



UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Electrónica

# Manejo de Excepciones en C++

ELO329: Diseño y Programación Orientados a Objetos

Departamento de Electrónica

Universidad Técnica Federico Santa María

# Manejo de Excepciones: Recordar

- ❑ El Manejo de Excepciones es un mecanismo para comunicar estados de error desde una parte del programa a otra.
- ❑ Comúnmente, una parte del programa detecta un error, pero no es conveniente mezclar las situaciones de excepción con el flujo normal y más probable del programa.
- ❑ Otra parte del programa puede hacerse cargo de todos los errores, pero éstos no siempre se generan en esa sección del código.
- ❑ Hay poca diferencia con Java

# Diferencias respecto de Java

- ❑ La sentencia throw admite argumentos escalares (=tipos básicos, int, etc) u objetos.
- ❑ Podemos o no indicar en una función o método el tipo de dato lanzado. Si se indica, sólo podemos lanzar ese tipo de dato. Si no se indica, cualquier tipo puede ser lanzado.
- ❑ La [biblioteca estándar](#) de C++ provee la clase exception de la cual podemos crear clases derivadas.

# Ejemplo: Inserción fuera de rango

- ❑ Una función recibe el requerimiento de inserción de un número en la posición n de un vector. La función descubre que n es mayor que el tamaño del vector, por lo tanto lanza o envía un excepción, la cual hace retornar inmediatamente la función al segmento de código llamador.

- ❑ Notar que no hay restricciones para la clase del objeto retornado.

```
void Insert( vector<int> & array, int index, int value ) {  
    if( index < 0 || index >= array.size())  
        throw string("Index out of bounds in Insert()");  
  
    //....  
    array[index] = value;  
}
```

- ❑ El prototipo también pudo ser:

```
void Insert(vector<int> &array, int index, int value) throw(string)
```

# Invocación a Insert() del escenario previo

- ❑ El bloque try rodea cada sección de código siendo probado.
- ❑ Una o más sentencias catch siguen al bloque try.

```
try {  
    cout << "Enter an index between 0 and "<< (VECSIZE-1) << ": ";  
    cin >> index;  
    Insert( scores, index, value );  
    cout << "Insertion successful.\n";  
} catch( string & s ) {  
    cout << s << endl;  
}
```

# Caso más general

- ❑ Para capturar varias excepciones posibles:

```
try {  
    // code here  
} catch (int param) { cout << "int exception"; }  
    catch (char param) { cout << "char exception"; }  
    catch (...) { cout << "default exception"; }
```

- ❑ Las funciones o métodos puedes indicar tipo lanzado:

```
float myfunction (char param) throw (int);
```

- ❑ Si queremos prohibir las excepciones:

```
int myfunction (int param) throw(); // C++98
```

```
int myfunction (int param) noexcept; // desde C++11
```

- ❑ Si queremos permitir cualquier tipo de excepción:

```
int myfunction (int param);
```

# Clases para excepciones

- ❑ Podemos definir nuestras propias clases para manejo de excepciones.
- ❑ Éstas pueden o no derivar de [exception](#)
- ❑ La clase exception tiene un método virtual con siguiente prototipo:  
`virtual const char* what() const throw() // C++98`  
`virtual const char* what() const noexcept // C++11`
- ❑ Si derivamos de exception, agregar  
`#include <exception>`  
`using namespace std;`
- ❑ La clase excepción usualmente lleva por nombre la excepción, por ejemplo RangeException.
- ❑ Consideremos al siguiente clase simple:  
`class RangeException { }; // para índices fuera rango`

# Clases para lanzar Excepción

- Esta versión de la función Insert construye y lanza un objeto RangeException si el índice está fuera del rango.

```
void Insert( vector<int> & array, int index, int value )  
{  
    if( index < 0 || index >= array.size())  
        throw RangeException();  
    array[index] = value;  
}
```



# Atrapando una Excepción

- Ahora la función llamadora puede nombrar un tipo específico de excepción en la sentencia catch.

```
try {  
    cout << "Enter an index between 0 and " << (VECSIZE-1) << ": ";  
    cin >> index;  
    Insert( scores, index, value );  
    cout << "Insertion successful.\n";  
}  
catch( RangeException & ) {  
    cout << "A range exception occurred.\n";  
}
```

# Atrapando múltiples Excepciones

```
try {  
    DoOneThing();  
    DoAnother();  
    DoSomethingElse();  
}catch( RangeException & ) {  
    cout << "A range exception occurred.\n";  
}catch( OpenFileError & ) {  
    cout << "Cannot open file.\n";  
}  
// etc...
```

# Clase RangeException (más completa)

- Una mejor versión de la clase RangeException nos permite pasar un string a su constructor. También incluimos un método GetMsg que retorna el mismo string.

```
class RangeException {  
public:  
    RangeException(const string & msg)  
    { m_sMsg = msg; }  
  
    string GetMsg() const  
    { return m_sMsg; }  
  
private:  
    string m_sMsg;  
};
```

# Class RangeException

```
void Insert( vector<int> & array, int index, int value ) {  
    if( index < 0 || index >= array.size())  
        throw RangeException("Index out of bounds in Insert()");  
    array[index] = value;  
}
```

# Re- envío de un Excepción

- Algunas veces es útil lanzar una excepción nuevamente y dejar que la función previa en la cadena de llamados se haga cargo de su manejo.

```
void TestVector(vector<int> & scores, int value) {  
    int index;  
    try {  
        cout << "Enter an index between 0 and " << (VECSIZE-1) << ": ";  
        cin >> index;  
        Insert1( scores, index, value );  
        cout << "Insertion successful.\n";  
    } catch( RangeException & R ) {  
        throw R;  
    }  
}
```

# Capturando Excepciones Desconocidas

- Si una excepción es lanzada en algún lugar en la cadena de llamados y nunca es atrapada, ésta puede ser capturada usando (...) como el parámetro de la sentencia try-catch.

```
void main() {  
    try {  
        Example2();  
    }  
    catch( ... ) {  
        cout << "Caught unknown exception in main()\n";  
    }  
}
```