INFORMÁTICA INDUSTRIAL

Manejo de excepciones en C++.

M. Abderrahim, A. Castro, J. C. Castillo
Departamento de Ingeniería de Sistemas y Automática

uc3m | Universidad Carlos III de Madrid

Agenda

- Manejo de excepciones

Manejo de excepciones

- Las excepciones son en realidad errores durante la ejecución. Si uno de esos errores se produce y no implementamos el manejo de excepciones, el programa sencillamente terminará abruptamente. Es muy probable que si hay ficheros abiertos no se guarde el contenido de los buffers, ni se cierren, además ciertos objetos no serán destruidos, y se producirán fugas de memoria.

- En programas pequeños podemos prever las situaciones en que se pueden producir excepciones y evitarlos. Las excepciones más habituales son las de peticiones de memoria fallidas.

Ejemplo 1: error de asignación de espacio

```
#include <iostream>
using namespace std;

int main() {
   int *x = NULL;
   long y = 100000000000000;

   x = new int[y];
   x[10] = 0;
   cout << "Puntero: " << (void *) x << endl;
   delete[] x;

   return 0;
}
```

```
WJRG-MacBookPro:Clase13 wladimir$ g++ error.cpp -o error
WJRG-MacBookPro:Clase13 wladimir$ ./error
error(94779) malloc: *** mmap(size=400000000000000) failed (error code=12)
*** error: can't allocate region
*** set a breakpoint in malloc_error_break to debug
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc
Abort trap
```

Ejemplo 2: error de asignación de espacio

```
#include <iostream>
using namespace std;

int main() {
   int *x = 0;
   long y = 100000000000000;

   x = new int[y];
   if(x) {
      x[10] = 0;
      cout << "Puntero: " << (void *) x << endl;
      delete[] x;
   } else {
      cout << "Memoria insuficiente." << endl;
   }
   return 0;
}
```

```
WJRG-MacBookPro:Clase13 wladimir$ g++ error2.cpp -o error2
WJRG-MacBookPro:Clase13 wladimir$ ./error2
error2(94913) malloc: *** mmap(size=400000000000000) failed (error code=12)
*** error: can't allocate region
*** set a breakpoint in malloc_error_break to debug
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc
Abort trap
```

Manejo de excepciones

- Pero esto tampoco funcionará, ya que es al procesar la sentencia que contiene el operador **new** cuando se produce la excepción. Sólo nos queda evitar peticiones de memoria que puedan fallar, pero eso no es previsible.
- Sin embargo, C++ proporciona un mecanismo más potente para detectar errores de ejecución: las excepciones.
- Para ello disponemos de tres palabras reservadas extra: **try**, **catch** y **throw**,

Ejemplo usando excepciones

```
#include <iostream>
using namespace std;
int main() {
   int *x;
   long y = 100000000000000;
   try {
      x = new int[y];
      x[0] = 10;
      cout << "Puntero: " << (void *) x << endl;
      delete[] x;
   }
   catch(std::bad_alloc&) {
      cout << "Memoria insuficiente" << endl;
   }
   return 0;
}
```

```
WJRG-MacBookPro:Clase13 wladimir$ g++ error3.cpp -o error3
WJRG-MacBookPro:Clase13 wladimir$ ./error3
error3(95082) malloc: *** mmap(size=400000000000000) failed (error code=12)
*** error: can't allocate region
*** set a breakpoint in malloc_error_break to debug
Memoria insuficiente
```

Manejo de excepciones

- La manipulación de excepciones consiste en transferir la ejecución del programa desde el punto donde se produce la excepción a un manipulador que coincida con el motivo de la excepción.
- Como vemos en este ejemplo, un manipulador consiste en un bloque **try**, donde se incluye el código que puede producir la excepción.
- A continuación encontraremos uno o varios manipuladores asociados al bloque **try**, cada uno de esos manipuladores empiezan con la palabra **catch**, y entre paréntesis una referencia o un objeto.

Manejo de excepciones

- En nuestro ejemplo se trata de una referencia a un objeto *bad_alloc*, que es el asociado a excepciones consecuencia de aplicar el operador **new**.
- También debe existir una expresión **throw**, dentro del bloque **try**. En nuestro caso es implícita, ya que se trata de una excepción estándar, pero podría haber un **throw** explícito.
- El **throw** se comporta como un **return**. Lo que sucede es lo siguiente: el valor *devuelto* por el **throw** se asigna al objeto del **catch** adecuado.

Ejemplo 1: sentencia *throw*

```
#include <iostream>

using namespace std;

int main() {
   try {
      throw 'x'; // valor de tipo char
   }
   catch(char c) {
      cout << "El valor de c es: " << c << endl;
   }
   catch(int n) {
      cout << "El valor de n es: " << n << endl;
   }

   return 0;
}
```

```
WJRG-MacBookPro:Clase13 wladimir$ g++ error4.cpp -o error4
WJRG-MacBookPro:Clase13 wladimir$ ./error4
El valor de c es: x
```

Sentencia *throw*

- En este ejemplo, al tratarse de un carácter, se asigna a la variable 'c', en el **catch** que contiene un parámetro de tipo **char**.

- En el caso del operador **new**, si se produce una excepción, se hace un **throw** de un objeto de la clase *std::bad_alloc*, y como no estamos interesados en ese objeto, sólo usamos el tipo, sin nombre.

Sentencia *throw*

- El manipulador puede ser invocado por un **throw** que se encuentre dentro del bloque **try** asociado, o en una de las funciones llamadas desde él.

- Cuando se produce una excepción se busca un manipulador apropiado en el rango del **try** actual. Si no se encuentra se retrocede al anterior, de modo recursivo, hasta encontrarlo. Cuando se encuentra se destruyen todos los objetos locales en el nivel donde se ha localizado el manipulador, y en todos los niveles por los que hemos pasado.

Ejemplo 2: sentencia *throw*

```
#include <iostream>
using namespace std;
int main() {
   try {
      try {
         try {
            throw 'x'; // valor de tipo char
         }
         catch(int i) {}
         catch(float k) {}
      }
      catch(unsigned int x) {}
   }
   catch(char c) {
      cout << "El valor de c es: " << c << endl;
   }
   return 0;
}
```

```
WJRG-MacBookPro:Clase13 wladimir$ g++ error5.cpp -o error5
WJRG-MacBookPro:Clase13 wladimir$ ./error5
El valor de c es: x
```

Manejo de excepciones

- En este ejemplo podemos comprobar que a pesar de haber hecho el **throw** en el tercer nivel del **try**, el **catch** que lo procesa es el del primer nivel.
- Los tipos de la expresión del **throw** y el especificado en el **catch** deben coincidir, o bien, el tipo del **catch** debe ser una clase base de la expresión del **throw**. La concordancia de tipos es muy estricta, por ejemplo, no se considera como el mismo tipo **int** que **unsigned int**.
- Si no se encontrase ningún **catch** adecuado, se abandona el programa, del mismo modo que si se produce una excepción y no hemos hecho ningún tipo de manipulación de excepciones. Los objetos locales no se destruyen, etc.
- Para evitar eso existe un **catch** general, que captura cualquier **throw** para el que no exista un **catch** concreto:

Ejemplo 3: sentencia *throw*

```
#include <iostream>
using namespace std;

int main() {
   try {
      throw 'x'; //
   }
   catch(int c) {
      cout << "El valor de c es: " << c << endl;
   }
   catch(...) {
      cout << "Excepción imprevista" << endl;
   }
   return 0;
}
```

```
WJRG-MacBookPro:Clase13 wladimir$ g++ error6.cpp -o error6
WJRG-MacBookPro:Clase13 wladimir$ ./error6
Excepción imprevista
```

uc3m | Universidad Carlos III de Madrid