Обработка исключительных ситуаций

Исключение — это событие при выполнении программы, которое приводит к её ненормальному или неправильному поведению.
Существует два вида исключений:

- *Аппаратные* (структурные, SE-Structured Exception), которые генерируются процессором. К ним относятся, например,
  - деление на 0;
  - выход за границы массива;
  - обращение к невыделенной памяти;
  - переполнение разрядной сетки.
- *Программные*, генерируемые операционной системой и прикладными программами – возникают тогда, когда программа их явно инициирует. Когда встречается аномальная ситуация, та часть программы, которая ее обнаружила, может сгенерировать, или *возбудить*, исключение.

Механизм структурной обработки исключений позволяет однотипно обрабатывать как программные, так и аппаратные исключения.

Обработка программных исключений

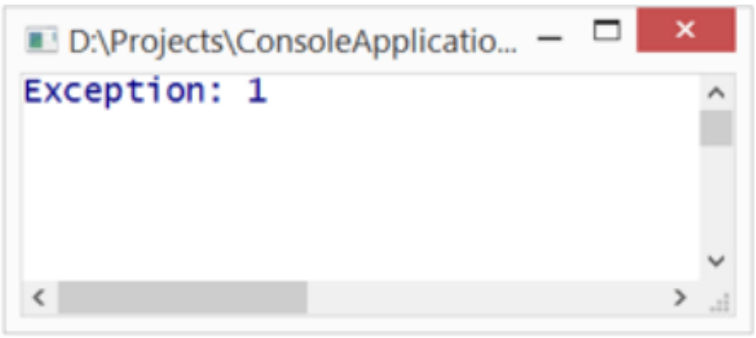
Фундаментальная идея *обработки исключительных ситуаций* состоит в том, что функция, обнаружившая проблему, но не знающая как её решить, *генерирует* исключение в надежде, что вызвавшая её (непосредственно или косвенно) функция сможет решить возникшую проблему. Функция, которая может решать проблемы данного типа, указывает, что она *перехватывает* такие исключения.
Для реализации обработки исключений в C++ используйте выражения `try`, `throw` и `catch`.
Блок `try` {…} позволяет включить один или несколько операторов, которые могут создавать исключение.
Выражение `throw` используется только в программных исключениях и означает, что исключительное условие произошло в блоке `try`. В качестве операнда выражения `throw` можно использовать объект любого типа. Обычно этот объект используется для передачи информации об ошибке.
Для обработки исключений, которые могут быть созданы, необходимо реализовать один или несколько блоков `catch` сразу после блока `try`. Каждый блок `catch` указывает тип исключения, которое он может обрабатывать.
Сразу за блоком `try` находится *защищенный раздел кода*. Выражение `throw` вызывает исключение, т.е. создает его.
Блок кода после `catch` является *обработчиком исключения*. Он перехватывает исключение, вызываемое, если типы в выражениях `throw` и `catch` совместимы. Если оператор `catch` задает многоточие (…) вместо типа, блок `catch` обрабатывает все типы исключений.
Поскольку блоки `catch` обрабатываются в порядке программы для поиска подходящего типа, обработчик с многоточием должен быть последним обработчиком для соответствующего блока `try`. Как правило, блок `catch(…)` используется для ведения журнала ошибок и выполнения специальной очистки перед остановкой выполнения программы.

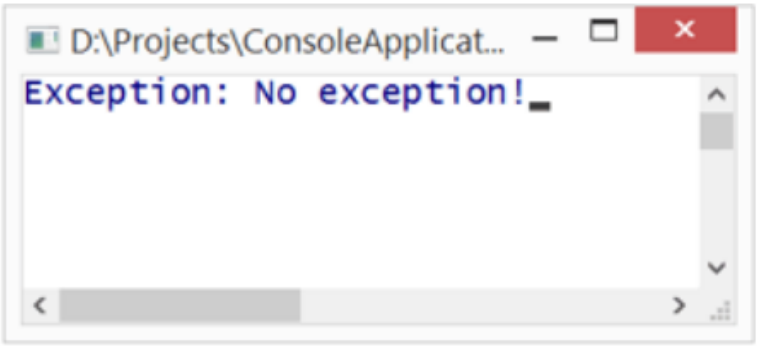
```
try { … // защищенный раздел кода
  throw параметр;
}
catch (параметр) { // обработка исключения }
catch (…) { // обработка остальных исключений }
```

Ниже приведен пример обработки программного исключения. В реальных программах посылка исключения командой `throw`, как правило, является следствием проверки какого-либо условия.

```cpp
#include <iostream>
using namespace std;
int main() {
  try {
    cout << «Exception: «;
    throw 1;
    cout << «No exception!»;
  } catch (int a) {
    cout << a;
  }
cin.get(); return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int main() {
  try {
    cout << «Exception: «;
    //throw 1;
    cout << «No exception!»;
  } catch (int a) {
    cout << a;
  }
cin.get(); return 0;
}
```



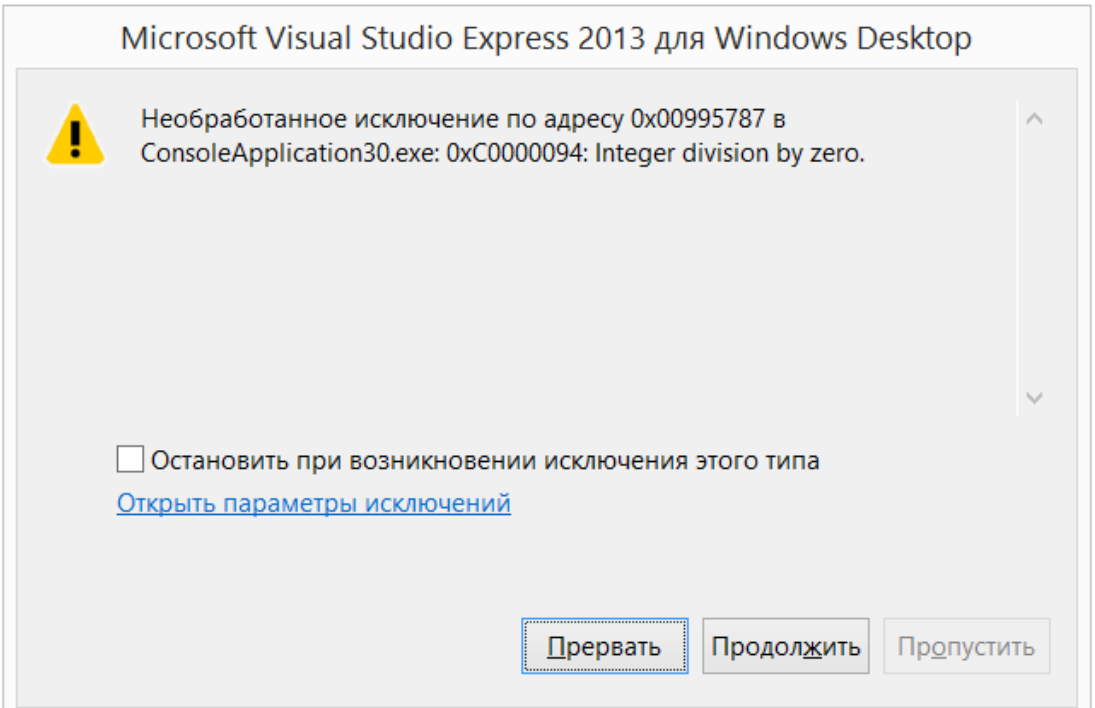


Обработка структурных исключений

Рассмотрим пример программы, генерирующей исключительную ситуацию «деление на 0».

```cpp
#include <iostream>
using namespace std;
int main() {
  int a = 0, b =10;
  cout << b/a << endl;cin.get();
  return 0;
}
```

При попытке запустить программу на выполнение видим следующее:



Для обработки исключительной ситуации необходимо операцию деления поместить в блок защищенного кода:

```
#include <iostream>
using namespace std;
int main() {
  int a = 0, b = 10;
  try {
    cout << b / a << endl;
  }
  catch (…)
  {
    cout << «error»;
  }
  cin.get();
  return 0;
}
```