

Лабораторная работа №3

Оценка сложности рекурсивных алгоритмов

1Цель работы

1.1 Научиться разрабатывать и оценивать сложность рекурсивных функций в программах на C#.

2Литература

2.1 Фленов, М. Е. Библия C#. – 3 изд. – Санкт-Петербург: БХВ-Петербург, 2016. – URL: <https://ibooks.ru/bookshelf/353561/reading>. – Режим доступа: для зарегистрир. пользователей. – Текст : электронный. – п.3.3.6.

3Подготовка к работе

3.1 Повторить теоретический материал (см. п.2).

3.2 Изучить описание лабораторной работы.

4Основное оборудование

4.1 Персональный компьютер.

5Задание

Для выполнения скопировать проект LabWork3 из папки МДК.01.01.

Для упрощения подсчета количества вызовов рекурсивной функции выводить в консоль отладки информацию, что вызвана рекурсивная функция. Для реализации вывода можно настроить действие у точки останова.

Для тестирования использовать модульные тесты (недостающие тесты дописать).

5.1 Написать и протестировать рекурсивную функцию вычисления факториала. Для некорректных данных возвращать 0.

Оценить сложность разработанного алгоритма.

5.2 Написать и протестировать рекурсивную функцию вычисления x^n , где n – любое целое.

Поиск x^n , где n – отрицательное, осуществляется по формуле:

$$x^n = 1/x^{-n}$$

Стандартный метод возведения в степень не использовать.

Оценить сложность разработанного алгоритма.

5.3 Написать и протестировать рекурсивную функцию быстрого вычисления x^n , где n неотрицательное целое, используя возведение в квадрат.

Для ускорения работы рекурсия должна вызываться в ветке алгоритма не более одного раза.

Пример (вместо 15 операций умножения будет 6 операций умножения):

$$a^{15} = a*(a^7)^2 = a*(a*(a^3)^2)^2 = a*(a*(a*(a^2)^2))^2$$

Для некорректных данных возвращать -1.

Стандартный метод возведения в степень не использовать.

Оценить сложность разработанного алгоритма.

6Порядок выполнения работы

6.1 Запустить MS Visual Studio и использовать решение с названием LabWork3.

6.2 Выполнить все задания из п.5 в решении LabWork3. Для каждого задания создать в консольном проекте в классе Functions рекурсивную функцию в виде статического метода (очень желательно – используя тернарный оператор). Для тестирования использовать:

- вызовы в консоли (с выводом количества вызовов),
- модульное тестирование (доработав набор модульных тестов для корректных и некорректных входных данных).

При выполнении заданий использовать минимально возможное количество команд и переменных и выполнять форматирование и рефакторинг кода.

6.3 Ответить на контрольные вопросы.

7Содержание отчета

7.1 Титульный лист

7.2 Цель работы

7.3 Ответы на контрольные вопросы

7.4 Вывод

8Контрольные вопросы

8.1 Что такое «рекурсия»?

8.2 Какие проблемы могут возникать при реализации рекурсивных алгоритмов на электронных вычислительных машинах?

8.3 Какое определение функции может быть названо рекурсивным? Привести примеры.

8.4 Что такое «глубина рекурсии»?

8.5 Что такое «рекурсивный спуск»?

8.6 Что такое «рекурсивный подъём»?

9Приложение

Рекурсивный вызов метода — это случай, когда метод вызывает сам себя.

Пример:

```
static void MethodName()  
{  
    MethodName();  
}
```

Рекурсивная функция представляет такую конструкцию, при которой функция вызывает саму себя.

Возьмем, к примеру, вычисление факториала, которое использует формулу $n! = 1 * 2 * \dots * n$.

Например, факториал числа 5 равен $120 = 1 * 2 * 3 * 4 * 5$.

Определим метод для нахождения факториала:

```
static int Factorial(int x)
{
    if (x == 0)
    {
        return 1;
    }
    else
    {
        return x * Factorial(x - 1);
    }
}
```

В рекурсивных методах обязательно должен быть предусмотрен выход из метода, иначе произойдет переполнение стека.

При создании рекурсивной функции в ней обязательно должен быть некоторый базовый вариант, который использует оператор `return` и помещается в начале функции. В случае с факториалом это

```
if (x == 0)
    return 1;
```

Все рекурсивные вызовы должны обращаться к подфункциям, которые в конце концов сходятся к базовому варианту. Так, при передаче в функцию положительного числа при дальнейших рекурсивных вызовах подфункций в них будет передаваться каждый раз число, меньшее на единицу. И в конце концов мы дойдем до ситуации, когда число будет равно 0, и будет использован базовый вариант.