Лабораторная работа № 14

«Разработка, отладка и испытание алгоритмов и программ с использованием рекурсии»

В языке C++ функция может из своего тела вызывать сама себя по имени. В этом случае такая функция называется рекурсивной. Рекурсия — это процесс определения чего-либо на основе самого себя, из-за чего рекурсию еще называют рекурсивным определением. Когда функция вызывает сама себя, новый набор локальных переменных и параметров размещается в оперативной памяти в стеке, а код функции выполняется с самого своего начала, причем используются уже новые значения переменных. При рекурсивном вызове функции новая копия ее кода НЕ создается. Новыми являются только значения, которые использует данная функция. При каждом возвращении из рекурсивного вызова старые локальные переменные и параметры извлекаются из стека, и сразу за рекурсивным вызовом возобновляется работа функции. При использовании рекурсивных функций стек работает подобно "телескопической" трубе, выдвигающейся вперед и складывающейся обратно.

Примером рекурсивной функции является функция factorial(), которая вычисляет факториал целого неотрицательного числа. Факториалом числа n (обозначается n!) называется произведение всех целых чисел, от 1 до n включительно (для 0, по определению, факториал равен 1). Например, 3! — это 1×2×3, или 6. Здесь показаны функция factorial() и эквивалентная ей функция, в которой используются итерации:

/* Рекурсивная функция */

```
#include <iostream>
#include <Windows.h>//подключение библиотеки для поддержки русского языка
using namespace std;
unsigned long long int factorial(unsigned long long int);//прототип рекурсивной функции рассчета факториала

int main()
{
    SetConsoleOutputCP(1251);//функция для поддержки вывода букв алфавита русского языка
    SetConsoleCP(1251);//функция для поддержки ввода букв алфавита русского языка
    unsigned long long int a;//поскольку факториал рассчитывается для неотрицательных чисел, то можно взять тип unsigned long long int
    bool f;//декларация логической переменной-флага для определения момента прекращения цикла do-while
    do
    {
        cout << "Вычислить факториал числа: ";//запрашиваем значение для рассчета факториала
        cin >> a;//помместить значение, введеннное пользователем в переменную a
        cout << factorial(a) << endl;//вызываем функцию factorial(), в которую передаем значение a, для которого функция должна рассчитать факториал
        //и вернуть результат, который окажется на месте вызова функции factorial() и будет распечатан, поскольку окажется в выражении cout<<
        cout << "Вычислить факториал еще раз (1 - Да, 0 - Нет)? ";//запрашиваем пользователя о продолжении программы
        cin >> f;//если пользователь введет 0 - завершится цикл и вслед за этим завершится программа, если введет 1 - цикл продолжится и можно будет
        //ввести новое число и функция factorial() рассчитает новый факториал числа, находящегося в переменной a
    }
    while (f==true);//цикл с постусловием (постпроверкой) работает 1 итерацию в любом случае, а дальше итерации будут выполняться, если пользователь
    //будет вводить 1, которая приводится к логическому значению ИСТИНА(Да, true)
    system("pause");
    return 0;
}
unsigned long long int factorial(unsigned long long int x)//определение функции рассчета факториала для входного аргумента x
{
    if (x < 2)//факториал для отрицательных значений рассчитываться не должен, а факториал 0! равен 1 и 1! равен 1
    {//поэтому для этих значений возвращаем из функции 1 (эта проверка и возвращаемое значение являются окончанием рекурсивного вызова, предохраняют
        return 1;// от бесконечного рекурсивного вызова
    }//также это является "дном" двунаправленной рекурсии
    else//если x равен 2 или больше
    {
        return x * factorial(x - 1);//то умножить данное число на вызов функции factorial()'а с аргументом x-1 (следующее, меньшее на единицу, число и так далее)
    }//из тела функции factorial() вызывается одноименная функция factorial() с такой же сигнатурой, значит функция factorial() - это рекурсивная функция
}
```

Тестируем рекурсивную функцию на неотрицательных значениях до 20:

```
D:\2019\Labs\x64\Debug\Lab4_7.exe
Вычислить факториал числа: 5
120
Вычислить факториал еще раз (1 - Да, 0 - Нет)? 1
Вычислить факториал числа: 3
6
Вычислить факториал еще раз (1 - Да, 0 - Нет)? 1
Вычислить факториал числа: 4
24
Вычислить факториал еще раз (1 - Да, 0 - Нет)? 0
Для продолжения нажмите любую клавишу . . .
```

```
int fact(int n)//НЕрекурсивная функция. Расчет факториала числа в цикле рассматривался в лабораторных ранее
{
        int t, answer;
        answer = 1;
        for(t = 1; t <= n; t++)
        {
                answer = answer * t;
        }
        return(answer);
}
```

Пример использования двух рекурсивных функций, причем обе – безвозвратные, однонаправленные:

```cpp
#include <iostream>
using namespace std;
void printStar(int);//прототип функции
void printRowStars(int);//прототип функции
//задача: распечатать перевернутый треугольник из заданного количества "звездочек" *, используя рекурсивные функции
int main()
{
    int a;
    do
    {
        cout << "Enter a: ";//запрашиваем количество строк, которое будет равняться и количеству "звездочек"
        cin >> a;//пользователь вводит требуемое ему количество "звездочек" и рядов треугольника из "звездочек"
        printRowStars(a);//вызываем функцию для печати треугольника, а из этой функции будет вызываться функция печати одной строки "звездочек"
    }
    while (a!=0);//запрашивать и печатать "треугольники" до тех пор, пока пользователь не введет 0
    system("pause");
    return 0;
}
void printStar(int x)//определение рекурсивной функции printStar()
{
    if (x < 0)
    {
        x = -x;
        cout << "The sign of X is changed.\n";
    }
    if (x == 0)//это условие окончания рекурсивных вызовов
    {
        cout << endl;//завершить печать строки "звездочек" переносом курсора на следующую строку
        return;//завершаем рекурсию
    }
    else//если x больше нуля
    {
        cout << '*';//то напечатать одну "звездочку"
        return printStar(x - 1);//и вызвать эту же самую функцию, но с параметром на единицу меньшим, поскольку 1 "звездочку" мы только что напечатали
    }//из тела функции printStar() вызывается функция printStar() - значит, функция printStar() - рекурсивная
}
void printRowStars(int y)//определение рекурсивной функции printRowStars()
{
    if (y < 0)
    {
        y = -y;
        cout << "The sign of Y is changed.\n";
    }
    if (y == 0)//это условие окончания рекурсивных вызовов
    {
        return;//завершаем функцию, которая ничего не возвращает (void), таким выражением
    }
    else//если y больше нуля,
    {
        printStar(y);//то надо напечатать строку "звездочек", для чего вызываем функцию printStar(y) и передаем ей требуемое количество "звездочек" для печати на консоль
        y = y - 1;//уменьшаем число звездочек и строк, ведь одну строку "звездочек" уже напечатали на консоль вызовом функции printStar(y)
        printRowStars(y);//у уменьшили на единицу в строке выше, поэтому в вызов printRowStars(y) передастся уже уменьшенный y
    }//из тела функции printRowStars() вызывается функция printRowStars() - значит, функция printRowStars() - рекурсивная, причем рекурсивная безвозвратная, однонаправленная
}
```

Тестируем:


D:\2019\Labs\x64\Debug\Lab4_8.exe
Enter a: 5
*****
****
***
**
*
Enter a: -15
The sign of Y is changed.
****
****
****
****
****
****
****
****
****
****
*****
****
***
**
*
Enter a: 3
***
**
*
Enter a: 1
*
Enter a: 0
Для продолжения нажмите любую клавишу . . .

9.1. Понятие рекурсии

Решить задачу рекурсивно – это значит разложить ее на подзадачи, которые затем аналогичным образом (т. е. рекурсивно) разбиваются на еще меньшие подзадачи. На определенном уровне подзадачи становятся настолько простыми, что могут быть решены тривиально.

В рекурсивном алгоритме важно предусмотреть способ его остановки, т.е. ввести условие, при котором рекурсивное обращение к функции прекращается.

9.2. Пример выполнения работы

Условие 1. Написать программу для вычисления $S = \sum_{i=1}^{n} (i+1)^2 / i$ двумя методами. Один метод вычисляет сумму без использования рекурсии, другой – с использованием рекурсии.

```
#include <iostream.h>
#include <math.h>
   double sum(int);
   double sumr(int);
int main ()
{
int n;
cout << "vvedite n ";  cin >> n;
cout << "s (ne rekurs) = " << sum(n) << endl;
cout << "s (rekurs) = " << sumr(n) << endl;
return 0;
}

double sum(int n)
{
    for (double s=0, int i=1; i<=n; i++) s += (pow(i+1,2))/i;
 return s;
}

double sumr(int n)
{
if (n==1) return 4;
  else return sumr(n-1)+pow(n+1,2)/n;
}
```

Условие 2. Найти max (a_1, ..., a_n), разбив задачу на **элементарные подзадачи:** max(max (max (a_1...a_{n-2}), a_{n-1}), a_n), … .

```
int maxr2(int i)
{
if (i==0) return a[0];
else{
  int mx=maxr2(i-1);
        if (a[i]>mx)  return a[i];
              else return mx;
}
}
```

Условие 3. Задача о Ханойской башне. Имеется три стержня s_1, s_2, s_3. На первом из них нанизаны n дисков различных диаметров, образующих правильную пирамиду – чем выше расположен диск, тем меньше его диаметр. Требуется переместить всю башню на второй стержень, причем диски можно переносить по одному, нельзя помещать диск на диск меньшего диаметра, для промежуточного хранения можно использовать третий диск.

```
void hanr(int n, int s1, int s2, int s3)
{
   if (n>0) {
   hanr(n-1,s1,s3,s2);
       cout << "perenesty s "<<s1<<" na "<<s2<<endl;
   hanr(n-1,s3,s2,s1);
     }
}
```

Порядок выполнения работы

Изучить теоретические сведения к лабораторной работе.

ЗАДАЧА 1

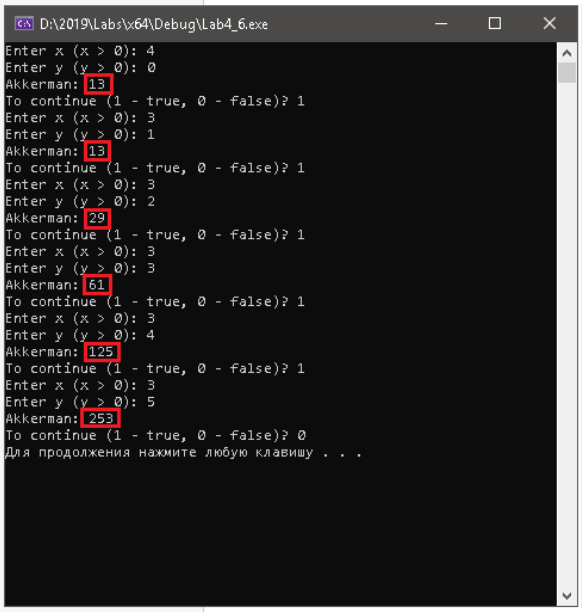
Реализовать алгоритм решения задачи. Написать рекурсивную функцию для вычисления значения функции Аккермана для неотрицательных чисел n и m.

$$A(m,n) = \begin{cases} n+1 & m = 0 \\ A(m-1,1) & m > 0, n = 0 \\ A(m-1, A(m,n-1)) & m > 0, n > 0 \end{cases}$$

```cpp
#include <iostream>
using namespace std;
int Akkerman(int, int);//прототип функции Аккермана

int main()//функцию Аккермана возовем в основной функции main() с фактическими (реальными) параметрами
{
    int x, y;
    bool f = false;//переменная-флаг, для определения продолжения или прекращения цикла
    do
    {
        cout << "Enter x (x > 0): ";
        cin >> x;
        cout << "Enter y (y > 0): ";
        cin >> y;
        int z = Akkerman(x, y);//функция Аккермана принимает вводимые пользователем данные и возвращает целочисленный результат, помещаемый в переменную x
        cout << "Akkerman: " << z << endl;//печатаем результат, полученный от вызванной выше функции Аккермана
        cout << "To continue (1 - true, 0 - false)? ";//спрашиваем пользователя, хочет ли он продолжить, т.е. ввести снова тестовые данные для функции Аккермана
        cin >> f;//пользователь вводит значение 0 или 1, которое помещается в логическую переменную (0 станет ЛОЖЬЮ(false, НЕТ), а 1 - ИСТИНОЙ(true, ДА)
    }
    while (f == true);//цикл выполняется, пока пользователь будет вводить значения, приводимые к значению ИСТИНА(ДА, true)
    system("pause");
    return 0;
}
int Akkerman(int m, int n)//определение функции Аккермана
{
    if (m < 0)//по заданию m неотрицательное, поэтому меняем знак для отрицательного m
    {
        m = -m;
    }
    if (n < 0)//по заданию n неотрицательное, поэтому меняем знак для отрицательного n
    {
        n = -n;
    }
    if (m == 0)//по заданию, если m равно нулю, то функция возвращает n+1
    {
        return n + 1;
    }
    if (m > 0)//если m положительное, то нужно дальше проверить n
    {
        if (n == 0)//если n равно нулю
        {
            Akkerman(m - 1, 1);//то вызовем функцию Аккермана с параметрами (m - 1, 1)
        }
        if (n > 0)//если n больше нуля
        {
            Akkerman(m - 1, Akkerman(m, n - 1));//то вызовем функцию Аккермана с параметрами (m - 1, Akkerman(m, n - 1))
        }//то есть второй параметр вызываемой функции является вызовом функции Аккермана с параметрами (m, n - 1))
    }//из тела функции Аккермана вызываются функции Аккермана, поэтому данная функция рекурсивная
}
```

Запустим программу для тестирования и сравним получаемые значения с таблицей рассчитанных значений. Чтобы подчеркнуть, что **фактические** параметры, передаваемые функции при ее вызове в main'е **НЕ обязаны** совпадать с именами **формальных** параметров в определении функции (за main'ом), мы запросили в main'е значения x и y, которые функция Akkerman(x, y) примет и будет считать, соответственно x как m, а y как n. Красными рамками выделены совпавшие значения в таблице-«эталоне» (https://ru.wikipedia.org/wiki/Функция_Аккермана) и рассчитанные нашей программой. Совпадут и значения, находящиеся левее выделенных красным цветом прямоугольников. А вот значения, находящиеся правее красных прямоугольников программе получить сложнее, поскольку возникает ошибка переполнения стека, в который заносятся промежуточные результаты, получаемые функцией Akkerman(). Хотя математически доказано, что функция Аккермана определена для всех положительных значений и гарантированно рано или поздно по времени вернет результат, но количество рекурсивных вызовов таково, что для их хранения не хватает размера стандартного стека (размер стека можно в коде задать и больший).



$n\|m$	0	1	2	3	4	5	m
0	1	2	3	5	13	65533	$\mathrm{hyper}(2,\ m,\ 3) - 3$
1	2	3	5	13	65533	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{65536} - 3$	$\mathrm{hyper}(2,\ m,\ 4) - 3$
2	3	4	7	29	$2^{65536} - 3$	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{65536}} - 3$	$\mathrm{hyper}(2,\ m,\ 5) - 3$
3	4	5	9	61	$2^{2^{65536}} - 3$	$A(4,\ \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{65536}} - 3)$	$\mathrm{hyper}(2,\ m,\ 6) - 3$
4	5	6	11	125	$2^{2^{2^{65536}}} - 3$	$A(4,\ A(5,\ 3))$	$\mathrm{hyper}(2,\ m,\ 7) - 3$
5	6	7	13	253	$2^{2^{2^{2^{65536}}}} - 3$	$A(4,\ A(5,\ 4))$	$\mathrm{hyper}(2,\ m,\ 8) - 3$
n	$n+1$	$n+2$	$2n+3$	$2^{n+3} - 3$	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3} - 3$	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{\vdots\ \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{65536}} - 3$ (всего n блоков $2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$)	$\mathrm{hyper}(2,\ m,\ n+3) - 3$

Отметим, что функцию Аккермана можно написать и с использованием тернарного оператора (точнее, тернарных операторов, вложенных друг в друга), причем сам тернарный оператор поместим в выражение с return. В тернарном операторе в самом конце справа нужно написать выражение, которое выполнится, если все проверки дали ЛОЖЬ (false), для чего специально используется значение –1 как заведомо не подходящее, поскольку значения m и n должны быть неотрицательными по условию задания.

```
int AkkermanTernarny(int m, int n)//функция Аккермана с использованием тернарных операторов, вложенных друг в друга
{
    return m == 0 ? n + 1 : (m > 0 && n == 0 ? AkkermanTernarny(m - 1, 1) : (m > 0 && n > 0 ? AkkermanTernarny(m - 1, AkkermanTernarny(m, n - 1)) : -1));
}
```

Разработать на языке C++ программу вывода на экран решения задачи в соответствии с вариантом индивидуального задания, указанным преподавателем.

Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.

Контрольные вопросы

Что такое рекурсия?
Как оформляется рекурсивная функция?
Какие опасности и сложности возможны при написании, использовании и усовершенствовании (поддержке другими программистами) рекурсивных функций?
Какие сильные стороны у рекурсивного подхода? Какие задачи решаются только с помощью рекурсии?
Могут ли рекурсивные вызовы функции продолжаться бесконечно?
Как должна быть оформлена рекурсивная функция, чтобы количество рекурсивных вызовов было конечным?
Что такое прямая и косвенная рекурсии?
Что такое рекурсия с возвратом (обратная, двунаправленная) и рекурсия без возврата (односторонняя, невозвратная)?

ЗАДАЧА 2

В каждом из 28-и вариантов требуется написать пользовательскую **рекурсивную** функцию и **вызвать ее** в функции main(), отвечающей за интерфейс взаимодействия с пользователем:

№	Задача
1	Написать рекурсивную функцию вычисления суммы цифр натурального числа.
2	Написать рекурсивную функцию для расчета степени n вещественного числа a (n – натуральное число).
3	Для данного N вычислить значение выражения, используя рекурсию: $P = \sqrt{7+\sqrt{14+\sqrt{21+\cdots+\sqrt{7N}}}}$
4	Написать рекурсивную функцию нахождения цифрового корня натурального числа. Цифровой корень данного числа получается следующим образом. Если сложить все цифры этого числа, затем все цифры найденной суммы и повторять этот процесс, то в результате будет получено однозначное число (цифра), которая и называется *цифровым корнем* данного числа.
5	Даны первый член и разность арифметической прогрессии. Написать рекурсивную функцию для нахождения: а) n-го члена прогрессии; б) суммы n первых членов прогрессии.
6	Даны первый член и знаменатель геометрической прогрессии. Написать рекурсивную функцию: а) нахождения n-го члена прогрессии; б) нахождения суммы n первых членов прогрессии.

7	Вычислить значение выражения, используя рекурсивный метод: P $=\sqrt{1+\sqrt{2+\sqrt{3+\cdots+\sqrt{n-1+\sqrt{n}}}}}$
8	Написать рекурсивную функцию для вычисления максимального элемента массива из *n* элементов.
9	Написать рекурсивную функцию для вычисления индекса максимального элемента массива из *n* элементов.
10	Написать рекурсивную функцию для вывода на экран цифр натурального числа в обратном порядке.
11	Написать рекурсивную функцию для ввода с клавиатуры последовательности чисел и вывода ее на экран в обратном порядке (окончание последовательности – ввод пользователем *нуля*).
12	Написать рекурсивную процедуру перевода натурального числа из десятичной системы счисления в двоичную.
13	Написать рекурсивную функцию, определяющую, является ли заданное натуральное число простым.
14	Описать рекурсивную функцию C(m, n) целого типа, находящую число сочетаний из n элементов по m, используя формулу: C(0, n) = C(n, n) = 1, C(m, n) = C(m, n–1) + C(m–1, n–1) при 0 < m < n (m и n – целые параметры; n > 0, 0 <= m <= n). Дано число N и пять различных значений M. Вывести числа C(M, N) вместе с количеством рекурсивных вызовов функции C, потребовавшихся для их нахождения.

15	Вычислить значение выражения, используя рекурсивный метод: y $=\sqrt{1+\sqrt{4+\sqrt{9+\cdots+\sqrt{81+\sqrt{100}}}}}$
16	Написать программу с рекурсивной функцией, вычисляющей разность цифр заданного натурального числа n.
17	Для данного N вычислить значение выражения, используя рекурсию: P = $2+\sqrt{4+\sqrt{6+\sqrt{\ \ldots+\ \ \sqrt{2*N}}}}$
18	Описать функцию, которая удаляет из строки все лишние пробелы. Пробелы считаются лишними, если их подряд идет более двух, если они стоят в конце строки после последней точки, если стоят после открывающегося парного знака препинания.
19	Дано n различных натуральных чисел. Напечатать все перестановки этих чисел.
20	Задано положительное и отрицательное число в двоичной системе. Составить программу вычисления суммы этих чисел, используя функцию сложения чисел в двоичной системе счисления.
21	Написать программу с рекурсивной функцией, вычисляющей: S$=\sqrt{3+\sqrt{3+\sqrt{3+\cdots}}}$ используется n корней.

22	Описать рекурсивную функцию логического типа, проверяющую, является ли симметричной строка *S* (является ли она палиндромом).
23	Описать рекурсивную функцию целого типа, находящую количество цифр в строке *S* без использования оператора цикла. С помощью этой функции найти количество цифр в строках, вводимых пользователем с клавиатуры.
24	Описать рекурсивную функцию вещественного типа, находящую приближенное значение корня k-й степени из числа x по формуле: y(0) = 1, y(n+1) = y(n) – (y(n) – x / y(n)k–1) / k, где y(n) обозначает SqrtK(x, k, n) (x – вещественный параметр, k и n – целые; x > 0, k > 1, n > 0). С помощью этой функции найти приближенные значения корня K-й степени из X при 6 различных значениях N для данных X и K.
25	Написать программу с рекурсивной функцией, вычисляющей: $A = \sqrt{1+\sqrt{8+\sqrt{27+\cdots\sqrt{n}}}}$
26	Написать рекурсивную функцию, определяющую, является ли целое число палиндромом (палиндром – число, читаемое слева направо и справа налево одинаково; отрицательное число не считать палиндромом; одноразрядное число считать палиндромом).
27	Дано n различных натуральных чисел. Напечатать все возможные варианты расстановки данных чисел друг относительно друга.
28	Написать рекурсивную функцию, возводящую вещественное число X в целочисленную степень Y.

9.3. Индивидуальные задания

Решить задачу двумя способами – с применением рекурсии и без нее.

1. Вычислить среднее значение элементов одномерного массива.
2. Вычислить произведение элементов одномерного массива.
3. Подсчитать количество цифр в заданном числе.
4. В упорядоченном массиве целых чисел a_i, $i = 1 \ldots n$ найти номер элемента c методом бинарного поиска, используя очевидное соотношение: если $c \le a_{n/2}$, тогда $c \in [a_1 \ldots a_{n/2}]$, иначе $c \in [a_{n/2+1} \ldots a_n]$. Если элемент c отсутствует в массиве, то вывести соответствующее сообщение.
5. Найти наибольший общий делитель чисел M и N используя метод Эйлера: если M делится на N, то НОД (N, M) = N, иначе НОД (N, M) = = НОД (M % N, N).
6. Вычислить значение полинома степени n по формуле

$$P_n = \sum_{i=0}^{n} a_i x^i = a_0 + x\left(a_1 + \ldots x\left(a_{n-1} + x a_n\right)\ldots\right).$$

7. Вычислить значение $x=\sqrt{a}$, используя формулу $x_n=\frac{1}{2}\left(x_{n-1}+a/x_{n-1}\right)$, в качестве начального приближения использовать значение $x_0=(1+a)/2$.

8. Найти максимальный элемент в массиве $a_1, \ldots, a_n$, используя метод деления пополам $\max(a_1, \ldots, a_n) = \max(\max(a_1, \ldots, a_{n/2}), \max(a_{n/2+1}, \ldots, a_n))$.

9. Вычислить $y(n)=\sqrt{1+\sqrt{2+\ldots+\sqrt{n}}}$.

10. Вычислить произведение $n \geq 2$ (n четное) сомножителей

$$y=\frac{2}{1}\cdot\frac{2}{3}\cdot\frac{4}{3}\cdot\frac{4}{5}\cdot\frac{6}{5}\cdot\frac{6}{7}\cdot\ldots .$$

11. Вычислить $y=x^N$ по следующему алгоритму: $y=(x^{N/2})^2$, если N четное; $y=x\cdot x^{N-1}$, если N нечетное.

12. Вычислить

$$y(n)=\cfrac{1}{n+\cfrac{1}{(n-1)+\cfrac{1}{(n-2)+\cfrac{1}{\ddots}}}}$$

$$\ldots.+\cfrac{1}{1+\cfrac{1}{2}}.$$

13. Вычислить произведение двух целых положительных чисел $p=a\cdot b$ по следующему алгоритму: $p=2\cdot(a\cdot b/2)$, если b четное; $p=a+(a\cdot(b-1))$, если b нечетное. Если $b=0$, то $p=0$.

14. Вычислить значение суммы $S = 1/1! + 1/2! + \ldots + 1/k!$

15. Проверить, является ли заданная строка палиндромом.

Домашнее задание:

Прочитать и освоить раздел «Рекурсивные алгоритмы» из методички на сервере: s1 / Предметы / ОАиП_Шаляпин / Литература / МетодичкаЧасть1.pdf (страницы 76 – 80).