Tratamiento de excepciones

Tratamiento de excepciones

El problema de la seguridad es uno de los clásicos quebraderos de cabeza de la programación. Los diversos lenguajes han tenido siempre que lidiar con el mismo problema: ¿Qué hacer cuando se presenta una circunstancia verdaderamente imprevista? (por ejemplo un error). El asunto es especialmente importante si se trata de lenguajes para escribir programas de "Misión crítica"; digamos por ejemplo controlar los ordenadores de una central nuclear o de un sistema de control de tráfico aéreo.

Tratamiento de excepciones

Antes que nada, digamos que en el lenguaje de los programadores C++ estas "circunstancias imprevistas" reciben el nombre de excepciones, por lo que el sistema que implementa C++ para resolver estos problemas recibe el nombre de manejador de excepciones. Así pues, las excepciones son condiciones excepcionales que pueden ocurrir dentro del programa durante su ejecución. Por ejemplo, que ocurra una división por cero, se agote la memoria disponible, Etc. que requieren recursos especiales para su control.

Manejo de excepciones en C++

El manejo de excepciones C++ se basa en un mecanismo cuyo funcionamiento tiene tres etapas básicas:

1: Se intenta ejecutar un bloque de código y se decide qué hacer si se produce una circunstancia excepcional durante su ejecución.

2: Se produce la circunstancia: se "lanza" una excepción (en caso contrario el programa sigue su curso normal).

3: La ejecución del programa es desviada a un sitio específico donde la excepción es "capturada" y se decide que hacer al respecto.

Manejo de excepciones en C++

Para las tres etapas anteriores existen tres palabras clave específicas: try, throw y catch.

Intento (try)

En síntesis podemos decir que el programa se prepara para cierta acción, decimos que "lo intenta". Para ello se especifica un bloque de código cuya ejecución se va a intentar ("try-block") utilizando la palabra clave try.

```
try {    // bloque de código-intento
...
}
```

Intento (try)

El juego consiste en indicar al programa que si existe un error durante el "intento", entonces debe lanzar una excepción y transferir el control de ejecución al punto donde exista un manejador de excepciones ("handler") que coincida con el tipo lanzado. Si no se produce ninguna excepción, el programa sigue su curso normal.

De lo dicho se deduce inmediatamente que se pueden lanzar excepciones de varios tipos y que pueden existir también receptores (manejadores) de varios tipos; incluso manejadores "universales", capaces de hacerse cargo de cualquier tipo de excepción.

Se lanza una excepción (throw).

Si se detecta una circunstancia excepcional dentro del bloque-intento, se lanza una excepción mediante la ejecución de una sentencia throw. Por ejemplo:

```
if (condicion) throw "overflow";
```

Se lanza una excepción (throw).

Es importante advertir que, salvo los casos en que la excepción es lanzada por las propias librerías C++ (como consecuencia de un error), estas no se lanzan espontáneamente. Es el programador el que debe utilizar una sentencia (generalmente condicional) para, en su caso, lanzar la excepción. El lenguaje C++ especifica que todas las excepciones deben ser lanzadas desde el interior de un bloque-intento y permite que sean de cualquier tipo. Como se ha apuntado antes, generalmente son un objeto (instancia de una clase) que contiene información. Este objeto es creado y lanzado en el punto de la sentencia throw y capturado donde está la sentencia catch. El tipo de información contenido en el objeto es justamente el que nos gustaría tener para saber que tipo de error se ha producido. En este sentido puede pensarse en las excepciones como en una especie de correos que transportan información desde el punto del error hasta el sitio donde esta información puede ser analizada.

La excepción es capturada en un punto específico del programa (catch).

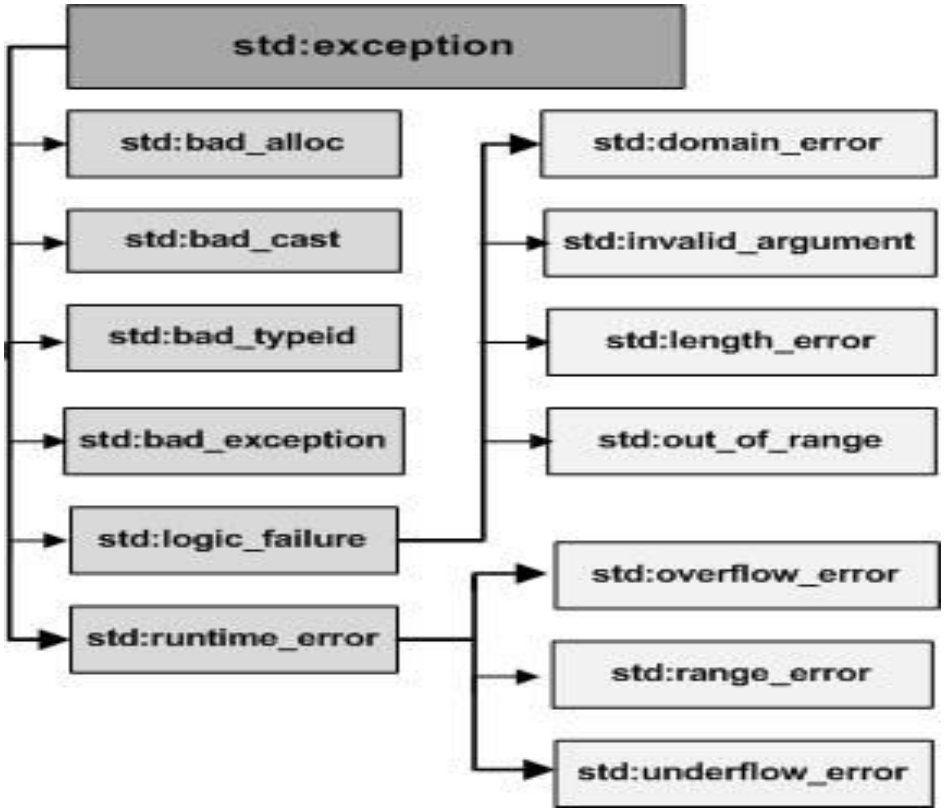
Esta parte del programa se denomina manejador ("handler"); se dice que el "handler" captura la excepción. El handler es un bloque de código diseñado para manejar la excepción precedido por la palabra catch. El lenguaje C++ requiere que exista al menos un manejador inmediatamente después de un bloque try. Es decir, se requiere el siguiente esquema:

```
try {         // bloque de código que se intenta
}
catch (...) { // bloque manejador de posibles excepción
}
...           // continúa la ejecución normal
```

catch

El "handler" es el sitio donde continúa el programa en caso de que ocurra la circunstancia excepcional (generalmente un error) y donde se decide qué hacer. A este respecto, las estrategias pueden ser muy variadas (no es lo mismo el programa de control de un reactor nuclear que un humilde programa de contabilidad). En último extremo, en caso de errores absolutamente irrecuperables, la opción adoptada suele consistir en mostrar un mensaje explicando el error. Puede incluir el "Avise al proveedor del programa" o bien generar un fichero texto (por ejemplo: error.txt) con la información pertinente, que se guarda en disco con objeto de que pueda ser posteriormente analizado y corregido en sucesivas versiones de la aplicación

Excepciones Estandart de C++



Lanzamos un TROW y atrapamos un CATCH

```
#include <iostream>
using namespace std;
int u;
int main() {
  cout << "Lanzar Throw\n";
  try{cin>>u;
  if(u==0){ throw "EL valor no debe ser 0";}
    }
  catch(const char* e){cout<<e; u=1;}
  cout<<"\n Division :"<<220/u;  }
```

https://replit.com/join/ftpjxsdvkw-maximo-arielari

Ejemplo Bad -ALLOC

```
#include <iostream>
using namespace std ;
const int TAM=1000000;
typedef char str[TAM];
struct registro{
      str mat1[TAM][TAM];
      str mat2[TAM][TAM];
      str mat3[TAM][TAM];
      str mat4[TAM][TAM];
};
```

```
int main() {
  cout << "Test de Error!\n";
  try{//bloque que estudia error
     registro *p;
     while(true){
        p=new registro;
        cout<<"\n Generando";
     }}
  catch(bad_alloc & caso){
cout<<"Error de:"<<caso.what();
  } }
```

Ingreso erróneo de información

Por defecto, cuando a un objeto de la familia std::basic_istream (como std::cin) lee algo diferente a la entrada que le proporciona, no lanza un error.

Pero podemos configurarlo para que lance excepciones en caso de lectura errónea usando la función std::basic_ios::exceptions:

Ingreso erróneo de información

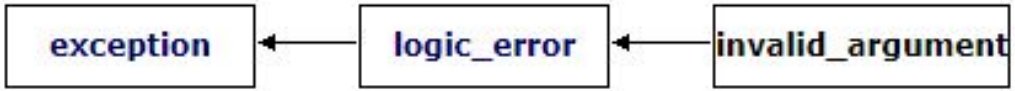
```
#include <iostream>

using namespace std;

int main() {  cout << "Prueba de error!\n";

  int a =0; int b=0;

  cin.exceptions(ios_base::failbit);

  try{ cout << "\nIngrese A: ";cin >> a;  cout <<"\nIngrese B: ";cin >> b;}

      catch (ios_base::failure &error){   cout << "Algo ha pasado al leer!";
cout<<"\n"<<error.what();}

cout <<"\nA: "<< a << "B:" << b;}
```

https://replit.com/join/acyksbdwgg-maximo-arielari

invalid_argument

Esta clase define el tipo de objetos lanzados como excepciones para informar un argumento no válido.

Es una excepción estándar que pueden lanzar los programas. Algunos componentes de la biblioteca estándar también lanzan excepciones de este tipo para señalar argumentos no válidos.



Trow en función

```
#include <iostream>
#include <limits>
#include <iostream>
using namespace std;
void validar(int c)
{  if (c > numeric_limits< char> ::max())
    throw invalid_argument("Supera el byte .");}
int main(){  try{  validar(256); }
  catch (invalid_argument& e){ cout << e.what() << endl;return -1;}
    return 0;}
```

https://replit.com/join/vwyfshdttk-maximo-arielari

TEST

```
#include <iostream>
using namespace std;
int u;int main() {
  cout << "Lanzar Throw\n";
  try{cin>>u;u=u-u;cout<<"\n División :"<<220/u;
  if(u==0){ throw "EL valor no debe ser 0";}}
  catch(const char* e){cout<<e;}
  catch(exception& e){cout<<"\n error :";}}
```

Bibliografía

https://www.zator.com/Cpp/E1_6.htm#:~:text=El%20manejo%20de%20excepciones%20C%2B%2B,circunstancia%20excepcional%20durante%20su%20ejecuci%C3%B3n.

Programación en C++ autor: Luis Joyanes Aguilar.