COLLEGE DE PARIS SUPÉRIEUR

DAKARTECH

Programmation orientée objets [C++]

Dr. Pape Abdoulaye BARRO

La POO

- Notion de Classe
- Fonctions membres
- Constructeurs, destructeurs et initialisation d'objet
- Héritage simple
- Héritage multiple
- Fonctions virtuelles et le polymorphisme
- **La gestion des exceptions**
- Généralités sur la STL

La gestion des exceptions

Généralités

- Quelques fois, même si le programme est correct, il peut y survenir des risques pouvant compromettre son bon exécution.
- En C++, avec sa bibliothèque standard, les erreurs sont fournies par des codes qui constituent des valeurs de retour des différentes fonctions. En examinant ces valeurs de retour en de nombreuses points du programme, il est alors possible de capturer les éventuelles anomalies (**exceptions**) et puis de procéder à leurs traitements.
- ***En production***, il est utile de séparer le code et la détection/correction des erreurs.

La gestion des exceptions

Généralités

- Une exception est une rupture de séquence qui peut être déclenchée par une instruction ***throw***, comportant une expression quelconque dont le type (**classe** ou non) sert à identifier l'exception en tant que telle.
- Il est recommandable, pour chaque exception donnée, de proposer une classe afin de mieux représenter l'anomalie concernée.
- Exemple d'erreur avec "**division par zéro**":

Erreur

```
#include <iostream>
using namespace std;

int main() {
        int x,a=5,b=0;
        x = a/b;
        cout << "x= " << x << endl;

        return 0;
}
```

La gestion des exceptions

Lever des exceptions avec « throw »

- En C++, la gestion des exceptions repose essentiellement sur *trois mots-clés* : **try**, **catch** et **throw**. Avec **try** qui définit le bloc de code test, **catch** qui définit le bloc de code à exécuter en cas de détection d'erreur et **throw** qui permet de lever une exception en cas de problème (on peut créer une erreur personnalisée à cet effet).
- **Lever des exceptions avec « throw »**:
  en réutilisant l'exemple d'erreur précèdent, il est possible d'identifier la nature de l'erreur. **La syntaxe est la suivante**: **throw expression**;

```
#include <iostream>
#include <string>

using namespace std;

int test_division(int a, int b)
{
        string x = " erreur: division par zero";
        if (b==0) {
                throw x;
        }
        return a/b;
}
```

```
int main() {
        int x,a=5,b=0;

        x = test_division(a, b);
        cout << "x= " << x << endl;

        return 0;
}
// Si une exception est levée et n'est interceptée nulle part, le programme se termine anormalement.
```

La gestion des exceptions

Intercepter/traiter une exception avec « try/catch »

- **Intercepter/traiter une exception avec « try/catch »** :
  Dans catch, entre parenthèses, il est possible de spécifier le type d'exception à intercepter et puis de personnaliser le message d'erreur.
  - **La syntaxe est la suivante**:

```
try {
    // bloc d'instruction à protégé
} catch( NomDeException e ) {
    // bloc d'instruction pour la gestion de l'exception
}
```

  - Reprenons l'exemple précèdent:

```
#include <iostream>
#include <string>

using namespace std;

int test_division(int a, int b)
{
    string x = "erreur: division par zero";
    if (b==0) {
        throw x;
    }
    return a/b;
}
```

```
int main() {
    int x,a=5,b=0;
    try{
        x = test_division(a, b);
        cout << "x= " << x << endl;
    }catch(string s){
        cout << s << endl;
    }
    return 0;
}
```

La gestion des exceptions

Identification et interception d'une exception via une classe

- **Identification et interception d'une exception via une classe** :

  Il est possible de créer une classe pour caractériser un exception.

  - En utilisant le mot clé **throw** puis l'objet correspondant à l'exception, il est possible d'identifier le type d'erreur!: (throw objet;)
  - Puis de l'intercepter (**try**) et de le traiter (**catch**).

```
try{
    // bloc d'instruction à protégé
}catch (classe &o){
    // Traitement de l'exception associée à la classe
}
```

  - Exemple de classe d'exception:

```
class erreur {
        public:
            string cause;
            erreur(string s) : cause(s) { }
            // Le constructeur de copie (nécessaire pour le catch):
            erreur(const erreur &e) : cause(e.cause) { }
};
```

La gestion des exceptions

Identification et interception d'une exception via une classe

- **Utilisation dans un main**

```
#include <iostream>
#include <string>
using namespace std;
// class erreur

int test_division(int a, int b)
{
        erreur x ("division par zero !");
        if (b==0) {
                throw x;
        }
        return a/b;
}

int main(){
    int a,b
    cout<<"a=";
    cin>>a;
    cout<<"b=";
    cin>>b;
    // ...
    // suite

    try{
        cout << "resultat= " << test_division(a, b) << endl;
    }
    catch (erreur &e)
    {
        // récupération de la cause.
        cout << "Erreur, cause: " << e.cause << endl;
    }
    catch (...)
    {
        // le gestionnaire d'exception universel
        // pour toutes autres erreurs
        cout << "Erreur inattendue !" << endl;
    }

    return 0;
}
```

La gestion des exceptions
standard

- C++ nous fournit un certains nombres d'exceptions standard qui sont définies dans la classe ***exception*** qui est la classe de base de toutes les exceptions lancées par la bibliothèque standard.
- Elle est définit comme suite:

```
class exception
{
        public:
            //Constructeur.
            exception() throw(){ }
            //Destructeur.
            virtual ~exception() throw();
            // what envoie une chaîne contenant des infos sur l'erreur.
            virtual const char* what() const throw();
};
```

- **throw** indique que ces méthodes ne vont pas lancer d'exceptions

La gestion des exceptions
standard

- Il devient possible de créer sa propre classe d'exception en héritant de la classe **exception**.
- Réécrivons la classe erreur:

```
class erreur: public exception
{
        private:
            int num;
            string msg;

        public:
            erreur(int n=0, string const& m=" ") throw() : num(n),msg(m) { }
            virtual ~erreur() throw() { }
            virtual const char* what() const throw() {
                    return msg.c_str();
            }
};
```

La gestion des exceptions

standard

- **Utilisation dans un main**

```
#include <iostream>
#include <string>
#include <exception>
using namespace std;
// class erreur

int test_division(int a, int b)
{
        if (b==0) {
                throw erreur(1,"division par zero !");
        }
        return a/b;
}

int main(){
    int a,b
    cout<<"a=";
    cin>>a;
    cout<<"b=";
    cin>>b;
    // ...
    // suite

    try{
        cout << "resultat= " << test_division(a, b) << endl;
    }
    catch (exception const& e)
    {
        // récupération de la cause.
        cout << "Erreur, cause: " << e. what()<< endl;
    }
    catch (...)
    {
         cout << "Erreur inattendue !" << endl;
    }

    return 0;
}
```

La gestion des exceptions

standard

- La bibliothèque standard est capable de lancer 5 types d'exceptions que sont:
  - **std::bad_alloc** – erreur lancée par new (en mémoire).
  - **std::bad_cast** - erreur lancée lors d'un dynamic_cast
  - **std::bad_typeid** – erreur lancée lors d'un typeid
  - **std::bad_exception** – erreur lancée si aucun catch ne correspond à un objet lancé
  - **ios_base::failure** – erreur lancée en manipulant un flux
- Si on ne souhaite pas créer une classe pour gérer les exceptions, on pourra faire appel au fichier standard ***stdexcept*** contenant des classes d'exceptions pour les erreurs les plus courants.
- Il contient exactement 9 classes subdivisées en 2 catégories: les **logic errors** et les **runtime errors.**
- les **logic errors** sont: domain_error, invalid_argument, length_error, out_of_range et logic_error (toutes autres erreurs logiques);
- les **runtime errors** sont: range_error, overflow_error, underflow_error, runtime_error (toutes autres erreurs d'exécution).

La gestion des exceptions

exemple avec *domain_error*

- **exemple avec *domain_error***

```
#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>
using namespace std;
int test_division(int a,int b) {
   if(b==0)
      throw domain_error("division par zero !");
   else
      return a/b;
}
int main(){
   int a=4, b=0;
    try{
        cout << "resultat= " << test_division(a, b) << endl;
    } catch (exception const& e) {
        // récupération de la cause.
        cout << "Erreur, cause: " << e. what()<< endl;
        throw; //relance de l'exeption reçue pour la traiter une deuxiéme fois, plus loin dans le code.
     }
    return 0;
}
```

FIN

pape.abdoulaye.barro@gmail.com