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**

О Т Ч Е Т

по лабораторной работе №_2_

Вариант № 25

Дисциплина:_ Технология разработки программных систем _

Название: Тестирование программного обеспечения _

Студент ИУ6-42б _____ И.С. Марчук
(Группа) (Подпись, дата) (И.О. Фамилия)

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

2021 г.

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

Задание: Вариант 25

Программа должна генерировать случайным образом слова в количестве N и удалять слово из строки, если оно существует и начинается на гласную букву.

Исходный код программы для тестирования представлен на рисунке 1.

```
#include <iostream>
#include <set>
#include <time.h>

using namespace std;

int main()
{
        string Sg,Ss;
        int i,n,n1,n2,L,k;
        bool Flag;
        set<char> Setx={'E','Y','U','O','A'};

        n=5; //кол-во слов
        srand(time(0));
        sg=""; L=rand()%(n2-n1+1)+n1;
        k=0; // счетчик букв в слове
        Ss="";
        while(k<L)
        {
        Ss+=(char)(rand()%26+65); k++;
        }
        Sg+=' '+Ss; //одно слово уже есть
        i=0; //счетчик слов
        while(i<n)
        {
        L=rand()%(n2-n1+1)+n1;
        k=0; //счетчик букв в слове
        Ss="";
        while(k<L)
        {
        Ss+=(char)(rand()%26+65);
        k++;
        }
        i++;
        Sg+=' '+Ss;
        }
        cout<<Sg<<endl;
        cout<<"Word to remove: "<<flush; cin>>Ss;
        i=Ss.find(Sg);
        L=Ss.length();
        if(i>0 && Setx.count(Ss[1])) Sg.erase(i,L);
        cout<<Sg<<endl;
}
```

Рисунок 1 – Исходный текст программы

Основная часть

1. Структурный контроль

Результаты тестирования структурным контролем представлены в таблице 1.

Таблица 1 – Результаты тестирования структурным контролем

Номер вопроса	Строки, подлежащие проверке	Результаты проверки	Вывод
1.1 Все ли переменные инициализированы?	Все строки программы	n1, n2 - ? sg = "";	Не все переменные инициализированы. Переменные n1 и n2 приведут к ошибке вычисления L в 16 строке/ Используются переменные sg и Sg, однако из них объявлена только Sg
1.4 Присутствуют ли переменные со сходными именами?	9-10	Sg, sg, Ss, i, n, n1, n2, L, k	Однобуквенные обозначение переменных и выбор неосмысленных названий. Схожие имена: Sg, sg и Ss, n1 и n2.
3.3 Существуют ли циклы, которые не будут выполняться из-за нарушения условия входа? Корректно ли продолжатся вычисления?	20-24, 32-36	while (k < L)	Вход в цикл не происходит, т.к. происходит некорректное сравнение с переменной L из-за ошибки ее вычисления ранее.
2.2 Корректно ли производятся вычисления неарифметических переменных?	Вся программа	Ss += (char)(rand() % 26 + 65);	Вычисления производятся некорректно – вместо слов программа формирует числа.

Вывод:

Структурный контроль позволяет найти большое количество общих ошибок.

Достоинства:

Выполнение структурного контроля возможно на любом этапе разработки программного продукта, когда есть код. Не нужно тратить ресурсы на создание тестирующего ПО. Также такой вид контроля не требует

выполнения программы и позволяет обнаружить общие ошибки программиста.

Недостатки:

Не все ошибки могут быть выявлены таким методом в силу невнимательности и сложности программы. Так, например, не всегда получается заметить неправильную передачу параметров и другие ошибки в логике программы. Хотя, те же ошибки можно автоматически выявить средствами ПО. Большие программы трудно инспектировать данным методом.

2. Методы белого ящика

Схема алгоритма для тестирования

В соответствии с вариантом задания №25 была протестирована схема алгоритма, представленная на рисунке 2.

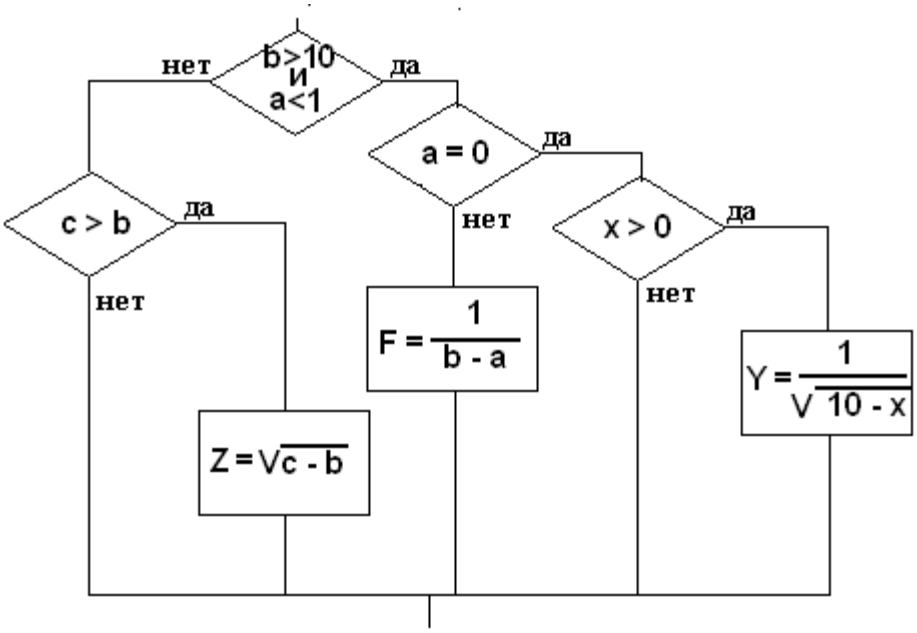

Рисунок 2 – Схема алгоритма

Метод покрытия операторов

Результаты тестирования по методу покрытия операторов представлены в таблице 2.

Таблица 2 – Результаты тестирования по методу покрытия операторов

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат
1	Проверить оператор $Z = \sqrt{c - b}$	a = 0 b = 5 c = 9 x = 0 (любой)	$Z = 2$
2	Проверить оператор $F = \frac{1}{b-a}$	a = -1 b = 11 c = 0 (любой) x = 0 (любой)	$F = \frac{1}{12}$
3	Проверить оператор $Y = \frac{1}{\sqrt{10-x}}$	a = 0 b = 11 c = 0 (любой) x =6	$Y = 0.5$

Метод покрытия решений

Результаты тестирования по методу покрытия решений представлены в таблице 3.

Таблица 3 – Результаты тестирования по методу покрытия решений

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат
1	Нет, нет	a = 0 b = 5 c = 5 x = 0 (любой)	Нет оператора
2	Нет, да	a = 0 b = 5 c = 9 x = 0 (любой)	$Z = 2$
3	Да, нет	a = -1 b = 11 c = 0 (любой) x = 0 (любой)	$F = \frac{1}{12}$
4	Да, да, нет	a = 0 b = 11 c = 0 (любой) x = 0	Нет оператора
5	Да, да, да	a = 0 b = 11 c = 0 (любой) x =6	$Y = 0.5$

Метод комбинаторного покрытия условий

По схеме алгоритма можно выделить 10 комбинаций условий:

1) $b > 10$ и $a < 1$
2) $b > 10$ и $a \geq 1$
3) $b \leq 10$ и $a < 1$
4) $b \leq 10$ и $a \geq 1$
5) $c > b$
6) $c \leq b$
7) $a = 0$
8) $a \neq 0$
9) $x > 0$
10) $x \leq 0$

Вышеперечисленные комбинации можно покрыть шестью тестами. Результаты тестирования представлены в таблице 4.

Таблица 4 – Таблица результатов тестирования для комбинаторного покрытия условий

Номер теста	Значение исходных данных	Комбинации	Ожидаемый результат
1	a = -1 b = 11 c = 0 (любой) x = 0 (любой)	1, 8	$F = \frac{1}{b-a}$ $F = \frac{1}{12}$
2	a = 0 b = 11 c = 0 (любой) x = 0	1, 7, 10	Нет оператора
3	a = 0 b = 11 c = 0 (любой) x = 6	1, 7, 9	$Y = \frac{1}{\sqrt{10-x}}$ $Y = 0.5$
4	a = 0 b = 5 c = 9 x = 0 (любой)	4, 5	$Z = \sqrt{c-b}$ $Z = 2$
5	a = 1 b = 11 c = 10 x = 0 (любой)	2, 6	Нет оператора
6	a = 0 b = 10	3, 6	Нет оператора

	c = 10 x = 0 (любой)		

Тестирование с использованием стратегии «белого ящика» позволяет выявить ошибки в логике программы и проверить ее внутреннюю структуру. Для этого необходимо исчерпывающее количество маршрутов тестирования.

В данной задаче тесты метода покрытия решений и метода покрытий операторов совпали, но это необязательное условие. Наиболее исчерпывающим методом тестирования является метод комбинаторного покрытия условий.

2. Метод черного ящика

Требования к тестируемой программе

Написать программу, которая генерирует строку случайным образом и определяет самое короткое и самое длинное слово. Входным параметром является количество слов в строке.

На рисунке 3 представлен вид интерфейса тестируемой программы.

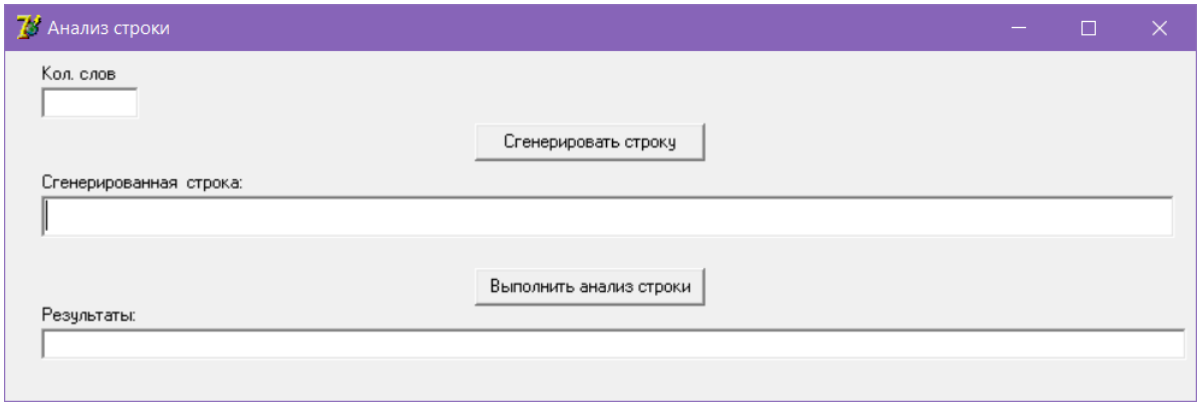

Рисунок 3 – Интерфейс программы

Метод эквивалентного разбиения

Выделенные классы эквивалентности исходных данных представлены в таблице 5.

Таблица 5 – Классы эквивалентности

Входное условие	Правильные классы эквивалентности	Неправильные классы эквивалентности
Тип входных данных	1) Целое положительное число	2) Целое неположительное число 3) Вещественное число 4) Строка
Пустые поля	5) Непустое поле	6) Пустое

Результаты тестирования представлены в таблице 6.

Таблица 6 – Тестирование методом эквивалентного разбиения

Номер теста	Проверяемые классы эквивалентности	Значения исходных данных	Ожидаемый результат	Полученный результат	Вывод
1	1, 5	3	Сгенерирована строка из 3х слов. Правильно определены самое короткое и самое длинное слово.	Сгенерирована строка из 3х элементов. Правильно определяет длину самого короткого и самого длинного слова. Но неправильно выводит самое длинное слово – обрезает последнюю букву.	Программа работает некорректно - вывод результата частично неверный.
2	6	Пустое поле	Сообщение о пустом поле	Сгенерирована строка из одного слова. Оно является самым коротким и самым длинным.	Программа работает некорректно – вместо сообщения об ошибке результат вычислений.
3	2	-5	Сообщение о некорректном вводе	Сгенерирована строка из одного слова. Оно является самым коротким и самым длинным.	Программа работает некорректно – вместо сообщения об ошибке результат вычислений.
4	3	5,7	Сообщение о некорректном вводе	Сгенерирована строка из 5 слов – десятичная часть отбрасывается.	Программа работает некорректно – вместо сообщения об ошибке результат вычислений.
5	4	‘aaa’	Сообщение о некорректном вводе.	Сгенерирована строка из одного слова.	Программа работает некорректно – вместо сообщения об ошибке результат вычислений.

Метод граничных условий

Метод граничных условий позволяет протестировать программу на значениях лежащих на границах классов эквивалентности, отбросив при этом большое количество лишних для тестов значений.

Программа должна работать при вводимом значении больше нуля и меньше 20: при количестве слов равном или меньше нулю строка не существует, а при количестве больше 20 – трудно проверить правильность вычислений, так как слова уже не будут помещаться в поле вывода.

Таблица 7 – Таблица результатов тестирования для метода граничных условий

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат	Реакция программы	Вывод
1	Проверка нижней границы	0	Сообщение об ошибке ввода	Сгенерирована строка из одного слова. Оно является самым коротким и самым длинным.	Программа некорректно работает на значениях, близких к граничным
2	Проверка нижней границы	1	Сгенерирована строка из 1 слова. Оно является словом максимальной и минимальной длины.	Сгенерирована строка из 1 слова. Оно является словом максимальной и минимальной длины.	Программа работает корректно
3	Проверка верхней границы	20	Сгенерирована строка из 20 слов. Определены слова максимальной и минимальной длины.	Сгенерирована строка из 20 слов. Определены слова максимальной и минимальной длины.	Программа работает корректно
4	Проверка верхней границы	21	Сообщение об ошибке ввода	Сгенерирована строка из 21 слова. Определены слова максимальной	Программа не учитывает, что на значениях больше 20

				и минимальной длины.	читать слова из строки будет не удобно
5	Проверка максимальных значений, при которых программа остается работоспособной	5850		При вводе очень больших чисел (больше чем 5800) программа не выдает сообщение об ошибке, однако текстовое поле остается пустым в половине попыток выполнения теста	Программа некорректно работает на значениях, больше чем 5800. Время от времени выдавая строку со словами, а время от времени пустую строку. В программе нет проверки на максимально допустимые значения.

Анализ причинно-следственных связей

В таблице 9 представлены результаты тестирования, полученные на основе анализа причинно-следственных связей.

Таблица 8 – Таблица истинности для работы программы

В поле ввода записаны корректные данные	Нажата кнопка «Сгенерировать строку»	Нажата кнопка «Выполнить анализ строки»	Результат
0	0	1	Сообщение о необходимости корректного ввода
0	1	X	Сообщение о необходимости корректного ввода
1	0	1	Сообщение о необходимости нажать первую кнопку
1	1	1	Выполнение программы

Таблица 9 – Таблица результатов тестирования на основе анализа причинно-следственных связей

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат	Реакция программы	Вывод
1	Проверка некорректного ввода (1-2 строки таблицы истинности)	-7	Сообщение о необходимости корректного ввода	Сгенерирована строка и выполнен анализ.	Программа работает некорректно
2	Проверка факта нажатия кнопок (3 строка таблицы истинности)	3	Сообщение о необходимости нажать первую кнопку	Выполнен анализ нулевой строки	Программа работает некорректно
3	Проверка выполнения программы	3	Сгенерирована строка из 3 слов. Определены слова максимальной и минимальной длины.	Программы выполнена, но вывод результата некорректный	Программа работает некорректно

Метод черного ящика не позволяет найти ошибки в структуре и логике программы, так как при его выполнении внутренняя часть программы неизвестна. Но с его помощью возможно выяснение обстоятельств, в которых поведение программы не соответствует спецификации. Одни и те же ошибки можно обнаружить разными методами «черного ящика».

Заключение

В результате исследования методов тестирования были получены следующие результаты:

1. выявлены ошибки различных видов;
2. оценена специфика каждого метода тестирования;
3. оценена трудоемкость тестирования для каждого метода.