# Методические указания по выполнению лабораторной работы.

## Лабораторная работа по теме «Рекурсивные алгоритмы и методы решения задач».

Цель работы: Приобретение навыков написания программ с использованием рекурсивных процедур и функций

Порядок выполнения работы:

1. Изучить теоретический материал.
2. Получить индивидуальное задание и разработать программу в соостветствии с поставленной задачей.
3. Показать работающую программу преподавателю.
4. Ответить на контрольные вопросы.

### Краткие теоретические сведения.

***Рекурсия*** — это такой способ организации вычислительного процесса, при котором процедура или функция в ходе выполнения составляющих ее операторов обращается сама к себе.

Рекурсия достаточно распространённое явление, которое встречается не только в областях науки, но и в повседневной жизни. Например, эффект Дросте, треугольник Серпинского и т. д. Самый простой вариант увидеть рекурсию – это навести Web-камеру на экран монитора компьютера, естественно, предварительно её включив. Таким образом, камера будет записывать изображение экрана компьютера, и выводить его же на этот экран, получится что-то вроде замкнутого цикла. В итоге мы будем наблюдать нечто похожее на тоннель.

В программировании рекурсия тесно связана с функциями, точнее именно благодаря функциям в программировании существует такое понятие как рекурсия или рекурсивная функция. Простыми словами, рекурсия – определение части функции (метода) через саму себя, то есть это функция, которая вызывает саму себя, непосредственно (в своём теле) или косвенно (через другую функцию).  Типичными рекурсивными задачами являются задачи: нахождения n!, числа Фибоначчи. Вообще говоря, всё то, что решается итеративно можно решить рекурсивно, то есть с использованием рекурсивной функции. Всё решение сводится к решению основного или, как ещё его называют, базового случая. Существует такое понятие как шаг рекурсии или рекурсивный вызов. В случае, когда рекурсивная функция вызывается для решения сложной задачи (не базового случая) выполняется некоторое количество рекурсивных вызовов или шагов, с целью сведения задачи к более простой. И так до тех пор пока не получим базовое решение. Разработаем программу, в которой объявлена рекурсивная функция, вычисляющая n!

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | // factorial.cpp: определяет точку входа для консольного приложения.    #include "stdafx.h"  #include <iostream>  using namespace std;    unsigned long int factorial(unsigned long int);// прототип рекурсивной функции  int i = 1; // инициализация глобальной переменной для подсчёта кол-ва рекурсивных вызовов  unsigned long int result; // глобальная переменная для хранения возвращаемого результата рекурсивной функцией    int main(int argc, char\* argv[])  {         int n; // локальная переменная для передачи введенного числа с клавиатуры         cout << "Enter n!: ";         cin >> n;         for (int k = 1; k <= n; k++ )         {          cout << k << "!" << "=" << factorial(k) << endl; // вызов рекурсивной функции         }         system("pause");         return 0;  }    unsigned long int factorial(unsigned long int f) <code class="cpp comments">// рекурсивная функция для нахождения n!  {         if (f == 1 || f == 0) // базовое или частное решение               return 1; // все мы знаем, что 1!=1 и 0!=1         //cout << "Step\t"<< i <<endl;         i++;         //cout <<"Result= "<< result << endl;         result=f\*factorial(f-1); // функция вызывает саму себя         return result;  } |

Вышеприведенный листинг вычисляет факториал рекурсивно в консольном режиме. Современные программы редко выполняются в консольном режиме, поэтому при выполнении работы обязательно использование оконного режима.

При написании рекурсивных методов следует непременно указать в соответствующем месте условный оператор, например if, чтобы организовать возврат из метода без рекурсии. В противном случае возврата из вызванного однажды рекурсивного метода может вообще не произойти. Подобного рода ошибка весьма характерна для реализации рекурсии в практике программирования.

В программе объявлен тип данных unsigned long int, так как значение факториала возрастает очень быстро, например уже 10! = 3 628 800. Если не хватит размера типа данных, то в результате мы получим неверное значение. В коде объявлено больше операторов, чем нужно, для нахождения n!. Это сделано для того, чтобы, отработав, программа показала, что происходит на каждом шаге рекурсивных вызовов.

Рассмотрим ещё одну типичную задачу — нахождение чисел Фибоначчи, используя рекурсию. Далее приведен код рекурсивного решения такой задачи. Вводим в ком строке порядковый номер числа из ряда Фибоначчи, и программа найдёт все числа из ряда Фибоначчи порядковые номера которых меньше либо равны введённому.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | // fibonacci.cpp: определяет точку входа для консольного приложения.    #include "stdafx.h"  #include <iostream>   // библиотека для форматирования выводимой информации на экран  #include <iomanip>  using namespace std;    unsigned long fibonacci(unsigned long);// прототип рекурсивной функции поиска чисел из ряда Фибоначчи    int main(int argc, char\* argv[])  {         unsigned long entered\_number;         cout << "Enter number from the Fibonacci series: ";         cin >> entered\_number;         for (int counter = 1; counter <= entered\_number; counter++ )         cout << setw(2) <<counter << " = "<< fibonacci(counter) << endl; // функция setw() резервирует два места под вывод номера         system("pause");         return 0;  }    unsigned long fibonacci(unsigned long   entered\_number) // функция принимает один аргумент  {         if ( entered\_number == 1 || entered\_number == 2) // частный случай               return (entered\_number -1); // ряд чисел Фибоначчи всегда начинается с 0, 1, ...         return fibonacci(entered\_number-1) + fibonacci(entered\_number-2); // формула поиска n-го числа  } |
|  |  |

В **строке 6** подключена библиотека <iomanip>для того, чтобы воспользоваться функцией setw(), которая в свою очередь выравнивает первый столбец чисел, то есть номера. Как мы можем заметить, сначала числа однозначные от 1 – 9, а потом идут двузначные. Если убрать данную функцию, то произойдет сдвиг влево чисел от 1 и до 9.  Результат работы программы на рисунке 1.

Недостаток рекурсии в том, что она затрачивает значительно больше компьютерных ресурсов, нежели итерация. Это выражается в большой нагрузке, как на оперативную память, так и на процессор. Если очевидно решение той или иной задачи итеративным способом, то им и надо воспользоваться иначе, использовать рекурсию!” В зависимости от решаемой задачи сложность написания программ изменяется при использовании того или иного метода решения. Но чаще задача, решённая рекурсивным методом  с точки зрения читабельности кода, куда понятнее и короче.



Рисунок 1 Результат выполнения программы.

### Задания для выполнения.

1. Задача о Ханойской Башне. Число дисков фиксированно, более 5. Процесс должен быть визуализирован в графическом виде.
2. Задача о Ханойской Башне для произвольного числа дисков, вводимого пользователем. Процесс достаточно визуализировать в текстовом виде.

Задача о Ханойской Башне.

Эта задача известна ещё с древних времён, и суть её состоит в следующем: Имеется три стержня, на один из которых насажены диски разных диаметров. Диаметры дисков упорядочены по убыванию, т. е. в самом низу находится диск с наибольшим диаметром, а у каждого последующего диска диаметр меньше предыдущего. Необходимо поочерёдно переставить диски с одного стержня на другой, используя третий стержень как промежуточный. При этом обязательно должно соблюдаться условие, чтобы диски с диаметром большего размера не располагались над дисками с меньшим диаметром.

1. Задача «Печать числа в двоичном виде». Число вводится пользователем. Программа должна вывести число в двоичном виде. Выполнить подсчет числа вызовов рекурсивной функции, визуализировать процесс на каждом шагу. Для решения задачи отметим, что в двоичной системе счисления числа представлены в виде суммы степеней 2. Подобно тому, как в десятичной системе счисления число 234 означает 2\times10^2 + 3\times10^1 + 4\times10^0, число 101 в двоичной системе означает 1\times2^2 + 0\times2^1 + 1\times2^0. В двоичных числах используются только цифры 0 и 1.
2. Символьная строка содержит выражение вида ‘A@B@C…’, где буквы заменяются на введенные пользователем числа, а вместо любого символа ‘@’ допускается вставить любой из знаков ‘+’,’-‘,’\*’,’/’, либо один (и только один) знак равенства. Написать процедуру подбора расстановки, обеспечивающей точное равенство.
3. Задача о рюкзаке. Есть 10 предметов, о которых известны их веса и стоимости (заданы пользователем). Требуется поместить в рюкзак предметы таким образом, чтобы они не превысили допустимый вес (вводится пользователем) для рюкзака при максимальной стоимости выбранных предметов.
4. Задача о кроликах. Результат решения задачи дожен быть визуализирован на каждом шаге (с помощью графика или последовательно появляющихся чисел). Имеется пара кроликов, каждый месяц приносящая приплод из двух крольчат. Новорожденные крольчата через два месяца уже способны приносить приплод. Необходимо вычислить, сколько кроликов будет через введенное пользователем количество месяцев, если учесть что (выбрать один из двух вариантов):
   1. Кролики не умирают, из воспроизводство не прекращается.
   2. Кролики имеют ограниченную продолжительность жизни.
5. Написать приложение, определяющее является ли введенное пользователем слово или выражение палиндромом. Например: «Аргентина манит негра», «А роза упала на лапу Азора».

Очевидно, что строка из одного символа является палиндромом. Если символов больше, то строка является палиндромом, если выполняются два условия:

* Первый и последний символы строки s совпадают.
* Строка s без первого и последнего символа является палиндромом.

### Контрольные вопросы.

1. Какие программы называют рекурсивными? Что такое рекурсия?
2. В чем преимущества и недостатки использования рекурсии?
3. Что такое граничное условие для программ, использующих рекурсию?
4. В каких типовых задачах применение рекурсии оправдано?
5. Какие проблемы могут возникать при использовании рекурсии на ЭВМ?
6. Сформулируйте тезис Черча-Тьюринга.
7. Для чего нужна визуализация выполнения рекурсивных програм или подпрограмм на каждом шагу? Для чего используются breakpoints (брейкпоинты) в среде разработки?
8. Что такое итерация? В чем отличие рекурсивного метода решения задачи от итерационного?