



PОО – Langage C++

Les exceptions

1^{ère} année ingénieur informatique

Mme Wiem Yaiche Elleuch

2018 - 2019

plan

- Définition d'une exception: throw..try..catch
- Cheminement/propagation des exceptions
- Redéclenchement/relance d'une exception
- Spécification d'interface
- Exceptions standards

Définition d'une exception

- **Exception:** est une **erreur, anomalie, incident** au cours de **l'exécution d'un programme** (exemples: échec allocation, erreur d'indice, disque plein, problème d'ouverture de fichier, etc)
- **Gestion des exceptions:** mécanisme très puissant de traitement de ces anomalies, qui a été intégré au langage C++
- **But: Séparer, dissocier, découpler :**
 - la **détection** de l'erreur dans un programme
 - Le **traitement** de l'erreur
- La détection d'un incident et son traitement dans les programmes importants doivent se faire dans des parties différentes du code.
- **Résultat:** développer des composants réutilisables destinés à être exploités par de nombreux programmes.

Les exceptions

- une **exception** est une **rupture de séquence déclenchée** (levée, lancée) par une instruction **throw**, comportant une expression d'un **type** donné.
- Il y a alors **branchement** à un ensemble d'instructions nommé **gestionnaire d'exception** (choisi selon le **type** de l'exception).
- Le modèle de gestion des exceptions proposé par C++ ne permet pas de reprendre l'exécution à partir de l'instruction ayant levé l'exception, mais le permet après le bloc catch correspondant
- ➔ **chaque exception est caractérisée par un type**, et le choix du bon gestionnaire se fait en fonction du type de l'expression mentionnée à **throw**.

Structure de contrôle try ... catch

- Pour qu'une partie de programme puisse **intercepter** une exception, il est nécessaire qu'elle figure à l'intérieur d'un bloc précédé du mot clé **try**.
- 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 en existe.
- Un bloc try doit être suivi d'un ou de plusieurs blocs **catch** représentant les différents gestionnaires correspondants.
- Si plusieurs blocs catch existent, ils doivent intercepter un type d'exception différent.
- Un gestionnaire d'exception peut contenir des instructions **exit** ou **abort** qui mettent fin à l'exécution du programme
- À la fin du bloc catch, le programme continue sur l'instruction qui suit le dernier catch

jargon

- Une exception a un type (de préférence, utiliser type classe)
- Une exception est **déclenchée / lancée / levée** par throw
- Le bloc try contient l'instruction qui peut générer l'exception.
- Une exception est **attrapée/capturée/interceptée** par un bloc catch

Exception déclenchée dans main

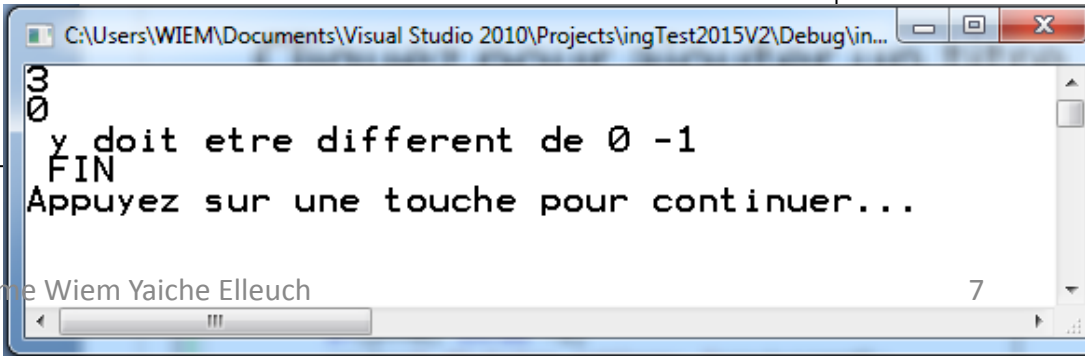
throw ... try ... catch

```
void main()
{
    int x,y;
    cin>>x>>y;
    try
    {
        if(y==0) throw -1;
        cout<<" division entiere: "<<x/y<<endl;
    }
    catch (int e)
    {
        cout<<" y doit etre different de 0 "<<e<<endl;
    }
    cout<<" FIN "<<endl;
    system("PAUSE");
}
```

Exception de type int, →
catch est de type int

Bloc try

Gestionnaire
d'exception



```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\ingTest2015V2\Debug\in...
3
0
y doit etre different de 0 -1
FIN
Appuyez sur une touche pour continuer...
```

déroulement

- À l'exécution, le programme se déroule normalement comme si les instructions try et les blocs catch n'étaient pas là.
- Par contre, au moment où l'ordinateur arrive sur une instruction throw, il saute toutes les instructions suivantes et appelle le destructeur de tous les objets déclarés à l'intérieur du bloc try. Il cherche le bloc catch correspondant à l'objet lancé.
- Arrivé au bloc catch, il exécute ce qui se trouve dans le bloc et reprend l'exécution du programme *après* le bloc catch.


```

void main()
{
    int x,y;
    string msg="ERREUR";
    cin>>x>>y;
    try
    {
        if(y==0) throw msg;
        cout<<" division entiere: "<<x/y<<endl;
    }
    catch (string m)
    {
        cout<<" y doit etre different de 0 "<<m<<endl;
    }

    cout<<" FIN "<<endl;
    system("PAUSE");
}

```

Exception de type string →
catch est de type string

```

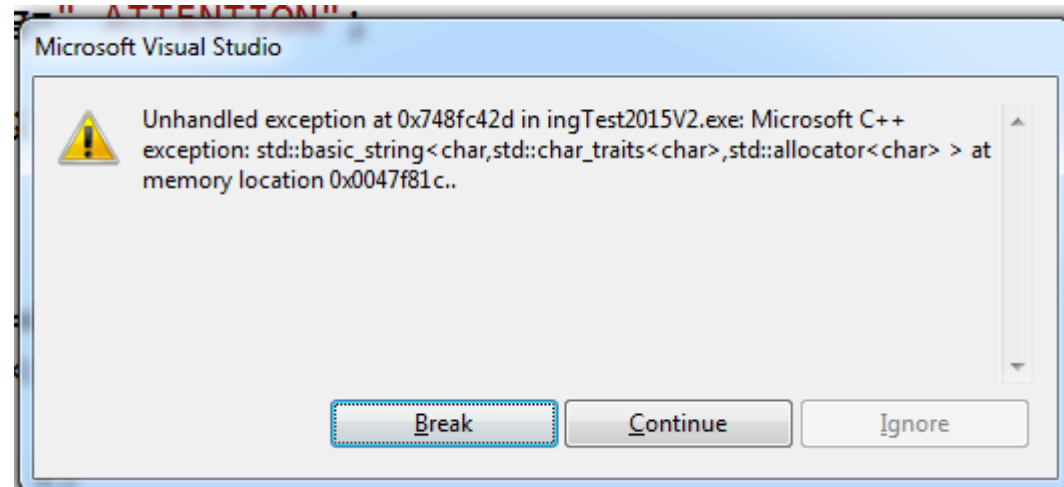
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\ingTest2015V2\Debu...
3
0
y doit etre different de 0 ERREUR
FIN
Appuyez sur une touche pour continuer...

```

Les fonctions **terminate** et **abort**

```
void main()
{
    string msg=" ATTENTION";
    int x, y;
    cin>>x>>y;

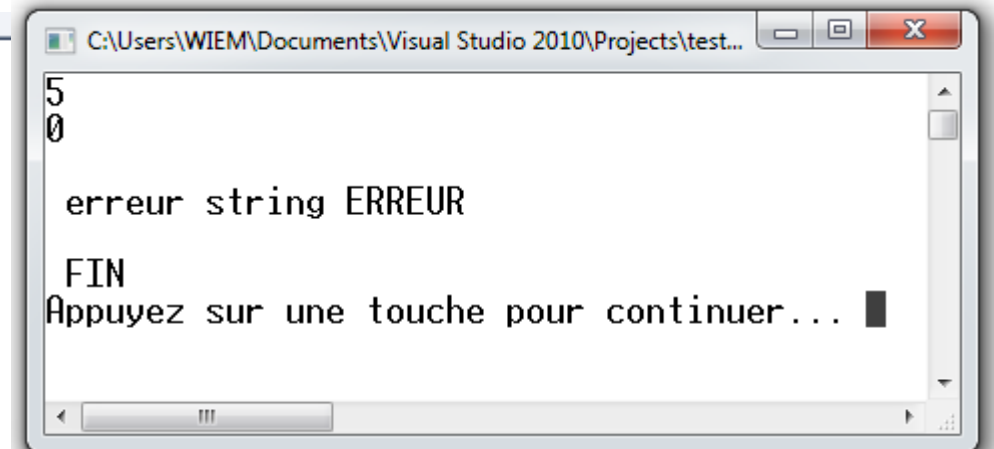
    try
    {
        if(y==0) throw msg;
        cout<<x/y<<endl;
    }
    catch(int i)
    {
        cout<<"\n ERRREUR "<<i<<endl;
    }
    cout<<"\n FIN "<<endl;
    system("PAUSE");
}
```



- **Remarque**: dans cet exemple, l'exception est de type string, et aucun catch de type string n'a été défini → une fonction spéciale **terminate**, qui par défaut appelle la fonction **abort**, est appelée pour terminer l'exécution du programme.

exemple

```
(Global Scope)
void main()
{
    int x,y;
    string msg="ERREUR";
    float res=0.0;
    try
    {
        cin>>x>>y;
        if(y==0) throw msg;
        if(y==1) throw -1;
        res=(float)x/y*(y-1);
        cout<<"\n la division est "<<res<<endl;
    }
    catch (int i)
    {
        cout<<"\n erreur int "<<i<<endl;
    }
    catch (string m)
    {
        cout<<"\n erreur string "<<m<<endl;
    }
    cout<<"\n FIN "<<endl;
}
```



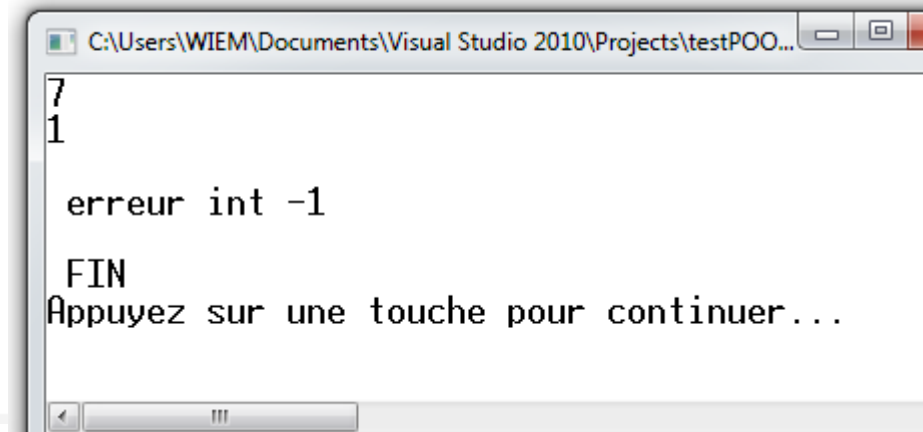
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test...

5
0

erreur string ERREUR

FIN

Appuyez sur une touche pour continuer...



C:\Users\WIEM\Documents\Visual Studio 2010\Projects\testPOO...

7
1

erreur int -1

FIN

Appuyez sur une touche pour continuer...

Quelques schémas: exception lancée dans main

```
void main()
{
    try
    {
        throw (...);
    }
    catch(...)
    { ...
    }
    system("PAUSE");
}
```

```
void main()
{
    try
    {
        throw (...);
    }
    catch(...)
    { ...
    }
    try
    {
        throw (...);
    }
    catch(...)
    { ...
    }
    system("PAUSE");
}
```

```
void main()
{
    try
    {
        ... throw (...);
        ... throw (...);
    }
    catch(type1..)
    { ...
    }
    catch(type2..)
    { ...
    }
    system("PAUSE");
}
```

Quelques schémas: exception lancée par une fonction appelée

```
void fct(..)
{
    throw ( );
}

void main()
{
    try
    {
        fct ( .. );
    }
    catch(...)
    { ...
    }
    system("PAUSE");
}
```

```
(Global Scope)
void fct(..)
{
    try
    { ...
        throw ( );
    }
    catch(..)
    { ...
    }
}

void main()
{
    try
    {
        fct ( .. );
    }
    catch(...)
    { ...
    }
    system("PAUSE");
}
```

100 %

D'une manière générale

```
try
{
    ....// instructions susceptibles de lever une exception
        // soit directement par throw(expression)
        // soit par les fonctions appelées
}
catch( type1 ..)
{
    ..// traitement de l'exception de type1
}
catch( type2 ..)
{
    ..// traitement de l'exception de type2
}
catch( typeN ..)
{
    ..// traitement de l'exception de typeN
}
```

Exemple: Exception déclenchée par la fonction division appelée par main

```
(Global Scope)
int division (int a, int b)
{
    if(b==0) throw -1;
    return a/b;
}
void main()
{
    int x,y, res=0;
    cin>>x>>y;
    try
    {   res=division(x,y);
        cout<<" division entiere: "<<res<<endl;
    }
    catch (int e)
    {   cout<<" y doit etre different de 0 "<<e<<endl;
    }

    cout<<" FIN "<<endl;
    system("PAUSE");
}
```

100 %

plan

- Définition d'une exception: throw..try..catch
- Cheminement/propagation des exceptions
- Redéclenchement/reliance d'une exception
- Spécification d'interface
- Exceptions standards

Cheminement des exceptions

- Lorsqu'une exception est levée par une fonction, on cherche tout d'abord un gestionnaire dans l'éventuel bloc try associé à cette fonction
- S'il n'existe pas, (ou si aucun bloc try n'est associé), on poursuit la recherche dans un éventuel bloc try associé à une fonction appelante et ainsi de suite.
- Si aucun gestionnaire d'exception n'est trouvé, on appelle la fonction **terminate**, qui par défaut appelle **abort**

```
test.cpp* X
(Global Scope)

int division (int a, int b)
{
    try
    {   if(b==0) throw -1;
        return a/b;
    }
    catch(string m)
    {   cout<<" exception dans division "<<m<<endl;
    }
}

void main()
{
    int x,y, res=0;
    cin>>x>>y;
    try
    {   res=division(x,y);
        cout<<" division entiere: "<<res<<endl;
    }
    catch (int e)
    {   cout<<" exception dans main "<<e<<endl;
    }
    cout<<" FIN "<<endl;
    system("PAUSE");
}
```

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\ingTest2015V2\...
3
0
exception dans main -1
FIN
Appuyez sur une touche pour continuer...
```

Dans cet exemple, l'exception déclenchée est de type int, elle est attrapée par le bloc catch de main, car aucun bloc catch de la fonction division n'est de type int.

plan

- Définition d'une exception: throw..try..catch
- Cheminement/propagation des exceptions
- Redéclenchement/reliance d'une exception
- Spécification d'interface
- Exceptions standards

Redéclenchement d'exception avec **throw;**

- Dans un gestionnaire, l'instruction *throw* (sans expression) retransmet l'exception au niveau englobant.
- Cette possibilité permet par exemple de compléter un traitement standard d'une exception par un traitement complémentaire spécifique.
- voici un exemple dans lequel une exception (ici de type *int*) est tout d'abord traitée dans la fonction *division*, avant d'être traitée dans *main*:

```
myExceptionDerivee.h  myException.h  test.cpp* X
(Global Scope)
int division (int a, int b)
{
    try
    {
        if(b==0) throw -1;
        return a/b;
    }
    catch (int i)
    {
        cout<<"\n exception dans division "<<endl;
        throw;
    }
}

void main()
{
    int x,y;
    cin>>x>>y;
    try
    {
        cout<<" division entiere: "<<division(x,y)<<endl;
    }
    catch (int i)
    {
        cout<<" exception dans main "<<endl;
    }
    cout<<" FIN "<<endl;
    system("PAUSE");
}
```

```
C:\Users\WIEM\Documents\Visual Studio 2010\Project...
3
0
division entiere:
exception dans division
exception dans main
FIN
Appuyez sur une touche pour contin
```

plan

- Définition d'une exception: throw..try..catch
- Cheminement/propagation des exceptions
- Redéclenchement/reliance d'une exception
- **Spécification d'interface**
- Exceptions standards

Spécification d'interface: la fonction **unexpected**

- Une fonction peut spécifier les exceptions qu'elle est susceptible de déclencher/provoquer (elle-même, ou dans les fonctions qu'elle appelle à son tour).
- Elle le fait à l'aide du mot clé *throw*, suivi, entre parenthèses, de la liste des exceptions concernées.
- Dans ce cas, toute exception non prévue et déclenchée à l'intérieur de la fonction (ou d'une fonction appelée) entraîne l'appel d'une fonction particulière nommée **unexpected**.

```
void division (...) throw (int,string) {      }  
// la fonction division ne peut déclencher que des  
exceptions de type int et string
```

Spécification d'exceptions

1. **void f() throw (string):** seule l'exception du type string peut être lancée dans la fonction f.
2. **void g() throw (int, double):** uniquement les exceptions du type int et double peuvent être lancées dans la fonction g.
3. **int f() throw():** aucune exception ne peut être lancée dans la fonction f.
4. **double f():** toutes les exceptions peuvent être levées dans la fonction f, c'est l'approche par défaut.

Spécification d'exceptions

- Si une fonction lance une exception non déclarée dans son entête, une fonction unexpected est appelée.
- Cette fonction appellera une fonction terminate.
- La fonction terminate aura pour tâche d'appeler la fonction abort pour arrêter l'exécution du programme.

plan

- Définition d'une exception: throw..try..catch
- Cheminement/propagation des exceptions
- Redéclenchement/reliance d'une exception
- Spécification d'interface
- Exceptions standards

Les exceptions standards

- La bibliothèque standard comporte quelques classes fournissant des exceptions spécifiques susceptibles d'être déclenchées par un programme.
- Toutes ces classes dérivent d'une classe de base nommée **exception** et sont organisées suivant la hiérarchie suivante:

```
exception
  logic - error
    domain error
    invalid _argument
    length - error
    out_of_range
  runtime_error
    range - error
    overflow error
    underflow - error
  bad_alloc
  bad_cast
  bad_exception
  bad _typeid
```

Les exceptions déclenchées par la bibliothèque standard

- **bad_alloc**: échec d'allocation mémoire par `new`;
- **bad_cast** : échec de l'opérateur *dynamic_cast*;
- **bad_typeid** : échec de la fonction *typeid* ;
- **bad_exception**: erreur de spécification d'exception; cette exception peut être déclenchée dans certaines implémentations par la fonction *unexpected* ;
- **out_of_range**: erreur d'indice; cette exception est déclenchée par les fonctions *at*, membres des différentes classes conteneurs ainsi que par l'opérateur `[]` du conteneur *bitset*;
- **invalid_argument**: déclenchée par le constructeur du conteneur *bitset*;
- **overflow_error** : déclenchée par la fonction *to_ulong* du conteneur *bitset*.

Les exceptions utilisables dans un programme

- Toutes les classes précédentes sont utilisables pour les exceptions déclenchées par l'utilisateur, soit telles quelles, soit sous forme de classes dérivées.
- Il est cependant préférable d'assurer une certaine cohérence à son programme; par exemple, il ne serait pas raisonnable de déclencher une exception *bad_alloc* pour signaler une anomalie sans rapport avec une allocation mémoire.

Les exceptions standards

- la classe de base *exception* dispose d'une fonction membre *what* censée fournir comme valeur de retour un pointeur sur une chaîne expliquant la nature de l'exception.
- Cette fonction, virtuelle dans *exception*, doit être redéfinie dans les classes dérivées et elle l'est dans toutes les classes citées ci-dessus
- toutes ces classes disposent d'un constructeur recevant un argument de type chaîne dont la valeur pourra ensuite être récupérée par *what*.

Création d'exceptions dérivées de la classe *exception*

Il est recommandé de créer des classes dérivées de la classe *exception*, pour au moins deux raisons:

1. on est sûr d'intercepter **toutes les exceptions** avec le simple gestionnaire:

```
catch (exception & e) { ..... }
```

Ce ne serait pas le cas pour des exceptions non rattachées à la classe *exception*.

1. On peut s'appuyer sur la fonction **what**, à condition de la redéfinir dans les classes dérivées. Il est alors facile d'afficher un message explicatif concernant l'exception détectée, à l'aide du simple gestionnaire suivant:

```
catch (exception & e) // attention à la référence, pour bénéficier  
// de la ligature dynamique de la fonction virtuelle what  
{ cout << "exception interceptée: " << e.what() << "\n" ;  
}
```

```

class myException: public exception
{
    char* msg;
public:
    myException(char*m){msg=m;}
    const char*what() const{ return msg;}
};

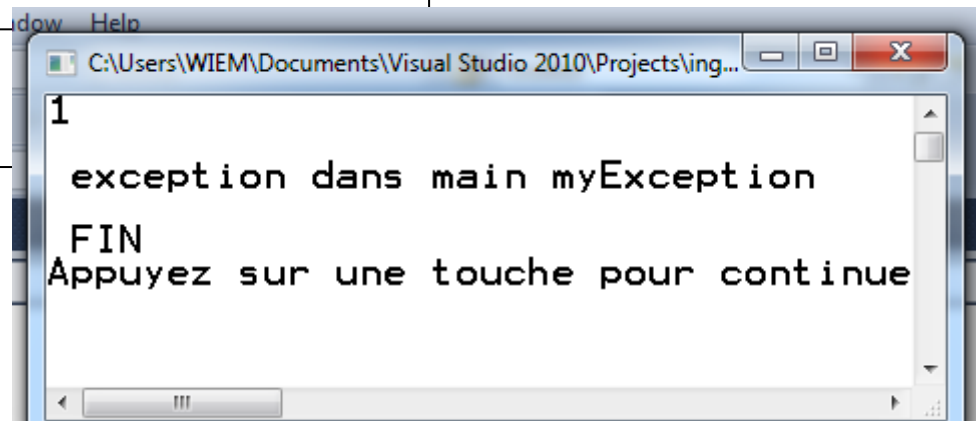
```

```

void main()
{
    int x;
    cin>>x;
    try
    {
        if(x==1) throw myException("myException ");
        if(x==2) throw exception("exception ");
    }
    catch(exception &e)
    {
        cout<<"\n exception dans main "<<e.what()<<endl;
    }

    cout<<"\n FIN "<<endl;
    system("PAUSE");
}

```



choix du gestionnaire d'exception: l'examen des blocs catch se fait dans l'ordre

```
int division (int a, int b)
{
    myException ex;
    if(b==0) throw ex;
    return a/b;
}

void main()
{
    int x,y, res=0;
    cin>>x>>y;
    try
    {
        res=division(x,y);
        cout<<" division entiere: "<<res<<endl;
    }
    catch (myException&e)
    {
        cout<<" myException "<<endl;
    }
    catch (exception&e)
    {
        cout<<" exception "<<endl;
    }

    cout<<" FIN "<<endl;
    system("PAUSE");
}
```

```
3
0
myException
FIN
Appuyez sur une touche pour
```

(Global Scope)

```
int division (int a, int b)
{
    myException ex;
    if(b==0) throw ex;
    return a/b;
}

void main()
{
    int x,y, res=0;
    cin>>x>>y;
    try
    {
        res=division(x,y);
        cout<<" division entiere: "<<res<<endl;
    }
    catch (exception&e)
    {
        cout<<" exception "<<endl;
    }
    catch (myException&e)
    {
        cout<<" myException "<<endl;
    }

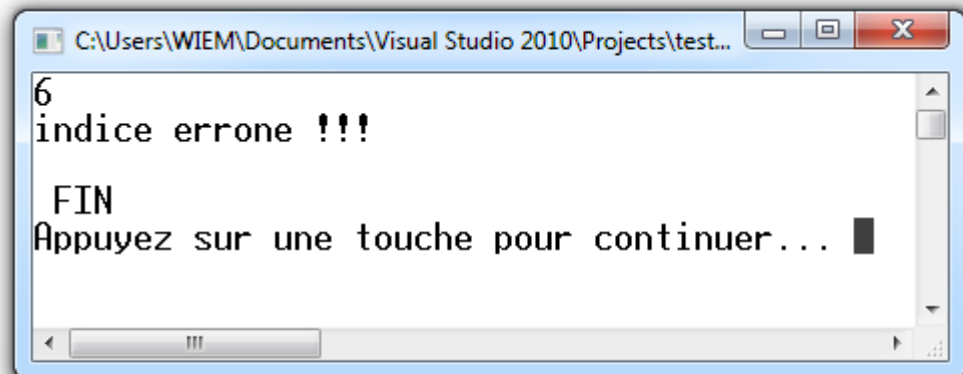
    cout<<" FIN "<<endl;
    svstem("PAUSE");
}
```

```
3
0
exception
FIN
Appuyez sur une touche pour conti
```

Exemple classe out_of_range

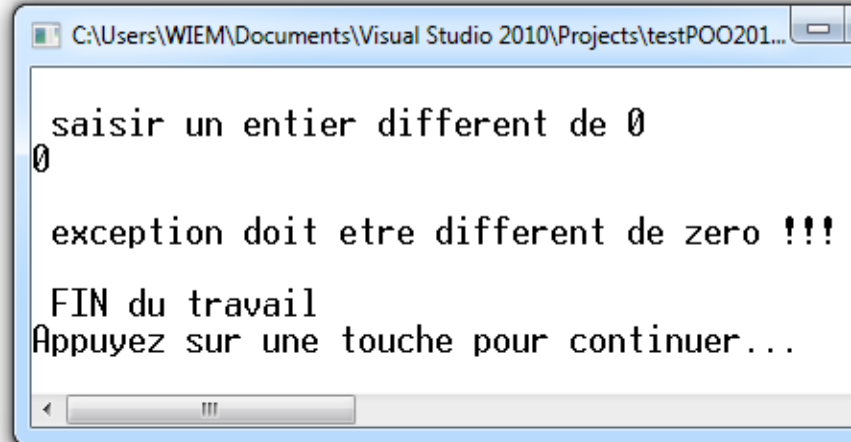
```
int fct(int *t)
{
    int i;
    cin>>i;
    if(i<0 || i>5) throw out_of_range("indice errone !!!");
    return t[i];
}

void main()
{
    int tab[5]={11,22,33,44,55};
    try
    {
        cout<<fct(tab)<<endl;
    }
    catch(exception& e)
    {
        cout<<e.what()<<endl;
    }
    cout<<"\n FIN "<<endl;
    system("PAUSE");
}
```



Exemple simple 1

```
void main()
{
    exception ex("doit etre different de zero !!! ");
    int x;
    try
    {
        cout<<"\n saisir un entier different de 0 "<<endl;
        cin>>x;
        if(x==0) throw ex;
    }
    catch(exception& e)
    {
        cout<<"\n exception "<<e.what()<<endl;
    }
    cout<<"\n FIN du travail "<<endl;
    system("PAUSE");
}
```



C:\Users\WIEM\Documents\Visual Studio 2010\Projects\testPOO201...

saisir un entier different de 0
0
exception doit etre different de zero !!!
FIN du travail
Appuyez sur une touche pour continuer...

Name	Value	Type	Call Stack
------	-------	------	------------

Exemple simple 2

```
void main()
{
    exception ex;
    int x;
    try
    {
        cout<<"\n saisir un entier different de 0 "<<endl;
        cin>>x;
        if(x==0) throw ex;
    }
    catch(exception& e)
    {
        cout<<"\n exception !!!"<<endl;
    }
    cout<<"\n FIN du travail "<<endl;
    system("PAUSE");
}
```

