

**Программирование с  
использованием  
подпрограмм**

**Варшавский Павел Романович**

**к.т.н., доцент кафедры ПМ**

# Список литературы по курсу

1. Князев А.В. Основы языка С++. Учебное пособие. –М.: Издательство МЭИ, 2013 – 80 с. ISBN 978-5-7046-1425-8.
2. Князев А.В. Работа со сложными структурами данных на языке С++. Учебное пособие. –М.: Издательство МЭИ, 2015 – 48 с. ISBN 978-5-7046-1658-0.
3. Программирование. Сборник задач. Учебное пособие. –Санкт-Петербург: Лань, 2019 – 140 с. ISBN 978-5-8114-3857-0.

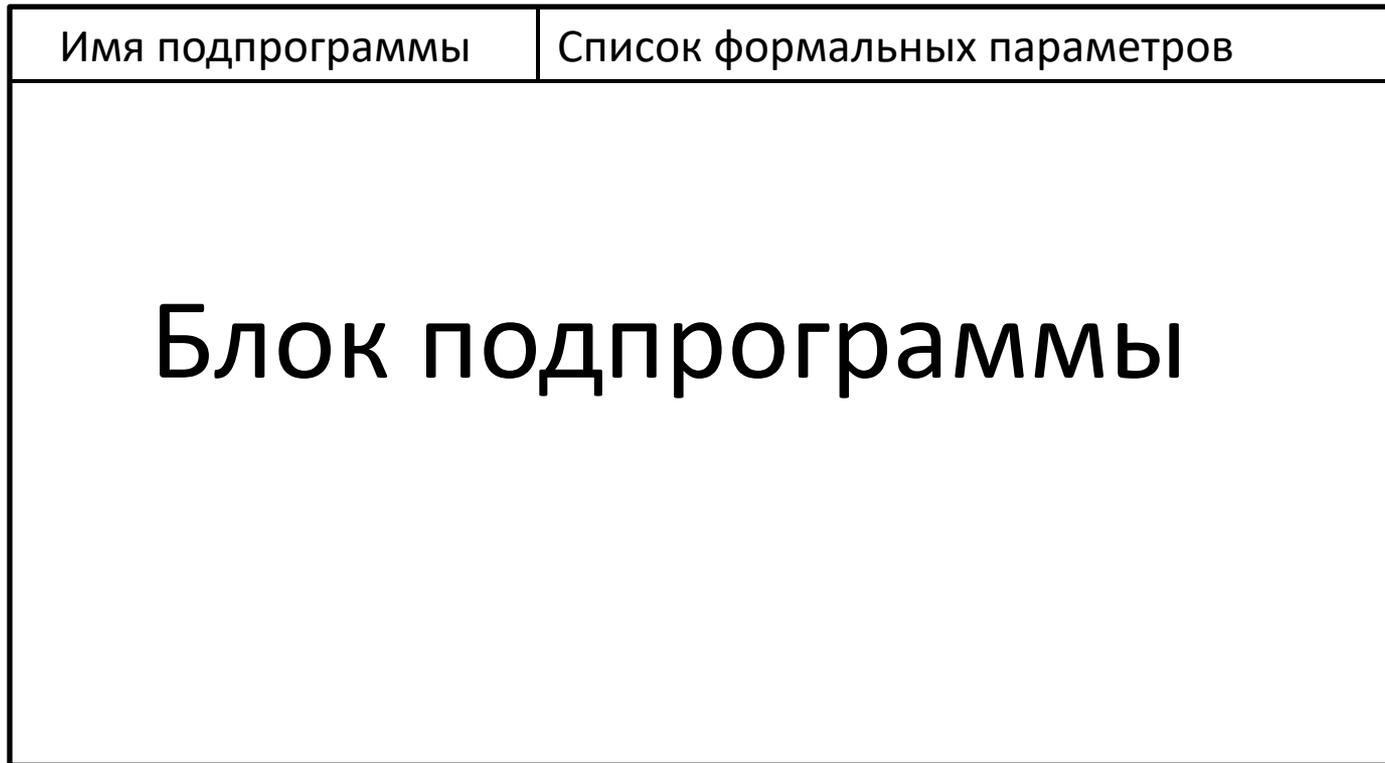
URL: <https://e.lanbook.com/book/121485>

**ПОДПРОГРАММЫ**

**ФУНКЦИИ В С/С++**

- **Подпрограмма** – именованная, логически законченная группа операторов языка, которую можно вызвать для выполнения определенной задачи (класса задач) любое количество раз из различных мест программы.
- **Подпрограмма** – это программа, которую можно вызвать по имени, передать ей исходные данные в виде значений входных параметров и получить результаты.

# Структура подпрограммы



- Подпрограмма состоит из **заголовка** и **блока**.
- В заголовке указывается **тип подпрограммы**, **имя подпрограммы** и **список формальных параметров**.
- В списке формальных параметров для каждого параметра указывается: **тип параметра**, его **имя** и **способ передачи**.
- **Блок подпрограммы** – это программа, которая может иметь свои **локальные переменные**.
- В теле подпрограммы наряду с локальными переменными используются **формальные параметры**.

- **Алгоритм подпрограммы** описывает процесс преобразования её входных параметров в результаты.
- В языке **C/C++** подпрограммы реализованы в виде **функций**. Как известно, программа на C/C++ состоит из одной или нескольких функций.
- **Функция** получает входные параметры и возвращает единственное значение.
- **Функция должна быть описана перед своим использованием.**

# Описание функции

*тип\_функции*      *имя\_функции*      (*список\_формальных\_параметров*)

*тип\_функции* – тип возвращаемого функцией значения (например, *целое*, *вещественное*, *строковое* и др.).

*список\_формальных\_параметров* – список аргументов с указанием их типов и имен, разделенных запятыми, и заключенных в скобки. Если функция не имеет параметров, то скобки все равно необходимы.

- пример заголовка функции:

```
int FUN(int i, float F)
```

- блок имеет структуру обычной программы:

```
{
```

```
    <операторы>
```

```
    return значение; // Возврат значения функции
```

```
}
```

- функции могут быть определены до функции **main**, а также после функции **main**.
- В случае, если функции определены после функции **main** или **даже в другом файле**, необходимо до первого вызова функции сообщить программе тип возвращаемого функцией значения, а также количество и типы ее аргументов. Подобное сообщение называется **прототипом функции**.

- **прототип функции** может полностью совпадать с заголовком функции, хотя при объявлении функции компилятору необходимо знать только имя функции, количество аргументов и их типы и тип возвращаемого функцией значения. Поэтому имена аргументов в прототипе функции могут отсутствовать.

```
int FUN(int x, float y)
```

```
int FUN(int, float)
```

- для того, чтобы функция вернула какое-либо значение, в ней должен быть оператор возврата **return значение;**
- если возврата значения функции не требуется, то тип возвращаемого значения функцией – **void**, а оператор возврата просто – **return** и его можно даже опустить в теле функции.

```
void FUN(int x, float y)  
{  
  
}
```

# Вызов функции

Вызов функции представляет из себя указатель функции, то есть **имя функции** и в скобках **список фактических параметров**.

*имя\_функции ([список\_ фактических\_ параметров])*

Пример вызова функции: **FUN(10, 11.2)**

При вызове функции фактические параметры должны соответствовать формальным параметрам:

- по типу;
- по количеству;
- по порядку.

Таким образом, функция пишется относительно формальных параметров, для которых указываются типы, а выполняется для фактических параметров, которые подставляются на место формальных при вызове функции.

# Обмен данными между вызывающей программой и подпрограммой

Обмен данными между вызывающей программой и подпрограммой осуществляется при помощи аппарата **формальных – фактических** параметров.

При вызове подпрограммы, ей передаются фактические параметры, используя которые, она выполняется.

Существует два способа передачи данных:

- **по значению;**
- **по ссылке.**

При передаче данных по значению в подпрограмму передаётся **КОПИЯ** фактического параметра.

Из этого следует, что, если подпрограмма изменит значение этого параметра, в вызывающей программе он останется неизменным.

При передаче данных по ссылке, в подпрограмму передаётся **адрес** фактического параметра, то есть подпрограмма «допускается» к памяти вызывающей программы (какая-то область памяти становится общей для них).

В этом случае, если подпрограмма изменяет значение фактического параметра, эти изменения происходят в памяти вызывающей программы. Так можно передавать результаты расчетов.

В C/C++ по умолчанию данные передаются по значению.

Для реализации передачи данных по ссылке в списке формальных параметров перед именем параметра, передаваемом по ссылке, ставится **&**.

```
int FUN(int x, float &y)
```

**x** – по значению;

**y** – по ссылке.

# Побочный эффект при выполнении функции

Функции могут передавать вызывающей программе результат через параметры по ссылке.

Передача такого дополнительного результата, наряду с основным, передаваемым через результат функции, называется побочным эффектом при выполнении функции.

# Пример программы с функцией

Задано два квадратных уравнения:

$$a*x^2 + b*x + c = 0$$

$$d*x^2 + e*x + f = 0$$

Решить эти уравнения в области действительных чисел и напечатать сообщение, если действительных корней нет.

Для решения этой задачи напишем функцию решения квадратного уравнения и применим её к каждому уравнению.

Назовём логическую функцию **cor**, у неё будет 5 параметров, три из них передаются по значению **a**, **b**, **c** – это коэффициенты уравнения, два параметра по ссылке: **x1**, **x2** – корни уравнения.

*Результат функции – признак наличия действительных корней, получающий значение **true**, если есть действительные корни, и **false**, если таких корней нет.*

```
bool cor (int a, int b, int c, float &x1, float &x2)
{
    float D=sqr(b) - 4*a*c;
    if (D >= 0)
    {
        x1=(-b + sqrt(D))/(2*a);
        x2=(-b - sqrt(D))/(2*a);
        return true;
    }
    else
    {
        return false;
    }
}
```

```
int main ()
{
    int a, b, c, d, e, f;
    float x1, x2;
    cout<<"введите коэффициенты уравнений";
    cin>>a>>b>>c>>d>>e>>f;
    // Решение первого уравнения
    if (cor(a, b, c, x1, x2)==true) // Вызов функции
        { cout<<x1<<x2<<endl; }
    else { cout<<"действительных корней нет"<<endl; }
    // Решение второго уравнения
    if (cor(d, e, f, x1, x2)==true) // Вызов функции
        { cout<<x1<<x2<<endl; }
    else { cout<<"действительных корней нет"<<endl; }
    return 0;
}
```

# Локализация имён

В программе, содержащей подпрограммы, может существовать несколько уровней описания объектов программы (типов, констант, переменных, функций).

Рассмотрим взаимодействие объектов в таких ситуациях на примере переменных.

Пусть есть программа и в ней описана подпрограмма.

Переменные, описанные в подпрограмме доступны только в этой подпрограмме, они **локальные переменные** этой подпрограммы и недоступны во внешней программе.

Переменные, описанные во внешней программе – **глобальные переменные**, доступны в программе и в подпрограмме, а это значит, что подпрограмма может использовать значения глобальных переменных и изменять их.

Если имя локальной переменной совпадает с именем глобальной переменной, то в подпрограмме глобальная переменная становится недоступной.

## ПРИМЕР

```
int F(int x)
{
    int c;
    // Раздел операторов функции
    c = 10;
    Return a + c + x;
}

int main()
{
    int a, b, c, d, e;
    // Раздел операторов программы
    a = 5;
    c = 100;
    d = F(10); // первый вызов функции
    a = 25;
    e = F(10); // второй вызов функции
    return 0;
}
```

В этом примере **a**, **b**, **c**, **d**, **e** – глобальные переменные.

Глобальная переменная **c** недоступна в функции, так как её имя совпало с именем локальной переменной, остальные переменные доступны.

При вычислении функции используется значение глобальной переменной **a** и локальной **c**.

В зависимости от того, какое значение имела глобальная переменная **a** в момент вызова функции, получается различные значения результата функции.

Так при первом обращении функция возвратит результат равный **25**, а при втором обращении **45**.