

Лабораторная работа «Тестирование интеграции»

Цель работы. Получить практические навыки отладки программ с помощью отладчика среды программирования.

Блок Схема



Программа на языке C#

```
1 using System;
2
3 class Program
4 {
5     static void Main(string[] args)
6     {
7         int[,] array1 = GenerateArray(10, 10);
8         int[,] array2 = GenerateArray(10, 10);
9
10        int count1 = CountPositiveOddSumRows(array1);
11        int count2 = CountPositiveOddSumRows(array2);
12
13        Console.WriteLine($"Количество строк с положительной суммой нечетных элементов в первом массиве: {count1}");
14        Console.WriteLine($"Количество строк с положительной суммой нечетных элементов во втором массиве: {count2}");
15
16
17        if (count1 == 0 && count2 == 0)
18        {
19            Console.WriteLine("Нет строк с положительной суммой нечетных элементов ни в одном из массивов.");
20        }
21        PrintArray(array1);
22        Console.WriteLine();
23        Console.WriteLine();
24        PrintArray(array2);
25    }
26
27    static int PrintArray(int[,] array1)
28    {
29        for (int i = 0; i < 10; i++)
30        {
31            for (int j = 0; j < 10; j++)
32            {
33                Console.Write(array1[i, j]);
34                Console.Write(" ");
35            }
36            Console.WriteLine();
37        }
38    }
```

```
39    static int[,] GenerateArray(int rows, int cols)
40    {
41        Random random = new Random();
42        int[,] array = new int[rows, cols];
43
44        for (int i = 0; i < rows; i++)
45        {
46            for (int j = 0; j < cols; j++)
47            {
48                array[i, j] = random.Next(-10, 11);
49            }
50        }
51        return array;
52    }
53
54    static int CountPositiveOddSumRows(int[,] array)
55    {
56        int count = 0;
57        for (int i = 0; i < array.GetLength(0); i++)
58        {
59            int sum = CalculateOddSum(array, i);
60            if (sum > 0)
61            {
62                count++;
63            }
64        }
65        return count;
66    }
67
68    static int CalculateOddSum(int[,] array, int rowIndex)
69    {
70        int sum = 0;
71        for (int j = 0; j < array.GetLength(1); j++)
72        {
73            if (j % 2 != 0)
74            {
75                sum += array[rowIndex, j];
76            }
77        }
78        return sum;
79    }
80 }
81
```

Отладка

52
53
54 `static int CountPositiveOddSumRows(int[,] array)`
55
56 `int count = 0;`
57 `for (int i = 0; i < array.GetLength(0); i++)`
58 `{`
59 `int sum = CalculateOddSum(array, i);`
60 `if (sum > 0)`
61 `{`
62 `count++;`
63 `}`
64 `}`
65 `return count;`
66 `}`
67
68 `static int CalculateOddSum(int[,] array, int rowIndex)`
69

50 % Проблемы не найдены.

Локальные

Поиск (Ctrl+E) Глубина поиска: 3

Имя	Значение	Тип
array	{int[10, 10]}	int[,]
rowIndex	9	int
sum	0	int

Окно интерпретации

Контрольные вопросы

1. Тестирование программы — это процесс проверки, оценки и верификации системы или компонента, с целью выявления возможных ошибок, дефектов и несоответствий требованиям. Оно включает в себя выполнение различных тестовых случаев и анализ результатов.
2. Отладка программы — это процесс поиска и устранения ошибок в коде. Это может включать анализ кода, использование отладочных инструментов и тестирование изменений для гарантии, что ошибки исправлены, а новые не были введены.
 3. Стадии тестирования в разработке ПО могут включать:
 - a. **Системное тестирование:** проверка всей системы как единого целого.
 - b. **Модульное тестирование:** оценка отдельных компонентов или модулей.
 - c. **Интеграционное тестирование:** проверка взаимодействия между модулями.
 - d. **Приемочное тестирование:** валидация системы в соответствии с требованиями пользователя.
 - e. **Регрессионное тестирование:** проверка исправлений на наличие новых ошибок.
4. Тестирование методом **покрытия операторов** направлено на обеспечение выполнения всех операторов программы хотя бы один раз. Оно помогает выявить ошибки в логике и функционировании.
5. Тестирование методом **покрытия решений** проверяет все возможные ветвления в коде, то есть каждая ветвь логического оператора должна быть выполнена хотя бы раз. Это позволяет удостовериться в том, что все логические пути кода протестированы.
6. Тестирование методом **покрытия решений** проверяет все возможные ветвления в коде, то есть каждая ветвь логического оператора должна быть выполнена хотя бы раз. Это позволяет удостовериться в том, что все логические пути кода протестированы.

7. Метод **комбинаторного покрытия условий** гарантирует, что все возможные комбинации условий в логических выражениях проверяются. Это позволяет учитывать взаимодействие различных условий, что может выявить ошибки, возникающие только при определенных их сочетаниях.
8. Метод **комбинаторного покрытия условий** гарантирует, что все возможные комбинации условий в логических выражениях проверяются. Это позволяет учитывать взаимодействие различных условий, что может выявить ошибки, возникающие только при определенных их сочетаниях.
9. Метод **эквивалентных разбиений** состоит в делении входных данных на классы эквивалентности, где данные внутри класса обрабатываются одинаково. Тесты выбираются таким образом, чтобы покрыть каждый класс, что уменьшает количество необходимых тестов.
10. Метод **анализа причинно-следственных связей** помогает идентифицировать и проверить связи между входами и выходами системы. Это позволяет более точно моделировать и тестировать сценарии, которые могут привести к ошибкам, основываясь на логических зависимостях.

Вывод: Получил практические навыки отладки программ с помощью отладчика среды программирования