Лабораторная работа №11мм. Рекурсия

Во всех заданиях **нельзя применять циклы** — пользуйтесь рекурсией (циклы допустимы лишь в функции `main()`).

Решения задач оформлять в виде отдельных функций `recurcia`. В `main` необходимо продемонстрировать работу полученных функций.

0. Найти факториал целого числа. (`double Factorial(int n)`)

   Подсказка

   ```
   Факториал числа n равняется
      1, если n равен нулю или единице,
      n*(n-1)!, если n>1
   ```

   Продемонстрируйте работу подпрограммы в основной программе, выведите на консоль факториалы целых чисел от нуля до десяти.

   Способ решения этой задачи приведен в конце заданий (есть и в лекциях).

Вариант 1

1. Написать рекурсивную подпрограмму `void IntNumbersPrint(int a, int b)`, печатающую все целые числа от a до b включительно (a ≤ b, не забудьте это проверить).

   Подсказка

   ```
   Напечатать числа от a до b –
      напечатать a,
      напечатать числа от a + 1 до b.
   ```

2. Даны два целых числа *a* и *b*. Написать рекурсивную функцию (подпрограмму) `void IntNumbersPrintAD(int a, int b)`, которая выводит все числа от a до b включительно, в порядке возрастания, если $a < b$, или в порядке убывания в противном случае.

Вариант 2

3. Даны два целых числа *a* и *b*. Написать рекурсивную функцию (подпрограмму) `void IntNumbersPrintAscending(int a, int b)`, которая выводит все числа от a до b включительно в порядке возрастания, независимо от того, какое из двух чисел a и b больше.
4. Дано натуральное число *n*. Написать функцию `bool DegreeOfTwo(int a)`, которая возвращает *true*, если число n является точной степенью двойки, или false в противном случае.

Вариант 3

5. Даны целые числа *a* и *b*. Найти их наибольший общий делитель (`int GreatestCommonDivisor(int a, int b)`), используя *алгоритм Евклида*:

   НОД(A, B) = НОД(B, A % B), если B ≠ 0;
   НОД(A, 0) = A.

6. Создать функцию `void PrimeFactors(int n, int k = 2)`, выводит на консоль все простые множители положительного числа (единица и само исходное число не учитываются, используем цикл).

Вариант 4

7. Дан список целых чисел. Вычислить сумму его элементов (`int ListSum(IList<int> list, int n = 0)`), пользуясь следующим рекурсивным определением:

   Подсказка

   ```
   Сумма элементов =
       первый элемент
       +
       сумма всех остальных элементов.
   ```

8. Создать функцию `int DigProduct(int n)`, которая возвращает произведение цифр числа.

Вариант 5

9а. Дано натуральное число. Составить рекурсивную функцию вывода на экран цифр заданного натурального числа, определить количество и сумму его цифр.

10. Реализуйте алгоритм вычисления суммы с использованием рекурсии

$$y(x,i)=\sin x+\sin(\sin x)+\ldots+\sin(\sin\ldots(\sin x)),$$

9б. Задана последовательность положительных чисел, признаком конца которых служит отрицательное число. Используя рекурсию, подсчитать количество чисел и их сумму.

Отдельно остановимся на двух рекурсивных функциях.

Возьмем, к **примеру, вычисление факториала**, которое использует формулу n! = 1 * 2 * … * 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);
    }
}
```

Итак, здесь у нас задается условие, что если вводимое число не равно 0, то мы умножаем данное число на результат этой же функции, в которую в качестве параметра передается число x-1. То есть происходит рекурсивный спуск. И так, пока не дойдем того момента, когда значение параметра не будет равно единице.

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;`.

И, кроме того, 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.

Теперь рассмотрим второй пример на определение чисел Фибоначчи

Другим распространенным показательным примером рекурсивной функции служит функция, вычисляющая числа Фиббоначчи. n-й член последовательности Фибоначчи определяется по формуле: f(n)=f(n-1) + f(n-2), причем f(0)=0, а f(1)=1. То есть последовательность Фибоначчи будет выглядеть так 0 (0-й член), 1 (1-й член), 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ... Для определения чисел этой последовательности определим следующий метод:

```
1
2 static int Fibonachi(int n)
3 {
4     if (n == 0)
5     {
6         return 0;
7     }
8     else if (n == 1)
9     {
10        return 1;
11    }
12    else
13    {
14        return Fibonachi(n - 1) + Fibonachi(n - 2);
15    }
16}
17
```

Либо, если сократить первые две условные конструкции, так:

```
1
2 static int Fibonachi(int n)
3 {
4     if (n == 0 || n == 1)
5     {
6         return n;
7     }
8     else
9     {
10        return Fibonachi(n - 1) + Fibonachi(n - 2);
11    }
12}
13
```

Это простейшие пример рекурсивных функций, которые призваны дать понимание работы рекурсии. В то же время для обоих функций вместо рекурсий можно использовать циклические конструкции. И, как правило, альтернативы на основе циклов работают быстрее и более эффективны, чем рекурсия. Например, вычисление чисел Фибоначчи с помощью циклов:

```
static int Fibonacci(int n)
{
    int a = 0;
    int b = 1;
    int tmp;

    for (int i = 0; i < n; i++)
    {
        tmp = a;
        a = b;
        b += tmp;
    }

    return a;
}
```

В то же время в некоторых ситуациях рекурсия предоставляет элегантное решение, например, при обходе различных древовидных представлений, к примеру, дерева каталогов и файлов

Достоинства рекурсии:

· простота, наглядность и компактность записи алгоритмов, по природе своей являющихся рекурсивными;

· рекурсивная процедура показывает, что надо делать, а нерекурсивная – как надо делать;

· легко почти дословно программировать вычисления по рекуррентным формулам;

· легко доказывается корректность программы – её эквивалентность тем формулам, по которым решается задача;

Задания

25.Напишите рекурсивную функцию, перемножающую два целых числа, одно из которых неотрицательно, без использования операции умножения.

Контрольные вопросы

1. Какой процесс называется итеративным?

2. Каким образом реализуются итеративные процессы?

3. Какой алгоритм называется рекурсивным?

4. Какая функция называется прямо рекурсивной? Косвенно рекурсивной?

5. Что такое стек? В какой последовательности происходит заполнение стека и выбор элементов из стека?

6. Что должно обязательно присутствовать в теле рекурсивно описанной функции?

7. Перечислите различия между итерацией и рекурсией.

8.Что произойдет, если рекурсивный алгоритм будет вызывать сам себя «бесконечное» число раз?

9. Как предотвратить бесконечное выполнение рекурсивного алгоритма?

10. Верно ли, что решение задачи, реализуемое рекурсивным алгоритмом, можно выразить, используя итерацию?