## Лабораторная работа №2

Тема: Подпрограммы в языке С++.

Цель: освоение составления и тестирования алгоритмов и программ обработки данных с использованием функций пользователя.

Теоретические положения

### Структура сложной программы

Любая программа на языке высокого уровня может быть разбита на ряд логически завершенных программных единиц - подпрограмм. Такое разделение вызвано двумя причинами.

1. *Экономия памяти.* Каждая подпрограмма записывается в программе один раз, в то время как обращаться к ней можно многократно из разных точек программы.
2. *Структурирование программы.* Алгоритм решения задачи может быть достаточно сложным, поэтому целесообразно выделить самостоятельные смысловые части алгоритма и оформить их в виде подпрограмм.

В языке С++ существует один вид подпрограмм, который называется функция. Каждая программа должна иметь главную функцию (main), которая служит точкой входа в программу. Кроме главной функции в программе может быть оформлено произвольное число функций.

Описание подпрограммы само по себе никаких действий не вызывает. При запуске программы выполнение начинается с операторов главной функции main(). Чтобы выполнить подпрограмму, в нужной точке главной функции необходимо записать обращение к подпрограмме.

### Определение подпрограммы-функции в С++

Функция – это автономная часть программы, реализующая определенный алгоритм и допускающая обращение к ней из различных частей программы.

Каждая функция по отношению к другим является внешней.

Ниже схематично приведена структура программы, в которой описана подпрограмма-функция.

#include "stdafx.h"

//Раздел описаний функций

тип имя\_функции(тип имя\_параметра\_1, тип имя\_параметра\_2,...)

 {

 тело функции

 }

main() //начало главной функции

 {

 Обращение к подпрограмме:

 ……………………..

 } //конец главной функции

Для того чтобы функция могла быть вызвана, т.е. была доступна, необходимо, чтобы до ее вызова о ней было известно компилятору. Это значит, что либо текст функции должен быть расположен до функции, из которой она вызывается, либо перед функцией записывается прототип функции, а сама функция – в конце программы. Прототип функции по форме аналогичен заголовку функции, в конце которого ставится ";".

### Общий вид описания функции

Описание функции содержит заголовок со списком формальных параметров и тело функции:

Тип Имя(список формальных параметров)

{

 Описание локальных переменных;

 Операторы тела функции;

 return результат;

}

Тип, указываемый в заголовке функции определяет тип результата ее работы, который будет возвращаться в точку вызова. Если тип не указан, то по умолчанию подразумевается *int* (целый). Для возврата значения в теле функции должен быть оператор *return*. В дальнейшем будем называть такую функцию *типизированной*.

Если функция не должна возвращать результат, то она считается *нетипизированной*, что задается ключевым словом *void,* стоящим на месте типа. В этом случае оператор *return* в функции не требуется.

void Имя(список формальных параметров)

{

 Описание локальных переменных;

 Операторы тела функции;

}

Параметры, указанные в заголовке функции, называются *формальными*. Список формальных параметров обеспечивает передачу исходных данных в функцию.

Параметры, указываемые при ее вызове, называются *фактическими*.

Формальные и фактические параметры должны быть согласованы друг с другом по количеству, типу и порядку следования. Это означает, что количество формальных параметров должно быть равно количеству фактических параметров, и каждый формальный параметр должен иметь тот же тип и занимать в списке то же место, что и соответствующий ему фактический параметр.

### Обращение к функции

Обращение к *нетипизированной* функции оформляется как отдельный оператор: <имя функции>(<список фактических параметров>); В этом случае при выполнении данного оператора управление будет передано в тело функции, а после ее выполнения – к следующему за вызовом функции оператору вывивающей программы.

Обращение к *типизированной* функции не является специальным оператором, а включается в состав выражения. Результат выполнения функции возвращается в основную программу через имя функции. Обращение к функции записывается аналогично записи стандартной функции (например, sin(x), ехp(x) и т.п.) в виде операнда <имя функции>(<список фактических параметров>). При вычислении выражения операнд обращения к функции заменяется возвращаемым значением функции.

**Пример**: функция вычисляет и возвращает максимальное значения из двух заданных значений, представленных параметрами функции.

#include "stdafx.h"

**int max(int a, int b)**

 **{**

 **int c;**

 **if (a>b)**

 **c=a;**

 **else**

 **c=b;**

 **return c;**

**}**

void main()

{ int x,y,z;

 cout<<"Введите x и y:";

 cin>>x>>y;

 z=max(x,y);

 cout<<"max="<<z;

 }

**Задание.**

1. **Самостоятельно изучить возможности языка программирования C++ для описания функций пользователя и обработки данных с использованием функций пользователя.**
2. **Разработать программу на языке C++, реализующую обработку последовательности согласно индивидуальному заданию из табл. 2.1:**
* **ввод элементов последовательности с клавиатуры и вывод результатов их обработки должны осуществляться в функции main();**
* **обработка элементов последовательности должна осуществляться при помощи функции пользователя, необходимые элементы должны передаваться в пользовательскую функцию в качестве параметров;**
* **если в задании используется число *M*, то его значение следует вводить с клавиатуры в функции main(), и передавать в пользовательскую функцию в качестве параметра.**
* Таблица 2.1. Варианты индивидуальных заданий:

|  |  |
| --- | --- |
|  | Определить количество простых чисел в последовательности |