# 59-60 ОАП Пр13 Организация функций.

## Теоретические сведения

**Определение функции. Обращение к функции.** В С++ используется лишь один тип подпрограмм — функция. Здесь вообще не принято употреблять термин «подпрограмма», потому что функция является основной программной единицей в С++, минимальным исполняемым программным модулем. Всякая программа обязательно включает в себя основную функцию с именем main. Если в программе используются и другие функции, то они выполняют роль подпрограмм.

Рассмотрим пример. Требуется составить программу нахождения наибольшего значения из трех величин — max (а, b, с). Для ее решения можно использовать вспомогательный алгоритм нахождения максимального значения из двух, поскольку справедливо равенство: max (а, b, с) = max (max (a, b), с).

Вот программа решения этой задачи с использованием вспомогательной функции.

#include <iostream>  
**using namespace** std;  
//Определение вспомогательной функции   
**int** \_max(**int** х, **int** у)  
{  
 **return** x > y ? x : y;   
}  
//Основная функция  
**int** main (**int** argc, **char**\*\* argv)  
{   
 **int** a,b,c,d;  
 cout<<*"Введите a,b,с:"*;  
 cin >> a >> b >> c;  
 d =\_max(\_max(a,b),c);  
 cout <<*"\nmax(a,b,c) = "* << d;  
}

Формат определения функции следующий:

тип имя\_\_функции (спецификация\_параметров) {тело\_\_ функции}

Тип функции — это тип возвращаемого функцией результата. Если функция не возвращает никакого результата, то для нее указывается тип **void**.

Имя функции — идентификатор, задаваемый программистом или main для основной функции.

Спецификации параметров — это либо «пусто», либо список имен формальных параметров функции с указанием типа для каждого из них.

Тело функции — это либо составной оператор, либо блок.

Здесь мы впервые встречаемся с понятием блока. Признаком блока является наличие описаний программных объектов (переменных, массивов и т.д.), которые действуют в пределах этого блока. Блок, как и составной оператор, ограничивается фигурными скобками.

В С++ действует правило: тело функции не может содержать в себе определения других функций. Из всякой функции возможно обращение к другим функциям, однако они всегда являются внешними по отношению к вызывающей.

Оператором возврата из функции в точку ее вызова является оператор **return**. Он может использоваться в функциях в двух формах:

**return**; или **return** выражение;

В первом случае функция не возвращает никакого значения в качестве своего результата. Во втором случае результатом функции является значение указанного выражения. Тип этого выражения должен либо совпадать с типом функции, либо относиться к числу типов, допускающих автоматическое преобразование к типу функции.

Оператор **return** может в явном виде отсутствовать в теле функции. В таком случае его присутствие подразумевается перед закрывающей тело функции фигурной скобкой. Такая подстановка производится компилятором.

Формат обращения к функции (вызова функции) традиционный:

имя\_функции(список\_фактических\_праметров)

Однако в С++ обращение к функции имеет своеобразную трактовку: обращение к функции — это выражение. В этом выражении круглые скобки играют роль знака операции, для которой функция и фактические параметры (аргументы) являются операндами. Приоритет операции «скобки» самый высокий (см. табл. 4.2), поэтому вычисление функции в выражениях производится раньше других операций.

Между формальными и фактическими параметрами при вызове функции должны соблюдаться правила соответствия по последовательности и по типам. Фактический параметр — это выражение того же типа, что и у соответствующего ему формального параметра.

Необходимо усвоить еще один важнейший принцип, действующий в С++: передача параметров при вызове функции происходит только по значению. Поэтому выполнение функции не может изменить значения переменной, указанных в качестве фактических параметров.

**Прототип функции.** Оказывается, совсем не обязательно было в предыдущем примере помещать полное определение функции \_max() перед основной частью программы. Вот другой вариант программы, решающей ту же самую задачу.

#include <iostream>  
**using namespace** std;  
//Прототип функции \_max   
**int** \_max(**int**, **int**);  
//Основная функция  
**int** main (**int** argc, **char**\*\* argv)  
{   
 **int** a,b,c,d;  
 cout<<*"Введите a,b,с:"*;  
 cin >> a >> b >> c;  
 d =\_max(\_max(a,b),c);  
 cout <<*"\nmax(a,b,c) = "* << d;  
}//Определение функции \_max   
**int** \_max(**int** х, **int** у)  
{  
 **return** x > y ? x : y;   
}

Здесь использован прототип функции. Прототипом называется предварительное описание функции, в котором содержатся все необходимые сведения для правильного обращения к ней: имя и тип функции, типы формальных параметров. В прототипе имена формальных параметров указывать необязательно (как это сделано в примере), хотя их указание не является ошибочным. Можно было написать и так, как в заголовке определения функции:

**int** \_max (**int** х, **int** у);

Точка с запятой в конце прототипа ставится обязательно!

Можно было бы записать прототип и в теле основной функции наряду с описаниями других программных объектов в ней. Вопрос о размещении описаний связан с понятием области видимости, который будет обсужден немного позже.

А теперь рассмотрим программу для вычисления наибольшего общего делителя для суммы, разности и произведения двух чисел.

#include <iostream>  
#include <math>  
**using namespace** std;  
**int** nod2(**int**,**int**);  
**int** main(**int** argc, **char**\*\* argv)  
{  
 **int** a,b,Rez;  
 cout << *"a="*; cin >> a;  
 cout << *"b="*; cin >> b;  
 Rez=nod2(nod2(a+b, abs(a-b)),a\*b);  
 cout << *"NOD paвeн”* << Rez;  
}  
**int** nod2(**int** M, **int** N)   
{  
 **while**(M!=N)  
 {  
 **if**(M>N) M=M-N; **else** N=N-M;  
 )  
 **return** M;  
}

Может возникнуть вопрос: если основная часть программы является функцией, то кто (или что) ее вызывает? Ответ состоит в следующем: программу вызывает операционная система при запуске программы на исполнение. Обработка возвращаемых main результатов будет осуществляться системными средствами. Обычно, если программа завершается без ошибок, то возвращают 0.

## Контрольные вопросы

1. Для чего предназначаются подпрограммы?
2. Что включает в себя заголовок функции?
3. Чем отличаются формальные и фактические параметры?
4. Чем отличаются параметры-переменные от параметров-констант?
5. Для чего предназначена рекурсия?

## Задания к практической работе

Составить программу вычисления функции  и суммы , представляющей собой формулу разложения заданной функции  в ряд.

Варианты заданий приведены в табл. 7.2.

В первой графе содержится порядковый номер задания. Рекомендуется выбирать номер задания, соответствующий порядковому номеру фамилии студента в списке группы.

Во второй графе приводится формула функции .

В третьей графе помещается формула разложения функции  в ряд.

В четвертой графе показан диапазон значения аргумента , для которого следует выполнить вычисления. Рекомендуется вычислить  и для 11 точек заданного диапазона изменения .

Шаг изменения аргумента определить по формуле:

,

где  – начальное значение ;

 – конечное значение .

В пятой графе указано значение n – количество членов суммы .

При составлении программы необходимо использовать процедуры или функции, основанные на нерекурсивном и рекурсивном методах вычислений.

Таблица 7.2

Варианты заданий

| № | Функция Y | Сумма S | Диапазон изменения аргумента | n |
| --- | --- | --- | --- | --- |
|  |  |  | 0,1 ≤ *x* ≤1 | 10 |
|  |  |  | π ≤ *x* ≤ 9π  5 5 | 40 |
|  |  |  | 0,1 ≤ *x* ≤ 1 | 10 |
|  |  |  | 1 ≤ *x* ≤ 2 | 15 |
|  |  |  | 0,1 ≤ *x* ≤ 1 | 25 |
|  |  |  | 0,1 ≤ *x* ≤ 1 | 10 |
|  |  |  | 0,1 ≤ *x* ≤ 0,8 | 40 |
|  |  |  | 0,1 ≤ *x* ≤ 1 | 20 |
|  |  |  | 0,1 ≤ *x* ≤ 1 | 10 |
|  |  |  | 0,2 ≤ *x* ≤ 1 | 10 |
|  |  |  | π ≤ *x* ≤ π  5 | 20 |
|  |  |  | 0,1 ≤ *x* ≤ 1 | 10 |
|  |  |  | 0,1 ≤ *x* ≤ 1 | 20 |
|  |  |  | 0,1 ≤ *x* ≤ 1 | 30 |
|  |  |  | 0,1 ≤ *x* ≤ 0,5 | 40 |
|  |  |  | 0,1 ≤ *x* ≤ 1 | 35 |
|  |  |  | 0,1 ≤ *x* ≤ 1 | 15 |
|  |  |  | -2 ≤ *x* ≤ -0,1 | 40 |
|  |  |  | 0,1 ≤ *x* ≤ 1 | 20 |
|  |  |  | 0,1 ≤ *x* ≤0,8 | 40 |