Chapitre 11 : Gestion des exceptions

De nombreux problèmes peuvent survenir pendant l'exécution d'un programme, l'insuffisance de mémoire, la perte d'un fichier, la saisie non valide d'une valeur sont des exemples d'erreurs. Le rôle d'un programme consiste à prévoir ces erreurs, à en informer les utilisateurs et éventuellement à mettre en œuvre des solutions de reprise et de correction de ces erreurs d'exécution.

C++ propose, à part l'utilisation de la valeur de retour des fonctions, une nouvelle solution pour intercepter les erreurs : le mécanisme des exceptions.

1. Gestion des erreurs en utilisant les valeurs de retour des fonctions

```
int  positive(int v)
{    if(v<0) return 0;
     cout<<«Valeur positive \n»;
     return 1 ;
}
//--------------------------
int  inf(int v,int max)
{    if(v>max) return 0;
     cout<<"Valeur inférieure à : "<<max<<endl;
     return 1;
}
//-------------------------------------------------------------
int  sup(int v,int min)
{    if(v<min) return 0;
     cout<<"Valeur supérieure à : "<<min<<endl;
     return 1;
}
//-------------------------------------------------------------
int main()
{
     int minimum=10;
     int maximum=100;
     int n,retour;
     cout<<"Saisir une valeur entière : "; cin>>n;
     retour=positive(n);
     if(retour==1)
     {    retour=inf(n,maximum);
          if(retour==1)
          {    retour=sup(n,minimum);
               if(retour==1)
               {    cout<<"Valeur correcte !";
                    return 1;
               }

               else
               {
```

```
                    cout<<"Valeur inférieure à : "<<minimum;
                    return 0;
                }
            }
            else
            {
                cout<<"Valeur supérieure à : "<<maximum<<endl;
                return 0;
            }
        }
        else
        {
            cout<<"Valeur négative !\n";
            return 0;
        }
}
```

Remarque

- Dans ce programme le contrôle d'erreurs est alourdie par les 'if' imbriqués.
- Un constructeur ne peut pas renseigner sur sa valeur de retour, qui n'en possède pas, pour indiquer qu'une erreur s'est produite.

2. Mise en œuvre des exceptions

Pour mettre en œuvre des exceptions sans se baser sur les valeurs de retour des fonctions, on doit :

- Définir une classe d'exception.
- Lancer l'exception (**throw**).
- Intercepter l'exception.

2.1. Définir une classe d'exception

Une classe d'exception correspond à une classe C++ qui peut fournir des informations sur une erreur.

Exemple

```
class erreur
{
      ……….
};
```

Remarque

On peut ajouter à la classe 'erreur' autant de données et de fonctions membres.

2.2. Lancer l'exception

Toute fonction qui souhaite lancer une exception doit utiliser un nouvel opérateur du langage C++ **throw**, suivi par un objet créé à partir d'une classe d'exception, cet opérateur permet de quitter la fonction qui l'utilise et d'informer la fonction appelante qu'une exception à été générée.

Exemple

```
void positive(int v)
{
      if(v<0)
      {
            erreur er;
            throw er;
      }
      cout<<"Valeur positive \n";
}
```

Remarque

Si 'v' est < à 0 la fonction construit un objet **statique** de la classe d'exception 'erreur' et envoie cet objet avec l'opérateur **throw** qui ressemble à **return**.

La fonction précédente peut être simplifiée :

```
void positive(int v)
{
      if(v<0) throw erreur();
      cout<<"Valeur positive \n ";
}
```

Commentaire

Il y a création d'un objet de type 'erreur' sans nom, cette solution n'est intéressante que si on n'a pas besoin d'appeler des fonctions membres de la classe d'exception.

Remarque

Toute fonction susceptible d'envoyer un ou plusieurs exceptions peut l'indiquer dans sa déclaration juste après la liste de ces arguments.

Exemple

```
void contrôle(int v)  throw (erreur,invalid)
{
      ……
      ……
}
```

2.3. Intercepter l'exception

Pour intercepter une exception, C++ fournit une syntaxe basée sur l'utilisation des blocs **try** et d'un ou plusieurs blocs **catch** :

```
try
{
     //Appels de fonctions pouvant générer des erreurs
}
catch (classe d'exception)
{
     //Ce bloc s'exécute pour une exception  de type
     //'classe d'exception'
}
catch(…)
{
     //Ce bloc s'exécute pour toutes les exceptions
     //en dehors de la classe d'exception
}
```

Description

Si une exception est envoyée par une de ces fonctions appelées dans le bloc 'try', le mécanisme d'exception entraînera les étapes suivantes :

- Tous les objets créés dans le bloc 'try' sont détruits'
- Le programme sort du bloc 'try' après la fonction qui a entraîné l'exception et n'exécute pas les instructions situées après cette fonction.
- Le C++ exécute dans l'ordre soit le bloc 'catch' correspondant à l'exception interceptée si elle existe, soit le bloc 'catch(…)' si aucune de ces conditions n'est remplie, cela entraîne la fin de l'exécution du programme, si un des blocs 'catch' a été utilisé, le programme continu à exécuter les instructions situées après ce bloc s'il y en a.

Remarque

Dans un bloc 'catch' on peut par exemple :

- arrêter le programme avec ou sans message utilisateur.
- corriger le problème avec ou sans message utilisateur
- uniquement informer l'utilisateur.

Exemple

```
1.
class     erreur { };
2.
void positive(int v)
{     if(v<0) throw erreur();
      cout<<"Valeur positive \n";
}
void inf(int v,int max)
{     if(v>max) throw erreur();
      cout<<"Valeur inférieure à : "<<max<<endl;
}
void sup(int v,int min)
{     if(v<min) throw erreur();
      cout<<"Valeur supérieure à : "<<min<<endl;
}
//-------------------------------------------------------
main()
{
      int minimum=10; int maximum=100;
      3. try
      {
            int n;
            cout<<"Saisir une valeur entière : "; cin>>n;
            positive(n);
            inf(n,maximum);
            sup(n,minimum);
            cout<<"Valeur correcte !\n";
      }
      catch(erreur er)
      {
            cout<<"valeur incorrecte! \n";
      }
      catch(...)
      {
            cout<<"Erreur inconnue !\n";
      }
}
```

Exercice

Toutes les fonctions de l'exemple précédent utilisent la même classe d'exception 'erreur'. Proposer plusieurs classes d'exception chacune responsable de l'affichage d'un message d'erreur.

Solution

```
class erreur_positive
{ };
class erreur_inf
{ };
class erreur_sup
{ };
void positive(int v)
{    if(v<0) throw erreur_positive();
     cout<<"Valeur positive \n";
}
void inf(int v,int max)
{    if(v>max) throw erreur_inf();
     cout<<"Valeur inférieure à : "<<max<<endl;
}
void sup(int v,int min)
{    if(v<min) throw erreur_sup();
     cout<<"Valeur supérieure à : "<<min<<endl;
}
main()
{
     int minimum=10; int maximum=100;
     try
     {
          int n;
          cout<<"Saisir une valeur entière : "; cin>>n;
          positive(n);
          inf(n,maximum);
          sup(n,minimum);
          cout<<"Valeur correcte !\n";
      }
      catch(erreur_positive  er)
      {
          cout<<"Erreur ! valeur negative ! \n";
      }
      catch(erreur_inf  er)
      {
       cout<<"Erreur ! valeur superieure A : "<<maximum<<endl;
      }
      catch(erreur_sup  er)
      {
       cout<<"Erreur ! valeur inferieure A : "<<minimum<<endl;
      }
      catch(...)
      {
          cout<<"Erreur inconnue !\n";
      }
}
```

Exécution

```
Saisir une valeur entiÞre : -7
Erreur ! valeur negative !
```

```
Saisir une valeur entiÞre : 230
Valeur positive
Erreur ! valeur superieure A : 100
```

```
Saisir une valeur entiÞre : 8
Valeur positive
Valeur infÚrieure ó : 100
Erreur ! valeur inferieure A : 10
```

```
Saisir une valeur entiÞre : 78
Valeur positive
Valeur infÚrieure ó : 100
Valeur supÚrieure ó : 10
Valeur correcte !
```

Prolongement

Afficher les messages d'erreurs à l'aide des fonctions membres des classes d'exception.

```
class erreur_positive
{
  public :
     void affiche(){cout<<"Erreur ! valeur negative ! \n";}
};
class erreur_inf
{
     int maxi;
  public:
     erreur_inf(int ma){maxi=ma;}
     void affiche()
     {cout<<"Erreur ! valeur superieure A : "<<maxi<<endl;}
};
class erreur_sup
{
      int mini;
    public:
     erreur_sup(int mi){mini=mi;}
     void affiche()
     {cout<<"Erreur ! valeur inferieure A : "<<mini<<endl;}
};
void positive(int v)
{
     if(v<0) { erreur_positive er; throw er;}
     cout<<"Valeur positive \n";
}
```

```
void inf(int v,int max)
{    if(v>max) { erreur_inf er(max); throw er;}
     cout<<"Valeur inférieure à : "<<max<<endl;
}
void sup(int v,int min)
{    if(v<min) { erreur_sup er(min); throw er;}
     cout<<"Valeur supérieure à : "<<min<<endl;
}
//-----------------------------------------------------------
main()
{
     int minimum=10; int maximum=100;
     try
     {
          int n;
          cout<<"Saisir une valeur entière : "; cin>>n;
          positive(n);
          inf(n,maximum);
          sup(n,minimum);
          cout<<"Valeur correcte !\n";
      }
      catch(erreur_positive  e)
      {
          e.affiche();
      }
      catch(erreur_inf  e)
      {
          e.affiche();
      }
      catch(erreur_sup  e)
      {
          e.affiche();
      }
      catch(...)
      {
          cout<<"Erreur inconnue !\n";
      }
}
```

Exécution

```
Saisir une valeur entiÞre : -23
Erreur ! valeur negative !
```

```
Saisir une valeur entiÞre : 567
Valeur positive
Erreur ! valeur superieure A : 100
```

```
Saisir une valeur entiÞre : 3
Valeur positive
Valeur infÚrieure ó : 100
Erreur ! valeur inferieure A : 10
```

```
Saisir une valeur entiÞre : 89
Valeur positive
Valeur infÚrieure ó : 100
Valeur supÚrieure ó : 10
Valeur correcte !
```

Remarque

Les exceptions sont symbolisées par des classes dans lesquelles on peut intégrer les notions d'héritage et de polymorphisme.

3. Hiérarchie des classes d'exception

Reprendre le programme précédent en tenant compte de l'héritage et du polymorphisme.

Programme

```
class erreur
{
  public:
    virtual void affiche(){cout<<"Erreur !";}
};
class erreur_positive:public erreur
{
  public :
    void affiche(){cout<<"Erreur ! valeur negative ! \n";}
};
class erreur_inf:public erreur
{
    int maxi;
  public:
    erreur_inf(int ma){maxi=ma;}
    void affiche()
    {cout<<"Erreur ! valeur superieure A : "<<maxi<<endl;}
};
```

```
class erreur_sup:public erreur
{
      int mini;
  public:
      erreur_sup(int mi){mini=mi;}
      void affiche()
      {cout<<"Erreur ! valeur inferieure A : "<<mini<<endl;}
};
//-------------------------------------------------------------
void positive(int v)
{
     if(v<0)
     { erreur_positive *er; er=new erreur_positive; throw er;}
     cout<<"Valeur positive \n";
}
void inf(int v,int max)
{
     if(v>max)
     { erreur_inf *er; er=new erreur_inf(max); throw er;}
     cout<<"Valeur inférieure à : "<<max<<endl;
}
void sup(int v,int min)
{
     if(v<min)
     { erreur_sup *er; er=new erreur_sup(min); throw er;}
     cout<<"Valeur supérieure à : "<<min<<endl;
}

//-------------------------------------------------------------
main()
{
        int minimum=10; int maximum=100;
        try
        {
          int n;
          cout<<"Saisir une valeur entière : "; cin>>n;
          positive(n);
          inf(n,maximum);
          sup(n,minimum);
          cout<<"Valeur correcte !\n";
        }
        catch(erreur *e)
        {
            e->affiche(); delete e ;
        }
        catch(...)
        {
            cout<<"Erreur inconnue !\n";
        }
}
```

Exécution

```
Saisir une valeur entiere : -35
Erreur ! valeur negative !
```

```
Saisir une valeur entiere : 300
Valeur positive
Erreur ! valeur superieure A : 100
```

```
Saisir une valeur entiere : 6
Valeur positive
Valeur inferieure a : 100
Erreur ! valeur inferieure A : 10
```

```
Saisir une valeur entiere : 67
Valeur positive
Valeur inferieure a : 100
Valeur superieure a : 10
Valeur correcte !
```

Références bibliographiques

- La bible du programmeur C/C++, Kris Jamsa et Lars Klander, Editions G. Reynald, 1999.
- Comment programmer en C++, Deitel et Deitel, Editions G. Reynald, 2001.
- Apprendre le C++, C. Delannoy, Editions Eyrolles, 2007.
- Exercices en langage C++, C. Delannoy, Editions Eyrolles, 2007.
- C++ pour les programmeurs C, C. Delannoy, Editions Eyrolles, 2007.