FACULTAD DE INFORMÁTICA

Excepciones

Baltasar Fernández Manjón

http://www.fdi.ucm.es/profesor/balta/

Excepciones

- Permiten tratar situaciones anómalas que aparecen durante la ejecución de un programa
- Su gestión permite la detección y corrección de errores en ejecución
  - Simplifican los programas
    - Se diferencia el código normal del código de tratamiento de errores
  - Sólo se deben usar cuando no se puede resolver la situación anómala directamente en ese contexto
    - Se tiene que seguir haciendo el control de errores habitual

Excepciones

- Sintaxis

```
try {
  // código que puede producir una excepción
}
catch (tipoExcepción objeto){
  // código que gestiona la excepción
}
```

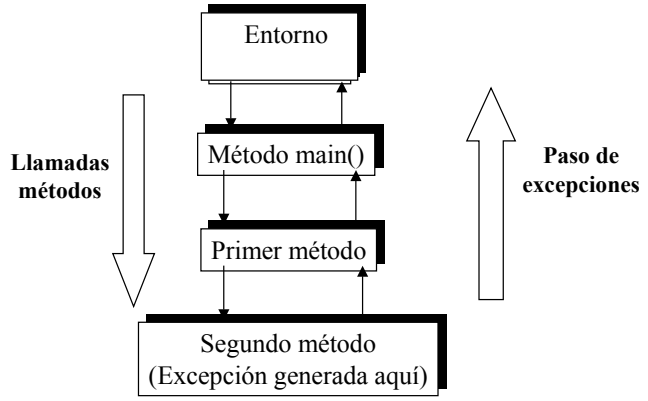
- El bloque ***try*** envuelve el código "peligroso"
- Uno o varios bloques ***catch*** se ocupan de capturar y gestionar la excepción o excepciones producidas
  - Cada bloque ***catch*** puede tener, como mucho, un parámetro que permite acceder al valor del objeto de la excepción

- try
  - Es el bloque de código donde se prevé que se genere una excepción
  - Debe ir seguido por lo menos por un bloque *catch*
- catch (TipoExcepcion e)
  - Es el código que se ejecuta cuando se produce la excepción del tipo indicado
  - Pueden aparecer varias instrucciones *catch* para tratar distintas excepciones
  - Cuando se produce una excepción se compara si coincide con la excepción del primer *catch*. Si no coincide con la del segundo y sucesivos

Funcionamiento general

- Si se produce una excepción dentro del bloque `try`
  - Se ignora el resto del código del `try`
  - Se pasa el control a los bloques `catch` para que, por orden, capturen y gestionen la excepción
  - Si la excepción no se captura y gestiona la función termina su ejecución
  - Si se captura la excepción el flujo de programa continúa después de los bloques `catch`
- Si no se produce una excepción en el bloque `try`
  - Se ignoran los bloques `catch`

Propagación de excepciones



Todas las excepciones que se puedan generar se han de manejar o delegar si no se invoca **`terminate`**

Throw

- La palabra reservada **`throw`** permite lanzar de forma explícita una excepción
  - **`throw tipoDato_o_clase;`**
  - Se puede usar
    - cuando se produce un error y falta contexto para poder corregirlo
    - para propagar una excepción tratada parcialmente
  - Puede lanzar casi cualquier cosa (objetos de excepción, un entero, etc)
    - **`throw miExcepción;`**
    - **`throw -1;`**

Relanzado de excepciones

- En un bloque catch se pueden lanzar excepciones
  - Aunque no es muy habitual
- La excepción se lanza hacia el siguiente bloque catch
  - mas exterior o
  - situado mas arriba en la pila de llamadas
- Se puede relanzar la misma excepción o lanazar una nueva
  - rethrow;
    - Propaga la misma excepción
  - throw nuevaExcepción;
    - Lanza una nueva excepción al siguiente bloque catch

Los objetos de excepción

- La clase base de los objetos de excepción es **`exception`**
  - **`#include <exception>`**
  - El constructor de **`exception`** admite una cadena como parámetro
    - Permite describir la excepción
  - Proporciona una función virtual **`what()`** que devuelve la cadena descriptiva de la excepción

Capturas de excepciones en el catch

- Cada bloque del catch captura sólo un tipo de excepción
- Para capturar cualquier excepción
  - **`catch(...){}`**
  - ¡Cuidado! Hay que ponerlo como último bloque -> si no los otros bloques núnca se ejecutan

Jerarquía de excepciones estándar

- Clase base **`exception`**
  - Algunas clases derivadas
    - **`runtime_error`**, **`logic_error`**
    - **`bad_alloc`, `bad_cast`, `bad_typeid`**
      - Estas excepciones son producidas por **`new`, `dynamic_cast`** y **`typeid`** cuando se producen errores
- Esta jerarquía de excepciones se tiene en cuenta en el orden de captura y tratamiento en los **`catch`**
  - **`catch(...)`**
    - Captura todas las excepciones
  - **`catch( exception unaExcepción)`**
    - Captura todas las excepciones derivadas de **`exception`**

Especificación de excepciones de una función

- Una función debería listar las excepciones que puede lanzar y que no captura y trata
  - Tanto en la declaración como en la definición

```
int mifunción( TipoParámetro parámetro )
  throw ( ExcepciónTipoA, ExcepciónTipoB)
{
   // cuerpo de la función
}
```

  - Sólo debería lanzar **`ExcepciónTipoA`** **y** **`ExcepciónTipoB`** (y clases derivadas de ellas)
    - Si lanza otro tipo de excepción se invoca la función **`unexpected()`**
      - **No se produce error ni en compilación ni en ejecución**
    - Por defecto su comportamiento es que el programa termina

Excepciones de las funciones

- Si no hay lista de excepciones podría lanzar cualquier excepción
  - Si se lanza se propaga dicha excepción
- Si la lista de excepciones es vacía no se debería lanzar ninguna excepción
  - Si se lanza alguna se invoca a **`unexpected()`**
- El funcionamiento por defecto de **`unexpected()`** es terminar el programa invocando **`terminate()`**
  - Se puede modificar su comportamiento mediante
    - set_terminate(miFunciónDeTerminación);
    - set_unexpected(miFunciónDeExcepciónInesperada);
  - Argumentos de las funciones set
    - Se les pasa un puntero a una función
      - La función no tiene parámetros
      - El tipo devuelto es **`void`**

Uso de las excepciones

- Las excepciones provocan una alteración en el flujo de programa
  - Pueden dificultar la legibilidad del programa si se usan de forma inadecuada
- Se deben utilizar cuando se produce una situación excepcional y no se dispone del contexto para solucionarla
  - No deben sustituir al control habitual de errores

Ejemplo

```
#include <iostream>
#include <exception>
#include <string>
#include <stdlib>
#include <sstream>

using namespace std;
string intToString(int numero); // prototipo

class ExcepcionEdadPersona : public exception {

};  // ExcepcionEdadPersona
```


Ejemplo

```
class Persona {
  private:
      string nombre;
      int edad;
  public:
      Persona(string nom, int ed) throw (ExcepcionEdadPersona): nombre(nom){
        if (ed<0 || ed>125)
            throw ExcepcionEdadPersona();
        else
            edad= ed;
      }; // constructor
       string toString(){
        return nombre+ " " + intToString(edad);
      }; //toString()
}; // Persona
```

Ejemplo

```
// convierte un entero a cadena usando un flujo de salida de cadenas
// es necesario incluir <sstream>
string intToString(int numero)
{
        string cadena;
        ostringstream auxStream;
        auxStream << numero << ends;
        cadena = auxStream.str();
        return cadena;
}; //intToString
int main(){
   try {
         Persona yo("balta", 57);
         cout << yo.toString();
         Persona tu("otro", -3);          cout << tu.toString();
   }  catch (ExcepcionEdadPersona &exc){
        cout << "la edad no es correcta. Objeto no creado" << endl;
        cout << "mensaje: "<<exc.what();
   }  } // main
```