

Manejo de Excepciones

Lectura sugerida de texto en línea:

- Eckel, Bruce. *Thinking About C++*, 2nd Edition, Prentice-Hall, 2000. Gratis en www.bruceeckel.com.

Agustin J. Gonzalez
ELO329

Manejo de Excepciones

- **El Manejo de Excepciones** es un mecanismo interno 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 estos no se generan en esa sección del código.

Situación común

- 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.
- El código llamador presumiblemente repite el lazo solicitando un nuevo índice y vuelve a llamar a la función.

Función Insert() en el escenario previo

La función Insert usa la sentencia **throw** para retornar tan pronto como se detecta que el índice es muy grande. El throw causa el retorno inmediato de la función.

```
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;
}

// more...
```

- 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;  
}
```

Clases para excepciones

- Podemos definir nuestras propias clases para manejo de excepciones. Ésta define el tipo de objeto lanzado cuando una excepción ocurre.
- La clase excepción usualmente lleva por nombre la excepción, por ejemplo *RangeException*.

```
class RangeException { };  
// use for out of range subscripts
```

Clases 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;  
}
```

Paréntesis son requeridos!

Atrapando una Excepción

- Ahora la función llamadora puede nombrar un tipo de excepción específico 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

Usamos múltiples sentencias catch para atrapar todos los tipos de excepciones que pueden ser lanzadas.

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

Clase RangeException

Una mejor versión de la clase RangeException nos permite pasar un string a su constructor. También hay 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;
};
```

Clase RangeException

Cuando la función Insert detecta un índice errado, ésta pasa un string al constructor the 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;
}
```

Clase RangeException

Cuando el llamador atrapa la excepción enviada por Insert, éste ahora puede llamar GetMessage para desplegar el mensaje almacenado en el string.

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

Re-envío de un Excepción

Algunas veces es útil lanzar una excepción nuevamente y dejar que la función previa en la cadena 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;
    }
} // more...
```

Re-envío de excepciones

En este ejemplo la función llamadora debe tener una sentencia catch para atrapar la excepción enviada por TestVector.

```
void Example2()
{
    vector<int> scores(VECSIZE);
    int value = 99;

    try {
        TestVector( scores, value );
    }
    catch( RangeException & R ) {
        cout << R.GetMsg() << endl;
    }
}
```

Envío de Múltiples Excepciones

Una misma función puede lanzar más de una excepción. Ejemplo:

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

    if( value < 0 )
        throw BadArrayValue();

    array[index] = value;
}
```

Capturando Excepciones Desconocidas

Si una excepción es lanzada en algún lugar en la cadena de llamados a función 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";
    }
}
```




Fin