

Les exceptions

Introduction au C++ et à la programmation objet

E. Courcelle

CALMIP, UMS 3669

Décembre 2019

Que faire en d'erreur ?

Lorsqu'un objet rencontre une condition d'erreur, que doit-il faire ?

- Prendre une décision ? **Surtout pas !**

L'objet **ne sait pas** comment l'erreur doit être traitée: tout dépend du contexte, or l'objet doit fonctionner dans des contextes différents.

L'objet doit:

- Soit traiter l'erreur lui-même, s'il le peut
- Soit **signaler** au programme appelant qu'il a rencontré une situation d'exception

Comment signaler une condition d'exception ?

On peut signaler une condition d'exception par trois moyens:

- Terminer la fonction en retournant un code d'erreur: **Pas toujours possible !**
(quel code renverra la fonction `sqrt` ?)
- Déposer un code d'erreur dans une variable globale (`errno` en C). **Possible mais lourd**
- Générer une exception; **Le plus souple**

Qu'est-ce qu'une exception ?

C'est un **objet** qui va contenir des informations sur l'exception à condition que le programmeur les y mette:

- Message d'erreur
- Code numérique d'erreur
- Numéro de ligne, nom de fichier, ...

Cet objet sera "**lancé**" comme une bouteille à la mer par la fonction qui a rencontré la condition d'exception, puis la fonction est interrompue.

L'objet va alors **remonter la pile d'appels**, et chaque fonction appelante a le choix entre:

- Intercepter l'exception et la traiter
- L'ignorer, dans ce cas l'objet passera au niveau supérieur, mais la fonction sera interrompue

Si aucune fonction ne traite l'exception, **le programme lui-même est interrompu** par le système d'exploitation.

Lancer une exception

La fonction suivante implémente la division d'un complexe par un flottant:

```
complexe& complexe::operator/=(float x) {  
    r /= x;  
    i /= x;  
    return *this;  
}
```

Si on appelle la fonction ci-dessus avec $x=0$, le programme s'arrêtera brutalement avec un message d'erreur. Dans la version ci-dessous, on laisse au programmeur la possibilité de gérer ce cas:

```
complexe& complexe::operator/=(float x) {  
    if ( x == 0 ) throw ( "division par zero" );  
    r /= x;  
    i /= x;  
    return *this;  
}
```

Attraper et afficher une exception

La fonction `main` ci-dessous propose un traitement d'erreur:

```
int main() {  
    complexe c(5,6);  
    try  
    {  
        float x;  
        cout << "Entrez un diviseur: ";  
        cin >> x;  
        c /= x;  
    }  
    catch ( const char * e )  
    {  
        cout << e << "\n";  
    }  
    return 0;  
}
```

Attraper et traiter une exception

La fonction main ci-dessous essaie de traiter l'exception et de réagir en conséquence:

```
int main() {
    complexe c(5,6);
    do
    {
        try
        {
            float x;
            cout << "Entrez un diviseur: ";
            cin >> x;
            c /= x;
            break;
        }
        catch ( const char * msg )
        {
            cout << msg << " Recommencez\n";
        }
    } while (true);
    return 0;
}
```


Le traitement peut se faire à différents niveaux

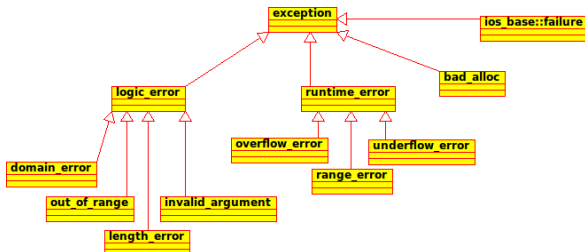
La fonction main ci-dessous ne traite rien, car le traitement est fait dans la fonction inputEtDivise

```
void inputEtDivise(complexe& c) {
    do
    {
        try
        {
            float x;
            cout << "Entrez un diviseur: ";
            cin >> x;
            c /= x;
            break;
        }
        catch ( const char * msg )
        {
            cout << msg << " Recommencez\n";
        }
    } while ( true );
}

int main() {
    complexe c(5,6);
    inputEtDivise(c);
    cout << "Partie reelle      : " << c.get_r() << "\n";
    cout << "Partie imaginaire: " << c.get_i() << "\n";
}
```

La hiérarchie d'exceptions

- Il est préférable d'envoyer des **objets**, et pas simplement une chaîne de caractères.
- Il est par ailleurs **plus que conseillé** d'utiliser la **hiérarchie d'exceptions** prédéfinie par la bibliothèque standard du C++



Envoyer un objet exception

La fonction de division:

```
complexe& complexe::operator/=(float x) {  
    if ( x == 0 ) {  
        domain_error e ( "division par zero" );  
        throw e;  
    }  
    r /= x;  
    i /= x;  
    return *this;  
}
```

Soit, en plus concis:

```
complexe& complexe::operator/=(float x) {  
    if ( x == 0 ) {  
        throw domain_error ( "division par zero" );  
    }  
    r /= x;  
    i /= x;  
    return *this;  
}
```

Des traitements plus ou moins fins

Une nouvelle version de la première fonction main:

```
int main() {  
    complexe c(5,6);  
    try  
    {  
        float x;  
        cout << "Entrez un diviseur: ";  
        cin >> x;  
        c /= x;  
    }  
    catch ( exception & e )  
    {  
        cout << e.what() << "\n";  
    }  
    return 0;  
}
```

Des traitements plus ou moins fins (2)

Une nouvelle version de la seconde fonction main:

```
void inputEtDivise(complexe& c) {
    do
    {
        try
        {
            float x;
            cout << "Entrez un diviseur: ";
            cin >> x;
            c /= x;
            break;
        }
        catch ( const domain_error& e )
        {
            cout << e.what() << " Recommencez\n";
        }
    } while (true);
}

int main() {
    complexe c(5,6);
    try
    {
        inputEtDivise(c);
        cout << "Partie reelle      : " << c.get_r() << "\n";
        cout << "Partie imaginaire: " << c.get_i() << "\n";
    }
    catch ( const exception& e )
    {
        cout << e.what() << "\n";
    }
}
```

Des traitements plus ou moins fins

Dans les codes qui précèdent, l'utilisation du caractère & est **fondamentale**: en effet, passer l'exception par référence permet:

- D'utiliser l'édition de lien dynamique et non pas statique
- D'ajuster les traitements à l'exception **réellement émise**
- D'avoir des traitements plus ou moins fins, suivant qu'on attrape l'exception plus ou moins haut dans la chaîne d'héritage

Il est indispensable que toutes les exceptions émises dérivent d'un même objet

Les exceptions non capturées ne sont **pas traitées** et remontent toute la chaîne d'appels, éventuellement jusqu'au système d'exploitation (arrêt du programme).

traitement de plusieurs types d'exceptions

Le programme suivant traite toutes les exceptions possibles:

```
try {  
    blabla  
}  
catch (const & domain_error e) {  
    blabla  
}  
catch (const & bad_alloc e) {  
    blabla  
}  
catch (...) {  
    cout << "Autre exception\n";  
};  
}
```

Exceptions et constructeurs

OUIIIIIIIIIIIIIIIIIIIIIIII !

NOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOON !

(à suivre)