

Manejo de errores:

Los errores de programación pueden clasificarse en varios tipos.

Errores de diseño: son los más difíciles de corregir y prevenir. Se deben a un defecto del algoritmo.

Errores de sintaxis: son errores en el programa fuente que se detectan en la fase de compilación.

Avisos: son errores, pero no lo suficientemente graves como para impedir la generación del código objeto

Errores de enlazado: se refieren a funciones que no están definidas en ninguno de los ficheros objetos ni en las librerías.

Errores de ejecución: los que ocurren con el ejecutable. Normalmente el programa terminará bruscamente. (por ejemplo acceso fuera de rango, memoria insuficiente, etc)

Que se espera de un programa:

Robustez	El sistema debe gestionar el error de un programa de modo diferente al flujo general de la ejecución del mismo.
Reusabilidad	En contextos distintos, el manejo de los errores puede variar
Extensibilidad	Un error puede transformarse en otros, que puedan manejarse

Estas son algunas alternativas tradicionales para el manejo de errores:

Devolución de un valor especial	Se devuelve un valor que no tiene sentido en el dominio, indicando que se produjo un error	a) no siempre existe un valor especial b) no es robusto
Retorno silencioso	La función finaliza como si hubiera cumplido su objetivo, pero no es así	Dificultad para reutilizarla y para extenderla Es imposible manejar el error
Finalización de ejecución	Colocar un exit cuando no pueda realizarse	No hay reutilización ni extensibilidad
Comprobaciones de error	Se detecta un eventual error a priori (antes de realizar una operación) o bien a posteriori (después de haber solicitado el servicio). Se puede hacer: Mediante operaciones específicas Mediante devolución de códigos de error(a posteriori) Mediante variables globales	La comprobación de errores no funciona cuando se acceden a las abstracciones desde diversos hilos (<i>threads</i>) No son soluciones robustas Son más rápidas que el manejo de excepciones

Como manejar asertos y excepciones en C++

Asertos (assert):

Los **Asertos** (aserciones) son condiciones a analizar y que determinan en la correcta ejecución de un programa. De producirse, se detiene la ejecución.

No se usan para detectar errores en tiempo de ejecución. No son reutilizables ni extensibles

Se usan para detectar situaciones que nunca deberían ocurrir (postcondiciones, invariantes...)

Si se dan, es un **error de implementación** que se debe corregir.

Deben deshabilitarse una vez la aplicación haya sido probada exhaustivamente

Ejemplo.

```

void main()
{
    int i,x;
    for ( i = 1; i <= 5; i++)
    {
        cout<<"Ingrese un valor positivo"<<endl;
        cin>>x;
        assert(x > 0);
        cout<<x*x;
    }
}

```

Excepciones (exception):

Una excepción es un evento que se produce en un momento de ejecución y que impide que la ejecución prosiga por su flujo normal.

En C++, cualquier variable (simple u objeto) del lenguaje puede representar una excepción.

Cuando se utiliza el manejo de excepciones, se separan los roles de quien **detecta** el error y quien **maneja** (o se recupera) el error. Puede que interese manejar una misma excepción de distinta manera si el ‘escenario’ es diferente.

Lanzamiento de excepciones : throw

Si se da una situación excepcional en el código, se envía información acerca de lo sucedido , usando un objeto y lanzándolo fuera de contexto (ámbito’)

Ej:

```
throw error (“ocurrió un error”)
```

error es una clase, se genera un objeto de la misma y se le pasa al constructor la constante cadena “ocurrió un error”.El objeto lanzado deberá tener un constructor copia correcto.

Exceptuando los casos en que son lanzadas por las propias librerías C++ (como consecuencia de algún error), las excepciones **no** se lanzan de modo espontáneo.

El programa deberá tener la sentencia adecuada para lanzar la excepción en su caso.

Es similar a un mecanismo de return.

Por lo general, se arrojan distintos tipos de objetos para los distintos tipos de errores.

Cuando se lanza una excepción se pasa el **flujo de la ejecución** a otra rutina, desapilándose las llamadas a rutinas hasta que se encuentra un manejador apropiado

Ejemplo (Stroustrup):

```

class Vector
{
    int* p;
    int sz;
public:
    class Range { }; //una clase para excepciones definida dentro de Vector
    int& operator[] (int i)
    {
        if (0<=i && i < sz)
            return p[i];           //en el rango, OK
            throw Range;           // excepción: se lanza un objeto Range
    }
}

```

Bloque try :

Si se está en una función que puede lanzar una excepción, o si se llama a una función que arroja una excepción, esa función terminará en el proceso de arrojar la excepción.

Para esto se puede usar un bloque especial en la función: try, que es un scope común dentro del cual se pone todo las sentencias necesarias, algunas de las cuales pueden lanzar una excepción, sin hacer chequeo de errores.

```
try {  
    // invocación a rutinas que pueden  
    // lanzar algún tipo de excepción  
}
```

Con el bloque try el programa intenta realizar cierta acción .

Si se da un error (excepción) durante el intento, entonces se lanza una excepción y se transfiere el control de ejecución al punto en el cual exista un manejador de excepciones (handler) que coincida con el tipo lanzado.

De no producirse excepción, el programa sigue su curso normal.

Puede lanzarse excepciones de diversos tipos y pueden existir también manejadores de varios tipos. Puede ocurrir que se lance una excepción para la que no existe manejador adecuado.

Todo bloque try debe estar seguido por el bloque manejador de la excepción.

Bloque manejador de la excepción : catch

El objeto creado y lanzado por throw se captura en sentencia catch.

El tipo de información contenida en el objeto generalmente determina el tipo de error que se ha producido.

En el bloque se especifica el tipo de excepción a manejar y el código de manejo. El valor devuelto por el throw se asigna al objeto del catch adecuado.

Para un bloque try debe haber al menos un bloque catch. Un bloque catch puede ir detrás de un try u otro catch.

Habitualmente se dice que el handler captura la excepción.

El handler es el bloque de código para manejar la excepción que comienza con la palabra catch.

C++ exige al menos un manejador inmediatamente después de un bloque try.

Si en un bloque try se ha lanzado una excepción y no hay ningún catch apropiado para manejarla es como si ésta se volviese a lanzar, igual que si no se hubiese puesto el try

Si desde un catch se utiliza un throw sin parámetros, se relanza la excepción que está siendo manejada.

Si una excepción nunca es manejada se llama a la función terminate de <exception>.

Dicha función, por omisión, llama a abort de <cstdlib>.

La funcionalidad de terminate se puede modificar con set_terminate en <exception>

El soporte de especificación lanzadas por una excepción puede no estar implementado.

C++ permite atrapar la excepción por valor, por referencia o por puntero.

La mejor opción es hacer catch por referencia.

Pasos del bloque try-catch:

Si se produce una circunstancia excepcional:

Se lanza la excepción señalada por la sentencia **throw**

El programa busca por un manejador ("handler") adecuado a la excepción

Si se encuentra el "handler", la pila (stack) es recorrida hacia abajo hasta el punto donde está el manejador, y el control del programa es transferido al manejador.

Si no encuentra ningún manejador se invoca la función terminate()

Si no se produce ninguna excepción, el programa continúa normalmente , sin ‘ver’ los bloques catch que pueden seguir.

```
class except1 : public exception {
```

```

        datos y funciones miembros de la clase except1
    } //si se derivan de exception...

class excep2 :public exception {
        datos y funciones miembros de la clase except2
    }

...
    try {
        // código durante la ejecución del cual se puede lanzar una excepcion
    }
    catch(excep1&)
    {
        ... // se atrapa la excepción, se trata el caso, incluyendo la posibilidad de exit o relanzamiento
    }
    catch(excep2 &)
    {
        .... // idem anterior
    }
    //continúa el programa

```

Especificación de excepciones

A declarar funciones y métodos se puede especificar qué excepciones pueden ser lanzadas desde su cuerpo. Es una práctica conveniente porque documenta el método y permite controlar el un eventual lanzamiento erróneo de excepciones.

Las funciones con especificador de excepción no son susceptibles de sustitución inline.

En C++ esto se hace con la palabra reservada `throw` seguida de la enumeración de posibles excepciones lanzadas entre paréntesis y separadas por comas después de la declaración y definición de un método o función .

Si no se especifica `throw` , se podrá lanzar cualquier excepción

Ejemplo:

```

void f()           // puede lanzar cualquier excepción
void g() throw () // g no puede lanzar excepciones
void h() throw( B) // h sólo puede lanzar objetos de tipo h

```

Excepciones y constructores:

Si un constructor falla, es conveniente lanzar una excepción, dado que los constructores no pueden retornar un código de error por no retornar tipo.

Excepciones y destructores:

Los destructores no deben lanzar excepciones debido al *unwinding*.

Se llama *unwinding* al proceso que se da cuando desde un bloque `try` se lanza una excepción ; consiste en que todos los stack frames se desapilan (debido al `throw` el control sale del bloque `try` correspondiente), hasta encontrar el handler adecuado.

En el proceso de desapilamiento se destruyen todos los objetos automáticos creados hasta el momento de la excepción, para lo cual se ejecutará el destructor correspondiente a cada uno. Si uno de estos destructores lanza una excepción, en el runtime system del C++ se puede dar una situación sin salida.

Si la excepción del destructor no tiene el handler adecuado, se llama a `terminate()`.

El mejor modo de prevenir esta situación es nunca lanzar una excepción desde un constructor.

La Standard C++ Library y las excepciones:

STL de C++ soporta un conjunto de excepciones estándar (ver Stroustrup).

exception: es la clase base para todas las excepciones de la librería estándar de C++.

La función what() permite trabajar con las cadenas de caracteres que corresponden al mensaje de error correspondiente.

logic_error: es una clase derivada de exception que reporta errores lógicos del programa, los cuales presumiblemente se detecten antes de la ejecución del programa.

runtime_error: derivada de exception, reporta errores en tiempo de ejecución, los cuales presumiblemente pueden ser detectados sólo cuando se ejecuta el programa.

De logic_error derivan domain_error, invalid_argument, length_error, out_of_range, bad_cast, bad_typeid

De runtime_error derivan range_error, overflow_error (para ow aritmético), bad_alloc