7. ПОДПРОГРАММЫ

7.1. Основные понятия

Подпрограмма - это программа, которую можно выполнять в разных местах в составе одной или нескольких программ.

Программа, в составе которой выполняется подпрограмма, по отношению к подпрограмме называется *главной, основной* или *вызывающей программой*. В то же время, она сама может быть подпрограммой (подчиненной программой) по отношению к вызывающей ее программе.

Например, в программе 1 функция (программа) F(x), для вычисления которой используется функция (подрограмма) cos(x), является главной по отношению к cos и подчиненной (подпрограммой) по отношению к программе main, в составе которой вычисляется F(x).

Для подпрограмм используются обозначения, аналогичные функциям в математике. Перед использованием функции необходимо дать ее определение, например: F(x) = x * x + 2 (7.1)

После этого можно использовать, например, значение функции при x=5:

F(5) = 5 * 5 + 2 = 27 (7.2)

Вызов подпрограммы (команда «выполнить подпрограмму») записывается аналогично использованию функции (6.2):

P (a1, a2, ... , an) (7.3)

где P - *имя подпрограммы* (аналогично имени функции F),
a1, a2, ... , an - *аргументы* вызова (*фактические параметры*), конкретные величины, подставляемые вместо формальных параметров при выполнении подпрограммы.

Формальный параметр – это входная или выходная переменная подпрограммы, указанная в заголовке ее определения (как x в левой части определения (7.1)).

При изменении имени формального параметра смысл подпрограммы не изменяется (как если бы в определении 7.1 заменить x, например, на y).

В языках программирования подпрограммы, как правило, делят на два вида: подпрограмму, обладающую значением, называют *функцией*, не обладающую значением - *процедурой*. Такая терминология используется, например, в языке Pascal.

Вызов подпрограммы, обладающей значением, называется *указателем функции* и обозначает значение функции так же, как принято в математике. Такой вызов можно использовать в качестве операнда выражения, как, например, F(a+5) в выражении

F(a+5) – 3

Вызов подпрограммы, не обладающей значением, не может служить операндом выражения. Он записывается как самостоятельный оператор и

обозначает действия, выполняемые подпрограммой после замены формальных параметров на соответствующие фактические параметры.

На схемах вызов подпрограммы записывают в прямоугольнике с двойными боковыми сторонами. Подпрограмма описывается отдельной схемой, в начальном овале которой указывают имя подпрограммы (рис. 7.1).

В языке С, в отличие от других языков, все подпрограммы называют *функциями*. Функция может обладать значением (*возвращать значение*) либо нет. Алгоритм подпрограммы записывается в форме *определения функции,* имеющего вид:

```
заголовок
{  тело функции
}
```

С-программа состоит из определений функций, одна из которых должна иметь имя main (главная). С нее начинается выполнение программы.

Между определениями функций могут находиться определения *глобальных переменных*, действующие на всю программу. Переменные, определенные внутри функции, включая ее формальные параметры, являются *локальными*, т. е. могут использоваться только в этой функции.

Заголовок функции записывается в следующем порядке: тип значения функции (при отсутствии значения пишется **void**), имя функции, затем в скобках задаются описания формальных параметров, разделенные запятыми:

тип_значения имя_функции (тип имя, тип имя, …, тип имя)
или **void**

При отсутствии параметров в скобках указывается слово **void** или скобки остаются пустыми.

Тип значения функции разрешается не указывать. В этом случае (*по умолчанию)* подразумевается тип **int.**

В начале *тела функции* размещены определения типа (описания) всех использованных в ней локальных переменных. Далее следуют реализующие функцию операторы.

Перед именем выходного параметра в заголовке и теле функции пишется звездочка *, а в ее вызове - амперсенд &, т. е. выходной фактический параметр должен иметь вид:

& переменная

Это правило не касается параметров, являющихся массивами (см. раздел 7.3 и пример 7.5). Знак амперсенд "&" в данном случае обозначает операцию получения адреса переменной. Более подробно механизм подстановки параметров описан в разделе 7.3.

Например, оператор ввода значения переменной y

scanf("%f", &y);

представляет собой вызов стандартной функции scanf(), для которой фактический параметр y является результатом работы, т. е. выходным параметром, и поэтому записывается со знаком амперсенд &.

Входной фактический параметр можно записывать в форме выражения, в частном случае - в виде переменной или константы.

Примеры заголовков функций:

void main (**void**) - функция main без значения и без параметров;
main () - целочисленная функция main без параметров;
float F (**float** x) - вещественная функция F от вещественного параметра x;
int fakt (**int** k) - целочисленная функция fakt от целочисленного параметра k;
void pfakt (**int** k, **int** *f) - функция без значения pfakt с входным целочисленным параметром k и выходным целочисленным параметром f.

Порядок и типы фактических параметров вызова функции должны совпадать с порядком и типами формальных параметров в заголовке этой функции.

В отличие от других языков, в языке С заголовок и вызов функции всегда пишутся со скобками для параметров, даже, если параметры отсутствуют и в скобках ничего не записано.

Выполнение функции заканчивается *оператором возврата,* обозначающим прекращение выполнения функции и возврат в вызывающую программу в точку, следующую за вызовом функции. Для функции, обладающей значением, оператор возврата имеет вид:

return выражение;

Значение заданного выражения становится значением функции, указатель функции заменяется ее значением.

Для функции без значения оператор возврата не содержит выражения:

return;

В конце тела функции, перед закрывающей фигурной скобкой, оператор return без выражения можно не писать: он будет вставляться автоматически.

Если определение функции в программе расположено после вызова этой функции, то до вызова функции в программе должен быть ее *прототип* (объявление, описание), записываемый как заголовок функции, заканчивающийся точкой с запятой ';'. В отличие от заголовка, в прототипе можно не задавать имена формальных параметров (их типы должны

присутствовать). Прототипы можно размещать в тех же местах, где и определения переменных.

Примеры прототипов функций, заголовки которых приведены выше:

```
float  F (float  x);
void  pfakt (int,  int *);
```

Дело в том, что заголовок или прототип функции нужен транслятору, чтобы проверять правильность количества и типов фактических параметров вызовов этой функции, а также тип ее значения. Если тип фактического параметра не совпадает с типом формального параметра или тип значения функции не соответствует его использованию, транслятор обеспечивает необходимое *преобразование типа*.

При отсутствии определения и прототипа функции к моменту трансляции ее вызова (в большинстве языков, включая C++ и Pascal, это запрещено, но в C допускается) параметры и значение этого вызова не проверяются и не преобразуются. Программист сам должен следить за их правильностью. Это часто приводит к трудно обнаруживаемым ошибкам.

В программе можно использовать также функции, определения которых (в транслированном виде) находятся в системных или личных библиотеках. Прототипы *библиотечных функций* обычно находятся в специальных *заголовочных файлах*, подобных файлам stdio.h, math.h и другим, содержащим прототипы стандартных функций языка C. Они вставляются в программу оператором

#include <файл> или **#include** "файл"

Если имя файла задано в угловых скобках < >, он ищется только в системном каталоге (директории). Файл, заданный в кавычках, предварительно ищется в текущем каталоге.

Пример 1. Составить программу вычисления c = n! / (m! * (m-n)!)

В вычисляемом выражении требуется трижды вычислять факториал вида k!, где k! = 1*2*...*k при целом k > 0, 0! = 1.

Вычисление факториала удобно оформить как подпрограмму.

Решение 1. Использование подпрограммы, не обладающей значением.

Пусть вызов подпрограммы pfakt (k, f) обозначает действие - операцию присваивания f = k!;. Здесь k - исходные данные, а f - результат. Таким образом, подпрограмма pfakt() имеет входной параметр k и выходной параметр f.

Обозначим факториалы: n!, m! и (n-m)! через f1, f2 и f3, соответственно. Получим схему на рис. 7.1 и программу1.

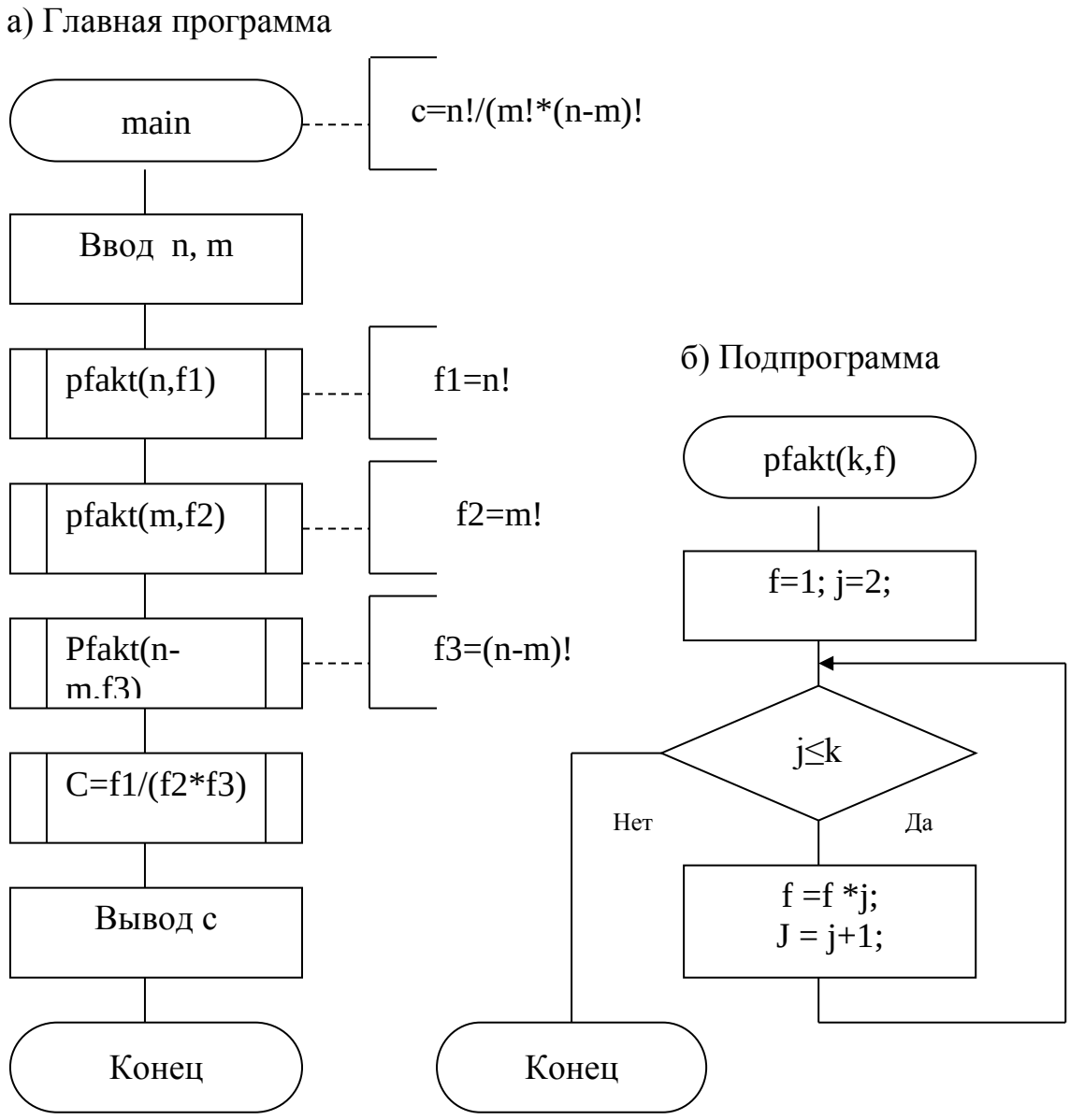

Рис. 7.1. Использование подпрограммы, не обладающей значением

Подпрограмма вычисления факториала pfakt содержит повторение умножений, т. е. имеет циклическую структуру. Текущим множителем служит вспомогательная переменная j, изменяющаяся с шагом +1.

Программа 1

```
/* Вычисление c=n!/(m!*(m-n)!) с помощью подпрограммы,
                                не обладающей значением    */
#include <stdio.h>
```

```
/*                    Подпрограмма: f = k!;                    */
void pfakt (int k, int *f)
{  int j;       /* текущий множитель                          */
   *f=1;
   for (j=2; j<=k; j++)
      *f = *f * j;
   return;          /* здесь не обязателен */
}
/*              Вычисление  c = n! / (m! * (n-m)!)             */
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);
   pfakt (n, &f1);                /* f1 = n!                    */
   pfakt (m, &f2);                /* f2 = m!                    */
   pfakt (n-m, &f3);              /* f3 = (n-m)!                */
   c = f1 / (f2 * f3);
   printf ("\n c = %d", c);
 }
```

Решение 2. Использование подпрограммы, обладающей значением (функции).

Пусть функция fakt(k) обозначает значение k!. Получим схему на рис. 2 и программу 2.

Программа 2

```
/* Вычисление c=n!/(m!*(m-n)!) с помощью подпрограммы,
                                   обладающей значением       */
#include <stdio.h>
/*                          Функция  k!                        */
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                      */
}
/*              Вычисление  c = n! / (m! * (n-m)!)             */
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);
}
```

В программе 2 результат работы подпрограммы fakt передается как значение функции. Поэтому f является не параметром, а вспомогательной локальной переменной и записывается без звездочки.

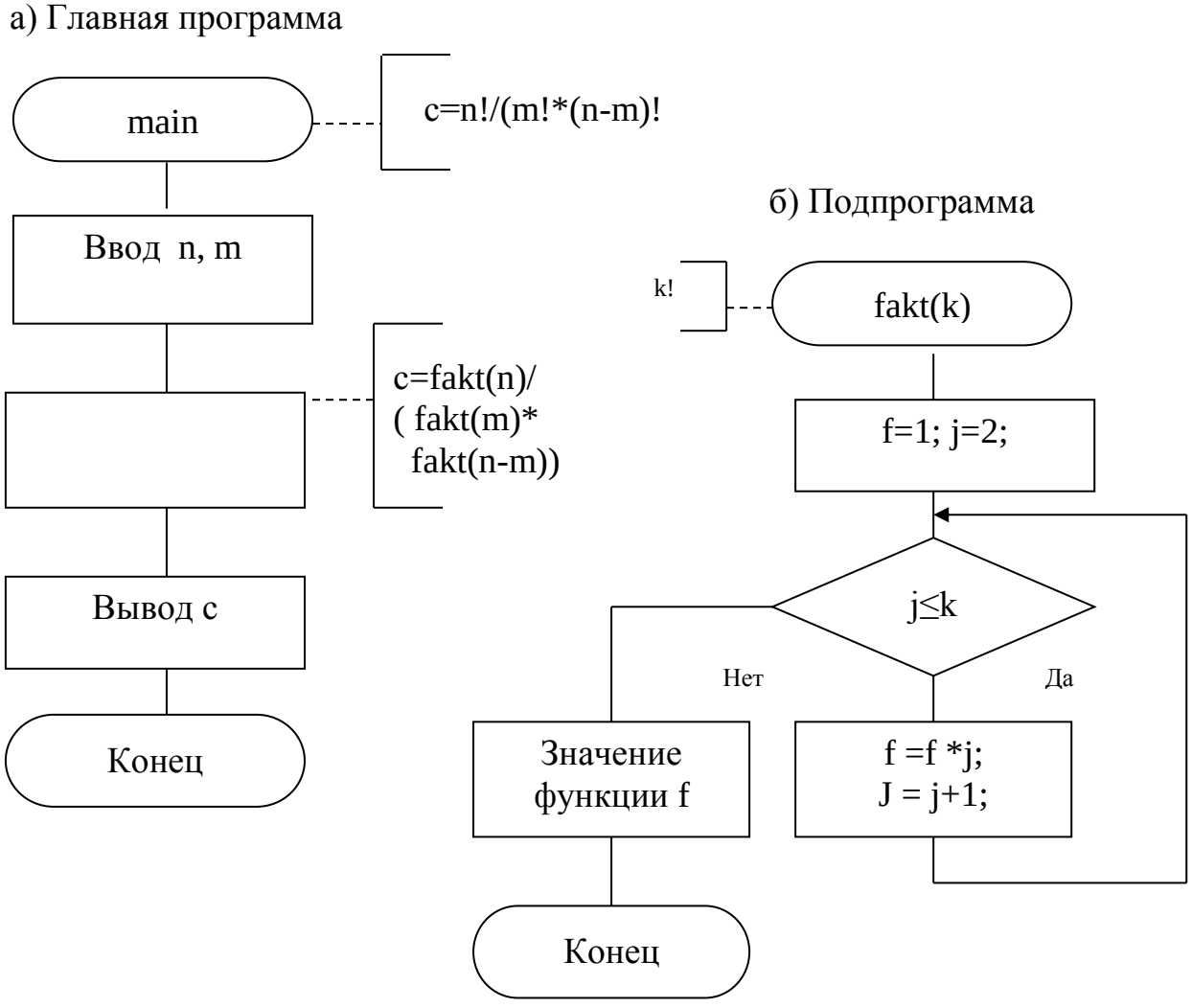

Рис. 7.2. Использование подпрограммы, обладающей значением

В программах 1 и 2 и их схемах при выводе результата вместо переменной с можно написать присваиваемое ей выражение, например, заменить строки:

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

строкой:

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

Тогда переменная с и ее определение становятся ненужными.

Из приведенных примеров видно, что, если результатом работы подпрограммы является одно значение, его удобнее оформить как значение функции. Если результат содержит несколько значений, их обычно делают выходными параметрами (одно из них можно сделать значением функции).

Подпрограмма, не имеющая значения, может:

- изменить значение своего выходного параметра (точнее - подставленной вместо него переменной - фактического параметра),
- изменить значение глобальной переменной - общей переменной вызывающей программы и подпрограммы,
- выполнить ввод/вывод данных,
- изменить состояние операционной среды.

С точки зрения простоты и наглядности программы использование выходных параметров лучше, чем изменение глобальных переменных, поскольку из вызова подпрограммы с параметрами (без анализа ее алгоритма) явно видно, какие фактические параметры могут измениться при ее выполнении.

Чтобы определить, какие глобальные переменные могут измениться при вызове подпрограммы, необходимо анализировать не вызов, а алгоритм не только этой подпрограммы, но и всех подчиненных ей программ: вызываемых из нее, вызываемых из тех, которые вызываются из нее и т. д. Глобальные переменные затрудняют понимание программы, и поэтому нежелательно использовать их без особой необходимости.

Функция, кроме вычисления значения, также может выполнять все перечисленные выше действия в качестве *побочного эффекта*.

Для функций, основной задачей которых является вычисление значения, эти побочные эффекты (особенно изменение глобальных переменных) нежелательны и их использование считается плохим стилем программирования, поскольку делает программу более запутанной и менее надежной.

В то же время, существуют функции, для которых побочные эффекты являются основным результатом, а значение играет второстепенную роль. Примером является стандартная функция ввода scanf, или функция, значением которой является *код завершения*, показывающий, насколько успешно удалось ей решить свою задачу.

В большинстве языков запрещается вызывать в виде оператора подпрограмму, обладающую значением. Там, где это разрешено, например в языке С, значение такого вызова не используется и теряется.

7.2. Применение подпрограмм

Применение подпрограмм позволяет достигать разных целей.

1. Избегать дублирования одинаковых частей программы, оформляя их в виде подпрограмм. Программа станет короче и в то же время более

медленной: на передачу параметров и переходы к подпрограмме и обратно тратится дополнительное время (экономишь память - проиграешь время!).

2. Сделать структуру программы более четкой и понятной, выделив ней самостоятельные части в виде подпрограмм, а основную программу записав кратко и наглядно с помощью обращений к этим подпрограммам.

Разбитая таким образом на части **(***модули*) программа значительно понятнее и удобнее в разработке, чем единый монолитный текст. Ее проще модернизировать и использовать в новых разработках. В этом заключается основная идея *модульного программирования***.**

Модули можно хранить в разных файлах и транслировать раздельно друг от друга.

С точки зрения наглядности программы иногда бывает полезно использовать подпрограмму, даже если она вызывается всего один раз. В то же время это несколько удлиняет программу и замедляет ее работу - вот еще один пример противоречивости разных качеств: простоты программы и ее эффективности по времени и памяти.

3. Расширять язык программирования, добавляя в него новые операции в виде функций и операторы в виде процедур.

Для каждого языка создаются расширяющие его большие библиотеки из десятков и сотен подпрограмм. В виде библиотечных подпрограмм, в частности, обычно реализуют в языках операции ввода и вывода данных.

При разработке сложной программы программист фактически создает таким образом для нее свой язык, реализуемый в виде набора подпрограмм, из которых можно создать библиотеку.

4. Повторно использовать имеющиеся программы при разработке новых программ.

Разработку многомодульной программы из нескольких подпрограмм удобно вести *сверху вниз:* от вызывающих программ к вызываемым. При этом сначала составляется основная программа с использованием вызовов пока еще не разработанных подпрограмм, затем - вызываемые из нее подпрограммы, которые могут обращаться к отсутствующим пока подпрограммам более низкого уровня и т. д.

Такой нисходящий подход надо умело сочетать с разработкой *снизу вверх*: от программ нижнего уровня (не вызывающих другие программы) к использующим их программам более высокого уровня и т. д. до главной программы.

Пример 2. Составить подпрограмму ввода целого числа, перед которым возможны пробелы, символы "новой строки" и знак (подобным образом вводится число по формату %d функции scanf).

Тест:

Вход:

12

-5 +234<Ctrl-z><Enter> <Ctrl-z> - конец файла

Программа 3

```
/*                  Ввод целого числа  znach                  */
void vvod_chisla (int * znach)
{   int sim, znak;
    *znach = 0;  znak = 1;
    /*       Пропуск пробелов и "новых строк" до числа   */
    while ((sim=getchar()) == ' ' || sim=='\n');
    /*       Чтение знака                                 */
    if (sim == '-' || sim == '+')
    {  if (sim == '-') znak = -1;
       sim = getchar();
    }
    /*       Чтение цифр                                  */
    while (sim>='0' && sim<='9')
    {  *znach = *znach * 10 + sim - '0';
       sim = getchar();
    }
    *znach = *znach * znak;
}
```

В программе 3 ввод символа выполняется присваиванием sim=getchar(). Функция getchar вводит символ из стандартного входного файла (клавиатуры) и ее значением является код введенного символа. Аналогичный ввод выполняет вызов scanf("%c", &sim); .

7.3. Передача параметров

Передача параметров - это подстановка фактических параметров вместо формальных параметров при вызове подпрограммы.

Существует около десятка способов передачи параметров, из которых чаще всего в распространенных языках программирования используется передача параметров по значению и по ссылке.

1. При передаче параметра *по значению* (значением) фактический параметр может быть выражением (в частном случае, переменной или константой).

До входа в подпрограмму формальному параметру присваивается значение фактического параметра. Последующие изменения значения формального параметра в подпрограмме не могут повлиять на вызывающую программу.

Таким способом можно передавать только входные параметры, т. е. данные от вызывающей программы в подпрограмму.

Пример 4. Пусть, например, имеется определение подпрограммы P:

```
void P (int x, int y)
```

```
{
   x++;  y++;                /* Тело подпрограммы */
}
```

Вызывающая программа содержит:

```
int m, n;
  ...
m=5; n=7;
P (m, n);
```

Если параметры передаются по значению, то при вызове подпрограммы P (m, n); формальным параметрам присваиваются значения фактических параметров:

x=m; y=n;

а затем выполняется тело подпрограммы, в результате чего x получит значение 6, а y - значение 8, но переменные m, n в вызывающей программе не изменят своих значений.

2. При передаче параметра *по ссылке* (передаче ссылкой, передаче переменной) фактический параметр может быть только переменной, тип которой совпадает с типом формального параметра.

Подпрограмме передается не значение фактического параметра, а его адрес (ссылка на него). При выполнении подпрограммы во всех ее действиях с формальным параметром, в том числе присваиваниях, в действительнсти участвует фактический параметр, который может изменить свое значение.

Подпрограмма выполняется так, как будто бы на месте вызова выполнялось измененное тело подпрограммы, в котором имя формального параметра везде заменено на имя фактического параметра.

Таким способом можно передавать как входные, так и выходные параметры, т. е. данные от вызывающей программы в подпрограмму и в обратном направлении.

Если бы в приведенном выше примере 6.4 параметры передавались по ссылке, то вызов подпрограммы P (m, n); выполнялся бы как ее измененное тело, в котором имя x заменено на m, а имя y заменено на n:

m++; n++;

в результате чего переменные m, n в вызывающей программе получили бы значения m=6 и n=8.

Разные параметры одного вызова могут передаваться разными способами. В различных языках способ передачи параметров задается по-разному.

В языке PL/1, например, способ передачи определяется формой записи фактического параметра: если фактический параметр записан в виде переменной, тип которой совпадает с типом формального параметра, то этот параметр передается по ссылке; в противном случае он передается по значению.

В языке Pascal программист указывает способ передачи каждого параметра не в вызове подпрограммы, а в заголовке ее определения. Если в заголовке подпрограммы перед именем формального параметра указано слово **var** (variable - переменная), этот параметр передается по ссылке. В противном случае он передается по значению.

В языке С параметры всегда передается по значению, что позволяет использовать непосредственно только входные параметры.

Поэтому передачу выходных параметров программисту приходится реализовать самому с помощью указателей.

Для этого выходной формальный параметр (назовем его X) в подпрограмме описывается и используется как указатель на данные того типа, который нужно передавать. В заголовке подпрограммы указывается его описание в виде

тип *X (или тип* X)

Фактический параметр вызова в этом случае должен иметь такой же тип: тип*. Поэтому обычно он задается как адрес некоторой переменной, имеющей нужный тип, например

тип A;

и записывается в виде &A.

Для доступа подпрограммы к фактическому параметру в ее теле обычно используется операция косвенной адресации *X. Она, в частности, позволяет подпрограмме изменить значение переменной, адрес которой передан (по значению!) в качестве фактического параметра. В этом случае, например, *X равносильно A и присваивание

*X = ...; равносильно A = ...;

Упрощенное правило: перед именем выходного параметра в заголовке и теле подпрограммы ставится *, а в вызове &.

Так, если в примере 3 требуется сделать параметр у выходным (реализовать передачу параметра у по ссылке), программа запишется следующим образом.

```
void P (int x, int *y)
{
   x++;  (*y)++;              /* Тело подпрограммы       */
}
/*  Вызывающая программа                                 */
  . . .
int m, n;
  . . .
m=5;  n=7;
P (m, &n);
/* После вызова переменные будут равны:   m=5,   n=8 */
```

Если параметр является массивом, в качестве фактического параметра передается его имя (которое является указателем начала массива), и не требуется операция получения адреса &, например:

```
/*                    Подпрограмма                    */
void Q (char t[100], . . . )     /* Можно написать char t[] */
{       . . .
     t[5]='#';
        . . .
}
                              . . .
/*                Вызывающая программа                */
     . . .
char s[100];
     . . .
Q (s, . . .);                  /* Вызов.    s[5] присвоится '#'   */
```

Пример 5. Умножение матриц. Даны две матрицы вещественных чисел: X[10][20) и Y[20][30). Составить программу вычисления их произведения - матрицы Z[10][30), элементы которой определяются по формуле (6.4).

$$Z_{i,j} = \sum_{k=0}^{19} X_{i,k} * Y_{k,j} \qquad (7.4)$$

для i=0, ..., 9 и j=0, ..., 29

Непосредственно по формуле (7.4) получаем программу 4, оформленную как подпрограмма. Подпрограмма содержит три вложенных цикла. Цикл по i выполняется 10 раз, цикл по j повторяется 10*30=300 раз, самый внутренний цикл по k повторяется 10*30*20=6000 раз.

Основная часть времени работы подпрограммы затратится на выполнение тела самого внутреннего цикла, главным образом, на доступ к элементам массивов - вычисление их адресов в зависимости от индексов.

Поэтому использование в этом теле *скалярной переменной* (переменной без индексов) s, вместо переменной с индексами Z[i][j], может привести к заметному ускорению работы всей подпрограммы. Однако хороший транслятор и сам может произвести аналогичную замену непосредственно в машинной программе.

Поэтому в подобных случаях полезно экспериментально на компьютере оценить, имеет ли смысл такая экономия, несколько усложняющая текст программы.

Программа 4. Умножение матриц

```
/*          Подпрограмма умножения матриц: Z = X * Y          */
void umn_matr (float X[10][20],  float Y[20][30],  float Z[10][30])
{   int  i, j, k;
    float  s;
    for (i=0;  i<10;  ++i)
        for (j=0;  j<30;  ++j)
        {  s = 0;
            for (k=0;  k<20;  ++k)
                s = s + X[i][k] * Y[k][j];
            Z[i][j] = s;
        }
}
                                . . .

/*                  Фрагмент вызывающей программы                  */
float A[10][20],  B[20][30],  C[10][30];
                    . . .
umn_matr (A, B, C);          /* Умножение матриц: C = A * B       */
```

Недостаток данной функции состоит в том, что ее можно использовать только для матриц фиксированного размера, невозможно задавать размеры матриц в виде параметров.

Язык C++ включает усовершенствованный C, в котором можно использовать передачу параметра по ссылке, как в языке Pascal, только вместо слова **var** перед именем формального параметра (точнее, после его типа) в заголовке подпрограммы пишется амперсенд &. В этом случае в теле подпрограммы не требуется писать звездочку * перед формальным параметром, а в фактическом параметре не нужен амперсенд & (они вставляются транслятором автоматически). Так, чтобы в примере 4 передавать параметр y по ссылке, а x - по значению, в языке C++ можно записать следующим образом.

```
void P (int x, int & y)
{  x++;  y++;   }          /* Тело подпрограммы  */
. . .
P (m, n);                  /*  Вызов подпрограммы */
```

Иногда возникает необходимость иметь *переменное имя подпрограммы*, в частности, чтобы сделать его параметром другой подпрограммы. Примером может служить имя подынтегральной функции для подпрограммы вычисления интеграла. Для этого в языке C/C++ можно использовать указатель на функцию.

Например, описание указателя на функцию, возвращающую целое значение, имеет вид **int** (*f) ( );

Использование указателя на функцию можно проиллюстрировать следующим примером, в котором f определяется как переменный указатель (адрес) целочисленной функции от двух вещественных аргументов.

```
int (*f) (float, float);      /* Определение указателя на функцию */
int  g (float, float);        /* Прототип функции g               */
int  h (float, float);        /* Прототип функции h               */
float a, b, c, d;
   . . .
f = g;      (*f) (a, b);          /* Вызов функции g */
   . . .
f = h;      (*f) (c, d);          /* Вызов функции h */
```

В некоторых языках, в том числе в C/C++, имеется полезная возможность создавать подпрограммы с *переменным числом параметров*. Примерами таких подпрограмм являются стандартные функции scanf() и printf ().

7.4. Блочная структура программы и области действия имен

Каждый объект программы (константа, переменная, массив, структура, функция, тип, метка) имеет определенную *область действия* - часть программы, в которой он доступен. Объект может иметь имя. *Областью действия* (областью видимости) имени называется часть программы, в которой имя соответствует определенному объекту. В других частях программы это же имя может обозначать другие объекты.

В ряде языков программирования (Fortran, Basic и т. п.) областью действия имени является вся программа. В языках с *блочной структурой программы* (Algol-60, PL/1, Pascal, C и др.) областью действия имени является *блок*, в котором оно описано, исключая вложенные в него блоки, содержащие описание такого же имени. Другими словами, имя считается *локализованным* в блоке, в котором оно описано. В частности, блоком является тело функции. Блок является оператором и может содержаться в другом блоке.

блок ::= { [описание...] [оператор...] }

Блочная структура программы позволяет следующее.

1. Использовать имя для разных целей в разных частях программы (например, разрабатываемых разными программистами).
2. Экономить память за счет хранения данных разных блоков по очереди в одной и той же области.

При разработке больших программ используется модульное программирование: их разбивают на независимые части - программные модули. Модулем может быть, например, подпрограмма. Подробнее модульное программирование рассматривается во второй части учебника.

Модуль - это часть программы, которая может размещаться в отдельном файле и транслироваться отдельно от других ее частей.

Разные модули могут иметь общие объекты, обозначаемые одним и тем же именем. *Имя*, используемое в данном модуле, но определенное в другом модуле, называется *внешним* (для данного модуля). Внешнее имя описывается в каждом использующем его модуле.

Область действия внешнего имени объединяет области действия всех его описаний.

В языке С при описании внешнего имени перед его типом пишется служебное слово **extern** (external - внешний). По умолчанию внешними являются имена, описанные вне функции - *глобальные имена*.

Упражнения и задачи

7.1. Составить программу вычисления факториала

7.2. Составить программу вычисления площади круга. Радиус вводить с клавиатуры. Вычисление площади круга оформить в виде определения функции.

7.3. Составить определения следующих функций.

а) Расстояние между двумя точками, заданными своими координатами на плоскости.

б) max (x, y).

в) min (x, y).

г) max (a, b, c).

д) min (a, b, c).

е) 1, если заданный символ является латинской буквой, и 0 в противном случае.

ж) 0, если нельзя построить треугольник из трех отрезков заданной длины (каждая сторона треугольника должна быть меньше суммы двух других сторон); 1 - треугольник равносторонний, 2 - равнобедренный, 3 - любой другой.

з) Число Фибоначчи f(n), где f(0) = 0, f(1) = 1, f(j) = f(j-1) + f(j-2) для целого j>1.

и) Наибольший общий делитель натуральных чисел A и B.

к) Количество десятичных цифр заданного целого числа.

л) Значение числа, полученного выписыванием в обратном порядке десятичных цифр заданного натурального числа.

м) Количество единичных битов в двоичной записи заданного числа типа unsigned long

н) Ввод и вычисление значения двоичного целого числа.

7.4. Составить определение функции F(X) = √X с погрешностью не более 0.00001.

7.5. Дана последовательность из 150 действительных чисел. Вычислить сумму и среднее арифметическое значение первых 100 чисел и последующих 50 чисел.

7.6. Составить подпрограммы для решения следующих задач.

а) Поменять местами значения переменных x, y.

б) Значения переменных x, y, z поменять местами так, чтобы оказалось x ≥ y ≥ z.

в) Вычислить длину окружности, площадь круга и объем шара одного и того же заданного радиуса.

г) Вычислить площадь и периметр прямоугольного треугольника по длинам двух катетов.

д) Для квадрата ABCD на плоскости даны координаты XA, YA, XC, YC диагонально расположенных вершин A и C. Вычислить координаты вершин B и D.

е) Вывод заданного целого числа в двоичной системе счисления.

ж) Вывод заданного целого числа в шестнадцатеричной системе счисления.

7.7. Пусть p, q, r - целочисленные беззнаковые переменные. Составить подпрограммы для решения следующих задач.

а) *Упаковка*. Поместить младшие 3 бита переменной p в младшие 3 бита переменной r, а младшие 6 битов переменной q - в следующие 6 битов переменной r без изменения остальных битов.

б) *Распаковка*. Присвоить младшим битам переменной p младшие 3 бита переменной r, а младшим битам переменной q - следующие 6 битов переменной r. Остальные биты p и q обнулить.

7.8. Дано целое N>0 и массив из N целых чисел. Составить подпрограммы вычисления следующих величин.

а) Сумма чисел, кратных 5.

б) Сумма чисел, являющихся нечетными и отрицательными.

в) Сумма чисел, удовлетворяющих условию: модуль числа меньше его номера.

г) Количество чисел, являющихся отрицательными.

д) Количество чисел, являющихся нечетными.

е) Количество чисел, кратных 3 и не кратных 5.

ж) Произведение чисел, имеющих четные порядковые номера и являющихся нечетными числами.

з) Сумма и количество чисел, кратных 5.

и) Количество положительных и количество отрицательных чисел.

к) Порядковые номера и количество чисел, кратных трем.

7.9. Дано целое N>0 и массив из N вещественных чисел. Составить подпрограммы для получения следующих величин.

а) Среднее арифметическое квадратов положительных чисел.

б) Сумма элементов с четными номерами и суммы элементов с нечетными номерами.

в) Разность наибольшего и наименьшего числа.

г) Наибольшее среди четных по порядку чисел.

д) Наименьшее среди нечетных по порядку чисел.

е) Наибольшее среди модулей (абсолютных величин) чисел.

ж) Наибольшее среди отрицательных чисел.

з) Сумма положительных и количество отрицательных чисел.

7.10. Дан массив вещественных чисел и количество его элементов. Составить подпрограммы для решения следующих задач.

а) Определить порядковый номер наименьшего числа.

б) Определить, образуют ли числа возрастающую последовательность.

в) Определить, со скольких отрицательных чисел она начинается.

г) Определить, сколько раз в массиве меняется знак.

д) Определить, сколько чисел больше своих соседей.

е) Найти количество чисел в наиболее длинной последовательности подряд идущих нулей.

ж) Найти, сколько чисел принимают наибольшее значение.

7.11. Дана строка символов (символьный массив, завершающийся нулевым байтом). Составить подпрограммы для решения следующих задач.

а) Найти общее количество символов и количество пробелов в строке.

б) Подсчитать количество цифр и количество латинских букв в строке.

в) Найти порядковый номер первой запятой.

г) Найти порядковый номер последней запятой.

д) Найти количество символов до первой запятой.

е) Определить, сколько раз в строке встречается сочетание символов ":=".

ж) Строка содержит выражение со скобками. Определить максимальное число уровней вложенности скобок.

з) Выяснить, имеется ли в заданном строке пара соседних одинаковых символов.

и) Удалить из строки пробелы, сдвинув остальные символы.

7.12. Данная строка символов представляет собой последовательность слов, разделенных произвольным числом пробелов. Составить подпрограммы определения следующих величин:

а) количества слов в строке;

б) количества слов, начинающихся с буквы ‘A’;

в) количества слов, оканчивающихся буквой ‘W’;

г) количества слов, начинающихся и оканчивающихся одной и той же буквой;

д) количества слов, содержащих заданную букву;

е) количества слов, имеющих длину больше трех, но меньше семи символов.

ж) максимальной длины слова.

7.13. Составить подпрограмму ввода действительного числа с возможным знаком

а) с фиксированной точкой, например: -3.14 или 345.782;

б) с плавающей точкой, например: -31.4E-1 или 1e6.

7.14. Составить подпрограмму вывода действительного числа с возможным знаком;

а) с фиксированной точкой, например: -3.14 или 345.782;

б) с плавающей точкой, например: -31.4E-1 или 1e6.

7.15. Входной текст - записанное без пробелов выражение из целых чисел. Концом выражения служит любой символ, не являющийся цифрой и знаком операции. Составить программу вычисления значения выражения.

а) Допускаются операции: + и -.

Пример. Вход:-12+50-1= Выход: 37.

б) Допускаются операции: +, -, * и /.

Пример. Вход:-48/8*2+5*10-1= Выход: 37.

в) Допускаются операции: +, -, *, / и скобки (и).

Пример. Вход:-48/(8*2-12)+5*(7+3*(12-7)/5)-1= Выход: 37.

7.16. Для многоразрядных (длинных) чисел составить подпрограммы реализации следующих операций.

а) сравнение двух длинных чисел: X=Y, X<Y, X>Y, X≤Y, X≥Y;

б) умножение длинного числа на короткое (двухбайтовое);

в) умножение двух длинных чисел;

г) вычитание двух длинных чисел X-Y (для X≥Y);

д) целочисленное деление с остатком двух длинных чисел («столбиком», как учат в школе; очередную цифру частного лучше подбирать методом дихотомии);

е) преобразование длинного числа в число типа long и обратно;

ж) извлечение квадратного корня из длинного числа («столбиком»);

з) получение наибольшего общего делителя двух длинных чисел.