**Новосибирский государственный технический университет**

Лабораторная работа № 6 по дисциплине программирование

Тема: Обработка исключительных ситуаций

|  |  |
| --- | --- |
| Группа: АВТ-907  Студент: Смоленский И.В. | Преподаватель: Булатов А. Д. |

Новосибирск, 2020

**Задание**:

Вариант 14.

Добавить в классы и демонстрационную программу обработку исключений при возникновении ошибок: недостатка памяти, выход за пределы диапазона допустимых значений, деление на ноль и т.д. Изменить основную программу так, чтобы она демонстрировала обработку исключений.

Теоретические сведения:

Цель при использовании исключительных ситуаций C++ состоит в упрощении обнаружения и обработки ошибок в программах. В идеале, если ваши программы обнаруживают неожиданную ошибку (исключительную ситуацию), им следует разумным образом ее обработать вместо того, чтобы просто прекратить выполнение.

## Проверка исключительной ситуации

Прежде чем ваши программы могут обнаружить и отреагировать на исключительную ситуацию, вам следует использовать оператор C++ **try** для разрешения обнаружения исключительной ситуации. Например, следующий оператор try разрешает обнаружение исключительной ситуации для вызова функции file\_соpy:

try {  
    file\_copy("source.тхт", "target.тхт") ;   
};

Сразу же за оператором try программа должна разместить один или несколько операторов **catch**, чтобы определить, какая исключительная ситуация имела место (если она вообще была):

catch (file\_open\_error) {  
    cerr << "Ошибка открытия исходного или целевого файла" << endl;  
    exit(1);  
}

catch (file\_read\_error) {  
    cerr << "Ошибка чтения исходного файла" << endl;  
    exit(1);  
}

catch (file\_write\_error) {  
    cerr << "Ошибка записи целевого файла" << endl;  
    exit(1);  
}

Как видите, приведенный код проверяет возникновение исключительных ситуаций работы с файлами, определенных ранее. В данном случае независимо от типа ошибки код просто выводит сообщение и завершает программу. Если вызов функции прошел успешно и исключительная ситуация не выявлена, C++ просто игнорирует операторы catch.

Существует 3 формы записи catch

catch ( тип имя) { //тело обработчика }

catch ( тип ) { //тело обработчика }

catch ( …) { //тело обработчика }

**Генерация исключительной ситуации**

Сам C++ не генерирует исключительные ситуации. Их генерируют ваши программы, используя оператор C++ **throw**. Например, внутри функции file\_copy программа может проверить условие возникновения ошибки и сгенерировать исключительную ситуацию:

unsigned Bankomat::loadMoney(unsigned sumLoad)

{

try

{

if (sumLoad < 0)

{

throw std::invalid\_argument("Invalid arg");

}

}

catch (const std::invalid\_argument & err)

{

std::cerr << " Invalid argument: " << err.what() << std::endl;

}

sumBank += sumLoad;

return sumBank;

}

Как видите, программа использует оператор throw для генерации определенных исключительных ситуаций.

## Обработчик исключительной ситуации

Когда C++ встречает оператор throw, он активизирует соответствующий обработчик исключительной ситуации (функцию, чьи операторы вы определили в классе исключительной ситуации). Используя операторы catch, ваша программа может определить, какая именно исключительная ситуация возникла, и отреагировать соответствующим образом.

unsigned Bankomat::takeMoney(unsigned sumTake)

{

try

{

if (sumTake < 0)

{

throw std::invalid\_argument("Invalid arg");

}

}

catch (const std::invalid\_argument & err)

{

std::cerr << " Invalid argument: " << err.what() << std::endl;

}

if ((sumBank >= sumTake) && (sumTake <= sumMaxGet))

{

sumBank -= sumTake;

}

return sumBank;

}

Если вы проследите исходный код, который генерирует каждое из сообщений, то сможете убедиться, что поток управления при возникновении исключительной ситуации проходит в обработчик исключительной ситуации и обратно к оператору catch. Так, первая строка вывода генерируется обработчиком исключительной ситуации, т.е. функцией messdown. Вторая строка вывода генерируется в операторе catch, который обнаружил исключительную ситуацию.

Реализация обработки исключений:

Исключения всегда выбрасывались при записывании в знаменатель 0, при попытке деления на 0 и при переполнении памяти, выделяемой под переменную, которая хранит данные о сумме, умножении или вычитании дробей.

Все методы, в которых выбрасывается и обрабатывается исключение:

unsigned Bankomat::loadMoney(unsigned sumLoad)

{

try

{

if (sumLoad < 0)

{

throw std::invalid\_argument("Invalid arg");

}

}

catch (const std::invalid\_argument & err)

{

std::cerr << " Invalid argument: " << err.what() << std::endl;

}

sumBank += sumLoad;

return sumBank;

}

unsigned Bankomat::takeMoney(unsigned sumTake)

{

try

{

if (sumTake < 0)

{

throw std::invalid\_argument("Invalid arg");

}

}

catch (const std::invalid\_argument & err)

{

std::cerr << " Invalid argument: " << err.what() << std::endl;

}

if ((sumBank >= sumTake) && (sumTake <= sumMaxGet))

{

sumBank -= sumTake;

}

return sumBank;

}

**Вывод:**   
  
Мной были обработаны и изучены исключения, которые могут возникнуть в результате работы программы.