**Лабораторная работа 12. Функции, простая рекурсия**

**Рекурсивный вызов** — вызов **функции** (процедуры) из неё же самой, непосредственно (простая **рекурсия**) или через другие **функции** (сложная или косвенная **рекурсия**)

Рекурсия, при которой рекурсивные вызовы на любом рекурсивном срезе, инициируют не более одного последующего рекурсивного вызова, называется линейной. Это наиболее простой и часто встречающийся тип рекурсии.   
Если в функции по меньшей мере в одной рекурсив­ной ветви встречаются два или более рекурсивных вызова, то говорят о нелинейной, или параллельной, рекурсии.

**Примеры линейной рекурсии.**

// Пример 1 – рекурсивная функция вычисления n!

int factorial(int n) {

if (n <= 1) {

return 1;

}

return n \* factorial(n - 1);

}

int main() {

std::cout << factorial(5);

return 0;

}

// Пример 2 – вывод всех цифр целого числа в обратном порядке

void printDigitsReverse(int n) {

if (n < 10) {

std::cout << n;

return;

}

Std::cout << n % 10;

printDigitsReverse(n / 10);

}

int main() {

printDigitsReverse(1234);

return 0;

}

// Пример 3 – нахождение суммы элементов массива

int sumArray(int\* a, int n) {

if (n == 0)

return a[n];

return a[n] + sumArray(a, n - 1);

}

int main() {

int a[]{ 5, 2, 3, 6, 4 };

std::cout << sumArray(a, size(a) - 1);

return 0;

}

**Примеры каскадной (параллельной) рекурсии**

// Пример 4 – вычисление n-го числа Фибоначчи

int fibonacci(int n)

{

if (n == 0)

return 0; // базовый случай (условие завершения)

if (n == 1)

return 1; // базовый случай (условие завершения)

return fibonacci(n - 1) + fibonacci(n - 2);

}

int main() {

std::cout << fibonacci(8);

return 0;

}

// Пример 5 – бинарный поиск

int binarySearch(int key, int\* a, int left, int right) {

if (left > right) return -1;

int middle = (right + left) / 2;

if (key == a[middle]) {

return middle;

}

if (key < a[middle]) {

return binarySearch(key, a, left, middle - 1);

}

else {

return binarySearch(key, a, middle + 1, right);

}

}

int main() {

int a[]{ 5, 12, 34, 69, 84, 175 }; //!!! отсортировано

std::cout << binarySearch(34, a, 0, size(a) - 1);

return 0;

}

**Задачи для самостоятельной работы (не использовать циклы)**

1. Ввести целое число n. Рекурсивной функцией вывести на экран все числа от 1 до n и их квадраты.

2. Ввести последовательность чисел (окончание ввода – 0) и вывести их же в обратном порядке.

3. Написать и протестировать рекурсивные функции для   
а) вычисления количества цифр в заданном числе  
b) вычисления суммы цифр в десятичной записи заданного числа.  
c) вывода двоичного представление введенного с клавиатуры целого числа (рекурсивный метод, с выводом одного разряда двоичного числа).

4. Написать протестировать рекурсивные методы   
а) сложения двух чисел, используя только прибавление единицы  
б) умножения двух чисел, используя только операцию сложения.

5. Вычислить, используя рекурсию, выражения

а)  b) 

(в выражении присутствуют ровно n радикалов) Ответ: 3 (с точностью eps)

6. Написать и протестировать рекурсивную функцию вычисления биномиального коэффициента по формуле: image

7. Написать и протестировать рекурсивную функцию в которой методом деления отрезка пополам (методом дихотомии) найти с точностью eps корень уравнения f(x) = 0 на отрезке [a,b] (считать верным, что *eps > 0, a < b, f(a)·f(b) < 0*).   
**Функцию реализовать для следующих уравнений:**  
а) х sin x – 0,5 = 0; a = 0, b = π   
b) lg(x2 – 3x +2) = 0; a = 2.1, b = 5 c) ; a = π, b = 2π

***Метод дихотомии:***Если *f(a)* и *f(b)* имеют разные знаки, то между точками *a* и *b* существует корень *R*.   
Пусть – средняя точка в интервале   
Если *f(m) = 0*, то корень *R=m*.   
Если нет, то либо *f(a)* и *f(m)* имеют разные знаки , либо *f(m)* и *f(b)* имеют разные знаки.   
Если, то корень лежит в интервале    
В противном случае он лежит в интервале   
Затем это действие выполнить для нового интервала – половины исходного интервала.   
Процесс продолжается до тех пор, пока интервал не станет меньше eps.

8. Создать рекурсивные функции для работы с массивами, протестировать их на подходящих примерах

**с линейной рекурсией (один рекурсивный вызов)**

Для проверки того, что в одномерном массиве (метод должен возвращать bool)

а) есть положительные элементы

b) нет отрицательных элементов

c) есть хоть один элемент, больший заданного значения

Для поиска в одномерном массиве

d) максимального элемента

e) количества положительных элементов

Для сортировки одномерного массива  
 f) методом поиска

**с каскадной (параллельной) рекурсией (несколько рекурсивных вызовов)**

Для поиска в одномерном массиве

g) максимального элемента

h) суммы всех отрицательных элементов   
i) произведения тех элементов, квадрат которых меньше заданного числа W