Министерство образования и науки РФ
ФГБОУ ВО ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра «Информационная безопасность систем и технологий»

ОТЧЕТ

о лабораторной работе №2

# ОБРАБОТКА ОШИБОК

Дисциплина: Технологии и методы программирования
Группа: 18ПИ1
Выполнил: Нестеров И.С.
Количество баллов:
Дата сдачи:
Принял: к.т.н., доцент Лупанов М. Ю.

Пенза 2019

1 Цель работы

1.1 Освоить процесс обработки ошибок в программах на основе механизма исключений.

2 Задание к лабораторной работе

2.1 Добавить к модулю шифрования методом Гронсвельда русскоязычных сообщений, разработанному при выполнении предыдущей работы, обработку исключений.

2.2 . Добавить к модулю шифрования методом маршрутной перестановки, разработанной при выполнении предыдущей работы, обработку исключений.

3 Результаты работы

3.1 Изменение в файле modAlphaCipher.h представлено на Рисунке 1. Первое исключение, которое нужно проверить это ввод ключа. Конструктор принимает на вход строку символов, которая является ключом. Эта строка должна содержать символы русского алфавита в верхнем регистре. Ошибочным будет использование символов в нижнем регистре, а также символов, не являющихся буквами русского алфавита. Кроме того, недопустима ситуация, когда в качестве ключа передается пустая строка. Поэтому создаем класс-исключение cipher_error как производный от класса std::invalid_argument. В классе добавляем новый метод, который будет проверять ключ на валидности и переводить в нем символы в верхний регистр, если это необходимо. Назовем этот метод getValidKey(). Затем нужно проверить открытый текст, состоящий только из прописных букв. Для строчных букв выполним их преобразование в прописные. А вот символы, которые не являются буквами русского алфавита, просто удалим из открытого текста, не возбуждая исключения. Таким образом, возможна только одна ситуация, приводящая к исключению — пустой открытый текст. В классе добавляем новый метод getValidOpenText(), который будет проверять открытый текст. Для проверки зашифрованной строки исключения должны возбуждаться в двух случаях: когда получена пустая строка

и когда в зашифрованном тексте есть символы, отличные от прописных русских букв. Зашифрованная строка будет проверяться в методе getValidOpenText(). Полный текст модифицированной программы представлен в Приложении А.

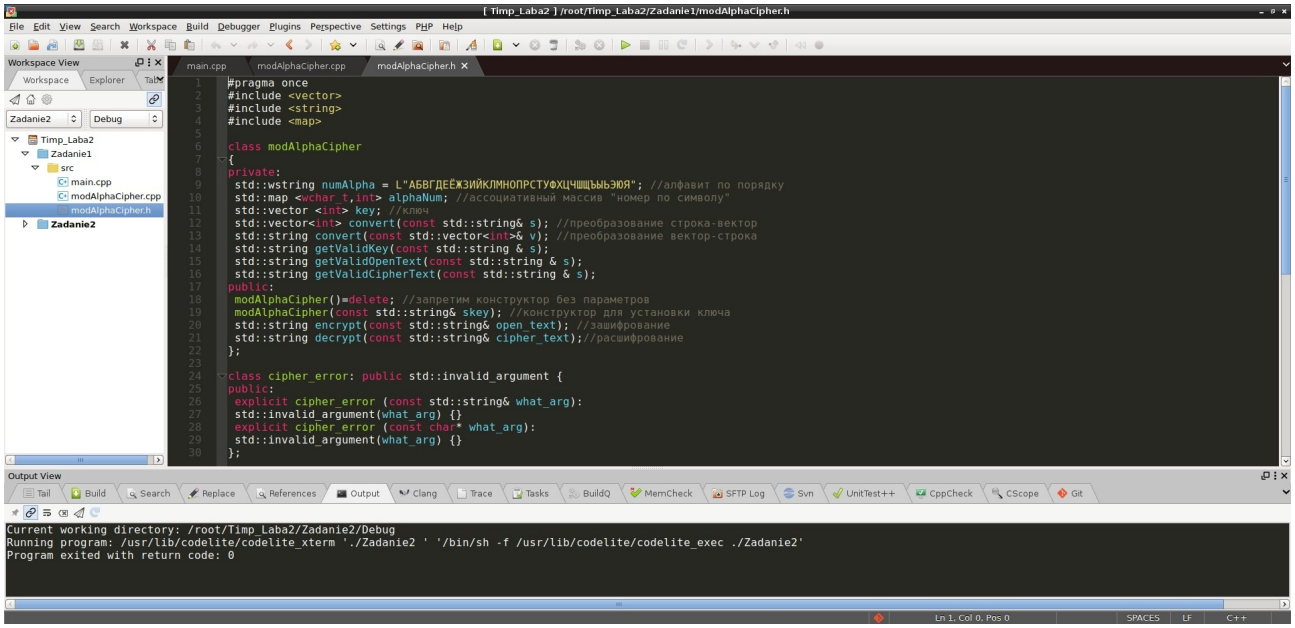


Рисунок 1 - файл modAlphaCipher.h

3.2 Файл swapcipher.h представлен на Рисунке 2. Первое исключение, которое нужно проверить это ввод ключа. Конструктор принимает на вход число, которое является количеством столбцов матрицы. Если вводимое число меньше или равно нулю, то будет возбуждено исключение. Поэтому создаем класс-исключение cipher_error как производный от класса std::invalid_argument. В классе добавляем новый метод, который будет проверять ключ на валидности и переводить в нем символы в верхний регистр, если это необходимо. Назовем этот метод getValidKey(). Затем нужно проверить открытый текст, состоящий только из прописных букв. Для строчных букв выполним их преобразование в прописные. А вот символы, которые не являются буквами русского алфавита, просто удалим из открытого текста, не возбуждая исключения. Таким образом, возможна только одна ситуация, приводящая к исключению — пустой открытый текст. В классе добавляем новый метод getValidOpenText(), который будет проверять открытый текст. Для проверки зашифрованной строки

исключения должны возбуждаться в двух случаях: когда получена пустая строка и когда в зашифрованном тексте есть символы, отличные от прописных русских букв. Зашифрованная строка будет проверяться в методе getValidOpenText(). Полный текст модифицированной программы представлен в Приложении Б.

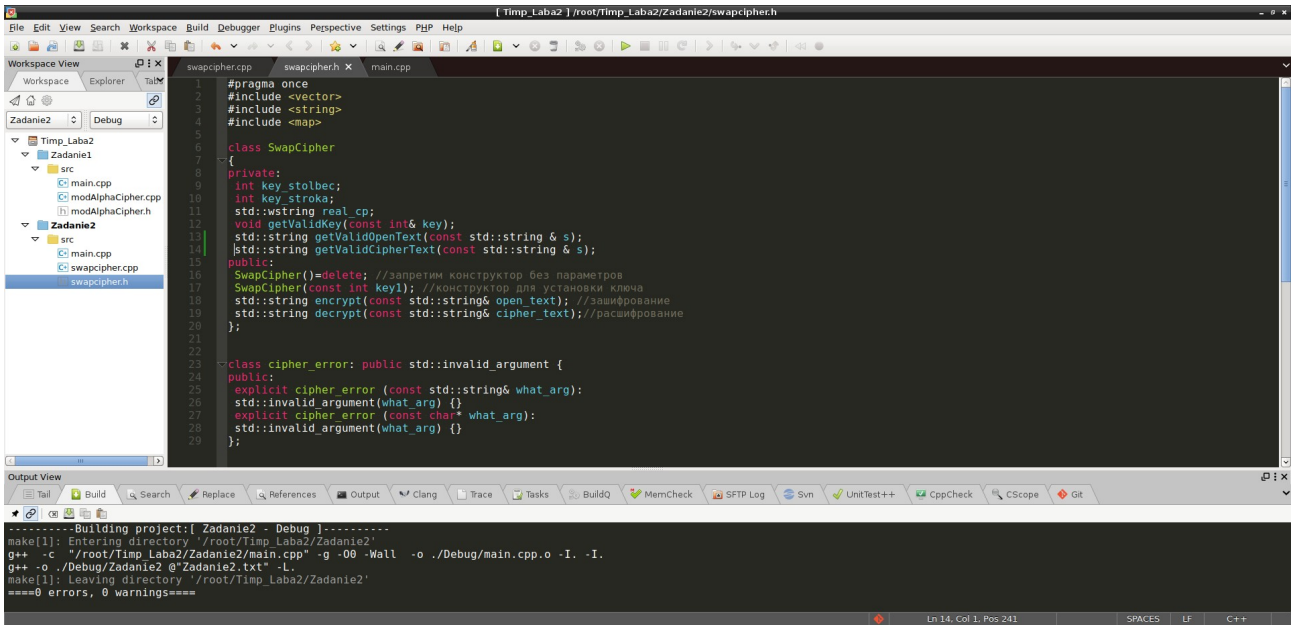


Рисунок 2 - файл swapcipher.h

4 Вывод

В результате выполнения работы был изучен обработчик ошибок языка Си++, а также была добавлена обработка ошибок к многомодульному проекту маршрутной перестановки, и получены практические навыки в создании класса-исключения.

## Приложение А

## Текст модифицированной программы

```
#include "modAlphaCipher.h"
#include <locale>
#include <codecvt>

std::locale loc("ru_RU.UTF-8");
std::wstring_convert<std::codecvt_utf8<wchar_t>, wchar_t> codec;
using namespace std;
modAlphaCipher::modAlphaCipher(const std::string& skey)
{
 for (unsigned i=0; i < numAlpha.size(); i++)
    alphaNum[numAlpha[i]]=i;
 key = convert(getValidKey(skey));
}

std::string modAlphaCipher::encrypt(const std::string& open_text)
{
 std::vector<int> work = convert(getValidOpenText(open_text));
 for(unsigned i=0; i < work.size(); i++)
    work[i] = (work[i] + key[i % key.size()]) % alphaNum.size();
 return convert(work);
}

std::string       modAlphaCipher::decrypt(const       std::string&
cipher_text)
{
 std::vector<int> work = convert(getValidCipherText(cipher_text));
 for(unsigned i=0; i < work.size(); i++)
     work[i] = (work[i] + alphaNum.size() - key[i % key.size()]) %
alphaNum.size();
 return convert(work);
}
```

```
inline std::vector<int> modAlphaCipher::convert(const std::string&
s)
{
 std::wstring ws = codec.from_bytes(s); // перекодируем из UTF-8 в
UTF-32
 std::vector<int> result;
 for(auto c:ws)
    result.push_back(alphaNum[c]);
 return result;
}

inline std::string modAlphaCipher::convert(const std::vector<int>&
v)
{
 std::string result;
  std::wstring ws = codec.from_bytes(result); // перекодируем из
UTF-8 в UTF-32
 for(auto i:v)
    ws.push_back(numAlpha[i]);
 result = codec.to_bytes(ws);
 return result;
}

inline std::string modAlphaCipher::getValidKey(const std::string &
s)
{
 std::string result;
 std::wstring ws = codec.from_bytes(s);
 if (ws.empty())
 throw cipher_error("Empty key");
 std::wstring tmp(ws);
 for (auto & c:tmp) {
```

```
 if (!isalpha(c, loc)) //((c < L'A' && c > L'Я') || (c < L'a' && c
> L'я')) - почему это условие не работает???
 throw cipher_error(std::string("Invalid key ") + s);
 if (islower(c, loc)) //(c >= L'a' && c <= L'я')
 c = toupper(c, loc);
 }
 result = codec.to_bytes(tmp);
 return result;
}

inline      std::string      modAlphaCipher::getValidOpenText(const
std::string & s)
{
 std::string result;
 std::wstring tmp;
 std::wstring ws = codec.from_bytes(s);
 for (auto c:ws) {
 if (isalpha(c, loc)) {
 if (islower(c, loc))
 tmp.push_back(toupper(c, loc));
 else
 tmp.push_back(c);
 }
 }
 if (tmp.empty())
 throw cipher_error("Empty open text");
 result = codec.to_bytes(tmp);
 return result;
}

inline     std::string     modAlphaCipher::getValidCipherText(const
std::string & s)
{
```

```
 std::string result;
 std::wstring ws = codec.from_bytes(s);
 if (ws.empty())
 throw cipher_error("Empty cipher text");
 for (auto c:ws) {
 if (!isupper(c, loc))
 throw cipher_error(std::string("Invalid cipher text ")+s);
 }
 result = codec.to_bytes(ws);
 return result;
}
```

## Приложение Б

## Текст модифицированной программы табличной маршрутной перестановки

```
#include "swapcipher.h"
#include <locale>
#include <codecvt>
std::locale loc("ru_RU.UTF-8");
std::wstring_convert<std::codecvt_utf8<wchar_t>, wchar_t> codec;

SwapCipher::SwapCipher(const int key1)
{
    key_stolbec = key1;
    getValidKey(key_stolbec);
}

std::string SwapCipher::encrypt(const std::string& open_text)
{
    std::string result;
                                    std::wstring        ws        =
codec.from_bytes(getValidOpenText(open_text));
    int i = 0, j = 0;
    key_stroka = ((ws.length() - 1) / key_stolbec) + 1;
    while(ws.length() % key_stroka != 0)
        ws += L'*';
    for (i = key_stolbec - 1; i >= 0; i--)
             for (j = i; j < key_stolbec*key_stroka; j = j +
key_stolbec)
            real_cp += ws[j];
    ws = L"";
    for (i = 0; i < key_stolbec*key_stroka; i++)
          if (((real_cp[i] >= L'A') && (real_cp[i] <= L'Я')) ||
real_cp[i] == L'Ё')
              ws += real_cp[i];
```

```
    result = codec.to_bytes(ws);
    return result;
}

std::string SwapCipher::decrypt(const std::string& cipher_text)
{
    getValidCipherText(cipher_text);
    std::string result;
    std::wstring ws;
    std::wstring ciphertext;
    int i = 0, j = 0;
    for (i = key_stroka; i > 0; i--)
         for (j = (key_stroka * key_stolbec) - i; j >= 0; j = j -
key_stroka)
            ciphertext += real_cp[j];
    for (i = 0; i < key_stolbec*key_stroka; i++)
          if (((ciphertext[i] >= L'A') && (ciphertext[i] <= L'Я'))
|| ciphertext[i] == L'Ё')
               ws += ciphertext[i];
    result = codec.to_bytes(ws);
    return result;
}

void SwapCipher::getValidKey(const int & key)
{
 if (key <= 0)
     throw cipher_error("Empty key");
}

inline  std::string  SwapCipher::getValidOpenText(const  std::string
& s)
{
 std::string result;
```

```
 std::wstring tmp;
 std::wstring ws = codec.from_bytes(s);
 for (auto c:ws) {
 if (isalpha(c, loc)) {
 if (islower(c, loc))
 tmp.push_back(toupper(c, loc));
 else
 tmp.push_back(c);
 }
 }
 if (tmp.empty())
 throw cipher_error("Empty open text");
 result = codec.to_bytes(tmp);
 return result;
}

inline       std::string       SwapCipher::getValidCipherText(const
std::string & s)
{
 std::string result;
 std::wstring ws = codec.from_bytes(s);
 if (ws.empty())
 throw cipher_error("Empty cipher text");
 for (auto c:ws) {
 if (!isupper(c, loc))
 throw cipher_error(std::string("Invalid cipher text ")+s);
 }
 result = codec.to_bytes(ws);
 return result;
}
```