Exceptions

définitions

C++ offre des outils de gestion des erreurs appelées exceptions. Par exemple dans la classe *Chaine* définie en cours, le fait qu'on ne puisse pas acquérir de mémoire pour représenter la chaîne, ou qu'on essaie d'accéder à un caractère de la chaîne qui n'existe par sont des erreurs à traiter par les exceptions.

L'auteur d'une classe peut détecter des erreurs, mais ne sait pas comment les traiter, en revanche, l'utilisateur saurait traiter ces erreurs mais ne sait comment les détecter…On pourrait imaginer différents mécanismes de traitement d'erreurs :

- Terminer le programme … c'est un peu radical.
- Retourner un code d'erreur : ce n'est pas toujours possible…(constructeurs ou pas de possibilité de trouver une valeur codant l'erreur)
- Appeler une fonction de traitement d'erreurs…

Le mécanisme d'exception est là pour uniformiser ces traitements : une fonction qui rencontre une erreur lève (lance) une exception, en espérant que son appelant (direct ou non) saura traiter cette erreur. Une fonction qui voudra traiter cette exception indiquera qu'elle est disposée à intercepter cette exception .

En C++ une exception est un objet … ce peut être un entier ou un char * mais en général on a intérêt à définir une classe *Ex* qui traite les comportements exceptionnels de la classe *x.*

La gestion des exceptions s'articule autour de 4 concepts :

- La définition de la classe qui permet la gestion des exceptions.
- Le lancement d'une exception : une erreur a été détectée on lance une exception *throw exception*
- La mise sous surveillance d'un bloc d'instructions susceptible de générer des exceptions : *try { instructions à surveiller }*
- Le traitement de l'exception *: catch(l'exception à traiter) { le traitement }*

try, catch, throw

Traitons en exemple la définition de l'opérateur [] de la classe *chaîne* : le concepteur de la classe traite le cas où l'indice est <0 ou > à la longueur de la chaîne en levant une exception de la classe *Echaine*

```
class Echaine
{
    . . .
}
```

Nous pouvons définir l'opérateur [] de la classe *chaine* de la façon suivante :

```
char & chaine::operator [] (int i){
    if(i<0 || i>=strlen(rep)) throw Echaine() ;
    return rep[i]
}
```

Un utilisateur de la classe utilisera alors l'opérateur [] de la façon suivante :

```
chaine s ;
. . .
try{
     . . . s[i]. . .
}catch(Echaine e){
    . . . traitement de l'exception. . .
}
```

De la même façon on peut écrire pour une fonction qui réserve de la mémoire :

```
chaine::chaine (const & chaine ch){
    try{
        rep=new char [strlen(ch.rep)+1] ;
    } catch( bad_alloc){ throw Echaine() ;}
    strcpy(rep, ch.rep) ;
}
```

L'utilisateur écrivant alors :

```

try{
   chaine s ;
    . . .
}catch(Echaine e){
    . . . traitement de l'exception. . .
}
```

Pour distinguer les différentes sortes d'exceptions levées, on peut choisir des types exceptions différents ou choisir un même type et coder par un message les différentes exceptions :

```
class Echaine
{
    private :
       string mess ;
    public :
       Echaine(string m) : mess(m){}
       string Message(){return mess;}
};
```

```
chaine:: chaine (const chaine & s)
{
   try {
      tete = new char [s.longueur+1];
   }catch (bad_alloc)
   {   Echaine e("création : manque de mémoire");
       throw e ;
       // ou throw Echaine ("création : manque de mémoire");
   }
   strcpy(tete, s.tete);
}
```

Un exemple simple : la classe *E* est une classe exception qui stocke un numéro de message accessible par la fonction *numero()*. La fonction *f1* ne lance qu'une exception de type *E* alors que la fonction *f2* lance de façon aléatoire des exceptions de type *E* ou de type *int* ou de type *char **.

Le . h

```
class E
{
    int num;
  public :
    E( int n) : num(n){};
    int numero(){ return num;}
};
void f1();
void f2()throw(E, int, char *);
```

Le .cpp

```
void f1()
 {
     throw E(1);
 }

 char * message = "bonjour";
 void f2()throw(E, int, char *)
 {  int i =random(3);
    switch(i)
    { case 0 : throw E(2) ;
      case 1 : throw int(10);
      case 2 : throw message;
    }
}
```

Le programme surveille les appels de fonctions *f1* et *f2* et traite les différentes exceptions lancées par les fonctions :

```
int main(int argc, char **argv)
{
    randomize();


    try{
       f1();
    }catch( E e)
    {
      cout<<"exception : "<<e.numero();
    }
    try{
       f2();
    }
    catch( E e)      { cout<<"exception : "<<e.numero();}
    catch( int k)    { cout<<"exception : "<<k;}
    catch( char * m) { cout<<m;}

    getch();
```

```
    return 0;
}
```

Pour intercepter toutes les exceptions on écrira **`catch (…) {  }`**

throw

Throw se construit de la façon suivante :

- **throw objet** ; lancement d'une exception de type objet
- **throw**; L'exception en train d'être traitée doit être relancée.
- **void f() throw (A, B)** Indique la liste d'exceptions que *f* peut lancer. Aucune autre exception ne peut être propagée depuis *f*. Si une exception autre que *A* et *B* est générée dans *f*, elle sera considérée comme une exception non attendue et le contrôle du programme sera remis à la fonction *unexpected*.
- **void f2() throw ()**Indique que *f2* ne lance aucune exception. Si une autre fonction (par exemple, operator new) dans le corps de *f2* lance une exception, celle-ci devra être interceptée et gérée dans le corps même de *f2*. Sinon, cette exception constitue une violation de la spécification d'exceptions de *f2*. La fonction *unexpected* sera appelée.

Le mécanisme de lancement d'exception consiste à dépiler tous les environnements des différents appels de fonctions, jusqu'à une fonction qui intercepte l'exception. ce dépilement s'effectue en appelant les différents destructeurs de différentes variable locales de la pile : il faut donc prendre soin qu'une exception ne fasse pas référence à une variable locale … qui est donc détruite lorsqu'on traite l'exception.

Exemple :

```
void f4()throw(char *)
 {
    char x[20];
    strcpy(x,"ah ! ah ! ");
    throw x;
 }
```

```
try{ f4();}
catch( char * m){ cout<<m;}
```

Dans ce cas *m* n'as plus de sens … il a été détruit quand l'appel de *f4* a généré une exception.

terminate, unexpected

Si une exception est lancée et que le déroulement de la pile ne permet pas de l'intercepter, la fonction *terminate()* est appelée. (*terminate* se trouve dans *except.h*).On peut

modifier la fonction *terminate* appelée en utilisant *set_terminate* .

Si une exception de type *E* est lancée par une fonction qui n'a pas *E* dans la liste des exceptions qu'elle est susceptible de générer, alors la fonction *unexpected()* est appelée. On peut modifier la fonction *unexpected* en faisant un appel de la fonction *set_unexpected()* ;

Exemple :

```
void fin()
{
   cout<<" c'est fini !";
} ;
void inattendu()
{
   cout<<" c'est inattendu !";
} ;
```

```
set_terminate(fin);
set_unexpected(inattendu);
```

Dans la fonction *OnException*, on peut utiliser la *fonction Application->ShowException(E)* ; qui affiche un message concernant l'exception.

Héritage d'exceptions

Dans l'exemple suivant :

```
class E
{
    int num;
  public :
    E( int n) : num(n){};
    int numero(){ return num;}
};

class ED : public E
{
   public :
    ED( int n) : E(n){};
};


void f3()throw(E, ED);
```

La partie *catch* intercepte les exception *E* et les exceptions *ED* :

```
try{
   f3();
}
catch( E e){ cout<<"exception E: "<<e.numero();}
```

Si on veut séparer le traitement des exceptions *E* et *ED*, il faut écrire l'interception de l'exception *ED* avant le traitement de l'exception *E*.

```
try{
  f3();
}
catch( ED e){ cout<<"exception ED : "<<e.numero();}
catch( E e) { cout<<"exception E : "<<e.numero();}
```

Exceptions

Le fichier *except.h* contient les exceptions suivantes :

bad_alloc exception lancée par l'opérateur *new*, lorsque une allocation mémoire est impossible.
bad-typeid exception lancée lors de mauvaise utilisation de l'opérateur typeid
bad-cast exception lancée lors de mauvaise utilisation de l'opérateur *dynamic_cast*

Les Exceptions et le débogage

Pendant une exécution dans l'environnement C++Builder, le programme s'arrête sur les lancements d'exception, on peut alors poursuivre l'exécution en appuyant sur le bouton exécution ou F7 ou F8.

Les Exceptions et la vcl

La propriété *Message* des Exceptions de VCL contient toujours un message décrivant le type de l'erreur.

1. Pour générer une exception de la VCL, dérivée de la classe Exception on doit écrire :

```
MonException * e ;
E = new MonException(" bla bla ") ;
```

2. Pour intercepter une exception de la VCL, on doit écrire :

```
catch( MonException & e) …
```