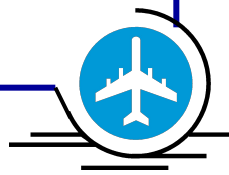
Введение в программирование

Лекция 7.

ПОДПРОГРАММЫ



Подпрограммы

Подпрограмма - это программа, которая выполняется в составе одной или нескольких программ.

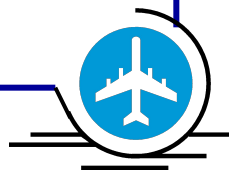
Для подпрограмм используются обозначения, аналогичные функциям в математике.

Перед использованием функции необходимо дать ее определение, например:

$$P(x) = x * (x + 1)$$

После этого ее можно использовать, например, для получения значения функции P(x) при x=10:

$$P(10) = 10 *(10 + 1) = 110$$



Подпрограммы могут быть двух типов:

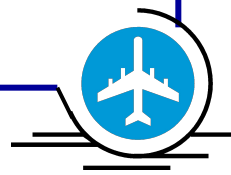
- *функция* – подпрограмма, обладающая значением;
- *процедура* – подпрограмма, не обладающая значением.

В языке С все подпрограммы называют функциями и различают:

- *функции, обладающие значением;*
- *функции, не обладающие значением.*

Алгоритм подпрограммы записывается в форме ***определения функции*.**

Программа на языке С состоит из определений функций, одна из которых **main().**

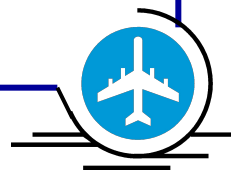


Определение функции имеет вид:

```
/*          Заголовок функции>                                  */
[<тип_значен>] <имя_функции> ([<тип>< имя>[,< тип><имя>]…])
   { /*          Тело функции                                   */
     <Определение переменных функции>
     < Операторы функции>
     [ return  [<выражение>]; ]
      }
```

Отсутствие значения функции указывается как тип значения **void.**

Разрешено не указывать тип значения, тогда **по умолчанию** подразумевается тип **int.**



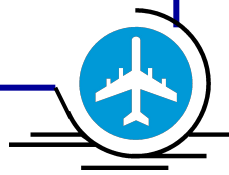
В заголовке заданы типы и имена формальных параметров.

При отсутствии параметров скобки могут быть пустые или в скобках указывается слово void.

Тело функции начинается с объявления локальных переменных, которые могут использоваться только в той функции, где объявлены.

Формальные параметры локальны для функции.

Далее следуют операторы, реализующие функцию.



Подпрограммы

Выполнение подпрограммы завершается оператором возврата, осуществляющим возврат в вызывающую программу в точку, следующую за вызовом.

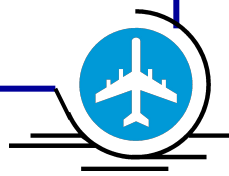
Оператор возврата имеет вид:

return [<выражение>];

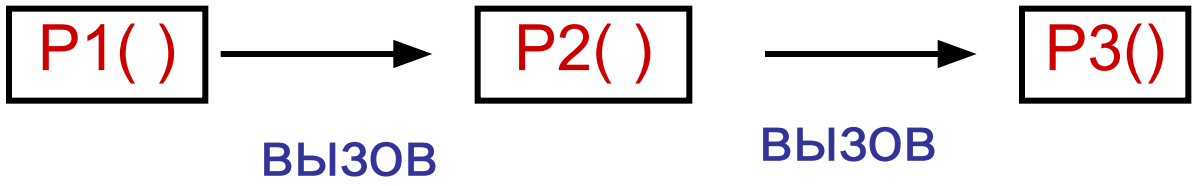
В теле функции, обладающей значением, обязательно должен быть оператор возврата.

Тип вычисленного выражения должен совпадать с типом значения функции.

Если функция не возвращает значение, оператор возврата может отсутствовать.



Подпрограмма выполняется после вызова.



P2( ) – главная по отношению к P3( ) и
подчиненная по отношению к P1( ).

Вызов подпрограммы имеет вид:

<имя_функции> ([<имя> [, < имя>] …]);

имена фактических параметров.

Вызов функции, обладающей значением:

<имя_переменной> =

<имя_функции> ([<имя>[, < имя>] …]);

Примеры заголовков функций, не обладающих значением:

void p_f (float x, float *z) - вещественные входной параметр **x** и выходной параметр **z**;

void p_fakt (int k, int *f) - целочисленные входной параметр **k** и выходной параметр **f**.

Примеры вызова функций p_f и p_fakt:

```
float a, b;
    int n, p;
    ….
a=15.2;          p_f (a, &b);
n=4;             p_fakt (n, &p);
```

Вызов функции, не обладающей значением, может записываться только как **самостоятельный оператор**.

Примеры заголовков функций, обладающих значением:

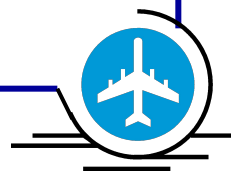
float f (float x) - вещественная функция f от вещественного параметра x;

int fakt (int k) - целочисленная функция fakt с входным целочисленным параметром k.

Примеры вызова функций f и fakt:

```
float a, b, c;
     int n, p;
     ….
a=15.2;  b=20.5;          c = f (a) *f(b);
n=4;                      p = fakt (n) + fakt(5);
```

Вызов функции, обладающей значением, может являться **операндом в выражении.**



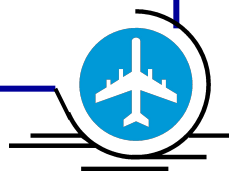
Параметры подпрограмм

Параметры предназначены для передачи входных и выходных данных подпрограммы.

При вызове подпрограммы происходит **согласование параметров:**

- порядок и типы формальных параметров в определении подпрограммы должны совпадать с порядком и типами фактических параметров при вызове подпрограммы.

После вызова подпрограммы выполняются ее операторы, где **фактические параметры подставляются вместо формальных параметров.**



Передача параметров и согласование формальных и фактических параметров может осуществляться:

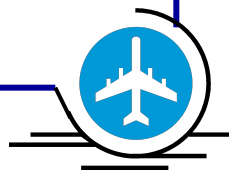
- **по значению;**
- **по ссылке** (по адресу).

Передача параметров по значению:

Фактический параметр может быть выражением.

Формальному параметру присваивается значение фактического параметра.

Изменения параметра в подпрограмме не влияют на его значение в вызывающей программе.



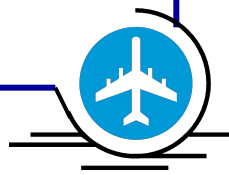
Передача параметров по ссылке:

Фактический параметр может быть только переменной.

Передается не значение фактического параметра, а его адрес. Фактический параметр заменяет формальный параметр, действия выполняются над фактическим параметром.

Изменения параметра в подпрограмме меняют его значение в вызывающей программе.

Входные параметры передаются по значению, а выходные – по ссылке.



При передаче параметра по ссылке в вызове функции нужно получить его адрес с помощью операции &, а в заголовке функции тип параметра должен быть указатель.

Упрощенное правило

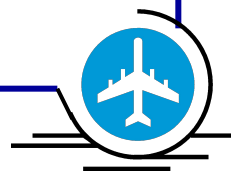
передачи параметра по ссылке:

1. символ * - перед именем параметра в заголовке и теле подпрограммы;
2. символ & - перед именем параметра при вызове .

Например, параметр **x** передается по ссылке,

заголовок функции - ***f (int n, int *x),***

вызов функции - ***f (n, &x).***



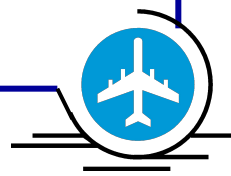
Массивы всегда передаются по ссылке. Имя массива является адресом его первого элемента, поэтому на параметры – массивы правило не распространяется (т.е. не нужны символы * и &).

Пример заголовка функции с параметром массив:

float f (int n, float m[ ]) - вещественная функция f от целочисленного параметра n и вещественного массива m.

Пример вызова этой функции:

```
int  k;  float z[100],  t;        t = f (k, z);
```

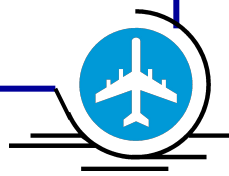


Область действия переменных.

Каждая переменная до использования должна быть описана.

Локальные переменные. Областью действия локальной переменной является блок, в котором эта переменная объявлена.

Глобальные переменные. Областью действия глобальной переменной является вся программа. Глобальные переменные объявляются до всех функций или между определениями функций.



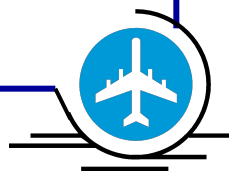
Подпрограммы

Если вызов функции выполняется в программе раньше определения функции, то в начале программы необходимо поместить объявление -прототип функции.

Прототип функции – это заголовок функции, завершающийся символом ‘;’ (без тела функции)

Например,

float f (int n, float m[ ]); - прототип вещественной функции f от целочисленного входного параметра n и вещественного массива m.



Задача 7.1. Составить программу вычисления c = n! / (m! * (m-n)!)

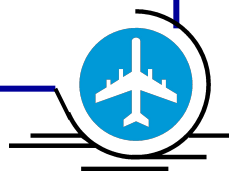
Вычисление факториала удобно оформить как подпрограмму.

Решение А. Использование подпрограммы, не возвращающей значение.

Пусть вызов подпрограммы p_fakt (k, f) обозначает действие - операцию присваивания f = k!.

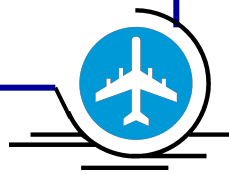
k - исходные данные, f - результат.

Подпрограмма p_fakt() имеет входной параметр k и выходной параметр f.



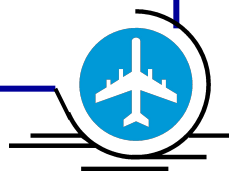
Подпрограммы

```
/* Программа 7.1а. Вычисление c=n!/(m!*(m-n)!)                    */
   /* с помощью подпрограммы,  не обладающей значением            */
   #include <stdio.h>
void p_fakt (int k, int *f);      /* прототип функции              */
void main (void)
{  int n, m, c;                   /* исходные данные и результат   */
      int f1, f2, f3;             /*          n!, m!, (n-m)!       */
   printf("\nВведите два исходных целых числа ");
      scanf("%d %d", &n, &m);
   p_fakt (n, &f1);                        /* f1 = n!              */
   p_fakt (m, &f2);                        /* f2 = m!              */
   p_fakt (n-m, &f3);                      /* f3 = (n-m)!          */
   c = f1 / (f2 * f3);   printf ("\n c = %d", c);
    }
```

Подпрограммы

```
/*  Подпрограмма: f = k!;                                   */
void p_fakt (int k, int *f)
   {  int j;                          /* текущий множитель     */
  *f=1;
   for (j=2; j<=k; j++)
          *f = *f * j;
   return;                            /* не обязателен         */
   }
```



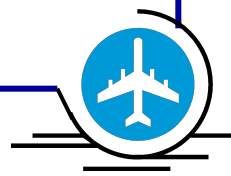
Решение Б. Использование подпрограммы, возвращающей значение (функции).

Пусть функция fakt(k) обозначает значение k!.

```
/* Программа 7.2б. Вычисление c=n!/(m!*(m-n)!)                    */
   /*с помощью подпрограммы, обладающей значением                 */
    #include <stdio.h>
int fakt   (int k);          /* прототип функции                  */
void main(void)
   {  int n, m, c;           /* исходные данные и результат       */
  printf("\nВведите два исходных целых числа ");
      scanf("%d %d", &n, &m);
  c = fakt(n) / (fakt (m) * fakt (n-m));
  printf ("\n c = %d", c);
    }
```

Подпрограммы

```
/*                Функция !                              */
   int fakt   (int k)
{  int f;                          /*     k!             */
   int j;                          /* текущий множитель  */
   f=1;
      for (j=2; j<=k; j++)
          f = f * j;
   return f;                       /* значение функции   */
   }
```

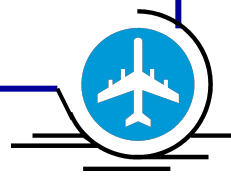


Подпрограммы

В программе 7.1б результат работы подпрограммы fakt передается как значение функции. Поэтому **f** является не параметром, а вспомогательной локальной переменной и записывается без звездочки.

В программе 7.1а вызов функции, не обладающей значением, может записываться только как **самостоятельный оператор**.

В программе 7.1б вызов функции, обладающей значением, может являться **операндом в выражении**.



В программе 7.1а при выводе результата можно заменить строки:

```
c = f1 / (f2 * f3);
        printf ("\n c = %d", c);
```

строкой:

```
        printf ("\n c = %d", f1/(f2*f3));
```

Тогда переменная **c** и ее определение становятся ненужными.

В программе 7.1б также можно обойтись без переменной **c**.