Министерство науки и высшего образования Российской Федерации

Новосибирский Государственный технический университет

Кафедра автоматизированных систем управления



**Отчет по лабораторной работе № 5**

**по дисциплине «Технологии и методы программирования»**

**«**Тестирование программных продуктов**»**

Выполнили:

студент группы АВТ-214

Крипаков Леонид Дмитриевич

Сушкин Никита Максимович

г. Новосибирск

2024 г

**Оглавление**

Задание № 1. 3

Черный ящик 3

Разбиение на классы эквивалентности 3

Анализ причинно-следственных связей 4

Белый ящик 5

Задание № 2. 9

Черный ящик 9

Белый ящик 10

Исходный текст класса и тестирующей программы. 12

# Задание № 1.

Разработать программу определения суммарной длины тени, которую отбрасывают на ось абсцисс отрезки, параллельные этой оси и заданные целочисленными абсциссами начала и конца отрезка.

# Черный ящик

1. Ознакомиться с теоретическими сведениями по стратегиям тестирования.

2. Для предложенных вариантов разработать программу для каждого варианта.

3. Подготовить для этих программ тесты по методикам стратегии "черного ящика".

4. Предлагаемые тесты свести в таблицу.

# Разбиение на классы эквивалентности

|  |  |  |
| --- | --- | --- |
| Входное условие | Допустимые классы эквивалентности | Недопустимые классы эквивалентности |
| Количество отрезков – целое число от 1 до бесконечности | 0 < x < ∞ | x <= 0 |
| x – целое число | x – не целое число |
| Координаты начала и конца отрезка – целое число и дробное. | x – целое число | ------------------------------------------ |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер теста | Назначение теста | Значения исходных данных | Ожидаемый результат | Фактический результат | Результат тестирования |
| 1 | Тестирование на пустом вводе | N | Программа должна выдать ошибку. |  | Не успешно |
| 2 | Тестирование на вводе неправильных данных для n | N= abc | Программа должна выдать ошибку при конвертации типа данных из строки в int |  | Не успешно |
| 3 | Тестирование на отрицательном значении n | N= -5 5 | Программа должна принять число. |  | Успешно |
| 4 | Тестирование на не целом значении координат отрезка | N = 1 1  2,2 3,4 | Программа должна принять число. |  | Успешно |

# Анализ причинно-следственных связей

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | N = 0 | N > 0 | N < 0 | X1>X2 | X1<X2 | x1 & x2 целые | x1 || x2 не целые | Результат |
| 1 | 1 | 0 | 0 | 1 | 0 | x | x | неверный ввод |
| 2 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | результат |
| 3 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | результат |
| 4 | 0 | 1 | 0 | 0 | 1 | х | х | результат |
| 5 | 0 | 0 | 1 | х | х | х | х | результат |

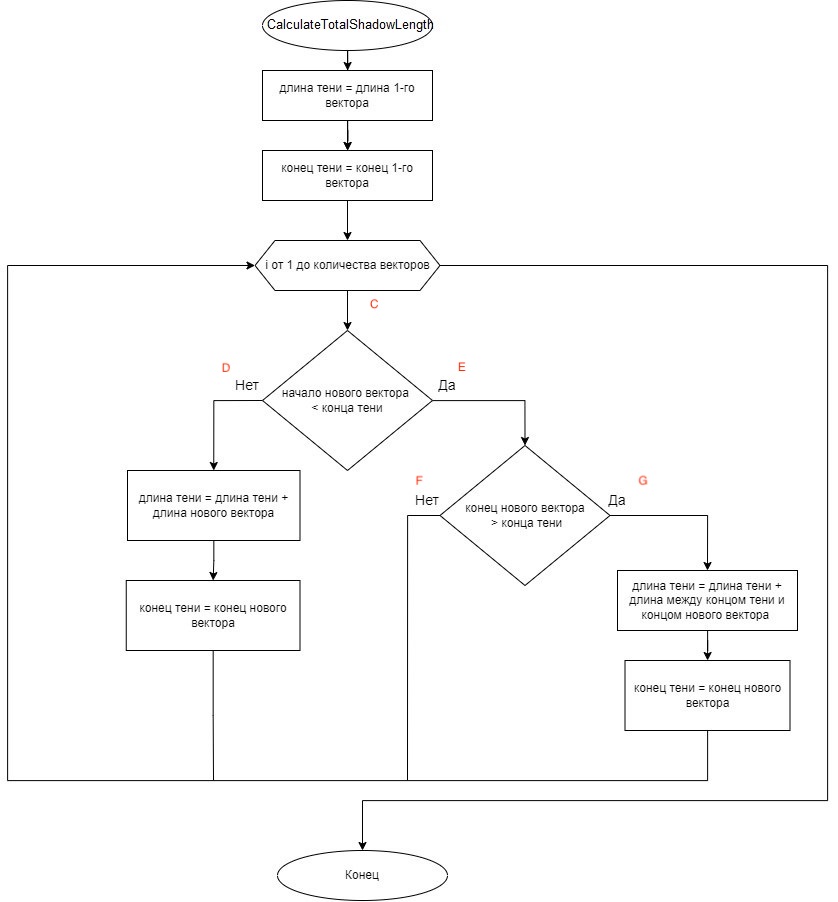
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер теста | Значения исходных данных | Ожидаемый результат | Фактический результат | Результат тестирования |
| 1 | 1) 0 0 | 0 | 0 | не успешно |
| 2 | 1)-50 -40  2)-30 -20  3)-10 0  4) 10 10 | 30 | 30 | не успешно |
| 3 | 1)-1,5 4  2)2 -5  3)3,1 0  4)10 12  5)11 11 | 11 | 11 | не успешно |
| 4 | 1) 1,1 2 | 0,8(9) | 0,8(9) | не успешно |

# Белый ящик

1. Ознакомиться с теоретическими сведениями по стратегиям тестирования.

2. Для программ, разработанных для тестирования по стратегии чёрного ящика, составить блок-схемы алгоритмов. Если размер программы велик, то выделить фрагмент кода, содержащий не менее двух операторов if.

3. Подготовить тесты по методам стратегии "белого ящика".

4. Для каждого метода предлагаемые тесты свести в таблицу.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Номер теста | Назначение теста | Путь | Значения исходных данных | Ожидаемый результат | Фактический результат | Результат тестирования |
| 1 | Метод покрытия операторов | CEG | 3 5  10 20 | 12 | 12 | неуспешно |
| 2 | Метод покрытия решений (покрытия переходов) | CD, CEG, CEF | 1 5  4 6  10 15  12 13 | 11 | 11 | неуспешно |
| 3 | Метод покрытия условий | CD, CEG, CEF | 1 5  8 10  6 9  2 3 | 10 | 10 | неуспешно |

Тесты:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using ConsoleApp3;

namespace TestProject1

{

[TestClass]

public class WhiteBoxTests

{

[TestMethod]

public void StatementCoverage()

{

var expected = 10;

var data = new int[][] {

new int[] { 1, 5 },

new int[] { 4, 6 },

new int[] { 10, 15 },

new int[] { 12, 13 }

};

var actual = Program.CalculateTotalShadowLength(data);

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void BranchCoverage()

{

var expected = 7;

var data = new int[][] {

new int[] { 1, 5 },

new int[] { 10, 13 },

};

var actual = Program.CalculateTotalShadowLength(data);

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void ConditionCoverage()

{

var expected = 8;

var data = new int[][] {

new int[] { 1, 5 },

new int[] { 8, 10 },

new int[] { 6, 9 },

new int[] { 2, 3 }

};

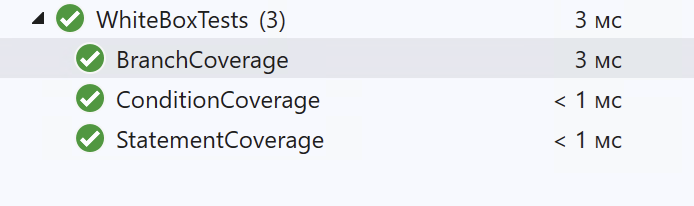
var actual = Program.CalculateTotalShadowLength(data);

Assert.AreEqual(expected, actual);

}

}

}



Программа:

namespace Segments;

internal static class Program

{

private static void Main()

{

var success = false;

string? input;

uint n = 0;

while (success != true)

{

Console.Write("Введите количество отрезков: ");

input = Console.ReadLine();

success = uint.TryParse(input, out n);

}

var segments = new double[n][];

for (var i = 0; i < n; i++)

{

double firstCoordinate = 0;

success = false;

while (success != true)

{

Console.Write("Введите начальную координату отрезка " + (i + 1) + ": ");

input = Console.ReadLine();

success = double.TryParse(input, out firstCoordinate);

}

success = false;

double endCoordinate = 0;

while (success != true)

{

Console.Write("Введите конечную координату отрезка " + (i + 1) + ": ");

input = Console.ReadLine();

success = double.TryParse(input, out endCoordinate);

}

segments[i] = new[] { firstCoordinate, endCoordinate };

}

double totalShadowLength = CalculateTotalShadowLength(segments);

Console.WriteLine("Суммарная длина тени: " + totalShadowLength);

}

private static double CalculateTotalShadowLength(double[][] segments)

{

Array.Sort(segments, (a, b) => a[0].CompareTo(b[0]));

double totalLength = 0;

double currentEnd = int.MinValue;

foreach (var segment in segments)

{

double start = segment[0];

double end = segment[1];

if (start > currentEnd)

{

totalLength += end - start;

currentEnd = end;

}

else if (end > currentEnd)

{

totalLength += end - currentEnd;

currentEnd = end;

}

}

return Math.Abs(totalLength);

}

}

# Задание № 2.

Разработать программу определения суммарной длины тени, которую отбрасывают на ось абсцисс отрезки, параллельные этой оси и заданные целочисленными абсциссами начала и конца отрезка.

# Черный ящик

1. Ознакомиться с теоретическими сведениями по стратегиям тестирования.

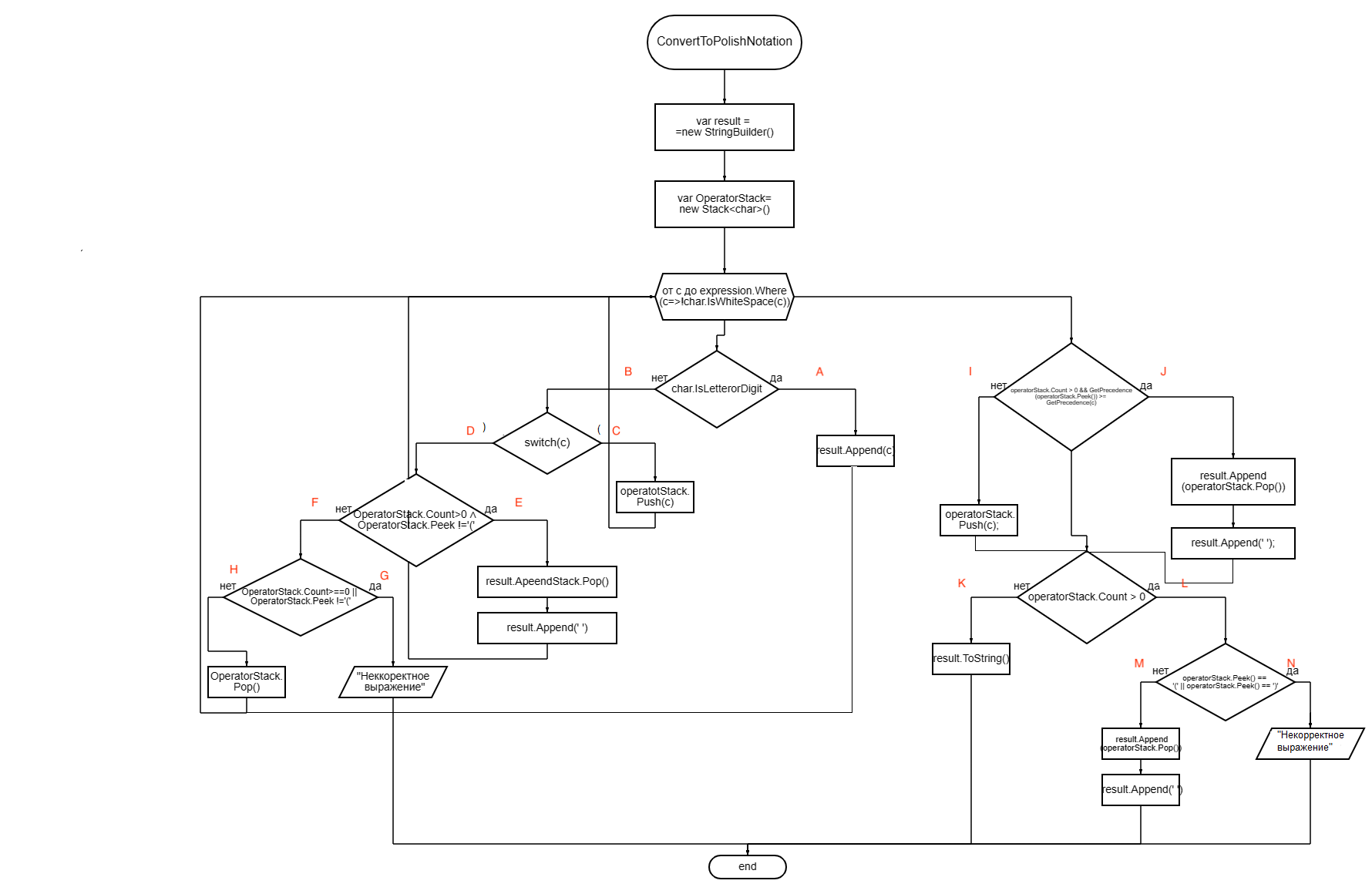
2. Для предложенных вариантов разработать программу для каждого варианта.

3. Подготовить для этих программ тесты по методикам стратегии "черного ящика".

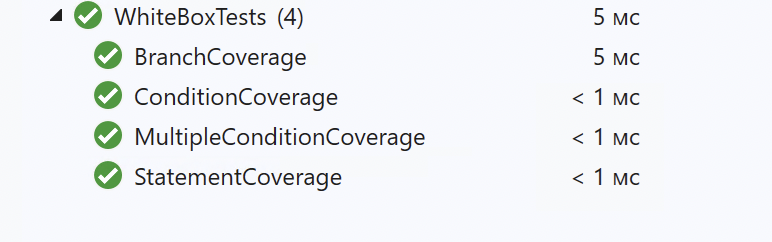
4. Предлагаемые тесты свести в таблицу.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер теста | Назначение теста | Значения исходных данных | Ожидаемый результат | Фактический результат | Результат тестирования |
| 1 | Тестирование на корректность перевода простого выражения в польскую нотацию без скобок | 2+2+2+2 | 2 2 + 2 + 2 + = 8 |  | Не успешно |
| 2 | Тестирование на корректность перевода сложного выражения в польскую нотацию | 3 \* (10 + 13) | 3 10 13 + \* = 69 |  | Не успешно |
| 3 | Тестирование на обработку ошибочного ввода | 10++10 | 10 + 10 + |  | Не успешно |
| 4 | Тестирование на обработку деления на ноль | 20/0 | сообщение об ошибке "Деление на ноль" |  | Не успешно |

# Белый ящик



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Назначение теста | Путь | Значения исходных данных | Ожидаемый результат | Фактический результат | Результат тестирования |
| 1 | Метод покрытия операторов | BDFHILM | (( с - с )/ с)\*a | cc- c/ a\* | cc- c/ a\* | неуспешно |
| 2 | Метод покрытия решений | BDFHILM, AJK, BCLN, BDEIK, BDFG | a+c/b-(f+f) | acb/ + ff+ - | acb/ + ff+ - | неуспешно |
| 3 | Метод покрытия условий | BDFHILM, AJK, BCLN, BDEIK, BDFG | 1. a-f\*(с-с) 2. a-f\*(с-с)) 3. a-f\*((с-с) | 1. afсс- \* - 2. Некорректное выражение 3. Некорректное выражение | 1. afсс- \* - 2. Некорректное выражение 3. Некорректное выражение | неуспешно |
| 4 | Метод комбинаторного покрытия условий | BDFHILM, AJK, BCLN, BDEIK, BDFG | 1. a+b 2. a-f\*(с-с) 3. a-f\*(с-с)) 4. a-f\*((с-с) | 1. ab+ 2. afсс- \* - 3. Некорректное выражение 4. Некорректное выражение | 1. ab+ 2. afсс- \* - 3. Некорректное выражение 4. Некорректное выражение | неуспешно |



# Исходный текст класса и тестирующей программы.

Тесты:

public class WhiteBoxTests

{

[TestMethod]

public void StatementCoverage()

{

var expected = "cc- c/ a\* ";

var actual = Program.ConvertToPolishNotation("(( c - c )/ c)\*a");

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void BranchCoverage()

{

var expected = "acb/ + ff+ - ";

var actual = Program.ConvertToPolishNotation("a+c/b-(f+f)");

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void ConditionCoverage()

{

var expected = "afсс- \* - ";

var expected\_err = "Некорректное выражение";

var actual = Program.ConvertToPolishNotation("a - f \* (с - с)");

Assert.AreEqual(expected, actual);

actual = Program.ConvertToPolishNotation("a - f \* (с - с))");

Assert.AreEqual(expected\_err, actual);

actual = Program.ConvertToPolishNotation("a - f \* ((с - с)");

Assert.AreEqual(expected\_err, actual);

}

[TestMethod]

public void MultipleConditionCoverage()

{

var expected = "ab+ ";

var expected\_err = "Некорректное выражение";

var actual = Program.ConvertToPolishNotation("a + b");

Assert.AreEqual(expected, actual);

expected = "afсс- \* - ";

actual = Program.ConvertToPolishNotation("a - f \* (с - с)");

Assert.AreEqual(expected, actual);

actual = Program.ConvertToPolishNotation("a - f \* (с - с))");

Assert.AreEqual(expected\_err, actual);

actual = Program.ConvertToPolishNotation("a - f \* ((с - с)");

Assert.AreEqual(expected\_err, actual);

}

Программа:

using System.Text;

namespace PolishInverseNotation;

internal static class Program

{

private static void Main()

{

Console.Write("Введите выражение: ");

var expression = Console.ReadLine();

var polishNotation = ConvertToPolishNotation(expression!);

Console.WriteLine(polishNotation);

}

private static string ConvertToPolishNotation(string expression)

{

var result = new StringBuilder();

var operatorStack = new Stack<char>();

foreach (var c in expression)

{

if (char.IsLetterOrDigit(c) || char.IsWhiteSpace(c))

{

result.Append(c);

continue;

}

switch (c)

{

case '(':

operatorStack.Push(c);

continue;

case ')':

{

while (operatorStack.Count > 0 && operatorStack.Peek() != '(')

{

result.Append(operatorStack.Pop());

result.Append(' ');

}

if (operatorStack.Count == 0 || operatorStack.Peek() != '(')

{

Console.WriteLine("Некорректное выражение");

Environment.Exit(0);

}

operatorStack.Pop();

continue;

}

}

while (operatorStack.Count > 0 && GetPrecedence(operatorStack.Peek()) >= GetPrecedence(c))

{

result.Append(operatorStack.Pop());

result.Append(' ');

}

operatorStack.Push(c);

}

while (operatorStack.Count > 0)

{

if (operatorStack.Peek() == '(' || operatorStack.Peek() == ')')

{

Console.WriteLine("Некорректное выражение");

Environment.Exit(0);

}

result.Append(operatorStack.Pop());

result.Append(' ');

}

return result.ToString();

}

private static int GetPrecedence(char c)

{

switch (c)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

default:

return 0;

}

}

}