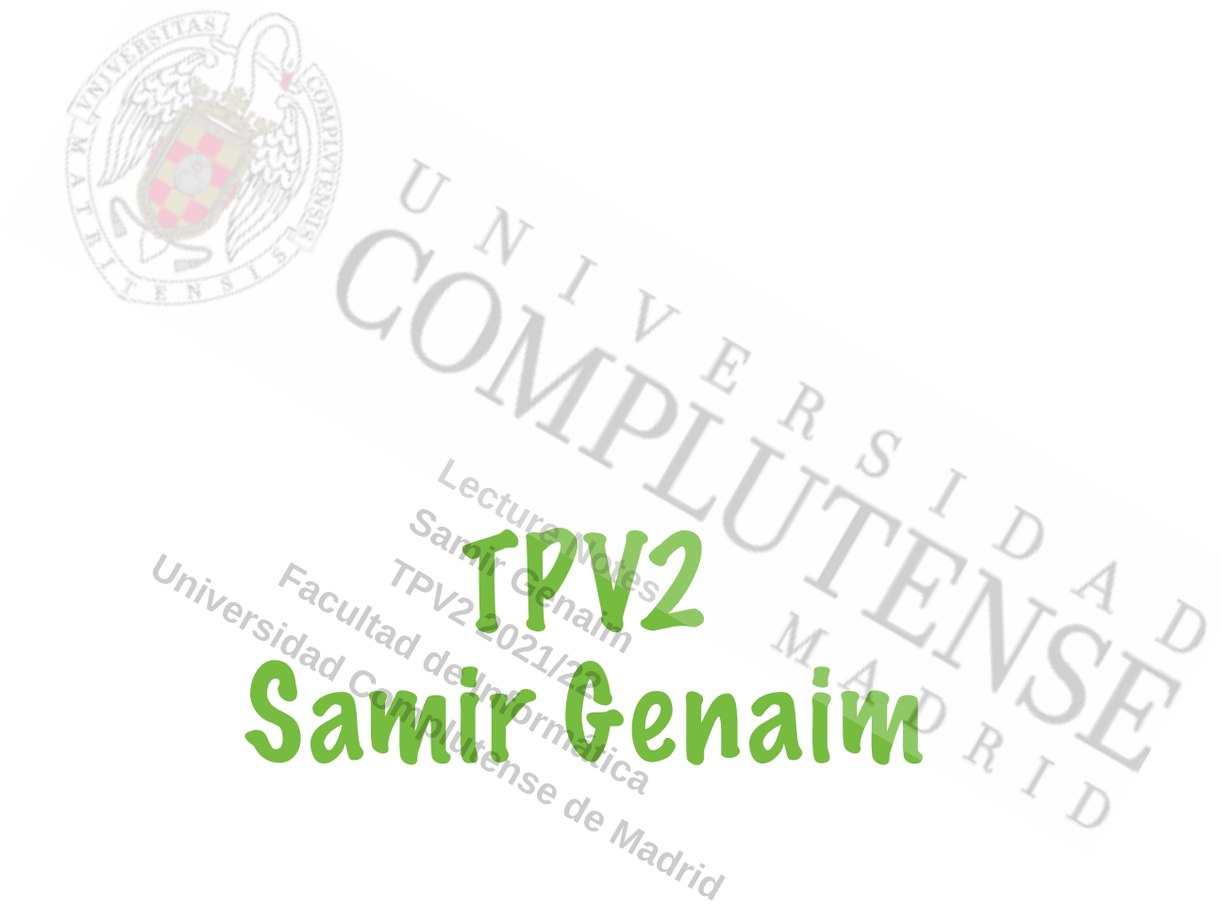
Exceptions en C++



TPV2

Samir Genaim

Circunstancias Excepcionales

- Una circunstancia excepcional es un problema que surge durante la ejecución de un programa, p.ej., intento de dividir por cero, archivo no encontrado, acceso a nullptr, etc.
- Cuando nos encontramos en una circunstancia excepcional, no queremos detener el programa, sino "anunciar" lo que ha ocurrido y tal vez se pueda gestionar en otro lugar del programa (y si nadie lo gestiona el programa se detiene).
- C++ tiene un mecanismo, como en otros lenguajes, que nos permite gestionar circunstancias excepcionales sin complicar el programa (como añadir parámetros que indican errores, valores de salida especiales, etc.)
- Este mecanismo permite transferir el control de una parte del programa (donde ha ocurrido el problema) a otra parte del programa (donde se gestiona este problema). Está basado en la instrucciones (o keywords): try, catch, y throw.

Lanzar una Excepción: throw

Para lanzar una excepción en general se usa la instrucción "throw exp", y lo que se lanza es el resultado de la evaluación de exp (el valor de exp).

```
double division(double a, double b) {
   if (b == 0) {
      throw "Division by zero condition!";
   }
   return a / b;
}
```

Lanza una excepción de tipo "const char*" para indicar que hay un problema que no se puede resolver localmente. La ejecución "normal" se detiene, y empieza una búsqueda de otra parte del código que puede gestionar la excepción lanzada para seguir la ejecución "normal" allí (transferir el control a esa parte)

Capturar una Excepción: try y catch

```cpp
...
try {
    double a = 0.0, b = 0.0, c = 0.0;
    std::cout << "Enter two numbers: ";
    std::cin >> a >> b;
    c = division(a, b);
    std::cout << a << "/" << b << " = " << c << std::endl;
} catch (const char *e) {
    std::cerr << "There was an error: " << e << std::endl;
}
... // A
```

- Si "division" lanza una excepción (de tipo const char *) la ejecución sigue (después del throw) en el bloque de código del catch y después sigue en (A) (no ejecuta el resto del bloque try). En ese bloque, la variable \`e' refiere a la excepción lanzada.
- Si \`division\` no lanza una excepción, después del bloque try la ejecución sigue en (A).

La forma general de try-catch

```
...
try {
  // code-try
} catch (TypeDecl_1 e) {
  // code-catch-1
} catch (TypeDecl_2 e) {
  // code-catch-2
} catch (TypeDecl_3 e) {
  // code-catch-3
} ...
// A
```

- Si la ejecución de "code-try" lanza una excepción, la ejecución sigue (después del throw) en el primer bloque de código "code-catch-i" cuyo TypeDecl_i es compatible con el tipo de la excepción lanzada y después sigue en (A).
- Si el tipo de la excepción no es compatible con ningún TypeDecl_i, la excepción no captura y sigue "hacia fuera" (a lo mejor la captura otro catch, y si no detiene el programa).
- Si "code-try" no lanza una excepción, después del "code-try" la ejecución sigue en (A).

throw lanza una copia

`throw exp`

- Si el valor de "exp" es un objeto, lo que se lanza es una copia de este objeto (usando la constructora de copia correspondiente).

- A veces usa la constructora de movimiento si se garantiza que el objeto no se va a usar más, o si se lo pedimos explícitamente, p.ej., throw std::move(x).

- A veces usa copy elision si no tiene sentido copiarlo, es decir usa el mismo objeto, p.ej., throw A(). Copy elision es una optimización de C++ para evitar copias innecesarias.

- Si dentro de un catch ejecutamos la instrucción throw (sin el parámetro exp), relanzaría la misma excepción que se ha lanzado el throw anterior sin copiarla (puede ser distinto de parámetro e del catch, porque e puede ser una copia).

Capturar todas la excepciones

```
...
try {
   // code-try
} catch (...) {
   // code-catch
}
```

Usando ... en lugar de la declaración de tipo, la cláusula catch correspondiente capturaría todos las excepciones, en este caso no tenemos una referencia a la excepción, pero podemos relanzarla usando throw (sin parámetros) .

La búsqueda del catch adecuado (I)

- La forma más sencilla de entender el mecanismo de búsqueda de la cláusula catch que captura una excepción es usando la pila de llamadas.
- La pila de llamada es una pila de entornos (frames o contexts) donde "viven" las variables locales, cada vez que llamamos a un método añadimos un frame y cuando salimos de ese método quitamos ese frame (llamando a las destructoras de los objetos locales, etc.)
- Suponemos que lo mismo pasa con bloques de código de la forma {...}, porque pueden tener variables locales: cada vez que entramos en un bloque de código añadimos un frame correspondiente y al salir de ese bloque quitamos ese frame.
- Esto pasa en la teoría, en la practica el compilador hace optimizaciones para evitar la creación de frames para cada bloque de código.

La búsqueda del catch adecuado (II)

- Suponemos que el frame del bloque de código de un try tiene anotaciones con las cláusulas catch correspondientes, los tipos que pueden capturar, etc.
- Cuando se lanza una excepción, empezamos a buscar desde la cima de la pila hacia abajo el primer frame que corresponde a un bloque try que tiene un catch compatible con el tipo de la excepción.
- Si encontramos uno compatible, quitamos todos los frames de la pila que están por encima de ese frame (llamando a las destructoras de lo objetos locales, etc), eso se llama stack unwinding, y entramos en el bloque el ese catch
- Si ninguno es compatible, se detiene el programa (a lo mejor sin hacer stack unwinding, depende del compilador)

std::exception

- Hay algunas instructions de C++ que lanzan excepciones para indicar errores en la ejecución, p.ej., dynamic_cast (lanza std::bad_cast), typeid, new, etc.
- Hay métodos en la librerías que lanzan excepciones para indicar errores en la ejecución, p.ej., std::vector::at, std::string::substr, etc.
- Todas esas excepciones heredan de la clase std::exception, que tiene un método what() que devuelve un "const char*" para explicar lo que se ha ocurrido.
- Si en tu programa quieres definir tus propias excepciones, y así el tipo explica mejor el error, es una buena costumbre heredar de std::exception

Resumen

- El mecanismo de excepciones nos ayuda a gestionar circunstancias excepcionales sin complicar el programa, como añadir parámetros que indican errores, valores de salida especiales, etc.
- Hay instrucciones y librerías de C++ que lanzan excepciones, todas heredan de std::exception.
- La búsqueda del catch adecuado afecta al tiempo de ejecución, porque normalmente no se puede resolver durante la compilación.
- Usar el mecanismo de excepciones con moderación, sólo para circunstancias excepcionales y no para todo tipo de error.