# Практическая работа №1

**Изучение отладочных возможностей сред программирования**

1. **Цель работы**
   1. Познакомиться с основными инструментами отладки и профилирования в Visual Studio 2022;
   2. Научиться ставить точки останова (breakpoints), использовать пошаговое выполнение кода, инспектировать значения переменных и применять профилировщик для анализа производительности программы.
2. **Литература**
   1. Документация по отладчику — Visual Studio. – Текст : электронный // Microsoft Learn : официальный сайт. – 2024. – URL: <https://learn.microsoft.com/ru-ru/visualstudio/debugger/?view=vs-2022> (дата обращения 18.09.2024).
3. **Подготовка к работе**
   1. Повторить теоретический материал (см. п.2).
   2. Изучить описание лабораторной работы.
4. **Основное оборудование**
   1. Персональный компьютер.
5. **Задание**
   1. Создайте новый проект на языке C# (например, консольное приложение).
   2. Щёлкните на серой полоске слева от строки для добавления точки останова.
   3. Нажмите F5 для запуска программы в режиме отладки. Программа остановится на точке останова.
   4. Используйте пошаговое выполнение для анализа выполнения программы (клавиши F10 и F11).
   5. Откройте окна "Locals" и "Autos", чтобы наблюдать за значениями переменных в процессе выполнения программы.
   6. Остановите отладку и откройте меню "Debug" -> "Performance Profiler" (или "Analyze" -> "Performance Profiler").
   7. В окне профилировщика выберите опцию "CPU Usage" для анализа использования процессора, затем нажмите "Start" для запуска анализа.
   8. Запустите программу с профилировщиком. После завершения анализа остановите выполнение и изучите результаты.
   9. Составить отчет по проделанной работе.
6. **Порядок выполнения работы**
   1. Повторить теоретический материал п. 3.1;
   2. Выполнить анализ и тестирование требований к ПО п. 5.1-5.2;
   3. Ответить на контрольные вопросы п. 8;
   4. Заполнить отчет п. 7.
7. **Содержание отчета**
   1. Титульный лист;
   2. Цель работы;
   3. Ответы на контрольные вопросы п. 6.3;
   4. Вывод по проделанной работе.
8. **Контрольные вопросы**
   1. Как использовать точки останова и что они позволяют сделать?
   2. В чем разница между "Step Over", "Step Into" и "Step Out"?
   3. Как используется окно "Call Stack" для анализа вызовов функций?
   4. Для чего нужен профилировщик в Visual Studio и как его применять?
9. **Приложение**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading;  class Program  {  static void Main(string[] args)  {  // Линейный алгоритм  Console.WriteLine("Начало линейного алгоритма...");  List<int> data = GenerateData(1000000);  int sum = Sum(data);  Console.WriteLine($"Сумма: {sum}");    // Квадратичный алгоритм  Console.WriteLine("Начало квадратичного алгоритма...");  int duplicatesCount = FindDuplicates(data);  Console.WriteLine($"Количество дубликатов: {duplicatesCount}");  // Сортировка данных  Console.WriteLine("Начало сортировки...");  var sortedData = SortData(data);  Console.WriteLine("Сортировка завершена.");  // "Медленный" алгоритм  Console.WriteLine("Начало медленного алгоритма...");  SlowMethod();  Console.WriteLine("Медленный алгоритм завершен.");  Console.WriteLine("Программа завершена.");  }  // Генерация большого массива данных  static List<int> GenerateData(int count)  {  Random random = new Random();  return Enumerable.Range(0, count).Select(x => random.Next(0, 100)).ToList();  }  // Линейный алгоритм: сумма элементов списка  static int Sum(List<int> data)  {  int sum = 0;  for (int i = 0; i < data.Count; i++)  {  sum += data[i];  }  return sum;  }  // Квадратичный алгоритм: поиск количества дубликатов  static int FindDuplicates(List<int> data)  {  int duplicates = 0;  for (int i = 0; i < data.Count; i++)  {  for (int j = i + 1; j < data.Count; j++)  {  if (data[i] == data[j])  {  duplicates++;  break;  }  }  }  return duplicates;  }  // Сортировка данных (быстрая сортировка)  static List<int> SortData(List<int> data)  {  var sorted = data.ToArray();  Array.Sort(sorted);  return sorted.ToList();  }  // "Медленный" метод для профилирования  static void SlowMethod()  {  Thread.Sleep(2000); // Задержка для имитации долгого выполнения  }  } |