Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информационные технологии»

О. А. Кравченко, Д. А. Литвинов

ПОДПРОГРАММЫ И ПРОГРАММИРОВАНИЕ С ИХ ИСПОЛЬЗОВАНИЕМ НА ЯЗЫКЕ С

ПОСОБИЕ
по курсу «Основы алгоритмизации и программирования»
для студентов специальностей 1-36 04 02
«Промышленная электроника» и 1-40 01 02
«Информационные системы и технологии
(по направлениям)»
дневной и заочной форм обучения

Электронный аналог печатного издания

Гомель 2009

УДК [004.45+004.43](075.8)
ББК 32.973.26-018.2я73
К78

Рекомендовано к изданию научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 5 от 25.06.2007 г.*)

Рецензент: канд. техн. наук, доц. каф. «Промышленная электроника»
ГГТУ им. П. О. Сухого *Н. И. Вяхирев*

Кравченко, О. А.

К78 Подпрограммы и программирование с их использованием на языке С : пособие по курсу «Основы алгоритмизации и программирования» для студентов специальностей 1-36 04 02 «Промышленная электроника» и 1-40 01 02 «Информационные системы и технологии (по направлениям)» / О. А. Кравченко, Д. А. Литвинов. – Гомель : ГГТУ им. П. О. Сухого, 2009. – 45 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://lib.gstu.local. – Загл. с титул. экрана.

ISBN 978-985-420-834-3.

Рассматриваются вопросы разработки и конструирования программ с использованием подпрограмм на языке С, синтаксис описания, объявление и вызов функции, механизм передачи параметров разных типов.

Для студентов специальностей 1-36 04 02 «Промышленная электроника» и 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной и заочной форм обучения.

УДК [004.45+004.43](075.8)
ББК 32.973.26-018.2я73

ISBN 978-985-420-834-3

1. Функции

1.1. Общие сведения

При разработке больших и сложных алгоритмов и программ логически независимые или повторяющиеся последовательности действий оформляют в виде вспомогательных алгоритмов (для алгоритмов) и подпрограмм (для программ).

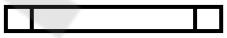
Качество программы зависит от умения программиста разбить всю программу на логически обоснованные модули (функции) с хорошо обоснованным интерфейсом и хорошо документированным текстом программы.

Функции нужны для упрощения структуры программы. Разбив задачу на подзадачи и оформив каждую из них в виде функций, мы поступаем по принципу, известному еще с древних времен: «Разделяй и властвуй».

Передача в функцию различных аргументов позволяет, записав ее один раз, использовать многократно для разных данных.

Внимание*! *Не бойтесь написать функции, которые вызываются один раз. Не пытайтесь запихнуть всю программу в одну функцию main. Это очень плохой стиль. Наоборот, старайтесь разбить программу на небольшие единицы – функции, которые соответствуют логическим частям программы.

В графической схеме алгоритма обращение к вспомогательному алгоритму оформляется в виде блока:



В программе на языке С вспомогательному алгоритму соответствует подпрограмма – функция.

Пример. Вычислить значение $f = z^2$, где $z = a^2 + b^2$.

Решение. Разработаем вспомогательный алгоритм возведения числового значения (x) в квадрат. Назовем алгоритм ***sqr***. На рис. 1.1. изображена графическая схема этого алгоритма.

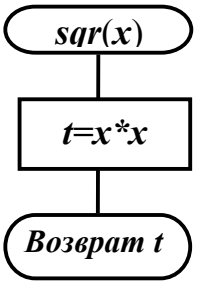


Рис.1.1. Вспомогательный алгоритм

Здесь *x* – формальный параметр – исходное данное, *t* – локальный параметр – результат.

Основной алгоритм будет иметь вид, изображенный на рис. 1.2.

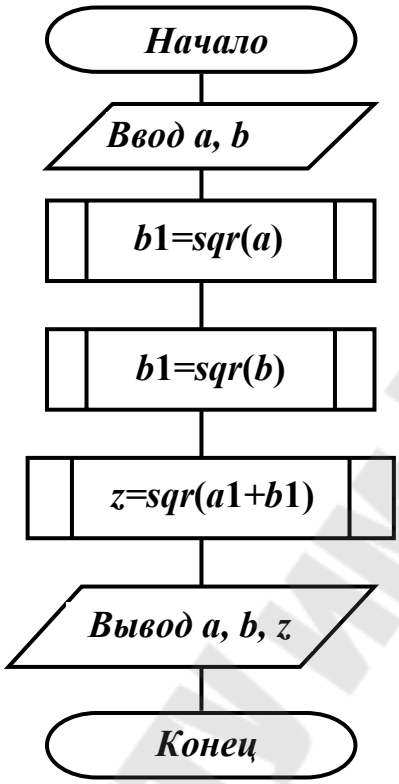


Рис. 1.2. Основной алгоритм

При выполнении основного алгоритма будет три раза вызываться вспомогательный алгоритм ***sqr***. Первый раз формальный аргумент ***x*** будет заменен значением ***a***. Функция ***sqr*** возвратит значение a^2, затем это значение присваивается переменной ***a*1**. Второй раз формальный аргумент ***x*** будет заменен значением ***b***. Функция ***sqr*** возвратит значение b^2, затем значение b^2 присваивается переменной ***b*1**. При вызове ***sqr*** в третий раз формальный аргумент ***x*** будет заменен предварительно вычисленным значением ***a*1+*b*1**, затем значение $(a1+b1)^2$ присваивается переменной ***z***.

Забегая вперед, покажем, как будет выглядеть программа на языке С, использующая подпрограмму, разработанную пользователем.

```
/* Программирование с использованием подпрограмм. Пример 1 */
    #include <stdio.h>
    void main()
    {
     float sqr(float x);   // Объявление функции (Прототип функции)
     float a,b,z,a1,b1;
```

```
  puts("Введите значения a и b");
  scanf("%f %f",&a,&b);
  a1=sqr(a);     b1=sqr(b);
  z=sqr(a1+b1);
  printf("z=%.3f при a=%.3f b=%5.3f\n",z,a,b);
  puts("******");
  fflush(stdin); getchar();
}    // Отсутствует return, т. к. main() описана как функция
     // без возвращающего значения

/*    Описание функции sqr(x)  */
float sqr(float x)
{
  float t;
  t=x*x;
  return t;   // или return(t);
}
```

Приведенная программа состоит из двух функций: ***main()*** и ***sqr(x)*** (основной и разработанной пользователем). Первая вызывает вторую для выполнения. Вызов осуществляется по имени функции с использованием фактического аргумента. В вызванной функции формальный аргумент заменяется фактическим, выполняются описанные в ней вычисления и осуществляется возврат значения по оператору ***return*** в точку вызова функции.

Итак, программа на языке С состоит из одной главной функции ***main*** и любого количества других функций.

Функция – это логически самостоятельная часть программы, имеющая имя, которой могут передаваться параметры и которая может возвращать какое-то значение.

Можно представлять функцию как некоторую машину или «черный ящик», в который вы вкладываете данные и из которого получаете результат. Внутренне устройство и способ действия функции невидимы и неизвестны для остальной части программы. Внешний мир должен знать о некоторой функции лишь только то, что в нее вводится, что из нее выводится и какие необычные эффекты вызывает выполнение этой функции.

Функции, предназначенные для решения определенного класса задач, можно выделить в отдельный файл, который может представлять собой библиотеку пользователя.

Библиотеки функций позволяют:

– использовать одни и те же функции разными программистами;

– повысить структурированность и наглядность программ;

– облегчить чтение, освоение и корректировку программы.

Обычно функция ***main*** вызывает другие функции, которые в свою очередь вызывают другие подпрограммы и т. д. «***Вызывающая***» функция временно приостанавливает свою работу и управление передается «***вызванной функции***». «***Вызывающая***» функции «отдыхает», пока работает «***вызванная функция***». Когда вызванная функция выполнит все свои операторы или когда она встретит ключевое слово ***return*** (возврат), управление возвращается к вызывающей функции. Вызывающая функция продолжает выполнять свой код с той точки, в которой она приостановилась.

1.2. Описание функции

Описание (определение) функции – это ее текст на языке С. В нем определяется имя, формальные параметры, операторы тела функции и тип возвращаемого результата.

Синтаксис описания:

//Заголовок функции

[*тип*]<имя_функции>([список формальных параметров или void])

{ // *Начало тела функции*

<описание данных>

<операторы>

[return(выражение)]

} // *Конец тела функции*

В квадратных скобках записано то, что может быть опущено. {…} – тело функции (совокупность действий в фигурных скобках).

Заголовок функции имеет вид:

[*тип*]<имя_функции>([список формальных параметров или void])

Здесь ***тип*** – задает тип возвращаемого значения с помощью оператора ***return***.

Тип – имя одного из простых допустимых типов данных (арифметический, символьный, указатель на любой допустимый тип (в том числе на скаляр, массив, структуру, файл или функцию)). Если тип отсутствует, то считается, что функция возвращает значение ***int*** с помощью оператора ***return***.

Если <***тип***> – это слово ***void***, то функция не возвращает никакого значения. Использование ***void*** при описании возвращаемого значения позволяет получить более короткий и производительный код программы. Это возможно тогда, когда функция действительно не должна возвращать значений или для возврата значений используются указатели либо внешние переменные.

Имя функции:

<имя_функции> – идентификатор, с помощью которого функция вызывается на выполнение.

<имя_функции> является особым типом указателя, называемым указателем на функцию. Его значением является адрес точки входа в функцию.

Например sqr, sin, printf, …

Список формальных параметров (аргументов)

Список определяет типы и имена формальных параметров (формальных аргументов), с помощью которых производится обмен данными между вызываемой и вызывающей функциями в процессе выполнения программы.

Элементы списка формальных параметров разделяются запятыми.

Каждый элемент списка формальных параметров – это объявление одного формального параметра в виде:

<тип><имя_формального_параметра>

Например: int 1, float x, char *d.

Список формальных параметров может отсутствовать или содержать ключевое слово *void*. В этом случае в функцию не передаются никакие аргументы.

Оператор *RETURN*

Синтаксис оператора:

***Return* (<выражение>);**

С помощью оператора return формируется результат выполнения функции в виде одного скалярного значения любого типа.

В теле функции может быть один или несколько операторов *return*.

<Выражение> вычисляется, преобразуется к типу возвращаемого значения и передается в точку вызова функции. Тип результата выражения должен быть совместим с типом возвращаемого значения функции. Скобки необязательны, но возможны случаи, когда выполнение программы прекращается без выдачи каких-либо сообщений, если нет скобок.

Примеры операторов return для возврата результата:

Return(x*x); return t; return(*s); return (0);

Внимание! Функцию, возвращающую результат с помощью оператора return, можно использовать в выражениях. Например:

a1=sqr(a); z=sqr(sqr(a)+sqr(b)); R=sqr(sin(x+1.3));

1.3. Объявление функций

Стандарт ANSI языка С требует, чтобы функции, возвращающие отличные от ***int*** значения, были объявлены до первой ссылки на них.

Это предварительное объявление функции называется прототипом функции и представляет собой заголовок функции, после которого поставлен символ ";".

Прототип функции делает возможным доступ к функции, помещает ее в область видимости файла программы.

Для определения прототипов встроенных функций надо подключить их головные файлы с помощью директивы предпроцессора. Например:

#include <stdio.h>

#include <string.h>

Объявление функции пользователя может быть внешним, т. е. располагаться в программном файле вне функций.

Объявление функции пользователя может быть внутренним, т. е. располагаться в начале тела вызывающей функции до ее выполняемых операторов.

Область действия объявления определяется блоком функции, в которой она объявлена, или файлом, в котором она объявлена как внешняя.

Синтаксис объявления функции:

[*extern*] <заголовок_функции>;

Таким образом, объявление функции – это заголовок функции, после которого стоит символ ";".

Внимание*! *Целесообразно в процессе отладки очередной функции сначала написать и оттранслировать ее текст, а затем заголовок функции скопировать для объявления функции.

Примеры объявления функций:

1. // функция не возвращает результата
   void obr(float a[m][n],float b[m]);
2. float sqr(float x); // результат типа float
3. char *str(char *c); // результат – указатель на тип char
4. // функция без параметров и не имеет результата
   void d(void);

5. // параметр – указатель на любой тип
int p(void *c);
6. int (*fp)(int i,int s);
// Указатель на функцию, возвращающий результат типа int.
7. FILE *fu(); //возвращаемый результат – указатель на файл
8. ff (int i); // результат типа int по умолчанию.

Объявление вызываемой функции обязательно должно быть в вызывающей функции, если текст (определение) вызываемой функции расположен в данном файле, но после текста вызывающей функции или в другом исходном файле программы. Если текст вызываемой функции расположен до текста вызывающей функции, объявление вызываемой функции не обязательно.

Предварительное объявление функции позволяет компилятору проверить соответствие формальных и фактических параметров.

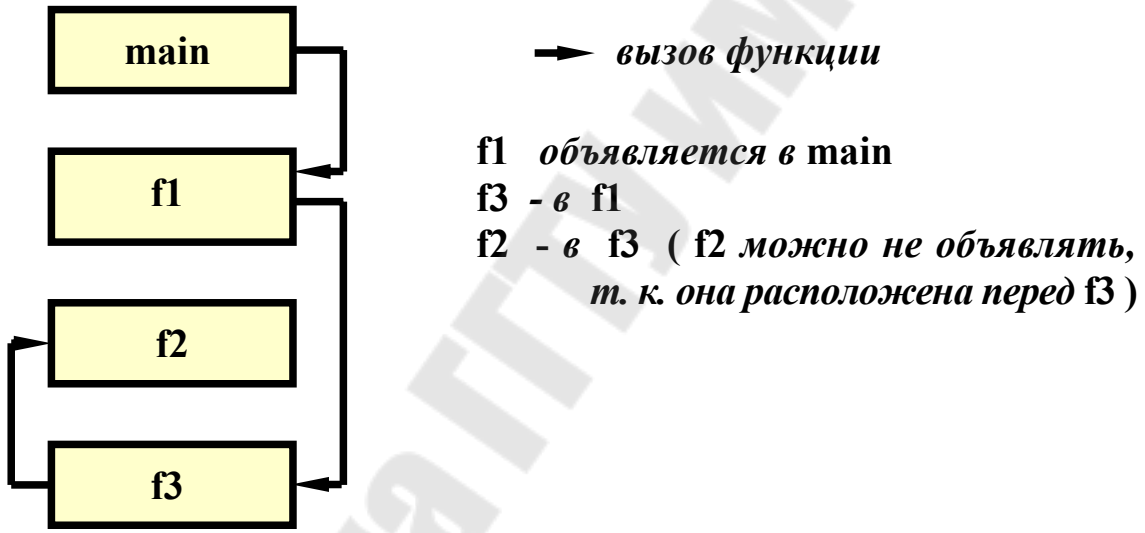


Рис. 1.3. Пример схемы вызова одних функций другими функциями

Внимание*! *Общеупотребительная практика – помещение прототипов функций в заголовочный файл, подключаемой директивой #include к тексту программы. Или в самом начале программы до main.

В тексте программы может содержаться произвольное количество объявлений одной и той же функции, но только одно ее описание.

1.4. Вызов функции

Вызов функции активизирует функцию, т. е. передает ей управление и фактические параметры, если они есть. Чтобы вызвать функцию, надо указать ее имя, передать ей список фактических парамет-

ров в соответствии с описанными в заголовке формальными параметрами. Соответствие между фактическими и формальными параметрами должно соблюдаться в количестве, типе, порядке следования.

Для вызова функции используется оператор вызова функции. Его синтаксис:

<имя_функции>([список фактических параметров]);

При выполнении вызова происходит передача фактических параметров, если они есть. Формальные аргументы заменяются на фактические.

Например:

Clrscr(); // вызов функции без параметров очистки экрана.

Puts(“Введите строку”);

Если функция имеет результат, возвращаемый с помощью оператора ***return***, то ее можно вызывать и из выражений и с помощью оператора вызова.

Например:

```
Y=sin(x);
printf(“%.4f\n”, 2*sin(x));
z=sqr(x+2);
i=printf(”%.4f\n”, 2*sin(x));
```

В последнем примере с помощью оператора return функция printf возвращает количество успешно выведенных значений списка данных.

1.5. Область действия переменных

Каждая переменная, используемая в теле функции, должна быть объявлена как формальный аргумент либо внутри, либо вне функции.

Переменные, которые не являются формальными параметрами, не описаны в теле функции, но используются в теле функции, называют ***глобальными***.

Глобальные переменные описываются в файле как внешние.

Переменные, которые не являются формальными параметрами, объявленные в теле функции, называются ***локальными***. Они доступны только той функции, в теле которой они описаны.

Пример программы:

```
/* Пример 2.   Область действия переменных */
int a=3;              //Внешнее описание a (плохой стиль)
#include <stdio.h>
void main()           // void может отсутствовать
{
  int f1(int x);      //Объявление функции f1
```

```
 printf("%d\n",f1(4));
 fflush(stdin); getchar();
}    // Отсутствует return, т. к. main() описана как функция
     // без возвращающего значения

/*    Описание функции f1(x)  */
int f1(int x)
{
 return (x+a);
}
```

После выполнения программы должны получить значение 7.

Внимание*! *Все, что передается в функцию и обратно, должно отражаться в ее заголовке. Это требование не синтаксиса, а хорошего тона.

2. Механизм передачи параметров в функцию

2.1. Основные определения

Рассмотрим механизм передачи параметров на примере программы.

```
/* Программирование с использованием подпрограмм.  Один результат передается с помощью return, а другой – через заголовок
 (по значению и по ссылке)
      */
const double pi=3.141592653589793;
float PL(float r, float &L);        // Объявление функции
#include <stdio.h>
void main()
{
 float r,s,L;
 puts("Введите радиус круга r");
 scanf("%f",&r);
 s=PL(r,L);
 printf("Площадь круга=%.5g, длина окружности=%.5g, радиус круга=%.2f\n",s,L,r);
 fflush(stdin); getchar();
}    // Отсутствует return, т. к. main() описана как функция
     // без возвращающего значения
/*    Описание функции PL(r,L)  */
float PL(float r, float &L)
```

```
{
  L=2*pi*r;
  return (pi*r*r);
}
```

При r=1 получим s=3.14159 L=6.2832

Здесь r передается по значению, а L – по ссылке. Имена формальных и фактических параметров совпадают, что допустимо.

Обрабатывая вызов функции, компилятор вставляет в код программы последовательность машинных команд, выполняющих следующие действия:

1. Запись в стек копий переменных или констант, перечисленных в списке аргументов (если список аргументов в прототипе функции не void). Стек – структура данных.

2. Вызов функции.

Функция в процессе исполнения извлекает копии аргументов из стека, т. е. значения фактических аргументов последовательно ставятся в соответствие формальным аргументам.

Передача параметров может быть двух видов: по значению и по наименованию (по ссылке, по адресу).

Когда в списке параметров функции записано выражение вида ***double x*** или ***int y*** и т. п., это значит, что в функцию при ее вызове должно быть передано значение соответствующего аргумента. Для этого в стеке создается его копия, с которой и работает функция. Естественно, что изменение этой копии не может оказать никакого влияния на ячейку памяти, в которой хранится сам параметр. Именно поэтому на месте такого параметра можно при вызове функции задавать и выражение. Например:

Z=sqr(x+i);

Выражение вычисляется, и его результат записывается в стек на место, выделенное для соответствующего параметра.

Ссылка – это выражение вида ***float &L***, ***int &d*** и т. п. в списке формальных параметров.

При вызове функции на месте такого параметра записывается имя переменной, например s=pl(r,L) (прототип float PL(float r, float &L);) тем самым в функцию передается адрес переменной. Этот адрес обрабатывается также, как и остальные параметры: в стеке создается его копия.

Функция, работая с копией адреса, имеет доступ к ячейке памяти, в которой хранится значение переменной, и тем самым может его изменить.

Можно передавать в функцию указатель. В этом случае в теле функции следует применять операции разадресации и взятия адреса явным образом.

Пример:

```
/* Программирование с использованием подпрограмм. Пример 4
  Один результат передается с помощью return, а другой-через заголовок
  (по значению и по адресу (указателю))          */
const double pi=3.141592653589793;
float PL(float r, float *L);        // Объявление  функции
#include <stdio.h>
void main()
{
 float r,s,L;
 puts("Введите радиус круга r");
 scanf("%f",&r);
 s=PL(r,&L);                //& – Взятие адреса
 printf("Площадь круга=%.5g, длина окружности=%.5g, радиус круга=%.2f\n",s,L,r);
 fflush(stdin); getchar();
}    // Отсутствует return, т. к. main() описана как функция
     // без возвращающего значения

/*    Описание функции PL(r,L)  */
float PL(float r, float *L)
{
 *L=2*pi*r;            //* L – значения по адресу L (* – разадресация)
 return (pi*r*r);
}
```

Внимание*! *Входные данные в функцию надо передавать по значению или по константной ссылке, результаты ее работы – через возвращаемое значение по оператору return, а при необходимости возвращать более одной величины – через параметры по ссылке или указателю.

2.2. Структуры – параметры функции

Пример. Найти сумму двух комплексных чисел.

```
/*1-ый вариант */
#include<stdio.h>
struct complex sumc(struct complex p1, struct complex p2);
```

```
struct complex
   {
      float a;
      float b;
};
void main()
{
 struct complex n1, n2, s;
 n1.a = 2;
 n1.b = 3;
 n2.a = 1;
 n2.b = 5;
 s = sumc(n1,n2);
 printf("\n%gi+%g",s.a,s.b);
}

struct complex sumc(struct complex p1, struct complex p2)
{
  struct complex tmp;
  tmp.a = p1.a+p2.a;
  tmp.b = p1.b+p2.b;
  return (tmp);
}
/* 2-ой вариант*/
/* Входные и выходные данные передаются через заголовок функции */
#include<stdio.h>
void sum(struct complex *p1, struct complex *p2, struct complex *ps);
struct complex
          {
            float a;      // Вещественная часть числа
            float b;     //Мнимая часть числа
          };

void main()
{
 struct complex n1,n2,s;
 n1.a = 2;
 n1.b = 3;
 n2.a = 1;
```

```
 n2.b = 5;
 sum(&n1, &n2, &s);
 printf("\n%gi+%g",s.a,s.b);
}

void sum(struct complex *p1, struct complex *p2, struct complex *ps)
{
 ps->a = p1->a+p2->a;      //  (*ps).a = (*p1).a+(*p2).a;
 ps->b = p1->b+p2->b;      //  (*ps).b = (*p1).b+(*p2).b;
}
```

2.3. Массивы – параметры функции

При передаче массива в функцию размер массива можно указать при описании параметра или описать параметр с пустыми скобками в заголовке функции. В последнем случае добавляется дополнительный параметр, через который передается в функцию количество элементов массива.

Например:

float Mas (float V[15]);

float Mas1(float V[], int n);

Альтернативный способ – передача функции массива с использованием параметра-указателя.

Например:

float Mas3(float *a, int n);

Возникают проблемы неполного использования памяти при реализации универсальных функций для обработки многомерных массивов.

Существует несколько способов решения проблемы передачи в качестве параметров многомерных массивов.

1-й путь. Описание соответствующего параметра должно в явном виде указывать все размерности, кроме старшей, например:

int b[][40][20].

2-й путь. Замена многомерного массива одномерным и имитация доступа к многомерному массиву внутри функции.

3-й путь. Использование вспомогательных массивов указателей на массивы.

4-й путь. Использование классов для представления массивов.

Передача одномерных массивов в функцию

Пример. Даны два массива чисел. Определить, сумма элементов какого из них меньше.

Решение. Пусть имеем массивы A[n] и B[m]. Необходимо:

- Ввести массив A из n элементов.

- Ввести массив B из m элементов.
- Вывести массив A.
- Вывести массив B.
- Найти сумму s1 элементов массива A.
- Найти сумму s2 элементов массива B.
- Сравнить найденные суммы и вывести соответствующее сообщение.

Подзадачи:

1. Ввод массива (vvod1m)
2. Вывод массива (vivod1m)
3. Нахождение суммы элементов массива (sum)

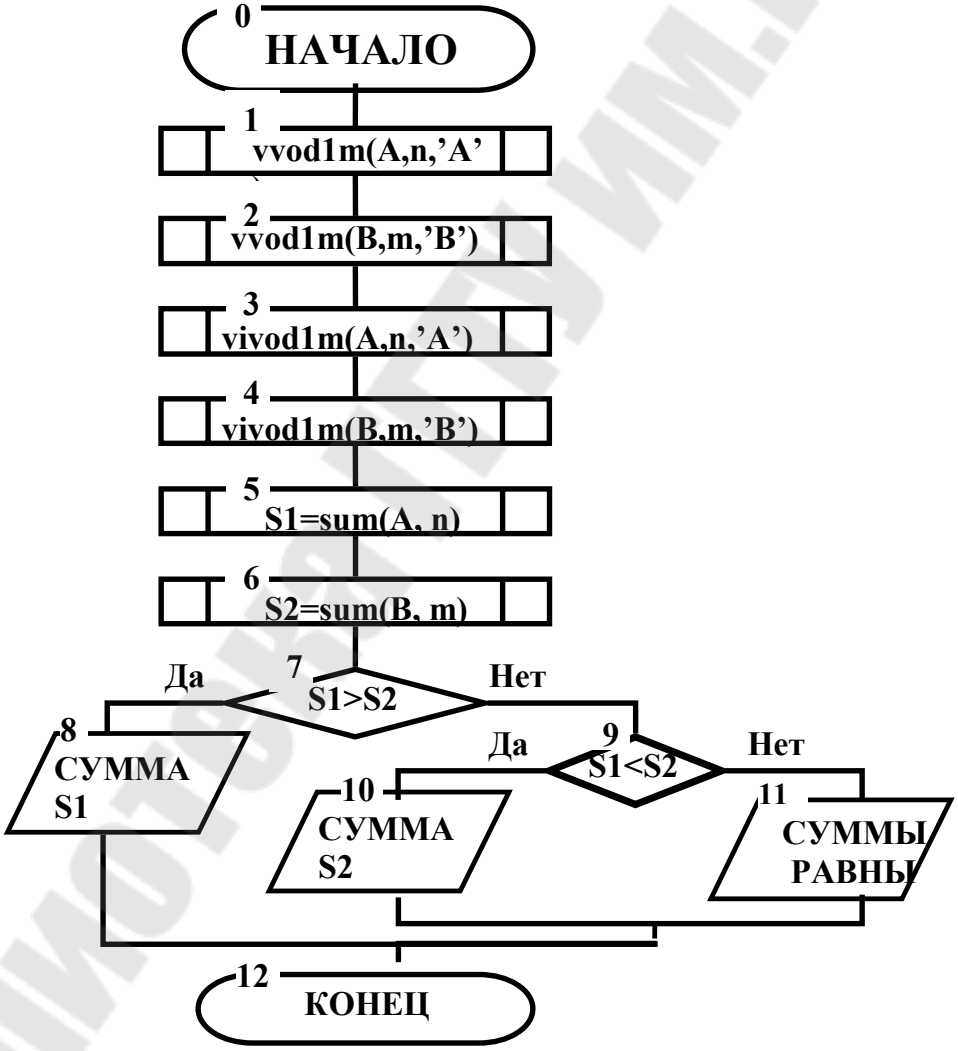


Рис. 2.1. Графическая схема основного алгоритма

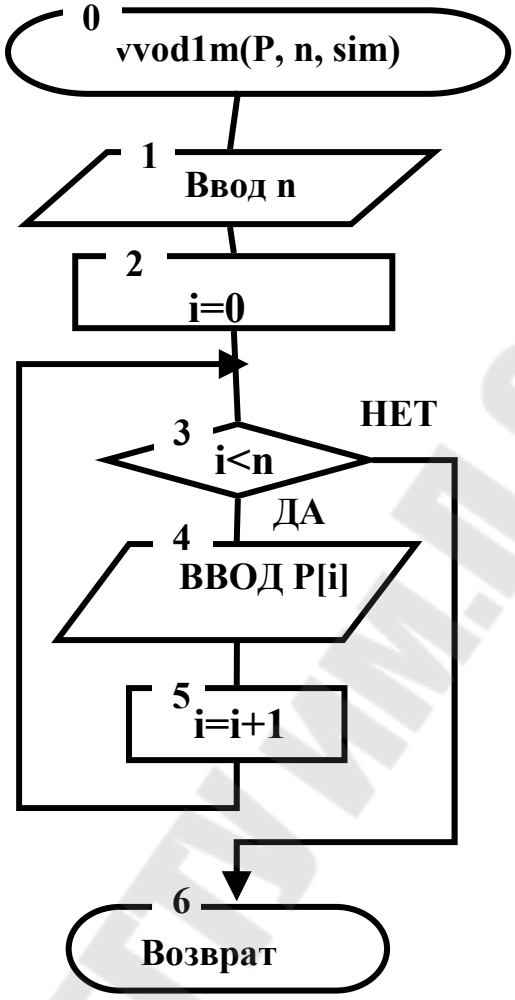


Рис. 2.2. Графическая схема алгоритма ввода массива

```
/* Описание функции vvod1m  */

void vvod1m(int *p, int *n, char sim)
{
  int i;
  printf("\nВведите размерность массива %c:", sim);
  scanf("%d", n);
  for(i=0; i<*n; i++)
   {
    printf("%c[%d]=", sim, i);
    scanf("%d", p+i);
   }
}
```

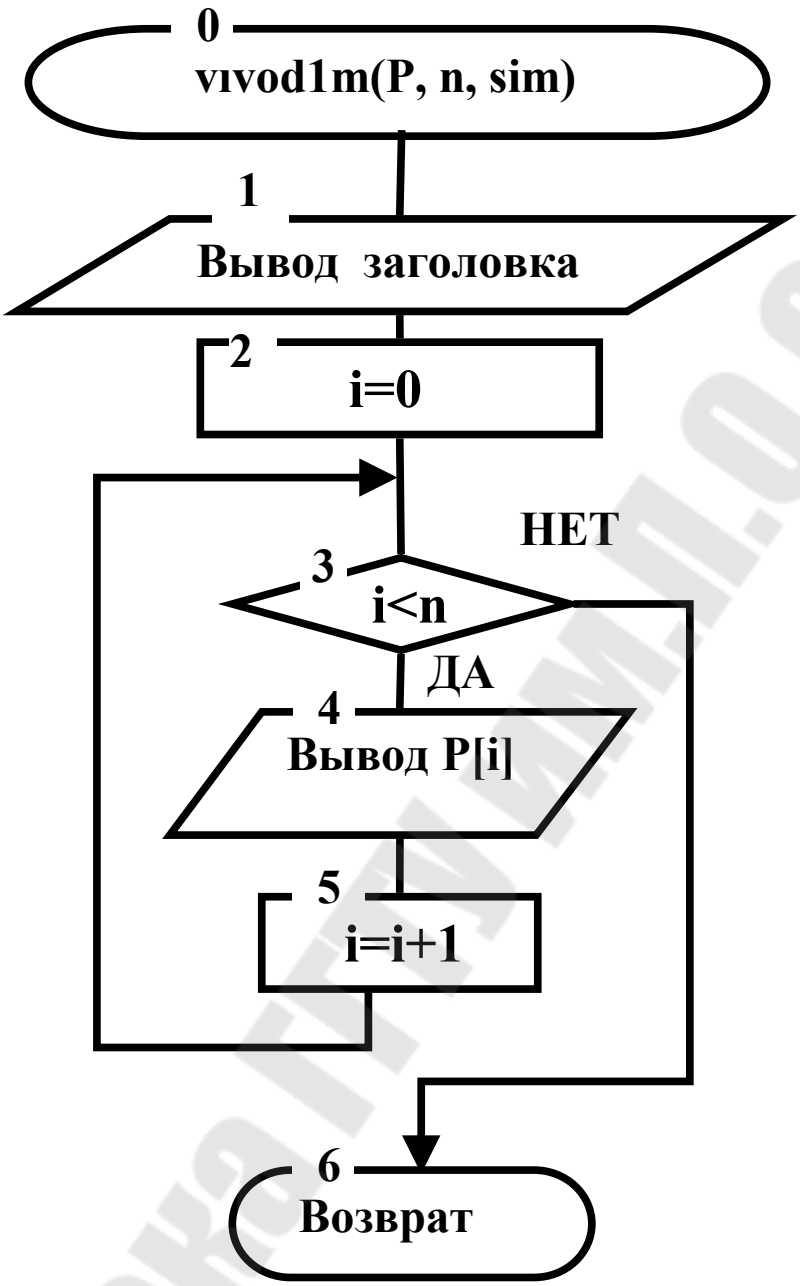


Рис. 2.3. Графическая схема алгоритма вывода массива

```
/* Описание функции vivod1m  */
void vivod1m(int *p,int n,char sim)
{
  int i;
  printf("Массив %c\n",sim);
  for(i=0;i<n;i++)
    printf("%c[%d]=%d\n", sim, i, *(p+i));
}
```

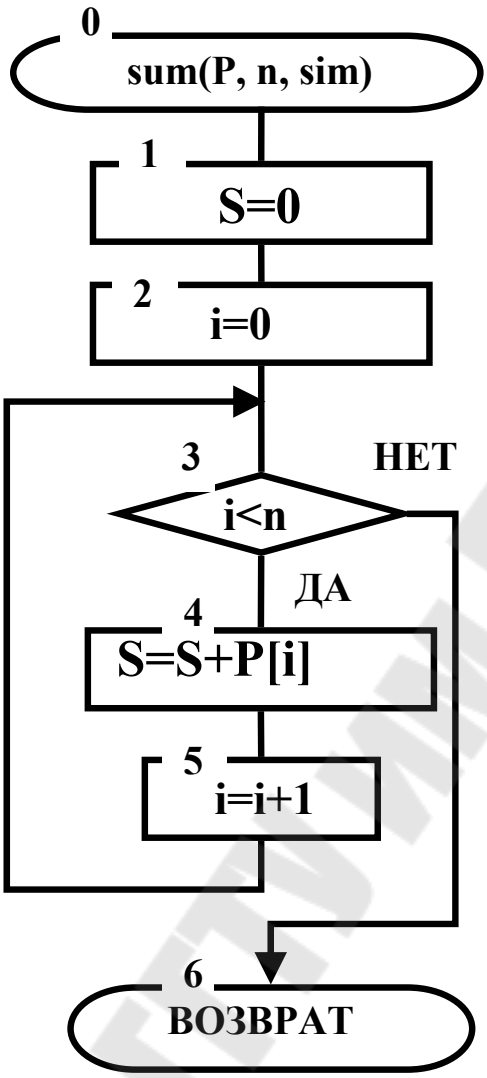


Рис. 2.4. Графическая схема алгоритма вычисления суммы элементов массива

```
/* Описание функции sum  */

int sum(int *p,int n)
{
  int i,s=0;
  for(i=0;i<n;i++)
    s += *(p+i);
  return(s);
}

/* Программирование с использованием подпрограмм
   Передача одномерных массивов в функцию. Пример 5     */
```

```
#include <stdio.h>
            //Объявление функций
void vvod1m (int *p, int *n, char sim);     // ввода массива
void vivod1m(int *p, int n, char sim);      // вывода массива
int sum(int *p, int n);                     // вычисления суммы элементов
массива

// Определяется, сумма элементов какого из двух массивов больше

void main(void)
{
 int a[50];
 int b[50];
 int n,m,s1,s2;
 vvod1m(a, &n, 'a');      // Обращения к функции vvod1m, не воз-
вращающей значения
 vvod1m(b, &m, 'b');    // по оператору return
 s1=sum(a, n);
 s2=sum(b, m);

 printf("\nСумма элементов массива a равна %d\n",s1);
 printf("\nСумма элементов массива b равна %d\n",s2);

 vivod1m(a, n, 'a');       // Обращения к функции vivod1m, не воз-
вращающей значения
 vivod1m(b, m, 'b');     // по оператору return
 if(s1>s2)          puts("Сумма элементов 1-го массива больше");
   else if(s1<s2)  puts("Сумма элементов 2-го массива больше");
  else      puts("Суммы равны ");
 fflush(stdin); getchar();
}

/* Описание функции vvod1m  */

void vvod1m(int *p, int *n, char sim)
{
   int i;
   printf("\nВведите размерность массива %c:", sim);
   scanf("%d", n);
```

```
   for(i=0; i<*n; i++)
    {
     printf("%c[%d]=", sim, i);
     scanf("%d", p+i);
    }
}

/* Описание функции vivod1m  */

void vivod1m(int *p,int n,char sim)
{
 int i;
 printf("Массив %c\n",sim);
 for(i=0;i<n;i++)
   printf("%c[%d]=%d\n", sim, i, *(p+i));
}

/* Описание функции sum  */

int sum(int *p,int n)
{
  int i,s=0;
  for(i=0;i<n;i++)
    s += *(p+i);
  return(s);
}
```

Пример. Передача двумерных массивов в функцию.

Найти количества положительных элементов среди элементов, находящихся в строках и столбцах с четными номерами, матриц X и Y .

Решение. Пусть имеем массивы X[mx][nx] и Y[my][ny] . Необходимо:

- Ввести массив X из mx строк и nx столбцов.
- Ввести массив Y из my строк ny столбцов.
- Вывести матрицу X.
- Вывести матрицу Y.
- Найти количество (kx) положительных элементов среди элементов, матрицы X, находящихся в строках и столбцах с четными номерами.
- Найти количество (ky) положительных элементов среди элементов матрицы Y, находящихся в строках и столбцах с четными номерами.
- Вывести kx, ky.

Из плана решения задачи следует необходимость решения следующих подзадач:

1. Ввод матрицы.
2. Вывод матрицы.
3. Нахождение количества положительных элементов среди элементов матрицы, находящихся в строках и столбцах с четными номерами ().

Назовем подзадачи 1, 2, 3 vvod2m, vivod2m, kolvo соответственно.

Решение подзадачи 1 – ввода матрицы и реализация его в виде алгоритма, возвращающего матрицу (P) по оператору return и число ее строк и столбцов (m и n) через заголовок функции.

Графическая схема алгоритма ввода матрицы (P) изображена на рис. 2.5.

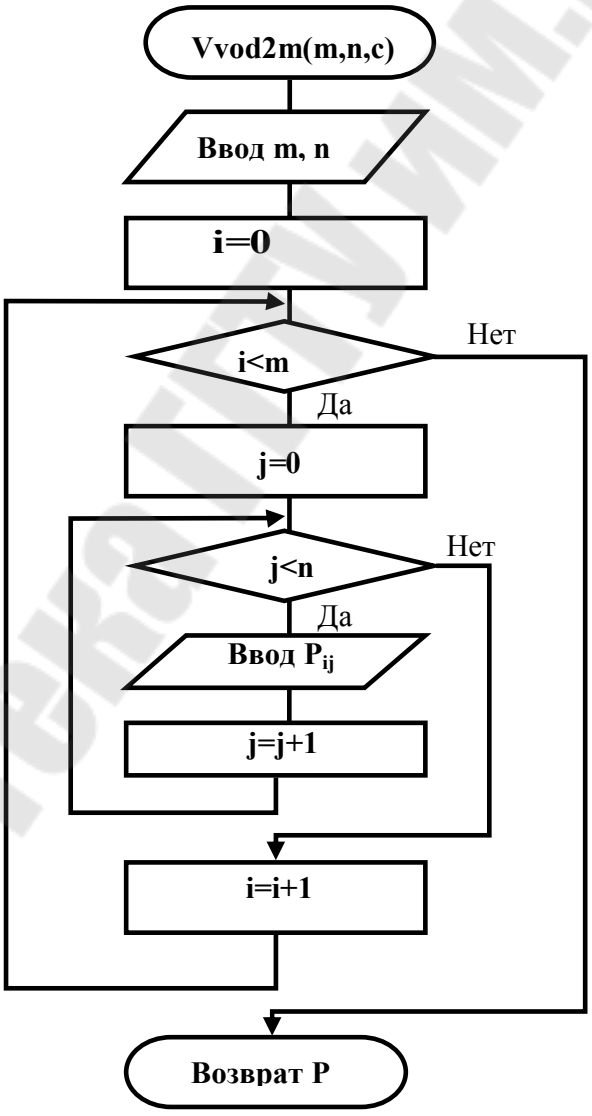


Рис. 2.5. Графическая схема ввода матрицы

```
      /*Описание функции Vvod2m */

int *vvod2m(int *m, int *n, char c)
 {
    int  mm, i, j, *p;
    puts("Введите число строк и столбцов матрицы (m и n)");
    scanf("%d %d", m, n);
    mm=*m**n;  //Количество эл. в матрице
    p=new int [mm];
    for(i=0; i<*m; i++)
      for(j=0; j<*n; j++)
        {
         printf("%c[%d][%d]=", c, i+1, j+1);
         scanf("%d", p+i**n+j);     //&p[i**n+j]
        }
    return(p);
}
```

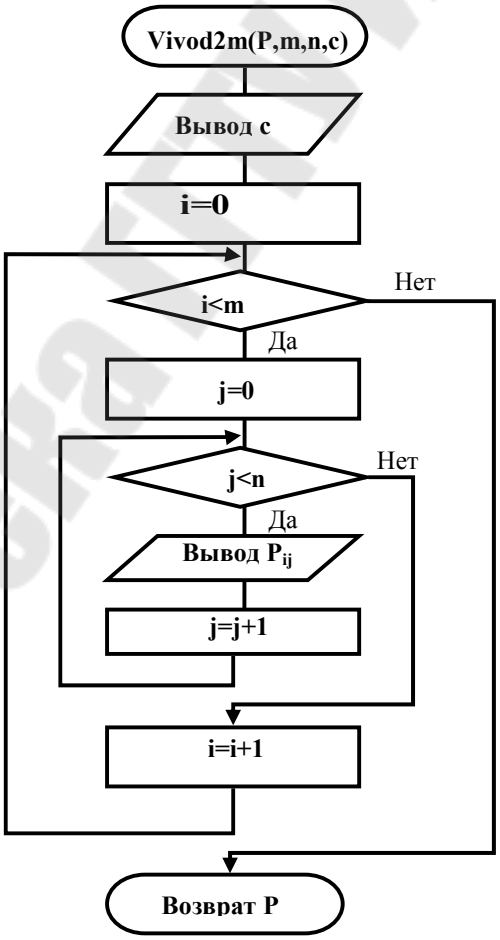


Рис. 2.6. Графическая схема алгоритма вывода матрицы

```
/* Описание функции Vivod2m */
void vivod2m(int *p, int m, int n, char c)
{
 int i,j;
 printf("Матрица %c\n", c); //Вывод заголовка матрицы
 for(i=0; i<m; i++)
   {
     for(j=0; j<n; j++)
        printf("%6d ",*(p+i*n+j));  // p[i*n+j]
     printf("\n");
   }
 }
```

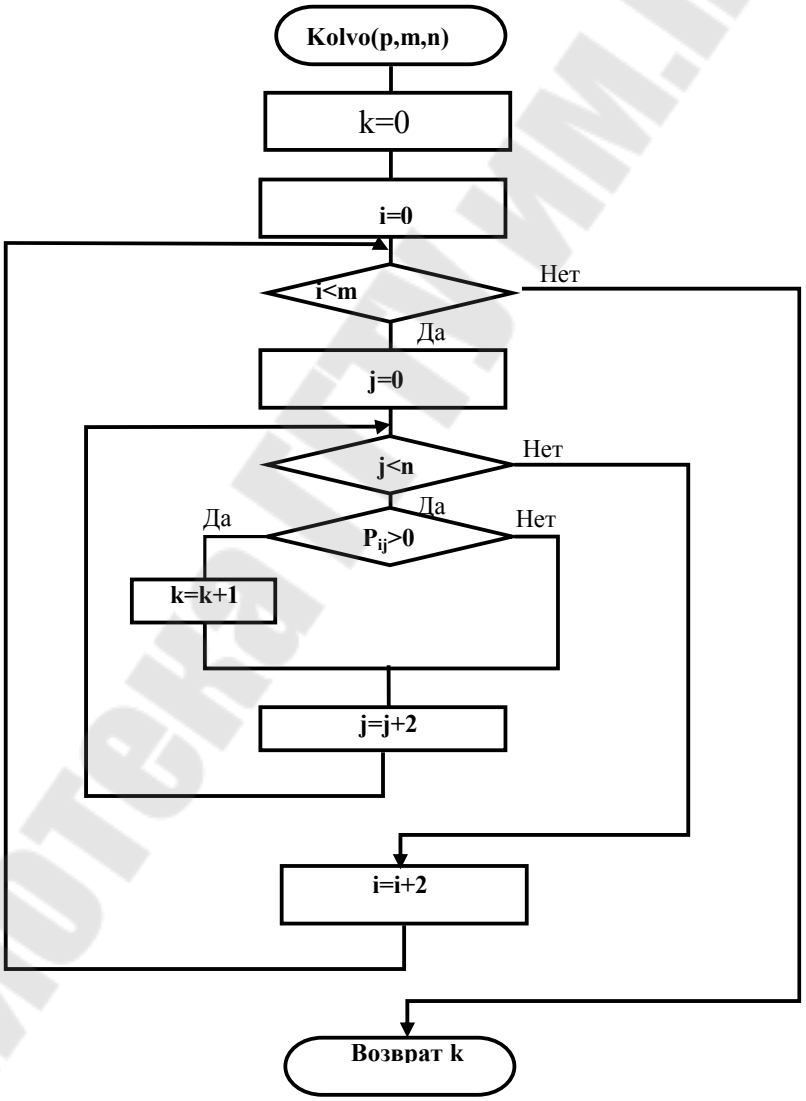


Рис. 2.7. Графическая схема нахождения количества положительных элементов матрицы среди элементов, находящихся в строках и столбцах с четными номерами

```
/*Описание функции  kolvo */
int  kolvo (int *p, int m, int n)
{
  int k=0, i, j;
  for(i=0; i<m; i+=2)
     for(j=0; j<n; j+=2)
          if(*(p+i*n+j)>0) k++;    //p[i*n+j]
  return(k);
}
```

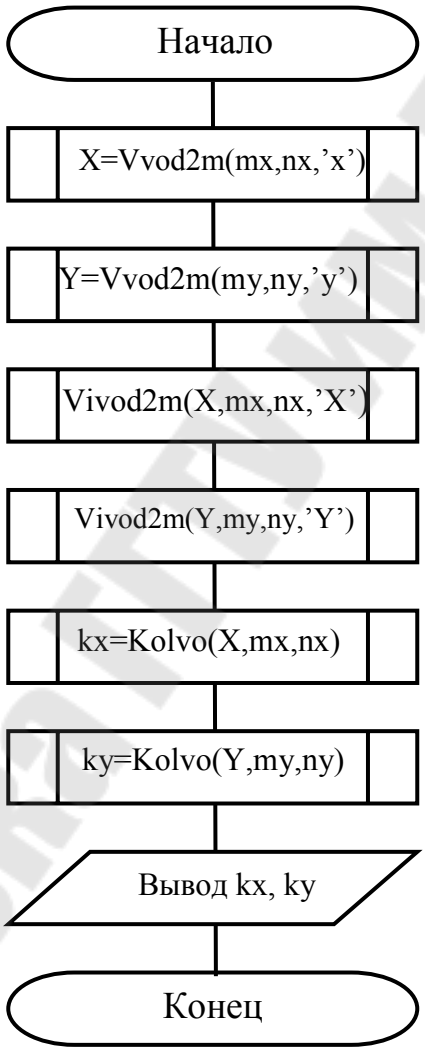


Рис. 2.8. Графическая схема основного алгоритма

/* Пример 6. Передача двумерных массивов в функцию (ч/з одномерный) */

```
#include <stdio.h>
                                  //Объявление функций
int *vvod2m ( int *m, int *n, char c);
```

```
void vivod2m(int *p, int m, int n, char c);
int kolvo(int *p, int m, int n);

void main(void)
{
  int *X, *Y;  //
  int mx, nx, my, ny;
  int  kx, ky;

  X=vvod2m(&mx, &nx, 'X');
  Y=vvod2m(&my, &ny, 'Y');
vivod2m(X, mx, nx, 'X');
vivod2m(Y, my, ny, 'Y');
kx=kolvo(X, mx, nx);
ky=kolvo(Y, my, ny);

printf("Искомое кол-во матр.X=%d\n", kx);
printf("Искомое кол-во матр.Y=%d\n", ky);

delete[] X;
delete[] Y;
}// Конец main()

 /* Описание функций */
 int *vvod2m(int *m, int *n, char c)
 {
    int mm,i,j,*p;
    puts("Введите m и n");  scanf("%d %d",m,n); mm=*m**n;
    p=new int [mm];
    for(i=0; i<*m; i++)
      for(j=0;j<*n;j++)
        {
          printf("%c[%d][%d]=",c,i+1,j+1);
          scanf("%d",&p[i**n+j]);
        }
    return(p);
}

void vivod2m(int *p,int m,int n,char c)
```

```
{
 int i,j;
 printf("%c\n",c);
 for(i=0; i<m; i++)
   {
     for(j=0;j<n;j++)
        printf("%6d ",p[i*n+j]);
     printf("\n");
 }
}

int kolvo(int *p, int m, int n)
{ int k=0, i, j;
 for(i=0; i<m; i+=2)
    for(j=0;j<n;j+=2)
        if(p[i*n+j]>0) k++;
 return(k);
}
```

3. Передача функции в качестве параметра в функцию

3.1. Указатель на функцию

Указатель на функцию – такой тип переменной, которой можно присваивать адрес точки входа в функцию, т. е. адрес первой исполняемой команды. Эта переменная может в дальнейшем использоваться для вызова функции вместо ее имени.

Общий вид определения указателя на функцию:

***Тип_результата* (*имя_указателя_на_функцию) (*список типов параметров*);**

Примеры:

***Void* (**pf*) (*int*);**

Здесь ***pf*** – указатель на функцию, невозвращающую результат и принимающую параметр типа ***int***.

***int* (**f*) (*float, int* *);**

Здесь *f* – указатель на функцию, возвращающую результат типа ***int*** и имеющую два параметра – первый типа *float*, второй параметр – указатель на число типа ***int***.

При определении указателя на функцию разрешается одновременно с указанием типа параметров задавать их имена. Например:

char* (*comp) (char *s1, char *s2);

Здесь ***comp*** – указатель на функцию, возвращающую результат – указатель на ***char*** и принимающую в качестве параметров два указателя на ***char***.

Указатели на функцию передаются в качестве аргумента другим функциям, если последние реализуют какой-либо метод обработки функций, не зависящий от вида конкретной обрабатываемой функции. Эта конкретная (фактическая) функция вызывается из тела обрабатываемой функции по переданному ей указателю.

Чтобы сделать программу легко читаемой, при описании указателей на функцию используют переименование типов (***typedef***). Например:

typedef float (*func) (float);

Здесь описан тип ***func*** как указатель на функцию с одним параметром типа ***float*** и возвращающую результат типа ***float***.

Указатели на функцию используются часто. Например, многие библиотечные функции в качестве аргументов получают указатель на функцию. Использование указателей на функцию в качестве аргументов позволяет разрабатывать универсальные функции, например численного решения уравнений, численного интегрирования и дифференцирования. Массивы указателей на функции используются для организации меню.

3.2. Пример передачи имени функции в функцию

Вычислить $f((a+b)/2)\cdot(b-a)$ для трех функций $f(x)=3x$, $f(x)=e^x sinx$, $f(x)=xcosx$ и чисел a и b.

Решение.

Ниже на рис. 3.1 дана графическая интерпретация выражения $f((a+b)/2)\cdot(b-a)$.

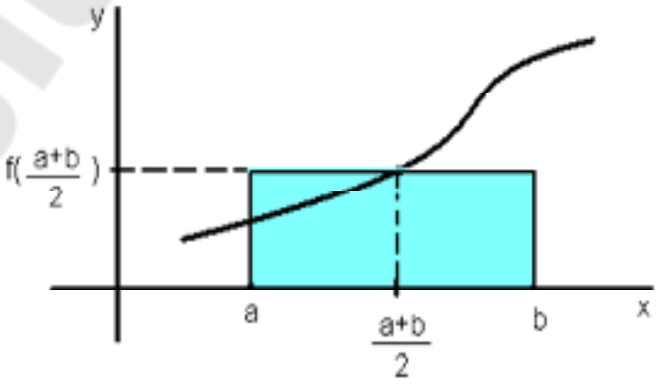


Рис. 3.1. Графическая интерпретация

Для решения поставленной задачи необходимо разработать четыре функции. Одну из них назовем ***f*1**. Она предназначается для вычисления значения **3*x*** . Вторую назовем ***f*2**. Она предназначается для вычисления значения $e^x sinx$. Третью функцию назовем ***f*3**. Она предназначается для вычисления значения ***xcosx***. Четвертую функцию назовем ***IT***. Она предназначается для вычисления значения $f((a+b)/2)\cdot(b-a)$ для заданной функции ***f***.

Функции ***f*1**, ***f*2**, ***f*3** будут передаваться в функцию ***f*** в качестве параметра. Поэтому создадим новый тип – указатель на функцию, принимающую в качестве параметра вещественное значение (***float***) и возвращающую указатель на вещественный тип (float). Пусть ***func*** – имя нового типа:

***typedef float* (*func)(*float*)**;

Функции ***f*1**, ***f*2**, ***f*3** должны возвращать значения типа ***float***. Описание их будет выглядеть следующим образом:

```
/* Описание f1 */
float f1(float x)
{ float ff=3*x;
  return (ff);
}
/* Описание f2*/
float f2(float x)
{ float ff=exp(x)*sin(x);
  return (ff);
}
/* Описание f3 */
float f3(float x)
{ float ff=x*cos(x);
  return (ff);
}
```

Функция ***IT*** будет иметь три формальных параметра. Ее описание будет иметь вид:

```
float IT(func f, float a, float b)
{ float t=(b-a)*f((b-a)/2);
  return(t);
 }
```

Программа, использующая четыре разработанных функции, может иметь вид:

```
#include<stdio.h>
#include<math.h>
#include<conio.h>
typedef float (*func)(float);
float f1 (float x);
float f2 (float x);
float f3 (float x);
float IT(func f, float a, float b);
void main()
{
   float a, b;
   puts("Введите числа a, b");
   scanf("%f%f",  &a, &b);
   printf("Для a=%f b=%f \n", a, b);
   printf("Значение f(f1, a, b ) равно %f\n",IT(*f1,a,b));
   printf("Значение f(f2, a, b ) равно %f\n",IT(*f2,a,b));
   printf("Значение f(f3, a, b ) равно %f\n",IT(*f3,a,b));
   getch();
}
//Описание f1
float f1(float x)
{ float ff=3*x;
  return (ff);
}
//Описание f2
float f2(float x)
{ float ff=exp(x)*sin(x);
  return (ff);
}
//Описание f3
float f3(float x)
{ float ff=x*cos(x);
  return (ff);
}
//Описание IT
float IT(func f, float a, float b)
{ float t=(b-a)*f((b-a)/2);
  return(t);
}
```

В рассмотренном примере функция main() является вызывающей для функции ***IT***. При первом обращении к функции ***IT*** ей будет передана функция ***f*1** и значения *a* и *b* (***IT*(*f*1, *a*, *b*)**). В результате будет вычислено значение *3x* при *x=(b–a)/2*. При втором обращении к функции IT ей будет передана функция *f*2 и значения a и b (***IT*(*f*2, *a*, *b*)**). В результате будет вычислено значение $e^x sinx$ при *x*=(*b–a*)*/2*. При третьем обращении к функции *IT* ей будет передана функция *f*3 и значения *a* и *b* (***IT*(*f*3, *a*, *b*)**). В результате будет вычислено значение *xcosx* при *x*=(*b–a*)*/2*. Легко подсчитать (например в MathCad), что при a=1, в=2 должны быть получены значения 1.5, 0.790439 и 0.43879. Такие же значения получаются и при запуске программы, если ввести *a*=1, *b*=2.

Заметим, что если не создавать новый тип *func*, то прототип функции *IT* будет иметь вид:

float IT(float (*f)(float), float a, float b);

4. Рекурсии

4.1. Основные понятия

Рекурсия – это такой способ организации вычислительного процесса, при котором подпрограмма в ходе выполнения составляющих ее операторов обращается сама к себе, прямо или косвенно.

Рекурсивная форма организации алгоритма обычно дает более компактный текст программы, чем другие способы решения этой же задачи, но требует дополнительных затрат оперативной памяти (ОП) и времени для организации рекурсивных вызовов функций. Рекурсивное построение алгоритма является наиболее естественным и экономичным путем решения таких задач, как вычисление факториала, определение сумм и произведений членов ряда, вычисляемых до получения заданной точности, формирование и использование связанных структур данных, например очередей, списков, деревьев.

В рекурсивной функции должны выполняться следующие правила:

– при каждом вызове функции в нее должны передаваться модифицированные данные;

– на каком-то этапе должен быть прекращен дальнейший вызов функции. Рекурсивный процесс должен шаг за шагом так упрощать задачу, чтобы в конце концов для нее появилось нерекурсивное решение (тривиальное решение);

– после каждого вызова рекурсивной функции в точку возврата должен передаваться некоторый результат для дальнейшего использования.

При правильно организованной рекурсивной функции осуществляется ее многократный последовательный вызов. При этом система:

– сохраняет в стеке значения всех локальных переменных функции и ее параметров для всех предыдущих вызовов;

– выделяет ОП для всех предыдущих вызовов;

– для внутренних переменных выделенная ОП сохраняется в течение всего времени выполнения программы.

Компилятор не ограничивает количество рекурсивных вызовов одной функции. Но операционная среда может налагать практические ограничения, так как слишком большое количество рекурсивных вызовов может вызвать переполнение стека.

4.2. Примеры рекурсивных функций

Пример рекурсивной функции, k-раз выводящую одну и ту же фразу

Функция ***strk*** имеет два параметра: *s* – выводимая строка; *k* – количество повторений строки. Функция strk выполняется следующим образом. При *k*>0 функция выводит строку *s* и обращается к функции *strk* с параметрами *s* и *k*-1. При следующем обращении передается *s* и значение *k* еще на 1 меньшее и т. д. до *k*=0. При вызове функции с *k*=0 вывод *s* не производится и последовательно осуществляется возврат обратно во все вызванные функции *strk* и выход в главную функцию.

Ниже приведен текст программы с вызовом функции *strk* из главной функции *main*:

```
#include<stdio.h>
 #include<math.h>
 #include<conio.h>
 void strk(char *s,int k);
 main()
 {
  int K;
  char *str;
  clrscr();
  puts ("Введите строку ");
  gets(str);
  puts("Введите число повторений строки");
  scanf("%d",&K);
  strk(str,K);
  getch();
```

```
 return(0);
}
/* Функция многократного вывода строки */
 void strk(char *s, int k)
  {
   if(k>0)    //анализ на тривиальность решения
     { puts(s);
       strk(s,k-1); // вызов функции strk
     }
  }
```

П*ример рекурсивной функции* (*Fact*) *вычисления факториала числа N* (*N!=1·2·3·…·N*)

Функция ***Fact*** имеет один параметр – значение *N*. При выполнении программы функция main вызывает функцию *Fac*t и передает функции *Fact* параметр *N*. Если *N*=0, то возвращается значение 1. Иначе *Fact*(*N*) вызывает *Fact*(*N*–1) и т. д. до вызова *Fact*(0). *Fact*(0) возвратит значение 1 в точку вызова *Fact*(1) и последовательно осуществляется возврат обратно во все вызванные функции *Fact* и выход в главную функцию.

Ниже приведен текст программы с вызовом функции *Fact* из главной функции *main*:

```
#include<stdio.h>
#include<math.h>
#include<conio.h>
long Fact(int N);
main()
{
 int N;
 long NF;
 puts ("Введите целое число ");
 scanf("%d",&N);
 NF=Fact(N);
 printf("Для N=%d N!=%ld\n",N,NF);
 getch();
 return(0);
}
/*Рекурсивная функция вычисления  N! */
 long Fact(int N)
```

```
{
 if(N)  // анализ на тривиальность решения
       return( N*Fact(N-1));
 else return(1);
}
```

Пример рекурсивной функции (sum) вычисления суммы элементов одномерного массива

Функция ***sum*** имеет два параметра: массив *x* и число его элементов *n*. При выполнении программы функция *main* вызывает функцию *sum* и передает функции *sum* фактический массив *a* и число его элементов *n*. Если *n*=1, то возвращается значение *x*[0]. Иначе *sum*(*x*,*n*) вызывает *sum*(*x*,*n*–1) и т. д. из оператора *s*=*x*[*n*-1]+*sum*(*x*,*n*-1);. При *n*=1 решение тривиально. Возвращаемый результат равен *x*[0]. Начинается обратный ход рекурсивных вычислений: возврат из вызванных функций результатов вычислений.

Ниже приведен текст программы с вызовом функции *sum* из главной функции *main*.

```
#include<stdio.h>
#include<math.h>
#include<conio.h>
long sum(int a[],int n);
main()
{
 int i,n,a[10];
 clrscr();
 puts("Введите число элементов массива");
 scanf("%d",&n);
 puts("Вдедите массив целых чисел");
 for(i=0;i<n;i++)
  {
   printf("Введите a[%d]=",i);
   scanf("%d",&a[i]);
  }
 for(i=0;i<n;i++)
   printf("%3d ",a[i]);
 printf("\n");
 printf("Сумма %d элем. массива a равна %ld \n",n,sum(a,n));
 getch();
```

```
 return(0);
}
/* Функция вычисления суммы n элементов массива */
  long sum(int x[],int n)
  { long s;
    if(n==1)    //анализ на тривиальность решения
      s=x[0];
    else s=x[n-1]+sum(x,n-1);
    return(s);
   }
```

Литература

1. Климова, Л. Н. Практическое программирование. Решение типовых задач / Л. Н. Климова. – Москва : КУДИЦ_ОБРАЗ, 2001. – 592 с.

2. Павловская, Т. А. МС/С++. Программирование на языке высокого уровня / Т. А. Павловская. – Санкт-Петербург : Питер, 2002. – 464 с.

3. Демидович, Е. М. Основы алгоритмизации и программирования. Язык СИ : учеб. пособие / Е. М. Демидович. – Санкт-Петербург, 2006. – 440 с.

Приложение

Примеры программ

Пример 1. Составить программу для вычисления интеграла функции $\frac{x^2+1.5}{\tan(x)}$, методом прямоугольников с заданной точностью на заданном интервале.

Решение

```
#include<math.h>
#include<stdio.h>
#include<conio.h>
typedef float(*func)(float);
float f(float x);
float integral(float a,float b,int n,func f);
void main()
{float E,S1,S2,a,b;
int n;
puts("Vvedite intervali");
scanf("%f %f",&a,&b);
puts("Vvedite tochnost");
scanf("%f",&E);
n=2;
S1=integral(a,b,n,*f);
do
{
 n=n*2;
 S2=S1;
 S1=integral(a,b,n,*f);
}
while(fabs(S1-S2)>E);
printf("a=%6.3f b=%6.3f E=%E\n",a,b,E);
printf("integral:%8.6f\n",S1);
}

float f (float x)
{
  return((x*x+1.5)/tan(x));
}
```

```
float integral(float a,float b,int n ,func f)
{
 float S,h;
 int i;
 h=(b-a)/n;
 S=(f(a)+f(b))/2;
 for(i=1;i<n;i++)
  S=S+f(h*i+a);
 return(S*h);
}
```

Пример 2. Составить программу для вычисления интеграла функции $\sqrt{x} \cdot \cos(x^2)$, методом Симпсона с заданной точностью на заданном интервале.

Решение

```
#include<math.h>
#include<stdio.h>
#include<conio.h>
typedef float (*func)(float);
float f(float x);
float integral(float a,float b,int n,func f);
void main()
{
 float e,S1,S2,a,b;
 int n;
 puts("Vvedite intervali");
 scanf("%f %f",&a,&b);
 puts("Vvedite tochnost");
 scanf("%f",&e);
 n=8;
 S1=integral(a,b,n,*f);
 do
 {
  n=n*2;
  S2=S1;
  S1=integral(a,b,n,*f);
 }
```

```
 while(fabs(S1-S2)>e)
 printf("a=%6.3f b=%6.3f e=%e\n",a,b,e);
 printf("integral:%8.6f\n",S1);
 }
float f (float x)
{
        return(sqrt(x)*cos(pow(x,2)));
 }
}
float integral(float a,float b,int n ,func f)
{
 float S,h;
 int i;
 h=(b-a)/n;
 S=f(a)+f(b);
 for(i=1;i<n;i++)
 {
  if(i%2==0)
  {
   S=(S+2*f(h*i+a));
  }
  else S=(S+4*f(h*i+a))
 }
 S=S*h/3;
 return(S);
}
```

Пример 3. Составить программу для вычисления интеграла функции $\sqrt{x}\cdot\cos(x^2)$, методом Ньютона с заданной точностью на заданном интервале.

Решение

```
#include <stdio.h>;
#include <math.h>;

float f(float);
float integral(float (*)(float),float,float,float);
```

```
int main()
{
  float s;
  s=integral(f,1,2,0.000001);
  printf("s=%.10f\n",s);
  return(0);
}

float f(float x)
{
  return(sin(x)-7*x+9);
}


float integral(float (*y)(float),float a,float b,float e)
{ long int i,m=3;
  float s,s2,dx;
  dx=(b-a)/m;
  s=y(a)+y(b);
  for(i=1;i<m;i++)
  {
   if (i%3==0) s=s+2*y(a+dx*i);
    else s=s+3*y(a+dx*i);
  }
  s=s*dx*3/8;

  s2=y(a)+y(b);
  do {
   m=2*m;
   dx=(b-a)/m;
  for(i=1;i<m;i++)
  {
   if (i%3==0) s2=s2+2*y(a+dx*i);
    else s2=s2+3*y(a+dx*i);
  }
   s2=s2*dx*3/8;
   if(fabs(s-s2)<=e) break;
   else { s=s2;}
```

```
   printf("n=%ld %.10f\n",m,s2);
  }
  while(m<1000000);
  return(s2);
}
```

Пример 4. Составить программу для транспонирования матриц A и B.

Решение

```
#include <stdio.h>
#include <conio.h>
#include <math.h>.h>
 void vvod(float *p,int *n,int *m,char sim);
 void vivod(float *p,int n,int m,char sim);
 void vivod2(float *p,int m,int n,char sim);
 void main(void)
{
 float A[100];float  a[100];float B[100];float  b[100];
 int n1,m1,n2,m2,n3,m3,n4,m4; float d;
 clrscr();
  vvod(A,&n1,&m1,'A');
  vvod(B,&n2,&m2,'B');
  vivod(A,n1,m1,'A');
  vivod(B,n2,m2,'B');
  vivod2(A,n1,m1,'a');
  vivod2(B,n2,m2,'b');
fflush(stdin);
getchar();
}
  void vvod (float *p,int *n,int *m,char sim)
  { int i,j;
   printf("Введите n и m %c\n",sim);
   scanf("%d%d",n,m);
    for(i=0;i<*n;i++)
     for(j=0;j<*m;j++)
     {printf("%c[%d][%d]=",sim,i,j);
      scanf("%f",p+i**n+j);
     }
   }
```

```
 void vivod(float *p,int n,int m,char sim)
 { int i,j;
  printf("Матрица %c\n",sim);
   for(i=0;i<n;i++)
    {for(j=0;j<m;j++)
     printf("%5.1f",p[i*n+j]);
     printf("\n");}
 }
 void vivod2(float *p,int n,int m,char sim)
{ int i,j;
 printf("Матрица %c транспонированная\n",sim);
  for(i=0;i<m;i++){
   for(j=0;j<n;j++)
    printf("%5.1f",p[j*n+i]);
    printf("\n");}
 }
```

Пример 5. Составить программу для вычисления значения функции $L = -0.31 \cdot a + e^{-b} + c$, где a, b, c минимальные элементы массивов A, B, C соответственно.

Решение

```
#include<stdio.h>
#include<math.h>
void vvod(int *w,int *n,char s);
void vivod(int *w,int n,char s);
float min(int *w,int n);


void main()
{FILE *fp;
 int X[10],T[10],Z[10],a,b,c,nX,nT,nZ,key;
 float L;
  puts("Нажмите 0 для вывода на принтер или другую для - экрана");
     scanf("%d",&key);
     if (key==0) fp=stdprn;
     else fp=stdout;
 vvod (X,&nX,'X');
 vvod (T,&nT,'T');
```

```
vvod (Z,&nZ,'Z');
vivod(X,nX,'X');
vivod(T,nT,'T');
vivod(Z,nZ,'Z');
a=min(X,nX);
b=min(T,nT);
c=min(Z,nZ);
puts("Минимальные значения массивов X, T, Z");
fprintf(fp,"a=%d b=%d c=%d\n",a,b,c);
L=-0.31*a+exp(-b)+c;
fprintf(fp,"L=%.4f\n",L);
fflush(stdin);
getchar();
}



void vvod(int *w,int *n,char s)
{
 int i;
 printf("Введите кол-во элементов массива %c\n",s);
 scanf("%d",n);
 printf("Введите массив %c\n",s);
 for (i=0;i<*n;i++)
  scanf("%d",w+i);
}



void vivod (int *w,int n,char s)
{
 int i;
 printf("Массив %c\n",s);
 for (i=0;i<n;i++)
  printf("%d  ",*(w+i));
 printf("\n");
}
float min(int *w,int n)
{
```

```
int i,m;
m=*(w+0);
for(i=0;i<n;i++)
 if(m<*(w+i)) m=*(w+i);
return (m);
}
```

Содержание

Учебное электронное издание комбинированного распространения

Учебное издание

Кравченко Ольга Алексеевна
Литвинов Дмитрий Александрович

ПОДПРОГРАММЫ И ПРОГРАММИРОВАНИЕ С ИХ ИСПОЛЬЗОВАНИЕМ НА ЯЗЫКЕ С

Пособие
по курсу «Основы алгоритмизации и программирования» для студентов специальностей 1-36 04 02 «Промышленная электроника» и 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной и заочной форм обучения

Электронный аналог печатного издания

Редактор *С. Н. Санько*
Компьютерная верстка *М. В. Аникеенко*

Подписано в печать 26.02.09.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 2,79. Уч.-изд. л. 2,39.
Изд. № 119.
E-mail: ic@gstu.gomel.by
http://www.gstu.gomel.by

Издатель и полиграфическое исполнение:
Издательский центр учреждения образования
«Гомельский государственный технический университет
имени П. О. Сухого».
ЛИ № 02330/0131916 от 30.04.2004 г.
246746, г. Гомель, пр. Октября, 48.