAII et capture

```
void write_to_file (const std::string & message)
{
  static std::mutex mutex;

  std::lock_guard<std::mutex> lock(mutex);

  std::ofstream file("example.txt");
  if (!file.is_open())
    throw std::runtime_error("unable to open file");

  file << message << std::endl;
}
```