Лениногорский филиал федерального государственного бюджетного образовательного учреждения высшего образования «Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ»

Кафедра Естественнонаучных и гуманитарных дисциплин

Методические указания к лабораторным работам

по дисциплине

Программирование на языках высокого уровня

направления подготовки:

09.03.02 Информационные системы и технологии

Лабораторная работа № 1

Программирование простейших циклов на языке Си. Работа в системе Turbo C (версия 2.0)

Структура программы

Любая программа на языке Си состоит из одной или более "функций", являющихся основными модулями программы. Функция, с которой начинается выполнение программы, всегда носит имя ***main*.** Простая программа, состоящая только из функции main, имеет следующую структуру:

Директивы препроцессора
main ()
{ Описания переменных
Операторы
}

Заголовок функции – main(). Круглые скобки после имени main как раз и указывают, что это функция. Тело функции заключается в фигурные скобки и состоит из описаний переменных и операторов, описывающих процесс обработки данных.

В программу можно включать комментарии, начинающиеся с пары символов **/*** и заканчивающиеся парой ***/** (они могут быть везде, где могут быть пробелы).

Пример простой программы:

```
/* программа сложения двух целых чисел */
#include <stdio.h>
main ()
        { int a, b; /* описание целочисленных переменных a и b */
          printf ("Задайте два числа: "); /* вывод сообщения */
          scanf ("%d %d", &a, &b); /* ввод значений a и b */
          printf ("%d + %d = %d\n", a, b, a+b); /* вывод результата */
        }
```

При выполнении этой программы на экране появится сообщение:

Задайте два числа:

и затем программа будет ждать, пока вы не введете числа (ввести нужно в той же строке, разделяя числа пробелом). Например:

Задайте два числа: 328 54

Затем появится результат в виде:

328 + 54 = 382

В этой программе директива препроцессора ***#include <stdio.h>*** служит для включения в программу библиотечного файла stdio.h, содержащего об'явления стандартных функций ввода/вывода, таких как ***printf, scanf***. Тело функции main содержит три оператора вызова функций printf и scanf.

Обратите внимание, что все ключевые слова в языке Си пишутся строчными буквами, директивы препроцессора начинаются с первой позиции строки, а операторы можно размещать с любой позиции. Для наглядности принята ступенчатая форма

записи программы.

Описания переменных и основные типы данных

При описании переменных указываются имена переменных и типы значений этих переменных:

тип_1 имя_1;
тип_2 имя_2;

Имя (идентификатор) - это последовательность латинских букв и цифр, начинающаяся с буквы. Если несколько переменных имеют один и тот же тип, то их можно описать вместе, перечислив имена через запятую:

тип имя_1, имя_2, ... ;

К основным типам данных относятся целые числа (***int, short, long, unsigned***), символы (***char***) и вещественные числа или числа с плавающей точкой (***float, double***).

Примеры описаний переменных:

```
float x,y,z; /* вещественные числа */
double x1,x2; /* вещ. числа двойной точности */
char simv; /* символ */
int i,j; /* целые числа */
long summa; /* длинное целое */
short k1,k2; /* короткие целые */
unsigned count; /* беззнаковое целое (неотрицательное) число */
```

Объем памяти, занимаемой данными различных типов, зависит от типа ЭВМ и конкретной реализации языка Си.

При описании переменной можно инициализировать переменную, например:

```
int k=0; /* k присваивается начальное значение 0 */
```

Типы используемых в программе констант определяются по их виду, например:

123 -65 - целые константы;
-34.6 3.14159 .12E-5 7e4 - константы с плавающей точкой
(.12E-5=.0000012 7e4=70000.);
'A' 'a' '2' '%' - символьные константы.

Рассмотренные типы являются простыми. Более сложные структурированные типы данных, а также описание нестандартных типов данных будут рассмотрены позднее.

Определение символических констант

Часто возникает необходимость использовать в программе именованные константы. Использование символических имен вместо значений делает программу более понятной. Для определения символических констант служит директива препроцессора ***#define***. В начало программы до или после директив #include для каждой константы нужно добавить строку вида:

#define имя значение

Например:

```
#define PI 3.14159
#define RADIUS 16.75
```

Обратите внимание на прописные буквы в именах констант. По традиции символические константы пишутся прописными буквами в отличие от имен переменных. Конечно, вы можете написать константы и строчными буквами, но при этом вы должны чувствовать свою вину, поскольку нарушаете традицию.

Операторы

Оператор присваивания

Оператор служит для присвоения переменной значения и имеет формат:
переменная = выражение

При выполнении оператора вычисляется значение выражения и присваивается переменной.

Примеры:

```
x=0.1;
i=i+1;
y=(sin(x)-10)*x;
k=n % 3;
```

Выражения могут состоять из операндов - переменных, констант и вызовов функций, круглых скобок и знаков операций **+** (сложение), **-** (вычитание), ***** (умножение), **/** (деление), **%** (вычисление остатка от целочисленного деления), **++** (увеличение на 1), **--** (уменьшение на 1) и некоторых других.

Операции *,/,% имеют более высокий приоритет, чем + и -. Операции с одинаковым приоритетом выполняются слева направо, если нет скобок.

Операндами операции % должны быть значения целого типа , результат имеет тот же тип.

Некоторые арифметические стандартные функции:

abs(x), fabs(x) - вычисляется абсолютное значение x;
atan(x) - вычисляется арктангенс x;
tan(x) - вычисляется тангенс x; x задается в радианах;
acos(x) - вычисляется арккосинус x;
cos(x) - вычисляется косинус x; x задается в радианах;
asin(x) - вычисляется арксинус x;
sin(x) - вычисляется синус x; x задается в радианах;
exp(x) - e возводится в степень x;
log(x) – вычисляется натуральный логарифм x;
log10(x) – вычисляется десятичный логарифм x;
sqrt(x) – вычисляется $\sqrt{x}$ (корень квадратный из x).

Функция abs возвращает целое значение типа int, аргумент также должен быть целым. Остальные функции возвращают вещественное (double) значение при вещественном аргументе.

При использовании указанных функций в программу нужно включить директиву ***#include <math.h>*.**

Оператор-выражение

В языке Си любое выражение, заканчивающееся точкой с запятой (;), является оператором.

Примеры:

```
i++; /* увеличение значения i на 1, эквивалентно оператору i=i+1; */
i--; /* уменьшение i на 1 */
a+2; /* смысла не имеет, хотя синтаксически верно */
```

Примечание. Рассмотренный выше оператор присваивания является частным случаем оператора-выражения, поскольку в выражениях можно использовать операцию присваивания (=) наравне с другими.

Оператор вызова функции

Оператор вызова функции имеет вид:

имя_функции (аргумент1, ... , аргументN);

Он тоже является частным случаем оператора-выражения.

Примерами операторов вызова функции являются уже знакомые вам операторы вызова функций форматированного ввода/вывода printf и scanf (см. стр. 7). Рассмотрим эти функции детальнее.

Использование функции *printf*

Функция printf служит для вывода на экран монитора сообщений, данных, результатов вычислений. Число аргументов - один или более. Первый аргумент функции - это форматная строка, которая может содержать тексты, подлежащие выводу на экран, управляющие символы, форматы вывода значений переменных или выражений. Остальные аргументы - это переменные или выражения. Вернемся к примеру программы на стр.7.

В операторе

printf ("Задайте два числа: ");

аргумент только один - форматная строка, содержащая текст. В операторе

printf ("%d + %d = %d\n", a, b, a+b);

четыре аргумента. Первый аргумент - форматная строка (строка символов в кавычках) показывает, как должны быть напечатаны значения остальных аргументов (a,b,a+b). Каждому из аргументов a,b и a+b соответствует одна спецификация преобразования (формат) %d. Это спецификация вывода целого числа. Кроме форматов, форматная строка содержит последовательности символов " + ", " = ", которые нужно вывести, и управляющий символ '\n' (перевод строки), чтобы после вывода результата курсор переместился в начало следующей строки.

Функция printf выводит на экран то, что указано в форматной строке, подставляя вместо каждого формата значение очередного аргумента из списка. Число форматов должно быть равно числу аргументов после форматной строки. Не забудьте это основное требование!

Ниже приведены некоторые форматы:

%d - для вывода целого числа со знаком (типов int, short);
%ld - для вывода целого числа со знаком (типа long);
%u - для вывода целого числа без знака (типа unsigned);
%f - для вывода вещественного числа (типов float, double) в формате числа с фиксированной точкой (с точностью по умолчанию 6 цифр после точки);
%e - для вывода вещественного числа в экспоненциальном формате:
[-]d.dddddde{ }dd (здесь d - десятичная цифра);
%c - для вывода символа;
%s - для вывода строки символов.

Например, если число 123.68 вывести в формате %f, то будет напечатано 123.680000, если же указать формат %e, то результатом будет 1.236800e+02. В форматах %f и %e можно указать точность, например:
%.1f 123.7
%.4e 1.2368e+02

Использование функции *scanf*

Функция scanf предназначена для ввода значений переменных с клавиатуры во время выполнения программы.

Список аргументов этой функции почти такой же, как у функции printf. Первый аргумент - это форматная строка, содержащая форматы ввода значений переменных. Сами переменные, точнее указатели на переменные, записываются в списке аргументов после форматной строки.

Примеры:

```
int a,b;
float x,y;
double z;
scanf ("%d %d", &a, &b);
scanf ("%f %e %lf", &x, &y, &z);
```

При выполнении первого оператора scanf будут прочитаны вводимые пользователем с клавиатуры два целых числа и присвоены переменным *a* и *b*. При втором вызове функции scanf будут введены три вещественных числа и присвоены соответственно переменным x, y и z.

Числа во входном потоке могут разделяться либо пробелами, либо символами "новой строки", либо символами табуляции. Например, входной поток может выглядеть так:

```
-52 1374
0.5 -17.472
345678.7654
```

Тогда результат выполнения операторов scanf будет такой:
a=-52; b=1374; x=0.5; y=-17.472; z=345678.7654.

Функция scanf использует практически тот же набор форматов, что и функция

printf. Основные отличия в случае функции scanf следующие:

1. Формат %hd служит для ввода коротких целых чисел (типа short).

2. Форматы %f и %e эквивалентны и используются для ввода чисел типа float. Обе спецификации допускают наличие (или отсутствие) знака, строки цифр с десятичной точкой или без нее и поля показателя степени.

3. Форматы %lf и %le определяют тип вводимых данных как double.

Составной оператор

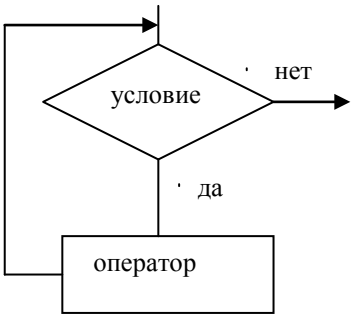
Составной оператор состоит из одного или большего числа операторов любого типа, заключенных в фигурные скобки.

Пример:

```
{ x=1; y=2; z=3; }
```

Оператор цикла while

Оператор цикла служит для многократного выполнения одних и тех же действий. Оператор while используется при программировании циклов с предусловием:



Формат оператора while:

while (условие)
оператор

Оператор while выполняется так, как изображено на схеме. Проверяется условие; если оно истинно, то выполняется оператор, входящий в состав while (так называемое "тело цикла"). Затем снова проверяется условие... Тело цикла будет выполняться до тех пор, пока условие не станет ложным. Затем управление передается следующему оператору программы.

Обратите внимание, что тело цикла – это один оператор: либо простой, либо составной.

Условие - это выражение, которое кроме арифметических операций может содержать операции отношения:

> больше,
>= больше или равно,
< меньше,
<= меньше или равно,
== равно,

!= не равно.

Пример оператора:

```
while (i<=n)
{ s=s+i/(i+1); i++; }
```

Пример.

Задача. Дано действительное число x. Вычислить значение sinx с помощью ряда

$$y = \sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - ... + (-1)^n \frac{x^{2n+1}}{(2n+1)!} + ...$$

с точностью 10^{-5} (т.е. учитывая только те члены ряда, которые по абсолютной величине больше либо равны 10^{-5}). Для проверки результата вычислить sinx с помощью стандартной функции.

Обозначим очередной член данного ряда через a_n .

$a_0 = x$

$$a_1 = -\frac{x^3}{3!} = -a_0 * \frac{x^2}{2*3}$$

$$a_2 = +\frac{x^5}{5!} = -a_1 * \frac{x^2}{4*5}$$

….

$$a_n = -a_{n-1} * \frac{x^2}{2n(2n+1)}$$

Программа:

```
/* Приближенное вычисление sin x */
#include <stdio.h>
#include <math.h>
#define E 1e-5 /* точность вычисления */

main()
{
        float x,        /* аргумент функции */
        y;              /* сумма ряда */
        float a;        /* очередной член ряда */
        int n;          /* номер итерации */
        printf ("x=");
        scanf ("%f",&x);
        y=0; a=x; n=0;
        while ( fabs(a) >= E )
                { y=y+a;
                  n++;        /* n=n+1; */
```

```
            a=-a*x*x/(2*n*(2*n+1));      /* выч-е очередного члена ряда через
                                            предыдущий */
        }
    printf ("y=%f\n",y);
    printf ("sinx=%f\n",sin(x));
}
```

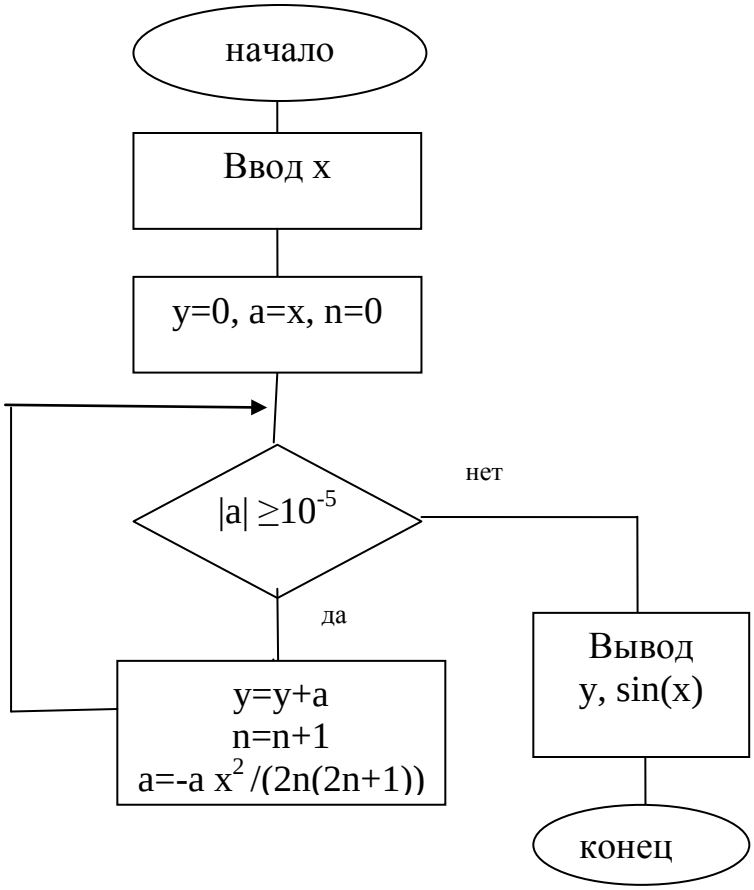


Рис.1. Блок-схема алгоритма вычисления y=sin x

Примеры результатов выполнения программы:

x=3.14159
y=0.000002
sinx=0.000003

x=0
y=0.000000
sinx=0.000000

x=1.5708
y=1.000004
sinx=1.000000

x=-1.5708
y=-1.000004
sinx=-1.000000

Ввод и отладка программы в TURBO C

1. Запустите систему TURBO C.
2. Если окно *Edit* не пустое, выберите команды меню **File | New** для создания нового файла.

3. В окне *Edit* введите текст своей программы, соблюдая ступенчатую запись.После ввода каждой строки нажимайте [Enter].Если заметите ошибки, их легко исправить, вернув курсор в нужную позицию с помощью клавиш управления движением курсора. Работа с редактором Turbo-системы описана на стр.11.

4. Выберите команды меню **Compile | Compile** для компиляции программы.

При компиляции могут быть обнаружены синтаксические ошибки в программе. В таком случае в верхней части окна редактирования появится первое сообщение об ошибке, а курсор укажет на предполагаемое местоположение ошибки в программе. Исправьте ошибку и повторите компиляцию.

5. После исправления всех синтаксических ошибок сохраните программу в файле на диске, выполнив команды меню **File | Save** (или нажав клавишу [F2]).Если на экране появится запрос на изменение имени файла, сотрите *NONAME.C* (это имя дается вначале всем новым файлам) и введите любое другое имя (идентификатор длиной до 8 символов). Расширение *C* добавляется к имени автоматически.

Замечание. При многократном сохранении программы сохраняется обычно две последние версии программы: последняя - в файле с расширением *C*, а предпоследняя - в файле с расширением *BAK*.

6. Запустите программу на выполнение, выбрав команды меню **Run| Run** (или нажав одновременно клавиши [Ctrl] и [F9]).

По запросу программы введите тестовые исходные данные. Проверьте результат. Вернуться к экрану с результатами можно, выбрав команды меню **Run | User Screen** (или нажав одновременно клавиши [Alt] и [F5]).

Для возврата к экрану системы нажмите любую клавишу.

Если результаты оказались неверными, проверьте программу. После редактирования программы повторите действия с пункта 4.

Если программа зациклилась, попытайтесь прервать ее выполнение с помощью одновременного нажатия клавиш [Ctrl] и [Break]. Если это сделать не удалось, перезагрузите систему. Если в окне редактирования появится не Ваша программа, выполните команды меню **File | Load** (можно с помощью клавиши [F3]). На запрос имени файла введите то имя, которое дали файлу при его сохранении. Проверьте и отредактируйте программу и повторите действия с пункта 4.

Порядок выполнения работы.

1. Познакомиться с описанием языка Си и примером программы.
2. Получить у преподавателя индивидуальное задание.

3. Составить схему и программу на языке Си и подобрать тесты для проверки программы на ЭВМ.

4. Отладить программу на компьютере и показать результаты тестирования преподавателю.

5. Оформить и сдать отчет по лабораторной работе.

Задания.

1) Дано действительное число x. Вычислить значение y с помощью стандартной функции и с помощью ряда с точностью 0,0001:

1. $y = \cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{(2n)!} + \ldots$

2. $y = e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} - \ldots + \frac{x^n}{n!} + \ldots$

3. $y = \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \ldots + (-1)^{n-1} \frac{x^n}{n} + \ldots$, где $|x| < 1$.

4. $y = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \ldots + \frac{1}{n^2} + \ldots$

5. $y = 1 - \frac{1}{3} + \frac{1}{5} - \ldots + (-1)^n \frac{1}{2n+1} + \ldots$

6. $y = \frac{1}{x} + \frac{1}{2!x^2} - \frac{1}{3!x^3} + \ldots + \frac{(-1)^{n+2}}{n!x^n} + \ldots$

7. $y = ex = x + \frac{x}{1!} + \frac{x}{2!} + \ldots + \frac{x}{n!} + \ldots$

8. $y = \frac{1}{e^x} = -x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \ldots + \frac{(-x)^n}{n!} + \ldots$

9. $y = e^x = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \ldots + (-1)^n \frac{x^n}{n!} + \ldots$

10. $y = \ln(x) = 2\left(\frac{x-1}{x+1} + \frac{(x-1)^3}{3(x+1)^3} + \frac{(x-1)^5}{5(x+1)^5} + \ldots \right)$, где x>0

11. $y = \ln(1-x) = -\left(\frac{x^2}{2} + \frac{x^3}{3} + \ldots + \frac{x^n}{n} + \ldots \right)$, где |x|<1

12. $y = arctg(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$, где $|x| \le 1$

13. $y = \ln\frac{x+1}{x-1} = 2\left(\frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \ldots \right)$, где |x|>1

14. $y = \ln(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \ldots$, где $0 < x \le 2$

15. $y = \ln(x) = \frac{x-1}{x} + \frac{(x-1)^2}{2x^2} + \frac{(x-1)^3}{3x^3} + \ldots$, где $x > 0.5$

2) Дано натуральное число n. Проверить справедливость равенства:

16. $1+2+3+\ldots+n = \frac{n(n+1)}{2}$

17. $1^2+2^2+3^2+\ldots+n^2 = \frac{n(n+1)(2n+1)}{6}$

18. $1+2+2^2+\ldots 2^{n-1}=2^n-1$

3) Дано натуральное число n.

19. Определить количество цифр в числе n.
20. Определить сумму его цифр.
21. Определить первую цифру числа n.

Лабораторная работа № 2

Обработка числовых последовательностей

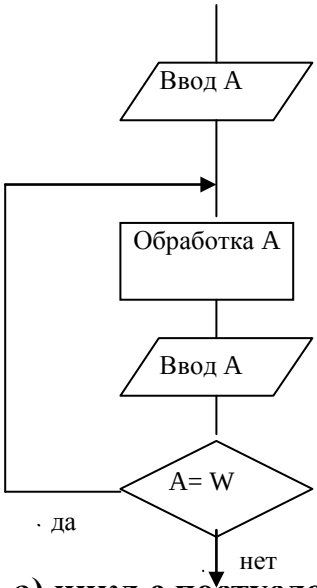
Существует круг задач, в которых необходимо как-то обработать заданную числовую последовательность, причем для получения результата достаточно просмотреть последовательность один раз. Например, чтобы вычислить среднее арифметическое заданной последовательности чисел, можно суммирование чисел и подсчет их количества совместить со вводом. Тогда не нужно будет хранить всю последовательность в памяти компьютера (в виде массива), достаточно иметь одну скалярную переменную целого или вещественного типа и поочередно присваивать ей вводимые значения.

Числовая последовательность может задаваться с указанием количества чисел или иметь какой-то признак конца.

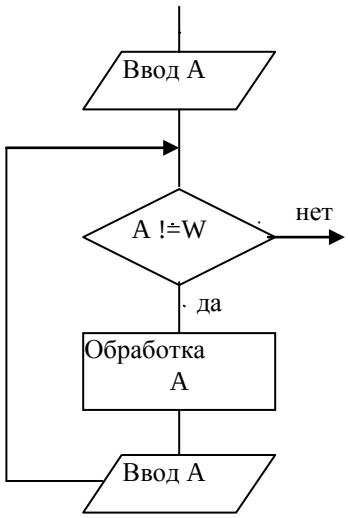
Пусть последовательность задается в виде

$A_1, A_2, ..., A_n, W$

(n заранее неизвестно, W – признак конца последовательности), тогда процесс обработки в общем виде можно представить одной из двух схем:



а) цикл с постусловием



б) цикл с предусловием

Фрагменты программ на Си, соответствующие этим схемам:

а)
```
scanf ("%f",&a);
do
```

б)
```
scanf ("%f",&a);
while (a!=W)
```

```
{ /* обработка a */
. . .
scanf ("%f",&a);
}
while (a!=W);
```

```
{ /* обработка a */
. . .
scanf ("%f",&a);
}
```

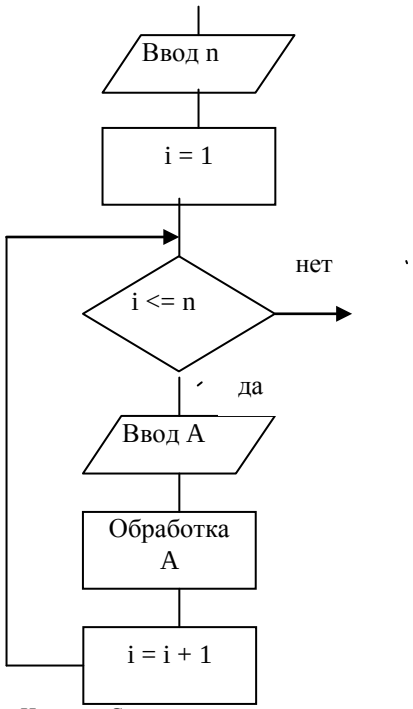
В этих фрагментах предполагается, что переменная ***a*** вещественного типа (***float***), поэтому указан формат ***%f***. Если же последовательность состоит из целых чисел типа int, то следует выбрать формат ***%d***; *W* – символическая константа, которая должна быть определена с помощью директивы ***#define***.

Числовая последовательность может быть задана и с указанием количества вводимых чисел:

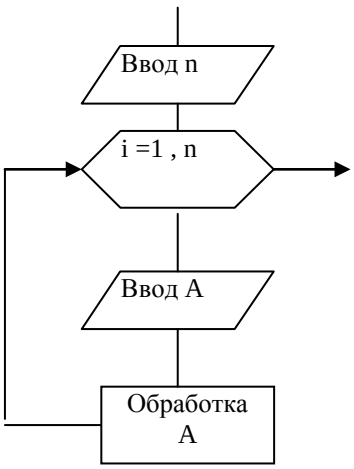
n, A_1 , A_2 , ..., A_n .

Тогда процесс обработки можно представить в виде:

а) полная форма



б) более короткая форма

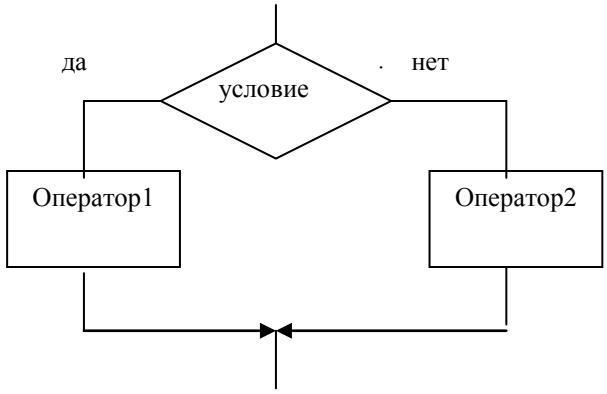


На языке Си этот процесс можно записать с помощью оператора цикла while или лучше оператора цикла for:

```
a) scanf   ("%d",&n); i=1;
while (i<=n)
{
scanf ("%f",&a);
/* обработка a */
. . .
i++;
}
```

```
б)  scanf ("%d",&n);
for (i=1; i<=n; i++)
{
scanf ("%f",&a);
/* обработка a */
. . .
}
```

Процесс обработки очередного числа может иметь линейную структуру (состоять из нескольких последовательно выполняемых операторов) или структуру ветвления, которая программируется с помощью оператора ***if***:



if (условие) оператор1
else оператор2

В операторе if *оператор1* и *оператор2* могут быть составными операторами, конструкция "*else оператор2*" может отсутствовать, например:

```
if   (a % 2 != 0)
        { s=s+a;
          k++;
        }
```

Примечание. В операторах ***if, while, do-while*** можно использовать не только условные выражения, но и вообще любые. Это связано с тем, что в Си значение "ложь" – это 0, а "истина" – любое ненулевое число. Поэтому предыдущий оператор if можно записать иначе:

```
if   (a % 2)   { s=s+a;   k++; }
```

Пример.

Задача. Даны целые числа n, A_1 ,A_2 , ... ,A_n. Вычислить сумму тех чисел последовательности, которые удовлетворяют условию $|A_i| < i^2$.

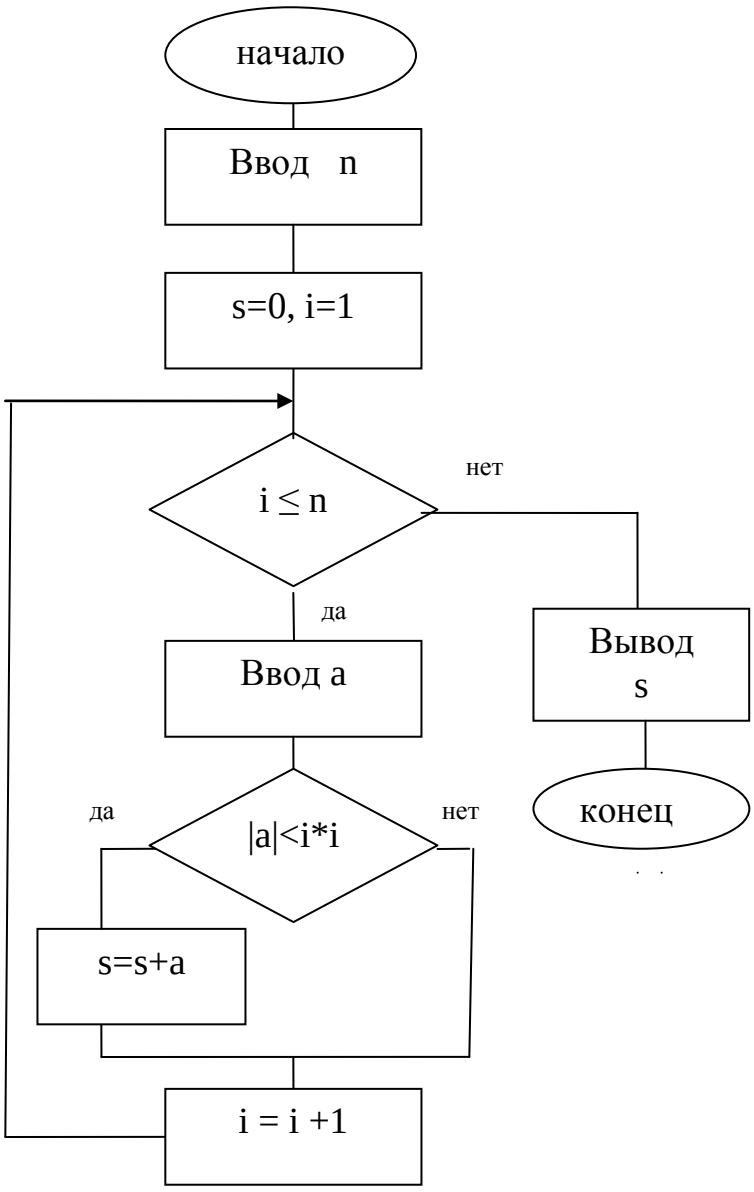


Рис. 1. Схема алгоритма решения задачи

Программа:

```
#include <stdio.h>
#include <math.h>
main()
{ int n;                /* количество чисел   */
   int a,               /* очередное число */
   s=0,                 /* сумма*/
   i;                   /* порядковый номер числа в посл-ти */
   printf ("\nВведите количество чисел: ");
   scanf ("%d",&n);
   printf ("Введите числовую последовательноть:\n");
   for (i=1; i<=n; i++)
       {scanf ("%d",&a);
```

```
        if (abs(a) < i*i)   s=s+a;
       }
   printf ("сумма=%d\n",s);
}
```

Тесты для проверки программы:

№ п/п	n	Исходная последовательность:	Ожидаемый результат:
1	6	1 -2 3 16 -5 40	Сумма=-4
2	4	-1 5 10 -20	Сумма=0

Результаты тестирования программы:

Введите количество чисел: 6
Введите числовую последовательность:
1 -2 3 16 -5 40
сумма=-4

Введите количество чисел: 4
Введите числовую последовательность:
-1 5 10 -20
сумма=0

Порядок выполнения работы.

1. Получить задание у преподавателя.
2. Составить схему и программу на языке Си, подобрать тесты для проверки программы на компьютере.
3. Отладить программу на компьютере и показать результаты тестирования преподавателю.
4. Оформить и сдать отчет по лабораторной работе.

Задания.

1. Даны натуральные числа A_1, A_2,… Признак конца последовательности 0. Получить количество и сумму тех членов последовательности, которые делятся на 5 и не делятся на 7.
2. Даны натуральные числа A_1, A_2,… Признак конца последовательности 0. Найти $\min(A_1+A_2, A_2+A_3,\ldots)$.
3. Даны натуральные числа A_1, A_2,… Признак конца последовательности 0. Найти наибольший член последовательности.
4. Даны натуральное число n и последовательность натуральных чисел A_1, A_2,…, A_n. Определить количество членов последовательности имеющих четные порядковые номера и являющихся нечетными числами.
5. Даны натуральное число n и последовательность действительных чисел A_1, A_2,…, A_n. Получить удвоенную сумму всех положительных членов последовательности.

6. Даны натуральное число n и последовательность действительных чисел A_1, A_2,…, A_n. Получить сумму отрицательных и количество положительных элементов последовательности.
7. Даны натуральное число n и последовательность действительных чисел A_1, A_2,…, A_n. Получить $\min(A_1, A_3, A_5, \ldots) + \max(A_2, A_4, A_6, \ldots)$.
8. Даны натуральное число n и последовательность действительных чисел A_1, A_2,…, A_n. Получить $(|A_1|, |A_2|, |A_3|, \ldots, |A_n|) * n$.
9. Даны натуральные числа A_1, A_2,… Признак конца последовательности 0. Найти наименьший член последовательности.
10. Даны натуральное число n и последовательность натуральных чисел A_1, A_2,…, A_n. Определить количество членов последовательности кратных 3 и не кратных 5.
11. Даны натуральные числа A_1, A_2,… Признак конца последовательности 0. Найти $\max(A_1 - A_2, A_2 - A_3, \ldots)$.
12. Даны натуральное число n и последовательность действительных чисел A_1, A_2,…, A_n. Получить $A_1 * A_2 + A_2 * A_3 + \ldots + A_{n-1} * A_n$.
13. Даны натуральное число n и последовательность действительных чисел A_1, A_2,…, A_n. Получить $(A_2 - A_1) * (A_3 - A_2) * \ldots * (A_n - A_{n-1})$.
14. Даны натуральное число n и последовательность действительных чисел A_1, A_2,…, A_n. Проверить упорядочены ли числа последовательности по убыванию;
15. Даны натуральное число n и последовательность действительных чисел A_1, A_2,…, A_n. Проверить есть ли в последовательности одинаковые соседние числа.
16. Даны вещественные числа A_1, A_2 , ...Признак конца последовательности число 999. Выяснить, составляют ли числа возрастающую последовательность.
17. Даны вещественные числа A_1, A_2 , ...Признак конца последовательности число 999. Получить среднее арифметическое всех чисел.
18. Даны вещественные числа A_1, A_2 , ...Признак конца последовательности число 999. Определить сумму положительных и произведение отрицательных чисел последовательности.
19. Даны вещественные числа A_1, A_2 , ...Признак конца последовательности число 999. Определить разность между наибольшим числом и наименьшим числом последовательности.
20. Даны вещественные числа A_1, A_2 , ...Признак конца последовательности число 999. Определить, сколько раз встречается в последовательности наибольшее число.

Лабораторная работа № 3.
Тема «Символьные последовательности»

Если для решения задачи достаточно просмотреть исходный текст один раз, то обычно текст вводится и обрабатывается посимвольно и не хранится целиком в памяти (в виде массива). В программе используется переменная типа ***char***, которой поочередно присваиваются значения символов исходного текста. Ввод и обработка символов происходит до тех пор, пока не встретится признак конца текста или количество введенных символов не достигнет заданной длины текста.

Процессы обработки последовательностей символов можно представить такими

же схемами, как и процессы обработки числовых последовательностей (см. описание предыдущей лабораторной работы). Только переменной будет присваиваться при вводе не число, а очередной символ.

Функции *getchar* и *putchar*

Функции ***getchar*** и ***putchar*** служат соответственно для ввода и вывода одного символа. Для посимвольного ввода/вывода текстов лучше использовать эти функции, нежели scanf и printf.

Функция ***getchar()*** не имеет аргументов. Она получает очередной поступающий с клавиатуры символ и сама возвращает его значение выполняемой программе.

Пример вызова функции getchar:

```
char ch;
ch=getchar();
```

Функция ***putchar*** имеет один аргумент - это символ, который требуется вывести на экран.

Примеры вызова функции putchar:

```
putchar (ch);/* ch - переменная типа char */
putchar ('S');
putchar ('\n');        /* перевод строки */
```

Определения функций getchar и putchar содержатся в файле stdio.h.

Пример.

Задача. Дан текст произвольной длины, оканчивающийся точкой. Проверить, есть ли в тексте сочетания "ВА".

Программа:

```
#include <stdio.h>
main()
{char s;       /* текущий символ текста */
  char prs;    /* предыдущий символ  */
  short net=1;        /* признак, имеется ли "ВА" в тексте */
                      /* net=1, если "ВА" нет*/
                      /* net=0, если "ВА" есть*/
  printf ("\nВведите текст.\n");
  s=getchar();        /* чтение первого символа */
  if (s!='.')
  { do
        { prs=s; s=getchar();
          if (prs=='B' && s=='A') net=0;
        }
    while (s!='.');
  }
  if (net) printf ("В тексте нет 'ВА'.\n");
  else printf ("В тексте есть 'ВА'.\n");
}
```

Тесты для проверки программы.

№ теста	Исходный текст	Ожидаемый результат
1	МОСКВА, БЕРЛИН, ВАРНА .	В тексте есть 'ВА'.
2	ПЭВМ IBM PC .	В тексте нет 'ВА'.
3	.	В тексте нет 'ВА'.

Порядок выполнения работы.

1. Получить у преподавателя задание.
2. Составить схему и программу на Си и подобрать тесты для проверки программы на ЭВМ.
3. Отладить программу и показать результаты тестирования преподавателю.
4. Оформить и сдать отчет по лабораторной работе.

Задания для самостоятельного выполнения:

1. Дан текст произвольной длины, оканчивающийся точкой. Текст состоит из слов разделенных пробелами. Подсчитать количество слов, оканчивающихся буквой А.
2. Дан текст произвольной длины, оканчивающийся «;». Проверить есть ли в тексте скобки.
3. Дан текст произвольной длины, оканчивающийся «;». Подсчитать количество цифр в тексте.
4. Дан текст произвольной длины, оканчивающийся точкой. Подсчитать количество сочетаний «:=».
5. Дан текст произвольной длины, оканчивающийся точкой. Текст состоит из слов разделенных пробелами. Подсчитать количество слов в данном тексте.
6. Дан текст произвольной длины, оканчивающийся точкой. Текст состоит из слов разделенных пробелами. Подсчитать количество слов, начинающихся с буквы К.
7. Дан текст произвольной длины, оканчивающийся точкой. Подсчитать количество сочетаний «СА».
8. Дан текст произвольной длины, оканчивающийся точкой. Подсчитать количество цифр и символов (пробелы, тире, точка с запятой и т.д.) в тексте.
9. Дан текст произвольной длины, оканчивающийся точкой. Найти порядковый номер первой запятой.
10. Дан текст произвольной длины, оканчивающийся точкой. Подсчитать количество букв в тексте.
11. Дан текст произвольной длины, оканчивающийся «;». Подсчитать количество заглавных букв в тексте.
12. Дано скобочное выражение, оканчивающееся точкой с запятой. Проверить правильность расстановки скобок в выражении.
13. Дано скобочное выражение, оканчивающееся точкой с запятой. Подсчитать количество уровней вложенности скобок в выражении.
14. Дана строка символов. Признак конца - символ '\n' (перевод строки). Удалить лишние пробелы, т.е. если подряд следует несколько пробелов, оставить только один.

15. Дана строка символов. Признак конца - символ '\n' (перевод строки). Удалить последовательности символов, заключенные в фигурные скобки.

Лабораторная работа № 4.

Одномерные массивы

Массив используется, когда дана упорядоченная совокупность однотипных данных (чисел, символов, строк символов и т.д.) с ограниченным числом элементов.

Примеры описаний массивов:

```
char text[10];        /* массив из 10 символов*/
int a[50];            /* массив из 50 целых чисел*/
float matr[5][10];    /* матрица вещ.чисел разм.5x10 */
```

Для обращения к элементу массива указываются имя массива и индексы элемента в квадратных скобках, например, text[0], a[i+1], matr[i][j]. Индексация начинается с 0 (в приведенном примере text[0] - первый элемент массива, последний элемент имеет индекс 9).

Ввод/вывод массивов осуществляется в цикле поэлементно.

Пример программы:

Первый вариант:

```
   /*****************************************************/
   /* Задача. Входная строка содержит последовательность     */
   /* слов, разделенных пробелами. Признак конца строки -   */
   /* символ '\n' (перевод строки). Вывести на экран слова       */
   /* длиной до пяти символов.                                                  */
   /*****************************************************/
#include <stdio.h>
#define DLSL 80   /* макс. длина слова */

main()
{ char s;       /* тек. символ */
  char sl[DLSL];    /* тек. слово*/
  int i,j;        /* индексы тек. символа в слове */
  int psl=1;   /* признак, что слово длиной до 5 симв. первое */
  printf ("\n\nВведите строку символов\n");
  s=getchar();
  while (s!='\n')
       { if (s==' ') s=getchar();
         else
              { i=0;
                   do
                        { sl[i++]=s;
                          s=getchar();
                        }
```

```
            while ((s!=' ') && (s!='\n'));
        if (i<5)
            { if (psl) /* если слово первое */
                { printf ("Слова длиной до 5 символов:\n");
                    psl=0;
                }
              for (j=0; j<i; j++)
                  putchar(sl[j]);
                putchar(' ');
            }
          }
        }
        if (psl) printf ("Слов длиной до 5 символов нет");
    }
```

Пример результатов тестирования программы:

```
Введите строку символов
май   апрель март весна лето
Слова длиной до 5 символов:
май март лето

Введите строку символов
декабрь январь февраль
Слов длиной до 5 символов нет
```

Второй вариант программы:

```
   /*****************************************************/
   /* Задача. Входная строка содержит последовательность     */
   /* слов, разделенных пробелами. Признак конца строки -   */
   /* символ '\n' (перевод строки). Вывести на экран слова       */
   /* длиной до пяти символов.                                                  */
   /*****************************************************/
#include <stdio.h>
#define DLSTR 80/* макс.длина строки */

main()
{ char str[DLSTR];          /* тек. строка */
  int i,j;          /* индексы тек. символа в строке */
  int n,k;        /* индексы перв. и посл. символов тек. Слова в строке */
  int net_sl=1;         /* признак, что слов длиной до 5 симв. нет */
  printf ("\n\nВведите строку символов\n");
  gets(str);    /* ввод строки в массив str с заменой символа '\n' на признак
                   конца строки '\0'   */
  printf ("Результат:\n");
  i=0;
```

```
  while (str[i]!='\0')
        { if (str[i]==' ')   i++;
          else
                { n=i;
                  do i++; while ((str[i]!=' ') && (str[i]!='\0'));
                  k=i;
                  if ( k-n < 5 )
                        { for (j=n; j<k; j++)
                                putchar(str[j]);
                          putchar(' ');
                          net_sl=0;
                          }
                  }
        }
if (net_sl) printf ("Слов длиной до 5 символов нет.");
printf ("\nДля завершения нажмите любую клавишу");
getch(); /* чтение символа без отображения его на экране */
}
```

Пример результатов тестирования программы:

Введите строку символов
весна лето осень зима
Результат:
лето зима
Для завершения нажмите любую клавишу

Введите строку символов
декабрь январь февраль
Результат:
Слов длиной до 5 символов нет.
Для завершения нажмите любую клавишу

Порядок выполнения работы.

1. Получить задание у преподавателя.
2. Составить программу на Си и подобрать тесты для проверки программы на ЭВМ.
3. Отладить программу на ЭВМ.
4. Оформить и сдать отчет по лабораторной работе.

Задания.

1. Дана последовательность из N различных вещественных чисел. Найти сумму чисел, расположенных между максимальным и минимальным числами.

2. Дан текст произвольной длины. Определить сколько раз встречается каждая цифра в тексте.
3. Задан текст, произвольной длины, состоящий из слов, разделенных пробелами. Вывести самое короткое слово.
4. Дан текст, произвольной длины. Определить символ, встречающийся в тексте с максимальной частотой.
5. Дан текст, длиной не более 80 символов, заканчивающийся точкой. Определить, симметричен ли он.
6. Заданы две упорядоченные по возрастанию последовательности из N чисел и M чисел. Составить программу, объединяющую их в одну упорядоченную по возрастанию последовательность.
7. Дан массив из 10 вещественных чисел. Упорядочить массив по возрастанию методом последовательного нахождения минимума.
8. Дан одномерный массив, размером в 12 элементов. Необходимо инвертировать массив, путем замены значения первого элемента на последний, второго на предпоследний и так далее.
9. Дан одномерный массив A, размером в N элементов. Необходимо определить является массив симметричным, т.е. A[0]=A[N-1], A[1]=A[N-2] и т.д.
10. Задан текст, произвольной длины, состоящий из слов, разделенных пробелами. Вывести только те слова, которые содержат букву ‘a’.
11. Дана последовательность из N вещественных чисел. Распечатать числа в обратном порядке по 6 чисел в строке.
12. Задан текст, произвольной длины, состоящий из слов, разделенных пробелами. Вывести только те слова, которые не содержат букву ‘a’.
13. Дан одномерный массив A, размером в N элементов. Упорядочить его элементы по убыванию методом пузырька.
14. Дан одномерный массив A, размером в N элементов. Вывести номер первого отрицательного элемента.
15. Дан текст, произвольной длины. Определить сколько раз встречается каждая латинская буква в тексте.
16. Дана строка символов. Признак конца ‘\n’. Вывести слова заканчивающиеся буквой ‘a’, с порядковым номером этих слов.

Лабораторная работа № 5.
Двумерные массивы (матрицы)

Массивы в C могут быть не только одномерными, т.е. когда данные визуально выстроены в одну линию. Массивы также могут быть и двумерными, трехмерными и так далее. C++ компиляторы поддерживают как минимум 12-ти мерные массивы!!! Естественно, что такими большими массивами на практике никто не пользуется, т.к. человеку сложно их визуально представить у себя в голове, не то что написать программу, которая оперирует таким сложными массивами. На практике редко

применяют массивы, более трехмерного. Одномерный – это строка, двумерный – матрица (таблица), трехмерный – куб, а вот дальше уже сложно, поэтому дальше, обычно, никто и не идет...

Двумерный массив – это так называемая матрица (или таблица), у которой есть строки и столбцы. По соглашению программистов первый индекс массива будет указывать на строки, а второй на столбцы.

Фактически двумерный массив — это одномерный массив одномерных массивов. Структура двумерного массива, с именем a, размером m на n показана ниже:

a[0][0]	a[0][1]	a[0][2]	a[0][3]	. . .	a[0][n]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	. . .	a[1][n]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	. . .	a[2][n]
. . .	. . .	. . .	. . .	. . .	. . .
a[m][0]	a[m][1]	a[m][2]	a[m][3]	. . .	a[m][n]

где, m – количество строк двумерного массива;
n – количество столбцов двумерного массива;
m * n – количество элементов массива.

Вот пример объявления и инициализации двумерного массива, состоящего из трех строк и пяти столбцов.

```
int aMatrix[3][5] = {{3, 5, 5, 7, 8},
                     {4, 1, 1, 2, 9},
                     {3, 8, 8, 9, 7}};
```

Двумерный массив имеет два индекса. Можно и так записать, как показано ниже. Разницы для компилятора не будет никакой. Разве лишь разница будет в визуальном восприятии для человека:

```
int aMatrix[3][5] = {{3, 5, 5, 7, 8}, {4, 1, 1, 2, 9}, {3, 8, 8, 9, 7}};
```

Либо вообще так, без указания фигурных скобок, которые логически разделяют строки друг от друга.

```
int aMatrix[3][5] = {3, 5, 5, 7, 8, 4, 1, 1, 2, 9, 3, 8, 8, 9, 7};
```

Последняя запись демонстрирует то, как на самом деле элементы массива размещаются в памяти компьютера.

Для доступа к элементам двумерного массива нужно, так же, как и для одномерного указать индекс. В данном случае нужно будет указывать два индекса. Например, чтобы перезаписать последний элемент второй строки, мы должны использовать такую запись

```
aMatrix[1][4] = 0;
```

В этом случае мы перезапишем значение 9 на 0.

Для прохода по двумерному массиву удобнее всего использовать два цикла *for*, вложенных друг в друга.

Пример программы:

Дана матрица размером n строк и m столбцов (0<n<=20, 0< m<=30). Найти сумму положительных элементов каждой строки.

```
#include <stdio.h>
#include <conio.h>
/* Сумма по строкам*/
#define NMAX 20
#define MMAX 30

main()
{float x;
  int i,j;
  float s;
  int n,m;
  puts("Введи n m"); scanf("%d%d",&n,&m);
  puts ("Введи матрицу");
  for (i=0;i<n; i++)
        {s=0;
          for(j=0;j<m;j++)
                {scanf("%f", &x);
                  if (x>0) s=s+x;
                }
          printf ("\n Строка %d, s= %.2f",i,s);
        }
getch();
return 0;
}
```

Пример результатов тестирования программы:

	1	2	3	4	5
1	-9	0	1	-7	4
2	1	2	5	0	4
3	-1	2	-9	7	-1
4	-2	-3	-8	-9	-4

Результат:
Строка 1 сумма = 5
Строка 2 сумма = 12
Строка 3 сумма = 9
Строка 4 сумма = 0

Порядок выполнения работы.

1. Получить задание у преподавателя.
2. Составить программу на Си и подобрать тесты для проверки программы на ЭВМ.
3. Отладить программу на ЭВМ.
4. Оформить и сдать отчет по лабораторной работе.

Задания для самостоятельного выполнения

1. Дана матрица размером n x m. Вычислить сумму положительных элементов каждой строки и найти номер строки, в которой эта сумма максимальна.
2. Определить, является ли заданная целочисленная квадратная матрица магическим квадратом, то есть такой, что суммы всех элементов во всех строках и столбцах одинаковы.
3. Дана матрица размером n x m. Найти минимальный элемент в каждом столбце матрицы (из ненулевых) и разделить на него все элементы этого столбца.
4. Дана матрица размером n x m. В каждой строке переставить местами наибольший и наименьший элементы матрицы.
5. Дана квадратная матрица размером n x n. Вычислить сумму элементов, расположенных над главной диагональю.
6. Дана квадратная матрица размером n x n. Найти наибольший элемент главной диагонали, и вывести всю строку, в которой он находится.
7. Дана матрица размером n x m. Найти наибольший положительный и наименьший отрицательный элементы матрицы.
8. Дана квадратная матрица размером n x n. Найти строку с наибольшим количеством отрицательных элементов.
9. Элементы матрицы A(n x n) назовем седловой точкой, если он одновременно является наименьшим в своей строке и наибольшим в своем столбце или наоборот. Найти индексы всех седловых точек матрицы.
10. Дана квадратная матрица размером n x n. Найти сумму отрицательных элементов в каждой строке.
11. Дан двумерный массив n x m элементов, найти количество четных и нечетных чисел в массиве.
12. Дан двумерный массив n x m элементов. Определить, сколько раз встречается число 7 среди элементов массива.
13. Дан массив из n x m элементов. Определить, сколько элементов массива больше заданного числа.
14. Дан массив из n x m элементов. Найти индексы первого наименьшего элемента массива.
15. Дан квадратный массив из n x n элементов. Найти сумму элементов побочной диагонали.
16. Дан квадратный массив из n x n элементов. Найти сумму элементов ниже главной диагонали.

Лабораторная работа № 6

Функции

Вы уже знакомы с некоторыми библиотечными функциями, такими как printf(),

scanf(), getchar(), putchar(), gets(), sin(), cos(), ... Теперь нужно знать, как создавать свои собственные функции.

Функция – это самостоятельная единица программы, предназначенная для решения определенной задачи. Функции в языке Си играют ту же роль, какую играют функции, подпрограммы и процедуры в других языках программирования.

Программа на Си может состоять из любого количества функций, одна из которых всегда носит имя main. Выполнение программы начинается с функции main(), которая может вызывать другие функции. Те в свою очередь тоже могут вызывать какие-то функции.

Описания функций могут размещаться в одном файле или в нескольких. Рассмотрим случай, когда все функции программы описываются в одном файле. В этом случае расположить в файле описания функций можно в любом порядке. Но проще описание каждой функции поместить перед ее использованием (вызовом) в других функциях.

Описание любой функции имеет вид:

Заголовок функции
{Описания локальных переменных
Операторы
}

Заголовок функции имеет формат:

тип_функции имя_функции (список_параметров)

Через параметры (аргументы) происходит передача данных между функцией и вызывающей программой. В списке параметров перечисляются через запятую описания параметров (перед каждым параметром указывается тип). Имена параметров могут быть любые, т.к. это формальные параметры. Если функция не имеет параметров, то пишутся пустые скобки.

Примеры заголовков функций:

```
main()
float max (float x, float y)
int prov (char mas[], int s)
void line (ink k, int n, char simv, int ps)
```

Тип функции, указанный перед именем функции, - это тип возвращаемого функцией значения. По умолчанию предполагается тип ***int***. Если функция никакое значение не возвращает, то указывается тип ***void.***

Допускается запись заголовка функции в иной форме, когда в списке параметров перечисляются только имена, а описания параметров выносятся за скобки. Например:

```
void line (k,n,simv,ps)
        int k,n;
        char simv;
        int ps;
```

Но такая форма записи уже устарела (и имеет недостатки).

Значение функции передается в вызывающую программу с помощью оператора возврата ***return.***

Пример описания функции:

```
/*функция определения наибольшего из двух чисел*/
float max ( float x, float y )
{if (x>y) return x;
        else return y;
}

        /*2-й вариант*/
float max (float x, float y)
{ float z;      /*z=max(x,y)*/
   z = (x>y) ? x : y ;
   return z;
}
```

Вы можете вызвать свою функцию в любом выражении, указав ее имя и аргументы (фактические параметры):

имя_функции (аргумент_1, аргумент_2, ...)

Аргументом может быть идентификатор, константа и выражение. Типы аргумента и соответствующего параметра в заголовке функции должны совпадать. Например, оператор

```
f = max(a,b) - max(c,d);
```

содержит два вызова приведенной выше функции `max`. При первом обращении функции `max` передаются значения переменных `a` и `b`, она возвращает наибольшее из этих чисел, которое подставляется вместо указателя функции `max(a,b)`. При втором вызове функции `max` формальным параметрам `x` и `y` присваиваются соответственно значения фактических параметров `c` и `d`. Оператор `return` возвращает наибольшее из этих значений в точку вызова функции.

Вызов функции без параметров имеет вид: **имя_функции ().**

Использование указателей при передаче параметров

Обмен информацией между вызывающей программой и функцией осуществляется благодаря параметрам. Если данные входные и функцией не изменяются, то передавать в списке параметров нужно значения данных. Сложнее вернуть вызывающей программе выходные данные, если их несколько (если результатом выполнения функции является одно единственное значение, то оно обычно, как вы уже видели, в число параметров не включается, а является значением

функции). Если функция должна вернуть вызывающей программе нес колько значений, в этом случае передавать ей в качестве фактических параметров нужно **адреса** переменных. Вспомните вызов функции scanf(). Если, например, нужно ввести значения двух переменных x и y типа int, вы должны написать:

scanf ("%d %d",&x,&y);

где &x,&y - адреса переменных x и y. Точно так же при обращении к своей функции для выходных параметров нужно указывать адреса.

В описании функции соответствующие формальные параметры должны быть описаны как **указатели** на переменные соответствующего типа.

Допустим, функция function1() должна изменить значения переменных a и b типа float. В вызывающей программе обращение к функции нужно записать так:

function1 (&a, &b);

а заголовок функции должен иметь вид:

void function1 (float *ptr1, float *ptr2)

Здесь prt1 и ptr2 - это указатели на переменные типа float (названия их могут быть изменены - это формальные параметры). Значениями указателей являются адреса. При вызове функции function1 переменной pt1 присваивается адрес a, а ptr2=&b.

Таким образом функции становятся доступными переменные вызывающей программы. Если нужно этим переменным присвоить новые значения, допустим, с помощью операторов присваивания, то это делается так:

*ptr1 = ... ;

*prt2 = ... ;

Итак, если ptr1 и ptr2 являются указателями на переменные, то чтобы взять значения этих переменных, нужно перед именами указателей использовать операцию ***.

Примеры решения задач

Задача 1. В двух заданных массивах обнулить отрицательные элементы. Для обнуления отрицательных элементов массива составить подпрограмму.

Для обнуления отрицательных элементов массива составим функцию, не обладающую значением, т.к. результатом работы функции должен быть измененный массив (а в массиве много элементов). Функция имеет следующие входные параметры: количество элементов массива – n, исходный массив m[]. Измененный массив m[] является также выходным параметром.

Программа:

```
#include <stdio.h>
#define NMAX 100          /* максимальное количество элементов массива */
 otr (int n, float m[])              /* заголовок функции */
```

```
  {int i;                          /* текущий номер элемента массива */
   for (i=0; i<n; i++)
        if (m[i]<0)
             m[i]=0;
  }
main()
{int n, k;
  int i;
  float m1[NMAX], m2[NMAX];
  printf("\n Введите размер первого массива\n");
  scanf ("%d",&n);
  printf("\n Введите массив из %d элементов\n",n);
  for (i=0; i<n; i++)
        scanf ("%f",&m1[i]);
  printf("\n Введите размер второго массива\n");
  scanf ("%d",&k);
  printf("\n Введите массив из %d элементов\n", k);
  for (i=0; i<k; i++)
        scanf ("%f",&m2[i]);
  otr(n,m1);    /*вызов функции для обнуления отриц. элем. массива m1*/
  otr(k,m2);    /*вызов функции для обнуления отриц. элем. массива m2*/
  printf("\n Измененный массив первый\n");
  for (i=0; i<n; i++)
        printf ("%.2f ",m1[i]);
  printf("\n Измененный массив второй\n");
  for (i=0; i<k; i++)
  printf ("%.2f ",m2[i]);
}
```

Пример результатов тестирования программы:

Дано: массив1: 1 -5 9 10 0 -5
массив2: -1 -8 -9 -4
Вывод: массив1: 1 0 9 10 0 0
массив2: 0 0 0 0

Задача 2. Даны две строки длиной до 80 символов. Определить число латинских букв в каждой строке.

Программа:

```
#include <stdio.h>
/*-------------------------------------------------------------------------------*/
```

```
/* Функция определения количества лат. букв в заданной строке */
/*-------------------------------------------------------------------------------*/
int KolLatBukv (char s[])
{       int i,   /* индекс очередного символа строки s */
        k=0;   /* количество лат. букв */
        for (i = 0; s[i] != '\0'; i++ )
                if (s[i] >= 'a' && s[i] <= 'z' | | s[i] >= 'A' && s[i] <= 'Z') k++;
        return k;
}
/*-----------------------*/
/* Главная функция */
/*-----------------------*/
void main()
{       char s1[81], s2[81];                /* заданные строки */
        printf ("\nВведите две строки символов\n");
        gets (s1);
        gets (s2);
        printf ("В 1-й строке %d лат. букв\n", KolLatBukv (s1));
        printf ("Во  2-й  строке  %d  лат. букв\n", KolLatBukv (s2));
}
```

Пример результата выполнения программы:

```
Введите две строки символов
AaBbcd 123 Zz
t = x + y * z / 10 ;
В 1-й строке 8 лат. букв
Во 2-й строке 4 лат. букв
```

Задача 3. Описать функцию, которая для заданного числового массива определяет сумму и количество положительных элементов.

Программа:

```
#include <stdio.h>
        /* Функция определения суммы и количества */
        /*положительных элементов заданного массива */
void SumPos (float m[], int n, float *s, int *k)
/*      Вх. параметры:
                m – указатель на заданный массив,
                n – число элементов массива.
        Вых. параметры:
        *s –   сумма положительных элементов массива,
        *k – количество положительных элементов         */
```

```
{ int i;
    for (i=0, *s=0, *k=0;   i<n;   i++)
       if (m[i] > 0)     (*s) += m[i],   (*k)++;
}
       /*-----------------------------------------------------------------*/
       /* Главная функция (для тестирования подпрограммы) */
       /*-----------------------------------------------------------------*/
void main()
{float a[6],   /* массив */
    s;          /* сумма положительных элементов */
    int k,              /* количество положительных эл-тов */
        i;              /* индекс элемента массива*/
    printf ("\nВведите 6 чисел\n");
    for ( i=0; i < 6; i++) scanf ("%d ", &a[i] );
    SumPos (a, 6, &s, &k);      /* вызов функции */
    printf ("Сумма положительных чисел =   %f\n", s);
    printf ("Количество положительных чисел:   %d\n", k);
}
```

Порядок выполнения работы.

1. Получить задание у преподавателя.
2. Составить программу на Си и подобрать тесты для проверки программы на ЭВМ.
3. Выполнить тестирование и отладку программы на ЭВМ, показать результаты преподавателю.
4. Оформить и сдать отчет по лабораторной работе.

Задания для самостоятельного выполнения

1. Заданы два массива вещественных чисел. Найти номера минимальных элементов в каждом из них. Нахождение номера минимального элемента оформить как подпрограмму.
2. Заданы два массива целых чисел. Найти сумму четных элементов в каждом из них. Подсчет суммы четных элементов оформить как подпрограмму.
3. Заданы два массива целых чисел. Найти сумму и количество четных элементов в каждом из них. Подсчет суммы и количества четных элементов оформить как подпрограмму.
4. Заданы два массива целых чисел. В каждом из них найти среднее арифметическое значение элементов, кратных пяти. Подсчет среднего арифметического значения элементов массива оформить как подпрограмму.
5. Задан текст, состоящий из n строк. В каждой строке заменить цифры на пробелы. Работу с каждой строкой оформить как подпрограмму.
6. Задан текст, состоящий из n строк. В каждой строке заменить заглавные латинские буквы на строчные, а строчные на заглавные (коды строчных и заглавных букв

отличаются на число 32, например символ ‘а’ код 97, а символ ‘А’ – код 65). Работу с каждой строкой оформить как подпрограмму.

7. Задан текст, состоящий из n строк. В каждой строке заменить два одинаковых соседних символа на два пробела. Работу с каждой строкой оформить как подпрограмму.
8. Задан текст, состоящий из m строк. В каждой строке заменить сочетание символов АВ на два пробела. Работу с каждой строкой оформить как подпрограмму.
9. Задан текст, состоящий из n строк. В каждой строке подсчитать количество заглавных латинских букв и количество строчных латинских. Работу с каждой строкой оформить как подпрограмму.
10. Заданы два массива целых чисел. В каждом из них найти сумму и количество элементов, кратных пяти. Подсчет суммы и количества элементов оформить как подпрограмму.
11. Задан текст, состоящий из n строк. В каждой строке заменить цифры на символ ‘!’. Работу с каждой строкой оформить как подпрограмму.
12. Заданы два массива целых чисел. В каждом из них найти количество четных элементов, кратных пяти. Подсчет количества элементов оформить как подпрограмму.
13. Задан текст, состоящий из n строк. В каждой строке подсчитать количество латинских букв. Работу с каждой строкой оформить как подпрограмму.
14. Задан текст, состоящий из n строк. В каждой строке заменить пробелы на символ ‘,’. Работу с каждой строкой оформить как подпрограмму.
15. Заданы два массива целых чисел. Найти количество четных элементов в каждом из них. Подсчет количества четных элементов оформить как подпрограмму.
16. Задан текст, состоящий из n строк. В каждой строке подсчитать количество цифр. Работу с каждой строкой оформить как подпрограмму.
17. Заданы два массива вещественных чисел. В каждом из них элементы увеличить в заданное количество раз. Изменение массива оформить как подпрограмму.
18. Заданы два массива целых чисел. В каждом из них найти наименьшее среди положительных элементов. Нахождение минимального элемента оформить как подпрограмму.
19. Задан текст, состоящий из n строк. В каждой строке заменить символ ‘,’ на символ ’ ’ (пробел). Работу с каждой строкой оформить как подпрограмму.
20. Заданы два массива вещественных чисел. В каждом из них заменить отрицательные элементы на число 0, а положительные увеличить в два раза. Изменение массива оформить как подпрограмму.
21. Заданы два массива целых чисел. В каждом из них уменьшить четные элементы в два раза. Изменение массива оформить как подпрограмму.
22. Задан текст, состоящий из n строк. В каждой строке все не латинские буквы заменить на пробелы. Работу с каждой строкой оформить как подпрограмму.
23. Заданы два массива целых чисел. В каждом массиве элементы кратные 4 уменьшить в два раза. Изменение массива оформить как подпрограмму.
24. Задан текст, состоящий из n строк. В каждой строке все не цифровые символы заменить на пробелы. Работу с каждой строкой оформить как подпрограмму.

25. Заданы два массива вещественных чисел. Найти сумму минимальных элементов этих массивов. Нахождение минимума в массиве оформить как подпрограмму.
26. Заданы два массива вещественных чисел. В каждом из них нечетные элементы увеличить в 4 раза. Изменение массива оформить как подпрограмму.
27. Задан текст, состоящий из n строк. В каждой строке заменить точки на запятые, а восклицательные знаки на точки. Работу с каждой строкой оформить как подпрограмму.
28. Заданы два массива целых чисел. В каждом из найти количество элементов, принадлежащее заданному диапазону целых чисел от а до в. Изменение массива оформить как подпрограмму.

Лабораторная работа № 7

Символьные строки и функции обработки строк

Строка символов - это последовательность символов произвольной длины, завершающаяся нуль-символом '\0' (все биты в байте нулевые).

Строковые константы записываются в кавычках, например:

"Как Вас зовут?"

Если в программе встречается строковая константа, компилятор выделяет для нее память объемом, равным длине строки (количеству символов) + 1 (для нуль-символа). Признак конца строки '\0' добавляется автоматически.

Строковые переменные описываются либо как массивы символов, либо как указатели на символы, например:

```
char stroka[81];     /* строка длиной до 80 символов */
char *str;           /* указатель на строку */
```

В первом случае память для строки выделяется (имя массива является адресом его первого символа), во втором случае – нет (выделяется только для указателя).

При выполнении оператора

str= "Группа 4101";

указателю str присваивается адрес памяти, где размещена строковая константа "Группа 4101". Для массива такой оператор будет неверным, т.к. адрес массива - величина постоянная.

Еще раз обратите внимание, что в Си имя массива отождествляется с адресом его первого элемента. Для описанного выше массива stroka имя массива stroka и &stroka[0] идентичны. Другими словами, имя массива является указателем на первый символ, причем его нельзя изменять.

Если строка будет вводиться с клавиатуры, то лучше описать переменную как массив символов, иначе предварительно придется выделять память для строки, например, с помощью функции ***malloc().***

Функции gets() и puts()

Для ввода и вывода строк символов служат функции gets() и puts().

Функция gets() вводит с клавиатуры строку, заменяя символ "перевод строки" на нуль-символ, и помещает ее по указанному адресу. Например:

```
gets (stroka);          /* ввод строки в массив строка */
```

Функция puts() выводит указанную строку на экран. Например, оператор

```
puts(str);
```

выведет на экран строку, на которую указывает переменная str, курсор после вывода переместится на новую строку.

Функции обработки строк

Для работы со строками символов в библиотеке Turbo C имеется ряд функций, например, функция определения длины строки strlen(), копирования строк strcpy(), сцепления строк strcat(), сравнения строк strcmp(), нахождения в строке указанного символа strchr(). Прототипы таких функций находятся в файле ***string.h***, их перечень приведен в следующем разделе.

Примеры обращения к функциям обработки строк:

```
char s1[81], s2[81], s3[81];
char *s;
gets(s1);
printf ("Длина строки = %d\n", strlen(s1));
strcpy(s2,s1);          /* копирование строки s1 в массив s2 */
if ((s=strchr(s2,'a'))!=NULL)   /* есть буква 'a' в строке s2 */
        *s = 'b';       /* замена в строке s2 первой буквы 'a' на 'b' */
gets(s3);
if (strcmp(s1,s3)==0) printf ("Строки одинаковые\n");
```

Рассмотрим одну из библиотечных функций – функцию сцепления двух заданных строк strcat(). Определение функции:

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

Функция копирует строку s2 (на которую ссылается указатель s2) в конец строки s1 и возвращает значение s1 - ссылку на сцепленную строку. Например, если s1="2007 ", s2="год", то после сцепления строк s1="2007 год".

Работу функции можно описать так:

```
char *strcat (char *s1, char *s2)
        {char *rs;     /* ссылка на результирующую строку*/
         rs=s1;        /* запоминание адреса начала строки s1 */
         while (*s1!='\0')
                s1++; /* поиск конца строки s1 */
                /* копирование строки s2 в конец s1 */
```

```
        while (*s2!='\0')
                {*s1=*s2; s1++; s2++; }
        *s1='\0';
        return rs;
        }
```

Как видите, функция не проверяет, достаточно ли памяти для результирующей строки. Вызывающая программа должна позаботиться об этом.

А теперь посмотрите на более компактную (но менее понятную) запись этой функции:

```
char *strcat (char *s1, char *s2)
        {char *rs;
         rs=s1;        /* запоминание адреса начала строки s1 */
         while (*s1!='\0') s1++;   /* поиск конца строки s1 */
         while ((*s1++ = *s2++) !='\0');         /* копирование s2 в конец s1, включая
                                                    нуль-символ */
        return rs;
        }
```

Даже еще можно сократить текст функции, записав второй оператор **while** короче:

```
        while (*s1++ = *s2++);
```

Непонятно? Каждый раз очередной символ из второй строки копируется в первую, затем значения указателей s1 и s2 увеличиваются на 1, т.е. происходит продвижение указателей к следующим символам строк. Этот процесс повторяется до тех пор, пока не скопируется нуль-символ (т.к. выход из цикла происходит при нулевом значении выражения в скобках).

Пример драйвера для функции сцепления строк strcat():

```
#include <stdio.h>
/*************************/
/*Тестирование ф-ции strcat*/
/*************************/
main()
{char str1[81],str2[81];
  puts ("Введите две строки");
  gets (str1);
  gets (str2);
  if (strlen(str1)+strlen(str2) < 81)
        {puts ("Результат:");
         puts (strcat(str1,str2));
```

```
        printf ("Строки после вызова функции сцепления:\n%s\n%s\n", str1,str2);
      }
else puts ("Не хватает памяти для результирующей строки");
getch();
}
```

Примечание. Так как в этой программе вызывается библиотечная функция определения длины строки ***strlen()***, при работе в среде *Borland C++* нужно включить в программу директиву #include <string.h> и изменить имя функции ***srrcat()***, чтобы оно не совпадало с библиотечным.

Библиотечные функции обработки строк

1. strlen - определить длину строки (число символов без завершающего нуль-символа).

Объявление функции: **int strlen(char *s);**

Пример ее вызова:

```
char s[81];
gets(s);
printf ("Длина строки = %d\n", strlen(s));
```

2. strcmp - сравнить две строки.

Объявление: **int strcmp (char *s1, char *s2);**

Значение функции = { 0, если строки одинаковые;
n, разность кодов двух первых несовпадающих символов, если строки разные.

Например, если s1="abcde", s2= "abca12", функция вернет число 3 (‘d’ – ‘a’=100 - 97).

3. strncmp - сравнить первые *n* символов двух строк.

Объявление: **int strncmp (char *s1, char *s2, int n);**

Значение функции = { 0, если первые n символов строк совпадают;
k, разность кодов двух первых несовпавших, символов в противном случае.

Например, если s1="123456", s2= "12789", n=4, функция вернет число -4 (‘3’ – ‘7’ =51 -55). Если же n=2, то результатом будет 0.

4. strcpy - копировать строку *s2* в *s1*.

Объявление: **char *strcpy (char *s1, char *s2);**

Значением функции является *s1* - ссылка на первую строку.

Пример ее вызова:

```
char a[20], b[20];
strcpy (a, “Группа 4251”);
strcpy (b, a); /* копирование строки из массива a в b */
```

5. strncpy - копировать не более *n* символов строки *s2* в *s1*.

Объявление: **char *strncpy (char *s1, char *s2, int n);**

Значением функции является *s1* – ссылка на первую строку.

Например, если s1="1234567", s2= "abcde", n=4, функция вернет адрес строки s1="abcd567". Если задать *n* > длины строки *s2*, то в *s1* скопируется вся строка *s2* (включая нуль-символ) и строки получатся одинаковыми. Так при n=8, s2="abcde", результатом будет s1="abcde".

6. strcat - сцепить две строки *s1* и *s2* (копировать вторую строку в конец первой).

Объявление: **char *strcat (char *s1, char *s2);**

Значением функции является *s1* - ссылка на результирующую строку.

Например, если s1="abc", s2="defgh", то после сцепления строк s1="abcdefgh".

7. strncat - сцепить две строки *s1* и *s2*, причем из второй строки копировать не более *n* символов.

Объявление: **char *strncat (char *s1, char *s2, int n);**

Значением функции является *s1* – ссылка на результирующую строку (*n* символов строки *s2* копируется в конец строки *s1*. Если *n* > длины *s2*, то копируется вся строка).

Например, если s1="abc", s2="12345", n=3, то после сцепления строк s1="abc123".

8. strchr - найти в строке *s* первое вхождение символа *c*.

Объявление: **char *strchr (char *s, char c);**

Значение функции - ссылка на первый символ *c* в строке *s* или *NULL* (пустая ссылка), если символа нет в строке.

Пример вызова функции:

```
char str[81], *p;
gets (str);
p = strchr(str, ‘a’);
if (p == NULL) puts (“В строке нет буквы ‘a’ ”);
else *p= ‘b’; /* замена в строке 1-й буквы a на b */
```

9. strrchr - найти в строке *s* последнее вхождение символа *c*.

Объявление: **char *strrchr (char *s, char c);**

Значение функции - ссылка на последний символ *c* в строке *s* или *NULL* (пустая ссылка), если символа нет в строке.

Пример вызова функции:

```
/* удаление всех пробелов в строке */
char str[81], *p;
. . .
while((p=strrchr(str, ‘ ’))!=NULL)
    strcpy (p, p+1);
```

10. strpbrk - найти в строке *s1* любой из множества символов, входящих в строку *s2*.

Объявление: **char *strpbrk (char *s1, char *s2);**

Значение функции - ссылка на любой символ в строке *s1*, имеющийся в *s2*, или *NULL* (пустая ссылка), если символов из *s2* нет в *s1*.

Пример вызова функции:

```
char str1[81], str2[81], *p;
. . .
p = strpbrk(str1, str2);
```

Например, если str1=”a1d2”, str2=”4321”, то указателю *p* будет присвоен адрес символа ‘1’ в строке *str1* (функция просматривает поочередно символы первой строки и ищет их во второй строке, пока не найдет или не кончится 1-я строка).

11. strstr - найти в строке *s1* первое вхождение строки *s2*.

Объявление: **char *strstr (char *s1, char *s2);**

Значение функции - ссылка на первое вхождение *s2* в *s1* или *NULL* (пустая ссылка), если подстроки *s2* нет в *s1*.

Например, если s1="ab1111234cd1123", s2= "1123", функция вернет адрес третьего символа ‘1’ в строке *s1*, с которого начинается подстрока "1123".

Пример вызова функции:

```
char text []= "Группа 4172", *p;
if ((p = strstr(text, "4172" ))!=NULL)
    strcpy(p, "4272"); /* В результате text = "Группа 4272" */
```

Дополнительные функции обработки строк

12. strrsub - найти в строке *s1* последнее вхождение строки *s2*.

Определение: **char *strrsub (char *s1,char *s2);**

Значение функции - ссылка на последнее вхождение *s2* в *s1* или *NULL* (пустая ссылка), если подстроки *s2* нет в *s1*.

13. delchr - удалить в строке *s* первое вхождение символа *c*.

Определение: **char *delchr (char *s,char c);**

Значением функции является *s* - ссылка на строку.

14. delrchr - удалить в строке *s* последнее вхождение символа *c*.

Определение: **char *delrchr (char *s,char c);**

Значением функции является *s* - ссылка на строку.

15. delnchr - удалить в строке *s n* первых символов.

Определение: **char *delnchr (char *s,int n);**

Значением функции является *s* - ссылка на строку.

16. delchrn - удалить в строке *s* все символы, кроме *n* первых .

Определение: **char *delchrn (char *s,int n);**

Значением функции является *s* - ссылка на строку.

17. delstr - удалить в строке *s1* первое вхождение строки *s2*.

Определение: **char *delstr (char *s1,char *s2);**

Значением функции является *s1* - ссылка на первую строку.

18. chngchar - заменить в строке *s* каждый символ *c1* на символ *c2*.

Определение: **char *chngchar (char *s,char c1,char c2);**

Значением функции является *s* - ссылка на строку.

19. chngstr - заменить в строке s первое вхождение строки *s1* на строку *s2*.

Определение: **char *chngstr (char *s,char *s1,char *s2);**

Значением функции является *s* - ссылка на строку.

Порядок выполнения работы

1. Ознакомьтесь с теоретическим материалом и примером решения задачи (описанием функции ***strcat()*** и драйвером этой функции). Ответьте на контрольные вопросы.
2. Получите задание у преподавателя.
3. Составьте функцию и драйвер, подберите тесты для проверки функции на копьютере.
4. Если аналогичная функция есть в библиотеке Turbo C или Borland C++, выполните тестирование библиотечной функции с помощью вашего драйвера, добавив директиву **#include <string.h>** (разумеется, свою функцию пока включать в программу не нужно). Затем выполните тестирование и отладку своей функции (результаты должны быть такими же, как и при выполнении библиотечной функции).
5. Оформите и сдайте отчет по лабораторной работе.

Лабораторная работа № 8

Графы

Граф - это пара (V,E), где V - конечное непустое множество вершин, а E - множество неупорядоченных пар (u,v) вершин из V, называемых ребрами. Ребро s=(u,v) соединяет вершины u и v. Ребро s и вершина u (а также s и v) называются **инцидентными**, а вершины u и v - **смежными**. Степень вершины равна числу инцидентных ей ребер.

На рис.1а приведен пример графа. В этом графе 7 вершин и 5 ребер.

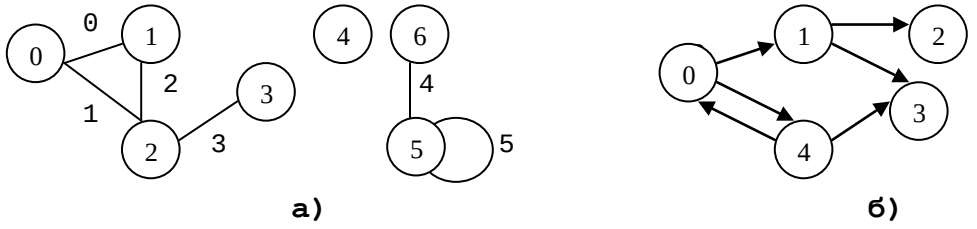


Рис. 1. Примеры графов: а) - неориентированный граф; б) - орграф

Ориентированный граф, или **орграф**, (V,E) отличается от обычного

графа тем, что E - это множество упорядоченных пар (u,v) вершин, называемых дугами. Дуга (u,v) ведет от вершины ***u*** к вершине ***v***. Вершина ***u*** называется **предшественником *v***, а вершина ***v*** - **преемником *u***. Пример орграфа приведен на рис. 1 б.

Графы представляются в программе чаще всего в виде **матрицы смежности** или **матрицы инцидентности**. На рис. 2 приведен вид таких матриц для графа, изображенного на рис. 1 а.

В матрице смежности 1 на пересечении i-й строки и j-го столбца означает, что вершины i и j смежны, а в матрице инцидентности - что вершина i и ребро j инцидентны. Если же ребро j является петлей вершины i, то элемент матрицы инцидентности g2[i][j] = 2.

g1 =

	вершины 0	1	2	3	4	5	6
0	0	1	1	0	0	0	0
1	1	0	1	0	0	0	0
2	1	1	0	1	0	0	0
3	0	0	1	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	1	1
6	0	0	0	0	0	1	0

а)

g2 =

	ребра 0	1	2	3	4	5
0	1	1	0	0	0	0
1	1	0	1	0	0	0
2	0	1	1	1	0	0
3	0	0	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	0	1	2
6	0	0	0	0	1	0

б)

Рис. 2. Примеры внутреннего представления графа:
а) – матрица смежности; **б)** - матрица инцидентности

Для орграфа элемент матрицы смежности g1[i][j] = 1, если есть дуга i –> j.

Описание на Си графа, представленного в виде матрицы смежности:

int g1[NMAX][NMAX];

где NMAX - это константа, задающая максимальное число вершин в графе.

Граф, представленный в виде матрицы инцидентности, можно описать так:

int g2[NMAX][RMAX];

где RMAX - константа, задающая максимальное число ребер в графе (зависит от максимального числа вершин NMAX : RMAX = NMAX * (NMAX - 1) / 2, если

граф без петель).

Здесь ребра пронумерованы, начиная с 0.

Внешнее представление графа может отличаться от внутреннего. Например, граф можно задать в виде количества вершин и последовательности ребер, где каждое ребро - пара смежных вершин:

```
7          количество вершин
0  1
0  2                                   ( пример для графа,
1  2       ребра                         изображенного
2  3                                   на рис. 11.1а  )
5  5
6  5
```

Пример решения задачи

Задача. Задан граф без петель в виде количества вершин n<=7 и матрицы смежности. Сформировать для этого графа матрицу инцидентности.

Программа:

```
#include <stdio.h>
#include <conio.h>
#define   NMAX   7      /* максимальное число вершин графа */
#define   RMAX 21       /* максимальное число ребер        */
 /*-------------------------------------------------------*/
 /*            функция ввода матрицы смежности          */
 /*-------------------------------------------------------*/
 void   VVOD_MATR_SM ( int g1 [NMAX][NMAX] , int      n )
   /*   Входные данные:      n   –   количество вершин       */
        /*   Выходные данные:   g1 – матрица смежности   */
{      int   i,j ;   /* параметры циклов */
       printf ("Введите матрицу смежности:\n\n");
       printf (" ¦ ");
       for (j=0; j<n; j++)   printf ("%d   ",j);
       putchar ('\n');
       for (i=0; i<2*n+2; i++)   putchar ('-');
       for (i=0; i<n; i++)
       { printf ("\n%d¦ ",i);
          for (j=0; j<n; j++)   scanf ("%d", &g1[i][j]);
       }
       putchar ('\n');
```

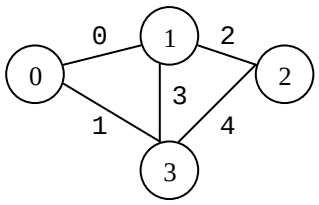
```
}
/*-----------------------------------------------------------*/
/*        функция вывода матрицы инцидентности       */
/*-----------------------------------------------------------*/
void   VIVOD_MATR_IN   ( int g2 [NMAX][RMAX], int      n,   int   k )
  /*   Входные данные:     g2 – матрица смежности ,
   n   –   количество вершин ,
   k   –   количество ребер          */
{   int   i,l ;   /* параметры циклов */
      printf ("Матрица инцидентности\n\n");
      printf ("   ¦   ");
      for (l=0; l<k; l++)   printf ("%3d ", l );
      putchar ('\n');
      for (i=0; i<3*k+2; i++)   putchar ('-');
      for (i=0; i<n; i++)
      {   printf ("\n%d¦ ", i);
           for (l=0; l<k; l++)
  printf ("%3d ",g2[i][l]);
      }
       putchar ('\n');
}
      /*-----------------------------*/
      /*        главная функция        */
      /*-----------------------------*/
void main()
{
      int   g1 [NMAX][NMAX] ,     /* матрица смежности */
                  g2 [NMAX][RMAX] = {0} ,        /* м-ца инцидентности */
                  n ,                    /* количество вершин */
                  k ;                    /* количество ребер    */
  int   i, j; /* индексы элементов матриц g1,g2 */
       printf ("\nВведите количество вершин:");
       scanf ("%d", &n);
      VVOD_MATR_SM (g1, n);     /* ввод матрицы смежности g1 */
          /* Формировавание матрицы инц-ти g2 */
       k=0;
       for (i=0; i<n; i++)
          for (j=i; j<n; j++)
              if (g1[i][j])
             {   g2[i][k]=1;
                  g2[j][k]=1;
                  k++;
             }
       VIVOD_MATR_IN (g2, n, k );     /* вывод м-цы   g2 */
       getch();
```

}

Тесты

1. Исходные данные: Ожидаемый результат:

n=4 матрица смежности матрица инцидентности




```
 ¦ 0 1 2 3
---------
0¦ 0 1 0 1
1¦ 1 0 1 1
2¦ 0 1 0 1
3¦ 1 1 1 0
```

```
 ¦ 0 1 2 3 4
-----------
0¦ 1 1 0 0 0
1¦ 1 0 1 1 0
2¦ 0 0 1 0 1
3¦ 0 1 0 1 1
```

2. Граф, изображенный на рис. 1 а. n=NMAX=7. Вид матрицы смежности приведен на рис. 11.2а. Ожидаемый результат: матрица инцидентности, приведенная на рис. 2 б.

3. Полный граф (все вершины смежны между собой), число вершин максимальное.

Исходные данные:

```
n=7, матрица смежности:
       ¦ 0 1 2 3 4 5 6
      ---------------
      0¦ 0 1 1 1 1 1 1
      1¦ 1 0 1 1 1 1 1
      2¦ 1 1 0 1 1 1 1
      3¦ 1 1 1 0 1 1 1
      4¦ 1 1 1 1 0 1 1
      5¦ 1 1 1 1 1 0 1
      6¦ 1 1 1 1 1 1 0
```

Ожидаемый результат:

```
 матрица инцидентности:
  ¦ 0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
 ---------------------------------------------------------------------
 0¦ 1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1¦ 1  0  0  0  0  0  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0
 2¦ 0  1  0  0  0  0  1  0  0  0  0  1  1  1  1  0  0  0  0  0  0
 3¦ 0  0  1  0  0  0  0  1  0  0  0  1  0  0  0  1  1  1  0  0  0
 4¦ 0  0  0  1  0  0  0  0  1  0  0  0  1  0  0  1  0  0  1  1  0
 5¦ 0  0  0  0  1  0  0  0  0  1  0  0  0  1  0  0  1  0  1  0  1
 6¦ 0  0  0  0  0  1  0  0  0  0  1  0  0  0  1  0  0  1  0  1  1
```

4. В графе нет ребер.

Исходные данные: Ожидаемый результат:

```
n=3     м-ца смежности
           ¦ 0 1 2
          --------
          0¦ 0 0 0
```

```
м-ца инцидентности
           ¦
          --
          0¦
```

```
1¦ 0 0 0                1¦
2¦ 0 0 0                2¦
```

Контрольные вопросы и упражнения

1. Нарисуйте матрицу смежности для орграфа на рис. 1 б. Как по матрице смежности определить преемников и предшественников заданной вершины?

2. Приведите пример неориентированного графа. Нарисуйте матрицу смежности для этого графа. Определите степень каждой его вершины. Как это сделать по матрице смежности?

3. Как по матрице смежности определить, какие вершины имееют петли?

4. Нарисуйте матрицу инцидентности для орграфа на рис. 11.1.б. Как по матрице инцидентности определить степень каждой вершины?

5. Как по матрице инцидентности определить, сколько предшественников и сколько преемников у каждой вершины?

6. Как по матрице инцидентности определить, какие вершины имееют петли?

Задания

1. Задан граф в виде количества вершин $n<=10$ и последовательности ребер (каждое ребро задается парой смежных вершин). Получить матрицу смежности.

а) Напечатать матрицу смежности. Проверить, есть ли в графе петли.

б) Напечатать матрицу смежности. Проверить, есть ли в графе вершины, не смежные с другими.

в) Напечатать для каждой вершины номера смежных вершин.

г) Проверить, есть ли в графе вершина, смежная со всеми другими вершинами.

д) Определить степень каждой вершины графа.

е) Напечатать номера вершин со степенью 1.

ж) Определить степень графа (максимальную степень его вершин).

2. Задан орграф в виде количества вершин $n<=10$ и последовательности дуг (дуга задается парой “предшественник преемник”).

а) Напечатать номера вершин, имеющих более двух преемников.

б) Напечатать номера вершин, не имеющих предшественников.

в) Для каждой вершины напечатать номера всех предшественников.

г) Проверить, есть ли в графе вершины, имеющие только одного преемника.

3. Задан орграф в виде количества вершин n<=10 и матрицы смежности.

а) Напечатать номера вершин, имеющих и предшественников и преемников.

б) Напечатать список дуг орграфа в виде: v1 –> v2, где *v1* – предшественник, *v2* – преемник.

в) Напечатать номер вершины, имеющей наибольшее число преемников.

г) Определить число вершин, соединенных дугами в обоих направлениях.

4. Задан граф в виде количества вершин n<=7, количества ребер k<=28 и матрицы инцидентности.

а) Для каждой вершины напечатать список инцидентных ей ребер.

б) Определить степень каждой вершины графа.

в) Проверить, есть ли вершины со степенью 0.

г) Определить число вершин, инцидентных только одному ребру.

д) Определить наибольшее число смежных между собой ребер, инцидентных одной и той же вершине.

е) Проверить, есть ли в графе петли.

Домашние задания:

1. Решить пример: $y = 3 + \frac{\ln 9}{\sin a} - x$, a и x – произвольные числа.

2. Найти площадь треугольника, зная три стороны a, b и c.

3. Напишите программу вычисления произведения двух натуральных чисел A и B, не используя операцию умножения.

4. Какие числа будут получены при выполнении следующего фрагмента программы?

```
int j;
. . .
for (j = 20; j > 10; j--)
        cout<<j%3;
```

5. Запишите следующий фрагмент программы без помощи оператора for. Дополните определением типов переменных. Нарисуйте схему.

```
for(r = 0, n = 500; n > 0; n--)
```

```
{ cin>>z;
r = r + z*n;
}
```

6. Какое значение примет величина s после выполнения следующего фрагмента

```
int j, n = 5, s = 0;
. . .
for (j = 0; j < n; j++)
s = s + (j + 1)*2;
```

7. Напишите программу получения таблицы из n значений функции:

$$y=\begin{cases} x, & \text{если } x>=5; \\ 2*x, & \text{если } -5<=x<5; \\ x^2, & \text{если } x<-5. \end{cases}$$

Заданы количество значений n (n > 0), начальное значение аргумента функции x_0 и шаг табулирования t.

8. Последовательность неотрицательных вещественных чисел завершается числом -1. Напишите программу нахождения минимального члена последовательности и его номера.

9. Дано целое натуральное число n и числовая последовательность вещественных чисел A_1, A_2, …, A_n. Найти максимум числовой последовательности.

10. Дана последовательность действительных чисел, продолжающаяся до конца файла. Определить, образуют ли заданные числа возрастающую последовательность.

11. Дана последовательность действительных чисел, продолжающаяся до конца файла. Найти количество чисел в наиболее длинной последовательности подряд идущих положительных чисел.

12. Запишите оператор многовариантного ветвления для проверки значения целочисленной величины n. При совпадении с одним из возможных значений: 2, 3, 4 или 5, выводите соответствующий текст - «Неудовлетворительно», «Удовлетворительно», «Хорошо», «Отлично», в противном случае - сообщение «Результат не определен».

13. Дан текст, завершающийся символом точка с запятой ;. Напишите программу для решения следующей задачи. Заменить в исходном тексте каждое сочетание символов != на сочетание символов ==. Подсчитать количество замен.

14. Дан текст, продолжающийся до конца файла. Напишите программу подсчета количества слов, длиной более 10 символов. Словом считается любая последовательность символов, не содержащая пробелов, символов табуляции и новой строки.

15. Дано натуральное число N>0 и массив A из N различных целых чисел. Напишите программу для решения следующей задачи: переставить местами наибольший и наименьший элементы заданного массива и напечатать полученный массив.

16. Дан текст, произвольной длины, продолжающийся до конца файла. Напишите программу для решения следующей задачи. Определить, сколько раз каждый символ латинского алфавита входит в текст.

17. Нарисуйте схему заданного фрагмента, используя базовые алгоритмические структуры. Выделите базовые алгоритмические структуры на схеме (например, штриховой линией) и подпишите их названия.

```
float x, max;
int k, i, nom;
cin>>k;
if (k < 1) cout<<”\nВходная последовательность пуста\n”;
else
```

```
{cin>>max;
        nom=0;
        for (i = 1; i<k; i++)
                { cin>>x;
                if (x > max)
                        {max = x;
                        nom= i;
                        }
                }
        cout<<"\nМаксимум= "<<max<<"   номер = \n"<<nom;
}
```

18. Перепишите заданный фрагмент, заменив оператор цикла do while, оператором цикла while.

```
int   n, s=0;
cin>>n;
do
    { s+= 5;
       n--;
    }
while ( n>0 );
```

19. Дано целое N ≥ 0 и массив из N вещественных чисел. Разработайте методом структурного программирования сверху вниз и напишите программу для сортировки массива по не убыванию (чтобы каждый элемент был не меньше предыдущего элемента) следующим способом.

Сортировка вставкой. Каждый элемент, начиная со второго, вставлять в нужное место среди предыдущих элементов, увеличивая, таким образом, на единицу длину уже упорядоченной последовательности, пока она не охватит весь массив.

20. Дано целое N ≥ 0 и массив из N целых чисел. Напишите программу для сортировки массива по убыванию следующим способом.

Сортировка подсчетом индекса. Для каждого элемента подсчитать количество меньших или равных ему элементов, которое и определит его индекс в упорядоченном массиве.

21. Дано целое N ≥ 0 и массив из N целых чисел. Напишите программу для сортировки массива по возрастанию следующим способом.

Сортировка подсчетом количества значений. Подсчитать в массиве количество экземпляров каждого возможного значения (все значения могут быть целыми числами в диапазоне от A до B). Заполнить массив заново в требуемом порядке нужными значениями в подсчитанных количествах.

22. Дано целое n>0 и последовательность вещественных чисел $x_1, x_2,\ldots,x_n$. Написать программу для выдачи на экран только положительных чисел последовательности.

23. Дано целое n>0 и последовательность вещественных чисел x_1, x_2,…,x_n. Написать программу для выдачи на экран сначала всех отрицательных, а потом положительных чисел последовательности.

24. Найти сумму элементов двумерного квадратного массива, расположенных под побочной диагональю.

25. **Дан массив из n x m элементов. Найти индексы первого наименьшего элемента массива.**

26. Дан квадратный массив из n x n **элементов. Найти сумму элементов побочной диагонали.**
27. Дан квадратный массив из n x n **элементов. Найти сумму элементов ниже главной диагонали.**

28. Задан текст, состоящий из m строк. В каждой строке заменить сочетание символов AB на два пробела. Работу с каждой строкой оформить как подпрограмму.
29. Задан текст, состоящий из n строк. В каждой строке подсчитать количество заглавных латинских букв и количество строчных латинских. Работу с каждой строкой оформить как подпрограмму.

30. Заданы два массива целых чисел. В каждом из них найти сумму и количество элементов, кратных пяти. Подсчет суммы и количества элементов оформить как подпрограмму.