МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Информационная безопасность систем и технологий»

Отчет

о лабораторной работе №3

на тему «Модульное тестирование»

Дисциплина: Методы и средства программирования

Группа: 22ПИ2

Выполнил: Никитина М. А.

Количество баллов:

Дата сдачи:

Принял: Н. А. Сидоров

2023

1 Цель работы: освоить процесс модульного тестирования разрабатываемых программ.

2 Задание на лабораторную работу

2.1 Адаптировать приведенные тестовые сценарии к модулю шифрования сообщений методом Гронсвельда, разработанному при выполнении предыдущих работ и выполнить модульное тестирование.

2.2 Разработать тестовые сценарии для модуля шифрования методом маршрутной перестановки, разработанного при выполнении предыдущих работ.

2.3 Разработать модульные тесты и провести тестирование модуля шифрования методом маршрутной перестановки.

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

3 Порядок выполнения работы

3.1 Был адаптирован приведенный тестовый сценарии к модулю шифрования сообщений методом Гронсвельда. Результат представлен на рисунке 1. Код программы представлен в приложении А.

```
stud@virtdeb:~/C++Projects/UnitTest++/Debug$ ./UnitTest++
/home/stud/C++Projects/UnitTest++/main.cpp:64:1: error: Failure in MaxShiftKey:
Expected SGDPTHBJAQNVMENWITLORNUDQSGDKZYXCNF but was SGDPTHBJAQNVMENWTLORNUDQSGD
KZYXCNF
/home/stud/C++Projects/UnitTest++/main.cpp:88:1: error: Failure in MaxShiftKey:
Expected THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG but was THEQUICKBROWNFOXUMPSOVERTHE
LAZYDOG
FAILURE: 2 out of 22 tests failed (2 failures).
Test time: 0.00 seconds.
```

Рисунок 1 - Тестирование метода Гронсвельда

3.2 Были разработаны тестовые сценарии для модуля шифрования методом маршрутной перестановки, разработанного при выполнении предыдущих работ. Результат представлен в таблицах 1-2.

3.3 Были разработаны модульные тесты и проведено тестирование модуля шифрования методом маршрутной перестановки. Результат представлен на рисунке 2. Код программы представлен в приложении Б.

Таблица 1 - Тестовый сценарий для ключа

№	Тест	Исходный текст	Ключ	Зашифрованный текст	Расшифрованный текст
1	Верный ключ	privet	2	rvtpie	privet
2	Ключ длиннее сообщения	hello	10	Искл.	Искл.
3	Ключ меньше или равен 1	hello	1	Искл.	Искл.

Таблица 2 - Тестовый сценарий для исходного текста

№	Тест	Ключ	Исходный текст	Зашифрованный текст	Расшифрованный текст
1	Строка из прописных	2	PRIVET	RVTPIE	PRIVET
2	Строка из строчных	2	privet	rvtpie	privet
3	Строка, содержащая символы, отличные от букв	2	pri12!	Искл.	Искл.
4	Пустой текст	2	Пустая строка	Искл.	Искл.
5	Нет букв	2	123!@#	Искл.	Искл.

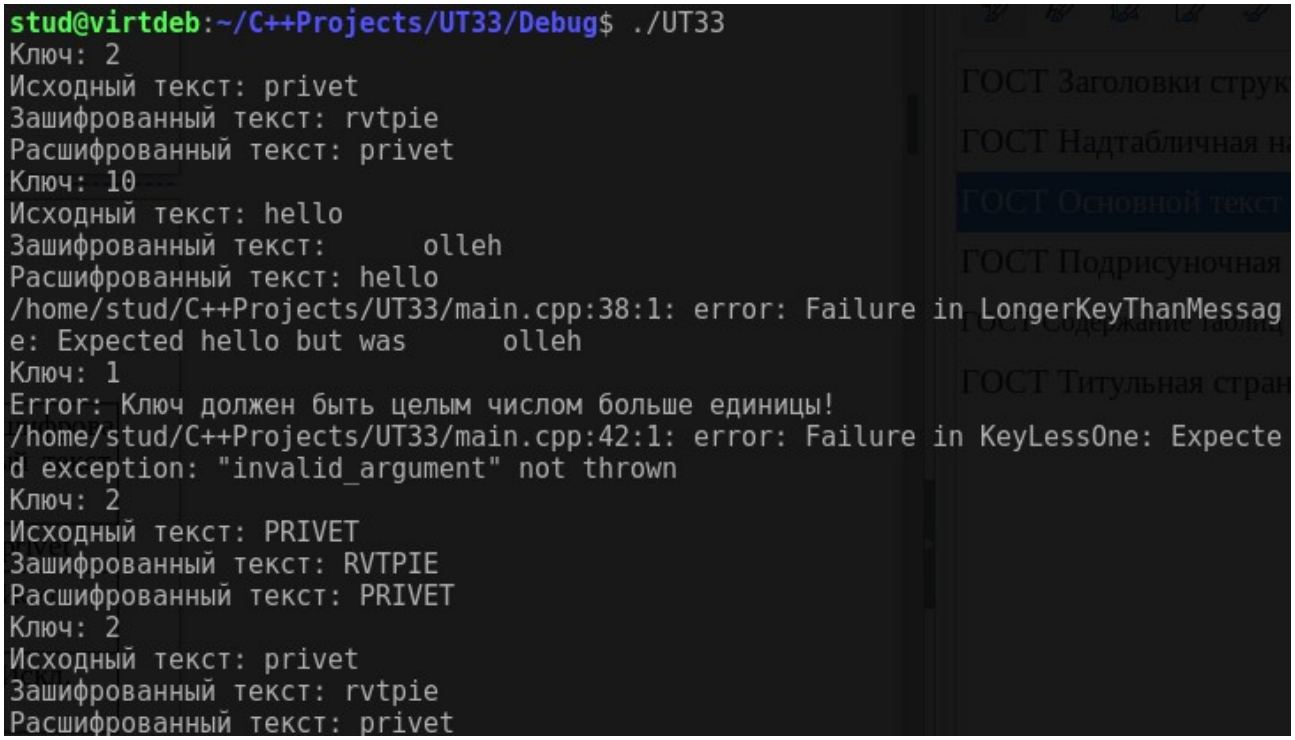

Рисунок 2 - Тест маршрутной перестановки

4 Выводы

С помощью данной лабораторной работы был освоен процесс модульного тестирования разрабатываемых программ. Полный результат работы можно посмотреть по ссылке: https://github.com/Popitka994/MiSP/blob/main/Lb3.

Приложение А

Код модуля main.cpp:

```
#include <iostream>
#include "modAlphaCipher.h"
#include <UnitTest++/UnitTest++.h>
SUITE(KeyTest)
{
    TEST(ValidKey) {
        CHECK_EQUAL("BCDBC",modAlphaCipher("BCD").encrypt
("AAAAA"));
    }
    TEST(LongKey) {
        CHECK_EQUAL("BCDEF",modAlphaCipher("BCDEFGHIJK").
encrypt("AAAAA"));
    }
    TEST(LowCaseKey) {
        CHECK_EQUAL("BCDBC",modAlphaCipher("bcd").encrypt
("AAAAA"));
    }
    TEST(DigitsInKey) {
                                CHECK_THROW(modAlphaCipher
cp("B1"),cipher_error);
    }
    TEST(PunctuationInKey) {
                                CHECK_THROW(modAlphaCipher
cp("B,C"),cipher_error);
    }
    TEST(WhitespaceInKey) {
```

```
                        CHECK_THROW(modAlphaCipher   cp("B
C"),cipher_error);
    }
    TEST(EmptyKey) {
        CHECK_THROW(modAlphaCipher cp(""),cipher_error);
    }
    TEST(WeakKey) {
                                CHECK_THROW(modAlphaCipher
cp("AAA"),cipher_error);
    }
}
struct KeyB_fixture {
    modAlphaCipher * p;
    KeyB_fixture()
    {
        p = new modAlphaCipher("B");
    }
    ~KeyB_fixture()
    {
        delete p;
    }
};

SUITE(EncryptTest)
{
    TEST_FIXTURE(KeyB_fixture, UpCaseString) {
        CHECK_EQUAL("UIFRVJDLCSPXOGPYKVNQTPWFSUIFMBAZEPH"
, p->encrypt("THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG"));
    }
```

```
    TEST_FIXTURE(KeyB_fixture, LowCaseString) {
        CHECK_EQUAL("UIFRVJDLCSPXOGPYKVNQTPWFSUIFMBAZEPH"
,p->encrypt("thequickbrownfoxjumpsoverthelazydog"));
    }
                                TEST_FIXTURE(KeyB_fixture,
StringWithWhitspaceAndPunct) {
        CHECK_EQUAL("UIFRVJDLCSPXOGPYKVNQTPWFSUIFMBAZEPH"
,p->encrypt("THE QUICK BROWN FOX JUMPS OVER THE LAZY
DOG!!!"));
    }
    TEST_FIXTURE(KeyB_fixture, StringWithNumbers) {
        CHECK_EQUAL("IBQQZOFXZFBS", p->encrypt("Happy New
2019 Year"));
    }
    TEST_FIXTURE(KeyB_fixture, EmptyString) {
        CHECK_THROW(p->encrypt(""),cipher_error);
    }
    TEST_FIXTURE(KeyB_fixture, NoAlphaString) {
                                            CHECK_THROW(p-
>encrypt("1234+8765=9999"),cipher_error);
    }
    TEST(MaxShiftKey) {
        CHECK_EQUAL("SGDPTHBJAQNVMENWITLORNUDQSGDKZYXCNF"
,modAlphaCipher("Z").encrypt("THEQUICKBROWNFOXUMPSOVERTHE
LAZYDOG"));
    }
}
SUITE(DecryptText)
{
```

```
    TEST_FIXTURE(KeyB_fixture, UpCaseString) {
        CHECK_EQUAL("THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG"
,p->decrypt("UIFRVJDLCSPXOGPYKVNQTPWFSUIFMBAZEPH"));
    }
    TEST_FIXTURE(KeyB_fixture, LowCaseString) {
                                            CHECK_THROW(p-
>decrypt("uifRVJDLCSPXOGPYKVNQTPWFSUIFMBAZEPH"),cipher_er
ror);
    }
    TEST_FIXTURE(KeyB_fixture, WhitespaceString) {
        CHECK_THROW(p->decrypt("UIF RVJDL CSPXO GPY KVNQT
PWFS UIFMBAZ EPH"),cipher_error);
    }
    TEST_FIXTURE(KeyB_fixture, DigitsString) {
                                            CHECK_THROW(p-
>decrypt("IBQQZOFX2019ZFBS"),cipher_error);
    }
    TEST_FIXTURE(KeyB_fixture, PunctString) {
                                            CHECK_THROW(p-
>decrypt("IFMMP,XPSME"),cipher_error);
    }
    TEST_FIXTURE(KeyB_fixture, EmptyString) {
        CHECK_THROW(p->decrypt(""),cipher_error);
    }
    TEST(MaxShiftKey) {
        CHECK_EQUAL("THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG"
,
modAlphaCipher("Z").decrypt("SGDPTHBJAQNVMENWTLORNUDQSGDK
ZYXCNF"));
```

```
    }
}
int main(int argc, char **argv)
{
    return UnitTest::RunAllTests();
}
```

Код модуля modAlphaCipher.cpp:

```
#include "modAlphaCipher.h"
#include <locale>
modAlphaCipher::modAlphaCipher(const std::string& skey)
{
    for (unsigned i = 0; i < numAlpha.size(); i++) {
        alphaNum[numAlpha[i]] = i;
    }
    key = convert(getValidKey(skey));
    if (key.empty()) {
        throw cipher_error("Weak key");
    }
}
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::vector<int> result;
    for(auto c:s) {
        result.push_back(alphaNum[c]);
    }
    return result;
}
inline     std::string     modAlphaCipher::convert(const
std::vector<int>&v)
{
    std::string result;
    for(auto i:v) {
        result.push_back(numAlpha[i]);
    }
    return result;
```

```
}
inline   std::string   modAlphaCipher::getValidKey(const
std::string &s)
{
    if (s.empty())
        throw cipher_error("Empty key");
    std::string tmp;
    for (auto c : s) {
        if (!isalpha(c))
                  throw  cipher_error(std::string("Invalid
key"));
        tmp.push_back(toupper(c));
    }
    if (tmp == "AAA")
        throw cipher_error("Weak key");
    return tmp;
}
inline std::string modAlphaCipher::getValidOpenText(const
std::string & s)
{
    std::string tmp;
    for (auto c:s) {
        if (isalpha(c)) {
            if (islower(c))
                tmp.push_back(toupper(c));
            else
                tmp.push_back(c);
        }
    }
```

```
    if (tmp.empty())
        throw cipher_error("Empty open text");
    return tmp;
}
inline                                          std::string
modAlphaCipher::getValidCipherText(const std::string & s)
{
    if (s.empty())
        throw cipher_error("Empty cipher text");
    for (auto c:s) {
        if (!isupper(c))
                  throw  cipher_error(std::string("Invalid
cipher text"));
        }
                       return s;
}
```

Код модуля modAlphaCipher.h:

```
#pragma once
#include <vector>
#include <string>
#include <map>
#include <locale>
class modAlphaCipher
{
private:
    std::string numAlpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    std::map <char,int> alphaNum;
    std::vector <int> key;
    std::vector<int> convert(const std::string& s);
```

```
    std::string convert(const std::vector<int>& v);
    std::string getValidKey(const std::string & s);
    std::string getValidOpenText(const std::string & s);
      std::string getValidCipherText(const std::string &
s);
public:
    modAlphaCipher()=delete;
    modAlphaCipher(const std::string& skey);
    std::string encrypt(const std::string& open_text);
    std::string decrypt(const std::string& cipher_text);
};
class cipher_error: public std::invalid_argument
{
public:
    explicit cipher_error (const std::string& what_arg):
        std::invalid_argument(what_arg) {}
    explicit cipher_error (const char* what_arg):
        std::invalid_argument(what_arg) {}
};
```

Приложение Б

Код модуля maim.cpp:

```
#include <iostream>
#include <UnitTest++/UnitTest++.h>
#include "cipher.h"
using namespace std;

string check(int key, const string& msg)
{
   try {
      cout << "Ключ: " << key << endl;
         if (key <= 1) throw invalid_argument("Ключ должен быть целым числом больше единицы!");
       if (msg.empty()) throw invalid_argument("Исходный текст не может быть пустым!");
      Cipher cipher(key);
      cout << "Исходный текст: " << msg << endl;
      string encrypted = cipher.encrypt(msg);
      cout << "Зашифрованный текст: " << encrypted << endl;
      string decrypted = cipher.decrypt(encrypted);
      cout << "Расшифрованный текст: " << decrypted << endl;
      return encrypted;
   } catch (const exception& e) {
      cerr << "Error: " << e.what() << endl;
      return "";
   }
}
```

```
int runTests()
{
   return UnitTest::RunAllTests();
}

//Проверка ключа
SUITE(KeyTest)
{
   TEST(ValidKey) {
      CHECK_EQUAL("rvtpie", check(2, "privet"));//Верный ключ
   }

   TEST(LongerKeyThanMessage) {
      CHECK_EQUAL("hello", check(10, "hello"));//Ключ больше, чем
длина
   }

   TEST(KeyLessOne) {
         CHECK_THROW(check(1, "hello"), invalid_argument); //Ключ
меньше 1
   }
}

//Проверка исходного текста
SUITE(TextTest) {
   TEST(UppercaseLetters) {
          CHECK_EQUAL("RVTPIE", check(2, "PRIVET"));//Прописные
буквы
```

```
    }
    TEST(LowercaseLetters) {
        CHECK_EQUAL("rvtpie", check(2, "privet"));//Строчные
    }
    TEST(NonAlphabeticCharacters) {
            CHECK_THROW(check(2, "pri12!"), invalid_argument);//Есть
неалфавитные символы
    }
    TEST(EmptyText) {
                CHECK_THROW(check(2, ""), invalid_argument);//Пустая
строка
    }
    TEST(NonAlphabeticLetters) {
            CHECK_THROW(check(2, "123!@#"), invalid_argument);//Нет
букв
    }
}

int main(int argc, char **argv)
{
    runTests();
    return 0;
}
```

Код модуля cipher.cpp:

```
#include "cipher.h"
#include <string>
using namespace std;
```

```
Cipher::Cipher(int k) : key(k) {}

string Cipher::encrypt(const string& msg) {
    int kolvo_strok = (msg.length() + key - 1) / key;
    size_t index = 0;
    char table[kolvo_strok][key];

    for (int i = 0; i < kolvo_strok; i++) {
        for (int j = 0; j < key; j++) {
            if (index < msg.length()) {
                table[i][j] = msg[index];
                index++;
            } else {
                table[i][j] = ' ';
            }
        }
    }

    string encrypted;
    for (int j = key - 1; j < key && j >= 0; j = j - 1) {
        for (int i = 0; i < kolvo_strok; i++) {
            encrypted += table[i][j];
        }
    }
    return encrypted;
}

string Cipher::decrypt(const string& encrypted) {
```

```
    int kolvo_strok = encrypted.length() / key;
    size_t index = 0;
    char table[kolvo_strok][key];

    for (int j = key - 1; j < key && j >= 0; j = j - 1) {
        for (int i = 0; i < kolvo_strok; i++) {
            table[i][j] = encrypted[index];
            index++;
        }
    }

    string msg;
    for (int i = 0; i < kolvo_strok; i++) {
        for (int j = 0; j < key; j++) {
            msg += table[i][j];
        }
    }
    return msg;
}
```

Код модуля cipher.h:

```
#ifndef CIPHER_H
#define CIPHER_H
#include <string>
using namespace std;
class Cipher {
private:
    int key;
public:
```

```
    Cipher(int k);
    string encrypt(const string& msg);
string decrypt(const string& zashifrovan);
};
#endif
```