МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедра Автоматизированных систем управления

**ОТЧЕТ О ВЫПОЛНЕНИИ**

**Лабораторной работы № 5**

Дисциплина «*Технологии и методы программирования*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Выполнили:** |  | **Проверил:** |  | |
| Студенты | *Бузмаков А.И.,*  *Шестаков К.Д.* | Преподаватель | *Астапчук В. А., Пустовских Д. А.* | |
| Факультет | *АВТФ* |  |  | |
| Направление (специальность) подготовки | *09.03.01 – Информатика и вычислительная техника* | Балл: \_\_\_\_\_\_\_\_\_ | |  |
| Группа | *АП-227* | Оценка *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* | | |
| Шифр |  |  |  | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
| подпись | | подпись | | |
| Дата сдачи: | «5» мая 2024г. | Дата защиты: | «\_\_»\_\_\_\_\_\_\_\_\_\_20\_\_г. | |

Новосибирск 2024

**Цель работы**

Освоить приёмы создания тестов для тестирования программных модулей.

Задание

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.

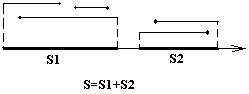
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер теста | Назначение теста | Значения исходных данных | Ожидаемый результат | Фактический результат | Результат тестирования |

5. Выполнить тестирование. Занести в таблицу результаты. В графу “фактический результат” записываются получаемые при прогоне значения выходных переменных. Если эти значения совпадают с ожидаемыми, то в программе ошибка не обнаруживается и в графе “результат тестирования” записываем “неуспешно”, что может свидетельствовать как об отсутствии ошибок, так и о наличии в программе ошибок, которые данным тестом не были найдены.

6. Сделать вывод о роли тестирования с использованием стратегии "черного ящика" и возможностях его применения. Сформулировать его достоинства и недостатки.

**Варианты заданий**

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



2. Разработать программу перевода символьного арифметического выражения, возможно со скобками, в ПОЛИЗ (польскую инверсную запись). Использовать только целочисленные константы.

**Решение**

Для разработанной программы расчета длины тени мы создали юнит-тесты чтобы проверить ее работоспособность. Результаты тестирования приведены в таблице ниже:

Таблица 1 - Результаты тестирования программы №1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер теста | Назначение теста | Значения исходных данных | Ожидаемый результат | Реакция программы | Вывод |
| 1 | Проверка единичного отрезка, длина=0 | (0; 0) | 0 | 0 | Правильно |
| 2 | Проверка для случая нескольких отрезков, длина=0 | (0; 0)  (0; 0)  (0; 0) | 0 | 0 | Правильно |
| 3 | Случай с одним отрезком, обе координаты положительные | (5; 11) | 6 | 6 | Правильно |
| 4 | Случай с одним отрезком, обе координаты отрицательные | (-3; -8) | 5 | 5 | Правильно |
| 5 | Случай с одним отрезком, одна координата положительная, другая - отрицательная | (-5; 11) | 16 | 16 | Правильно |
| 6 | Случай с несколькими отрезкам, обе координаты положительные | (1; 3)  (5; 8)  (9; 13) | 9 | 9 | Правильно |
| 7 | Случай с несколькими отрезками, обе координаты отрицательные | (-1;-3)  (-5; -8)  (-9; -13) | 9 | 9 | Правильно |
| 8 | Случай с несколькими отрезками, одна координата положительная, другая - отрицательная | (-1; 3)  (-5; 2)  (-9; 1) | 12 | 12 | Правильно |
| 9 | Отрезки накладываются друг на друга, «лесенкой» | (1; 4)  (2; 7)  (5; 10) | 9 | 9 | Правильно |
| 10 | Отрезки накладываются друг на друга,  Один отрезок «поглощает» другие | (1; 11)  (2; 7)  (5; 10) | 10 | 10 | Правильно |

Затем, мы разработали программу для перевода символьного арифметического выражения в польскую инверсную запись и составили юнит-тесты для проверки правильности работы программы. Результаты тестирования приведены в таблице ниже:

Таблица 2 - Результаты тестирования программы №2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер теста | Назначение теста | Значения исходных данных | Ожидаемый результат | Реакция программы | Вывод |
| 1 | Проверка работы с одним числом | «123» | «123» | «» | Правильно |
| 2 | Проверка работы с простым выражением без скобок | «2+3\*4» | «2 3 4 \* +» | «» | Правильно |
| 3 | Проверка работы с выражением со скобками | «(2+3)\*4» | «2 3 + 4 \*» | «» | Правильно |
| 4 | Проверка работы с выражением, состоящим только из операций | «+\*+» | «\* + +» | «» | Правильно |
| 5 | Проверка работы с пустым выражением | «» | «» | «» | Правильно |
| 6 | Проверка работы с выражением, содержащим пробелы | «2 + 3 \* 4» | «» | «2 3 4 \* +» | Правильно |

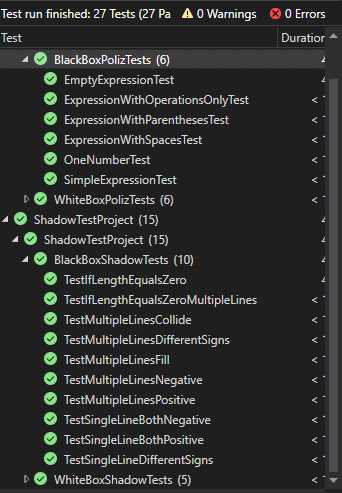


Рис. 1 - Результаты тестирования

**Вывод:**

Недостатки тестирования методом «чёрного ящика»:

- неизвестно, проверили ли все возможные варианты, большая вероятность, что часть кода не протестирована,

- невозможно оценить, насколько эффективно проведено тестирование,

- у сложной программы (алгоритма), чтобы проверить все предполагаемые возможные случаи, надо составлять явно избыточное количество тестов, чем в случаи «белого ящика».

Преимущества:

- тестирование с точки зрения пользователя,

- не требует специальных знаний (алгоритмов, языков программирования),

- тестированием может заниматься любой человек (не разработчик данной программы)

**Листинг кода программы с комментариями**

|  |
| --- |
| Shadow.cs |
| using System.Collections.Generic;  namespace ConsoleApp1  {      public class Line      {          public int X1 { get; set; } // Начальная координата отрезка          public int X2 { get; set; } // Конечная координата отрезка          public Line(int x1, int x2)          {              X1 = x1;              X2 = x2;          }      }      public class Shadow      {          List<Line> lines; // Список для хранения отрезков          public List<Line> Lines { get { return lines; } } // Свойство для доступа к отрезкам          public Shadow()          {              lines = new List<Line>(); // Инициализация списка отрезков          }          public void AddLine(int x1, int x2)          {              Line newLine;              if (x1 < x2) // Проверяем, чтобы первая координата была меньше второй                  newLine = new Line(x1, x2); // Создаем новый отрезок с заданными координатами              else                  newLine = new Line(x2, x1); // Создаем новый отрезок с поменяными координатами              for (int i = 0; i < lines.Count; i++) // Упорядочиваем отрезки по первой координате              {                  if (lines[i].X1 > x1)                  {                      lines.Insert(i, newLine); // Вставляем отрезок на правильную позицию                      return; // Выходим из метода                  }              }                lines.Add(newLine); // Если не найдена подходящая позиция, добавляем отрезок в конец списка          }          public int GetSum()          {              if (lines.Count == 0) return 0; // Если отрезков нет, возвращаем 0              int index = 0; // Номер текущего отрезка              int start = 0; // Начало суммы текущих последовательных отрезков              int end = 0; // Конец суммы текущих последовательных отрезков              int sum = 0; // Переменная для хранения суммы длин отрезков              while (index <= lines.Count) // Перебираем все отрезки              {                  // Если это последний отрезок или есть разрыв между отрезками                  if (index == lines.Count || lines[end].X2 < lines[index].X1)                  {                      sum += Math.Abs(lines[end].X2 - lines[start].X1); // Добавляем длину текущей суммы отрезков                      start = index; // Сбрасываем начальный индекс                      end = index; // Сбрасываем конечный индекс                  }                  else if (lines[end].X2 < lines[index].X2)                  {                      end = index; // Если конец текущего отрезка дальше, обновляем конечный индекс                  }                  index++; // Переходим к следующему отрезку              }              return sum; // Возвращаем сумму длин отрезков          }      }  } |

|  |
| --- |
| Poliz.cs |
| using System.Text;  public class POLIZ  {      public string ConvertToRPN(string expression)      {          List<string> outputList = new List<string>(); // Список для хранения выходной последовательности          Stack<string> stack = new Stack<string>(); // Стек для операций          StringBuilder number = new StringBuilder(); // Строитель для чисел          foreach (var c in expression.Where(c => !char.IsWhiteSpace(c))) // Проходим по каждому символу в выражении, игнорируя пробелы          {              if (char.IsDigit(c)) // Если символ - цифра              {                  number.Append(c); // Добавляем символ к числу              }              else // Если символ - оператор или скобка              {                  if (number.Length > 0) // Если есть накопленное число                  {                      outputList.Add(number.ToString()); // Добавляем число в выходную последовательность                      number.Clear(); // Очищаем число                  }                  if (c == '(') // Если символ - открывающая скобка                  {                      stack.Push(c.ToString()); // Помещаем скобку в стек                  }                  else if (c == ')') // Если символ - закрывающая скобка                  {                      while (stack.Count > 0 && stack.Peek() != "(") // Пока стек не пуст и на вершине стека не открывающая скобка                      {                          outputList.Add(stack.Pop()); // Извлекаем операторы из стека и добавляем их в выходную последовательность                      }                      stack.Pop(); // Извлекаем открывающую скобку из стека                  }                  else // Если символ - оператор                  {                      while (stack.Count > 0 && Priority(c.ToString()) <= Priority(stack.Peek())) // Пока стек не пуст и приоритет текущего оператора меньше или равен приоритету оператора на вершине стека                      {                          outputList.Add(stack.Pop()); // Извлекаем операторы из стека и добавляем их в выходную последовательность                      }                      stack.Push(c.ToString()); // Помещаем текущий оператор в стек                  }              }          }          if (number.Length > 0) // Если есть накопленное число          {              outputList.Add(number.ToString()); // Добавляем число в выходную последовательность          }          while (stack.Count > 0) // Пока стек не пуст          {              outputList.Add(stack.Pop()); // Извлекаем операторы из стека и добавляем их в выходную последовательность          }          return string.Join(" ", outputList); // Возвращаем выходную последовательность, объединенную пробелами      }      public int Priority(string c)      {          if (c == "(")          {              return 0; // Приоритет открывающей скобки          }          else if (c == "+" || c == "-")          {              return 1; // Приоритет сложения и вычитания          }          else if (c == "\*" || c == "/")          {              return 2; // Приоритет умножения и деления          }          else          {              return -1; // Неверный символ          }      }  } |