Лабораторная работа №9
Тема: «Рекурсия»

Цель работы: сформировать знания и умения по работе с подпрограммами, приобрести навыки написания программ с использованием рекурсивных функций.

Время выполнения: 4 часа.

Теоретические сведения

Рекурсивные функции - это функции, которые вызывают сами себя. Такие функции довольно часто используются для обхода различных представлений. Например, если нам надо найти определенный файл в папке, то мы сначала смотрим все файлы в этой папке, затем смотрим все ее подпак

Например, определим вычисление факториала в виде рекурсивной функции:

```
#include <iostream>

unsigned long long factorial(unsigned);

int main()
{
   unsigned n {5};
   auto result { factorial(n)};
   std::cout << "Factorial of " << n << " is equal to " << result << std::endl;
}

unsigned long long factorial(unsigned n)
{
   if(n > 1)
      return n * factorial(n-1);
   return 1;
}
```

В функции factorial задано условие, что если число n больше 1, то это число умножается на результат этой же функции, в которую в качестве параметра передается число n-1. То есть происходит рекурсивный спуск. И так далее, пока не дойдем того момента, когда значение параметра не будет равно 1. В этом случае функция возвратит 1.

Рекурсивная функция обязательно должна иметь некоторый базовый вариант, который использует оператор return и к которому сходится выполнение остальных вызовов этой функции. В случае с факториалом базовый вариант представлен ситуацией, при которой n = 1. В этом случае сработает инструкция return 1;.

Например, при вызове factorial(5) получится следующая цепь вызовов:

5 * factorial(4)

5 * 4 * factorial(3)

5 * 4 * 3 * factorial(2)

5 * 4 * 3 * 2 * factorial(1)

5 * 4 * 3 * 2 * 1

Другим распространенным показательным примером рекурсивной функции служит функция, вычисляющая числа Фиббоначчи. n-й член последовательности чисел Фибоначчи определяется по формуле: f(n)=f(n-1) + f(n-2), причем f(0)=0, а f(1)=1. Значения f(0)=0 и f(1)=1, таким образом, определяют базовые варианты для данной функции:

```
#include <iostream>

unsigned long long fib(unsigned);

int main()
{
   for(unsigned i{}; i < 10; i++)
   {
      auto n = fib(i);
      std::cout << n << "\t";
   }
   std::cout << std::endl;
}
unsigned long long fib(unsigned n)
{
   if (n == 0)
      return 0;
```

```
    if (n == 1)
        return 1;
    return fib(n - 1) + fib(n - 2);
}
```

Результат работы программы - вывод 10 чисел из последовательности Фиббоначчи на консоль:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

В примерах выше функция напрямую вызывала саму себя. Но рекурсивный вызов функции также может быть косвенным. Например, функция fun1() вызывает другую функцию fun2(), которая, в свою очередь, вызывает fun1(). В этом случае функции fun1() и fun2() также называются взаимно рекурсивными функциями.

Стоит отметить, что нередко рекурсивные функции можно представить в виде циклов. Например, для вычисления факториала вместо рекурсии используем цикл:

```
#include <iostream>

unsigned long long factorial(unsigned);

int main()
{
    unsigned n {6};
    std::cout << "Factorial of " << n << " : " << factorial(n) << std::endl;
}

unsigned long long factorial(unsigned n)
{
    unsigned long long result{1};
    for(unsigned i{1}; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

И нередко циклические конструкции более эффективны, чем рекурсия. Например, если функция вызывает себя тысячи раз, потребуется большой объем стековой памяти для хранения копий значений аргументов и адреса возврата для каждого вызова, что может привести к тому, что программа быстро исчерпает выделенную для нее память стека, поскольку объем памяти стека обычно фиксирован и ограничен. что может привести к аварийному завершению программы. Поэтому в таких случаях, как правило, лучше использовать альтернативные подходы, например, цикл. Однако, несмотря на накладные расходы, использование рекурсии часто может значительно упростить написание программы.

Индивидуальные задания к лабораторной работе №9

1. Вычислить значение выражения, используя рекурсивный метод: y= $\sqrt{1+\sqrt{2+\sqrt{3+\ldots+\sqrt{n-1+\sqrt{n}}}}}$

2. Для данного N вычислить значение выражения, используя рекурсию: P= $\sqrt{2+\sqrt{4+\sqrt{6+\ldots+\sqrt{2N}}}}$

3. Написать программу с рекурсивной функцией, вычисляющей разность элементов одномерного массива.

4. Написать рекурсивную функцию сложения целых чисел двумерного массива.

5. Написать рекурсивную процедуру, которая считывает вводимые с клавиатуры числа до тех пор, пока не будет обнаружен нуль. Затем введенные числа распечатываются в обратном порядке.

6. Написать рекурсивную процедуру, которая считывает вводимые с клавиатуры числа до тех пор, пока не будет обнаружен нуль. Затем введенные числа распечатываются в обратном порядке. Нуль не печатать.

7. Написать рекурсивную процедуру, переводящую целое число из десятичной системы счисления в восьмеричную.

8. Написать рекурсивную процедуру, переводящую целое число из десятичной системы счисления в шестнадцатеричную.

9. Написать рекурсивную функцию для поиска максимального элемента в одномерном массиве.

10. Написать программу с рекурсивной функцией, вычисляющей количество цифр заданного натурального числа n.

11. Написать программу с рекурсивной функцией, вычисляющей сумму цифр заданного натурального числа n.

12. Написать программу с рекурсивной функцией, вычисляющей сумму элементов одномерного массива.

13. Написать программу с рекурсивной функцией, вычисляющей:

$s=\sqrt{3+\sqrt{3+\sqrt{3+\ldots}}}$.

14. Написать программу с рекурсивной функцией, вычисляющей: $s=\sqrt{1+\sqrt{2+\sqrt{3+\ldots+\sqrt{n}}}}$

15. Вычислить значение выражения, используя рекурсивный метод: y=$\sqrt{1+\sqrt{4+\sqrt{9+\ldots+\sqrt{81}}}}$

16. Для данного N вычислить значение выражения, используя рекурсию: P=$\sqrt{7+\sqrt{14+\sqrt{21+\ldots+\sqrt{7N}}}}$

17. Написать программу с рекурсивной функцией, вычисляющей количество цифр заданного натурального числа n.

18. Написать рекурсивную функцию для поиска минимального элемента в одномерном массиве.

19. Написать рекурсивную функцию умножения вещественных чисел.

20. Составить программу определения является ли введенное число простым с использованием рекурсивной функции.

21. Написать рекурсивную процедуру, которая считывает вводимые с клавиатуры числа до тех пор, пока не будет обнаружен нуль. Затем введенные числа распечатываются в обратном порядке. Нуль тоже печатать.

22. Написать рекурсивную процедуру, переводящую целое число из восьмеричной системы счисления в десятичную.

23. Написать рекурсивную процедуру, которая считывает вводимые с клавиатуры числа до тех пор, пока не будет обнаружена единица. Затем введенные числа распечатываются в обратном порядке.

24. Написать рекурсивную процедуру, переводящую целое число из шестнадцатеричной системы счисления в десятичную.

25. Написать рекурсивную функцию вычитания целых чисел.

26. Написать программу с рекурсивной функцией, вычисляющей: $s=\sqrt{1+\sqrt{8+\sqrt{27+\ldots+\sqrt{n^3}}}}$

27. Написать программу с рекурсивной функцией, вычисляющей произведение цифр заданного натурального числа n.

28. Написать программу с рекурсивной функцией, вычисляющей произведение элементов одномерного массива.

29. Написать программу с рекурсивной функцией, вычисляющей разность цифр заданного натурального числа n.

30. Написать рекурсивную функцию, вычисляющую указанное число Фибоначчи. Последовательность Фибоначчи задается следующими соотношениями: $a_0=a_1=1;\ a_i=a_{i-1}+a_{i-2}\ (i\ge 2)$.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретическую часть лабораторной работы.

2. Реализовать индивидуальное задание по вариантам, представленные в теоретических сведениях, сделать скриншоты работающих программ. Написать комментарии.

3. Написать отчет, содержащий:

1. Титульный лист, на котором указывается:

а) полное наименование министерства образование и название учебного заведения;

б) название дисциплины;

в) номер практического занятия;

г) фамилия преподавателя, ведущего занятие;

д) фамилия, имя и номер группы студента;

е) год выполнения лабораторной работы.

2. Индивидуальное задание из раздела «Теоретические сведения» с кодом, комментариями и скриншотами работающих программ.

3. Построение блок-схем.