**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»**
**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

# ОТЧЕТ

**по лабораторной работе № 2**

## Вариант 11

**Название:** Тестирование программного обеспечения

**Дисциплина:** Технологии разработки программных продуктов

Студент ИУ6-43Б  В.К. Залыгин
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Е.К. Пугачев
(Подпись, дата) (И.О. Фамилия)

Москва, 2024

# 1 Цель лабораторной работы

Цель работы – приобрести навыки тестирования схем алгоритмов, исходных кодов программ и исполняемых модулей.

# 2 Структурный контроль

## 2.1 Задание

Программа должна формировать типизированный файл с информацией о фамилии человека, дне рождения, а также осуществлять поиск в файле информации о дне рождения человека.

## 2.2 Исходный код программы для тестирования

Исходный код для 11 варианта представлен на рисунке 1.

```
#include <iostream>
#include <string>

using namespace std;

struct fam {
    string ff;
    short year;
    unsigned char month;
    unsigned char day;
}

int main() {
    FILE* £;
    fam fb;
    int n, i;
    bool key;
    string fff;

    f=fopen("a.dat", "w");

    cout<<"Write data"<<endl;

    while (!EOF) {
        cin>>fb.ff>>fb. year>>fb.month>>fb.day;
        fwrite(sfb,sizeof(fb),1,£);
    }

    fclose(f);

    cout<<"Write lastname"<<endl;
    cin>>fff; key=true; f=fopen("a.dat", "r");
    while (fread(&fb, sizeof (fb),1,f)!=EOF && key) {
        if(fb.ff==fff) {
            cout<<"Date:"<<fb. year<< (short) fb.month<< (short) fb. day<<endl;
            key=false;
        }
    }
    if(!key) cout<<"No such lastname"<<endl;
    ftclose(f£);
}
```

Рисунок 1 – Исходный код программы

### 2.3 Результаты тестирования

Результаты тестирования сквозным структурным контролем представлены в таблице 1.

Таблица 1 – Результаты сквозного структурного контроля

| Номер вопроса | Строки, подлежащие проверке | Результат проверки | Вывод |
|---|---|---|---|
| 1.1 | Все строки | f – инициализируется, файл открывается<br>fb – вводится<br>n, i – не инициализируются, не используется<br>key – присваивается<br>fff – вводится | Не все переменные инициализированы. Но неинициализированные переменные не используются. Не влияет на работу программы. |
| 1.4 | 7, 14, 15, 18 | f, ff, fb, fff | Названия переменных схожи: f, ff, fb, fff. |
| 1.5 | 25 | Нет проверки на окончание потока ввода. | Данные вводятся без проверки на окончание. |
| 2.2 | 39 | В условии !key, хотя должно быть key (элемент не найден). | Ошибка в условии, некорректное вычисление условия срабатывания ветви. |
| 3.3 | 24 | В условии цикла !-1 = false. | Цикл никогда не выполнится. |

Вопросы структурного контроля интерфейса (раздел 4) не рассматриваются, так как программа не имеет подпрограмм, глобальных переменных, наложения переменных.

Вывод: сквозной структурный контроль позволяет до запуска программы найти общие ошибки написания кода без учета логических ошибок.

Плюсы подхода:

– Выявление ошибок на раннем этапе, до запуска программы;

– Не требует вычислительных мощностей для тестирования;

– Проводится одним человеком.

Минусы подхода:

– Ошибки, выявляемые структурным контролем, автоматически находятся программой анализа кода;

– Невозможно выявить логические ошибки.

## 3 Методы белого ящика

### 3.1 Схема алгоритма для тестирования

Схема алгоритма, подлежащего тестированию, для 11 варианта изображена на рисунке 2.

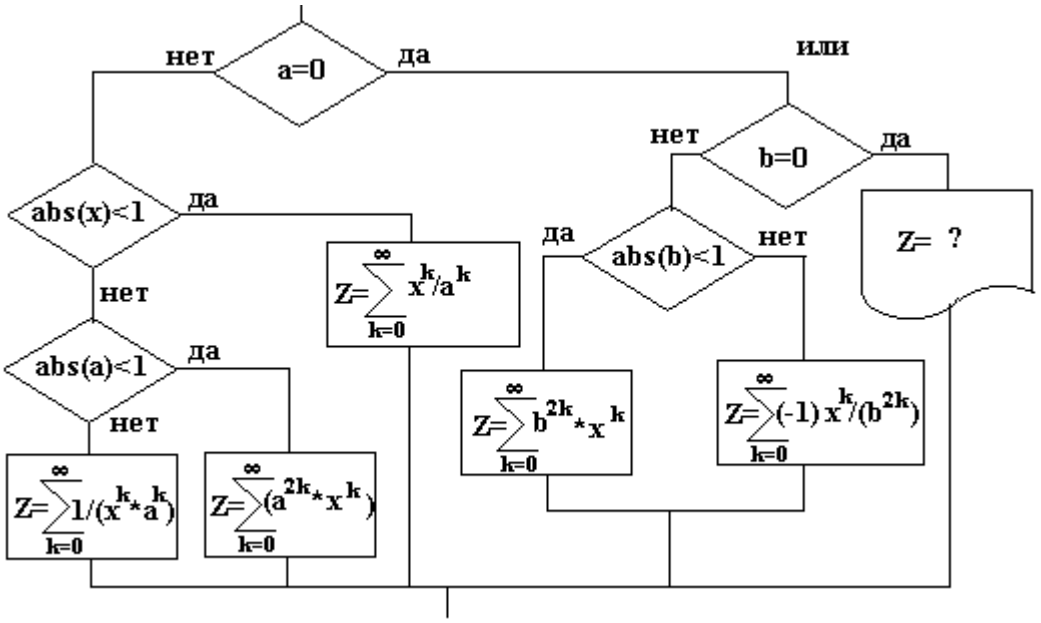
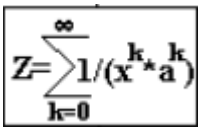
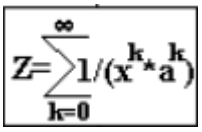
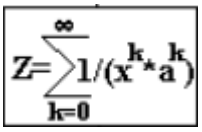
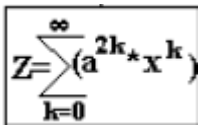
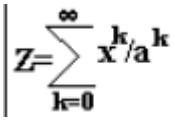


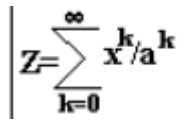
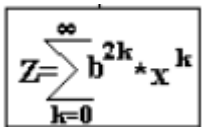
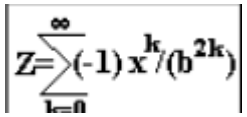
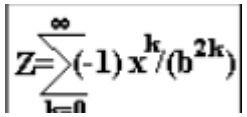

Рисунок 2 – Схема алгоритма

### 3.2 Метод покрытия операторов

Метод покрытия операторов подразумевает множество тестов, при котором каждый оператор выполняется хотя бы 1 раз. Тесты и их результаты представлены в таблице 2.

Таблица 2 – Тесты для метода покрытия операторов

| № | Назначение теста | Значение исходных данных | Ожидаемый результат |
|---|---|---|---|
| 1 | Проверить оператор  | a=1, x=1 | Выполняется |
| 2 | Проверить оператор (неправильные данные)  | a=0, x=1 | Недостижимо из-за условия |
| 3 | Проверить оператор (неправильные данные)  | a=1, x=0 | Недостижимо из-за условия |
| 4 | Проверить оператор  | a=2, x=2 | Мертвая ветвь, недостижимо из-за условия |
| 5 | Проверить оператор  | a=2, x=0 | Z=0 |

| | | | |
|---|---|---|---|
| 6 | Проверить оператор (неправильные данные)  | a=0, x=0 | Недостижимо из-за условия |
| 7 | Проверить оператор  | a=0, b=1 | Мертвая ветвь, недостижимо из-за условия |
| 8 | Проверить оператор  | a=0, b=2 | Выполняется |
| 9 | Проверить оператор (неправильные данные)  | a=0, b=0 | Недостижимо из-за условия |

Из таблицы можно сделать вывод, что в программе существует 2 мертвые ветви, недостижимые из-за непересекающихся условий. Для операторов, для которых можно определить неправильные условия, выделены тесты с неправильными данными, которые показывают, что выполнение таких операторов на неправильных данных невозможен.

**3.3 Метод покрытия решений**

Метод обеспечивает большее количество тестов за счет необходимости соблюдения условия, что должно проходить каждое решение на множестве тестов хотя бы 1 раз. Результаты метода отражены в таблице 3.

Таблица 3 – Тесты для метода покрытия решений

| № | Назначение теста | Значение исходных данных | Ожидаемый результат |
|---|---|---|---|
| 1 | нет, нет, нет | a!=0, x!=0 | Выполняется |
| 2 | нет, нет, да | a!=0, x!=0 | Мертвая ветвь, недостижимо из-за условия |
| 3 | нет, да | a!=0, x=0 | Z=0 |
| 4 | да, нет, да | a=0, b!=0 | Мертвая ветвь, недостижимо из-за условия |
| 5 | да, нет, нет | a=0, b!=0 | Выполняется |
| 6 | да, да | a=0, b=0 | Выполняется |

После тестирования можно сделать вывод, что существует всего 10 различных решений, 2 из которых являются недостижимыми, которые покрываются 6 тестами.

**3.4 Метод комбинаторного покрытия условий**

Данные метод предполагает набор тестов, при котором вычисляются все возможные комбинации условий в каждом решении.

По схеме можно выделить следующие комбинации:

1) a=0, abs(x)<1, abs(a)<1, b = 0

2) a=0, abs(x)<1, abs(a)<1, b != 0

3) a=0, abs(x)<1, abs(a)>1, b = 0

4) a=0, abs(x)<1, abs(a)>1, b != 0

5) a=0, abs(x)>=0, b = 0

6) a=0, abs(x)>=0, b != 0

7) a!=0, abs(x)>=0, b=0

8) a!=0, abs(x)<0,  b=0

9) a!=0, abs(x)>=0, b!=0, abs(b)<1

10) a!=0, abs(x)<0, b!=0, abs(b)<1

11) a!=0, asb(x)>=0, b!=0, abs(b)>=1

12) a!=0, abs(x)<0, b!=0, abs(b)>=1

Для данного набора условий можно составить набор тестов. Составленный набор тестов продемонстрирован в таблице 4.

Таблица 4 – набор тестов для комбинаторного метода

| № | Назначение теста | Значение исходных данных | Ожидаемый результат |
|---|---|---|---|
| 1 | a!=0,abs(x)>=1,abs(a)>=1 | a=1, x=1 | Выполняется |
| 2 | a!=0,abs(x)>=1,abs(a)<1 | a=1, x=1 | Мертвая ветвь, недостижимо из-за условия |
| 3 | a!=0,abs(x)<1 | a=2, x=0 | Z=0 |
| 4 | a=0,b!=0,asb(b)<1 | a=0, b=1 | Мертвая ветвь, недостижимо из-за условия |
| 5 | a=0,b!=0,abs(b)>=1 | a=0, b=2 | Выполняется |
| 6 | a=0,b=0 | a=0, b=0 | Нет вычислений |

Таблица 4 показывает, что в алгоритме присутствуют 2 мертвые ветви.

## 4 Метод черного ящика

### 4.1 Требования к тестируемой программе

Программа должна строить график функции по заданным в таблице значениям. Обеспечить возможность выбора вида графика: точки отдельно или точки соединены (исполняемый модуль v11.exe).

**4.2 Метод эквивалентного разбиения**

Результаты тестирования в рамках метода эквивалентного разбиения представлены в таблице 5.

Таблица 5 – Тесты для метода эквивалентного разбиения.

| № | Назначение теста | Значение исходных данных | Ожидаемый результат | Реакция программы | Вывод |
|---|---|---|---|---|---|
| 1 | Работа при возрастающих x и y. | Значения координат точек монотонно возрастают | Отображение графика | Отображение графика | Корректная работа |
| 2 | Работа при невозрастающи x y и возрастающих x | Иксы точек возрастают, игреки не возрастают. | Отображение графика | Отображение графика | Корректная работа |
| 3 | Работа при вещественные невозрастающи x и x | Иксы вещественные не возрастают | Отображение графика | Неправильный график | Отказ работы |
| 4 | Работе при целых невозрастающи x x | Иксы целые не возврастают | Отображение графика | Всплывающая ошибка “количество точек графика не может быть меньше двух” | Отказ работы, неправильн ое описание ошибки |
| 5 | Работа при одинаковых x целых | Одинаковые x | Ошибка | Всплывающая ошибка “количество точек графика не может быть меньше | Отказ работы, неправильн ое описание ошибки |

| | | | | | |
|---|---|---|---|---|---|
| | | | | двух” | |
| 6 | Работа при положительных координатах | Иксы и игреки положительны | Отображение графика | Отображение графика | Корректная работа |
| 7 | Работа при отрицательных координатах | Иксы и игреки отрицательны | Отображение графика | Отображение графика | Корректная работа |
| 8 | Ввод нецелых чисел | Координаты имеют вещественную часть | Отображение графика | Отображение графика | Корректная работа |
| 9 | Ввод нечисел | В поля ввода чисел вводятся слова | Сообщение об ошибке | Сообщение об ошибке “is not a valid floating point value” | Корректная работа |
| 10 | Ввод пустого значения | Поля ввода чисел остаются пустыми | Сообщение об ошибке | Сообщение об ошибке “is not a valid floating point value” | Корректная работа |
| 11 | Соединение точек | Выставлен параметр “соединить точки” | Точки соединяются | Точки соединяются | Корректная работа |
| 12 | Рисование только точек | Снят параметр “соединить точки” | Точки не соединяются | Точки не соединяются | Корректная работа |

### 4.3 Метод граничных условий

Результаты тестирования в рамках метода граничных условий представлены в таблице 6. Тестирование идет по различным значениям координат точек.

Таблица 6 – результаты тестирования методом граничных условий

| № | Назначение теста | Значение исходных данных | Ожидаемый результат | Реакция программы | Вывод |
|---|---|---|---|---|---|
| 1 | Ввод очень больших чисел | Координаты имеют | Отображение масштабирован | Отображение масштабированно | Корректная работа |

| | | | | | |
|---|---|---|---|---|---|
| | | значения порядка 10 миллиардов | ного графика | го графика | |
| 2 | Ввод очень маленьких чисел | Координаты имеют значения порядка 1 миллионной | Отображение масштабированного графика | Отображение масштабированного графика | Корректная работа |
| 3 | Ввод очень близких значений координат x | Иксы у точек имеют разность порядка 1 миллионной | Отображение масштабированного графика | Отображение масштабированного графика | Корректная работа |

## 4.4 Метод анализа причинно-следственных связей

Таблица 7 – результаты тестирования методом анализа причинно-следственных связей

| № | Назначение теста | Значение исходных данных | Ожидаемый результат | Реакция программы | Вывод |
|---|---|---|---|---|---|
| 1 | Соединение точек | Выставлен параметр “соединить точки” | Точки соединяются | Точки соединяются | Корректная работа |
| 12 | Рисование только точек | Снят параметр “соединить точки” | Точки не соединяются | Точки не соединяются | Корректная работа |

## 4.5 Диаграмма логической схемы работы

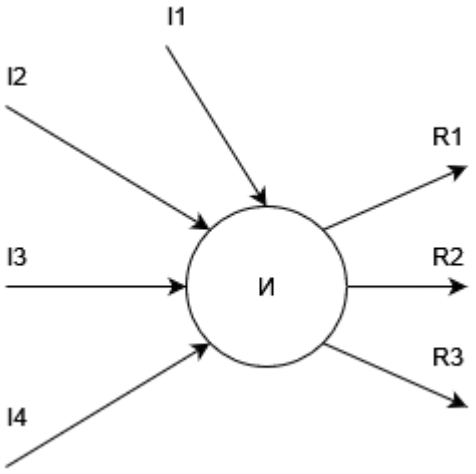


Рисунок 3 – диаграмма логической схемы работы

- I1 – ввод всех X;
- I2 – ввод всех Y;
- I3 – установка флага “соединять точки”;
- I4 – нажатие кнопки построить;
- R1 – сообщение об ошибке;
- R2 – график с соединенными точками;
- R3 – график с не соединенными точками.

## 5 Вывод

В результате исследования методов тестирования получены теоретические знания по различным видам тестирования в разделах ручного тестирования, “белого ящика”, “черного ящика”, исследованы программы по методам ручного тестирования, “белого ящика”, “черного ящика”.