УНИВЕРСИТЕТ СИНЕРГИЯ

РАЗРАБОТКА НА C++

УРОК 8. Обработка исключений.

Для чего нужен механизм исключений?

Исключения предпочтительнее использовать в современном C++ по следующим причинам:

- Исключение заставляет вызывающий код распознавать условие ошибки и обрабатывать его. Необработанное исключение останавливает выполнение программы.

- Исключение переходит к точке стека вызовов, которая может обрабатывать ошибку. Промежуточные функции могут позволить распространению исключения. Они не должны координироваться с другими слоями.

- Механизм очистки стека исключений уничтожает все объекты в области после создания исключения в соответствии с четко определенными правилами.

- Исключение позволяет четко разделить код, который обнаруживает ошибку и код, обрабатывающий ошибку.

Типы исключений

runtime_error

range_error

overflow_error

underflow_error

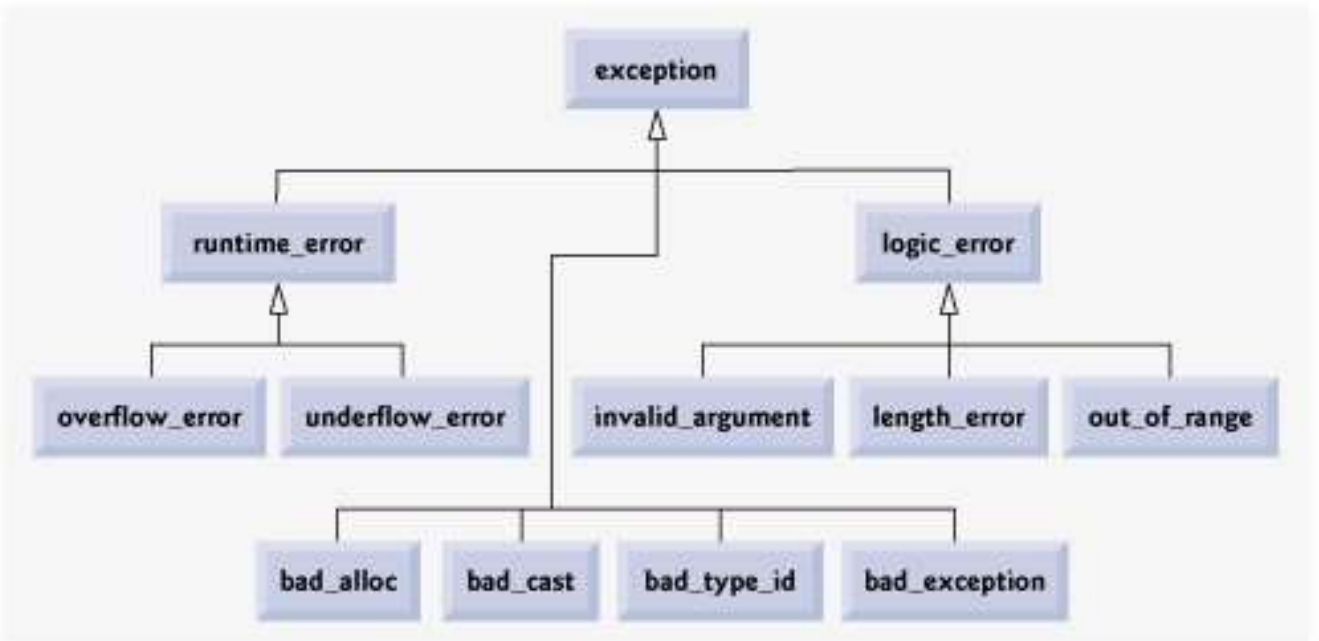
logic_error

domain_error

invalid_argument

length_error

out_of_range



Знакомство с механизмом исключений. Программа №1.


```
#include <iostream>

double divide(int a, int b)
{
    if (b != 0)
        return a / b;
    else
        std::cout << "Error! b must not be
equal to 0" << std::endl;
}

int main()
{
    int x = 500;
    int y = 0;
    double z = divide(x, y);

    std::cout << z << std::endl;
    std::cout << "The End..." <<
std::endl;
    return 0;
}

double divide(int a, int b)
{
    return a / b;
}
```

Генерация исключений

Для оповещения о возникновении исключения или ошибки используется оператор **throw**.

Оповещение о том, что произошло исключение, называется генерацией исключения (или «выбрасыванием» исключения).

Оператор **throw** генерирует исключение.

Через оператор **throw** можно передать информацию об ошибке.

```
#include <iostream>

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

int main()
{
    int x = 500;
    int y = 0;
    double z = divide(x, y);

    std::cout << z << std::endl;
    std::cout << "The End..." << std::endl;
    return 0;
}

double divide(int a, int b)
{
    return a / b;
}
```

Поиск исключений

Для поиска исключений используется блок, который определяется ключевым словом **try**. Блок «try» действует как наблюдатель в поисках исключений, которые были выброшены оператором **throw** или функцией в этом же блоке.

Блок «**try**» не определяет, как обрабатывать исключение. Он просто сообщает компилятору: «Если какой-либо из стейтментов внутри этого блока «try» сгенерирует исключение — поймай его!». Следовательно, после блока поиска исключений должен быть описан их обработчик.

Обработка исключений

Для обработки исключений используется блок, который определяется ключевым словом **catch**.

```
catch (...)
   {
      cout << "На ноль делить нельзя!";
   }
```

Блоки «try» и «catch» работают вместе. Блок «try» обнаруживает любые исключения, которые выброшены в нем, и направляет их в соответствующий блок «catch» для обработки. Блок «try» должен иметь, по крайней мере, один блок «catch», и он должен находиться сразу же за ним. Блоков «catch» может быть несколько, в таком случае они должны быть размещены друг за другом.

Как только исключение будет поймано блоком «try» и обработано блоком «catch», выполнение программы возобновится.

trow, try и catch вместе

```
try
   {
      if (a == 0)
         throw 0;

      cout << "Частное чисел: " <<
divide(a, b);
   }
catch (int power)
   {
      cout <<
"Программа завершена.";
   }
catch (const char* err)
   {
      cout <<
"На ноль делить нельзя!";
   }
```

```
double divide(float a,
float b)
{
   if (b == 0)
      throw "Деление
на ноль";
   return a / b;
}
```