Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ»

А.Р. БИКМУРЗИНА

ОСНОВЫ ПРОГРАММИРОВАНИЯ

Учебно-методическое пособие

Рекомендовано к изданию Учебно-методическим управлением КНИТУ-КАИ

Казань 2018

УДК 681.3.06
БИК 60

Рецензенты: доктор физико-математических наук, профессор А.П. Кирпичников (Казанский национальный исследовательский технологический университет);

доктор технических наук, профессор Р.Н. Гайнуллин (Казанский национальный исследовательский технологический университет).

Бикмурзина А.Р.
БИК 60 Основы программирования: Учебно-методическое пособие. Казань: Изд-во Казан. гос. техн. ун-та, 2018. 55 с.

ISBN 5-7579

Учебно-методическое пособие содержит комплекс лабораторных работ по разделу «Основы программирования» дисциплин «Основы информатики и программирования» и «Основы алгоритмизации и программирования». Рассмотрены вопросы разработки алгоритмов и программ на языках С и С++ для широкого круга задач: числовой и символьной обработки, работы с массивами. Содержится описание языка программирования и работы в программной среде Microsoft Visual Studio 2017, примеры алгоритмов и программ, методические указания и задания для выполнения лабораторных работ, контрольные вопросы. Предназначено для студентов направления 09.03.02 “Информационные системы и технологии”.

Ил. 5. Библиогр.: 5 назв.

УДК 681.3.06

ISBN 5-7579-0391-0

Введение

Данное учебно-методическое пособие содержит необходимый теоретический материал по программированию на языках *C/C++* и решению типовых задач дисциплины «Основы информатики и программирования», а также методические указания к лабораторным работам.

На первом занятии студенты знакомятся с работой в среде *Microsoft Visual Studio 2017* и с готовыми программами на языке *C(C++)*. На каждом из следующих занятий необходимо решить одну или несколько задач. По каждой лабораторной работе оформляется отчет, содержащий условие задачи, схему алгоритма решения задачи, текст программы на языке *C* (*C++*) и результаты тестирования программы.

Пособие может быть использовано для лабораторных работ по дисциплине «Основы алгоритмизации и программирования» и для самостоятельного освоения студентами курса программирования на языках *C* и *C++*. Вопросы объектно-ориентированного программирования на *C++* не рассматриваются.

ЛАБОРАТОРНАЯ РАБОТА №1

РАБОТА В СИСТЕМЕ ПРОГРАММИРОВАНИЯ Visual Studio 2017

Системы программирования

Система программирования — это инструментальная система для разработки новых программ на конкретном языке программирования. Системы программирования поддерживают все этапы процесса программирования, тестирования и отладки создаваемых программ.

Современные системы программирования предоставляют пользователям мощные и удобные средства разработки программ. В них входят:

- интегрированная среда разработки;
- средства создания и редактирования текстов программ;
- трансляторы (компиляторы и интерпретаторы);
- библиотеки стандартных программ и функций;
- редактор связей (компоновщик);
- загрузчик;
- отладочные средства, помогающие находить и устранять ошибки в программе;
- "дружественная" к пользователю диалоговая среда;
- многооконный режим работы;
- мощные графические библиотеки;
- визуальные средства автоматизации создания программного кода;
- другие специфические особенности.

Трансляторы предназначены для преобразования программ, написанных на языках программирования, в программы на машинном языке.

Трансляторы делятся на два класса: компиляторы и интерпретаторы. **Компилятор** переводит всю исходную программу на машинный язык без непосредственного ее выполнения. **Интерпретатор** последовательно переводит на машинный язык и сразу же выполняет каждый оператор

исходной программы. Для большинства языков программирования используются компиляторы.

Программа, подготовленная на каком-либо языке программирования, называется исходным модулем. Компиляторы из исходных модулей формируют объектные модули (файлы с расширением **.obj**), являющиеся входной информацией для **редактора связей**. Объектный модуль содержит текст программы на машинном языке и дополнительную информацию, обеспечивающую настройку модуля по месту его загрузки и объединение этого модуля с другими независимо откомпилированными модулями в единую программу.

Компоновщик, или **редактор связей** – это программа, редактирующая и объединяющая объектные модули в единые загрузочные, готовые к выполнению программные модули. Загрузочный модуль (файл с расширением **.exe**) может быть помещен с помощью **загрузчика** в оперативную память и выполнен либо из интегрированной среды разработки, либо вне системы программирования.

Отладчик позволяет управлять процессом исполнения программы, является инструментом для поиска и исправления ошибок в программе. Базовый набор функций отладчика включает:

- пошаговое выполнение программы (режим трассировки) с отображением результатов,
- остановка в заранее определенных точках,
- возможность остановки в некотором месте программы при выполнении некоторого условия;
- изображение и изменение значений переменных.

Системы программирования Microsoft Visual Studio

Microsoft Visual Studio - системы программирования компании Microsoft, позволяющие создавать программы на разных языках программирования (C++, Visual Basic, C# и др.). Системы включают интегрированную среду разработки программ и ряд других инструментальных средств, в том числе визуальные средства автоматизации создания программного кода. Данные системы позволяют разрабатывать

как консольные приложения, так и приложения с графическим интерфейсом, в том числе Windows-приложения, а также веб-сайты, веб-приложения.

Работа в интегрированной среде ***Visual Studio 2017***

Чтобы ввести и выполнить программу на языке C++ в среде ***Visual Studio 2017***, нужно создать проект. В проекте хранится информация обо всех файлах, используемых для создания приложения, пути к этим файлам и способы их компиляции.

После запуска *Visual Studio 2017* выберите в меню ***Файл*** команду ***Создать,*** далее ***Проект***. Появится окно с именем ***Создание проекта*** (рис. 1.2). В этом окне выберите слева язык программирования ***Visual C++***, в центре - шаблон проекта ***Консольное приложение Windows,*** в поле ***Расположение:*** укажите папку, где будет размещен проект, а в поле ***Имя:*** введите имя проекта, например, *lab1*, и нажмите кнопку **OK**.

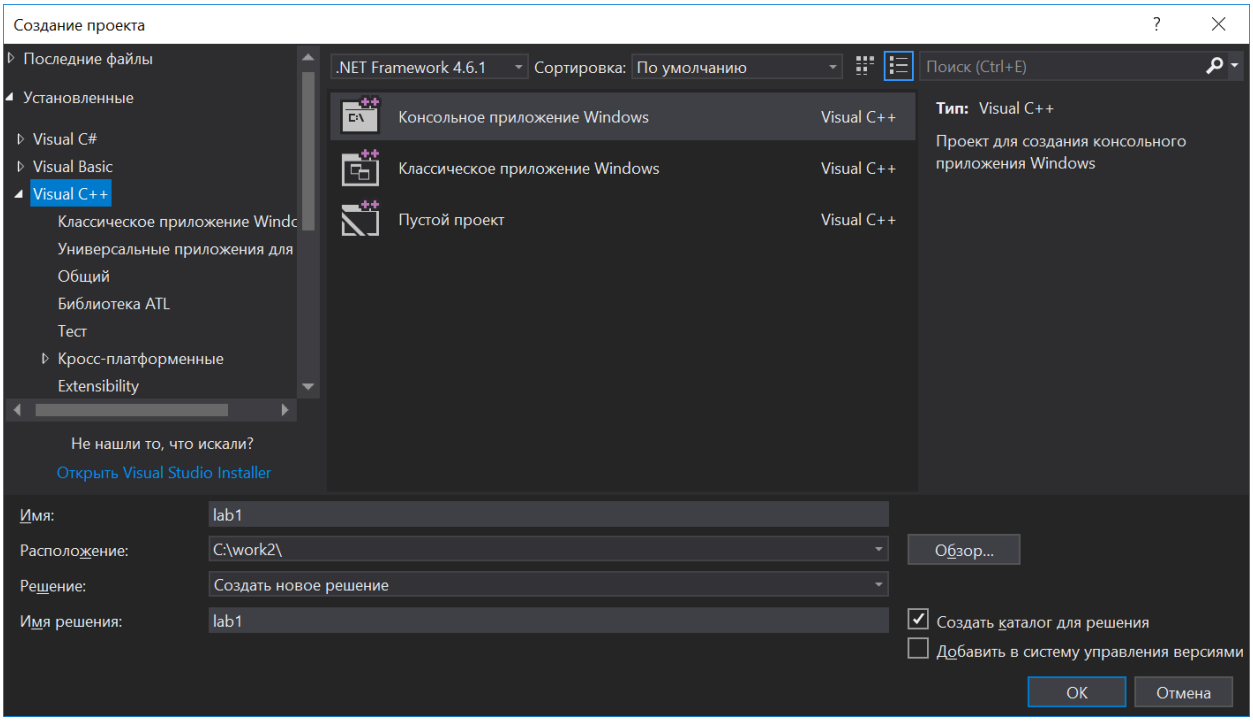


Рис.1.2. Окно «Создание проекта».

В результате будет создан проект ***lab1***, содержащий файл ***lab1.cpp*** с текстом программы:

```
// lab1.cpp: определяет точку входа для консольного приложения.
//
```

```
#include "stdafx.h"

int main()
{
        return 0;
}
```

Отредактируйте программу, оставив обязательно первую директиву *#include "stdafx.h".*

Для компиляции программы выберите в меню ***Сборка*** пункт ***Собрать решение***. Для запуска программы в пункте меню ***Отладка*** выберите ***Запуск без отладки*** или нажмите одновременно клавиши ***Ctrl*** и ***F5.*** При каждом запуске программы происходит автоматическое сохранение текста программы в файле *lab1.cpp* .

Пример простой программы на языке С (С++)

```
            /* Программа  1.1. Вычисление площади треугольника
                    по трем заданным сторонам  */
#include "stdafx.h"
#include <stdio.h>
#include <math.h>
#include <locale.h>

int   main ()
{       float a, b, c;        // стороны треугольника
        float  p, s;          // полупериметр и площадь треугольника
        setlocale(LC_ALL, "Rus"); // для вывода кириллицы
        printf ("\n Введите значения сторон треугольника: ");
        scanf_s ("%f %f %f", &a, &b, &c); // ввод значений a, b, c
        if (a>= b+c || b>= a+c || c>=a+b)
                printf ("Неверные данные");
        else
        {       p = (a+b+c)/2;
                s = sqrt (p* (p-a) * (p-b) * (p-c));
                printf ("Площадь треугольника = %f", s);
        }
       return 0;
}
```

Пример результатов выполнения программы

```
Введите значения сторон треугольника: 1   2   3
Неверные данные
Введите значения сторон треугольника: 4.5   5   6
Площадь треугольника = 11.009761
```

Пояснения к программе

1. Текст, ограниченный с двух сторон символами /* и */, − это комментарий (многострочный). Текст, начинающийся с символов //, - это тоже комментарий (однострочный).

2. Директивы ***#include*** служат для включения в программу необходимых библиотечных файлов. Файл ***stdio.h*** содержит объявления функций ввода/вывода (*scanf, printf, ...*). Файл ***math.h*** содержит математические функции (*sqrt, sin, cos, exp, ...*). В файле ***locale.h*** содержится вызываемая в программе библиотечная функция *setlocale*, которая служит для нормального вывода русского текста (для перекодировки кириллицы).

3. ***int main()*** **–** это заголовок главной функции программы. Тело функции заключено в фигурные скобки.

4. В теле функции вначале объявлены все переменные вещественного типа (***float***).

5. Оператор ***printf*** выводит на экран сообщение. Символ ***\n*** – это управляющий символ перевода строки.

6. Оператор ***scanf_s*** читает значения вводимых данных, преобразует их в тип ***float*** в соответствии с указанными форматами (***%f***) и присваивает их переменным ***a,b,c***, адреса которых передаются функции *scanf_s* (символ & означает адрес).

7. В операторе ***if*** проверяется корректность введенных данных. Если хотя бы одно из значений больше или равно суммы двух других, выводится сообщение об ошибке. Иначе вычисляется полупериметр и площадь треугольника по формуле Герона:

$$s = \sqrt{p(p-a)(p-b)(p-c)}$$

и результат выводится на экран.

8. Оператор возврата ***return*** завершает выполнение программы, возвращая в операционную систему код успешного завершения 0.

Примечание. Файл *stdafx.h*, который содержится в этом же проекте, служит для включения в программу необходимых библиотечных заголовочных файлов. На рис.1.3 представлено окно проекта, где слева указано первоначальное содержимое файла *stdafx.h*, а справа все файлы проекта. Рекомендуется все директивы #include с именами библиотечных файлов поместить в этот файл. Попробуйте переместить эти директивы из файла *lab1.cpp* в файл *stdafx.h* и снова выполнить программу.

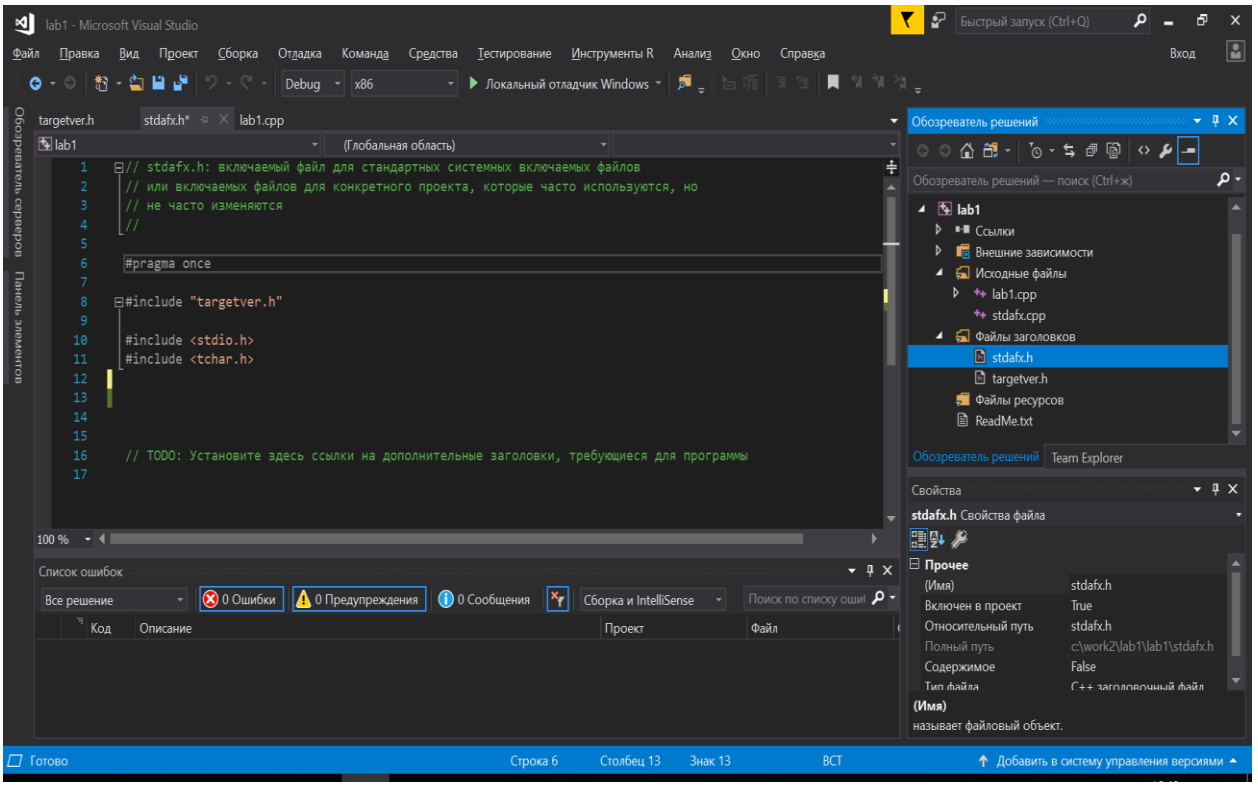


Рис. 1.3. Окно проекта

Задание

1. Запустите систему **Visual Studio 2017.** Создайте новый проект. Введите текст приведенной выше программы 1.1.

2. Выполните программу несколько раз с разными данными, в том числе и неверными.

3. Попробуйте внести в программу синтаксические ошибки. Выполните компиляцию программы и посмотрите, как выглядят сообщения об ошибках.

4. Добавьте в программу вывод периметра треугольника. Проверьте результат.

5. Добавьте в программу проверку, является ли треугольник равносторонним или равнобедренным (с выводом соответствующего сообщения). Проверьте результат работы программы с разными тестами.

6. Переместите все директивы *#include*, кроме первой *#include "stdafx.h"*, из файла *lab1.cpp* в файл *stdafx.h* и снова выполните программу.

ЛАБОРАТОРНАЯ РАБОТА №2

ПРОГРАММИРОВАНИЕ ПРОСТЕЙШИХ ЦИКЛОВ НА ЯЗЫКЕ С(С++)

Структура программы

Любая программа на языке *С* состоит из одной или более "функций", являющихся основными компонентами программы. Одна из функций, с которой начинается выполнение программы, называется главной и всегда носит имя ***main***. Остальные функции – это подпрограммы, которые могут вызываться либо из главной функции, либо из других подпрограмм. Простая программа, состоящая только из функции ***main***, имеет следующую структуру:

```
Директивы препроцессора
int  main ( )
{  Описания переменных
   Операторы
   return  0;
}
```

Заголовок функции – ***int main().*** Круглые скобки после имени ***main*** как раз и указывают, что это функция (без параметров). Тип ***int*** указывает, что функция должна вернуть в операционную систему целочисленный код завершения программы. Считается, что при успешном завершении программа должна вернуть число 0, поэтому последний оператор - ***return 0;***

Тело функции заключается в фигурные скобки и состоит из объявлений 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). Комментарии такого вида могут занимать несколько строк. Можно использовать и однострочные комментарии, начинающиеся с пары символов //.

Пример простой программы на языке *C (C++)*:

```
/* Программа 2.1. Сложение двух целых чисел */
#include <stdio.h>
int   main ()
{   int a, b;          // объявление 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
    return 0;
}
```

При выполнении этой программы на экране появится сообщение:

Задайте два числа:

и затем программа будет ждать, пока вы не введете числа (ввести можно 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 объявления 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 C (C++) пишутся строчными буквами, директивы препроцессора начинаются с символа #. Для наглядности принята ступенчатая форма записи программы.

Примечание. При выполнении программы в системе Visual Studio в программу следует добавить директивы ***#include <stdafx.h>*** и

#include **<*locale.h*>** , а перед первым оператором *printf()* вызов функции ***setlocale(LC_ALL, "Rus");*** для перекодировки выводимого русского текста.

Объявление переменных и основные типы данных

При объявлении (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. Можно использовать в имени символы подчеркивания вместо пробелов, когда имя состоит из нескольких слов. 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 типы (***int, short, long, unsigned***), символьный тип (***char***) и вещественные типы или типы 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 компьютера, операционной системы и конкретной реализации языка С.

При описании переменной можно инициализировать переменную, например:

```
int  k = 0;   /* k присваивается начальное значение 0 */
char  s = 'a';
```

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.

Можно объявить именованную константу иначе:

const float PI = 3.14159;

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;
```

Выражение может 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 тоже целый.

Библиотечные математические функции (объявлены в файле ***math.h***):

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) – число ***e*** ≈2.7 возводится в степень *x*;
log(x) – вычисляется натуральный логарифм *x*;
log10(x) – вычисляется десятичный логарифм *x*;

sqrt(x) – вычисляется $\sqrt{x}$;
pow(x,y) – *x* возводится в степень *y*.

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

При использовании указанных функций в программу нужно включить директиву ***#include <math.h>.*** Если этого не сделать, компилятор не сможет проверить правильность вызова функций и выполнить необходимые преобразования типов, вследствие чего вы можете получить неверные результаты.

Оператор-выражение

В языке С 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 = a+2;   // или   a += 2;
```

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 вывода ***printf()*** и ввода ***scanf()*** и ***scanf_s().*** 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 2.1. (из раздела «Структура программы»).

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, %i – 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 значение *x* = 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 задать точность, например:

```
printf(" %.1f  ", x);
printf(" %.4e ", x);
```

Результат:

```
123.7     1.2368e+02
```

Использование функций *scanf()* и *scanf_s()*

Функции *scanf()* и *scanf_s()* предназначены для ввода значений переменных с клавиатуры во время выполнения программы. При программировании в среде *Visual Studio* следует использовать функцию

scanf_s(), функция *scanf()* считается устаревшей, и компилятор выдает предупреждение об этом (хотя программа выполняется и с функцией *scanf*).

Список аргументов этих функций почти такой же, как у функции *printf().* Первый аргумент – это форматная строка, содержащая форматы ввода значений переменных. Сами переменные, точнее адреса переменных, 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.

Функции *scanf()* и *scanf_s()* используют практически тот же набор форматов, что и функция *printf().* Основные отличия от *printf()* 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*.

Ввод/вывод в языке С++

В языке С++ для ввода/вывода можно использовать другие операторы. Посмотрите следующий пример.

```
/* Программа 2.2. Сложение двух целых чисел */
#include <iostream.h>
int  main ()
{  int a, b;          // объявление целочисленных переменных a и b
   cout << "Задайте два числа: ";    // вывод сообщения
   cin >> a >> b;                    // ввод значений a и b
   cout << a<<" + "<< b<< " = "<< a+b;    // вывод результата
   return 0;
}
```

Пояснения к программе:

iostream.h – заголовочный файл, содержит объявления стандартных объектов-потоков языка *С++*;

cin – поток для ввода с клавиатуры, >> – операция для чтения данных из потока;

cout – поток для вывода на экран, << –операция для вывода данных в поток.

Примечание. В программах на С++ в *Visual Studio* есть особенности по сравнению с другими системами программирования. Это касается директив препроцессора. Вместо *#include <iostream.h>* нужно указать

```
#include <iostream>
using namespace  std;
```

Составной оператор

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

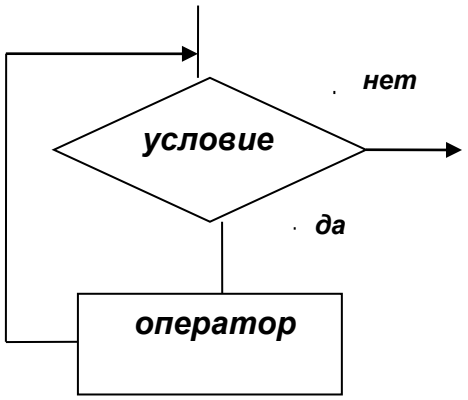
Пример:

```
{ 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. Если в цикле должны выполняться несколько операторов, заключите их в фигурные скобки.

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

> больше,
>= больше или равно,
< меньше,
<= меньше или равно,
== равно,
!= не равно.

Выражение может включать в себя также логические операции:

&& (И – логическое умножение),
|| (ИЛИ – логическое сложение),
! (НЕ –отрицание).

Пример оператора:

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

Пример решения задачи

Задача. Дано действительное число ***x***. Вычислить значение **sin *x*** с помощью ряда:

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

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

Если при вычислении очередного члена ряда отдельно вычислять числитель и знаменатель, где значения могут быть очень большие, то получится слишком большая погрешность. Поэтому выведем рекуррентную формулу вычисления очередного члена ряда через предыдущий член.

Обозначим очередной член данного ряда через a_n .

$a_0 = x$;

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

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

. . .

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

Поскольку для вычисления очередного члена ряда нужно знать только значение предыдущего члена, а все суммируемые значения хранить нет необходимости, то массив не нужен. Достаточно одной простой переменной для вычисления очередного члена ряда. Назовем ее ***a.***

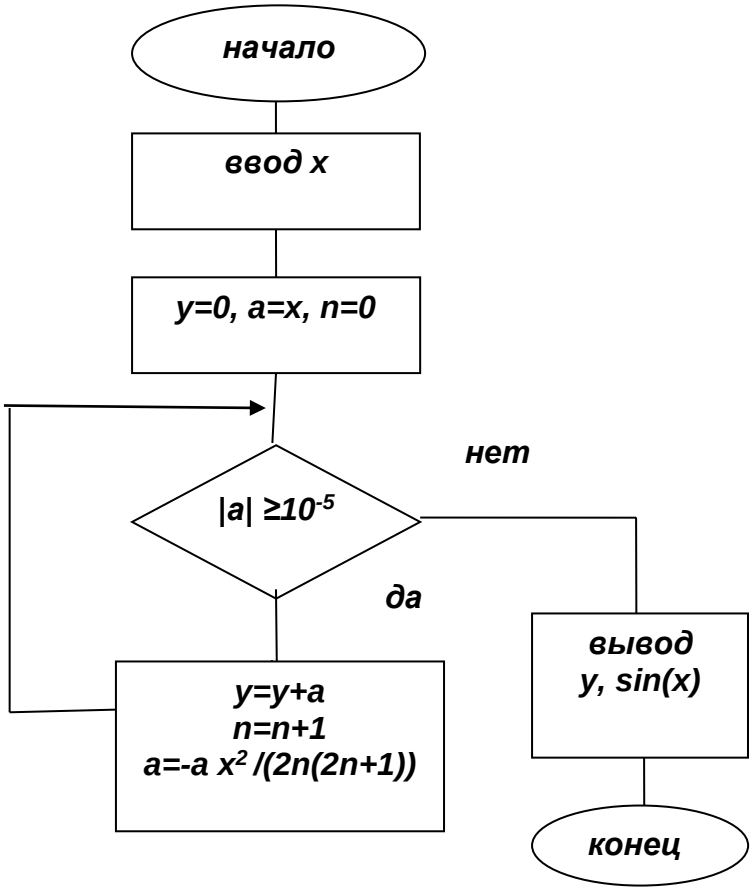


Рис. 2.1. Схема алгоритма вычисления $y=\sin x$

```
   /* Программа 2.2. Приближенное вычисление  y=sin x  */
#include <stdio.h>
#include <math.h>
#define   E  1e-5      // точность вычисления

int  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));       /* вычисление очередного члена
                                         ряда  через предыдущий   */
    }
    printf ("y=%f\n", y);
    printf ("sinx=%f\n", sin(x));
    return 0;
}
```

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

Контрольные вопросы и упражнения

1. Для чего служит директива препроцессора *#include <stdio.h>*?
2. Какое имя должна иметь главная функция программы?
3. Как описать целочисленную переменную ***n***?
4. Выберите правильные варианты описания вещественной переменной ***x***:

а) int x;
б) float x;
в) char x;
г) double x;
д) long x;

5. Для чего служит директива препроцессора *#define*?

6. Как можно иначе записать оператор *n = n+1;* ?

7. Какое значение будет иметь переменная ***k*** после выполнения следующих операций?

```
int  k=0;
k--;
```

8. Как ввести с клавиатуры значение переменной ***n*** типа *int* ? Какой из ответов верный:

а) *scanf (n);*
б) *scanf (“%d”, n);*
в) *scanf (“%d”, &n);*
г) *scanf (“%f”, n);*
д) *scanf (“%f”, &n);*

9. Для чего служит функция *printf()*? Определите результат выполнения фрагмента программы:

```
int  x=-2;
printf (“%d \n %d”, x*x, abs(x));
```

10. Как записывается и для чего нужен составной оператор?

11. Как изображается и программируется на языке С циклическая структура с предусловием?

12. Определите результат выполнения фрагмента программы:

```
int  s = 0, n = 1;
while ( n < 5)
{  s = s + n;  n++; }
printf (“Результат:%d”, s);
```

Порядок выполнения работы

1. Познакомьтесь с описанием языка С (С++) и примером решения задачи. Ответьте на контрольные вопросы.

2. Получите у преподавателя индивидуальное задание.

3. Составьте блок-схему и программу на языке С (С++) и подберите тесты для проверки программы на компьютере.

4. Отладьте программу на компьютере и покажите результаты тестирования преподавателю.

5. Оформите и сдайте отчет по лабораторной работе преподавателю.

Задания

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

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

б) $y = e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots$;

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

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

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

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

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

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

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

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

2. Дано натуральное число ***n***. Проверить справедливость равенства:

а) $1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$;

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

в) $1 + 2 + 2^2 + \ldots + 2^{n-1} = 2^n - 1$;

г) $1^3 + 2^3 + 3^3 + \ldots + n^3 = (1 + 2 + 3 + \ldots + n)^2$.

3. Дано натуральное число ***n***. Определить:

а) количество цифр в числе ***n***;

б) сумму его цифр;

в) первую цифру числа ***n***.

ЛАБОРАТОРНАЯ РАБОТА №3

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 с 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.

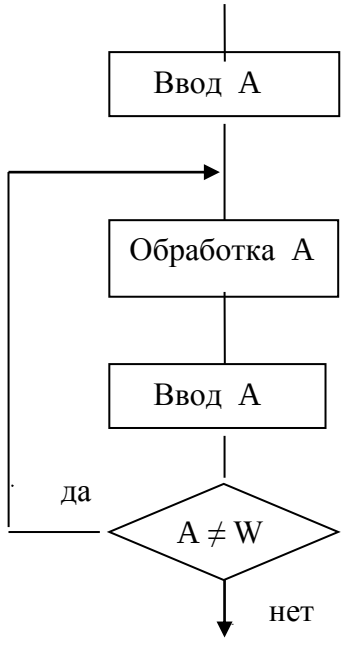
Числовая последовательность может задаваться с указанием количества чисел или иметь какой-то признак конца.

Пусть последовательность задается в виде

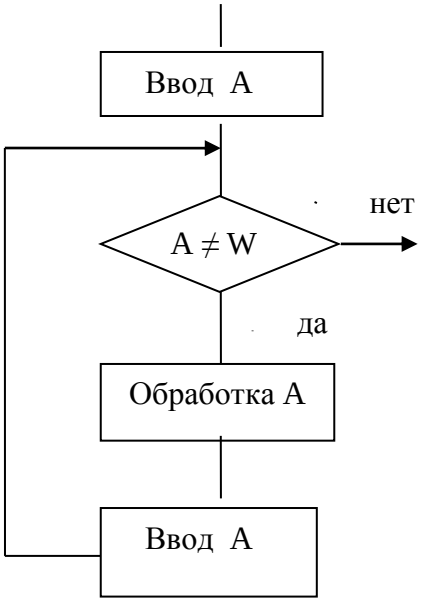
$$A_1, A_2, \dots, A_n, W$$

(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 языке С (С++), соответствующие этим схемам:

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

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

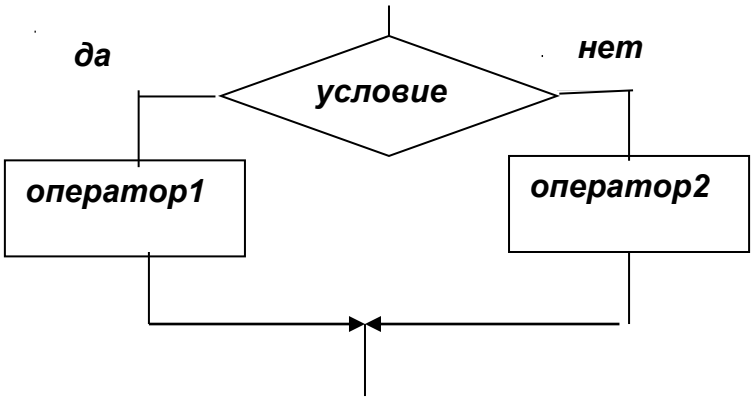
В этих фрагментах предполагается, что переменная ***a*** вещественного типа (*float*), поэтому указан формат ***%f***. Если же последовательность состоит из целых чисел типа *int*, то следует выбрать формат ***%i*** или ***%d***.

W – символическая константа, которая должна быть определена с помощью директивы ***#define***.

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

Структура ветвления

```
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 языке С значение "ложь" – это 0, а "истина" – любое ненулевое число. Поэтому предыдущий оператор if можно записать иначе:

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

Пример решения задачи

Задача. Даны вещественные числа A_1, A_2, A_3, … Признак конца последовательности 0. Определить сумму положительных чисел и сумму отрицательных чисел последовательности.

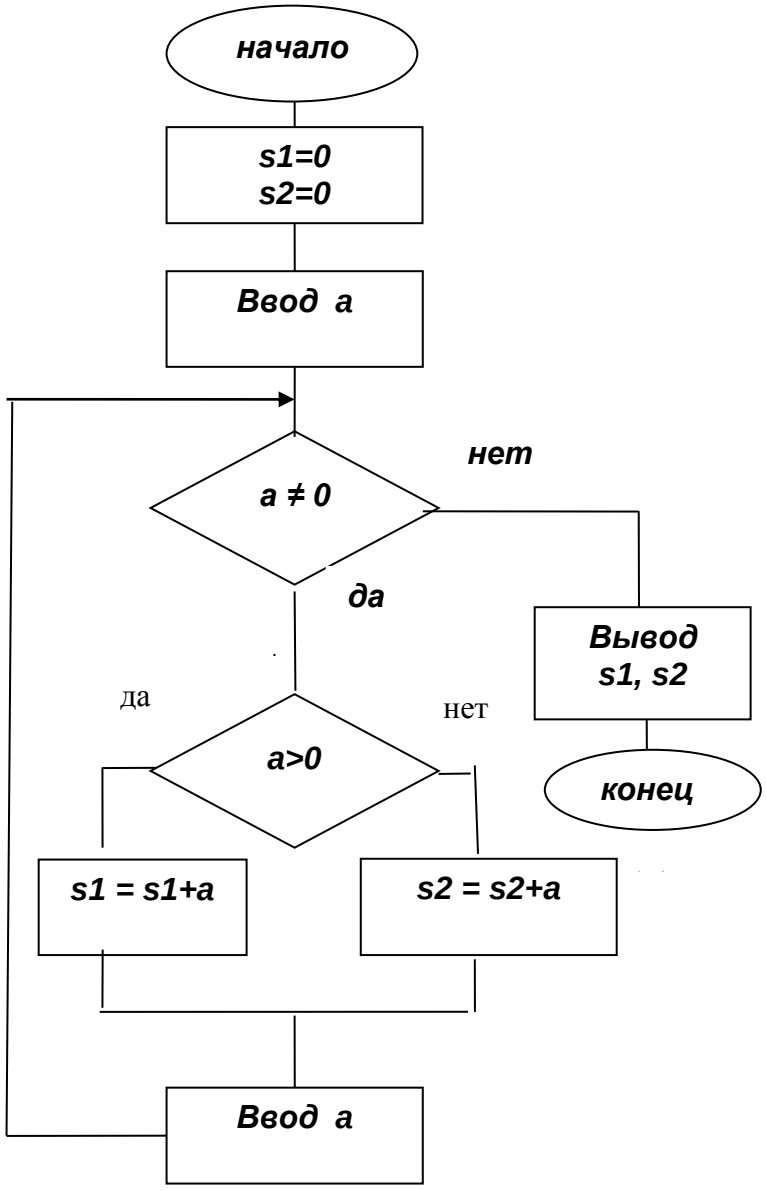


Рис. 3.1. Схема алгоритма решения задачи

// Программа 3.1

```
#include <stdio.h>
int   main()
{
      float  a,       // очередное число   последовательности
           s1=0,    // сумма  положительных чисел
           s2=0;    // сумма отрицательных чисел

      printf ("Введите числовую  последовательность, заканч. 0\n");
      scanf ("%f", &a);
      while ( a != 0)
      {
           if (a>0)  s1 = s1+a;
           else   s2 = s2+a;
           scanf ("%f", &a);
      }
      printf ("Сумма положительных чисел = %.2f\n", s1);
      printf ("Сумма отрицательных чисел = %.2f\n", s2);
      return  0;
}
```

Пример результата выполнения программы:

Введите числовую последовательность, заканч. 0

2.5 10 -6 4.8 -5.25 -1 0

Сумма положительных чисел = 17.30

Сумма отрицательных чисел = -12.25

Контрольные вопросы и упражнения

1. Какой схемой можно представить процесс обработки числовой последовательности, заканчивающейся заданным числом?
2. Как записать фрагмент программы, соответствующий этой схеме?
3. Каким оператором программируется цикл с постусловием?
4. Каким оператором программируется структура ветвления?
5. Определите результат выполнения фрагмента программы:

```
int  x=3, y=5, z=-1;
if  (x>y && x>z)  printf (“%d”, x);
else   if (z>y)   printf (“%d”, z);
          else  printf (“%d”, y);
```

6. Что нужно изменить в программе 3.1, если требуется определить не

сумму, а количество положительных и количество отрицательных чисел заданной последовательности?

Порядок выполнения работы

1. Ознакомьтесь с теоретическим материалом и примером решения задачи. Ответьте на контрольные вопросы.
2. Получите у преподавателя индивидуальное задание.
3. Составьте блок-схему и программу на языке С (С++) и подберите тесты для проверки программы на компьютере.
4. Отладьте программу на компьютере и покажите результаты тестирования преподавателю.
5. Оформите и сдайте отчет по лабораторной работе преподавателю.

Задания

1. Даны натуральные числа $A_1, A_2, A_3, \ldots$. Признак конца последовательности 0. Определить:

а) количество и сумму тех чисел последовательности, которые делятся на 5 и не делятся на 7;

б) наибольшее число последовательности и его порядковый номер;

в) среднее арифметическое четных чисел последовательности;

г) количество четных чисел и количество нечетных чисел в последовательности;

д) $\min (A_1 + A_2 , A_2 + A_3 , \ldots)$;

е) $(A_2 - A_1)*(A_3 - A_2)* \ \ldots \ * (A_n - A_{n-1})$;

ж) упорядочены ли числа последовательности по убыванию;

з) есть ли в последовательности одинаковые соседние числа.

2. Даны вещественные числа $A_1, A_2, A_3, \ldots$. Признак конца последовательности число 999. Определить:

а) составляют ли числа возрастающую последовательность;

б) среднее арифметическое всех чисел;

в) сумму и количество положительных и произведение отрицательных чисел последовательности;

г) разность между наибольшим числом и наименьшим числом последовательности;

д) сколько раз встречается в последовательности наибольшее число.

ЛАБОРАТОРНАЯ РАБОТА №4

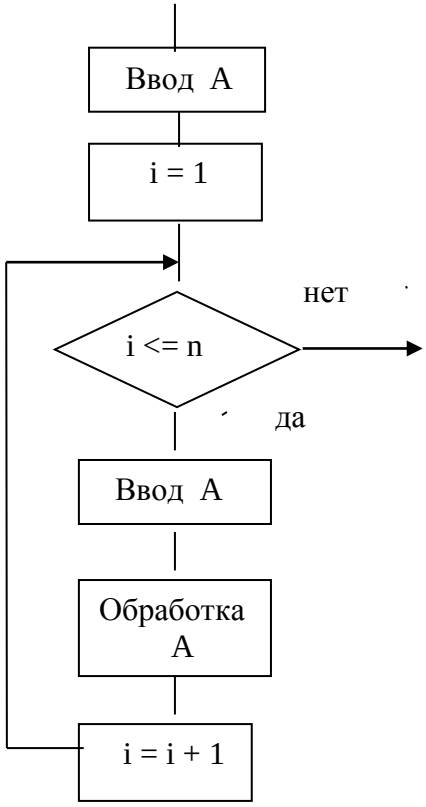
ОБРАБОТКА ЧИСЛОВЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

Числовая последовательность может быть задана с указанием количества вводимых чисел:

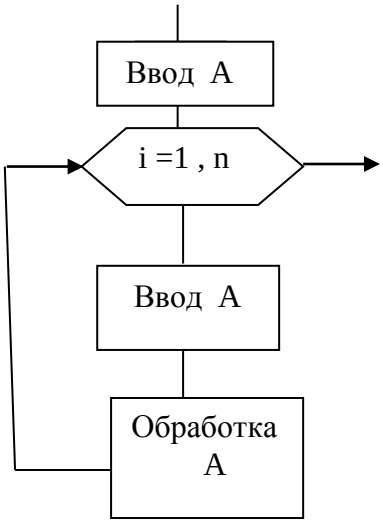
$$n, A_1, A_2, ..., A_n .$$

Тогда процесс обработки можно представить в виде следующих схем:

а) полная форма



б) более короткая форма



На языке C++ этот процесс можно записать с помощью оператора цикла ***while*** или лучше оператора цикла ***for***:

а)
```
cin >> n;  i=1;
while (i<=n)
{
   cin >> a;
   /* обработка a */
     . . .
   i++;
}
```

б)
```
cin >> n;
for (i=1; i<=n; i++)
{
     cin >> a;
     /* обработка a */
          . . .
}
```

Пример решения задачи

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

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

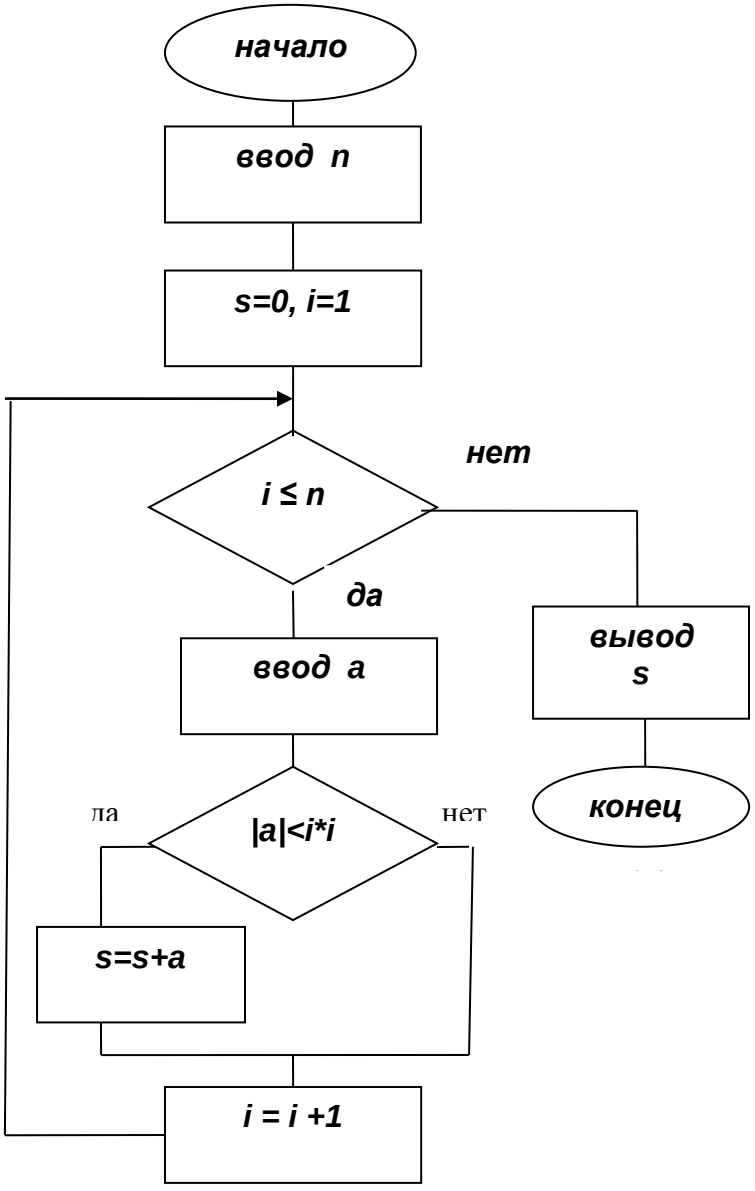


Рис. 4.1. Схема алгоритма решения задачи

// Программа 4.1 (C++)

```
#include <iostream.h>
#include <math.h>
void   main()
{
      int  n;          /* количество чисел  */
      int  a,          /* очередное число   */
           s=0,        /* сумма             */
           i;          /* порядковый номер числа в посл-ти */

      cout << "Введите количество чисел: ";
      cin >> n;
      cout << "Введите числовую  последовательность:\n";
      for (i=1; i<=n; i++)
      {
          cin >> a;
          if (abs(a) < i*i)  s=s+a;
      }
      cout  << "сумма=" << s << '\n';
}
```

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

Контрольные вопросы и упражнения

1. Какой схемой можно представить процесс обработки числовой последовательности, заданной в виде:

   $n, A_1, A_2, \ldots, A_n$?

2. Напишите фрагменты программы по этой схеме, используя операторы цикла ***while*** и ***for.***

3. Вместо следующего оператора ***for*** напишите фрагмент программы с оператором ***while:***

```
for (i=1, s=0;  i<=n;  i++)
{   scanf ("%f", &a);
    if  (a>0 && i%2)  s+=a;
}
```

4. Что определяется в предыдущем фрагменте программы?

5. Что определяется в следующем фрагменте программы?

```
for (k=0;  n>0;  n--)
{   scanf ("%i", &a);
    if  (a%2 == 0)  k++;
}
```

Порядок выполнения работы

1. Ознакомьтесь с теоретическим материалом и примером решения задачи. Ответьте на контрольные вопросы.

2. Получите у преподавателя индивидуальное задание.

3. Составьте блок-схему и программу на языке С (С++) и подберите тесты для проверки программы на компьютере.

4. Отладьте программу на компьютере и покажите результаты тестирования преподавателю.

5. Оформите и сдайте отчет по лабораторной работе преподавателю.

Задания

1. Даны натуральные числа n, A_1, A_2, ..., A_n. Определить количество членов A_k последовательности A_1, ..., A_n:

а) кратных 3 и не кратных 5;

б) имеющих четные порядковые номера и являющихся нечетными числами;

в) удовлетворяющих условию $2^k < A_k < k!$.

2. Даны натуральное число n, действительные числа $A_1,A_2,...,A_n$. Получить:

а) удвоенную сумму всех положительных членов последовательности A_1, A_2, ..., A_n;

б) среднее арифметическое четных чисел последовательности;

в) сумму отрицательных и количество положительных членов последовательности $A_1, A_2, ..., A_n$;

г) min $(|A_1|, ..., |A_n|)$;

д) min $(A_1, A_3, ...)$ + max $(A_2, A_4, ...)$;

е) $A_1 * A_2 + A_2 * A_3 + ... + A_{n-1} * A_n$;

ж) произведение и количество чисел последовательности, кратных 7;

з) максимальное число последовательности и его порядковый номер.

ЛАБОРАТОРНАЯ РАБОТА №5

ПОСЛЕДОВАТЕЛЬНАЯ ОБРАБОТКА СИМВОЛОВ

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 лабораторной работы №3), только переменной будет присваиваться при вводе не число, а очередной символ.

Функции *getchar()* и *putchar()*

Библиотечные 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 возвращает его значение выполняемой программе.

Примечание**.** При вводе данных с помощью функций *getchar()* и *scanf()* вначале вся введенная строка помещается в так называемый буфер ввода, а затем уже функции читают данные из этого буфера. Поэтому пока вы не нажмете клавишу <Enter>, функций *getchar()* и *scanf()* ничего читать

не будут, и вы можете редактировать данные.

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 */
```

Объявления функций *getchar()* и *putchar()* содержатся в файле *stdio.h*.

Пример решения задачи

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

```
// Программа  5.1
#include <stdio.h>
void  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

Номер теста	Исходный текст	Ожидаемый результат
1	МОСКВА, БЕРЛИН, ВАРНА **.**	В тексте есть 'ВА'.
2	IBM PC **.**	В тексте нет 'ВА'.
3	**.**	В тексте нет 'ВА'.

Контрольные вопросы и упражнения

1. Как ввести с клавиатуры один символ?
2. Какие функции можно использовать для вывода символа?
3. Что нужно изменить в приведенной выше программе, чтобы определялось количество сочетаний "ВА"?
4. Можно ли вместо оператора цикла ***do while*** в примере программы использовать оператор ***while***?
5. Можно ли убрать из программы следующую строку:

if (s!= '.')

6. Составьте схему к программе.

Порядок выполнения работы

1. Ознакомьтесь с теоретическим материалом и примером решения задачи. Ответьте на контрольные вопросы.
2. Получите у преподавателя индивидуальное задание.
3. Составьте блок-схему и программу на языке С и подберите тесты для проверки программы на компьютере.
4. Отладьте программу на компьютере и покажите результаты тестирования преподавателю.
5. Оформите и сдайте отчет по лабораторной работе преподавателю.

Задания

1. Дан текст произвольной длины, оканчивающийся символом ‘**;**’ **.** Определить:

а) количество цифр в тексте;

б) есть ли в тексте скобки.

2. Дана последовательность символов и количество символов в этой последовательности. Определить:

а) есть ли в тексте латинские буквы;

б) количество сочетаний **:=** .

3. Дан текст произвольной длины, оканчивающийся символом **#.** Определить количество строк в тексте (каждая строка заканчивается символом перевода строки '\n').

4. Дан текст произвольной длины, оканчивающийся точкой. Текст состоит из слов, разделенных пробелами. Подсчитать

а) количество слов в тексте;

б) количество слов, начинающихся с буквы **к;**

в) количество слов, заканчивающихся буквой **а.**

5. Дано скобочное выражение, оканчивающееся точкой с запятой. Определить:

а) правильно ли расставлены скобки в выражении;

б) количество уровней вложенности скобок в выражении.

6. Дана строка символов. Признак конца - символ '\n' (перевод строки). Необходимо:

а) вывести символы строки без лишних пробелов, т.е. если подряд следует несколько пробелов, оставить только один;

б) удалить последовательности символов, заключенные в фигурные скобки;

в) вывести последовательности символов, заключенные в скобки. Каждую такую последовательность выводить с новой строки;

г) проверить, есть ли одинаковые соседние символы;

д) определить наибольшее число подряд идущих одинаковых символов;

е) определить, каких символов больше: цифр или латинских букв;

ж) **з**аменить прописные латинские буквы на строчные (разность кодов соответствуюших строчных и прописных латинских букв = 32, например, 'a'-'A' = 32).

ЛАБОРАТОРНАЯ РАБОТА №6

ОБРАБОТКА МАССИВОВ

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

Примеры объявлений 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
int  x [ ] = {10, 20, 30, 40};   // массив из 4 целых чисел
int  y [100] = {0};              // массив из 100 нулевых элементов
```

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). Для обращения к элементу двумерного массива задается два индекса: в первых квадратных скобках – номер строки, а во вторых скобках – номер столбца матрицы.

Ввод/вывод числовых массивов осуществляется в цикле поэлементно. Например:

```
int  x[10];
int  i;
for ( i=0;  i<10;  i++)  scanf_s (“%d”,  &x[i]);
    //  C++:
float  m[4][6];
for (int  i=0; i<4; i++)
   for (int  j=0; j<6; j++)  cin >> m[i][j];
```

Для ввода символьных строк можно использовать библиотечную функцию ***gets()*** или ***gets_s()***, которая вместо символа перевода строки записывает в заданный массив после введенной строки символ '\0' (нулевой байт) – признак конца строки. Например:

```
char  s[81];
gets_s  (s);
```

Вывод символьных строк можно выполнить либо с помощью библиотечной функции ***puts()***, либо функции ***printf()***, задав формат ***%s***.

Например:

```
char str[ ] = "Привет";
puts(str);      /* или   printf("%s\n", str); */
```

Пример решения задачи

Задача. Входная строка содержит последовательность слов, разделенных пробелами. Признак конца строки – символ '\n' (перевод строки). Вывести на экран слова длиной до пяти символов.

```
/*  Программа 6.1. Первый вариант решения задачи:
 в массиве   запоминается текущее слово, текст
 вводится посимвольно  */

#include <stdio.h>
#include <conio.h>
#define  DLSL  80   // макс. длина слова

void   main()
{  char s;              // текущий  символ
   char sl[DLSL];       // текущее  слово
   int i,j;             // индексы текущего символа в слове
   int psl=1;     // признак, что слово длиной до 5 символов первое
   printf ("\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");
    printf ("\nДля завершения нажмите любую клавишу");
    getch();       // чтение символа без отображения его на экране
}
```

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

```
/*   Программа 6.2. Второй вариант решения задачи :
     в массиве запоминается вся строка, которая вводится
     с помощью функции    gets( )  */

#include <stdio.h>
#include <conio.h>
#define  DLSTR  80     // макс. длина строки

void   main()
{  char   str[DLSTR];   // входная  строка
    int    i,j;                 //  индексы текущего символа в строке
    int  n,k;       /* индексы первого и последнего символов текущего
                                  слова  в строке */
    int net_sl=1;          //  признак, что слов длиной до 5 симв. нет
    printf ("\nВведите строку символов\n");
    gets_s (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

Контрольные вопросы и упражнения

1. Как описать массив из 20 целых чисел?
2. Как ввести значения элементов этого массива?
3. Как описать и выполнить ввод строки длиной до 50 символов?
4. Для чего служит функция ***puts()***?
5. Что происходит в следующем фрагменте из программы 6.1:

```
i=0;
do
{   sl[i++]=s;
    s=getchar();
}
while ((s!=' ') && (s!='\n'));
```

6. Что происходит в следующем операторе ***for*** из программы 6.1:

```
for (j=0; j<i; j++)
      putchar(sl[j]);
```

7. Составьте схему к следующему оператору ***if*** из программы 6.1:

```
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(' ');
}
```

8. Что происходит в следующем фрагменте из программы 6.2:

```
n=i;
do  i++;  while ((str[i]!='  ') && (str[i]!='\0'));
k=i;
```

9. Что происходит в следующем операторе ***if*** из программы 6.2:

```
if ( k-n < 5 )
{   for (j=n; j<k; j++)
        putchar(str[j]);
     putchar('  ');
     net_sl=0;
}
```

Порядок выполнения работы

1. Ознакомьтесь с теоретическим материалом и примером решения задачи. Ответьте на контрольные вопросы.

2. Получите у преподавателя индивидуальное задание.

3. Составьте блок-схему и программу на языке С и подберите тесты для проверки программы на компьютере.

4. Отладьте программу на компьютере и покажите результаты тестирования преподавателю.

5. Оформите и сдайте отчет по лабораторной работе преподавателю.

Задания

1. Дана строка символов. Признак конца строки – символ '\n' (перевод строки). Строка состоит из слов, разделенных пробелами. Вывести:

а) слова, у которых первая и последняя буквы одинаковые, и количество таких слов;

б) слова, заканчивающиеся буквой 'a', с порядковыми номерами этих слов в исходной строке;

в) слова, заканчивающиеся слогом 'ва', и длину каждого из этих слов;

г) слова, которые начинаются с буквы 'с' и содержат более 4 символов, и количество всех слов;

д) самое короткое слово и его длину;

е) самое длинное слово и его порядковый номер в исходной строке.

2. Дан массив, состоящий из ***n*** целых чисел (n<=10). Необходимо:

а) переставить местами максимальный и минимальный элементы;

б) упорядочить массив по убыванию элементов методом последовательного нахождения максимума;

в) упорядочить массив по убыванию элементов методом последовательного нахождения минимума;

г) упорядочить массив по возрастанию элементов методом "пузырька".

3. Дана строка длиной не более 80 символов, оканчивающаяся точкой. Подчеркнуть все гласные буквы в строке.

4. Дана строка длиной не более 80 символов, оканчивающаяся точкой. Определить:

б) сколько раз каждая цифра встречается в строке;

в) сколько раз каждая буква латинского алфавита встречается в строке;

г) сколько раз каждая гласная буква латинского алфавита встречается в строке;

д) символ, встречающийся в тексте с максимальной частотой.

5. Дана целочисленная матрица из 4-х строк и 5-ти столбцов. Определить:

а) сумму элементов каждой строки матрицы;

б) сумму элементов каждого столбца и общую сумму элементов матрицы;

в) количество нулевых элементов в каждой строке матрицы;
г) количество отрицательных элементов в каждом столбце матрицы;
д) среднее арифметическое элементов матрицы и наибольший элемент;
е) наименьший элемент в каждой строке матрицы;
ж) наибольший элемент в каждом столбце матрицы;
з) номер строки, где сумма элементов наибольшая;
и) номер столбца с наименьшей суммой элементов.

6. Дана целочисленная квадратная матрица размером $n \times n$, где n – заданное число ($n<7$). Определить:
а) сумму и наибольший элемент главной диагонали матрицы;
б) сумму элементов, расположенных выше главной диагонали;
в) наименьший среди элементов, расположенных ниже главной диагонали матрицы;
г) среднее арифметическое элементов каждого столбца матрицы;
д) количество положительных элементов в каждой строке матрицы;
е) сумму и наименьший элемент побочной диагонали.

ЛАБОРАТОРНАЯ РАБОТА №7

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 С играют ту же роль, какую играют функции, подпрограммы и процедуры в других языках программирования.

Программа на С и С++ 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 (определения) 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, иначе придется функции предварительно объявить (объявление функции – это ее заголовок, после которого ставится точка с запятой).

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, так как это формальные параметры. Если функция не имеет параметров, то пишутся пустые скобки.

Примеры заголовков функций:

```
void 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 ее именем, – это тип возвращаемого функцией значения. По умолчанию предполагается тип ***int***. Если функция никакое значение не возвращает, то указывается тип ***void.***

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;
}
```

Свою функцию можно вызвать 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,a+b);

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 *a+b*. 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 поспедством параметров. Если данные входные и не изменяются функцией, 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 возвращается вызывающей функции с помощью оператора возврата *return*). Если функция должна вернуть вызывающей программе несколько значений, в этом случае передавать ей в качестве аргументов нужно адреса переменных. Вспомните вызов функции *scanf_s()*. Если, например, нужно ввести значения двух переменных ***x*** и ***y*** типа ***int***, вы должны написать:

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

(*&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 объявлены 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 переменных. При вызове функции *function1()* переменной *ptr1* присваивается адрес *a*, а *ptr2=&b*.

Таким образом, функции становятся доступны 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 ***** (операцию разадресации).

Если функции нужно передать массив в качестве аргумента, то при вызове указывается просто имя массива. Имя массива рассматривается как указатель на начальный элемент, поэтому подпрограмме передается не сам массив, а его адрес. Объявленный в заголовке функции параметр-массив будет указателем на массив-аргумент. В заголовке функции при объявлении одномерного массива-параметра размер массива можно не задавать (написать пустые квадратные скобки). Для двумерных массивов число строк можно не задавать, а вот число столбцов указывать необходимо.

Примеры решения задач

Задача 1. Даны две строки длиной до 80 символов. Определить число латинских букв в каждой строке.

```
// Программа 7.1

#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_s (s1);
        gets_s (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 2. Описать функцию, которая для заданного числового массива определяет сумму и количество положительных элементов.

```
// Программа 7.2
```

```
#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_s ("%d ", &a[i] );
    SumPos (a, 6, &s, &k);    /* вызов функции */
    printf ("Сумма положительных чисел =  %f\n", s);
    printf ("Количество положительных чисел:  %d\n", k);
}
```

Контрольные вопросы и упражнения

1. Что такое функция?

2. Как записывается заголовок функции?

3. От чего зависит тип функции, задаваемый в заголовке функции?

4. Когда при описании функции указывается тип ***void***?

5. Для чего нужны параметры функции? Как выглядит список параметров в заголовке функции?

6. Что делает оператор ***return***?

7. Как вызвать функцию для выполнения? Можно ли одну и ту же функцию вызывать несколько раз с различными параметрами?

8. Когда следует передавать функции в качестве аргументов значения переменных, а когда – их адреса?

9. Если при вызове функции в качестве аргумента указан адрес переменной, как следует объявить соответствующий формальный параметр?

10. Как через указатель обратиться к той переменной, на которую он ссылается?

11. Объясните следующую строку из программы 7.2.

```
if (m[i] > 0)   (*s) += m[i],  (*k)++;
```

Порядок выполнения работы

1. Ознакомьтесь с теоретическим материалом и примерами решения задач. Ответьте на контрольные вопросы.

2. Получите у преподавателя индивидуальное задание.

3. Составьте блок-схемы (отдельно для каждой функции) и программу на языке С и подберите тесты для проверки программы на компьютере.

4. Отладьте программу на компьютере и покажите результаты тестирования преподавателю.

5. Оформите и сдайте отчет по лабораторной работе преподавателю.

Задания

1. Даны две строки длиной до 80 символов. Необходимо:

а) определить, в какой строке больше цифр: в первой или во второй (вывести соответствующее сообщение). Описать в виде отдельной функции определение числа цифр в заданной строке;

б) определить, сколько раз заданный символ встречается в обеих строках. Описать в виде отдельной функции определение числа повторений заданного символа в заданной строке;

в) удалить заданный символ в каждой строке. Описать в виде отдельной функции удаление заданного символа в заданной строке;

г) заменить в каждой строке один заданный символ на другой заданный символ. Описать в виде отдельной функции замену символа в строке;

д) определить общее число гласных букв в обеих строках. Описать в виде отдельной функции определение числа гласных букв в заданной строке.

2. Дан числовой массив X[12]. Определить:

а) $\sum_{i=0}^{6} X[i] - \sum_{i=7}^{11} X[i]$.

Описать в виде отдельной функции определение суммы элементов заданной части массива.

б) max (X[0], X[1], … , X[5]) – max (X[6], X[7], … , X[11]).

Описать в виде отдельной функции определение максимального элемента в заданной части массива.

в) min (X[0], X[1], … , X[4]) + min (X[5], X[6], … , X[11]).

Описать в виде отдельной функции определение минимального элемента в заданной части массива.

3. Даны два массива из n целых чисел ($n \leq 10$). Для каждого массива

а) определить произведение элементов, которые по модулю меньше заданного числа. Описать в виде отдельной функции определение произведения элементов заданного массива, которые по модулю меньше заданного числа;

б) определить сумму элементов с четными индексами. Описать в виде отдельной функции определение для заданного массива суммы элементов с четными индексами;

в) определить количество нечетных элементов. Описать в виде отдельной функции определение количества нечетных элементов в заданном массиве;

г) определить сумму элементов до минимального элемента. Описать в виде отдельной функции определение для заданного массива суммы элементов до минимального элемента;

д) определить количество элементов до максимального элемента. Описать в виде отдельной функции определение количества элементов до максимального элемента заданного массива;

е) проверить, упорядочены ли элементы по возрастанию. Описать отдельную функцию проверки, упорядочены ли элементы заданного массива по возрастанию;

ж) проверить, упорядочены ли элементы по убыванию. Описать отдельную функцию проверки, упорядочены ли элементы заданного массива по убыванию;

з) упорядочить элементы по возрастанию. Описать отдельную функцию сортировки заданного массива по возрастанию элементов;

и) упорядочить элементы по убыванию. Описать отдельную функцию сортировки заданного массива по убыванию элементов;

к) удалить заданное число. Описать отдельную функцию удаления заданного числа в заданном массиве. Функция должна вернуть число удаленных элементов.

СПИСОК ЛИТЕРАТУРЫ

1. Хохлов Д.Г. Программирование на языке высокого уровня. Часть 1: Основы программирования: Учебник. - Казань: Мастер Лайн, 2009. - 266 с.

2. Павловская Т.А. C/C++. Программирование на языке высокого уровня. - СПб: Питер, 2010. - 461с.

3. Павловская Т.А., Щупак Ю.А. C/C++. Структурное программирование: Практикум. - СПб: Питер, 2010. - 240с.

4. Хохлов Д.Г. Введение в программирование: Учебное пособие.- Казань: Изд-во Казан. гос. техн. ун-та, 2005. - 136 с.

5. Хохлов Д.Г., Захарова З.Х. Введение в программирование. Практикум на языке С: Учебное пособие. - Казань: Изд-во Казан. гос. техн. ун-та, 2005. - 76 с.

СОДЕРЖАНИЕ