exceptions

Les exceptions :

Les exceptions sont un bon moyen de gérer les erreurs dans les programmes. La grande différence vis à vis du classique code d'erreur renvoyé par une fonction est qu'une exception se propage depuis l'appelé vers l'appelant jusqu'à ce qu'elle rencontre un bloc de code qui s'occupe de la traiter. Les exceptions peuvent aussi gérer les erreurs logiques dans un programme. Cela permet donc de facilement faire « remonter » les erreurs des fonctions appelées vers les fonctions appelantes. Dans cette partie nous allons découvrir comment on peut lever des exceptions en EZ et comment ceci se traduit en c++.

Classe exception		
Méthode	**Type de retour**	**Description**
exception(error is ***string*****,** remedy is ***string*****)**	void	Permet de lever une exception

Instructions	Description
throw	Permet de lever une exception.
try/catch	Permet de remonter l'exception.
stop	Permet d'arrêter le programme et affiche le message de l'exception.
continue	Continue le programme et donne la possibilité de corriger le problème rencontré.

Comment se traduit la classe exception en c++ :

La classe exception sera traduite de la manière suivante :

```
#include<sstream>
class EZException  {
private:
        std::ostringstream error_msg ; // cause de l'exception
public :

std::ostringstream getErrors(){
return error_msg ;
}
void fill(std::string msg){
     error_msg << msg;
}
}
```

Exemples :

```
program hello_exception

function div(a is integer, b is integer)
if b==0 then throw zero_exception, "entrer une valeur différente de 0 pour le diviseur"
else
  return a/b
end if
end function

procedure hello_exception()
    good_div is integer
    bad_div is integer
try
// ceci ne va pas lever d'exception
good_div = div(6, 2)
// ceci va lever une exception
bad_div = div(6,0)
catch stop
end try
print good_div
print bad_div
end
```

```
program file_exception

procedure file_with_exception
    // créer une instance de la classe file
    myfile is file("/path/to/file/myfile.txt")
    // ouvrir le fichier
    if myfile.open() then
        // vérifier qu'il n'est pas vide
        if !myfile.is_empty() then
            // itérer sur le nombre de lignes
            for i in 1..myfile.get_nbr_lines()
                // afficher ligne par ligne
                print "ligne ", i," : " ,myfile.get_line(i)
            end for
        else throw Empty_file_exception, "please write data"
        end if
    else throw Not_found_file_exception, "please make sure this file is created")
    end // fin du if
end // fin de procedure

procedure file_exception
```

```
try
    file_with_exception
catch stop
  print_exception_trace
end // fin du try
end
```

```
program file_exception

procedure file_exception
// créer une instance de la classe file
    myfile is file("/path/to/file/myfile.txt")
try
    // ouvrir le fichier
    if myfile.open() then
          // vérifier qu'il n'est pas vide
          if !myfile.is_empty() then
                // itérer sur le nombre de lignes
                for i in 1..myfile.get_nbr_lines()
                      // afficher ligne par ligne
                      print "ligne ", i," : " ,myfile.get_line(i)
                end for
          else throw Empty_file_exception, "please write data"
          end if
    else throw Not_found_file_exception, "please make sure this file is created")
    end // fin du if
catch  Not_found_file_exception
// création et écriture dans le fichier
     myfile.write("file created")
catch  Empty_file_exception
// écriture dans le fichier
     myfile.write("this file is no longer empty")
  end
end // fin du try
end
```

Remarques :

Quand l'utilisateur écrit « stop » après le catch le programme retourne le message de l'exception rencontrée.

Quand l'utilisateur écrit catch suivi du nom de l'exception, il a la possibilité de rajouter des traitements lié à cette dernière. Le identifiant de l'exception est **unique !** Ce qui implique que l'utilisateur n'a pas le droit de choisir deux identifiants d'exception identique.