LES EXCEPTIONS

- **Généralités**
- **Schéma du mécanisme d'exception**
- **La structure try ... catch**
- **Syntaxe du catch**
- **Syntaxe de throw**
- **Déclaration des exceptions levées par une fonction**
- **La fonction terminate()**
- **La fonction unexpected()**
- **Exemple complet**

Généralités sur les exceptions

Généralités :

Le langage C++ nous propose une gestion efficace des erreurs pouvant survenir pendant l'exécution :

- erreurs matérielles : saturation mémoire, disque plein ...
- erreurs logicielles : division par zéro ...

La solution habituellement pratiquée est l'affichage d'un message d'erreur et la sortie du programme avec le renvoi d'un code d'erreur au programme appelant.

Exemple :

```
void lire_fichier(const char *nom) {
  ifstream f_in(nom); // ouverture en lecture
  if ( ! f_in.good()) {
    cerr << "Problème à l'ouverture du fichier " << nom << endl;
    exit(1);
```

```
  }
  // lecture du fichier ...
}
```

En C++, une nouvelle structure de contrôle permet donc la gestion des erreurs d'exécution, la structure ***try ... catch*** .

Schéma du mécanisme d'exception

---

1. constuction d'un objet (d'un type quelconque) qui représente l'erreur
2. lancement de l'exception (*throw*)
3. l'exception est alors propagée dans la structure de contrôle try ... catch englobante
4. cette structure essaye attraper (*catch*) l'objet
5. si elle n'y parvient pas, la fonction *terminate()* est appelée.

```
try {
  // ...
  throw objet  // lancement de l'exception
}
catch ( type ) {
  // traitement de l'erreur
}
```

La structure try ... catch

- Quand une exception est détectée dans un bloc *try*, le contrôle de l'exécution est donné au bloc catch correspondant au type de l'exception (s'il existe).
- Un bloc *try* doit être suivi d'au moins un bloc *catch*
- Si plusieurs blocs *catch* existent, ils doivent intercepter un type d'exception différent.
- Quand une exception est détectée, les destructeurs des objets inclus dans le bloc *try* sont appelés avant d'appeler un bloc *catch*.
- A la fin du bloc *catch*, le programme continue son exécution sur l'instruction qui suit le dernier bloc *catch*.

Exemple : interception d'une exception de type xalloc

```
#include <except.h>

// ...

try {
  char *ptr = new char[1000000000];  /
  // ... suite en cas de succès de new (improbable ...)

}
catch ( xalloc ) {
  // en cas d'échec d'allocation mémoire par new
  // une exception xalloc est lancée par new

  // traitement de l'erreur d'allocation
}
```

Syntaxe de catch

- **catch (TYPE)** : intercepte les exceptions du type TYPE, ainsi que celles de ses classes dérivées.
- **catch (TYPE o)** : intercepte les exceptions du type TYPE, ainsi que celles de ses classes dérivées. Dans le catch un objet o est utilisable pour extraire d'autres informations sur l'exception.
- **catch (...)** : intercepte les exceptions de tous types, non traitées par les bocs catch précédents.

Exemple :

```
class Erreur {
  // ...
  int get_erreur() { /* ... */ }
};

try {
  // instruction(s) douteuse(s) qui peu(ven)t lancer une exception
}
catch ( xalloc ) {
  // ...
}
catch ( Erreur err ) {
  cerr << err.get_erreur() << endl;
  // ...
}
catch (...) {
  // ...
}
```

Syntaxe de throw

---

1. **throw obj** : elle permet de lever une exception donnée

   Exemple :

```
#include <except.h>
#include <iostream.h>

class Essai {
  public :
    class Erreur {  };
    // ...
    void f1() {
      throw Erreur(); // construction d'une instance de Erreur
                      // et lancement de celle-ci
    }
};

void main() {
  try {
    Essai e1;
    e1.f1();
  }
  catch ( Essai::Erreur ) {
    cout << "interception de Erreur" << endl;
  }
}
```

2. **throw** : l'instruction throw sans paramètre s'utilise si l'on ne parvient pas à resoudre une exception dans un bloc catch. Elle permet de relancer la dernière exception.

Déclaration des exceptions levées par une fonction

Cette déclaration permet de spécifier le type des exceptions pouvant être éventuellement levées par votre fonction (ou méthode).

Exemples de déclaration :

- void f1() **throw** (Erreur);
- void f2() **throw** (Erreur, Division_zero);
- void f3() **throw** ();

- void f4();

Remarques :

- Par défaut de déclaration (comme dans le cas de la fonction f4()), toute exception peut être lancée par une fonction.
- La fonction f3(), déclarée ci dessus, ne peut lancer aucune exception.
- La fonction f1() ne peut lancer que des exceptions de la classe Erreur (ou de classes dérivées).
- Si une fonction lance une exception non déclarée dans son entête, la fonction *unexpected()* est appelée.
- A la définition de la fonction, *throw* et ses paramètres doivent être de nouveau spécifié.

La fonction terminate()

Si aucun bloc *catch* ne peut attraper l'exception lancée, la fonction *terminate()* est alors appelée. Par défaut elle met fin au programme par un appel à la fonction *abort()*.

On peut définir sa propre fonction *terminate()* par un appel à la fonction *set_terminate()* définie dans *except.h* comme :

```
typedef void (*terminate_function)();
terminate_function set_terminate(terminate_function t_func);
```

Exemple :

```
#include <except.h>
#include <iostream.h>

void my_terminate() {
  cout << "my_terminate" << endl;
  exit(1);  // elle ne doit pas retourner à son appelant
}

void main() {
  try {
     set_terminate((terminate_function) my_terminate);
     // ...
  }
  catch ( Erreur ) {
    // ...
  }

}
```

La fonction unexpected()

Si une fonction (ou une méthode) lance une exception qui n'est pas déclarée par un *throw* dans son entête, la fonction *unexpected()* est alors appelée. Par défaut elle met fin au programme par un appel à la fonction *abort()*.

On peut définir sa propre fonction *unexpected()* par un appel à la fonction *set_unexpected()* définie dans *except.h* comme :

```
typedef void (*unexpected_function)();
unexpected_function set_unexpected(unexpected_function t_func);
```

Exemple :

```
#include <except.h>
#include <iostream.h>

void my_unexpected() {
  cout << "my_unexpected" << endl;
  exit(1);   // elle ne doit pas retourner à son appelant
}

class Erreur {};
class Toto {};

class Essai {
  public:
    void f1() throw (Erreur);
};

void Essai::f1() throw (Erreur) {
  throw Toto();
}

void main() {
  try {
     set_unexpected( (unexpected_function) my_unexpected);
     Essai e1;
     e1.f1();
  }
  catch ( Erreur ) {
    // ...
  }
}
```

Exemple complet

```
#include <except.h>
#include <iostream.h>

class Essai {
  public :
    class Erreur {
      public:
        Erreur(int n=0): _val(n) { }
        int get_val() { return _val; }
      private:
        int _val;
    };
    Essai() { cout << "Constructeur d'Essai" << endl; }
    ~Essai() { cout << "destructeur d'Essai" << endl; }
    // ...
    void f1() {
      throw Erreur(10); // construction d'une instance de Erreur
                        // initialiée à 10 et lancement de celle-ci
  }
};

void main() {
  try {
    Essai e1;
    e1.f1();
    cout << "bla bla bla" << endl;    //
  }
  catch ( Essai::Erreur e ) {
    cout << "Erreur numéro : " << e.get_val() << endl;
  }
}

/* résultat de l'exécution ***********
Constructeur d'Essai
destructeur d'Essai
Erreur numéro : 10
*******************************/
```
