Séance 7

Exception

Les exceptions ont été rajoutées à la norme du C++ afin de faciliter la mise en œuvre de code robuste.

Les exceptions

- Une exception indique une erreur pendant l'exécution

- Une *exception* est un événement, qui entraine une suspension définitive de l'élaboration d'une déclaration ou de l'exécution d'un ordre. A *la place*, un *preneur en mains d'exception* est exécuté.

- Une exception est elle-même suscitée soit fortuitement (en particulier à cause d'une erreur), soit par un ordre donné par le programmeur

- => Utiliser les exceptions vous donnera un code plus sûr

Exemple :

- Gestion de la saisie à partir du clavier ou lecture à partir d'un fichier ou écriture dans un fichier
- Erreur d'opération division par zéro ou racine carré d'u nombre négatif dans l'espace des réels

Exemple sans exception

```
1  int division(int a,int b) // Calcule a divisé par b.
2  {
3     return a/b;
4  }
5
6  int main()
7  {
8     int a,b;
9     cout << "Valeur pour a : ";
10    cin >> a;
11    cout << "Valeur pour b : ";
12    cin >> b;
13
14    cout << a << " / " << b << " = " << division(a,b) << endl;
15
16    return 0;
17 }
```

```
Valeur pour a : 5
Valeur pour b : 0
Exception en point flottant (core dumped)
```

Exemple sans exception

```
1 int division(int a,int b) // Calcule a divisé par b.
2 {
3    if( b!=0)   // Si b ne vaut pas 0.
4       return a/b;
5    else          // Sinon.
6       cout << "ERREUR : Division par 0 !" << endl;
7 }
```

Cas 1 : => On s'attend à un retour de type entier ?

```
1 int division(int a,int b) // Calcule a divisé par b.
2 {
3    if(b!=0)   // Si b ne vaut pas 0.
4       return a/b;
5    else          // Sinon.
6       return ERREUR;
7 }
```

Cas 2 : => Quelle est la valeur de ERREUR ?

Les exceptions

- Déclaration des exceptions
  - Exceptions prédéfinies (classe Exception)
  - Exceptions déclarées par l'utilisateur

- Lancé d'une exception
  - Automatique
  - Par le programmeur

- Capture et traitement de l'exception lancée

- Exemple

Exemple : **if** Delta < 0.0 **throw**(type t);

Remarque : L'*ordre* **throw** sans indication d'un identificateur d'exceptions sert uniquement à un preneur en mains d'exceptions.

➡ Exception capturer non traitée : le programme continue avec des données erronées

Gestion des exceptions

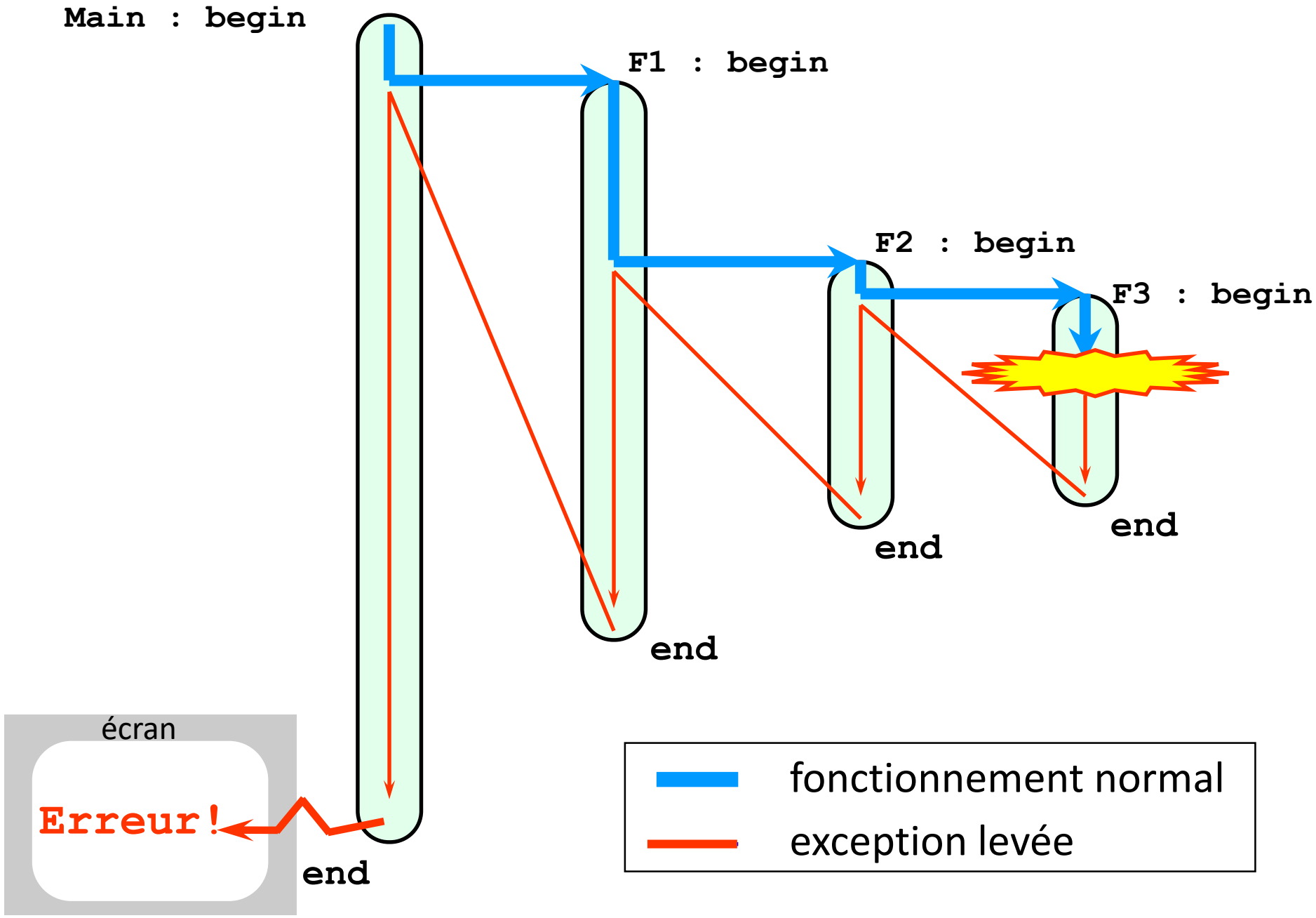
- L’utilisation

– Si une exception n'a pas de gestionnaire alors si une telle exception est levée le programme est immédiatement interrompu avec un message abscon.

– Quand plusieurs blocs try sont imbriqués, la recherche du bloc catch pour le déroutement se fait du bloc try le plus imbriqué à celui le moins imbriqué.

– Après la gestion d'une exception, si le programme n'a pas été interrompu alors ce dernier reprend juste après la fin du bloc try/catch.

Propagation d'une exception



Gestion des exceptions

- Le principe : Une exception est le déroutement ou rupture de séquence d'un programme déclenchée par l'instruction **throw**.

- Le déclenchement d'une exception se fait avec l'instruction **throw** qui reçoit en paramètre une variable.

- Le throw doit être dans un bloc **try** … **catch**

- L'utilisation du principe

  – Un morceau de programme dit bloc **try** mis sous surveillance pour détecter une exception qui pourrait s'y produire

```
try
{
    …
}
```

  – Le bloc **try** est suivi d'un ou plusieurs blocs dits blocs **catch** qui sont les gestionnaires d'exceptions. Chaque bloc a un type d'exception. Le programme est dérouté dans un bloc **catch** à chaque fois qu'une exception du type du bloc se produit.

```
catch(Type t)
{
    …
}
```

Déroutement

Le programme sera dérouté vers le bloc catch dont le type est celui de la variable.

```
1  try
2  {
3      // Le bloc sensible aux erreurs.
4  }
5  catch(int e) //On rattrape les entiers lancés (pour les entiers, une
   référencen'a pas de sens)
6  {
7     //On gère l'erreur
8  }
9  catch(string const& e) //On rattrape les strings lancés
10 {
11    // On gère l'erreur
12 }
13 catch(Personnage const& e) //On rattrape les personnages
14 {
15    //On gère l'erreur
16 }
```

Gestion des exceptions

```
1  int division(int a,int b)
2  {
3      try
4      {
5          if(b == 0)
6              throw string("Division par zéro !");
7          else
8              return a/b;
9      }
10     catch(string const& chaine)
11     {
12         cerr << chaine << endl;
13     }
14 }
```

On revient au Cas 1, on s'attend à un retour de type entier ?

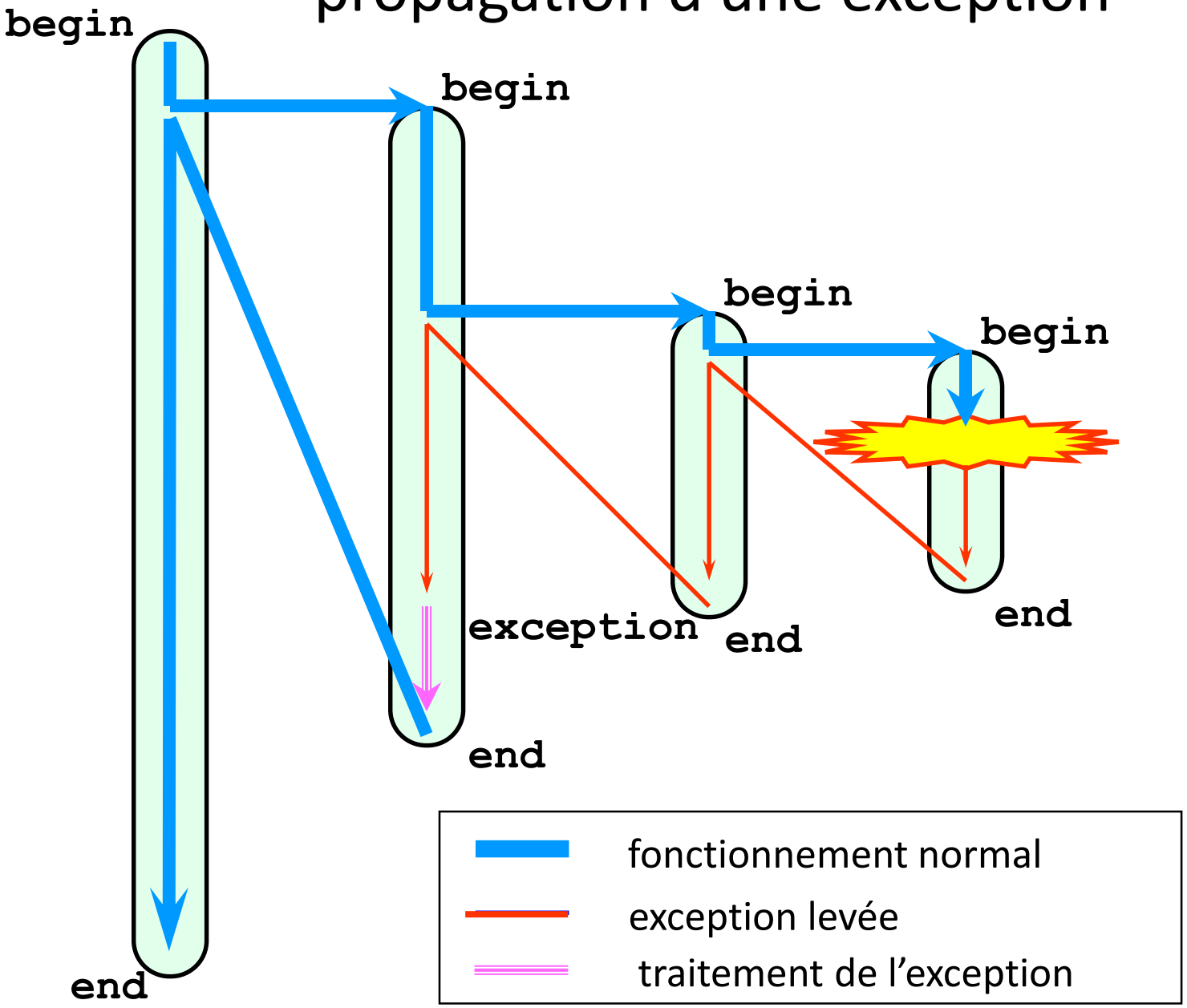
```
Valeur pour a : 5
Valeur pour b : 0
Division par zéro !
```

```cpp
int division(int a,int b) // Calcule a divisé par b.
{
    if(b==0)
        throw string("ERREUR : Division par zéro !");
    else
        return a/b;
}

int main()
{
    int a,b;
    cout << "Valeur pour a : ";
    cin >> a;
    cout << "Valeur pour b : ";
    cin >> b;

    try
    {
        cout << a << " / " << b << " = " << division(a,b) << endl;
    }
    catch(string const& chaine)
    {
        cerr << chaine << endl;
    }
    return 0;
}
```

Relancer une exception

```
try
{
}
catch (TypeException &e)
{
    // code de traitement local
    throw; // Relance l'exception
}
```

propagation d'une exception



Gestion des exceptions

L’utilisation

– Une fonction ou méthode appelée susceptible de lever une exception peut demander le déroutement à la fonction ou méthode appelante. Le prototypage de la fonction appelée doit le préciser :
  - … fonctionAppelee(..) throw(A, B) { …. }

– L'utilisation d'un throw sans argument dans un catch retransmet l'exception dans le niveau try/catch immédiatement supérieur.

L'exception standard

- On peut lancer un entier, un réel ou un caractère
- On peut aussi, lancer un objet qui contiendrait plusieurs attributs comme :
  - une phrase décrivant l'erreur ;
  - le numéro de l'erreur ;
  - le niveau de l'erreur (erreur fatale, erreur mineure...) ;
  - l'heure à laquelle l'erreur est survenue ;

Exception pour le contrôle de saisie

```
#include <iostream>
#include <exception>
using namespace std;

int saisirInt();

int main()
{
   cout << "Hello world!" << endl;
   int a = saisirInt();
   cout << " valeur saisie =  "
   cout << a << endl;
   return 0;
}
```

```
int saisirInt()
{
  int a;
  while(1)
  {
    try
    {
       cout << "entrer un entier compris entre 1 et 10: " ;
       cin >> a ;
       if(a <1 ||  a> 10) throw(10);
       return a ;
    }
     catch (int a)
    {
      cout << "erreur de saisie" << endl;
    }
  }//...
}
```

Fichier essai.h

```
#include <iostream>
using namespace std
class Essai {
 public :
   class Erreur {
       public:
          Erreur(int n=0): _val(n) { }
          int get_val() { return _val; }
       private:
          int _val;
    };
   Essai() { cout << "Constructeur d'Essai" << endl; }
   ~Essai() { cout << "destructeur d'Essai" << endl; }
   // ...
   void f1() {
    throw Erreur(10); // construction d'une
                      //instance de Erreur
                      //initialiée à 10 et
                      //lancement de celle-ci
   }
};// fin de la classe Essai
```

Fichier main.cpp

```
#include "essai.h"
void main() {
 try {
    Essai e1;
    e1.f1();
    cout << "bla bla bla" << endl;    //
  }
 catch ( Essai::Erreur e ) {
   cout << "Erreur numéro : " << e.get_val() << endl;
  }
}// fin de main
```

```
/****** résultat de l'exécution ********
Constructeur d'Essai
destructeur d'Essai
Erreur numéro : 10
**********************************/
```

```
#include <iostream>
using namespace std;

class erreur  { // Première exception possible, associée  à l'objet erreur.
public:
   int cause;  // Entier spécifiant la cause de l'exception.
               // Le constructeur. Il appelle le constructeur de cause.
   erreur(int c) : cause(c) {}   // Le constructeur de copie. Il est utilisé
                                 //par le mécanisme des exceptions :
   erreur(const erreur &source) : cause(source.cause) {}
}; // fin de class

class other {};   // Objet correspondant à toutes les autres exceptions.

int main(void)
{
   int i;          // Type de l'exception à générer.
   cout << "Tapez 0 une exception Erreur, "<< " taper 1 pour une exception Entière :";
   cin >> i;       // On va générer une des trois exceptions possibles.
   cout << endl;
…….. //next slide
```

```
try            // Bloc où les exceptions sont prises en charge.
  {
    switch (i)    // Selon le type d'exception désirée,
    {
    case 0:
      {
        erreur a(0);
        throw (a);   // on lance l'objet correspondant
                     // (ici, de classe erreur).
                     // Cela interrompt le code. break est
                    // donc inutile ici.
      }
    case 1:
      {
        int a=1;
        throw (a);   // Exception de type entier.
      }
    default:          // Si l'utilisateur n'a pas tapé 0 ou 1,
      {
        other c;     // on crée l'objet c (type d'exception
        throw (c);   // other) et on le lance.
      }
    }
  }             // fin du bloc try. Les blocs catch suivent :
```

```
    catch (erreur &tmp) // Traitement de l'exception erreur ...
    {                           // (avec récupération de la cause).
        cout << "Erreur erreur ! (cause " << tmp.cause << ")" << endl;
    }
    catch (int tmp)   // Traitement de l'exception int...
    {
        cout << "Erreur int ! (cause " << tmp << ")" << endl;
    }
    catch (...)       // Traitement de toutes les autres
    {                 // exceptions (...).
                      // On ne peut pas récupérer l'objet ici.
        cout << "Exception inattendue !" << endl;
    }
    return 0;
}
```

Classe Exception

- La classe exception est la classe de base de toutes les exceptions lancées par la bibliothèque standard.
- Pour l'utiliser il faut dériver une classe de cette classe
- il faut inclure le fichier d'entête correspondant <exception>

```
class exception
{
public:
   exception() throw(){ } //Constructeur.
   virtual  exception() throw(); //Destructeur.

   virtual const char* what() const throw(); //Renvoie une chaîne
                                             // contenant des infos sur l'erreur.
};
```

- L'exercice en TD