МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

В.Г. Михайлов

Лекционный ресурс электронного курса «Информационные технологии» (система дистанционного обучения MOODLE)

Электронное интерактивное учебное пособие

С А М А Р А

2011

Автор: **Михайлов Владимир Гаврилович**

Михайлов В.Г., Лекционный ресурс электронного курса «Информационные технологии» (система дистанционного обучения «Moodle») [Электронный ресурс] : электрон. интерактив. учеб. пособие / В.Г. Михайлов; Минобрнауки России, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т). - Электрон. текстовые и граф. дан. (1,6 Мбайт). - Самара, 2011. - 1 эл. опт. диск (CD-ROM). - Систем. требования: ПК Pentium; Windows 98 или выше.

Представлен комплект лекций по курсу «Информационные технологии», использованный как лекционный ресурс в системе дистанционного обучения MOODLE. Система размещена на сайте кафедры общей информатики СГАУ. СОД MOODLE предназначена для интерактивного освоения материалов курса и дает возможность контроля текущих знаний студентов.

Лекции включают базовые положения дисциплины «Информационные технологии». Даются основы программирования на языке высокого уровня Турбо Си, рассматриваются многочисленные примеры решения стандартных задач.

Учебный курс ориентирован на 5 факультет, 1 курс, I семестр (очное) для подготовки:

- бакалавров по направлениям подготовки 210400.62, 200500.62, 201000.62, 211000.62, 210100
- студентов по специальности 210601.65

Подготовлено на кафедре общей информатики СГАУ.

История развития вычислительной техники.

К первым ЭВМ

ЭВМ являются основой современной информационной среды. Их основное назначение - хранение данных и их преобразование по форме и содержанию. Процесс фиксации знаний и данных начался с появлением письменности и счета. Первые попытки автоматизации обработки данных начались с создания простейших механических арифметических устройств таких как абак или счеты. Затем появились арифмометры. Счетами пользовались еще до 80-ых годов прошлого века, а на арифмометрах "Феликс" практические занятия в КуАИ проводились и в 1967 году. Венцом механических устройств стала аналитическая машина Беббиджа (первая половина XIX века), в которой он за сто лет предвосхитил структуру будущих ЭВМ, но не смог полностью ее реализовать из-за технических трудностей (в 1991 году эта машина была воссоздана в Лондоне и успешно заработала, вычисляя различные функции).

В 1943 г. инженером Цузе в Германии было построено вычислительное устройство на электромагнитных реле. Эта машина имела 15 метров в длину, 2,5 метра в высоту, содержала 800 тысяч деталей и работала с десятичными числами длиной 23 разряда. Сложение выполнялось за 0,3 секунды, а умножение - за 3 секунды.

Работы над ЭВМ были начаты, скорее всего, в 1937 г. в США профессором Джоном Атанасовым, болгарином по происхождению. Этот проект не был завершен, но позднее в результате судебного разбирательства его признали родоначальником электронной вычислительной машины.

В 1944 г. в школе Мура (Филадельфия, США) заработала первая в мире ЭВМ на электронных лампах, известная как "ЭНИАК" (ENIAC - Electronic Numerical Integrator and Computer - электронный цифровой интегратор и вычислитель). Ее соавторами были американцы Экерт и Моучли. Официальный ввод ЭНИАК в эксплуатацию состоялся 15 февраля 1946 г. Эта машина проработала до 1955 г.

Каким был первый компьютер? Он содержал 18 тыс. вакуумных ламп, занимал площадь более 130 кв.м., весил 30 т. и потреблял 150 квт. мощности. ЭНИАК выполнял сложение за 0,2 мс, а умножение - за 2,8 мс. В машине использовалась десятичная система счисления. Это был скорее калькулятор, в котором смена программы выполнялась переключением проводов.

В СССР история развития вычислительной техники начинается в 1947 г. с организации под Киевом в монастыре "Феофания" лаборатории под руководством Лебедева, в которой была разработана МЭСМ - модель электронно-счетной машины. Ее блок-схема стала классической и повторялась в большинстве отечественных машин первого поколения. Уже 25 декабря 1951г. МЭСМ была официально принята в эксплуатацию госкомиссией во главе с академиком Келдышом. МЭСМ выполняла 50 операций в секунду. В оперативной памяти сохраняла 31 число и 63 команды. Машина содержала 1000 ламп и потребляла 25 квт. Для нее была разработана долговременная память в виде магнитного барабана (цилиндра), способного сохранить 5000 чисел. До 1953 г. МЭСМ оставалась единственной работающей в стране вычислительной машиной, поэтому использовалась для решения только особо важных задач. График ее работы утверждал президент АН СССР. Среди прочих задач МЭСМ позволила выполнить сложнейший по тем временам расчет устойчивости работы Куйбышевской ГЭС и значительно увеличить передачу электроэнергии потребителям в тяжелых послевоенных условиях. Она проработала до 1956 г., после чего была передана в Киевский политехнический институт. В 1952 г. была введена в эксплуатацию БЭСМ - самая быстродействующая машина в Европе, а затем - Урал-1 (такая машина работала в КуАИ до 1967 г.), с которого началось уже активное внедрение ЭВМ в народное хозяйство.

В Японии первая машина "Фуджик" была введена в эксплуатацию только в 1956г., а в ФРГ серийное производство ЭВМ началось с 1958г.

Большой вклад в развитие вычислительной техники внес Джон фон Нейман. После анализа работы ЭНИАКа Нейман с коллегами сформулировал ряд требований к структуре компьютера, которые нашли применение в последующих моделях и позволили значительно повысить их эффективность по сравнению с ЭНИАКом. Совокупность этих требований получила название "архитектура фон Неймана" и использовалась практически во всех машинах первых трех поколений. Важнейшие из них:

- машина должна работать в двоичной системе счисления
- программа, как и исходные данные, должна размещаться в памяти машины
- команды программы, как и числа, должны записываться в двоичном коде
- память должна иметь иерархическую организацию
- процессор строится на основе схем, выполняющих операцию сложения; создание специальных схем, выполняющих другие арифметические операции, нецелесообразно

- все разряды числа обрабатываются одновременно

Отход от этой схемы начал происходить только в машинах 4-го поколения.

Кроме этого, фон Нейман показал, как можно упростить процесс программирования ЭВМ за счет его разделения на 2 этапа. На первом этапе создавалась понятная и подробная блок-схема алгоритма, которая затем по четким правилам переводилась в машинные команды. В дальнейшем эта идея воплотилась в современных трансляторах.

Поколения ЭВМ

ЭНИАК дал старт первому поколению компьютеров. Поколения ЭВМ принято различать по той элементной базе, на основе которой строятся эти машины, поэтому 1-ое поколение характеризуется использованием электронных вакуумных ламп. Работа на этих машинах требовала большого терпения. Особенностью первых ЭВМ была их чрезвычайно низкая надежность. За день могло выйти из строя до 10 ламп. Характеристики самих ламп значительно отличались друг от друга, поэтому во многих случаях требовался их индивидуальный подбор. Ламповой машине для выхода на нормальный режим работы требуется до 1-2 часа. Кроме того, лампы чаще всего перегорали именно в момент включения, поэтому машины старались выключать реже, а иногда они вообще работали круглосуточно.

В машинах 1 и 2 поколений ввод данных выполнялся в основном с перфокарт - карточек из плотной бумаги. На этих карточках с помощью специального перфоратора (смесь пишущей машинки с дыроколом) пробивались последовательности отверстий, которые кодировали отдельные символы операторов программы. Чем больше была программа, тем выше получалась стопка перфокарт (до 0,5 метра). Для загрузки программы в машину использовалось устройство перфоввода. Это устройство выбирало по одной карточке из стопки и с помощью фотоэлементов считывало с нее последовательность отверстий. Такой способ ввода данных приводил к появлению дополнительных ошибок при набивке перфокарт и при их вводе (карту заминало, перекашивало, отверстия или фотоэлементы засорялись). Кроме того, бумажная карточка выдерживала только ограниченное количество чтений в перфовводе, поэтому работа по поддержанию программы в рабочем состоянии была очень трудоемкой. Работа вычислительного центра сопровождалась постоянным гулом электромоторов (охлаждение, вентиляция), стуком перфораторов, шумом устройств ввода и печати.

Машины первого поколения не могли иметь широкого применения в силу их высокой стоимости и малой надежности. Они использовались для решения трудоемких задач только в крупных организациях.

Эра ламповых машин закончилась в конце 50-годов. В 59 году появились машины второго поколения, основой которых стали полупроводниковые диоды и транзисторы (ШМ 1401). Их быстродействие, надежность и экономичность намного превзошли ламповые схемы. Скачкообразное улучшение характеристик позволило значительно расширить область применения полупроводниковых машин. Появилась возможность установки ЭВМ на движущиеся объекты - корабли, самолеты, космические аппараты. Создавались машины разных классов. Большие - СТРЕТЧ (США, 1961 г.), БЭСМ-6 (СССР, 1966 г., 1 млн. операций в сек.), средние семейства Урал, Минск. Наиболее мощная из них Минск-32 выполняла 65 тыс. операций в секунду. Появились и малые машины типа Мир (1966 г.) для инженерных расчетов. В качестве оперативной памяти стали использовать ферритовые сердечники, а для вывода информации стали применяться электроннолучевые трубки. В КуАИ еще в 1967 г. лабораторные работы выполнялись на своеобразной машине ПРОМИНЬ со штыревым вводом программы. В середине 60-х годов бум в области производства транзисторных машин достиг максимума. Проблемой второго поколения стала трудоемкость сборки (пайки деталей), т.к. количество деталей в ЭВМ достигало 100.000 штук. Выходом из этого положения стало появление в 1959 г. т.н. интегральных схем.

Интегральные схемы (ИС) стали основой машин третьего поколения. ИС представляют собой небольшую пластинку (в несколько кв.см.) из полупроводникового материала (например, кремния), на которую слой за слоем через специальные трафареты напыляются различные материалы. Материалы подбирают так, что они формируют диоды, транзисторы, конденсаторы, сопротивления и проводники, образующие электронную схему. Приоритет в изобретении ИС принадлежит американским ученым Килби и Нойсу. Массовый выпуск ИС начлся в 1962 г. Степень их интеграции составляла порядка 20 транзисторов на 1 кв.мм. К настоящему времени этот показатель улучшен в десятки тысяч раз, но уже и тогда весь ЭНИАК можно было бы разместить на пластинке в 1,5 кв.см. Начало развиваться новое направление в технике - микроэлектроника.

Первую серию машин на гибридных ИС выпустила фирма ШМ в 1964 г. Серия, получившая название ШМ-360, оказала значительное влияние на развитие вычислительной техники. В состав серии вошли 9 машин разной производительности (от 5тыс. до 5 млн. операций в сек.). В

СССР в 70 г. появилась мини-ЭВМ Наири-3. В 72 г. началось производство машин серии ЕС, аналогичных ШМ-360. Среди них - самая простая ЕС-1010 (10 тыс.опер./сек.) и самая мощная ЕС-1060 (2 млн. опер./сек.). Именно в этот период сформировалась мощная индустрия вычислительной техники, которая начала выпуск ЭВМ для массового коммерческого применения.

Переход к четвертому поколению произошел в начале 70-х годов. В 1971 появилось семейство ШМ-370 на монолитных интегральных схемах. В 1977 был продемонстрирован Apple II фирмы 'Макинтош', а в 1981 году - ШМ РС, с которыми большинство пользователей связало понятие 'персональный компьютер'. Основой этих машин стали СБИСы и микропроцессоры.

Возможности вычислительной техники лучше всего характеризуют т.н. суперкомпьютеры. В 1985 году многопроцессорный комплекс 'Эльбрус-2' выполнял 125 млн. операций в секунду. Еще мощнее был американский 'Cray-2' с производительностью 2 млрд. операций в секунду. Современные суперкомпьютеры по своим размерам превосходят первые ламповые ЭВМ, но и производительность у них неизмеримо выше. Это многопроцессорные комплексы (десятки тысяч процессоров), реализующие принципы параллельных вычислений. Производительность российского МВС-1000М (НИИ 'Квант') составляет 1 Тфлопс (триллион операций с плавующей запятой в секунду). Американский Cray XI (стоимость 2,5 млн. долларов) имеет более 50 Тфлопс и 65 Терабайт ОЗУ (в такой памяти можно удержать все библиотеки мира). После 2010 года достигнута производительность в один петафлопс - миллион миллиардов операций в секунду и этот показатель продолжает увеличиваться.

Основными задачами для таких машин является криптография, создание искусственного интеллекта, моделирование ядерных взрывов, долгосрочный прогноз погоды и моделирование других сложных систем.

Наш университет является одним из немногих, имеющих свою супер-ЭВМ.

Основные характеристики: платформа ШМ BladeCenter, 112 блейд-серверов ШМ BladeCenter HS22. Каждый сервер имеет по два четырехядерных процессора Intel Xeon 5560 с частотой ядра 2,8ГГц. Общий объем оперативной памяти 1,3Тб. Система хранения данных на ЮТб. Для межпроцессного взаимодействия распределенных приложений используется технология QDR InfiniBand, на оборудовании QLogic, с пропускной способностью до 40 Гбит/с. Связь с системой хранения данных кластера осуществляется также по технологии QDR Infiniband. Управляющая сеть Gigabit Ethernet используется для сетевой загрузки операционной системы на блейд-сервера, передачи управляющих

сообщений, статистических данных, а также для мониторинга работы узлов кластера. Пиковая производительность кластера - 10 ТФлопс.

Сегодня это самый мощный суперкомпьютер в Самарской области. Ресурсы суперкомпьютерного центра могут использоваться в удаленном режиме всеми образовательными и научными учреждениями, а также промышленными предприятиями Самарской области и всей России. Доступ к ресурсам суперкомпьютера обеспечивается телекоммуникационными возможностями региональной сети науки и образования.

Языки программирования.

Роль информатики

Появление информатики связано с появлением и развитием современной вычислительной техники, повлекшей разработку специфических методов решения задач. Поэтому информатику можно определить как совокупность знаний и навыков, необходимых для решения задач на ЭВМ.

Первые задачи носили чисто математический характер. Но машина считает иначе, чем человек. Если человеку проще вывести формулу, чтобы найти один искомый результат, то компьютеру проще вычислить множество значений и выбрать из них нужное (американцы называли ЭВМ цифровой мельницей.) Это привело к появлению специальных численных методов решения. С развитием ЭВМ расширялось и разнообразие решаемых задач. На компьютерах начали обрабатывать текстовую и графическую информацию, решать логические задачи. Сейчас задачи физики и математики занимают очень небольшую долю всего машинного времени персональных компьютеров (ПК) . Их главная роль сейчас в другом. Компьютеры, объединённые в сети, стали основой современной информационной среды. Их основное назначение - накопление, обработка и преобразование данных. Постепенно всё написанное человечеством на бумаге переводится в электронную форму.

Языки программирования и трансляция программ.

Компьютер не может работать сам по себе. Чтобы он выполнил хоть какое-то (пусть самое простое) действие, нужна программа.

- Программа - это последовательность команд для ЭВМ.
- Программирование - составная часть информатики.

С понятием программы тесно связано понятие алгоритма. Термин 'алгоритм' происходит от имени ученого 9 века Мухаммеда аль-Хорезми, который определил суть этого понятия. Алгоритм - это описание последовательности действий, в результате выполнения которых достигается искомый результат. Примерами алгоритма являются описания порядка построения геометрической фигуры с помощью циркуля и линейки или вычисления квадратного корня какого-либо числа.

- С этой точки зрения программу можно определить, как предельно детализированный алгоритм.

Для написания программ служат языки программирования. Но при этом каждая ЭВМ понимает только один язык - машинный. Команды этого языка 'зашиты' в процессоре ЭВМ и кодируются двоичными числами. Программирование в таких машинных командах требует высокой квалификации и является очень трудоемким процессом. Поэтому, для широкого использования были разработаны т.н. алгоритмические языки (или языки высокого уровня), команды которых (операторы) строятся на основе обычных слов. Программы стали более наглядными, но для каждого из этих языков пришлось создавать специальную программу - переводчик (транслятор), которая переводит команды программы с алгоритмического языка на машинный язык.

- Процесс перевода программы с языка высокого уровня на машинный язык называется трансляцией.

Трансляция выполняется самой ЭВМ. В результате трансляции создается новая программа, которая сохраняется в файле с расширением '.exe' (реже - '.com'). Эта программа полностью готова к выполнению, может работать на любом компьютере, уже не требуя наличия языков программирования.
Например, программа на языке Си ' lab1.c ' транслируется в программу в машинных кодах ' lab1.exe '.

Файлы и маршруты.

В ПК вся программы и все данные для них сохраняется на жестком магнитном диске (винчестере), которому присваивается имя C:. Иногда для удобства работы часть диска C: выделяется и оформляется как отдельный диск или несколько дисков с именами D:, E: и т.д..
Каждый отдельный блок информации сохраняется на диске в виде отдельного файла.

- Файл - это именованная область диска.

Каждый файл должен иметь своё имя. Имя файла включает собственно имя и т.н. расширение, отражающее характер данных, содержащихся в файле, например, 'lab1.c'. Здесь 'lab1' - собственно имя файла, а '.c' - расширение имени, сообщающее пользователю и операционной системе, что в данном файле хранится текст программы на языке Си. Ориентируясь на расширение, операционная система может автоматически подключать нужную программу для обработки этого

файла. Например, расширение .doc .автоматически запускает редактор WORD.
Чтобы облегчить поиск файлов, на диске создается система вложенных папок (разделов). Маршрут поиска файла (адрес файла), таким образом, задается цепочкой имен диска и папок:
D:\KC\311\LAB1.C

Структурапрограммы.

Ниже приведён пример программы, которая вычисляет площадь (s) и периметр (p) прямоугольного треугольника по заданным значениям двух катетов (ab, ac).

```
/* Программа расчета треугольника*/
#include < stdio.h >                    //указание на подключение
#include < math.h >                     //к программе служебных
#include <conio.h>                      //библиотек

int main()                              // заголовок главной функции
{                                       // начало функции
      float ab,ac,bc,s,p;               // объявление переменных
      clrscrQ;                          // команда -'очистить экран'
      printf("BBefln катеты\n");        // -:- 'вывести на экран'
      scanf("%f %f",&ab,&ac);           // -:- 'ввести с клавиатуры'
      bc=sqrt(ab*ab+ac*ac);             // -:- 'рассчитать гипотенузу'
      s=ab*ac/2;
      p=ab+ac+bc;
      printf("s=%7.2f\n",s);
      printf("p=%7.2f\n",p);
      printf(" \n");
      printf("Bbixofl:  0 + Enter !\n");
      scanf("");
}                                       // конец функции
```

Любая программа на языке Си должна содержать главную функцию 'mainQ', с выполнения которой начинается выполнение всей программы. Кроме этой функции в состав программ могут включаться и другие функции.
Программа состоит из отдельных команд - операторов.

- Все операторы (за отдельными исключениями) заканчиваются символом ';'.

- В одной строке допускается несколько операторов.
- Текст функции открывается и закрывается фигурными скобками

Все операторы можно разделить на исполняемые операторы и операторы-объявления. Исполняемые операторы образуют 'тело' программы. Они выполняют действия над данными, реализуя алгоритм решения задачи. Операторы-объявления служат для 'настройки ' программы и располагаются в разделах объявлений.

Кроме операторов в состав программы можно включать комментарии, которые начинаются символами '/*' и заканчиваются символами '*/'. Такие комментарии могут занимать несколько строк. В пределах одной строки комментарий можно выделять символами '//'.

В примере программа начинается с подключения библиотек:

#include <stdio.h> //
#include <math.h> //
#include <conio.h> //

В первом случае в исходную программму вставляется файл, поименованный stdio.h. Он содержит команды ввода-вывода printf и scanf. Файл **math.h** содержит математические функции. Файл **conio.h** содержит команды работы с экраном, в частности команду очистки экрана **clrscr().** Угловые скобки <> сообщают препроцессору, что поиск файла нужно осуществлять в стандартных директориях.

Основы языка Си

Алфавит.

Основой любого языка является алфавит. Алфавит языка Си включает:
латинские буквы: A - Z , a - z
цифры: 0-9
различные символы: [ ] { } ( ) " : = ;, . _ - + * / \ > < # % &
и знак пробела '—'.
Программа одну и туже букву в прописном и строчном варианте рассматривает как две разные буквы! Поэтому, имена 'Fan' и 'fan' являются двумя различными именами.
В комментариях и в символьных строках можно использовать буквы русского языка и другие символы.

Идентификаторы.

- Идентификаторы - это имена отдельных элементов программы: операторов языка, функций, переменных и др. элементов.

Имена могут включать только латинские буквы, цифры и знак <_>. Длина имени может быть любой, но учитываются только первые 31 символа.

- Имя не должно начинаться с цифры.

Примеры правильных имен: W, Al, DeltaX, _Fan
Примеры не правильных имен: 2Q, A&B

Данные. Типыданных.

В программе можно использовать самые различные данные: числа, отдельные символы (буквы), символьные (текстовые) строки, логические утверждения вида <Истина> - <Ложь> и т.д.

Пример:
123 -целое число
-15.456 - вещественное (дробное) число с фиксированной точкой
-0.25E+002 - вещественное число с плавающей точкой.

Эта запись эквивалентна числу $-0.25*10^2$

‘A’, – символ

‘Иванов И. И.’ – символьная строка

Данные в программе могут присутствовать в форме констант или в форме переменных.

Переменные.

Это параметры программы, которые служат для сохранения и обозначения данных. Значения переменных могут меняться в процессе выполнения программы. Каждая переменная должна иметь уникальное имя (т.е., два разных параметра не могут называться одинаково).

В качестве имен переменных нельзя использовать зарезервированные слова языка, например, имя ‘printf’.

Все переменные, которые используются в программе, должны быть перечислены в разделе объявлений после ключевых слов, определяющих их тип, например, int – целый тип или float – вещественный тип.

- Тип переменной определяет характер значений, которые могут получать эти переменные.

Типы числовых данных:

```
int   A1,A2,A3;     { целый тип              -32768..+32767 }
float B1,B2,B3;     { вещественный тип       ~10^-38-10^+38 }
```

Арифметические выражения

Действия над числовыми данными выполняются в арифметических выражениях. Арифметическое выражение может включать числовые константы, имена переменных, математические функции, знаки арифметических действий и круглые скобки. Например:

$$bc = \sqrt{ac^2 + ab^2} -> bc = sqrt(ac * ac + ab * ab)$$

Здесь:

sqrt – имя функции

выражение в скобках – аргумент функции

‘*’, ‘+’ – знаки арифметических действий.

Знаки действий:

+	сложение

–	вычитание
*	умножение
/	деление
%	целочисленное деление
++	инкремент
—	декремент

A % B результатом является целый остаток (5 % 3 -> 2)
Операцию % , например, удобно использовать для выделения четных и не четных чисел (или чисел любой кратности).

Порядок действий в выражениях регулируется круглыми скобками и приоритетами действий.
Полный список функций в языке Си можно посмотреть в редакторе (турбо Си) по команде Shift+Fl.
Некоторые из них:

Си	математика
sin(x)	sinx
cos(x)	cosx
tan(x)	itgx
atan(x)	arctgx
fabs(x)	| x |
sqrt(x)	$\sqrt{x}$
log(x)	][nx
loglO(x)	Igx
exp(x) -	e^x
pow(y,x)	f
a3	a*a*a
lg(x)	ln(x)/ln(10)

Оператор присваивания.

В этом операторе переменная получает свое значение. Общий вид оператора:

<имя переменной> = <выражение>;

Вначале вычисляется значение выражения. Найденный результат присваивается переменной. Выражение может быть любого типа, но тип результата должен быть совместим с типом переменной.
Пример: A:=0.5;
A:=B;
A:=(D+B)/2;
A:=A+1; - объяснить ! A=5 A+1=6 A<-6
это пример оператора с накоплением результата!

END.

Операторы ввода-вывода.

Эти операторы используются для ввода данных в программу и вывода полученных результатов на внешние устройства. По умолчанию, устройством ввода является клавиатура, а устройством вывода - дисплей.

- Ввод данных в программу с клавиатуры: scanf(cnncoKl);
- Вывод данных на экран монитора printf(cnHCOK2);

Оператор scanf:
список 1 содержит имена переменных, для которых нужно ввести значения с клавиатуры при выполнении программы. Кроме этого список включает т.н. форматы ввода (шаблоны), какого типа должны быть вводимые данные.
Пример: scanf(" %f %f", &ab, &ac);
здесь:
ab, ac - имена вводимых переменных
%f %f - форматы ввода (f- признак ввода вещественного числа
d - целого числа)
Когда машина встречает в программе оператор **scanf,** то выполнение программы прерывается до тех пор, пока с клавиатуры не будут введены значения всех указанных в скобках переменных. Данные нужно набирать в том порядке, в котором они стоят в списке.
Порядок ввода (два варианта):

1. число <Enter> число <Enter>
2. число '—' число <Enter>

Пустой **scanf** работает как команда остановки.

Оператор printf

Список 2 содержит шаблон оформления строки распечатки в двойных кавычках и имена переменных, значения которых содержатся в строке (не обязательный параметр).

Пример 1: printf("ВВеflМ катеты\п");

Пример 2: printf("s=%7.2f\n",s);

В первом примере на экран выводится только текст 'Введи катеты', вывод числовых значений отсутствует. Текст завершается служебным фрагментом '\п', который переводит курсор в начало следующей строки на экране.

Во втором примере на экран выводится текст 's=' и значение переменной s. Фрагмент %7.2f является форматом для распечатки числа. Здесь:

% - признак формата

f - признак типа числа (f - вещественное, d - целое число)

7.2 - формат изображения вещественного числа (7 - общее число позиций строки, выделяемых для распечатки числа, 2 - количество знаков после запятой). Формат для целой переменной %5d - указывает только общее количество позиций.

Операторы управления.

Введение

В обычном случае программа выполняется оператор за оператором сверху вниз. Такие программы называются линейными. Порядок обработки данных в них не меняется, сколько бы раз мы эту программу не запускали (см. пример 'Расчет треугольника'). Операторы управления позволяют изменять обычный порядок выполнения операторов и за счет этого реализовывать в одной и той же программе различные варианты ее работы. К операторам управления относятся операторы goto, if, switch, while, do, for, break, continue. С их помощью в программах организуются т.н. ветвления и циклы.

Оператор GOTO.

Общий вид:

goto <метка>;

Действие:
Оператор goto прерывает выполнение программы в данной точке и передает управление оператору, перед которым стоит указанная метка. Управление может передаваться и вверх и вниз по программе. Меткой может быть любое разрешённое символьное имя (идентификатор). Метка ставится перед оператором и отделяется от него двоеточием.
Пример

```
.....
goto met;
.....
met:    a=b+c;
.....
```

При передаче управления вверх по программе может возникнуть т.н. бесконечный цикл.
Пример

```
.....
```

m1:a=0;
У=a/2;
goto ml;
.....

Составной оператор.

Это блок (последовательно выполняемых) операторов программы, которые заключены в операторные скобки { - } :
{
<оператор 1>;
<оператор 2>;
<оператор 3>;
.....
<оператор n>;
}
В состав блока могут входить другие блоки (не пересекаясь). В программе необходимо следить за балансом { - }.
Составной оператор может использоваться в тех случаях, когда по правилам требуется ставить один оператор, а выполнить нужно несколько действий (например, в операторе If).

Условный оператор IF

Общий вид:
IF (условие) <оператор 1>; Else < оператор 2>;
Вариант без Else:
IF (условие) <оператор 1>;
Вариант с составными операторами (когда число команд в ветви 'Then' или 'Else' больше одного):
IF (условие) {
<блок операторов 1>
}
Else{
<блок операторов 2>
}

Если (условие) выполняется, то выполняется <оператор 1>, в противном случае выполняется < оператор 2> (если он имеется). В

качестве операторов 1 и 2 могут использоваться любые исполняемые операторы (в том числе и другие операторы If), а также составные операторы.
Условием может быть операция сравнения или логическое выражение. Операции сравнения:

Си	математика
<	$<$
<=	$\leq$
>	$>$
>=	$\geq$
==	$=$
!=	$\neq$

Пример решения системы уравнений:

$$y = \begin{cases} d + c, \text{если } a > Ъ \\ d - c, \text{если } a \leq Ъ \end{cases}$$

Вариант 1:

```
If (a>b) y=d+c; else y=d-c;
```

Вариант 2:

```
If (a>b) y=d+c;
If (a<=b)     y=d-c;
```

По этой схеме можно реализовывать алгоритм выбора со многими условиями (например, при создании 'меню' в программе).
Пример программы вычисления корней квадратного уравнения:

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
int main()
{
float a,b,c,d,x1,x2,p,q;
printf("BBeflM a,b,c\n");
scanf("%f %f %f ,&a,&b,&c);

if (a==0)
```

```
{    printf("Ошибка ввода!\п");
     printf("уравнение линейное\п");
}
else
     {
     d=b*b-4*a*c;
     if (d>=0)
          {
          x1=(-b+sqrt(d))/(a+a);
          x2=(-b-sqrt(d))/(a+a);
          printf("КорНМ действительные\п");
          printf("x1=%8.2f',x1);
          printf("x2=%8.2f',x2);
          }
     else
          {
          p=(-b+sqrt(-d))/(a+a);
          q=(-b-sqrt(-d))/(a+a);
          printf("КорНМ комплексные\п");
          рпг^("действительная  часть=%8.2г\п",p);
          рпг^("мнимая часть       =%8.2f\n",q);
          }
     }
рпг^("\пДля выхода 0 и Enter!\n");
scanf(" \n");
}
```

Пример выбора со многими условиями:

```
if (n==1)  printf("ОflНН\n");
if(n==2)printf("АВа\n");
if (n==3)  printf("ТрН\n");
……..
if (n==9)  printf(“Девять\n”);
```

Оператор switch

Позволяет выбрать вариант из любого количества вариантов (можно рассматривать как некое развитие оператора If). Общий вид:

```
          =q
switch (выражение) {
     case      v1: <оператор 1>; break;
     case      v2: <оператор 2>; break;
     case      v3: <оператор 3>; break;
     ……..
     case      vN: <оператор N>; break;
     default: <оператор >; break;
     }
```

(выражение) - выражение целого типа. Результат вычисления этого выражения (q) сравнивается с указанными величинами vl - vN ('это могут быть числа или константные выражения). Если значение q совпадает с одной из этих величин, то выполняется соответствующий <оператор>, в противном случае выполняется оператор, указанный после слова default (эта строка может отсутствовать!). Если совпадений нет, а ветвь default отсутствует,

Пример распечатки названий чисел от 0 до 9:

```
int a;
printf("BBeflM a\n");
scanf("%d",&a);

switch (a) {
     case 0: printf("Honb\n"); break;
     case 1: printf("Oflm-i\n"); break;
     case 2: printf("flBa\n"); break;
……..
     case 9: pṙг^О'ДевятьЛп"); break;
     default: pṙгЩ"Ошибка ввода!\п"); break;
        }
```

Циклы

Ведение

Термин 'цикл' определяет одновременно два понятия. Во-первых, это повторяемость действий, и, во-вторых, это фрагмент программы, который выполняется последовательно несколько раз.
В Си циклы организуются операторами **while, do - while** и **for.** (Циклы while и do принято называть циклами с выходом по условию, а цикл for - счетным циклом.) Кроме этого любой цикл можно организовать с помощью только операторов If и goto.
Составной частью любого цикла является т.н. счетчик цикла. В качестве счетчика используется переменная (или несколько переменных), значение которой при каждом повторе действий (итерации цикла) изменяется по определенному закону. Значение счетчика сравнивается с неким заданным значением. По результату сравнения цикл продолжается или завершается. Если счетчик не меняется или отсутствует, или не правильно проверяется, то возникает 'бесконечный' цикл.
Проверка счётчика может производиться в начале цикла или в конце. По этому признаку различают циклы с предусловием (while, for) и с постусловием (do - while).

Оператор цикла DO.

Общий вид:

```
Do                                  // 'делать'
      {
      <оператор 1>;
      <оператор 2>;                 // тело цикла
      ----------------------
      <оператор N>;
      }
while ( условие );                  // 'пока' -условие продолжения цикла
<оператор>;
```

Если при выполнении программы встречается оператор DO, то начинают выполняться операторы, образующие тело цикла (1 - N). После выполнения оператора N следует переход к пункту WHILE и проверка указанного там условия. Если условие выполняется, то происходит возврат программы к пункту DO и повторное выполнение операторов 1-N. Если условие не выполняется, то следует передача управления на <оператор>, следующий за WHILE.

Оператор DO организует выполнение операторов 1 - N до тех пор, пока выполняется указанное условие.

- Указанное условие определяет условие продолжения цикла.

Так как проверка условия производится в конце цикла, то цикл DO выполняется хотя бы один раз.

В качестве условия используется операция сравнения счетчика с неким пороговым значением (обычно ар выражением).

- Счетчик цикла и его изменения организует программист.

Пример суммирования чисел:

```
#include <math.h>
#include <stdio.h>
#include <conio.h>
/*пример накопления суммы целых чисел*/
int main()
{
int a,s;
s=0;
do  {
    clrscr();
    printf("s=%8d\n", s);
    printf("Введи a - !Выход - 0!\n");
    scanf("%d", &a);
    s=s+a;
    }
while (a != 0);
}
```

Оператор цикла while.

Общий вид:

```
While (условие) {                    // пока

                <оператор 1>;
                <оператор 2>;
                ----------------
                <оператор N>;
                }
```

Оператор While повторяет выполнение операторов 1- N до тех пор, пока выполняется указанное <условие>.

- Условие определяет продолжение цикла.

Условием может быть операция сравнения или логическое выражение. Так как истинность условия проверяется в начале каждой итерации, то цикл может не выполняться ни разу.

- Счетчик цикла и его изменения организует программист.

Выход из цикла можно выполнять в любой момент и из любой точки цикла, а вход в цикл - только через заголовок цикла while, т.к. иначе значение счетчика может оказаться неопределенным. Это требование справедливо и для других типов циклов do и for.
Пример табуляции функции:

```
#include <math.h>
#include <stdio.h>
#include <conio.h>
/•программа табуляции функции*/
int main()
{
float x,y,xn,xk,dx;
clrscr();
printf("Введи xn,xk,dx\n");
scanf("%f%f%f",&xn,&xk,&dx);
x=xn;
while (x<=xk)
     {
     y=sin(x);
     printf("x=%5.2f | y=%5.2f\n",x,y);
     x=x+dx;
     }
printf("\n");
printf("„«Для выхода набрать любой символ и Enter!\п");
scanf("\n");
}
```

Оператор цикла for.

Общий вид:

for(<выражение1>;<выражение2>;<выражение3>)
{
<оператор 1>;
<оператор 2>;

<оператор N>;
}

выражения 1,2 и 3 - задают параметры цикла, являются арифметическими выражениями типа int или float.

выражение1	выполняется один раз при инициализации цикла; обычно задаёт начальное значение счётчика,
выражение2	является операцией сравнения, используется для проверки счётчика,
выражение3	арифметическое выражение; выполняется каждую итерацию цикла; обычно используется для изменения счётчика.

Пример:

```
for (i=0; i<10; i++)
        { <оператор>;
        }
```

Пример вычисления факториала n!:

```
int n,l,fkt;
printf("BBefln N\n");
scanf("%d",&n);
fkt=1;
for (i=1; i<=n; i++)     fkt=fkt*l;
```

Пример паблици умножения

```
int i,n,k;
printf("\n");
printf("...|");
   for (n=1; n<=10; n++) printf("%d3",n);
printf("\n");
printfC—+");
   for (n=1; n<=10; n++) printf("—");
for (i=1; i<=10; i++)     {
   printf("\n");
   printf("%d2|",i);
   for (n=1; n<=10; n++)       {
      k=i*n;
      printf("%d3",k);
   }
}
```

```
   |  1  2  3  4  5  6  7  8  9 10
---+------------------------------
 1 |  1  2  3  4  5  6  7  8  9 10
 2 |  2  4  6  8 10 12 14 16 18 20
 3 |  3  6  9 12 15 18 21 24 27 30
 4 |  4  8 12 16 20 24 28 32 36 40
 5 |  5 10 15 20 25 30 35 40 45 50
 6 |  6 12 18 24 30 36 42 48 54 60
 7 |  7 14 21 28 35 42 49 56 63 70
 8 |  8 16 24 32 40 48 56 64 72 80
 9 |  9 18 27 36 45 54 63 72 81 90
10 | 10 20 30 40 50 60 70 80 90100
```

Массивы

Массив, как структура, был разработан для представления в программе большого количества данных с тем, чтобы упростить их именование и обеспечить возможность их обработки в цикле.
Массив - это упорядоченная последовательность однотипных элементов, имеющих одно общее имя (см. рис.).
(Обычно массив создается для сохранения большого числа данных, но может содержать и один элемент.)
По сути, массив представляет собой совокупность обычных переменных, но имена которых образуются по единой схеме: общее имя массива + порядковый номер в квадратных скобках (индекс). Поэтому, элементы массива, в отличие от простых переменных, называются еще индексированными переменными.
Все используемые в программе массивы, как и переменные, должны объявляться в разделах INT, FLOAT или других типов.
Массивы бывают одномерными и многомерными.
Пример объявления одномерных массивов:

```
int  a[100],b[100],c[100];
```

Здесь объявляются три одномерных массива с именами A,B,C, содержащие по 100 элементов целого типа с порядковыми номерами от 0 до 99 **(порядковый номер начинается с 0 !)**.

Работа с массивами. Индексы.

В программе с элементами массивов можно работать как с обычными переменными, учитывая только, что значения индексов должны быть обязательно определены.
Индексом может быть целое число, целая переменная или целое арифметическое выражение:

```
A[6]=0.5;
n=7;
A[n]=A[6];
A[n+2]=A[n];
```

Значения индексов не должны выходить за объявленный интервал!
Обычно массивы обрабатываются в циклах For, при этом счетчик цикла можно использовать в качестве переменного индекса элементов массивов.

Стандартные примеры

Пример 1:
Ввод массива с клавиатуры, вычисление суммы элементов, печать массива.

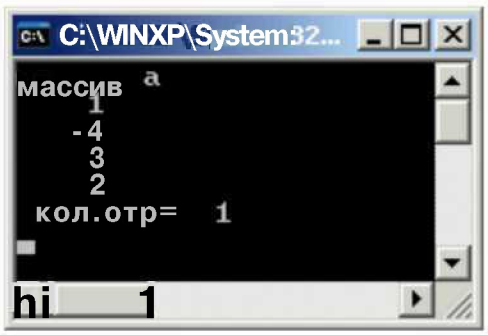
```
#include <stdio.h>
#include <math.h>
#include <conio.h>
main()
{
int s,i,a[4];

// ввод одномерного массива
    printf("Введи  a\n");
    for (i=0; i<4; i++) {
        scanf("%d",&a[i]);
        }
    printf("\n");

// сумма элементов массива
    s=0;
    for (i=0; i<4; i++) {
        s=s+a[i];
        }
// распечатка одномерного массива
    for (i=0; i<4; i++) {
        printf("A[%ld]=%5d\n",i,a[i]);
        }
    printf("\n");

// распечатка сумм,
printf("  s=%5d",s);

printf("\nДля выхода нажми любой символ и Enter!\n");
scanf("_");
}
```

Примеры обработки массивов

Пример 1

Подсчитать количество элементов массива, значения которых меньше 0.

```
int kol_otr,i,a[4]={1 ,-4,3,2};
clrscr();
kol_otr=0;    // количество отрицательных элементов
for (i=0; i<4; i++)
  if(a[i]<0)kol_otr=kol_otr+1;

    // распечатка одномерного массива
printff'MaccHB a\n");
for (i=0; i<4; i++)
  printf(11%5d\n11,a[i]);

printf(IIKon.OTp=%3d\nII,kol_otr);
```

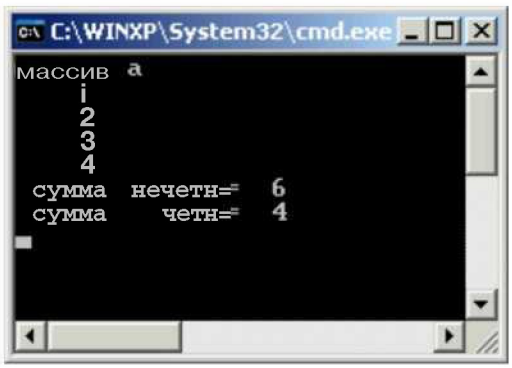


Пример 2

В массиве определить суммы элементов с чётными и нечётными индексами.

```
int s1,s2,i,a[4]={1,2,3,4};

s1=0;    // сумма элементов с нечетными индексами
s2=0;    // сумма элементов с четными индексами

for (i=0; i<4; i++)
  if (i%2!=0)
    s1=s1+a[i];
  else
```

```
      s2=s2+a[i];
clrscr();
        // распечатка одномерного массива
printf("Массив a\n");
for (i=0; i<4; i++)
   printf("%5d\n",a[i]);

printf(" сумма нечетн=%3d\n",s1);
printf(" сумма  четн=%3d\n",s2);
```

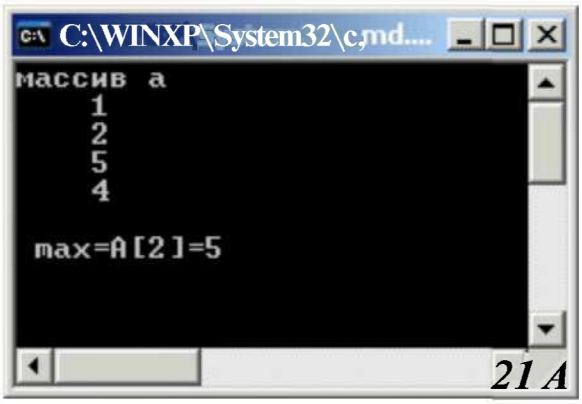


Пример 3

В массиве определить наибольший элемент и его индекс.

```
int ajnax, ijnax, i, a[4]={1,2,5,4};

// вариант 1
        // распечатка одномерного массива
printf("Массив a\n");
for (i=0; i<4; i++)
   printf("%5d\n",a[i]);

a_max=a[0];
i_max=0;
for (i=0; i<4; i++)
   if (a_max<a[i])
     {  a_max=a[i];
        i_max=i;
     }
```

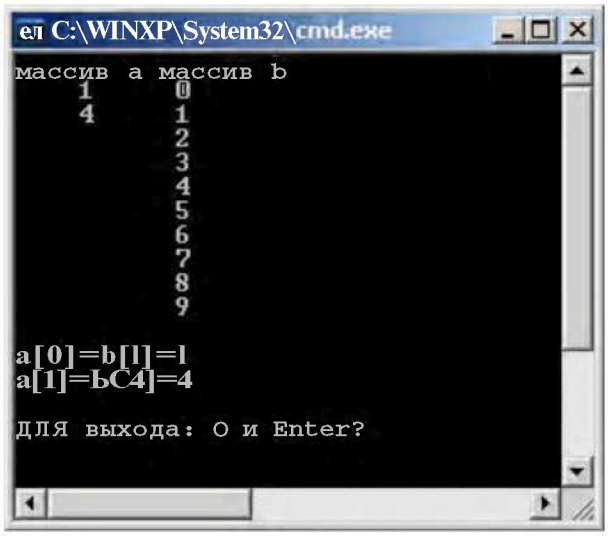
```
printf("\n max=A[%1d]=%d\n",i_max,a_max);
scanf("%d",&i);

// вариант 2   с одной переменной

i_max=0;
for(i=0;i<4;i++)
  if (a[i_max]<a[i]) i_max=i;

printf("max a[%1d]=%5d\n",i_max,a[i_max]);

// вариант 3   выделение max в распечатке таблицы
i_max=0;
for(i=0;i<4;i++)
  if (a[i_max]<a[i])  i_max=i;

for(i=0;i<4;i++)
  { printf("  a[%1d]=%5d",i,a[i]);    //без \n!
    if (i==i_max)
      printf("--> max\n");
    else
      printf("\n");
  }
```



Вложенные циклы

Пример 4
Выделить совпадающие элементы двух массивов

```
intij,a[2]={1,4},b[10]={0,1,2,3,4,5,6,7,8,9};
clrscr();

// распечатка массивов
printf("МассНВ а массив Ь\п");
for(i=0; i<10; i++)
{
if(i<2) printf("%5d %5d\n",a[i],b[i]);
else
   printff"____%5d\n".bfi1);
}
printf("\n");

//  сравнение элементов двух массивов
for(i=0; i<2; i++)
forG=0;j<10; j++)
if (a[i]==b[j]) printf("a[%1d]=b[%1d]=%d\n",ij,a[i]);

pпr^("\пДля выхода: 0 и Enter!\n");
scanfC1,11);
}
```



Пример 5
выполнить сортировку массива по возрастанию

```
int  i,j,  m,  b[10]={6,7,8,9,0,1,2,3,4,5};

      // печать исходного массива
for (i=0;  i<10;  i++)  printf ("%5д",  b[i]);

      // сортировка
for (i=0; i<9; i++)
   for G=0; j<9-i; j++)
      if(bD]>bD+i]){
         m=bO+1];
         b[j+1]=b[j];
         ЬШ=т;   }

      // печать полученного массива
for (i=0; i<10; i++)  printf( "%56",  b[i]);
```

Матрицы

Кроме одномерных (линейных) массивов можно использовать и многомерные массивы, в которых положение элемента задаётся несколькими индексами. Простейшим примером такого массива является матрица - двумерный массив. Положение элемента массива при этом определяется номерами его строки и столбца.

Пример организации матрицы A из 3 строк и 4 столбцов:

A∞ A 01 A 02 A ∞3

A10 A11 A12 A13

A 20 A 21 A 22 A 23

Пример объявления матрицы:

int A[3][4]={1,2,3A5,6,7,8,9,10,11,12}

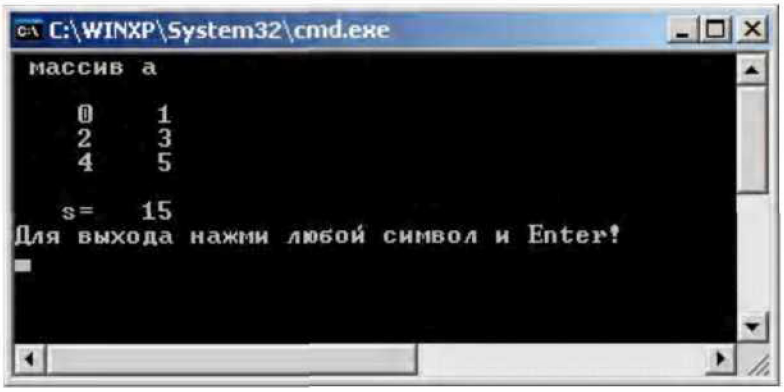
заполнение матрицы производится по строкам.

Пример 1

Ввод, распечатка матрицы по строкам, вычисление суммы элементов всей матрицы.

```
int   s, i, j, a[3][2]={0,1,2,3,4,5}; //заполнение по строкам

// ввод матрицы
рґпЩ'Ъведи a\n");
  for (i=0; i<3; i++)
    for Q=0; j<2; j++)
```

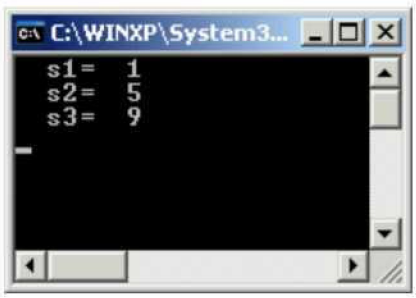
```
        scanf ("rod11, &a[i] 0]);

// сумма элементов массива
s=O;
for (i=0; i<3; i++)
   for Q=0; j<2; j++)
      s = s+ a[i] Ш;

// печать матрицы
clrscr();
printf(" массив a\n\n");
for (i=0; i<3; i++) {
   for Q=0; j<2; j++)
      printf ("%5cГ, a[i] 0] );
   printf("\n");
   }
// печать суммы
printf("11\n...s=%5d11,s);
```

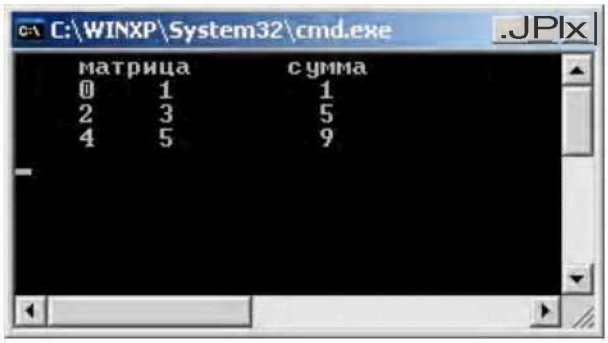


Пример 2
Вычисление сумм элементов матрицы в строках.

```
int s,ij,a[3][2]={0,1,2,3,4,5}; //заполнение по строкам

// сумма элементов матрицы в строках
// вариант 1
for (i=0; i<3; i++)
   {s=0;
```

```
    for G=0; j<2; j++)
      s=s+a[i][j];
    printf(M s%1d=%3d \nM, i+1, s);
  }

// вариант 2

printf("... .матрица. . . . .сумма\п");
for (i=0; i<3; i++)
  {s=0;
    for Q=0; j<2; j++)
      { s=s+a[i][j];
        printf ("%56", a[i][j]);
      }
    printf ("o/oЮcПп11, s);
  }
```





Пример 3

Вычисление произведения элементов в столбцах матрицы с сохранением результатов в одномерном массиве.

```
int p,ij,a[3][2]={0,1,2,3,4,5}; //заполнение по строкам
int b[2];
```

```
// ввод матрицы
/*      ргтЩ'Ъведи a\n");
for(i=0;i<3;i++)
for Q=0; j<2; j++)
  scanf("%d",&a[i]O]);
*/
clrscr();
// произведение элементов матрицы в столбцах
// с заполнением одномерного массива
//          вариант 1     вариант2
for(i=0;i<2;i++){ //for(i=0;i<2;i++) {
    p=1;  //b[i]=1;
for Q=0;j<3;j++)  //for Q=0;j<3;j++)
   P=P*aШИ;  //b[i]=b[i]*a[j][i];
  b[i]=p;       //  }
   }
//важная деталь - красиво оформить результат!
рпг^С'матрица A\n");
for(i=0;i<3;i++) {
printf("        ");
for Q=0; j<2; j++)
  printf("%5d",a[i]D]);
   printfC'Xn");
   }
printfOn");
printf("MaccHB B");
    for(i=0;i<2;i++)
    printf("%5d",b[i]);
```

Матрицы

Кроме одномерных (линейных) массивов можно использовать и многомерные массивы, в которых положение элемента задаётся несколькими индексами. Простейшим примером такого массива является матрица - двумерный массив. Положение элемента массива при этом определяется номерами его строки и столбца.

Пример организации матрицы A из 3 строк и 4 столбцов:

A ∞ A 01 A 02 A ∝з

Аю Аи A12 A13

A 20 A 21 A 22 A 23

Пример объявления матрицы:

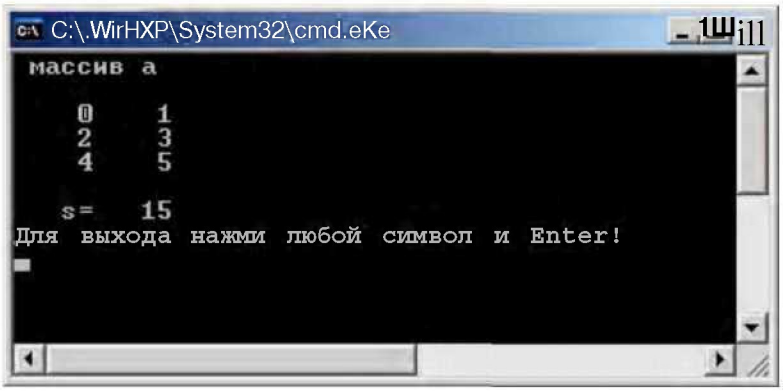
int A[3][4]={1,2,3,4,5,6,7,8,9,10,11,12}

заполнение матрицы производится по строкам.

Пример 1
Ввод, распечатка матрицы по строкам, вычисление суммы элементов всей матрицы.

```
int   s, i, j, a[3][2]={0,1,2,3,4,5}; //заполнение по строкам

// ввод матрицы
рṅпЩ'Ъведи a\n");
  for (i=0; i<3; i++)
    for Q=0; j<2; j++)
      scanf ("rod[11], &a[i] 0]);

// сумма элементов массива
s=0;
for (i=0; i<3; i++)
  for Q=0; j<2; j++)
    s = s+ a[i] Ш;
```

```
// печать матрицы
clrscr();
printf(" массив a\n\n");
for (i=0; i<3; i++) {
   for Q=0; j<2; j++)
      printf ("%56", a[i] 0] );
   printf(M \nM);
   }
// печать суммы
printf(II\n...s=%5dII,s);
```

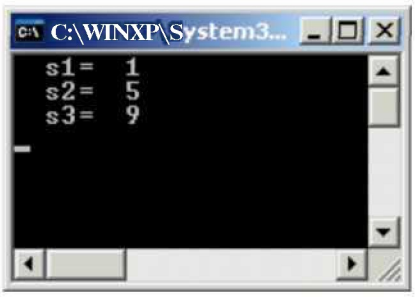


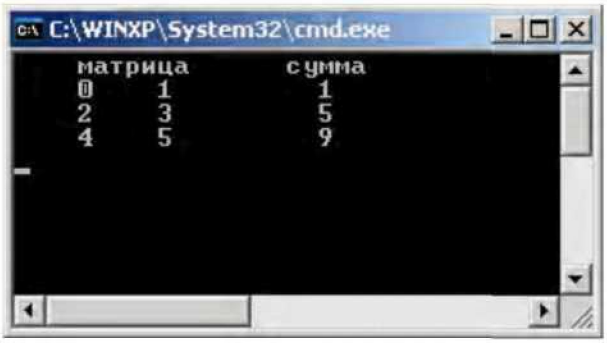
Пример 2

Вычисление сумм элементов матрицы в строках.

```
int s,ij,a[3][2]={0,1,2,3,4,5}; //заполнение по строкам

// сумма элементов матрицы в строках
// вариант 1
for (i=0; i<3; i++)
   {s=0;
      for Q=0; j<2; j++)
         s=s+a[i][j];
      printf(M s%1d=%3d \n", i+1, s);
   }

// вариант 2

printf("....матрица.....сумма\n");
```

```
for (i=0; i<3; i++)
  { s=0;
    for 0=0; j<2; j++)
      { s=s+a[i][j];
        printf("%5d",a[i][j]);
      }
    printf ("%10d\n", s);
  }
```





Пример 3

Вычисление произведения элементов в столбцах матрицы с сохранением результатов в одномерном массиве.

```
int p,ij,a[3][2]={0,1,2,3,4,5}; //заполнение по строкам
intb[2];
// ввод матрицы
/*      рґтЩ'Ъведи a\п[M]);
for (i=0; i<3; i++)
for Q=0; j<2; j++)
  scanf([M]%d[11],&a[i]O]);
*/
clrscr();
// произведение элементов матрицы в столбцах
```

```
// с заполнением одномерного массива
//            вариант 1      вариант2
for (i=0; i<2; i++) {  //for (i=0; i<2; i-++) {
     p=1;     lib[i]=1;
for Q=0; j<3; j++)    //for Q=0; j<3; j-++)
    P=P*aШИ;  //b[i]=b[i]*a[j][i];
   b[i]=p;          //   }
    }
//важная деталь - красиво оформить результат!
рпг^("матрица A\n");
for (i=0; i<3; i-++) {
printf(M          ");
for Q=0; j<2; j-++)
   printf(11%5d"ja[i]D]);
    printfC'Xn11);
    }
printfC'Xn11);
printf(MMaccHB  B");
     for (i=0; i<2; i++)
     printf(11%5d11Jb[i]);
```