Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Владимирский государственный университет   
имени Александра Григорьевича и Николая Григорьевича Столетовых»  
(ВлГУ)

Институт прикладной математики, физики и информатики

Кафедра физики и прикладной математики

ЛАБОРАТОРНАЯ РАБОТА №2  
по дисциплине  
«Основы программирования»  
на тему:  
«Реализация рекурсивных алгоритмов на С ++»

Выполнил:  
ст. гр. ПМИ-123

Рушев А.М.  
  
  
Принял:  
ст. преподаватель   
каф. ФиПМ  
Шишкина М.В.

Владимир,2024

**Цель работы**

Получение навыка разработки рекурсивных алгоритмов, написания рекурсивных функций на С++, закрепление навыков создания функций на языке С++.

**Постановка задачи**

1. Реализовать на языке С++ вычисление факториала целого неотрицательного числа, суммы элементов одномерного целочисленного массива.
2. Для каждой задачи реализовать рекурсивный и циклический алгоритмы.
3. Реализовать с использованием рекурсивной функции алгоритм быстрой сортировки.
4. Выполнить одно задание в соответствии со своим вариантом.

Вывести рекуррентное соотношение и реализовать функцию, выводящую на экран следующие последовательности, количество членов последовательности задать входным параметром.

1) 1, 2, 3, 4, ...

2) 0, 5, 10, 15, ...

3) 1, 1, 1, 1, ...

4) 1, -1, 1, -1, ...

5) 1, -2, 3, -4, 5, -6, ...

6) 2, 4, 8, 16, ...

7) 2, 4, 16, 256, ...

8) 0, 1, 2, 3, 0, 1, 2, 3, 0, ...

9) 1!, 3!, 5!, 7!, ...

10) 1, a, a2, a3, a4, a5,....

11) 1,1+а,1+a+a2,+1+a+a2+a3,...

12) 1 + 𝑎/1! + 𝑎2/2!+ 𝑎3/3!+ ⋯,

13) a^(m+1)

**Теоретическая часть**

Функция называется рекурсивной если в теле этой функции встречается вызов самой себя. Такой случай называется прямой рекурсией.

Рекурсия называется косвенной, если вызов функции осуществлён через вызов другой функции в теле, которой происходит вызов первой функции. При вызове функции, в стеке создаётся копия значений её параметров, после чего управление передаётся первому исполняемому оператору функции. При повторном вызове процесс повторяется.

Любой рекурсивный алгоритм может быть заменён на алгоритм, реализованный через цикл.

Пример вычисления факториала

int factorial1(int n) {

if (n < 2) {return 1;}

else {return n \* factorial1(n - 1);}

}

int factorial2(int n) {

int S = 1;

while (n > 1) {

S \*= n;

n--;

}

return S;

}

Пример нахождения суммы элементов массива.

int SumMas1(int p, int\* mas) {

if (p == 0) {return mas[p];}

else {

return mas[p] + SumMas1(p - 1, mas);

}

}

int SumMas2(int p, int\* mas) {

int S = 0;

for (int i = 0; i < p; i++) {

S += mas[i];

}

return S;

}

Рекуррентное соотношение

Рекуррентное отношение — это уравнение, рекурсивно определяющее последовательность, каждый член которой является функцией предыдущих членов.

xi=f(xi-1)

Например, для последовательности 1,2,3,4,5…., рекуррентное соотношение будет записано так:

xi=xi-1+1, начальное значение - x1=1.

Для последовательности 1, -1, 1, -1, 1,-1 xi=xi-1\*(-1), начальное значение x1=1.

Запишем рекуррентное соотношение для n!

n!=1\*2\*3\*…\*n;

n!=(n-1)!\*n, тогда xn=xn-1\*n, x0=1.

Алгоритм быстрой сортировки.

1. Выбрать опорный элемент в массиве. Элемент может быть любым, например, центральным.
2. Разделить массив на две части следующим образом: все элементы из левой части, большие или равные опорному, переносим в правую, все элементы из правой, которые меньше или равны опорному переносим в левую часть.
3. В результате предыдущего шага в левой части массива останутся элементы, которые меньше или равны центральному, а в правой — больше либо равны. Наглядно это можно показать таким образом:

| mas[i] <= mid | mid = mas[size/2] | mas[i] >= mid |

1. Рекурсивно повторяем действие для левой и правой части массива, до тех пор, пока размер обоих частей больше единицы.

Разберём алгоритм быстрой сортировки (сортировки Хоара) подробнее.

Результативность алгоритма не зависит от выбора опорного элемента, однако скорость сортировки увеличивается если опорный элемент делит массив на две примерно равные части.

Определение опорного элемента может быть произведено, например, так:

*M= mas [(left+right)/2]*, где

*left – номер крайнего левого элемента в массиве;*

*right - номер крайнего правого элемента в массиве;*

*M* получает значение номера опорного элемента;

Следующий этап – перемещение элементов, меньших опорного в часть левее него, больших опорного в часть справа него.

Для этого необходимо найти элементы, стоящие не в своей части и обменять их местами с элементами, стоящими не на своих местах в другой части.

Например, таким образом можно найти номер первого элемента, который необходимо переместить из левой части в правую.

*while (mas[f] < mid) f++;*

После завершения цикла в переменной *f* будет сохранен номер соответствующего элемента.

Далее необходимо повторить описанные выше шаги рекурсивно для левой и правой части.

Границы частей будут заданы крайним левым и крайним правым индексами, и номерами элементов, определяющими часть элементов меньших опорного и больше опорного.

if (first < l) Q\_S(mas, first, l);

if (f < last) Q\_S(mas, f, last);

Выход из рекурсии осуществляется при невыполнении соответствующих условий.

**Практическая часть**

**Листинг кода программы:**

#include <iostream>

using namespace std;

int RecursionSum(int\* arr, int n) {

if (n <= 0) {

return 0;

}

else {

return arr[n - 1] + RecursionSum(arr, n - 1);

}

return 0;

}

int RecursionFactorial(int f) {

if (f == 1) {

return 1;

}

else {

return RecursionFactorial(f - 1) \* f;

}

}

int CycleSum(int\* arr, int n) {

int result = 0;

for (int i = 0; i < n; i++) {

result += arr[i];

}

return result;

}

int CycleFactorial(int f) {

int factorial = 1;

for (int i = 1; i < f + 1; i++) {

factorial \*= i;

}

return factorial;

}

void Recursion\_Qsort(int\* arr, int start, int end) {

int pivot; // разрешающий элемент

int index; // индекс разрешающего элемента

int l\_hold = start; //левая граница

int r\_hold = end; // правая граница

pivot = arr[start];

while (start < end) // пока границы не сомкнутся

{

while ((arr[end] > pivot) && (start < end))

{

end--; // сдвигаем правую границу пока элемент [right] больше [pivot]

}

if (start != end) // если границы не сомкнулись

{

arr[start] = arr[end]; // перемещаем элемент [right] на место разрешающего

start++; // сдвигаем левую границу вправо

}

while ((arr[start] < pivot) && (start < end))

{

start++; // сдвигаем левую границу пока элемент [left] меньше [pivot]

}

if (start != end) // если границы не сомкнулись

{

arr[end] = arr[start]; // перемещаем элемент [left] на место [right]

end--; // сдвигаем правую границу влево

}

}

arr[start] = pivot; // ставим разрешающий элемент на место

index = start;

start = l\_hold;

end = r\_hold;

if (start < index) // Рекурсивно вызываем сортировку для левой и правой части массива

{

Recursion\_Qsort(arr, start, index - 1);

}

if (end > index)

{

Recursion\_Qsort(arr, index + 1, end);

}

}

void PrintMassiv(int\* arr, int n) {

for (int i = 0; i < n; i++) {

cout << arr[i] << "\t";

}

}

int f(int n) {

if (n == 0) {

return 1;

}

return f(n - 1) \* n;

}

//f(n) = f(n - 1) + 1

void Task1(int n) {

if (n == 1) {

cout << n;

}

else {

Task1(n - 1);

cout << ", " << n;

}

}

//f(n) = f(n-1) + 5, f(1) = 0

void Task2(int n, int x = 0) {

if (n == 0) {

return;

}

else {

cout << x << ", ";

Task2(n - 1, x + 5);

}

}

//f(n) = f(n - 1)

void Task3(int n) {

if (n <= 0) {

return;

}

Task3(n - 1);

cout << 1 << " ";

}

int Task4\_1(int n) {

if (n < 1) {

return 1;

}

else {

return Task4\_1(n - 1) \* (-1);

}

}

//f(n) = f(n) \* (-1)

void Task4(int n) {

if (n <= 0) {

return;

}

Task4(n - 1);

cout << Task4\_1(n - 1) << " ";

}

//f(n) = 1 f(x) = (f(x – 1) + 1) \* (-1) ^ x)

void Task5(int n) {

if (n != 0) {

Task5(n - 1);

if (n % 2 != 0) {

cout << n << ", ";

}

else {

cout << -n << ", ";

}

}

}

//f(n) = f(n - 1) \* 2

void Task6(int n, int s = 2) {

if (n == 0) {

return;

}

cout << s << ", ";

Task6(n - 1, s \* 2);

}

int Task7\_1(int n) {

if (n < 1) {

return 2;

}

else {

return Task7\_1(n - 1) \* Task7\_1(n - 1);

}

}

//f(n) = f(n - 1) \* f(n - 1)

void Task7(int n) {

if (n <= 0) {

return;

}

Task7(n - 1);

cout << Task7\_1(n - 1) << " ";

}

int Task8\_1(int n, int k = 0) {

if (n < 1) {

return 0;

}

else {

if (Task8\_1(n - 1) + 1 != 4) {

return Task8\_1(n - 1) + 1;

}

else {

return 0;

}

}

}

//f(n-1)+1 != 4 то f(n)=f(n-1)+1;

void Task8(int n) {

if (n <= 0) {

return;

}

Task8(n - 1);

cout << Task8\_1(n - 1) << " ";

}

//f(n) = f(n - 1) \* 2n \* (2n + 1)

void Task9(int n) {

if (n != 0) {

Task9(n - 1);

cout << f((2 \* n) - 1) << ", ";

}

}

//f(n) = f(n - 1) \* a

void Task10(int n, int i, int k) {

if (i < n) {

if (k < 2) {

cout << k << "\t";

Task10(n, i + 1, k + 1);

}

else {

cout << "a^" << i << "\t";

Task10(n, i + 1, k);

}

}

}

int Task11\_1(int n, int a, int k = 1) {

if (n < 1) {

return 1;

}

else {

int s = a;

for (int i = 0; i < k; i++) {

s \*= a;

}

return Task11\_1(a, n - 1, k + 1) + s;

}

}

//f(n) = f(n - 1) + a^n

void Task11(int n, int a) {

if (n <= 0) {

return;

}

Task11(n - 1, a);

cout << Task11\_1(n - 1, a) << " ";

}

//

void Task12(int a, int n, double res = 1, float f = 1, float s = 1, int i = 1) {

if (n == 0) {

return;

}

else {

cout << res << ", ";

s \*= a;

f \*= i;

res += (s / f);

Task12(a, n - 1, res, f, s, i + 1);

}

}

int Task13\_1(int n, int a, int m) {

if (n < 1) {

int s = 1;

for (int i = 0; i < m + 1; i++) {

s \*= a;

}

return s;

}

else {

return Task13\_1(n - 1, a, m) \* a;

}

}

//f(n) = f(n - 1) \* a

void Task13(int n, int a, int m) {

if (n <= 0) {

return;

}

Task13(n - 1, a, m);

cout << Task13\_1(n - 1, a, m) << " ";

}

int main()

{

setlocale(LC\_ALL, "rus");

const int n = 6;

int arr[n];

int f, start, end;

int a, m;

start = 0;

end = n - 1;

cout << "Введите число, для вычисления факториала: ";

cin >> f;

//Заполнение массива

for (int i = 0; i < n; i++) {

arr[i] = 1 + rand() % 50;

}

cout << "Вычисление факториала в цикле: "<< CycleFactorial(f) << endl;

cout << "Вычисление суммы в цикле: "<< CycleSum(arr, n) << endl;

cout << "Вычисление факториала при помощи рекурсии: " << RecursionFactorial(f) << endl;

cout << "Вычисление суммы при помощи рекурсии: " << RecursionSum(arr, n) << endl;

//сортировка массива

Recursion\_Qsort(arr, start, end);

cout << endl;

//вывод массива

PrintMassiv(arr, n);

int k, i;

cout << "\nВведите значение i: ";

cin >> i;

cout << "Введите значение k: ";

cin >> k;

cout << "Введите число a: ";

cin >> a;

cout << "Введите число m: ";

cin >> m;

//индивидуальное задание № 1

cout << "\nВывод задания 1: ";

Task1(n);

cout << endl;

//индивидуальное задание № 2

cout << "Вывод задания 2: ";

Task2(n);

cout << endl;

//индивидуальное задание № 3

cout << "Вывод задания 3: ";

Task3(n);

cout << endl;

//индивидуальное задание № 4

cout << "Вывод задания 4: ";

Task4(n);

cout << endl;

//индивидуальное задание № 5

cout << "Вывод задания 5: ";

Task5(n);

cout << endl;

//индивидуальное задание № 6

cout << "Вывод задания 6: ";

Task6(n);

cout << endl;

//индивидуальное задание № 7

cout << "Вывод задания 7: ";

Task7(n);

cout << endl;

//индивидуальное задание № 8

cout << "Вывод задания 8: ";

Task8(n);

cout << endl;

//индивидуальное задание № 9

cout << "Вывод задания 9: ";

Task9(n);

cout << endl;

//индивидуальное задание № 10

cout << "Вывод задания 10: ";

Task10(n, i, k);

cout << endl;

//индивидуальное задание № 11

cout << "Вывод задания 11: ";

Task11(n, a);

cout << endl;

//индивидуальное задание № 12

cout << "Вывод задания 12: ";

Task12(a, n);

cout << endl;

//индивидуальное задание № 13

cout << "Последовательность a^(m+1) для a = " << a << " и для m = " << m << ":\n";

Task13(n, a, m);

cout << endl;

system("pause");

return 0;

}

**Результат выполнения кода программы:**

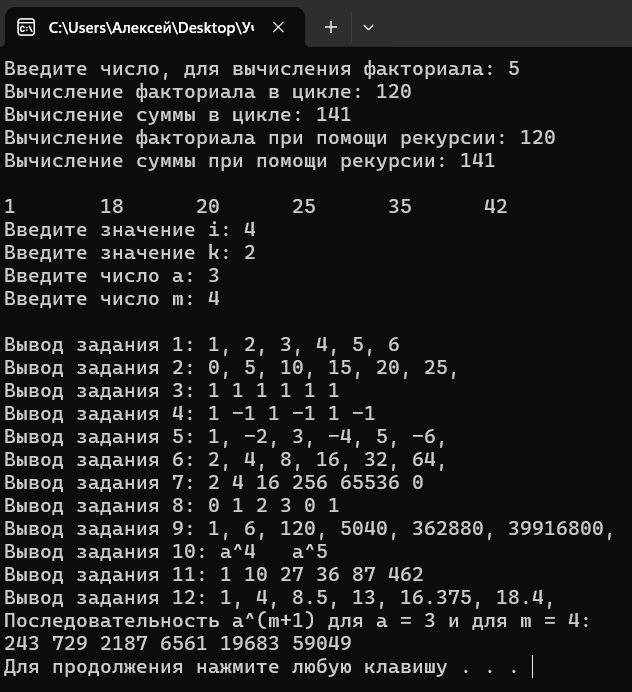


Рисунок 1 – Результат выполнения кода программы

**Вывод:** в ходе лабораторной работы получены навыки разработки рекурсивных алгоритмов и написание рекурсивных функций на С++, они являются важным этапом для углубленного изучения языка программирования. Знание рекурсивных алгоритмов позволяет более эффективно решать различные задачи, а также улучшает навыки анализа и реализации сложных программ.

Недостаток - скорость работы рекурсивного кода. Некоторые компиляторы некоторых языков программирования могут преобразовывать хвостовую рекурсию в цикл, но, в целом, итеративные реализации работают быстрее практически во всех случаях, когда использование циклов НЕ усложняет алгоритм, также недостатком является нужда в памяти, то, что рекурсия задействует большее значение памяти.

Другой недостаток - требования к размеру стека, но это решается настройками компиляции или выбором языка, который не ограничивает размер стека.

Преимущество - некоторые алгоритмы многократно проще, короче и понятнее записать в рекурсивном виде, чем через циклы.

Кроме того, были закреплены навыки создания функций на языке С++, которые помогают улучшить качество и эффективность программного кода.