МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение
Высшего профессионального образования
Алтайский государственный технический
университет им. И.И.Ползунова

Е.В.ЕГОРОВА

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ СИ

Учебное пособие

Барнаул 2014

УДК 681.3

Егорова Е.В. Программирование на языке Си. Учебное пособие / Алт. госуд. технич. ун-т им. И.И.Ползунова. - Барнаул: 2014. - 184 с.

Пособие предназначено для приобретения практических навыков алгоритмизации и программирования на языке высокого уровня Си.

Цель пособия - дать конкретную информацию для самостоятельной работы студента.

Пособие предназначено для студентов направления 231000 "Программная инженерия".

Рецензент: С.А.Кантор - зав.кафедрой Прикладной математики АлтГТУ.

ВВЕДЕНИЕ

Данное учебное пособие предназначено для практического изучения основ алгоритмизации и программирования на стандартном языке Си. Цель учебника не в том, чтобы дать традиционное подробное описание языка Си, а в том, чтобы позволить студенту самостоятельно, в идеале вообще без участия преподавателя, изучить основы алгоритмизации и основы программирования на языке Си. Рассматривается методика структурированного, корректного программирования. Программы, которые приводятся в качестве примеров, ориентированы в первую очередь на то, что быть понятными, а не на то, чтобы быть максимально эффективными. В книге нет четкого разделения теории и практики. Все теоретические положения сразу же связываются с практическим программированием и иллюстрируются примерами решения соответствующих задач. По мнению автора, такой "задачный подход" позволяет сделать познание более активным и целенаправленным, а, как следствие, и более результативным.

Учебник ориентирован на тех, кто знаком с основами информатики на уровне простейшей практической работы за компьютером. Конкретную интегрированную среду разработки Си-программ студент может выбрать по своему усмотрению, так как изложение материала от среды разработки практически не зависит. В книге рассматривается классический вариант языка Си («стандартный СИ»), который не зависит от реализаций и имеет место практически для любой конкретной среды программирования.

Структура учебника следующая. Пособие рассчитано на работу в двух семестрах по 16-17 учебных недель каждый. Основной материал разбит на 8 модулей. Изучение одного модуля занимает примерно 4 недели. Структура модуля следующая:

- теоретический материал по одной или нескольким темам, который сразу же по тексту иллюстрируется решением практических задач;
- задание к лабораторной работе, выполнение которой на компьютере позволит закрепить знания, полученные в теоретических разделах модуля;
- список контрольных вопросов по данному модулю, который дает дополнительную проверку полученных знаний.

В пределах модуля студенту предлагается следующий порядок работы. Вначале надо подробно изучить теоретический материал и по ходу изложения разобрать все примеры. Причем желательно не просто просмотреть примеры, а постараться воспроизвести их на бумаге или на компьютере, не заглядывая в оригинал. Затем следует выполнить на компьютере лабораторную работу, что позволит практически закрепить полученные знания. Выполнение лабораторной работы пропускать нельзя! Интуитивное ощущение "я знаю и смогу это сделать" еще не означает, что Вы без проблем справитесь с этим на практике. К лабораторной работе прилагается набор вариантов заданий. Обычно достаточно выполнения одного варианта, но если Вы не чувствуете себя уверенно по данной теме, выполните несколько вариантов лабораторной работы. В конце модуля представлены контрольные вопросы. Отвечая на них, можно проверить, насколько полно усвоен теоретический материал данного модуля.

1 ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ СИ

Цель изучения данного модуля - познакомиться с основами программирования на языке Си, а также приобрести опыт разработки на языке Си простейших программ линейной структуры. Но прежде, чем начать изучение непосредственно языка, следует познакомиться с историей его создания и с классификацией основных версий.

История создания. Почему язык называется СИ ?

Си - это язык программирования, широко распространенный в настоящее время среди системных программистов и разработчиков прикладных программ.

Си появился как инструментальный язык в процессе работы над операционной системой (ОС) UNIX Кена Томпсона из научно-исследовательской лаборатории фирмы Bell Telephone Laboratories. В 1969 г. Томпсон начал работать над этой системой для мини-ЭВМ PDP-7 фирмы DEC. В 1972 г. Деннисом Ритчи и Кеном Томпсоном был создан язык программирования Си. В нем сочетались лучшие свойства языка Ассемблер и языков высокого уровня. От Ассемблера были взяты гибкие и эффективные средства работы с памятью, от языков высокого уровня - широкий набор управляющих конструкций, возможность работы со сложными структурами данных, гибкие средства ввода-вывода. Сегодня компилятор с языка Си есть практически на любом компьютере в любой ОС. 95% ОС UNIX и почти все прикладное обеспечение для нее написано на языке Си.

Название языка соответствует третьей букве латинского алфавита "С". Совпадение не случайно и имеет свою историю. Один из языков, появившихся в 70-х годах, назывался APL (A Programming Language) - язык программирования А. Так была занята первая буква алфавита. Во время работы над ОС UNIX для PDP-7 был создан язык программирования B, который оказал сильное влияние на следующий язык, разработанный Ритчи и Томпсоном для тех же целей. Показывая эту преемственность, авторы назвали язык С.

Версии языка Си. Турбо-Си. Си++.

Деннис Ритчи разработал язык Си для компьютера типа PDP фирмы DEC, который работал под управлением операционной системы UNIX. Эта версия языка Си приобрела большую известность благодаря классическому труду Кернигана и Ритчи "Язык программирования СИ". Данную основополагающую классическую версию языка принято называть K&R - по первым буквам фамилий классиков.

Далее язык Си продолжал развиваться, и скоро встала проблема стандартизации всех тех улучшений и доработок, которые реализовались при развитии Си. Для этого в 1983 г. в институте ANSI (Американский национальный институт стандартов) был создан специальный комитет. Этот комитет утвердил ANSI-стандарт Си, который стал надмножеством стандарта K&R. Комитет не просто предложил стандарт, а еще подготовил 79 тестов, позволяющих быстро и эффективно определить степень совместимости со стандартом ANSI для любого Си-компилятора.

Калифорнийская фирма "Борланд" (Borland Inernational CA) разработала свой компилятор для IBM PC - Турбо Си. В этой разработке:

1) реализован ANSI-стандарт (выполняется 60 тестов из 79);

2) реализована специальная Турбо-среда, которая максимально автоматизирует процесс разработки программ.

В начале 80-х годов в Bell Laboratory Бьерном Страуструпом в результате дополнения и расширения языка Си был создан новый, по сути, язык, получивший название "С с классами". В 1983 г. это название было заменено на "С++". Это название указывает на эволюционную природу перехода от "С" к "С++", так как "++" - это операция приращения в Си (увеличение на 1). С++ - это мощная, объектно-ориентированная версия языка, обычный С сохранен в С++ как подмножество (есть очень немногие исключения).

Следует иметь в виду, что кроме Borland С++ существует еще версия С++, разработанная фирмой Microsoft.

В данной книге описываются основные возможности классического, или

стандартного, языка Си, который не зависит от реализаций и имеет место практически для любой конкретной среды программирования.

Характеристика языка Си

Си - это язык программирования общего назначения, хорошо известный своей эффективностью, экономичностью и переносимостью. Указанные преимущества Си обеспечивают хорошее качество разработки почти любого вида программного продукта. Использование Си в качестве инструментального языка позволяет получать быстрые и компактные программы. Во многих случаях программы, написанные на Си, сравнимы по скорости с программами, написанными на языке ассемблера. При этом они имеют лучшую наглядность и их более просто сопровождать.

Си сочетает эффективность и мощность в относительно малом по размеру языке. Хотя Си не содержит встроенных компонент языка, выполняющих ввод-вывод, распределение памяти, манипуляции с экраном или управление процессами, тем не менее системное окружение Си располагает библиотекой объектных модулей, в которой реализованы подобные функции. Библиотека поддерживает многие из функций, которые требуются. Это решение позволяет изолировать языковые особенности от специфики процессора, на котором выполняется результирующая программа. Строгое определение языка делает его независимым от любых деталей операционной системы или машины. В то же время программисты могут добавить в библиотеку специфические системные программы, чтобы более эффективно использовать конкретные особенности машины.

Си обеспечивает полный набор операторов структурного программирования и предлагает необычно большой набор операций. Многие операции Си соответствуют машинным командам, и поэтому допускают прямую трансляцию в машинный код. Разнообразие операций позволяет выбирать их различные наборы для минимизации результирующего кода.

Си поддерживает указатели на переменные и функции. Указатель на объект программы соответствует машинному адресу этого объекта. Посредством разумного использования указателей можно создавать эффективно выполняемые программы, так как указатели позволяют ссылаться на объекты тем же самым путем, как это делает машина. Си поддерживает арифметику указателей, и тем самым позволяет осуществлять непосредственный доступ и манипуляции с адресами памяти.

В своем составе Си содержит препроцессор, который обрабатывает текстовые файлы перед компиляцией. Среди его наиболее полезных приложений при написании программ на Си являются: определение программных констант, замена вызовов функций аналогичными, но более быстрыми макросами, условная компиляция. Препроцессор не ограничен процессированием только исходных текстовых файлов Си, он может быть использован для любого текстового файла.

Достоинства языка Си

Эффективность. По компактности и скорости выполнения программы на языке Си приближаются к программам, написанным на Ассемблере.

Мощность. Язык Си содержит большой набор современных управляющих конструкций и способов представления данных.

Способность поддерживать технологию структурного программирования. Управляющие конструкции (операторы) языка Си согласуются с технологией структурного программирования; Си обеспечивает полный набор операторов структурного программирования.

Способность поддерживать разработку модульных программ. Основной программной единицей является функция, есть возможность создавать многофайловые программы.

Мобильность. Программы, написанные на языке Си, могут быть легко перенесены на другой компьютер в другую операционную систему.

Лаконичность. Программы на языке Си получаются короче, чем на других языках программирования, при этом наглядность и ясность программы не теряется.

Недостатки языка Си

Главный недостаток: язык предъявляет высокие требования к квалификации использующего его программиста. Так, Си предоставляет программисту очень большую свободу в программировании, однако бесконтрольное использование этой свободы часто приводит к ошибкам.

1.1 ОСНОВНЫЕ ЭТАПЫ РЕШЕНИЯ ЗАДАЧ НА ЭВМ

Цель изучения данного материала - познакомиться с общей технологией разработки программ на ЭВМ. В принципе, эта технология не зависит от языка программирования. Ниже приводятся примеры на двух языках: Паскаль и Си.

Обычно выделяются следующие этапы решения задачи на ЭВМ:

1) постановка задачи;

2) для математических задач - выбор математического метода решения задачи и окончательная математическая постановка задачи (построение математической модели);

3) разработка алгоритма решения и наглядное изображение алгоритма в одной из принятых форм;

4) написание программы (запись алгоритма на языке программирования);

5) ввод текста программы в ЭВМ;

6) получение рабочей программы;

7) тестирование и отладка программы;

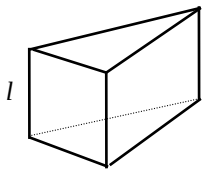
8) решение задачи на ЭВМ;

9) анализ результатов выполнения программы и оформление их в виде таблиц, графиков и т.д.;

10) оформление отчета о проделанной работе.

Рассмотрим выполнение всех этих этапов на примере. Пусть требуется решить на ЭВМ следующую задачу: найти объем прямой треугольной призмы.

1.1.1 Математическая постановка задачи



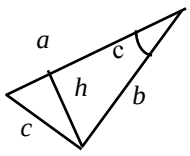
Объем прямой треугольной призмы вычисляется по формуле:

$$V=S*l, \qquad (1)$$

где S - площадь основания призмы;

l - высота призмы.

1.1.2 Выбор математического метода решения задачи



Основанием рассматриваемой призмы является треугольник. Для вычисления площади треугольника существует несколько формул:

$$S=(a*h)/2=(a*b*\sin C)/2=\sqrt{p*(p-a)*(p-b)*(p-c)}.$$

Будем использовать последнюю формулу:

$$S=\sqrt{p*(p-a)*(p-b)*(p-c)}, \qquad (2)$$

где a,b,c - стороны треугольника,

p=(a+b+c)/2 - полупериметр треугольника.

Окончательная формулировка задачи: разработать программу для вычисления величины V по формулам (1), (2), если заданы значения a,b,c,l.

1.1.3 Алгоритм решения задачи

Алгоритм решения задачи - это система правил (в виде последовательности арифметических и логических правил), однозначно определяющих процесс преобразования исходных данных в искомые результаты, то есть за конечное число шагов приводящих к решению поставленной задачи.

При составлении алгоритма следует использовать метод пошаговой разработки. Суть метода: алгоритм разрабатывается "сверху вниз", начиная со списка входных и выходных данных; на каждом шаге принимается небольшое число решений, приводящих к постепенной детализации алгоритма.

Итак, на начальном этапе необходимо разобраться с данными, используемыми в задаче.

В вычислительных задачах чаще всего используются два вида данных: числовые константы и переменные.

Константа не изменяет своего значения при выполнении программы. В тексте программы задается или явно в виде своего значения (например, числом: 2; 0.5; -1.73 и т.д.), или обозначается именем, которое должно быть предварительно объявлено (например, в языке Паскаль - в разделе объявления констант const, в языке Си - с помощью директивы define).

Переменная может изменять свое значение при выполнении программы. В тексте программы любая переменная обозначается именем, которое должно быть до использования объявлено с указанием типа. По возможности стремятся к тому, чтобы в условии задачи, в описании алгоритма и в тексте программы одинаковые по смыслу переменные обозначались одним именем (или похожими именами, если, например, в условии задачи используется греческий алфавит, а текст программы оформляется с помощью латинского алфавита).

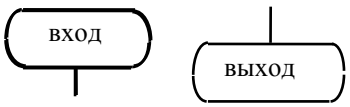
Переменные должны иметь конкретный тип, и этот вопрос программист должен продумать до разработки алгоритма, опираясь на условие задачи. Тип в первую очередь определяет, какие значения допустимо присваивать переменной в программе. В языках программирования обычно предусмотрено три основных типа данных. Это - два числовых типа: целый (integer - в Паскале, int - в Си) и вещественный (real - в Паскале, float - в Си) и символьный тип (char).

Опишем данные, используемые в нашей задаче. Это переменные a, b, c, l, V, p, S. Все переменные по смыслу задачи должны быть вещественного типа, то есть в программе они могут принимать значения дробных чисел.

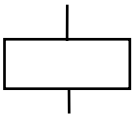
Исходные данные: a,b,c - стороны треугольника,
l - высота призмы.
Результаты: V - объем призмы.
Промежуточные данные: p - полупериметр треугольника,
S - площадь треугольника.

Словесное описание алгоритма обычно громоздко и неудобно для последующего программирования. Для наглядного графического изображения алгоритма часто используется структурная схема (СС). СС состоит из отдельных блоков (геометрических фигур), соединенных между собой. Форма геометрической фигуры характеризует функции, выполняемые соответствующим блоком. Внутри фигуры словесно или с помощью формул эти функции конкретизируются.

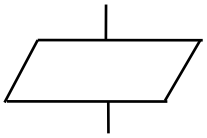
Основные элементы СС



Блок начала и блок конца СС.

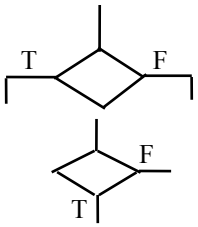


Вычислительный блок
(внутри блока записываются формулы, по которым проводятся вычисления).



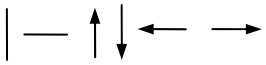
Блок ввода-вывода: ввод данных в оперативную память ЭВМ или вывод данных на экран или печатающее устройство (внутри блока записываются: слово "ввод" или "вывод"; данные, которые надо ввести или вывести, соответственно).

Или



Условный логический блок
(внутри блока записывается логическое выражение, значение которого проверяется; если логическое выражение истинно, программа выполняется по веточке "T" (true - истина, да), если логическое выражение ложно - по веточке "F" - (false - ложь, нет).

Указатель перехода к выполнению блока с номером, записанным в кружке.

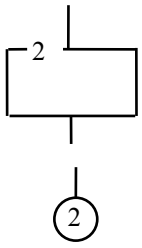


Линии соединения блоков

СС выполняется, начиная с блока "вход", в порядке, указанном стрелкой, или сверху вниз (при отсутствии стрелки). Заканчивается выполнение блоком "выход".

Для записи комментарий в СС и для ссылки из одного блока в другой можно помечать блоки (присваивать им метки, или номера). Номер - целое положительное число, записывается в левом верхнем углу блока. Одинаковые номера в СС встречаться не должны.

Пример.



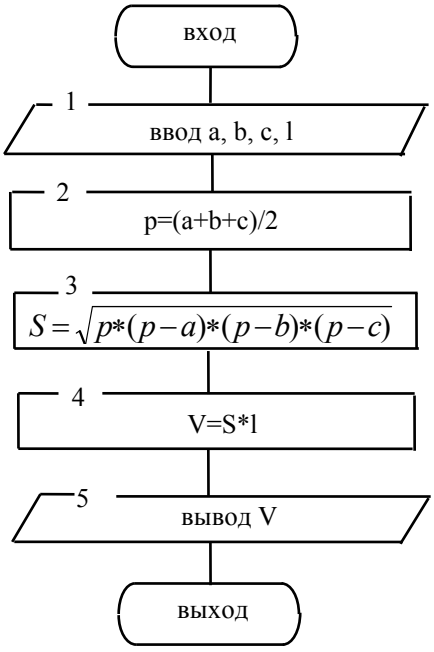
Данный блок имеет номер 2.

Эта ссылка на блок 2, то есть переход к выполнению блока с номером 2.

Процесс решения задачи на ЭВМ, то есть процесс обработки информации на ЭВМ, называется вычислительным процессом (ВП). Все ВП и, соответственно, алгоритмы и структурные схемы, реализующие эти ВП, подразделяются на линейные, разветвляющиеся и циклические. Любой ВП и, соответственно, любой алгоритм представляет собой комбинацию трех этих элементов.

Структурная схема линейного ВП не содержит разветвлений, все блоки выполняются последовательно один за другим сверху вниз. Структурная схема разветвляющегося ВП содержит разветвление, после которого идет однократное выполнение одной из двух "параллельных" ветвей алгоритма. Такое разветвление в структурной схеме изображается условным логическим блоком в виде ромба с двумя выходами влево и вправо. Структурная схема циклического ВП предусматривает многократное выполнение некоторой последовательности блоков. В структурной схеме цикл изображается с помощью условного логического блока в виде ромба с двумя выходами: вниз и влево или вниз и вправо, причем по одному из выходов идет возврат «наверх», то есть зацикливание.

Ниже представлена СС рассматриваемой задачи (обратите внимание, что это линейный алгоритм); рядом, в качестве пояснения - словесное описание алгоритма.



1. Ввод в память ЭВМ значений переменных a,b,c,l.
2. Вычисление значения переменной p по формуле: p=(a+b+c)/2.
3. Вычисление значения переменной S по формуле: S= p*(p-a)*(p-b)*(p-c).
4. Вычисление значения переменной V по формуле: V=S*l.
5. Вывод значения переменной V

1.1.4 Программирование

Программирование - это запись разработанного алгоритма на языке программирования (ЯП). В ЯП для указания выполняемых действий служит оператор. Программа на ЯП - это последовательность операторов, оформленная по специальным правилам.

Писать программу следует в соответствии со СС. Каждому блоку СС соответствует обычно один или несколько операторов ЯП.

Ниже приведены две программы для рассматриваемой задачи: на языке Паскаль и на языке Си.

```
(* Программа на языке Паскаль *)
program vtp(i,o);
   var a,b,c,l,p,s,v:real;
begin
  read(a,b,c);
  read(l);
  p:=(a+b+c)/2;
  s:=sqrt(p*(p-a)*(p-b)*(p-c));
  v:=s*l;
  write(v)
end.
```

```
/* Программа на языке Си*/
#include <stdio.h>
#include <math.h>
int main()
{float   a,b,c,l,p,s,v;
 scanf("%f%f%f",&a,&b,&c);
 scanf("%f",&l);
  p=(a+b+c)/2;
```

```
  s=sqrt(p*(p-a)*(p-b)*(p-c));
  v=s*l;
 printf(“%f”,v)
}
```

Следует отметить, что, хотя приведенные программы и являются правильными, для реального использования они мало пригодны, так как у них не очень удобный интерфейс. Интерфейс - это способ взаимодействия программы с конечным пользователем, который использует программу. Интерфейс должен быть удобным и понятным, а в данной программе отсутствуют простейшие сообщения о том, например, что и в каком порядке требуется вводить, о том, что получено в результате работы программы. Ниже приведены эти же программы, в которых выдача пользователю подобных сообщений предусмотрена.

```
(* Программа на языке Паскаль *)
program vtp(i,o);
   var a,b,c,l,p,s,v:real;
begin
  writeln('ВЫЧИСЛЕНИЕ ОБЪЕМА ПРЯМОЙ ТРЕУГОЛЬНОЙ ПРИЗМЫ');
  write('ВВЕДИТЕ a,b,c -СТОРОНЫ ТРЕУГОЛЬНИКА: ');
  read(a,b,c);
  write('ВВЕДИТЕ l - ВЫСОТУ ПРИЗМЫ: ');
  read(l);
  p:=(a+b+c)/2;
  s:=sqrt(p*(p-a)*(p-b)*(p-c));
  v:=s*l;
  writeln('ОБЪЕМ ПРИЗМЫ v=',v)
end.
```

```
/* Программа на языке Си*/
#include <stdio.h>
#include <math.h>
int main()
{float   a,b,c,l,p,s,v;
 printf(“ВЫЧИСЛЕНИЕ ОБЪЕМА ПРЯМОЙ ТРЕУГОЛЬНОЙ ПРИЗМЫ\n”);
 printf(“ВВЕДИТЕ a,b,c -СТОРОНЫ ТРЕУГОЛЬНИКА: “);
 scanf(“%f%f%f”,&a,&b,&c);
 printf(“ВВЕДИТЕ l - ВЫСОТУ ПРИЗМЫ: “);
 scanf(“%f”,&l);
  p=(a+b+c)/2;
  s=sqrt(p*(p-a)*(p-b)*(p-c));
  v=s*l;
 printf(“ОБЪЕМ ПРИЗМЫ v=%f\n”,v)
}
```

1.1.5 Ввод текста программы в ЭВМ

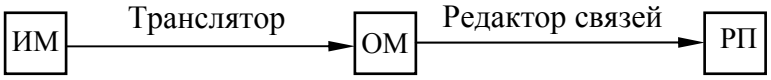
Ввод и редактирование текста исходной программы, его компиляция, отладка и выполнение в настоящее время проводятся программистом в рамках какой-либо интегрированной среды программирования (например, в среде Turbo Pascal - для языка Паскаль, в среде MinGW Developer Studio - для языка Си). Интегрированная среда существенно облегчает рутинную работу программиста, предоставляя в его распоряжение, например, редактор текстов, отладчик программ и др.

Этот и последующие этапы решения задач на ЭВМ связаны с навыками работы с вычислительной техникой и с конкретной интегрированной средой. При работе на персональном компьютере данные (текст программы, исходные данные) вводятся с помощью клавиатуры. Вводимые данные, а также результаты выполнения программы, отображаются на экране дисплея.

1.1.6 Получение рабочей программы

Текст программы на языке программирования называется исходным модулем (ИМ). ИМ не может непосредственно выполняться на ЭВМ. С помощью транслятора (компилятора) и редактора связей должна быть создана рабочая программа (РП), соответствующая данному ИМ. РП - это программа, готовая к выполнению на ЭВМ.

Ниже схематично изображен порядок создания РП.



Трансляция (компиляция) ИМ включает в себя два действия: анализ - определение правильности записи программы в соответствии с правилами языка программирования и синтез - генерирование соответствующей программы на машинном языке, которая называется объектным модулем (ОМ). В процессе анализа компилятор находит синтаксические ошибки в программе и выдает сообщения о них. После устранения ошибок пользователь должен вновь откомпилировать программу. Если ошибок нет, то компилятор вырабатывает ОМ, эквивалентный ИМ.

Программист может использовать в своей программе другие программы. Редактор связей соединяет все ОМ и создает готовую для выполнения на ЭВМ РП.

Транслятор и редактор связей - это системные программы, входящие в систему программирования и предоставляемые пользователю в готовом виде; в случае необходимости пользователь вызывает их для выполнения. Для этого существуют специальные команды интегрированной среды разработки программ.

1.1.7 Тестирование и отладка программы

Отсутствие синтаксических ошибок еще не означает, что программа написана правильно. Существуют ошибки, которые обнаруживаются только на этапе выполнения программы. Это логические ошибки, являющиеся результатом неправильно составленного алгоритма, и описки, например, в арифметических выражениях. Так, в рассматриваемой выше программе вместо оператора "v=s*l;" мог быть ошибочно записан оператор "v=s+l;". С точки зрения синтаксиса эта программа была бы верна, но на самом деле она будет выдавать неверный результат.

Для обнаружения в программе как можно большего количества ошибок этапа выполнения программист проводит тестирование программы. Тестирование заключается в составлении набора тестов и выполнении программы на этих тестах. Простейший тест - это набор исходных данных, для которых известен результат.

Тест, приводящий к неправильному выполнению программы, свидетельствует о наличии в программе ошибки. Далее необходимо локализовать местонахождение ошибки и исправить ее. Этот процесс называется отладкой. При исправлении одних ошибок в программу могут вноситься другие ошибки, поэтому исправленную программу необходимо вновь подвергнуть тестированию и т.д. В результате большинство (но, возможно, не все) ошибки будут удалены из программы, после чего можно проводить решение задачи на ЭВМ.

Тестирование и отладка часто проводятся в рамках интегрированной среды с помощью специальной системной программы, которая называется встроенным отладчиком и запускается обычно с помощью команды интегрированной среды Debug.

1.1.8 Решение задачи на ЭВМ и анализ результатов

Этот этап проводится для реальных прикладных задач и заключается в следующем: готовятся исходные данные для программы; программа запускается, производит необходимые действия и выдает полученные результаты. Пользователь анализирует эти результаты и оформляет их в надлежащем виде.

1.1.9 Оформление отчета о проделанной работе

Отчет о проделанной работе (в частности, отчет по лабораторной работе) должен содержать следующие разделы:

1) титульный лист отчета (заголовок работы, фамилия разработчика и другие сведения);

2) формулировка задачи;

3) математическая постановка задачи (при необходимости);

4) обозначение входных, выходных и промежуточных данных с указанием их типа;

5) алгоритм решения задачи, представленный в виде структурной схемы или словесного описания;

6) текст программы на языке программирования;

7) набор тестов для проверки программы и результаты отладки программы;

8) результаты выполнения программы на заданных исходных данных (для реальных прикладных программ).

Обозначение входных, выходных и промежуточных данных можно (по согласованию с преподавателем) не оформлять как отдельный раздел отчета, а оформить в виде комментариев в разделе описаний в тексте программы.

1.2 НАЧАЛЬНЫЕ СВЕДЕНИЯ О ЯЗЫКЕ ПРОГРАММИРОВАНИЯ СИ

1.2.1 Алфавит

Алфавит - совокупность допустимых в языке символов или групп символов, рассматриваемых как единое целое. В языке Си все компоненты формируются из множества символов стандарта ASCII.

1.2.2 Идентификаторы

Идентификаторы (имена) в языке программирования используются для обозначения имен переменных, функций, меток и т.п., применяемых в программе. Идентификаторы могут состоять из любого числа латинских букв, символов подчеркивания "_", цифр, но должны начинаться с буквы или символа подчеркивания. При этом надо учесть, что с символа подчеркивания начинаются имена системных зарезервированных переменных и констант. В библиотечных функциях также часто используются имена, начинающиеся с этого символа. Это делается в предположении, что пользователи вряд ли будут применять этот символ в качестве первого символа своих идентификаторов. Старайтесь не

использовать имен, начинающихся с символа подчеркивания, и Вам удастся избежать возможных конфликтов и взаимопересечений с множеством библиотечных имен.

В языке Си некоторые идентификаторы употребляются как служебные (ключевые, зарезервированные) слова (keywords), которые имеют специальное значение для компилятора. Их употребление строго определено, и эти слова не могут использоваться иначе.

В Си большие и маленькие латинские буква считаются различными. Например, "index" и "Index" - различные имена. По традиции в идентификаторах в Си-программах обычно используются маленькие буквы.

Замечание 1. В некоторых версиях Си разрешено использовать в идентификаторе знак доллара "$".

Замечание 2. Хотя идентификаторы в C++ могут быть любой длины, некоторые компиляторы Си различают имена только по первым 8 или 32 знакам.

1.2.3 Функции. Структура программы на языке Си

Программа на Си - это набор функций (процедура в Си является частным случаем функции).

Главная функция программы должна иметь имя main (главная), и именно с нее начинается выполнение программы. В программе на Си должна быть только одна функция с именем main.

Упрощенная общая форма записи функции:

```
<имя функции>()
{
 <внутренние описания>;
 <действия>;
}
```

Круглые скобки после имени обязательны. Они показывают, что это имя функции, а не имя переменной.

Тело функции размещается в фигурных скобках и включает в себя описания переменных и выполняемые над ними действия.

Ниже для знакомства с языком Си приведены два примера готовых Си-программ.

Пример 1. Приведенная ниже Си-программа состоит из 7 строк. Строки пронумерованы справа в виде комментариев.

```
/* Моя первая программа на языке Си */   /* 1 */
#include <stdio.h>                         /* 2 */
                                           /* 3 */
int main()                                 /* 4 */
{                                          /* 5 */
  printf("Hello, friend!\n");              /* 6 */
}                                          /* 7 */
```

Результат работы этой программы - на экране в текущей позиции появится сообщение:

Hello, friend!

и курсор перейдет на начало следующей строки.

Далее построчно разъясняются основные элементы этой программы.

1 строка - комментарии. Комментарии - это любая последовательность символов, заключенная между знаками "/*" и "*/".

2 строка - директива препроцессора. Препроцессор - это системная программа, осуществляющая выполнение некоторых директив до этапа компиляции программы.

3 строка - оставлена пустой для лучшего восприятия текста программы.

4 строка - заголовок определения функции. "main" - это обязательное имя основной

функции (так называемой основной программы). Круглые скобки после слова "main" (в принципе, не обязательно пустые) должны быть всегда. Слово “int” перед именем функции – это тип результата, выдаваемого этой функцией.

5 строка - знак "{" означает начало тела функции.

6 строка - это единственный оператор тела основной функции - оператор вызова стандартной функции вывода printf, которая выводит на экран сообщение "Hello, friend!" и переводит курсор в начало следующей строки (по символу "\n"). Символ ";" является частью оператора и обязателен в конце оператора.

7 строка - знак "}" означает конец тела функции, в нашем случае - всей программы.

Пример 2.

```
/* Пример программы на языке Си. Сложение двух целых чисел. */
#include <stdio.h>
int main()
{
  int a,b,sum;              /* Описание трех целых переменных a,b,sum */
  printf("\nВведите два целых числа: "); /* Вывод сообщения об ожидаемом вводе */
  scanf("%d %d",&a,&b);                  /* Ввод значений переменных  a,b          */
  sum=a+b;                  /* Оператор присваивания: сложить a и b; результат - в sum */
  printf("a+b=%d\n",sum);           /* Вывод результата - значения переменной sum */
  return 0;
}
```

Замечание 1. Перед словом "main" указан тип выдаваемого функцией результата, этот результат выдается в конце программы оператором return. Нулевой результат – это показатель успешного завершения программы. В принципе, указание оператора return в данном случае не обязательно, нулевой результат в случае успешного завершения будет выдаваться по умолчанию.

Замечание 2. Ввод-вывод в Си осуществляется посредством обращения к стандартным функциям ввода-вывода. В данном примере используются стандартные функции printf и scanf для вывода и ввода, соответственно.

1.2.4 Описание переменных

Каждая переменная, которая используется в программе на Си, должна быть описана ровно один раз. Описание начинается с указания типа, затем через запятую перечисляются имена переменных этого типа. В конце ставится точка с запятой.

Пример. Описание переменных a,b типа char (знаковый тип), x - типа int (целый тип), t - типа float (тип с плавающей точкой).

```
char a,b;
int x;
float t;
```

1.2.5 Определение констант

Определение констант обеспечивается препроцессором языка Си и имеет вид:

```
#define <имя константы> <значение константы>
#define <имя константы> <имя константы>
#define <имя константы> (<выражение из констант>)
```

Знак "#" записывается в первой позиции строки.

Примеры определения констант:

```
#define NULL 0
```

```
#define TRUE 1
#define PI 3.14
#define TT (M*N)
```

В последнем случае M и N должны быть уже определенными выше константами.

1.2.6 Операторы

В Си различают простые, составные операторы и пустой оператор.

Пример простого оператора – это оператор присваивания. В языке Си он имеет вид:

имя_переменной = выражение

и подробно рассматривается в п. 2.1.1 ниже. Выполнение оператора присваивания: вычисляется значение выражения, указанного справа от знака «=», и это значение присваивается переменной, имя которой указано слева от знака «=». При записи выражения в операторе присваивания используются стандартные арифметические операции "+", "-", "*", "/", которые подробно рассматриваются в п. 2.1.3. ниже.

Составной оператор - это группа операторов, заключенная в фигурные скобки. Простой оператор заканчивается знаком ";", а составной - знаком "}". Для составного оператора после закрывающейся фигурной скобки ставить знак ";" не надо.

Пустой оператор - это просто ";". Используется пустой оператор обычно в случае, если по синтаксису надо выполнить оператор, а по алгоритму ничего не надо делать.

Любой оператор может быть помечен меткой. Метка - это идентификатор, который записывается перед оператором и отделяется от него двоеточием. В пределах одной программы метки должны быть уникальны.

1.2.7 Точка с запятой

В языке Си точка с запятой ";" обозначает конец описания или оператора. Таким образом, каждый отдельный оператор должен заканчиваться точкой с запятой. Исключением является составной оператор: набор операторов, заключенных в фигурные скобки. После составного оператор точка с запятой не ставится.

Итак, в Си знак ";" - это признак конца простого оператора или описания в отличие от Паскаля, где ";" - это разделитель.

1.2.8 Комментарии

Комментарии в Си начинаются парой знаков "/*", оканчиваются парой знаков "*/". Один комментарий может занимать несколько строк. Вложенность комментариев в стандартном Си не допускалась, но в большинстве современных компиляторов возможна.

В языке C++ существует еще один вид комментариев - однострочный комментарий. Все символы, расположенные за парой знаков "//" и до конца строки, рассматриваются как комментарий.

1.2.9 Препроцессор языка Си

Препроцессор используется для обработки текста программы до этапа ее компиляции. Как правило, этап препроцессорной обработки и компиляции объединяют в одну программу и называют ее просто компилятором.

Инструкции препроцессора начинаются со знака "#" в первой позиции строки. Инструкции препроцессора могут появляться в любом месте программы, и их действие

распространяется до конца файла, если оно не прекращено другими инструкциями.

Препроцессор обеспечивает:

1) определение текста с помощью макроподстановок - команда #define;

2) включение в текст файлов - команда #include.

1.2.9.1 Команда #define

Общий вид макроопределения:

#define <идентификатор> <строка замены>

Идентификатор в макроопределении называют "имя макроса".

Каждое появление имени макроса в тексте программы означает макровызов. При этом происходит замена идентификатора строкой замены. Обычно команду #define используют для определения констант (см. п.1.2.5).

1.2.9.2 Команда #include

Общий вид команды: после названия команды "#include" записывается имя файла либо в угловых скобках "<" и ">", либо в двойных кавычках. Например,

#include <stdio.h>

#include "myfiler"

Команда предписывает препроцессору до этапа компиляции программы заменить инструкцию include содержимым указанного файла.

Если имя заключено в двойные кавычки, то считается, что оно относится к текущему каталогу; если в угловые скобки, то считается, что файл находится в отдельном каталоге (в "стандартных местах").

Обычно все программы на Си начинаются с инструкции

#include <stdio.h>

предписывающей препроцессору заменить эту инструкцию содержимым файла stdio.h. Этот файл обеспечивает доступ к стандартным средствам ввода-вывода. (Операции ввода-вывода не принадлежат собственно языку Си, соответствующие действия обеспечиваются стандартными общедоступными средствами). Название файла stdio.h интерпретируется следующим образом:

std - стандартный,

io - от "input/output" - ввод/вывод,

h - от "header files" - заголовочный файл.

1.2.10 Основные математические подпрограммы Си

Указанные ниже подпрограммы (ПП) находятся в библиотечном файле math.h. Переменные x, y описаны с типом double, p - с типом int, результат выполнения ПП - типа double. Подпрограммы в Си называются функциями.

Обращение к функции	Назначение функции
acos(x)	Тригонометрическая функция Дает в радианах значение arccos(x)
asin(x)	Тригонометрическая функция Дает в радианах значение arcsin(x)
atan(x)	Тригонометрическая функция Дает в радианах значение arctg(x) в диапазоне $(-\pi/2, \pi/2)$

cos(x)	Тригонометрическая функция Дает cos(x), где x измеряется в радианах
cosh(x)	Дает косинус гиперболический от x
exp(x)	Дает значение e в степени x: e^x
fabs(x)	Дает абсолютное значение x: $\lvert x \rvert$
log(x)	Дает натуральный логарифм x: ln x
log10(x)	Дает логарифм x по основанию 10: lg x
pow(x, y)	Дает x в степени y: x^y
pow10(p)	Степенная функция. Дает 10 в степени p: 10^p
sin(x)	Тригонометрическая функция Дает sin(x), где x измеряется в радианах
sinh(x)	Дает синус гиперболический от x
sqrt(x)	Дает квадратный корень из x.
tan(x)	Тригонометрическая функция Дает tg(x), где x измеряется в радианах
tanh(x)	Дает тангенс гиперболический от x

Замечание. ПП abs дает абсолютное значение целого аргумента i. Тип результата - int. Вид обращения: "abs(i)". Находится в stdlib.h.

1.3 КОНСТАНТЫ

В данном разделе рассматриваются типы констант в Си и явное представление значений констант в Си-программе.

Типы констант в Си (используемые ниже английские слова не являются ключевыми словами Си, это общепризнанные названия типов констант):

– integer (целые), есть несколько типов целых констант;

– character (символьные, или знаковые);

– floating point (с плавающей точкой), есть несколько типов констант с плавающей точкой;

– enumeration (перечисляемые);

– string (строки).

1.3.1 Целочисленные константы

Целые константы (integer, кратко int) могут быть записаны в тексте программы в десятичной, восьмеричной и шестнадцатеричной системе счисления. Ниже в таблице объясняется представление констант в различных системах счисления.

Представление констант в различных системах счисления

Система счисления	С чего должны начинаться константы	Используемые цифры и знаки
10 с.с	с цифр, отличных от 0	0,1,2,3,4,5,6,7,8,9
8 с.с.	с 0	0,1,2,3,4,5,6,7
16 с.с.	с 0X или 0x	0,...,9,A,B,C,D,E,F

Пример. Ниже дано представление одной и той же по значению константы в разных системах счисления (система счисления указана в скобках).

20 (10 с.с.)=024 (8 с.с.)=0x14 (16 с.с.)

Длинные целые константы (long integer) задаются с помощью записи за целым числом суффикса L или l. Аналогично суффикс U или u задаёт целочисленную константу без

знака (unsigned integer). Можно одновременно использовать оба суффикса L(l) и U(u) для одной и той же константы, это будет длинная беззнаковая константа (unsigned long integer).

Когда тип константы не задан явно с помощью суффикса, система сама определяет тип, исходя из значения константы. Пусть, например, в тексте программы записана константа 13. К какому она относится типу - char (см. п.1.3.2), int, unsigend int, long int ? В принципе, для Си это не играет большой роли (но имеет смысл для C++). Правила определения типа целой константы:

1) целая константа относится к типу integer, если она входит в интервал значений integer; например, константа -340 считается типа integer;

2) целая положительная константа относится к типу unsigned integer, если она выходит за интервал значений integer;

3) если же константа не входит в интервал значений integer и unsigned integer, то она считается типа long integer.

1.3.2 Символьные константы

Символьные, или знаковые, константы (char) формируются заключением одного знака между одиночными кавычками. Символьную константу можно рассматривать и как константу типа integer со значением, которое является кодом символа в кодировке ASCII.

Все символы можно условно разделить на две группы:

1) "видимые" символы, которые можно вывести на печать; например символы 'a' , '1';

2) "невидимые" символы, которые не имеют графического представления; например, символ "перевод строки" или символ "звуковой сигнал".

Так, в коде ASCII символы с номерами от 0 до 31 являются управляющими символами, они не соответствуют никаким печатным символам и их нельзя ввести с клавиатуры. Такие "невидимые" символы обозначаются в Си-программе с помощью управляющей escape-последовательности ("escape" - убегать, спасаться, становиться невидимым). Escape-последовательность (иногда называют "управляющий код" или "управляющий символ") состоит из символа обратной дробной черты "\" (обратный слэш), за которым следует ещё один символ. Ниже в таблице представлены escape-последовательности.

Escape-последовательности

Escape–последова-тельность	Значение кода в 16 с.с.	Обозначе-ние	Что делает
\a	0x07	BEL	Звуковой сигнал
\b	0x08	BS	Шаг назад
\f	0x0C	FF	Переход на новую страницу
\n	0x0A	LF	Переход на следующую строку
\r	0x0D	CR	Возврат каретки
\t	0x09	HT	Горизонтальная табуляция
\v	0x0B	VT	Вертикальная табуляция
\\	0x5C	\	Обратная дробная черта
\'	0x2C	'	Одиночная кавычка (апостроф)
\"	0x22	"	Двойная кавычка
\?	0x3F	?	Вопросительный знак

Все вышеприведённые знаки, а также вообще любые символы, могут быть заданы в программе в следующем виде:

1) '\ddd', где ddd - код символа в ASCII: от 1 до 3 восьмеричных цифр;

2) '\xhhh', где hhh - код символа в ASCII: от 1 до 3 шестнадцатеричных цифр.

Пример. Ниже приведены примеры разных способов задания символьных констант в программе.

1) ‘\n’ - escape-последовательность
   ‘\012’ - код в 8 с.с
   ‘\12’ - код в 8 с.с
   ‘\x0A’ - код в 16 с.с

   } Эти четыре варианта задают один и тот же знак: "знак перевода строки" ("знак перехода на следующую строку").

2) 'G'
   '\107'
   '\x47'

   } Эти три варианта задают один и тот же знак: букву "G"

3) '\a'
   '\7'
   '\07'
   '\0x7'

   } Эти четыре варианта задают один и тот же знак: звуковой сигнал

4) 'a' - знак-буква "a"

5) '\0' - Нулевой символ NULL имеет код, равный нулю; не путать со знаком ‘0’, код которого равен 48 в 10 с.с.

1.3.3 Константы с плавающей точкой

Константы с плавающей точкой (floating pointer) задаются с помощью обычных обозначений, например:

24.0, 2.4E1 (или 2.4e1), 240.0E-1.

Все константы с плавающей точкой задают значения с двойной точностью (с двойной длиной) double. Можно сделать константу с плавающей точкой и просто типа float, прибавив к ней суффикс F(f). Есть ещё суффикс L(l) - long double, но он иногда трактуется как просто double.

Диапазон и размер констант с плавающей точкой

Диапазон	Размер	Тип
3.4E-38 – 3.4E+38	32 бита (4 байта)	Float
1.7E-308 – 1.7E+308	64 бита (8 байт)	Double
1.7E-308 – 1.7E+308	64 бита (8 байт)	long double
3.4E-4932 – 3.4E+4932	80 бит (10 байт)	Long double в некоторых версиях Си

1.3.4 Константы перечисляемого типа

Перечисляемые константы (enumeration) являются идентификаторами, обозначающими значение типа, определённого пользователем (см. п.1.4.4).

1.3.5 Строковые константы

Строковые константы (string) формируются заключением в двойные кавычки последовательности знаков (эта последовательность может не содержать ни одного знака). Примеры строковых констант:

" " (строка содержит один знак "пробел"),

"" (пустая строка),

"A", "Ошибка".

Знак двойной кавычки может быть включён в константу типа string с помощью escape-последовательности; то есть в любом месте, где знак "двойная кавычка" должен быть элементом строковой константы, а не ее ограничителем, нужно записать: \". Например, чтобы получить строку:

...язык "Visual Basic"...

надо написать:

"...язык \"Visual Basic\"..."

В стандартном Си (в частности, в UNIX) с помощью обратного слэша "\" строковые константы могут быть продолжены на следующей строке текста программы. Например:

```
"очень, очень, очень, очень ... очень\
длинная строка"
```

Во многих версиях Си это делается проще: длинная строка разбивается на несколько отдельных строк. Например, рассмотрим следующий фрагмент программы:

```
char *p;              /* Описание переменной p как указателя на char */
p="очень, очень, очень,"    /* p указывает на длинную строку      */
" очень,\n очень "
"длинная строка.";
printf("%s",p);               /* Печать p */
```

В результате печати получим на экране:

```
очень, очень, очень, очень,
очень длинная строка.
```

В строковых константах могут содержаться знаки, не имеющие графического представления; так, в примере выше это знак '\n' - перевод строки.

Строковые константы являются фактически массивами знаков. Например, константа "Good day" эквивалентна массиву из 9 знаков:

'G', 'o', 'o', 'd', ' ', 'd', 'a', 'y', '\0'.

Следует обратить внимание на последний знак '\0'. Особенностью представления строковых констант в Си является то, что в памяти компьютера под строку отводится на 1 байт больше, чем требуется для размещения реальных символов строки. Этот последний байт заполняется нулевым значением, то есть это байт, в двоичной записи которого одни нули. Этот байт так и называется - нулевой байт, или нулевой знак, или нулевой символ; обозначается как '\0'. Нулевой знак - это первый знак по таблице ASCII, код его равен 0.

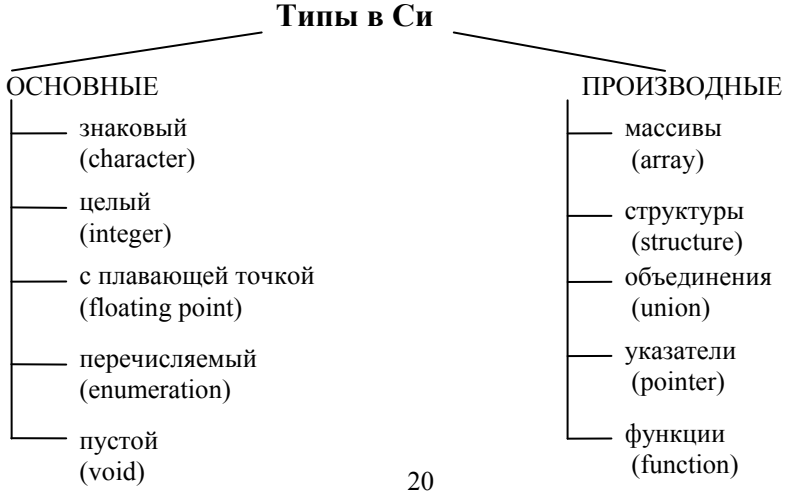
Итак, по соглашению строки в Си заканчиваются нулевым знаком '\0', и обработка строк основана на этом соглашении. Это позволяет снять ограничения с длины строк: строка может быть такой длины, какой позволяет память для её хранения.

Для строковых констант нулевой знак добавляется компилятором автоматически. Однако для строк, сформированных программистом явно, то есть с использованием массива знаков, программист должен сам добавить в конец строки нулевой знак.

1.4 ТИПЫ И ПЕРЕМЕННЫЕ

Тип - это, по сути, множество значений плюс набор операций, которые могут быть выполнены с этими значениями.

Переменная - это компонента программы, используемая для хранения значения сопоставленного с ней типа.



В схематичном виде представлены основные типы в Си. Используемые при этом английские слова - это общепризнанные названия типов, а не ключевые слова языка Си.

Типы character, integer, floating point, enumeration - арифметические типы, так как они могут интерпретироваться как число.

Типы character, integer, numeration - целые типы (integral).

Тип floating point относится как к числам обычной точности (float), так и двойной точности (double).

В Си определены пять базовых типов данных:

1) char –символьные,
2) int – целые,
3) float – с плавающей точкой,
4) double – двойной точности,
5) void – без значения.

На основе этих типов формируются другие типы данных. Размер, то есть объем занимаемой памяти, и диапазон значений этих типов данных для разных процессоров и компиляторов могут быть разными, но есть общие правила. Объект типа char всегда занимает 1 байт. Размер объекта типа int обычно совпадает с размером слова в конкретной среде программирования. В большинстве случаев в 16-разрядной среде (DOS или Windows 4.1) int занимает 16 битов, а в 32-разрядной (Windows 95/98/NT/2000 и т.д.) – 32 бита. Однако полностью полагаться на это нельзя, особенно при переносе программы в другую среду. Стандарт Си обуславливает только минимальный диапазон значений каждого типа данных, но не размер в байтах. /Шилдт Г. Полный справочник по Си/.

Базовые типы данных (кроме void) могут иметь различые спецификаторы (описатели, модификаторы), предшествующие им в тексте программы. Спецификатор типа так изменяет значение базового типа, чтобы он как можно точнее соответствовал цели своего использования в программе. Существую следующие спецификаторы: signed, unsigned, long, short. Обычно спецификаторы позволяют изменить диапазон используемых для типа значений без или с изменением объема занимаемой памяти.

Ниже рассмотрены все основные типы.

1.4.1 Знаки

Переменные типа char могут принимать значения символьных констант (см. п.1.3.2). Пример описания переменных c, ch как знаковых:

```
char c,ch;
```

Знак хранится как целое число, соответствующее его ASCII-коду; следовательно, знаки можно рассматривать как целые числа и наоборот. Поэтому компилятор не в состоянии обнаружить такие потенциальные ошибки, как неумышленное сложение знаковой и целой переменной!

Тип char может быть модифицирован с помощью спецификаторов signed и unsigned. Тип unsigend char подробнее рассматривается в п.7.1.1. Краткая характеристика знаковых типов приведена в таблице ниже.

Тип	Размер в байтах (битах)	Интервал изменения
Char	1 (8)	От -127 до 127
Unsigned char	1 (8)	от 0 до 255
Signed char	1 (8)	От -127 до 127

1.4.2 Целые переменные

Основное название целого типа – int. К типу int можно применять любой из четырех спецификаторов: signed, unsigned, long, short. При указании этих спецификаторов к типу int само слово int можно не писать.

Итак, в зависимости от числа двоичных разрядов, отводимых для представления значения, обычно рассматриваются следующие типы целых переменных: int, short int (или просто short), long int (или просто long), long long int (или просто long long), unsigned int (или просто unsigned). Тип long long int можно еще обозначать как __int64. Следует иметь в виду, что данный тип существует не для всех версий Си.

Пример описания целых переменных (слева - с использованием полных названий типов, справа - с использованием кратких названий):

```
int  i,n;                          int  i,n;
short int  low,high;               short  low,high;
long int  max;                      long  max;
```

Всегда

S(short int) <= S(int) <= S(long int),

где S(x) - размер области памяти, отводимой переменной типа x.

Для большинства 16-разрядных сред:

S(short int)=16 бит, S(int)=16 бит, S(long int)=32 бита.

Для большинства 32-разрядных сред:

S(short int)=16 бит, S(int)=32 бита, S(long int)=32 бита.

Размер типа long long int (__int64), как видно из названия, равен 64 битам.

Переменные типа short int употребляются только при необходимости экономии памяти. Следует иметь в виду, что использование short int иногда увеличивает время выполнения программы, так как в арифметических операциях значения типа short преобразуются в значения типа int до их использования.

Если знаковый бит в представлении числа не нужен, то переменной можно присвоить тип unsigned int (целое без знака) или просто unsigned (беззнаковое).

1.4.3 Плавающая точка

Существует два основных типа с плавающей точкой (вещественных типа): float (обычная точность, занимает 32 бита) и double (двойная точность, занимает 64 бита). Тип double можно модифицировать с помощью спецификатора long (занимает 80 бит).

Пример описания x, y как переменных типа float, a eps - типа double:

```
float x,y;
double eps;
```

В языке Си все арифметические операции с плавающей точкой выполняются с двойной точностью. За повышенную точность приходится платить увеличением времени выполнения программы.

1.4.4 Перечисляемые типы

Перечисляемые типы позволяют в качестве значений использовать идентификаторы. Существует несколько способов описания перечисляемого типа. Основной 1-ый способ описания перечисляемого типа имеет следующий вид:

```
typedef  enum  {a0,a1,...,an} E;
```

где E - имя описываемого перечисляемого типа;

a1, a2,...,an - перечисляемые константы (идентификаторы);

typedef - ключевое слово для описания типа в Си.

Одна и та же перечисляемая константа не может присутствовать в описании двух различных типов.

Пример 1. Ниже дано описание двух перечисляемых типов day и light с помощью ключевого слова typedef.

```
typedef enum{mon,tue,wed,thue,fri,sat,sun} day;
typedef enum{red,yellow,green} light;
```

После этого переменные d и signal перечисляемых типов day и light, соответственно, определяются следующим образом:

```
day d;          /* d - переменная перечисляемого типа day        */
light signal; /* signal - переменная перечисляемого типа light */
```

Для описания переменных перечисляемого типа можно использовать метки перечисляемого типа. Это 2-ой способ задания перечисляемого типа. Метки перечисляемого типа аналогичны имени перечисляемого типа.

Пример 2. Ниже дано описание меток day и light перечисляемого типа.

```
enum  day  {mon,tue,wed,thue,fri,sat,sun};
enum light  {red,yellow,green};
```

После этого определение перечисляемых переменных d и signal с помощью меток day и light, соответственно, будет выглядеть следующим образом:

```
enum  day  d;
enum  light  signal;
```

Можно упростить 2-ой способ определения перечисляемых переменных, объединив в одном описании и определение метки, и определение соответствующих переменных. Назовем это 3-им способом определения перечисляемых переменных.

Пример 3. Ниже дано определение перечисляемой переменной d, тип которой определяется меткой day.

```
enum  day {mon,tue,wed,thue,fri,sat,sun}  d;
```

Перечисляемые константы являются по сути целыми константами. Например, для типа day первая константа mon автоматически устанавливается в 0, а каждая следующая константа имеет значение на 1 большее. Перечисляемым константам можно явно присвоить определённые значения.

Пример 4. Рассмотрим определение перечисляемой переменной d, в котором некоторым перечисляемым константам явно присвоены значения.

```
enum  day {mon=1,tue,wed=5,thue,fri,sat,sun};
enum  day  d;
```

В этом случае: mon=1, tue=2, wed=5, thue=6, fri=7, sat=8, sun=9. Если в программе написать, например, оператор присваивания "d=fri;", а затем распечатать значение d, то получим на выводе целое число 7.

Перечисляемый тип обычно используется для повышения читабильности текста программы. Ошибочное использование перечисляемых значений в качестве целых чисел не может быть обнаружено компилятором!

1.4.5 Тип void (пустой)

Тип void - это пустой тип, не имеющий значения. Переменную типа void создать нельзя. С помощью типа void представляется пустое множество значений. Этот тип обычно используется в следующих случаях:

1) для указания типа функций, не возвращающих значений;

2) для указания того, что значение выражения не будет использовано, оно вычисляется только для получения побочных эффектов.

Ключевое слово void отсутствует в стандартном Си, оно было привнесено в стандарт ANSI C из языка C++.

1.4.6 Логические значения

В языке Си нет логических (булевых) значений, подобных true и false в Паскале, вместо них используются целые значения. Ненулевые значения соответствуют логическому значению true, а нуль - false. По соглашению, заранее определённые операторы и функции возвращают в вызывающую программу значение "1" в качестве true, и "0" - в качестве false.

Для ясности при обозначении булевых значений можно использовать константы TRUE и FALSE и определить их следующим образом:

```
#define TRUE 1
#define FALSE 0
```

Но для Си-программ такое определение, в принципе, совершенно излишне.

1.5 ПРЕОБРАЗОВАНИЕ ТИПОВ

1.5.1 Неявные преобразования типов

Неявные преобразования типов обычно выполняются при преобразовании фактического параметра к типу соответствующего формального параметра при работе с подпрограммами. Ниже в таблице представлены наиболее часто встречающиеся возможные преобразования основных типов.

Возможные преобразования типов

Преобразуемый тип	Список типов, в которые он может быть преобразован непосредственно
char	int, short int, long int
int	short int, long int, unsigned int, char, float, double
short int	аналогично int
long int	аналогично int
float	double, int, short int, long int
double	float, int, short int, long int

Замечание. Преобразование float или double в int, short int, long int может давать неожиданный результат, если преобразуемое значение слишком велико.

1.5.2 Арифметические преобразования

Большинство операций Си выполняют преобразование типов, чтобы привести операнды выражений к общему типу, или чтобы расширить короткие величины до размера целых величин, используемых в машинных операциях. Преобразования, выполняемые операциями Си, зависят от специфики операций и от типа операнда или операндов. Тем не менее, многие операции выполняют похожие преобразования целых и плавающих типов. Эти преобразования известны как арифметические преобразования, поскольку они применяются к типам величин, обычно используемых в арифметике. Арифметические преобразования, приведенные ниже, называются "обычные арифметические преобразования". Обычные арифметические преобразования осуществляются автоматически по следующей схеме.

1. Типы char, enum, short int преобразуются в int; тип float - в double.
2. Для любой пары операндов: если хотя бы один из операндов типа long double, то и

другой операнд преобразуется к типу long double; если хотя бы один из операндов типа double, то и другой операнд преобразуется к типу double; если хотя бы один из операндов типа long , то и другой операнд преобразуется к типу long; если хотя бы один из операндов типа unsigned, то и другой операнд преобразуется к типу unsigned.

3. В операторе присваивания конечный результат приводится к типу переменной в левой части оператора присваивания, при этом тип может как повышаться, так и понижаться.

1.5.3 Явные преобразования типов

Для явного преобразования типа используется операция "приведение". Если перед любым выражением поставить в круглых скобках имя типа, то выражение даст результат этого типа.

Пример 1. Рассмотрим следующее выражение: (long)2+3
Результат этого выражения - значение 5 типа long.

Пример 2. Стандартные математические функции exp, log и другие рассчитаны на параметр типа double. Если надо получить ln(x), и x - типа float, то можно записать

```
log((double)x)
```

1.6 ВВОД И ВЫВОД В СИ

1.6.1 Стандартные потоки ввода/вывода

Большинство программ читают вводимые данные из специального файла - потока (stream). В поток записываются и выводимые данные. Поток - это файл, ассоциированный с буфером. Буфер - это выделенный участок памяти. При буферном вводе информация с устройства ввода поступает в буфер, а оттуда передается прикладной программе. Аналогично и при буферном выводе - информация из программы вначале поступает в буфер, а затем - на устройство вывода.

Для любой программы в системах с Си всегда доступны следующие стандартные потоки ввода - вывода:

stdin - файл стандартного ввода или просто стандартное входное устройство;

stdout - файл стандартного вывода или просто стандартное выходное устройство.

Эти файлы всегда связываются (отождествляются) с пользовательским терминалом, то есть с клавиатурой и дисплеем.

Для чтения (записи) данных из (в) стандартного файла используются следующие стандартные функции (их описание находится в stdio.h):

scanf (printf) - форматированный ввод (вывод) из stdin (в stdout);

gets (puts) - чтение (запись) строки из stdin (в stdout);

getchar (putchar)- чтение (запись) знака из stdin (в stdout)

1.6.2 Вывод

1.6.2.1 Стандартная функция форматированного вывода printf

Общий вид обращения к printf:

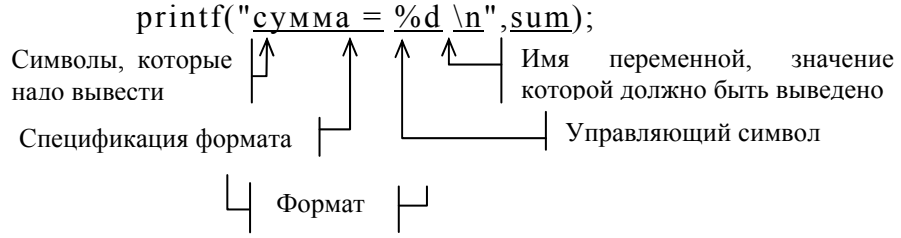
```
printf("<формат>"[,<выр.1>,<выр.2>,...]);
```

где выр.1, выр.2, ... - это произвольные выражения, значения которых надо вывести.

Простейшие выражения- это константа, переменная, вызов функции.

Формат или строка формата содержит символы, которые следует напечатать, управляющие символы (escape-последовательности) и спецификации формата.

Пример 1. Ниже рассмотрен пример обращения к функции printf.



Выполнение функции printf: строка формата выводится на экран, причем вместо спецификаций печатаются значения соответствующих выражений. Спецификации определяют, как выводить значения выражений. Для каждого выражения должна быть только одна, соответствующая ему, спецификация формата.

Любая спецификация начинается с символа % и обычно содержит одну или две буквы, обозначающие тип данных и способ их преобразования. Основные спецификации формата:

%d - целое десятичное число;

%i - целое десятичное число;

%u - целое число без знака;

%p - значение указателя;

%n - значение указателя;

%f или %F - десятичное число с плавающей точкой (обычно с 6 знаками после десятичной точки);

%e или %E- десятичное число с плавающей точкой в экспоненциальном виде;

%g или %G - будет %f или %e, что короче; незначащие нули после десятичной точки не указываются;

%c - символ;

%s - строка символов;

%x или %X - целое число в 16-ричном формате;

%o - восьмеричное число;

%% - символ %.

Перед буквами d, i, o, u, x, X может быть добавлена буква l, указывающая на long integer, то есть спецификация формата будет выглядеть, например, так : “%ld”.

Для типа long long int (__int64) указывается спецификация формата lld. В некоторых системах, например MinGW, для данного типа спецификация имеет вид: ”%I64d”

Для типа double перед буквами f, g, e необходимо добавлять букву l, то есть спецификация формата будет выглядеть, например, так : “%lf”.

Если выражение имеет тип, не соответствующий спецификации формата, то будет сделана попытка выполнить нужное преобразование.

Общий вид спецификации формата для функции printf:

%[<флаг>][<ширина>][.<точность>] <тип>

где <тип> - это буква d, или u, или p и т.д. (см. выше);

<флаг> - это обычно знак "-" который означает, что выдаваемое значение должно быть в поле вывода выровнено влево; по умолчанию - вправо;

<ширина> - число, задающие минимальный размер поля вывода, то есть общее число позиций, занимаемых печатным материалом. В избыточных позициях поля по умолчанию печатается пробел, но если первая цифра ширины поля есть 0, то поле дополняется нулями. Если размер поля недостаточен, то истинный размер диктуется самим значением.

<.точность> - число, которое:

а) для величин типа float и double указывает, сколько будет напечатано цифр после десятичной точки;

б) для строк задает число печатаемых от начала строки символов.

Пример 2. Ниже показано, что будет выведено на печать функцией printf для определенных значений, выводимых по определенным спецификациям формата.

Значение	Спецификация формата	Выдача	Примечания
360	%10d	360	7 пробелов перед числом
360	%-10d	360	7 пробелов после числа
3.14159265	%10f	3.141593	(*)
3.14159265	%10.0f	3	(**)
3.14159265	%10.3f	3.142	5 пробелов перед числом
Programmer	%10s	Programmer	
Programmers	%10s	Programmers	
Programmer	%10.7s	Program	3 пробела перед словом
Programmer	%10.0s	Programmer	
Programmer	%.3s	Pro	

Примечания.

(*)-если для спецификации f точность не указана, то предполагается 6 цифр после десятичной точки; 2 пробела перед числом.

(**) - если для спецификации f указана нулевая точность, то в выводимом результате отсутствует десятичная точка; 9 пробелов перед числом.

Из управляющих escape-последовательностей (см. п. 1.2.2.) наиболее часто используется '\n' - переход на новую строку.

Пример 3. Пусть имеем: n=3, x=5 - integer, c='?' - char. Тогда в результате работы следующего оператора:

```
printf("n=%d, x=%d\n c=%c",n,x,c);
```

получим следующий вывод:

```
n=3, x=5
c=?
```

Пример 4. Пропуск 35 пробелов даст на выводе следующий оператор:

```
printf("%35c",' ');
```

Пример 5. Выполнение функции printf фактически заключается в выводе строки формата, указанной в качестве первого параметра функции printf. Примеры ниже иллюстрируют этот факт.

```
Printf("Lena");            /*  На печати будет одно имя  Lena */
Printf("Lena","Olga");  /*  На печати будет одно имя  Lena. Вторая строка
                                с именем  Olga игнорируется */
Printf("Lena" "Olga");    /*  На печати будет два имени  LenaOlga */
```

Пример 6. Пусть имеем следующие описания переменных (переменные описаны и инициализированы, то есть им присвоены начальные значения):

```
int  n=1, m=2;     float  x=3, y=4;
```

Ниже приведены различные варианты (допустимые и недопустимые) вывода значений этих переменных.

```
printf("\n  n=%d  m=%d",n,m);   /* Нормальный вывод:  n=1  m=2  */
printf("\n  n=%f  m=%f",n,m);    /* Компиляция пройдет успешно, но на этапе
                                    выполнения будет ошибка         */
printf("\n x=%f y=%f",x,y);     /* Нормальный вывод: x=3.000000 y=4.000000*/
printf("\n  x=%d  y=%d",x,y);  /* Неудовлетворительный вывод: x=0  y=0 */
printf("\n  n=%d  ",n,m);        /* Вывод:  n=1.  Значение  переменной  m  не
        распечатается, так как в строке формата нет для нее спецификации*/
printf("\n  n=%d, m=%d, z=%d",n,m);   /*  Не  хватает  переменных,  лишняя
                спецификация %d.  Будет следующий вывод:  n=1, m=2, z=0 */
```

```
printf("\n  n=%d, m=%d, z=%f",n,m);     /*  Не хватает переменных,   лишняя
                                спецификация %f.  Будет ошибка выполнения  */
printf("\n  n=%d","m=%d",n,m);   /* Неудовлетворительный вывод:  n=176.
          Строка "m=%d" воспринимается как выражение, значение которого
        должно быть распечатано, что приводит к бессмысленному выводу */
printf("\n  n=%d" " m=%d",n,m);   /* Нормальный вывод:  n=1 m=2  */
printf("\n  n=%d",n," m=%d",m);  /* Вывод:  n=1.  Вторая  строка  "m=%d"  и
                переменная m игнорируются как лишние в списке вывода */
```

1.6.2.2 Функция puts

Функция puts (описание - в stdio.h) выводит строку на экран и завершает вывод символом новой строки '\n'.

Пример 1. Пусть имеем следующее описание строки:

```
char s[30];  /* s - строка       */
```

После того, как строка будет сформирована, например, введена с клавиатуры, ее можно распечатать с помощью следующего оператора

```
puts(s); /*вывод строки и перевод курсора на начало следующей строки */
```

Функция puts в отличие от printf может выводить только строку, зато работает быстрее и запись ее короче. В результате действия puts всегда происходит переход на новую строку.

1.6.2.3 Функции putchar и putch

Функция putchar (описание - в stdio.h) записывает единственный символ, который является ее аргументом, на экран в текущую позицию курсора и не добавляет '\n'. Оператор "putchar(ch);" эквивалентен оператору "printf("%c",ch);".

Функция putch (описание в conio.h) выводит символ, который является ее аргументом, непосредственно на консоль (на экран) в текущую позицию курсора, то есть осуществляет небуферизированный вывод.

Пример 2. Для вывода знака ch на экран можно использовать один из следующих операторов:

```
putchar(ch);
putch(ch);
```

1.6.3 Ввод

1.6.3.1 Стандартная функция форматированного ввода scanf

Функция scanf читает данные из стандартного входного потока stdin и сохраняет их по адресам, которые задаются адресными аргументами.

Общий вид обращения к scanf:

```
scanf("<формат>"[,&<имя1>,&<имя2>,...]);
```

где <имя1>, <имя2>,... - адресные аргументы; обычно это имена переменных, значения которых надо ввести, знак & перед каждым именем обычно обязателен;

<формат> - строка формата, в простейшем случае состоит из одних спецификаций формата вида "%<буква>"аналогично формату функции printf (см. п. 5.2.1.).

Все переменные, которые мы вводим, должны указываться для функции scanf с помощью адресов, поэтому перед именем переменной указывается операция получения

адреса - знак &. Знак & не нужно указывать перед именем массива знаков при вводе строки, так как имя массива без индексов является в Си адресом этого массива.

Пример 1.

```
scanf("%d",&n);   /* ввод целого числа - значения переменной n */
scanf("%d%f%d%c",&m,&x,&k,&ch); /*ввод целого m, вещественного x,
                                    целого k, символа ch       */
```

Для заданных спецификаций формата должно быть достаточное число адресных аргументов. В противном случае результат работы функции scanf непредсказуем и может привести к катастрофическим последствиям. Лишние адресные аргументы, указанные сверх требуемых строкой формата, игнорируются.

В общем случае строка формата содержит символы трех типов:

1) неотображаемые символы (знаки);
2) отображаемые символы (знаки);
3) спецификации формата.

Неотображаемые символы строки формата - это пробелы, символы табуляции и перехода на новую строку. Они обеспечивают чтение информации из входного потока до ближайшего знака, отличного от них самих.

Отображаемый знак должен совпадать с соответствующим знаком во входном потоке данных (см. пример 2 ниже).

Пример 2. Рассмотрим обращение к функции scanf, в котором строка формата содержит отображаемый символ ','.

```
scanf("%d,%d",&n,&m);
```

В этом случае при вводе соответствующие значения переменных n и m нужно обязательно разделить знаком ','. Например:

1,2<Enter>

Спецификация формата управляет преобразованием следующего поля ввода из входного потока; результат присваивается переменной, указываемой соответствующим аргументом.

Полями ввода могут быть следующие данные:

–все символы до следующего неотображаемого символа (не включая его);

–все символы до первого не преобразуемого по указанной спецификации формата символа (например, до цифры "8" или "9" при вводе по восьмеричному формату);

–n символов, где n - ширина поля ввода.

Общий вид спецификации формата для функции scanf:

```
%[*][N]c
```

где c - это символ формата (d, f, c и т.д. - см. ниже);

* - это допустимый признак гашения присваивания (отменяет присваивание следующего поля ввода, то есть входное поле будет считано, но не присвоено следующему аргументу);

N - ширина поля, то есть число, задающие максимальный размер поля ввода (меньшее число символов может быть считано, если scanf встретит неотображаемый или не преобразуемый символ).

Символы формата в большинстве своем аналогичны соответствующим символам (d, f и др.) оператора printf (см. п. 5.2.1). Основные спецификации формата scanf:

%d, %i (%o, %x) - чтение (ввод) десятичного (восьмеричного, шестнадцатиричного) целого числа;

%e, %f - чтение (ввод) чисел f с плавающей точкой;

%s - чтение (ввод) строки знаков; поле входных данных завершается знаком пробела, табуляции, перехода на новую строку;

%c - чтение (ввод) некоторого знака, включая пробел и другие неотображаемые знаки; для пропуска неотображаемого знака и чтения следующего литера используется спецификация 1s;

%p - чтение указателя;

%n - чтение указателя в увеличенном формате;

%[образец] - по этому формату вводятся строки.

Рассмотрим последнюю спецификацию подробнее. Квадратные скобки заключают в себя набор допустимых символов, из которых должно состоять поле ввода (вводимая строка). Если первым символом в скобках является "^",то вводится будут все символы из входного потока, кроме заключенных в скобки.

Интервал знаков можно представить конструкцией "первый -последний". Например, [0,1,2,3,4,5] равносильно [0 - 5]. При этом первый знак должен быть " меньше " второго. Если знак " - " надо интерпретировать как знак, то надо поставить его в скобках первым или последним.

Функция scanf может прекратить чтение текущего входного поля до того, как встретит обычный символ конца поля (неотображаемый символ: например, пробел). Причины:

1) встретился символ подавления присваивания; текущее входное поле будет считано, но не сохранено;

2) было считано N символов, где N - ширина входного поля;

3) встретился не преобразуемый символ, то есть такой, который не может быть преобразован в соответствии с указанной спецификацией (например, символы "8" и "9" для восьмеричного формата);

4) следующий символ во входном потоке не соответствует заданному множеству допустимых символов, указанному в квадратных скобках.

Если функция scanf прекратила чтение текущего входного поля по одной из этих причин, то следующий символ окажется не считанным и будет первым символом следующего входного поля или первым символом ввода для последующих операторов ввода из входного потока stdin.

Функция scanf может завершить свою работу по следующим причинам:

1) следующий символ во входном потоке противоречит соответствующему отображаемому символу в строке форматов;

2) строка форматов исчерпана;

3) следующий символ во входном потоке есть символ EOF.

После завершения функция scanf возвращает число элементов входных данных, значения которых присвоены переменным в соответствии со спецификациями формата. Это число может быть равно 0, если несоответствие знака входных данных и строки формата обнаружилось в самом начале ввода. По достижению конца файла выдается значение EOF (определяемое в stdio.h).

Ниже рассмотрено несколько примеров, иллюстрирующих работу функции scanf.

Пример 3.

```
int i,item;
float x;
item = scanf("%d%f",&i,&x);
       /* При входной строке  "1 5.8"  получим:  i=1,  x=5.8,  item=2 */
```

Пример 4.

```
scanf("%d%*c%d",&i,&j);  /* При вводе "50+20" получим:  i=50,  j=20,
                            символ '+' будет прочитан и проигнорирован */
```

Пример 5.

```
char str[20],     /* str - знаковый массив (строка) */
     c1,c2;
int i1,i2,i3,i4;
float x;
scanf("%s  %c  %c  %*s  %d  %3d  %d",str,&c1,&c2,&i1,&i2,&i3);
       /* При входной строке "abc defg -12 345678 9" получим:
            str = "abc\0", c1='d', c2='e', i1=-12, i2=345, i3=678.
          следующий ввод из stdin начинается с пробела, стоящего после цифр 345678. */
     /* str - массив символов, и имя массива str без индексов - это адрес
          самого массива, поэтому в операторе scanf перед именем str не нужна операция
```

```
        получения адреса & */
scanf("%2d%f%*d %[0-9]",&i1,&x,str);
      /* При входной строке "56789 0123 56A72" получим: i1=56, x=789.0,
        str="56\0".   Следующий ввод из stdin начнется с символа 'A' */
scanf("telephone %d-%d-%d",&i1,&i2,&i3);
  /*При входной строке "telephone 25-43-21" получим: i1=25, i2=43, i3=21*/
```

Пример 6. В программе ниже для каждого оператора scanf в виде комментария указано, как нужно вводить соответствующие данные с клавиатуры.

```
#include <stdio.h>
main()
{int a,b;
 char ch;
 printf("\nВведите два целых числа:");   /*  Ввод:  два числа,  разделенные  */
 scanf("%d%d",&a,&b);                    /*  любым количеством пробелов,  */
 printf("\nВы ввели a=%d, b=%d",a,b);    /*  знаков  табуляции  или  <ВК>  */

 printf("\nВведите два целых числа:");   /*  Ввод:  два числа,  разделенные  */
 scanf("%d   %d",&a,&b);                 /*  любым количеством пробелов,  */
 printf("\nВы ввели a=%d, b=%d",a,b);    /*  знаков  табуляции  или  <ВК>  */

printf("\nВведите два целых числа через запятую:");    /* Ввод: два числа, */
 scanf("%d , %d",&a,&b);          /*  разделенные запятой. Между числом и */
 printf("\nВы ввели a=%d, b=%d",a,b);    /* запятой могут быть или не быть*/
                                 /*  пробелы. Например: "3,5", "3  ,5", "3  ,  5" */

 printf("\nВведите два целых числа через запятую:");    /* Ввод:  два числа, */
 scanf("%d,%d",&a,&b);            /*  разделенные запятой. Между числом и */
 printf("\nВы ввели a=%d, b=%d",a,b); /*запятой не должно быть пробелов.*/
                                      /*  Например: "3,5"                  */

 printf("\nВведите два целых числа через запятую:"); /*  Ввод:  два  числа,  */
 scanf("%d,  %d",&a,&b);          /*  разделенные  запятой. Между первым  */
 printf("\nВы ввели a=%d, b=%d",a,b);          /* числом и запятой пробела  */
                /*  не должно быть. Между вторым числом и запятой могут  */
                                 /*  быть пробелы. Например: "3,5", "3, 5".  */

 printf("\nВведите целое число, сразу за ним символ:");     /*  Ввод:  число, */
 scanf("%d%c",&a,&ch);                 /*  сразу за ним (без пробела) символ.  */
 printf("\nВы ввели a=%d, ch=%c",a,ch);  /* Символ может быть пробелом. */
                                       /*  Например: "123b", "123 ".              */

 printf("\nВведите целое число, за ним через пробел символ:");    /*  Ввод:  */
 scanf("%d  %c",&a,&ch);                          /*  число, пробелы, символ.  */
 printf("\nВы ввели a=%d, ch=%c",a,ch);       /*  Пробел в качестве символа */
                                     /*  ввести нельзя. Например, "123 b".  */
}
```

Пример 7. Ниже приведены примеры нестандартных обращений к функции scanf, которые обычно приводят к ошибкам.

```
   int  n,m;
 scanf("%f %f",&n,&m);  /*  Будет ошибка на этапе выполнения  */
 scanf("%d %d",&n);       /*  Пользователю надо ввести два числа. Первое воспримется как
                  значение переменной n, второе игнорируется и никуда не записывается  */
 scanf("%d",&n,&m);       /*  Введется одно число и запишется как  n. Переменная  m  своего
        значения не изменит, потому что запись &m в списке параметров игнорируется
```

(нет соответствующей спецификации формата в строке формата) */
scanf("%d",&n,"%d",&m); /* Введется одно число и запишется как n. Переменная m своего значения не изменит, потому что запись "%d",&m в списке параметров игнорируется */

1.6.3.2 Функция gets

Функция gets используется для ввода строк. Общий вид обращения:

gets(s);

где s - строка (массив знаков), описанная, например, следующим образом:

char s[20]

Функция gets считывает строку из стандартного входного потока stdin: читает все знаки, включая пробелы, и помещает их в массив знаков s. Считывание идет до тех пор, пока не будет прочитан знак перехода на новую строку (соответствует нажатию на клавишу <Enter> - <ВК >, "Ввод"). Этот знак не помещается в строку s; вместо него в строку s добавляется нулевой символ '\0'.

Отличие scanf и gets: scanf читает все символы до тех пор, пока не встретится пробел, или табуляция, или конец строки ; gets считывает любые символы, пока не встретит конец строки, то есть пока пользователь не нажмет клавишу <Enter> (конец строки). Это означает, например, что с помощью scanf нельзя ввести строку, содержащую пробелы, а с помощью gets - можно.

1.6.3.3 Функции getchar, getch и getche

Описание функции getchar находится в stdio.h, функций getch и getche - в conio.h.

Все три функции читают единственный символ. Функция getchar производит буферизованный ввод, то есть читает символ из стандартного входного потока stdin, требует нажатия клавиши <Enter> и отображает введенный символ на экран. Функции getch и getche получают символ непосредственно с консоли (с клавиатуры) без последующего нажатия клавиши <Enter>.Отличие этих функций в том, что getch не выводит символ на экран (ввод символа без отображения, то есть без эха, на экране), а getche отображает введенный символ на экране (ввод символа с отображением, то есть с эхом, на экране). Функцию getch часто используют для остановки работы программы до нажатия какой-либо клавиши именно потому, что она не выдает эхо на экран.

Все три функции не имеют параметров. Они являются функциями типа char, и их значение может быть непосредственно присвоено символьной переменной.

Пример 1.

```
char ch;
ch=getch(); /* Ввод символа с клавиатуры и присваивание его переменной ch */
putch(ch); /* Вывод введенного символа ch */
```

Пример 2. Ниже показано, как очистить экран в начале работы программы и как остановить работу программы (например, задержать изображение с результатами работы программы) до нажатия любой клавиши.

```
#include <stdio.h>
#include <conio.h>
main()
{ . . .
  clrscr();  /* Очистка экрана. Прототип функции clrscr() - в conio.h */
  . . .
  puts("Для окончания работы нажмите любую клавишу");
  getch();  /* Ввод любого знака без отображения */
}
```

1.6.4 Очистка потока

Стандартная функция fflush (описание в stdio.h) обычно используется для очистки входного потока stdin. В начале работы программы входной поток очищается автоматически.

Рекомендуется перед любой стандартной подпрограммой буферизованного ввода очищать входной поток, что обеспечит защиту от предшествующего некорректного ввода. В некоторых случаях, например, перед обращением к gets или getchar, использование fflush практически всегда обязательно.

Пример 1.

```
scanf("%d",&n);
fflush(stdin);      /*  Комментарий 1  */
scanf("%d",&i);
fflush(stdin);      /*  Комментарий 2  */
gets(str);
```

Комментарий 1: очистка входного потока stdin в случае неправильного ввода.

Комментарий 2: очистка входного потока stdin в случае неправильного ввода и для последующего нормального выполнения функции gets (надо удалить из буфера символ "конец строки", оставшийся после выполнения функции scanf, чтобы функция gets отработала нормально).

Объяснение по второму комментарию следует разобрать подробнее. Функции scanf и grtchar оставляют после ввода в потоке ввода (входной строке) код <ВК>, при очередном вводе функция scanf код <ВК> пропускает, пока не встретит знак, требуемый по спецификации формата. Функция gets не оставляет после себя код <ВК>, а если в начальный момент своей работы встречает во входной строке код <ВК> (он мог остаться после, например, работы scanf), то вводит пустую строку. Именно поэтому перед обращением к gets следует всегда использовать fflush для очистки входного потока, в частности, для удаления возможного кода <ВК>.

Пример 2. Пусть требуется ввести две строки s1 и s2, описание которых дано ниже:

```
char  s1[10], s2[10];     int x;
```

Предположим, что к текущему моменту входной поток пустой, и рассмотрим несколько вариантов ввода строк.

```
1)  scanf("%s",s1);    /*  Ввод обеих строк пройдет нормально. Использовать fflush  */
    scanf("%s",s2);    /*  перед вводом второй строки, в принципе, не обязательно    */
2)  gets(s1);   /* Ввод обеих строк пройдет нормально. Использовать fflush */
    gets(s2);   /* перед вводом второй строки, в принципе, не обязательно    */
3)  gets(s1);             /* Вторая строка  s2  вводится автоматически как пустая.  */
    scanf("%d",&x); /* Если перед оператором "gets(s2);" записать "fflush(stdin);",  */
    gets(s2);             /*  то ввод второй строки пройдет нормально    */
4)  scanf("%s",s1);   /*  Этот случай аналогичен предыдущему случаю  3)   */
    gets(s2);
```

1.7 ЛАБОРАТОРНАЯ РАБОТА №1 "ЛИНЕЙНЫЙ ВЫЧИСЛИТЕЛЬНЫЙ ПРОЦЕСС"

Линейным называется вычислительный процесс, структурная схема которого не содержит разветвлений. В алгоритме линейной структуры все действия выполняются последовательно одно за другим. Для реализации алгоритмов линейной структуры обычно используются операторы ввода-вывода и операторы присваивания.

Цель лабораторной работы "Линейный вычислительный процесс" - научиться писать на языке Си линейные программы и усвоить на примере реализации конкретной задачи основы программирования на языке Си, рассмотренные в данном модуле 1.

Задание к лабораторной работе включает в себя разработку двух программ линейной структуры. Подробное описание заданий и конкретные варианты задач приведены в п.1.7.1 и в п.1.7.2.

Отчет по лабораторной работе (этой и всех последующих) должен содержать следующие элементы:

- титульный лист, где следует обязательно указать фамилию и группу автора работы; название дисциплины, в рамках которой выполнена работа; название и номер работы;
- формулировка задания к работе;
- алгоритм решения задачи (для лабораторных работ 1,2,3 - в виде структурной схемы, для последующих лабораторных работ возможно словесное описание алгоритма),
- текст программы с комментариями (обязательные комментарии: заголовок для каждой функции, в котором описывается назначение этой функции; описание всех используемых переменных и констант; заголовки отдельных функциональных блоков программы);
- тесты, иллюстрирующие все основные варианты работы программы.

1.7.1 Линейный вычислительный процесс. Вычисление заданной величины

ЗАДАНИЕ. Разработать линейную программу для вычисления заданной величины по определенной формуле. Вариант конкретного задания выбрать из списка заданий ниже в п.1.7.1.1. Пример написания линейной программы см. в п. 1.2.3.

1.7.1.1 Линейный вычислительный процесс. Вычисление заданной величины. Варианты заданий

N 1.

Вычислить площадь треугольника по формуле

$S = \frac{1}{2} \cdot b \cdot h$, где b - основание треугольника, h - высота, опущенная на это основание.

N 2.

Вычислить площадь равнобедренного треугольника по формуле

$S = \frac{1}{2} a \cdot \sqrt{b^2 - \frac{a^2}{4}}$, где a - основание треугольника, b - боковая сторона.

N 3.

Вычислить площадь равностороннего треугольника по формуле

$S = \frac{1}{4} a^2 \cdot \sqrt{3}$, где a - сторона треугольника

N 4.

Вычислить площадь ромба по формуле

$S = \frac{1}{2} d_1 \cdot d_2$, где d_1, d_2 - диагонали ромба.

N 5.

Вычислить площадь трапеции по формуле

$S = c \cdot h$, $c = (a+b)/2$, где h - высота трапеции, c - средняя линия трапеции: $c = \frac{a+b}{2}$, где a,b - основания трапеции

N 6.

Вычислить площадь параллелограмма по формуле

$S = b \cdot h$, где b - основание параллелограмма, h - высота параллелограмма.

N 7.

Вычислить площадь правильного шестиугольника по формуле

$S = \frac{3}{2}\sqrt{3} \cdot a^2$, где a - сторона шестиугольника.

N 8.

Вычислить медиану треугольника, соединяющую вершину A с серединой противоположной стороны a, по формуле

$m = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$, где a,b,c - стороны треугольника.

N 9.

Вычислить длину окружности l и площадь круга S по формулам:

$l = 2 \cdot \pi \cdot R$, $S = \pi \cdot R^2$, где R - радиус.

N 10.

Вычислить радиус r вписанного в треугольник круга по формуле

$r = \sqrt{\frac{(p-a)(p-b)(p-c)}{p}}$, где a,b,c - стороны треугольника, p - полупериметр треугольника: $p = \frac{a+b+c}{2}$.

N 11.

Вычислить радиус R круга, описанного около треугольника, по формуле

$R = \frac{a*b*c}{4 \cdot \sqrt{p(p-a)(p-b)(p-c)}}$, где a,b,c - стороны треугольника, p - полупериметр треугольника: $p = \frac{a+b+c}{2}$.

N 12.

Вычислить высоту треугольника, опущенную на сторону a, по формуле

$h = \frac{2 \cdot \sqrt{p(p-a)(p-b)(p-c)}}{a}$, где a,b,c - стороны треугольника, p - полупериметр треугольника: $p = \frac{a+b+c}{2}$.

N 13.

Вычислить площадь между окружностью радиуса R и заключенной внутри нее окружностью радиуса r по формуле

$S = \pi \cdot (R+r) \cdot (R-r)$

N 14.

Вычислить площадь боковой поверхности цилиндра по формуле $S = 2 \cdot \pi \cdot R \cdot H$, где H - высота цилиндра, R - радиус основания.

N 15.

Вычислить объем прямоугольного параллелепипеда по формуле $V = a \cdot b \cdot c$, где a,b,c - стороны параллелепипеда.

N 16.

Вычислить объем тора, образованного вращением круга радиуса r вокруг оси, отстоящей на расстояние R от центра, по формуле $V = 2 \cdot \pi^2 \cdot R \cdot r^2$.

N 17.

Вычислить путь, пройденный телом при равноускоренном прямолинейном движении: $S = v_0 \cdot t + \frac{at^2}{2}$, где v_0 - начальная скорость, a - ускорение, t - время.

N 18.

Вычислить скорость движения спутника по орбите высоты h по формуле $v = r_з \cdot \sqrt{\frac{g_з}{r_з + h}}$, где $r_з = 6{,}37 \cdot 10^6$ м, $g_з = 9{,}81$ м/ c^2 .

N 19.

Вычислить вес тела P на поверхности земли по формуле $P = \gamma \cdot \frac{m_з \cdot m}{R_З^2}$, где $m_з$ - масса земли, m - масса тела, $R_З$ - радиус земли, $\gamma = 6{,}67 \cdot 10^{-8}$ $см^3$/(г c^2) - гравитационная постоянная.

N 20.

Вычислить работу A по формуле $A = F \cdot S \cdot \cos(a)$, где F - сила, S - перемещение, a - угол между направлениями силы и перемещения.

N 21.

Вычислить силу тяготения по формуле $F = \frac{Q \cdot m_1 \cdot m_2}{R^2}$, где Q= $6.67 \cdot 10^{-8}$ $см^3$/(г c^2) - гравитационная постоянная, m_1, m_2 - массы тел, R - расстояние между телами.

N 22.

Вычислить потенциальную энергию тела E в однородном поле земного тяготения по формуле $E = m \cdot g \cdot h$, где m - масса тела, g=9.8 м/c^2 - ускорение свободного падения, h - высота тела.

N 23.

Вычислить силу гравитационного притяжения двух тел массой m_1 и m_2 по формуле

$P = G \cdot \frac{m_1 \cdot m_2}{R^2}$, где G= $6.67 \cdot 10^{-8}$ $см^3$ /(г $с^2$) - гравитационная постоянная.

N 24.

Вычислить период математического маятника по формуле

$T = 2 \cdot \pi \cdot \sqrt{\frac{l}{g}}$, где l - длина маятника, g= 9.8 м/$с^2$ - ускорение силы тяжести.

N 25.

Вычислить по закону взаимодействия точечных зарядов (закон Кулона) силу взаимодействия F:

$A = \frac{q_1 \cdot q_2}{4 \cdot \pi \cdot e \cdot r^2}$, где q_1, q_2 - величины зарядов, e - абсолютная диэлектрическая проницаемость среды, r - расстояние между точечными зарядами.

N 26.

Вычислить период собственных колебаний контура по формуле

$T = 2\pi\sqrt{L \cdot C}$, где L - индуктивность контура, C - емкость контура.

N 27.

Вычислить сопротивление двух параллельно соединенных проводников по формуле

$R_n = \frac{R_1 \cdot R_2}{R_1 + R_2}$, где R_1, R_2 - сопротивление проводников.

N 28.

Вычислить количество теплоты, выделяемой за время t в проводнике сопротивлением R, через который проходит ток силой I, по формуле

$Q = I^2 \cdot R \cdot t$

N 29.

Вычислить емкость плоского конденсатора по формуле

$C = \frac{e \cdot S}{4 \cdot \pi \cdot d}$, где S - площадь одной пластины (меньшей, если они равны), d - расстояние между пластинами, e - диэлектрическая проницаемость материала, находящегося между обкладками.

N 30.

Вычислить емкость сферического конденсатора по формуле

$C = \frac{e}{1/a - 1/b}$, где a, b - радиусы внутренней и внешней сферы, соответственно, e - диэлектрическая проницаемость материала, находящегося между сферами.

N 31.

Вычислить общую емкость конденсаторов с емкостями C_1, C_2,... C_n при параллельном соединении по формуле

$C_{пар} = C_1 + C_2 + ... + C_n$,

а также при последовательном соединении по формуле

$\frac{1}{C_{пос}} = \frac{1}{C_1} + \frac{1}{C_2} + \cdots \frac{1}{C_n}$.

Вычисления провести для 5 конденсаторов (n=5) с заданными емкостями C_1, C_2, C_3, C_4, C_5.

1.7.2 Линейный вычислительный процесс. Расчет по формулам

Цель данной работы - получение практического опыта в разработке и реализации на языке СИ линейных программ, в записи на языке Си сложные арифметические выражения с использованием стандартных математических функций, а также закрепление знаний по темам «Операции в Си», «Ввод-вывод в Си».

1.7.2.1 Задание к лабораторной работе

Написать линейную программу, которая последовательно для двух заданных значений аргумента x вычисляет y=f(x) по двум формулам вида:

$$y= f(x) = \begin{cases} выражение\,1, & при\ x, лежащем\ в\ определенном\ интервале\,1 \\ выражение\,2, & при\ x, лежащем\ в\ определенном\ интервале\,2 \end{cases};$$

Ниже для задания вида

$$y= f(x) = \begin{cases} x - 0{,}5, & при\ \ 0 < x \le 2 \\ x + 0{,}5, & при\ \ 2 < x \le 3 \end{cases};$$

приведен примерный протокол работы программы:

```
Введите x,  0<x<=2 :  1
Для  x=1  y=0.5
Введите x,  2<x<=3 :  2.7
Для  x=2.7  y=3.2
Для окончания работы нажмите любую клавишу
```

Замечание 1. В данной программе еще не требуется проверять, принадлежит ли введенный аргумент x заданному интервалу. Поэтому возможен случай, когда после ввода x, не принадлежащего заданному интервалу, программа завершится аварийно. Например, такая ситуация произойдет, если в выражении есть операция «квадратный корень», для которой пользователь введет операнд x<0. При выполнении данной лабораторной работы эту ситуацию не следует как-то учитывать и обрабатывать. Принадлежность аргумента x определенному интервалу будет проверяться в следующей лабораторной работе «Разветвляющийся вычислительный процесс» после изучения оператора if.

Замечание 2. При выполнении данной лабораторной работы следует обратить внимание на разработку удобного интерфейса, на использование функций ввода-вывода (см. по учебнику подраздел 1.5), на формат вывода (см. по учебнику подраздел 1.5), на использование стандартных математических функций (см. по учебнику подраздел 1.1.10), на операции в Си (см. по учебнику подраздел 2.1).

Замечание 3. Устная защита по данной лабораторной работе включает в себя обсуждение следующих тем: «Типы данных в Си», «Ввод-вывод в Си», «Операции в Си».

1.7.2.2 Пример выполнения лабораторной работы

Задание

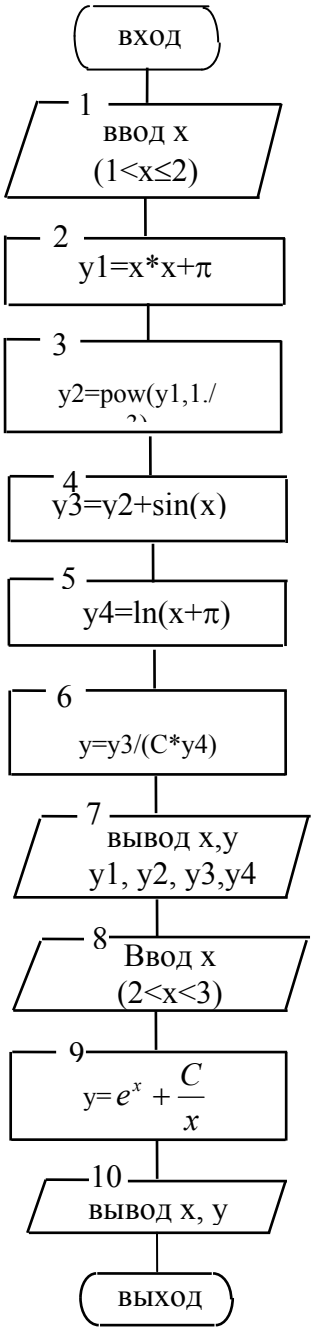
Дана функциональная зависимость y=f(x) в следующем виде:

$$y(x)=\begin{cases}\dfrac{\sqrt[3]{x^2+\pi}+\cdot\sin(x)}{1.423117*\ln(x+\pi)}, & \text{если } 1<x\le 2;\\ e^x+\dfrac{1.42317}{x}, & \text{если } 2<x<3\end{cases}$$

Написать программу для вычисления значения y(x) вначале по первой формуле для заданного значения аргумента x (предполагается, что для 1<x<=2), а затем по второй формуле для другого заданного значения аргумента x (предполагается, что для 2<x<3). Вычисление y по сложным формулам можно выполнить по частям.

Для каждой формулы задать аргумент из указанного интервала и сделать все расчеты на калькуляторе. Полученные данные использовать в качестве теста при проверке работоспособности программы.

Структурная схема алгоритма



Блоки 1-7: вычисление y по первой формуле с вычислением промежуточных результатов

Блоки 8-10: вычисление y по второй формуле

Обозначения

Дано: x- аргумент (вещественное число).
Результат: y - значение функции (вещественное число).
Промежуточные переменные: y1, y2, y3, y4 (вещественные числа)
Константы: PI = 3.141593 - константа π ,
C = 1.423117.

Программа

```
/* Вычисление y=f(x) для заданного  x  по двум формулам */
#include <stdio.h>
#include <math.h>
#include <conio.h>
#define PI 3.141593
#define C  1.412117
int main()
{
  float  x,y;
  printf("Вычисление  y=f(x) по заданному аргументу x по двум формулам \n");
(* Вычисление y по первой формуле для  1<x<=2 *);
  printf("Введите  x  (1<x<=2): ");
  scanf("%f",&x);
   y1=ln(x*x+PI);
   y2=pow(y1,1./3);
   y3=y2+sin(x);
   y4=ln(x+PI);
    y=y3/(C*y4);
  printf("  Для x=%f    y=%f\n",x,y);
(* Вычисление y по второй формуле для 2<x<3 *)
  write("Введите  x  (2<x<3): ");
  scanf("%f",&x);
    y=exp(x)+C/x;
  printf("  Для x=%f    y=%f\n",x,y);
  printf("  Для окончания работы нажмите любую клавишу\n");
  getch();
}
```

Тесты

x=1.5	y1= ...	Конкретные числовые значения
	y2= ...	
	y3= ...	
	y4= ...	
	y= ...	
x=2.5	y= ...	

Замечание 1. Поэтапное вычисление по сложным формулам с получением промежуточных результатов используется в следующих случаях:

1) формула длинная и сложная; тогда вычисление по частям делает ее нагляднее, при этом уменьшается вероятность появления ошибок при записи формулы;

2) в формуле есть одинаковые повторяющиеся фрагменты; чтобы не записывать их много раз, проще один раз записать этот фрагмент и обозначить его промежуточной переменной, а затем использовать уже эту переменную.

В данном примере поэтапное вычисление по первой формуле, в принципе, излишне и дано просто как пример такой возможности.

Замечание 2. В данном примере в операторе printf все сообщения даны на русском языке. Во многих системах русский алфавит будет распечатан неправильно. Проще всего решить эту проблему, записывая сообщения на английском языке.

Замечание 3. Обратите внимание на деление «1./3». Знак десятичной точки здесь нужен для того, чтобы деление происходило над вещественными числами, а не над целыми 1 и 3. Результат деления «1/3» будет равен 0.

1.7.2.3 Линейный вычислительный процесс. Расчет по формулам. Варианты заданий

N1

$$y(x)=\begin{cases} 5{,}329142\cdot|\pi-x|+\dfrac{5{,}329142}{\sqrt{x}+\pi\cdot x^2}, & 0<x\le 4 \\ \dfrac{5{,}329142-e^{x-\pi}+\cos^2 x}{\ln x}, & 4<x\le 5{,}8 \end{cases}$$

N2

$$y(x)=\begin{cases} \sin^2(\pi+x)+\dfrac{1{,}887452+\sqrt{x}}{x^3+1{,}887452}, & 1{,}4\le x<\pi \\ \sqrt[3]{1{,}887452+e^{x-\pi}}, & \pi\le x\le 5{,}2 \end{cases}$$

N3

$$y(x)=\begin{cases} 4{,}79854\cdot\sqrt[3]{x}+(\pi-x)^2, & 0<x\le 1{,}7 \\ \dfrac{e^{\pi+x^2}+4{,}79854\cdot\sin x}{\ln(4{,}79854+x)}, & 1{,}7<x\le 3{,}9 \end{cases}$$

N4

$$y(x)=\begin{cases} 3{,}987483\cdot\sqrt{x}+x^3, & 0\le x\le 0{,}42 \\ \dfrac{e^{x-\pi}\cdot 3{,}987483}{\sin^2(\pi-x)+\ln(3{,}987483+\sqrt[3]{x})}, & 0{,}42<x\le 3 \end{cases}$$

N5

$$y(x)=\begin{cases} 8{,}732105\cdot|x-\pi|+\dfrac{\sqrt{x}+\pi\cdot x}{8{,}732105}, & 0<x\le 1{,}75 \\ \dfrac{8{,}732105\cdot x^{\frac{1}{4}}-e^{x-\pi}}{\ln x}+\sin^2 x, & 1{,}75<x\le 2{,}05 \end{cases}$$

N6

$$y(x)=\begin{cases} \dfrac{\pi\cdot x^3+\sqrt{x}}{0{,}1274}+0{,}1274\cdot|x-\pi| & ,0<x\le 9 \\ \dfrac{0{,}1274-e^{x-\pi}+\sin^2 x}{\ln x}, & 9<x\le 10 \end{cases}$$

N7

$$y(x)=\begin{cases}\dfrac{e^{x-\pi}-2{,}423152}{2{,}423152\cdot(x-\sqrt{x})}, & 2<x\le 3\\ \sin(x^2-2{,}423152), & 3<x<5\end{cases}$$

N8

$$y(x)=\begin{cases}\sqrt[3]{x+3{,}247115}\cdot e^{x+\pi}+x^3, & 0<x<3\\ \dfrac{3{,}247115-\pi\cdot(x+\sqrt{\pi})}{3{,}247115+x}, & 3\le x\le 5\end{cases}$$

N9

$$y(x)=\begin{cases}\dfrac{2{,}379451\cdot\sin^2 x+\sqrt{2{,}379451-x}}{\left|2{,}379451-\cos x\right|}, & -1<x<1{,}8\\ e^{2{,}379451-x}+\ln(\pi+x), & 1{,}8\le x\le 4{,}1\end{cases}$$

N10

$$y(x)=\begin{cases}\dfrac{1{,}987215+\sqrt{x}+(\pi\cdot x)^3}{e^{\pi-x}}, & 1{,}3\le x\le 1{,}8\\ \dfrac{\left|\sin(1{,}987215-x)\right|}{\ln x+1{,}987215}, & 1{,}8<x\le 2{,}4\end{cases}$$

N11

$$y(x)=\begin{cases}\pi\cdot\sqrt{2{,}94784+x}+2{,}94784\cdot\sin x^2, & 0\le x\le 1{,}2\\ \dfrac{x+e^{x-2{,}94784}}{\pi+\ln(2{,}94784+x)}, & 1{,}2<x\le\pi\end{cases}$$

N12

$$y(x)=\begin{cases}2{,}78932\cdot tg(\pi\cdot x)+\sqrt{x}, & 0\le x\le 1{,}31\\ \dfrac{2{,}78932-\left|\cos(x)\right|}{2{,}78932\cdot e^{-x}}, & 1{,}31<x<\pi\end{cases}$$

N13

$$y(x)=\begin{cases}\sqrt[5]{1{,}2738}\cdot e^{x-1{,}2738}+x^3, & 0<x<3\\ \dfrac{2-\pi\cdot\lg(x+\sqrt{1{,}2738})}{3{,}28+x}, & 3\le x\le 5\end{cases}$$

N14

$$y(x)=\begin{cases}4{,}58974\cdot(\pi\cdot x)^3+\sqrt[4]{x}, & 0<x\le 1{,}275\\ \dfrac{4{,}58974-e^{\pi+x^2}}{\ln(x^2+\pi)-4{,}58974}, & 1{,}275<x\le 3\end{cases}$$

N15

$$y(x)=\begin{cases}(\pi+x^2)\cdot 3{,}789115+\sin(3{,}789115-x), & -1{,}4<x<2{,}37\\ \ln(3{,}789115\cdot x+\pi), & 2{,}37\le x\le 3\end{cases}$$

N16

$$y(x)=\begin{cases} x^3+0{,}99971\cdot\sqrt[5]{x}, & 0<x\le 2{,}42 \\ \dfrac{\sin^3(\pi\cdot x)+\ln(0{,}99971+\sqrt{x})}{e^{x-\pi}\cdot 0{,}99971}, & 2{,}42<x\le 3 \end{cases}$$

N17

$$y(x)=\begin{cases} \dfrac{\sqrt{3{,}98454+x}\cdot\cos(\pi\cdot x)}{x^3-3{,}98454}, & 0\le x\le 1 \\ e^{x+\frac{3{,}98454}{\pi+x}}, & 1<x<2 \end{cases}$$

N18

$$y(x)=\begin{cases} \dfrac{2{,}914813\cdot\sqrt[3]{x}+\sin^2 x}{2{,}914813\cdot\sqrt{x}}, & 0<x\le 1{,}4 \\ \dfrac{\pi\cdot e^{2{,}914813}-x}{\pi+\ln(x+2{,}914813)}, & 1{,}4<x<3{,}2 \end{cases}$$

N19

$$y(x)=\begin{cases} \cos^4(\pi+x)+\dfrac{x^{0{,}75}+0{,}288881}{2{,}88881+\sqrt{x}}, & 1{,}4\le x<\pi \\ \sqrt[3]{2{,}88881+e^{x^2-\pi}}, & \pi\le x\le 7{,}1 \end{cases}$$

N20

$$y(x)=\begin{cases} 0{,}47885\cdot\sqrt[4]{x}+(x-\pi)^3, & 0<x\le 1{,}78 \\ \dfrac{e^{x^3-\pi}-0{,}47885\cdot\cos x}{\ln(0{,}47885+x^3)}, & 1{,}78<x<3.9 \end{cases}$$

N21

$$y(x)=\begin{cases} \sqrt[5]{9{,}876543+x}\cdot e^{x^2+\pi}+x^3, & 1<x\le 2 \\ \dfrac{9{,}876543-\pi\cdot\lg(x+\pi)}{9{,}876543+x}, & 2<x\le 3 \end{cases}$$

N22

$$y(x)=\begin{cases} \dfrac{5{,}4321\cdot\cos^2(x+1)+\sqrt[3]{5{,}4321-x}}{\left|5{,}4321-\cos x\right|}, & 1<x\le 2 \\ e^{5{,}4321+x^2}+\ln(\pi+x^3), & 2<x\le 2{,}5 \end{cases}$$

N23

$$y(x)=\begin{cases} \dfrac{5{,}12345+(x\cdot\pi)^5-\sqrt{x}}{e^{x+\pi}}, & 0{,}25\le x\le 0{,}5 \\ \dfrac{\left|\cos(5{,}12345+x)\right|}{\ln x+5{,}12345}, & 0{,}5<x\le 0{,}75 \end{cases}$$

N24

$$y(x)=\begin{cases} \dfrac{2{,}99981\cdot\cos x^5+\pi\cdot\sqrt[3]{2{,}99981\cdot x}}{e^x}, & 0\le x\le 1{,}3 \\ \dfrac{x^3+e^{x-2{,}99981}}{\pi+x+\ln(x+\pi)}, & 1{,}3<x<2{,}3 \end{cases}$$

N25

$$y(x)=\begin{cases}\dfrac{0{,}7654321\cdot tg(\pi\cdot x^3)+\sqrt{x}}{x^7}, & 0<x\le 2\\ \dfrac{0{,}7654321-|\cos x|}{0{,}7654321\cdot\ln x}, & 2<x\le 4\end{cases}$$

N26

$$y(x)=\begin{cases}\dfrac{\sqrt[3]{0{,}765111}\cdot e^{0{,}765111+x}+x^3}{0{,}765111+x}, & 0<x<1\\ \dfrac{2-\pi\cdot\lg(x+\sqrt{x+0{,}765111})}{\left|x-(0{,}765111)^5\right|}, & 1\le x\le 2\end{cases}$$

N27

$$y(x)=\begin{cases}\dfrac{0{,}77777\cdot x^3+\sqrt[7]{x}}{\ln(0{,}77777+\sqrt{x})}, & 0<x\le 2{,}05\\ \sin^3(\pi\cdot x)+e^{x^3-\pi}\cdot 0{,}77777, & 2{,}05<x<3\end{cases}$$

N28

$$y(x)=\begin{cases}\dfrac{1{,}234567-e^{x^2-\pi}}{1{,}234567\cdot(\sqrt[3]{x}-x)}, & -1<x\le 3\\ \sin(x^2-1{,}234567), & 3<x<4\end{cases}$$

N29

$$y(x)=\begin{cases}\cos^2(x+\pi)+\dfrac{1{,}275+\sqrt{x}}{x^3+1{,}275}, & 1\le x\le\pi\\ \sqrt[5]{1{,}275+e^{x-\pi}+x^2}, & \pi<x\le 4\end{cases}$$

N30

$$y(x)=\begin{cases}1{,}2345\cdot\sqrt[3]{x+1}+(\pi-x)^2, & 0<x\le 0{,}7\\ \dfrac{e^{\pi+x}+1{,}2345\cdot\sin^2 x}{\ln(x+1.2345)}, & 0{,}7<x<1\end{cases}$$

1.8 КОНТРОЛЬНЫЕ ВОПРОСЫ ПО МОДУЛЮ 1

1.8.1 Основные этапы решения задач на ЭВМ
1.8.2 Основные элементы структурных схем
1.8.3 Структура и основные элементы Си-программы.
1.8.4 Препроцессор языка Си; основные команды препроцессора define и include.
1.8.5 Константы в языке Си.
1.8.6 Основные типы в Си. Преобразование типов.
1.8.7 Стандартные функции ввода в Си.
1.8.8 Стандартные функции вывода в Си.

2 ОПЕРАЦИИ И ОПЕРАТОРЫ

Цель изучения данного модуля - познакомиться с основными операциями и основными операторами управления в языке Си, а также приобрести опыт разработки разветвляющихся и циклических программ на языке Си.

2.1 ОПЕРАЦИИ В СИ

2.1.1 Арифметические операции

Операции в Си в зависимости от числа операндов бывают унарные и бинарные.

Унарная арифметическая операция- это унарный минус (-).

Бинарные арифметические операции - это сложение (+), вычитание (-), умножение (*), деление (/), нахождение остатка при целочисленном делении (%). Специальной операции целого деления в Си нет. Существует правило: если делимое и делитель целого типа, то деление производится нацело, то есть дробная часть результата отбрасывается и результат деления получается типа integer. Операция нахождения остатка допустима только для целых операндов.

Пример1. Ниже иллюстрируется выполнение арифметических операций: в левом столбике приводится выражение, в правом - значение этого выражения в виде комментария.

```
    -5              /**   " -5 "   **/
  2+3*4-5           /**   "  9 "   **/
  2+3/4-5           /**   " -3 "   **/
  2+3%4-5           /**   "  0 "   **/
  2+3.0/4-5         /**   "-2.25" **/
```

Пример 2. Ниже приведено два примера, которые иллюстрируют особенности выполнения операции деления "/' для целочисленных и вещественных операндов.

```
1) int a,b;
   float c;
   a=10; b=3;
   c=a/b;        /*  Результат: c=3.00.  Оба операнда a и b - целые, поэтому в результате
деления получилось целое число 3, которое затем присвоилось переменной c типа float и
преобразовалось в дробное число 3.00  */
2) float a,b,c;
   a=10; b=3;
   c=a/b;        /*  Результат: c=3.33.  Оба операнда a и b - с плавающей точкой, поэтому в
результате деления сразу получилось дробное число 3.33 , которое затем присвоилось
переменной c типа float  */
```

Пример 3. Приведенные ниже примеры иллюстрируют особенности выполнения операции деления "/" совместно с операцией приведения типа (см. п.1.4.3).

```
      int i=5;   float x;
1) x = i/2;            /*  Результат: x=2.000000  */
2) x = (float)i/2;     /*  Результат: x=2.500000. До начала деления операнд  i  преобразуется
                           к вещественному типу,  а если один из операндов  -  вещественный, то
                           и результат деления  -  с плавающей точкой  */
3) x = (float)(i/2);   /*  Результат: x=2.000000. Этот случай аналогичен случаю  1)  */
```

2.1.2 Увеличение и уменьшение

Для увеличения или уменьшения значения переменной на 1 существуют специальные операции:

"++" - увеличение;

"--" - уменьшение.

Пример1.

```
x++;    /*  Увеличение x на 1, т.е. этот оператор эквивалентен оператору x=x+1;  */
++a;    /*  Увеличение a на 1, т.е. этот оператор эквивалентен оператору a=a+1;  */
y--;    /*  Уменьшение y на 1, т.е. этот оператор эквивалентен оператору y=y-1;  */
--b;    /*  Уменьшение b на 1, т.е. этот оператор эквивалентен оператору b=b-1;  */
```

Операции увеличения и уменьшения можно использовать в выражениях. Существенно, с какой стороны от операнда стоит знак "++" или "--".

Если знак операции стоит перед именем (префиксная форма операции), то сначала выполняется увеличение (или уменьшение), а затем полученный результат используется в выражении.

Если знак операции стоит после имени переменной (постфиксная форма операции), то в выражении используется старое значение переменной, которое изменяется после вычисления программой данного выражения.

Пример 2.

```
int  i, j, x;
i=2;   j=2;       /*  i=2, j=2  */
x=(i++)+(--j);  /*  i=3, j=1; x=істар.+jнов.=2+1=3  */
```

Пример 3. Не следует в одном и том же выражении использовать несколько операций увеличения или уменьшения для одной и той же переменной, так как результат в этом случае сложно предугадать. Эту мысль иллюстрирует приведенная ниже программа.

```
#include <stdio.h>
int main()
{ int y=5, i=1;
  y=y+(++i)+(++i);
  printf("y=%d, i=%d",y,i);    /*  Результат: y=11, i=3.  */
}
```

Пример 4. Не следует в одном и том же выражении использовать операцию увеличения (уменьшения) и операцию присваивания для одной и той же переменной, так как результат в этом случае бывает сложно предугадать.

```
#include <stdio.h>
int main()
{ int y, i, j=100;
   y=(i=(j=1)+4)/(++j);
   printf("\ny=%d i=%d j=%d",y,i,j);    /*  Результат:  y=5  i=5  j=1  */
}
```

2.1.3 Операция присваивания

Знак присваивания - это знак равенства " = ". Общий вид присваивания:

< имя >=< выражение >

Тип выражения в правой части преобразуется к типу идентификатора в левой части. Следует иметь в виду, что при преобразовании "большего" типа к "меньшему", например, float в int, значение выражения "обрезается" и старшие разряды просто игнорируются.

Пример1.

```
int x, y, a;
x=5;        /*  x=5   */
y=x*2+7;    /*  y=17  */
a=y/4;      /*  a=4   */
```

Операция присваивания, кроме того, что выполняет присваивание, поставляет в качестве результата присвоенное значение. Так, например, операция "+" производит сложение и поставляет полученный результат: 2+3 → результат этого выражения 5; аналогично и операция "=" производит присваивания значения переменной и поставляет полученный результат: y=5 → результат этого выражения 5. Поэтому в Си разрешено включать присваивание в выражение.

Пример 2. Сравните этот пример с примером 1 выше.

```
a=(y=(x=5)*2+7)/4;      /*  x=5, y=17, a=4  */
```

Пример 3. В Си возможно многократное присваивание, которое выполняется, в отличие от большинства операций, справа налево.

```
a=b=c=x*y;              /* Сначала вычисляется значение x*y, затем это значение
                           присваивается  c, потом  b, и лишь затем  a */
```

Для бинарных арифметических и поразрядных операций (обозначим такую операцию как "#") существует специальная операция присваивания, обозначаемая как "#=". Присваивание

< имя > #= < выражение >

эквивалентно присваиванию

< имя > = < имя >#< выражение >.

Следует иметь в виду, что операция " x+=5 " выполняется быстрее, чем операция " x=x+5 ".

Пример 4.

```
int x,y;
x=y=5;     /*  x=5,  y=5  */
x+=1;      /*  x=6    */
y-=3;      /*  y=2    */
x*=y;      /*  x=12  */
x/=++y;    /*  x=4, так как   x = x/++y=12/++2=12/3=4;   y=3  */
x%=y++;  /*  x=1, так как   x=x%y++=4%3++=1,    y=4  */
```

2.1.4 Логические операции и операции отношения

Операции отношения используются для сравнения, обычно в условных выражениях, или, короче, в условиях. Список операций отношения в Си:

```
< .   меньше,
<=    меньше или равно,
>     больше,
>=    больше или равно,
==    равно (два знака "равно" подряд без пробела),
!=    не равно.
```

В языке Си есть три логические операции:

&& операция "и" (аналог AND в Паскале),

|| операция "или" (аналог OR в Паскале),

! операция "не" (аналог NOT в Паскале).

Подробно выполнение операций отношения и логических операций рассматривается в п. 2.2.2, посвященном построению условий в Си.

2.1.5 Поразрядные (побитовые) операции

Эти операции можно производить только с целочисленными операндами, результат операций - так же целый.

Побитовые операции дают возможность манипулировать с двоичными образами значений (с битами). Они позволяют обеспечить доступ к каждому биту информации. Эти операции часто используются в программах, связанных с принтером, модемом и др. устройствами.

Отличие поразрядных операций от логических и операций отношения (см. п.2.1.4) в том, что последние всегда дают в результате 0 или 1 ; для поразрядных операций это не так. Поэтому не следует поразрядные операции использовать в логических выражениях; операции дадут верный результат, только если операнды будут равны точно 0 или 1, а в логических выражениях операнды могут быть любые.

Ниже в таблице приведены названия поразрядных операций (первый столбик), их обозначения и примеры выполнения (второй и третий столбик).

Название операции	Пример	Результат
Сдвиг влево	7 << 5	224 или E016
Сдвиг вправо	7 >> 5	0
Дизъюнкция битов(ИЛИ)	7 | 5	7
Исключающее ИЛИ	7 /\ 5	2
Конъюнкция битов(И)	7 & 5	5
Обращение(отрицание битов)	~ 5	-6

Замечание. При операциях сдвига " << " и " >> " правый операнд преобразуется к типу int, а левый операнд сдвигается на соответствующее число разрядов. Освобождающиеся разряды заполняются нулями.

Пример. Ниже подробно объяснено выполнение основных битовых операций. Слева дано описание переменной и присваиваемое ей значение, справа в квадратных скобках - двоичный образ соответствующего значения.

```
1) int  a = -2;          /*    [a] = 1111 1111 1111 1110  */
   int  rez = ~a;        /*  [rez] = 0000 0000 0000 0001  */
                /* переменной res присваивается начальное значение 1 */
2) int  a = 10;          /*    [a] = 0000 0000 0000 1010  */
   int  b = 12;          /*    [a] = 0000 0000 0000 1100  */
   int  rez = a&b;  /*  [rez] = 0000 0000 0000 1000  */
                /* переменной res присваивается начальное значение 8 */
3) int  a = 10;          /*    [a] = 0000 0000 0000 1010  */
   int  b = 12;          /*    [a] = 0000 0000 0000 1100  */
   int  rez = a/\b;      /*  [rez] = 0000 0000 0000 0110  */
                /* переменной res присваивается начальное значение 6 */
4) int  a = 10;          /*    [a] = 0000 0000 0000 1010  */
   int  b = 12;          /*    [a] = 0000 0000 0000 1100  */
   int  rez = a|b;       /*  [rez] = 0000 0000 0000 1110  */
```

/* переменной res присваивается начальное значение 14 */

2.1.6 Операции: приоритет и порядок вычислений

В таблице ниже приведены все основные операции языка Си с указанием их приоритета. Большинство не рассмотренных пока операций изучается в следующих разделах. Отметим, что в Си круглые и квадратные скобки рассматриваются как операции, причем эти операции имеют наивысший приоритет.

ПРИОРИТЕТ	ОПЕРАЦИЯ	НАЗВАНИЕ
Высший	( )	Скобки
	[ ]	Квадратные скобки
Высший -1	+ +	Увеличение
	- -	Уменьшение
	(тип)	Приведение
	*	Содержимое
	&	Адрес
	-	Унарный минус
	~	Обращение
	!	Логическое НЕ
	Sizeof	Определение памяти
Высший -2	*	Умножение
	/	Деление
	%	Остаток
Высший -3	+	Сложение
	-	Вычитание
Высший -4	>>	Сдвиг вправо
	<<	Сдвиг влево
Высший -5	>	Больше
	>=	Больше или равно
	<=	Меньше или равно
	<	Меньше
	==	Равно
	!=	не равно
Низший +5	&	Поразрядное И
Низший +4	^	Исключающее ИЛИ
Низший +3	|	Поразрядное ИЛИ
Низший +2	&&	Логическое И
Низший +1	||	Логическое ИЛИ
Низший	?:	Условная операция
Низший - 1	=	Присваивание
	+=	Присваивание со сложением
	-=	и т.д.
	*=	
	/=	
	%=	
	>> =	
	<< =	
	&=	
	^=	
	|=	
Низший -2	,	Запятая

Каждая бинарная операция имеет определенный порядок выполнения. Большинство операций, например, операции сложения и умножения, выполняются слева направо. Некоторые операции, например, операция присваивания, выполняются справа налево.

2.2 ОПЕРАТОРЫ УПРАВЛЕНИЯ

2.2.1 Выражения и операторы

Виды операторов: пустой, простой и составной (см. п.1.2.6).

В Си любое выражение может быть преобразовано в оператор добавлением к нему точки с запятой, то есть запись вида:

выражение;

является оператором. Значение выражения при этом игнорируется. Обычно действие такого оператора состоит в создании побочного эффекта при вычислении значения выражения.

Пример подобного оператора - это оператор присваивания (см. п.2.1.3). Оператор присваивания - это выражение вида

< имя переменной > = < выражение >,

преобразованное в оператор добавлением к нему точки с запятой.

Примеры ниже иллюстрируют рассматриваемую связь выражений и операторов:

x=2 - это выражение, в котором используется операция присваивания;

x=2; - это оператор, полученный из предыдущего выражения добавлением к нему ";";

a++ - это выражение, в котором используется операция увеличения;

a++; - это оператор, полученный из предыдущего выражения добавлением к нему ";".

2.2.2 Построение условий

В Си существует 6 операций отношения (сравнения):

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

Замечание. Не нужно путать знак сравнения "==" (равно) и присваивания "=" (присвоить).

Операции отношения используются для сравнения значений двух выражений. Их результатом может быть истина или ложь. Истина кодируется в Си как 1, ложь - как 0, поэтому результат операции отношения можно рассматривать как число. Примеры выражений, использующих операции отношения: 2<3, a==b, k+y>7.

Сложное условие вида "1<=x<=2" трактуется в Си следующим образом. Результат сравнения 1<=x преобразуется в число 0 (ложь) или 1 (истина), и это число сравнивается с числом 2. Так как и 0, и 1 меньше 2, в результате всегда получится истина.

В подобных случаях для построения сложных условий используются логические операции:

1) "!" - логическое отрицание; "!a" (читается "не a") истинно, если a - ложно, и наоборот (not - аналог в Паскале);
2) "&&" - логическое И; "a&&b" (читается "a и b") истинно, если выражения a и b оба истинны, и ложно, если хотя бы одно из них ложно (and - аналог в Паскале);
3) "||" - логическое ИЛИ; "a||b" (читается "a или b") истинно, если среди выражений a и

b хотя бы одно истинно, и ложно, если оба выражения a и b ложны (or - аналог в Паскале).

Логические операции трактуют любые отличные от нуля операнды как 1 (истина). Нулевой операнд считается ложью.

Пример 1. 7&&5 → результат "1" (И&&И => И)
7||0 → результат "1" (И||Л => И)
!0 → результат "1" (!Л => И)

Приоритет операций сравнения в Си выше, чем логических операций &&, ||. Этот факт иллюстрируется в примере 2 ниже.

Пример 2. 1<=X && X<=2 /* X лежит в интервале [1,2] */
!(1<=X && X<=2) /* X не лежит в интервале [1,2] */
a>0 || b>0 || c>0 /* Среди чисел a, b, c есть положительные */

2.2.3 Разветвление

2.2.3.1 Условный оператор if

2.2.3.1.1 Оператор if имеет две формы:

if (l) s1
if (l) s1 else s2 где l - выражение, s1, s2 - операторы.

Выполнение оператора. Вычисляется выражение l. Если оно истинно (не равно 0), то выполняется оператор s1. Если выражение ложно (равно 0), то для первой формы выполнение оператора заканчивается, а для второй - выполняется оператор s2.

Операторы s1, s2 могут быть пустыми, простыми или составными операторами. Все операторы, кроме составных, кончаются ";" обязательно (в отличие от Паскаля, где перед словом "else" знак ";" не ставится).

Условие l оператора if может быть произвольным выражением, а не обязательно выражением, содержащим операцию сравнения или логическую операцию.

Пример 1.

```
/* Выбор максимального из двух целых чисел a и b */
#include <stdio.h>
 int main()
 {
  int a,b,c;
  printf("Введите два целых числа: ");
  scanf("%d%d",&a,&b);
  if (a>b)
    c=a;
  else
    c=b;
  printf("Дано: a=%d , b=%d \n",a,b);
  printf(" Наибольшее число c=%d \n",c);
 }
```

Пример 2.

```
/* Определение площади прямоугольника */
#include <stdio.h>
 int main()
 {
```

```
  float a,b,     /* Стороны прямоугольника */
        s;       /* Площадь прямоугольника */
  printf("\nВведите стороны прямоугольника: ");
  scanf("%f %f",&a,&b);
  if (a>0 && b>0)
    {
      s=a*b;
      printf("\n Площадь прямоугольника равна %f",s);
    }
  else
      printf(" Прямоугольник не существует");
  }
```

Пример 3.

```
/*  Замена числа его модулем  */
#include <stdio.h>
 int main()
 { float a;
   printf("\nВведите число: ");
   scanf("%f",&a);
   if (a<0) a=-a;
   printf("Модуль числа равен %f",a);
 }
```

2.2.3.1.2 Для вложенных конструкций if ключевое слово else относится к самому внутреннему слову if.

Конструкция вида:

```
if (L1)
  if (L2)
    S1
else
    S2
```

трактуется как:

```
if (L1)
{ if (L2)
    S1
  else
    S2 }
```

Если логика программы требует иного, используются фигурные скобки или пустой оператор, например:

```
if (L1)                    if (L1)
 { if (L2)                   if (L2)
      S1   }                    S1
else S2                    else
                                ; /* ";" обозначает здесь пустой оператор */
                           else S2
```

Пример 4.

```
/* Определение вида треугольника по трем сторонам */
#include <stdio.h>
  int main()
  { float a,b,c;     /* стороны треугольника */
    printf("\nВведите стороны треугольника: ");
    scanf("%f %f %f",&a,&b,&c);
    if (a>b+c || b>a+c || c>a+b)
      printf("\n Несуществующий треугольник");
    else  if (a==b && a==c)
            printf("\n Равносторонний треугольник");
          else if (a==b || a==c || b==c)
                 printf("\n Равнобедренный треугольник");
```

```
            else printf("\n Разносторонний треугольник");
}
```

2.2.3.1.3 Любое выражение с операцией присваивания, заключенное в круглые скобки, является выражением с определенным значением, которое получается в результате этого присваивания. Например, выражение (sum=5+3) имеет значение 8. Подобные выражения в круглых скобках часто используются в условии оператора if для сокращения текста программы.

Пример 5.

```
/* Фрагмент программы. Символ вводится с клавиатуры и присваивается переменной ch.
Затем введенный символ сравнивается с символом 'q' :если введено 'q', выдается
сообщение "конец работы программы", в противном случае выдается сообщение
"продолжение работы программы". */
 if ((ch=getch())=='q') /* (ch=getch()) имеет значение введенного знака */
     puts( "конец работы программы");
 else
     puts("продолжение работы программы");
```

2.2.3.1.4 В Си есть понятие "множественные выражения в круглых скобках". Для организации множественных выражений, расположенных внутри круглых скобок, используется операция запятая ",". Выражения внутри скобок вычисляются слева направо. Всё выражение как единое выражение в круглых скобках принимает значение, которое было вычислено последним. Операция запятая имеет самый низкий приоритет из всех операций языка Си.

Пример 6. Если ,например, oldch и ch - переменные типа char, то выражение (oldch=ch,ch=getch()) присваивает переменной oldch значение ch, затем считывает символ, вводимый с клавиатуры, и запоминает его в ch. Результатом всего выражения, в итоге, будет значение введенного с клавиатуры символа.

Пример 7.

```
/* Фрагмент программы. Будет выведено сообщение " Это символ 'b' " */
 ch='a';
 if ((oldch=ch,ch='b')=='a')
     puts(" Это символ 'a'\n");
 else
     puts(" Это символ 'b'\n");
```

При организации множественных выражений, перечисляемых через запятую, можно использовать не только выражения с операцией присваивания; можно использовать, например, обращение к функции вывода (см. пример 2 в п.2.2.3.2 ниже).

2.2.3.2 Ветвление в выражениях

В языке СИ есть условная операция , обозначим ее "?:", которую можно применять в выражениях. Ее вид:

< условие > ? < выражение 1 > : < выражение 2 >

При выполнении условной операции "?:" сначала проверяется < условие >. Если оно истинно, то вычисляется < выражение 1 >, если ложно - < выражение 2 >.

Условная операция может входить в любое выражение, ее результат - это результат <выражения1> или <выражения2 > в зависимости от условия.

Пример 1. Сравните этот пример с примером 1 из п.2.2.3.1.1.

```
/* Выбор максимального из двух целых чисел a и b с использованием условной
операции "?:" */
#include <stdio.h>
```

```
int main()
 {
  int a,b,c;
  printf("Введите два целых числа: ");
  scanf("%d%d",&a,&b);
     c= (a>b)? a:b ;
  printf("Дано: a=%d, b=%d \n",a,b);
  printf("Наибольшее число c=%d \n",c);
 }
```

Пример 2. Сравните этот пример с примером 2 из п.2.2.3.1.1.

```
/* Определение площади прямоугольника с использованием условной операции  "?:".
```

Замечание 1. При организации множественных выражений, перечисляемых через запятую, можно использовать операторы, например оператор вызова функции printf. В конце операторов ";" в этом случае на ставится, то есть это получается, по сути, не оператор, а выражение.

Замечание 2. Любое выражение, в том числе выражение, содержащее условную операцию, может быть преобразовано в оператор добавлением к нему точки с запятой.

Замечание 3. Если <выражение 1> или <выражение 2> для операции "?:" содержит знак "равно" ("="), то это выражение надо обязательно заключить в круглые скобки. */

```
 #include <stdio.h>
 int main()
 {
   float a,b,   /* Стороны прямоугольника */
           s;    /* Площадь прямоугольника */
   scanf("%f %f",&a,&b);
   (a>0 && b>0 ) ? (s=a*b, printf("s=%f",s)) :
                        (printf("Такой прямоугольник не существует"));
 }
```

Распространенная ошибка: операция "?:" используется во вложенных конструкциях define. Лучше так не делать, так как результат подстановок препроцессора трудно предсказуем.

Пример 3.

```
#include <stdio.h>
#define  max(x,y)  (x>y)?x:y
 void main(void)
 { int  a=2, b=1, c=3;
    printf("\n %d",max(b,c));             /* Вывод: 3. Это правильный результат */
    printf("\n %d",max(a,max(b,c)));  /* Вывод: 2. Это неправильный результат */
 }
```

2.2.3.3 Оператор переключения switch

Этот оператор реализует выбор дальнейших действий по значению выражения и используется для разветвления программы по нескольким направлениям (похож на оператор case в Паскале).

Общий вид оператора:

```
        switch (L)
        { case c1: s1;break;
          case c2: s2;break;
```

```
    ...
    case cn: sn;break;
    default: s0;break;
  }
```

где L - целое выражение (или выражение, которое может быть преобразовано в целое);

ci - метка; целое выражение с постоянным значением, обычно константа (или выражение, которое может быть преобразовано к указанному);

si - операторы, число которых больше или равно 0.

Служебные слова: switch - переключатель,
case - выбор, альтернатива,
default - умолчание,
break - разрыв.

Выполнение оператора. Вычисляется выражение L. Его значение поочередно сравнивается с константами c1,c2,... и т.д. При совпадении значения выражения L и какой-то метки-константы, например, ck, выполняются операторы sk. Если значение выражения L не совпало ни с одной из меток-констант, то выполняются операторы s0, указанные после слова default.

Замечание 1. Не требуется никакой упорядоченности при записи меток и default. Так, default может быть включен в любое место внутри оператора switch, а не обязательно в конец.

Замечание 2. Default может вообще отсутствовать. В этом случае, если выражение L не совпало ни с одной из меток, то никакие действия при выполнении оператора switch не производятся.

Замечание 3. Метки c1, c2 и т.д. должны быть уникальны.

Замечание 4. При выполнении оператора break (разрыв) управление передается на оператор, следующий за switch. Употребление операторов break не является обязательным. Если break отсутствует в конце какой-то группы операторов si, то выполняются все операторы, начиная с указанной группы (то есть si, si+1, si+2 и т.д.), пока не встретится break или не кончится оператор switch.

Замечание 5. Если действие двух или более альтернатив совпадает, то их можно объединить следующим образом:

```
switch (L)
{ case c11:        /*   Для меток  c11, c12,...,c1m  выполняются */
  case c12:        /*   общие действия  S1, то есть оператор  S1 */
  . . .            /*   помечен несколькими метками    */
  case c1m: S1;break;
  case c2: S2;break;
  ...
  }
```

Замечание 6. Для того, чтобы программа была читабельной и легко модифицируемой, рекомендуется использовать операторы break и альтернативу default (даже в случае, если действия по default заключаются в выполнении пустого оператора, который может быть просто опущен).

Пример 1.

```
/* Примитивный калькулятор */
#include <stdio.h>
int main()
{ float a,b;   /*  Два операнда  */
  char c;      /*  Операция      */
   printf("\nВведите подряд без пробелов: операнд,  операция, операнд");
   scanf("%f%c%f",&a,&c,&b);
   switch (c) {
```

```
        case '+': printf("=%f\n",a+b);break;
        case '-': printf("=%f\n",a-b);break;
        case '*': printf("=%f\n",a*b);break;
        case '/': printf("=%f\n",a/b);break;
        case '>':
        case '<':
        case '=': printf("\n Неарифметическая операция");
        default: printf("\n Неизвестная операция");
    }    /*  Конец оператора switch  */
}
```

Пример 2.

```
/* Автосоветчик */
#include <stdio.h>
int main()
{ typedef enum {sun,mon,tue,wed,thu,fri,sat} days;
  days today;
    printf("Введите день недели (0-воскр., 1-понед.,...)");
    scanf("%d",&today);
    switch (today) {
      case mon:
      case tue:
      case wed:
      case thu:
      case fri: puts("Изучайте СИ!");break;   /*  Для рабочих дней  */
      case sat: printf("Займитесь личными делами и ");
      case sun: printf("Расслабьтесь, а затем продолжите изучение СИ ");
    }
}
```

2.2.4 Циклы

Существует 3 вида операторов цикла: цикл с предусловием while, цикл с параметром for, цикл с постусловием do.

2.2.4.1 Оператор цикла с предусловием while

Общий вид оператора:

```
while  (L)  S
```

где L - выражение, S - оператор (простой или составной)

Выполнение оператора. Вычисляется выражение L. Если оно истинно (отлично от 0), то выполняется оператор S, затем снова вычисляется выражение L и т.д. Выполнение прекращается, когда при очередной проверке выражение L окажется ложным (равным 0).

Во избежание зацикливания внутри цикла должны быть действия, которые изменяют значение выражения L.

Аналогичный оператор Паскаля - while.

Пример 1.

```
/* Нахождение наибольшего общего делителя двух целых чисел */
#include <stdio.h>
```

```
int main()
{ int x,y;     /* Два исходных целых числа */
  printf("Введите два целых числа");
  scanf("%d%d",&x,&y);
  while (x!=y)    /* пока X не равно Y */
        {             /* начало цикла    */
         if (x>y)       /* если x>y         */
           x-=y;             /* то  x=x-y   */
         else            /* иначе             */
           y-=x;             /* y=y-x        */
        }             /* конец цикла      */
  printf("НОД = %d",x);     /* Вывод результата */
}
```

Пример 2.

```
/* Подсчет числа символов в строке и печать символов */
#include <stdio.h>
    int main()
    {char ch;  /* ch - очередной символ */
     int len;   /* len - длина строки */
     len=0;
     puts("Наберите предложение, затем нажмите <Ввод>");
        while ((ch=getch())!='\n')
           { putch(ch); len++; }
     printf("\nВаше предложение имеет длину %d символов\n",len);
    }
```

2.2.4.2 Оператор цикла с постусловием do

Общий вид оператора:

```
do  S  while  (L)
```

где S - оператор, L - выражение.

Выполнение оператора. Сначала выполняется оператор S, затем вычисляется выражение L. Если оно истинно, опять выполняется оператор S, и т.д. Когда выражение L становится ложным, выполнение цикла прекратится.

Аналогичный оператор Паскаля - repeat, но есть отличие: в Паскале оператор S повторно выполняется, если выражение L ложно, а в Си - наоборот, если L истинно..

Пример 1.

```
/*  Ввод чисел и подсчет их суммы */
/*  0 - признак конца ввода */
#include <stdio.h>
int main()
{ int s,x;
   printf("Введите целые числа ");
   printf("0 - признак конца ввода ");
   s=0;
   do {                /* Начало цикла */
     scanf("%d",&x);
     s+=x;
   }                   /* Конец цикла */
```

```
   while (x!=0);      /* Пока x не равно 0 */
   printf("s=%d",s);
}
```

Пример2.

```
/* Деление двух чисел */
#include <stdio.h>
int main()
{ float a,b,ratio;    /* ratio=a/b */
  char ch;
    do {
      printf("Введите два числа: ");
      scanf("%f%f",&a,&b);
      if (b==0.0)
          printf("\nВнимание! Деление на 0!");
      else {
           ratio=a/b;
           printf("\nРезультат деления двух чисел: %f",ratio);
            }
      printf("\nНажмите 'q' для выхода или любую  клав. для продолжения") ;
    } while ((ch=getch())!='q');
}
```

Пример 3.

```
/*  Фрагмент программы для проверки того,    */
/*  что аргумент x лежит в интервале от 0 до 1 */
#include <stdio.h>
int main()
{
 float x;
  do {
     printf("Введите x от 0 до 1, то есть в интервале [0,1]: ");
     scanf("%f",&x);
     if(x<0 || x>1)
         printf("Вы ввели недопустимый x\n");
  } while(x<0 || x>1);
}
```

2.2.4.3 Оператор цикла с параметром for

Общий вид оператора:

```
for  (L1; L2; L3)  S
```

где S - оператор (простой или составной);

L1, L2, L3 - выражения (их называют "заголовок for"):

L1 - задает начальные условия выполнения цикла (L1-старт, инициализация);

L2 - обеспечивает проверку условия выхода из цикла (L2-условие);

L3 - модифицирует условия, заданные выражением L1 (L3-шаг, изменение).

Выполнение оператора. Вычисляется выражение L1 (начальное присваивание). Затем вычисляется условие L2. Если оно ложно, цикл заканчивается, если истинно - выполняется оператор S. Затем вычисляется выражение L3 (шаг) и снова вычисляется условие L2 и т.д.

Цикл for полностью эквивалентен следующему циклу while:

```
L1;
while (L2)
        { S;
          L3;
        }
```

Любое из выражений L1, L2, L3 может быть опущено, однако соответствующие точки с запятой должны быть всегда.

Цикл for (;L2;) S; эквивалентен циклу
while (L2) S;

Если опущено L2, то по умолчанию вместо него всегда подставляется значение 1 (истина), в результате получается бесконечный цикл.

Цикл for (; ;) S; эквивалентен циклу
while (1) S;

то есть это также бесконечный цикл (может быть прерван явным выходом с помощью операторов break, goto, return, входящих в тело цикла S).

Цикл for называют циклом с параметром, или параметрическим циклом. Для языка Си это название достаточно условно, так как параметр может отсутствовать, как могут отсутствовать и любые из выражений L1, L2, L3/

Пример 1.

```
/* Суммирование первых n натуральных чисел */
#include <stdio.h>
int main()
    { int i,n,s;
      s=0;
      printf("Введите n");
      scanf("%d",&n);
      for (i=1; i<=n; i++)   /* В данном случае есть параметр цикла – переменная i */
            s+=i;
      printf("S=%d",s);
    }
```

Пример 2.

```
/* Определение, является ли число простым. Простые числа: 2, 3, 5, 7, 11, … */
#include <stdio.h>
int main()
{ int x,   /* Проверяемое число */
       i;    /* Возможный делитель */
  do { printf("Введите натуральное число (не менее 2): ");
       scanf("%d",&x);  }
  while (x<=1);
  for (i=2; i<=x/2 && x%i!=0; i++);
  if (i<=x/2)
    printf("Число составное");
  else
    printf("Число простое");
}
```

/* Внимание! В цикле for используется пустой оператор, так как все необходимые действия поместились в заголовке цикла */

С помощью оператора запятая можно вводить составные выражения в заголовок оператора for.

Пример 3.

```
/* Использование составных выражений в операторе for */
#include <stdio.h>
int main()
{
 int up,down;
 for (up=1,down=9;  up<=10;  up++, down--)
        printf("up=%d растет,  down=%d уменьшается ",up,down);
}
```

Наиболее существенное отличие рассматриваемого оператора for от аналогичного в Паскале заключается в том, что условие окончания цикла L2 и шаг L3 перевычисляются при каждой итерации. Это означает, что если некоторый идентификатор, используемый в L2 или L3, внутри цикла изменяется, то могут постоянно изменяться при прохождении тела цикла и условие окончания цикла L2, и шаг L3.

Условие окончания цикла L2 не обязательно должно зависеть от шага L3, хотя обычно бывает именно так (см. все три примера выше).

Пример 4. Необходимо подсчитать число символов в строке. Предположим, что входной поток настроен на начало строки, тогда:

```
for (c=0; getchar()!='\n'; c++);  /* После завершения цикла c - количество символов
                                     */
```

Обратите внимание, что в данном случае условие окончания не зависит от шага (от приращения c). Вообще же изменение условия окончания в самом цикле надо делать крайне осторожно, чтобы не было зацикливания.

2.2.5 Оператор break

Оператор break имеет форму:

```
break;
```

Выполнение оператора. Оператор break прерывает выполнение того циклического оператора (while, do, for) или оператора switch, в котором он появляется. Когда оператор break встречается внутри одного из этих операторов (while, do, for, switch), то происходит немедленный выход из этого оператора и переход к выполнению оператора, следующего за прерванным.

Появление оператора break вне операторов do, for, switch, while приводит к ошибке. Оператор break нельзя использовать в конструкции if...else или в теле главной функции main для выхода из нее.

Внутри вложенных операторов break завершает только тот оператор while, do, for, swith, внутри которого break непосредственно записан. Чтобы передать управление вне вложенной структуры, могут быть использованы операторы return и goto.

Пример 1.

```
/* Печатаются вводимые символы, пока не будет введен символ 'Q'/ *
#include <stdio.h>
int main()
{ char  ch;
   for (;;) {
      ch=getchar();          /* Чтение символа */
      if (ch=='Q') break;    /* Проверка символа */
      printf(%c",ch);        /* Печать символа */
       }
```

```
}
```

Пример 2.

```
/* Для всех чисел от 0 и ≤ 1000 распечатать те, куб которых ≤ 10000 */
#include <stdio.h>
int main()
{
 int  i;
   for (i=0;i<1000;i++) {
      printf(%d - %d\n", i, i*i*i);        /* Печать: число - куб его */
      if (i*i*i >=10000) break;
      }
}
```

2.2.6 Оператор continue

Оператор continue имеет форму:

```
continue;
```

Выполнение оператора. Оператор continue служит для пропуска оставшейся части выполняемой итерации того цикла, в котором continue встретился. Если условиями цикла допускается новая итерация, то она выполняется; в противном случае цикл завершается. Таким образом, оператор continue передает управление на начало следующей итерации цикла: в циклах while и do - -на проверку условия, в цикле for - на приращение.

Пример 1. Действие оператора continue в приведенных ниже примерах эквивалентно переходу на метку next (предполагается, что continue не находится внутри вложенного цикла).

```
/* Цикл while */  while (...)  {
                          ...  continue;  ...
                           next:;}
/* Цикл do */     do  {
                        ...  continue;  ...
                        next:;} while (...);
/* Цикл for */    for (...)  {
                        ...  continue;  ...
                        next:;}
```

Пример 2.

```
/* Программа печатает натуральные числа, кратные семи */
#include <stdio.h>
main()
{
 int  i;
   for (i=0;i<1000;i++) {
      if (i%7) continue;
      printf("%8d", i);        /* Печать числа, кратного 7 */
      }
}
```

2.2.7 Оператор goto и метки операторов

Оператор goto имеет форму:

```
goto < метка >;
```

Выполнение оператора. Оператор goto предназначен для безусловной передачи управления к оператору с указанной меткой. Метка должна находиться в той же функции, что и оператор goto.

Метка в Си - это идентификатор, за которым следует двоеточие. Перед каждым оператором программы может быть поставлена метка в виде:

< метка > : < оператор >

Неограниченное использование оператора goto приводит к ухудшению понимания программы и поэтому не рекомендуется. Одно из возможных применений оператора goto - быстрый выход из вложенных циклов, например:

```
for (...) {
  while (...) {
    for (...) {
        . . .
        goto exit1;
        . . .
    }
  }
}
exit1: puts("Быстрый выход из вложенных циклов");
```

2.3 ЛАБОРАТОРНАЯ РАБОТА №2 "ОПЕРАТОРЫ УПРАВЛЕНИЯ"

Разветвляющийся вычислительный процесс позволяет выбрать в программе один из нескольких альтернативных путей решения задачи. Обычно реализуется в программе с помощью условных операторов разветвления if или switch или условной операции "?:", работа которых зависит от значения определенного условия.

Циклический вычислительный процесс предусматривает многократное выполнение некоторой последовательности действий. Для реализации циклических алгоритмов используются циклические операторы do, while, for.

Операторы разветвления и цикла позволяют управлять, в зависимости от некоторых условий, последовательностью действий в программе. В результате операторы программы могут выполняться не просто в линейной последовательности "сверху-вниз".

Цель лабораторной работы "Операторы управления" - научиться писать на языке Си разветвляющиеся и циклические программы, а также усвоить на примере практических задач работу операторов управления, рассмотренных в данном модуле 2.

Задание к лабораторной работе №2 включает в себя следующие задачи.

1. Разработать программу с использованием оператора if для вычисления y=f(x), вариант конкретной функциональной зависимости f(x) и интервалы изменения аргумента x взять из задания к лабораторной работе #1 "Линейный вычислительный процесс. Расчет по формулам". Точная формулировка условия дана в п.2.3.1.

2. Разработать программу с использованием оператора switch для вычисления y=f(x). Вид функциональной зависимости f(x) взять из задания к лабораторной работе #1 "Линейный вычислительный процесс. Расчет по формулам". Точная формулировка условия дана в п.2.3.2.

3. Разработать разветвляющиеся программы с использованием любых разветвляющихся операторов для решения определенных неформализованных задач,

список заданий приведен в п.2.3.3 (количество задач и их номера уточняет преподаватель).

4. Разработать циклические программы для решения определенных неформализованных задач, список заданий приведен в п.2.3.4 (количество задач и их номера уточняет преподаватель).

Замечание 1. При выполнении данной лабораторной работы нельзя использовать массивы и оператор безусловной передачи управления goto. При выполнении первых трех заданий (разветвляющиеся программы) нельзя использовать циклы, то есть логика решения данных задач должна быть только «разветвляющейся».

Замечание 2. При тестировании программ необходимо проверять все возможные варианты работы программ.

2.3.1 Точная формулировка условия для первой задачи

Разработать разветвляющуюся программу с использованием оператора if. Программа для заданного значения вещественного аргумента x вычисляет y=f(x) по формулам вида:

$$y=f(x)=\begin{cases} выражение\,1, & при\ x, лежащем\ в\ определенном\ интервале\ 1 \\ выражение\,2, & при\ x, лежащем\ в\ определенном\ интервале\ 2 \end{cases};$$

Программа должна работать следующим образом. Вводится значение аргумента x (один раз). Затем, если значение x лежит в одном из заданных интервалов, то по соответствующему выражению вычисляется значение y и результат выводится на экран; если значение x не лежит ни в одном из заданных интервалов, то на экран выводится сообщение вида: «Для заданного x=… нет решения».

Пусть, например, условия вычисления y=f(x) будут следующие:

$$y=f(x)=\begin{cases} x+1, & при\ \ 0<x\le 3 \\ x-1, & при\ \ 3<x\le 7 \end{cases}.$$

Ниже приведены примеры результатов тестирования работы программы, соответствующей данному заданию, для различных значений аргумента x.

Пример 1. Введите x, 0<x<=7: 1
Для x=1 y=2

Пример 2. Введите x, 0<x<=7: 5
Для x=5 y=4

Пример 3. Введите x, 0<x<=7: 9
Для x=9 нет решения.

2.3.2 Точная формулировка условия для второй задачи

Разработать разветвляющуюся программу с использованием оператора switch. Программа для заданного номера варианта расчета n и заданного значения вещественного аргумента x вычисляет y=f(x) по формулам вида:

$$y=f(x)=\begin{cases} выражение\,1, & при\ x, лежащем\ в\ определенном\ интервале\ 1, \quad если\ n=1 \\ выражение\,2, & при\ x, лежащем\ в\ определенном\ интервале\ 2, \quad если\ n=2 \end{cases}.$$

То есть необходимо разработать программу с использованием оператора switch для вычисления значения функции y=f(x) для заданного аргумента x, если вид

функциональной зависимости определяется номером варианта n, указанным пользователем. Кроме того, не для каждого значения аргумента x возможно вычислить y=f(x) по заданному выражению. Например, если в выражении используется операнд sqrt(x), то, значит, значение аргумента x не может быть отрицательным.

Программа должна работать следующим образом. Вводится номер варианта расчета n и значение аргумента x (один раз). Затем с помощью оператора switch программа, в зависимости от значения n, вычисляет значение y по первому или второму выражению:

1) значение y вычисляется по первому выражению, когда задан вариант расчета n=1,

2) значение y вычисляется по второму выражению, когда задан вариант расчета n=2.

Если n не равно 1 или 2, программа выдает соответствующее сообщение вида: «Для заданного номера варианта n нет решения». При вычислении выражений может потребоваться дополнительная проверка, лежит ли заданное значение x в области допустимых значений (ОДЗ); эту поверку рекомендуется провести оператором if. Эту область допустимых значений программист должен сам определить заранее, исходя из вида выражения, или взять указанные пределы изменения аргумента x из условия лабораторной работы #1 «Лирейный вычислительный процесс. Расчет по формулам». Если аргумент x не лежит в области допустимых значений, программа выдает соответствующее сообщение вида: «Для n=… и x=… решение не существует: x вне ОДЗ».

Пусть, например, условия вычисления y=f(x) будут следующие:

1) y=sqrt(x), если n=1,

2) y=x+1, если n=2.

Ниже приведены примеры результатов тестирования работы программы, соответствующей данному заданию, для различных значений аргумента x.

Пример 1. Введите n и x: 1 4
Для n=1 и x=4 y=2

Пример 2. Введите n и x: 1 -1
Для n=1 и x=-15 решение не существует: x вне ОДЗ

Пример 3. Введите n и x: 3 4
Для заданного номера варианта n нет решения.

2.3.3 Разветвляющийся вычислительный процесс. Неформализованные задачи

ЗАДАНИЕ. Написать разветвляющуюся программу для решения следующей задачи.

N1.

Доказать, что любую целочисленную денежную сумму, большую 7 рублей, можно выплатить без сдачи трешками и пятерками, то есть для данного целого n>7 найти такие целые неотрицательные a и b, что 3a+5b=n.

N2.

Дано натуральное число n ($n \le 100$), определяющее возраст человека в годах. Дать для этого числа наименования "год", "года", или "лет", например:

1 год
23 года
45 лет

N3.

Дано натуральное четырехзначное число n ($n \le 9999$). Является ли это число полиндромом (перевертышем), как, например, числа 2222, 6116, 0440 ?

N4.

Дано натуральное четырехзначное число n. Верно ли, что это число содержит ровно три одинаковых цифры, как, например, числа 6676, 4544, 0006 ?

N5.

Дано натуральное четырехзначное число n. Верно ли, что все четыре цифры числа различны, как, например, различны все 4 цифры следующих чисел: 0123, 9760, 5432.

N6.

Дано натуральное число n ($n \le 100$). Определить первую и последнюю цифру числа n.

N7.

Дано натуральное число n ($n \le 100$). Определить:

1) сколько цифр в числе n;
2) чему равна сумма его цифр.

N8.

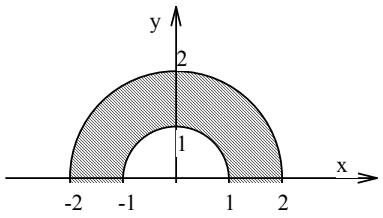
Дано натуральное число $10 \le n \le 100$. Определить, сколько цифр в числе, и найти предпоследнюю цифру числа n.

N9.

Дано натуральное число n ($n \le 99$). Выяснить, верно ли, что квадрат числа n равен кубу суммы цифр числа n. Например, для n=27 ответ «Да», так как n*n=729, (2+7)*(2+7)*(2+7)=9*9*9=729.

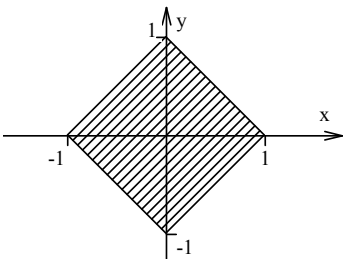
N10.

Даны вещественные числа x и y. Написать программу, определяющую, принадлежит ли точка с координатами (x,y) заштрихованной части плоскости.



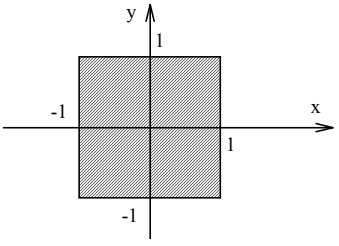
N11.

Даны вещественные числа x и y. Написать программу, определяющую, принадлежит ли точка с координатами (x,y) заштрихованной части плоскости.



N12.

Даны вещественные числа x и y. Написать программу, определяющую, принадлежит ли точка с координатами (x,y) заштрихованной части плоскости.



N13.

Написать программу, которая по заданному значению A определяет количество корней уравнения $x^2 = A + 1$ и выводит значения корней вместе с соответствующими сообщениями.

N14.

По данным вещественным числам A и B написать программу решения линейного уравнения A*x=B. Программа должна выводить одно из сообщений: "Уравнение не имеет решений", "Уравнение имеет бесконечное множество решений" или "Уравнение имеет единственное решение: x=<значение>".

N15.

Даны вещественные положительные числа a,b,c,d. Выяснить, можно ли прямоугольник со сторонами a,b уместить внутри прямоугольника со сторонами c,d так, чтобы каждая из сторон одного прямоугольника была параллельна или перпендикулярна каждой стороне второго прямоугольника.

N16.

Даны положительные a,b,c,x. Выяснить, пройдет ли кирпич с ребрами a,b,c в квадратное отверстие со стороной x. Просовывать кирпич в отверстие разрешается только так, чтобы каждое из его ребер было параллельно или перпендикулярно каждой из сторон отверстия. Ответ получить в текстовой форме: "можно" или "нельзя".

N17.

Даны вещественные числа a и R, представляющие собой длину стороны квадрата и радиус круга на плоскости. Написать программу, выводящую в зависимости от величин a и R одно из следующих сообщений:

1) "круг помещается в квадрат";
2) "квадрат помещается в круг";
3) "круг и квадрат не помещаются друг в друга".

N18.

Даны положительные x,y,z. Выяснить, существует ли треугольник с длинами сторон x, y, z. Ответ получить в текстовой форме: "существует" или "не существует". Если треугольник существует, то ответить, является ли он остроугольным.

N19.

Даны три вещественных положительных числа a, b, c. Написать программу, выводящую площадь треугольника с такими сторонами по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \quad p = \frac{a+b+c}{2}$$

или выводящую сообщение "треугольник не существует", если из отрезков длиной a, b, c треугольник построить нельзя.

N20.

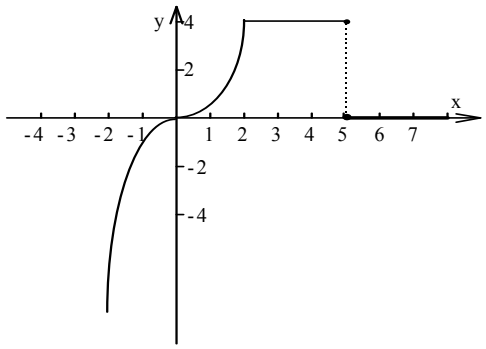
Даны четыре вещественных числа X , Y , X , Y , представляющих собой координаты точек A и B. Плоскость разделяется надвое прямой Y=X. Написать программу, определяющую, лежат ли точки A и B по одну сторону от прямой и выводящую соответствующее
сообщение.

N21.

Даны четыре вещественных числа X , Y , X , Y , являющиеся координатами точек A и B на плоскости. Написать программу, определяющую, какая из точек ближе к началу координат и выводящую соответствующее сообщение.

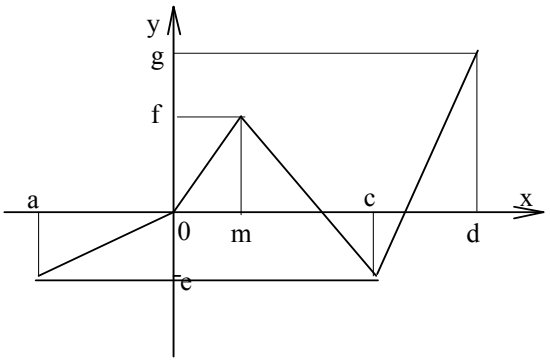
N22.

Написать программу для вычисления значения функции Y(x) для произвольного x. Функция Y(x) задана графически:



N23.

Написать программу для вычисления значения функции Y(x) для произвольного x. Функция Y(x) задана графически:



N24.

Даны вещественные числа x и y. Если x и y отрицательные, то каждое значение заменить на его модуль. Если отрицательно только одно из чисел x, y, то оба значения увеличить на 0.5. Если оба числа x, y неотрицательны, и ни одно из них не принадлежит отрезку [0.5; 2.0], то оба значения уменьшить в десять раз. В остальных случаях x, y не менять. Вывести исходные и полученные x, y

N25.

Даны вещественные числа a, b, c, d. Если $a \le b \le c \le d$, то каждое из чисел заменить на наибольшее из них. Если a> b> c> d, то числа оставить без изменения, в противном случае все числа заменить их квадратами. Вывести исходные и преобразованные числа a,b,c,d.

N26.

Даны числа x, y (x не равно y). Меньшее из этих двух чисел заменить их суммой, а большее - их удвоенным произведением. Вывести исходные и полученные значения x, y.

N27.

Даны четыре целых числа a, b, c, d, представляющие собой числители и знаменатели двух дробей $\frac{a}{b}, \frac{c}{d}$.

Написать программу, выводящую одно из трех сообщений:

1) "первая дробь больше второй",

2) "вторая дробь больше первой",

3) "дроби равны", -
в зависимости от отношения между дробями.

N28.

Даны три вещественных не равных между собой числа a, b, c. Переменной Y присвоить значение той переменной, которая находится между двумя другими на числовой оси. Например, если a=5, b=2, c=4, то ответ должен быть "Y=4".

N29.

Даны целые числа R, T. Если R не равно T, то заменить каждое из них одним и тем же числом, равным большему из исходных, а если R равно T, то заменить числа нулями. Вывести начальные и конечные значения чисел R и T.

N30.

Вычислить для заданных вещественных величин a, b, c:

$$Y = \frac{\max(a,b)\cdot\max(a,b,c)}{a+b+c\cdot\max(a^2,b^2)}.$$

N31.

Определить, верно ли, что при делении неотрицательного целого числа a на положительное целое число b получается остаток, равный одному из заданных чисел r или S.

2.3.4 Циклический вычислительный процесс. Неформализованные задачи

ЗАДАНИЕ. Написать циклическую программу для решения следующей задачи.

N 1.

Один молочник заполнил молоком N-литровый бидон и отправился к своим клиентам, живущим на K улицах. На каждой улице он продавал одинаковое количество литров молока L. Обслужив первую улицу, он шёл к колонке с водой и доливал бидон до краёв. Затем он обслуживал вторую улицу и снова шёл к колонке с водой. Так он действовал и дальше для всех клиентов.

Определить, сколько денег молочник должен вернуть жителям каждой улицы, если один литр он продавал за P рублей.

N 2.

В процессе лечебного голодания вес пациента за 30 дней снизился с 96 до 70 кг. Было установлено, что ежедневные потери веса пропорциональны весу тела. Вычислить, чему был равен вес пациента через k дней после начала голодания для k=1,2,3,...,29.

N 3.

Дано натуральное число n. Получить все способы выплаты суммы n с помощью монет достоинством 1, 3 и 5 копеек.

N 4.

Натуральное число из n цифр является числом Армстронга, если сумма его цифр, возведённых в n-ую степень, равна самому числу. Например: $153=1^3+5^3+3^3$.

Получить все числа Армстронга, состоящие из трёх цифр.

N 5.

Дано натуральное число n. Указать все тройки x,y,z таких натуральных чисел, что $n=x^2+y^2+z^2$.

N 6.

Пусть x_n и y_n - количество волков и лис в n-ом году. Установлено, что в результате их взаимного влияния их численность в следующем году определяется системой:

$$\begin{cases} x_{n+1} = 2 \cdot x_n \_ y_n \\ y_{n+1} = x_n + 2y_n \end{cases}.$$

Пусть x=a, y=b. Найти численность обоих видов за все годы, предшествующие полному вымиранию одного из них.

N 7.

Для любого целого k обозначим количество цифр в его десятичной записи Ц(k). Например: Ц(1)=1, Ц(9)=1, Ц(10)=2. Дано натуральное число n, действительное число x. Вычислить: $\frac{10^{Ц(1)}}{1} \cdot (1-x) + \frac{10^{Ц(2)}}{2} \cdot (1-x)^2 + \cdots + \frac{10^{Ц(n)}}{n} (1-x)^n$

N 8.

Для любого целого k обозначим количество цифр в его десятичной записи Ц(k). Например: Ц(1)=1, Ц(9)=1, Ц(10)=2. Дано натуральное число n. Вычислить:
$\frac{Ц(1)}{1^2} + \frac{Ц(2)}{2^2} + \cdots + \frac{Ц(n)}{n^2}$.

N 9.

Функция f(x)= $2.32 \cdot x - \frac{x^4}{4} + \frac{x^3}{3} - x^2$ имеет единственный максимум на интервале [0;1.5]. Написать алгоритм вычисления точки максимума с точностью h. Требуется вычислять последовательно значения заданной функции от начала интервала с шагом h, пока не будут выполняться условия, требуемые в задаче.

N 10.

Функция f(x)= $2.32 \cdot x - \frac{x^4}{4} + \frac{x^3}{3} - x^2 - 1.2$ имеет единственный максимум на интервале [0;1.5]. Известно, что f(0)<0, f(1.5)<0. Написать алгоритм, определяющий с точностью h интервал (a,b), вложенный в интервал [0;1.5], на котором функция положительна.

Требуется вычислять последовательно значения заданной функции от начала интервала с шагом h, пока не будут выполняться условия, требуемые в задаче.

N 11.

Функция f(x)= $x^2 + \sin^2 x$ монотонно возрастает на интервале [2;4]. Найти значение x с точностью h, для которого выполняется условие: f(x)=6.

Требуется вычислять последовательно значения заданной функции от начала интервала с шагом h, пока не будут выполняться условия, требуемые в задаче.

N 12.

Вычислить: $\cfrac{1}{1+\cfrac{1}{3+\cfrac{1}{5+\cfrac{1}{\ddots\cfrac{}{101+\cfrac{1}{103}}}}}}$

N 13.

Вычислить для заданного значения n:

$$\sqrt{3+\sqrt{6+\cdots+\sqrt{3\cdot(n-1)+\sqrt{3\cdot n}}}}$$

N 14.

Найти число помощников деда из известной сказки про репку, если:

- для вытягивания репки надо развить усилие в N кг;
- дед тянет с усилием D кг;
- каждый очередной i-ый помощник тянет с усилием S/i/(i+1),

где S - общее усилие всех предыдущих участников вытягивания репки.

N 15.

Написать программу, проверяющую, является ли данное целое число Z совершенным, и выводящую сообщение "ДА" или "НЕТ". Совершенным называется число, равное сумме всех своих делителей; например: 6=1+2+3.

N 16.

Написать программу, вычисляющую сумму всех целых чисел, меньших 1000, которые делятся на 3, или на 7, или на оба эти числа вместе.

N 17.

Написать программу, проверяющую, является ли данное число Z простым, и выводящую сообщение "ДА" или "НЕТ".

N 18.

Возьмём некоторое натуральное число n. Если это число чётно, то разделим его на 2, иначе умножим на 3 и прибавим 1. Если полученное число не равно 1, то будем повторять те же самые действия, пока не получим 1. Пока не известно, завершается ли этот процесс для любого n. Написать программу, проверяющую для данного n, завершается ли данный процесс не позднее, чем после m таких действий.

N 19.

Написать программу, выводящую все делители заданного натурального числа Z.

N 20.

Написать программу, выводящую наименьший делитель заданного натурального числа Z, отличный от 1.

N 21.

Написать программу, выводящую все целочисленные решения уравнения

a*x + b*y = c,

удовлетворяющие условиям |x|<=10, |y|<=20. Здесь a,b,c - заданные числа.

N 22.

Написать программу приближённого вычисления числа π по формуле

$\pi \cong 4\cdot\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\cdots+(-1)^{n+1}\cdot\frac{1}{2\cdot n-1}\right)$ для заданного значения n.

N 23.

Написать программу приближённого вычисления числа π по формуле

$$\pi \cong 4 \cdot \left(\frac{8}{9} \cdot \frac{24}{25} \cdot \frac{48}{49} \cdot \quad \cdots \quad \cdot \frac{(2n+1)^2 - 1}{(2n+1)^2} \right)$$ для заданного значения n.

N 24.

Вкладчик внёс в сберкассу x рублей. Написать программу, вычисляющую, сколько денег у вкладчика будет через n лет. Количество денег каждый год увеличивается на Z% по отношению к предыдущему году.

N 25.

В начальный момент времени имеется S кг дрожжей. Через каждый час количество дрожжей увеличивается на 15%, но на исходе часа M кг дрожжей удаляется. Написать программу, вычисляющую количество дрожжей через N часов.

N 26.

Написать программу, реализующую следующий алгоритм вычисления функции e для заданного x. Число x делить пополам до тех пор, пока не получится число Z, меньшее 0.01. Подсчитать число выполненных делений n. Вычислить: $Y=\frac{(z+1)^2+1}{2}$

Число Y возвести в квадрат n раз. Полученное число n будет приближённо равно e^x.

N 27.

Написать программу, выводящую сумму расстояний до начала координат всех точек, имеющих целочисленные координаты и находящиеся внутри прямоугольника: $1 \le x \le 10, \quad 1 \le y \le 5$.

N 28.

Написать программу вычисления остатка от деления числа n на число m (n,m - целые). Функции div , mod , trunc и round не использовать. Словами алгоритм формулируется следующим образом: из числа n вычитать число m, пока не получится число, меньшее m.

N 29.

Написать алгоритм вычисления n-ого члена последовательности Фиббоначи. Последовательность Фиббоначи определяется следующим образом:
$F_0 = 1, F_1 = 1, F_n = F_{n-1} + F_{n-2}$ *для* $n \ge 2$. При решении задачи массивы не использовать.

2.4 КОНТРОЛЬНЫЕ ВОПРОСЫ ПО МОДУЛЮ 2

2.4.1 Основные операции языка Си.
2.4.2 Построение условий в Си.
2.4.3 Условный оператор if - общий вид, реализуемый алгоритм, примеры.
2.4.4 Условная операция "?:" - общий вид, реализуемый алгоритм, примеры.
2.4.5 Оператор переключения switch - общий вид, реализуемый алгоритм, примеры.
2.4.6 Оператор цикла while - общий вид, реализуемый алгоритм, примеры.
2.4.7 Оператор цикла do - общий вид, реализуемый алгоритм, примеры.
2.4.8 Оператор цикла for - общий вид, реализуемый алгоритм, примеры.
2.4.9 Оператор break, continue, goto - общий вид, реализуемый алгоритм, примеры.

3 ПРОИЗВОДНЫЕ ТИПЫ (МАССИВЫ, СТРУКТУРЫ, ОБЪЕДИНЕНИЯ)

Цель изучения данного модуля освоить работу с производными типами в Си, а именно с массивами, структурами, объединениями, переменными структурами.

3.1 МАССИВЫ

Массив - это набор переменных (элементов) одинакового типа. Все они имеют одно имя, а различаются по номерам. Номера называются индексами элементов.

Определение массива имеет вид:

<тип> <имя>[n1][n2]...[nk];

где <тип> - это тип элементов массива (любой допустимый в языке тип);

<имя> - это имя массива;

n1,n2,...nk - размерности массива (константы или константные выражения); элементы i-того измерения имеют индексы от 0 до (ni-1).

Как видно из определения массива, в языке Си индексы всегда начинаются с нуля.

Пример 1. int a[20]; /* Массив из 20 целых элементов */

Элементы данного массива обозначаются следующим образом:

a[0] - 1-ый элемент,

a[1] - 2-ой элемент,

...

a[19] -20-ый элемент.

При обращении к элементам многомерного массива в квадратные скобки заключается каждый индекс.

Пример 2. float arr[3][2]; /* Описание двухмерного массива */

arr[1][1] - обращение к элементу arr[1][1];

arr[i][j] - обращение к элементу arr[i][j], где i,j - целые выражения.

Запись вида a[i,j] тождественна записи a[j], так как действие операции "запятая" интерпретируется следующим образом "определить i, затем определить j, затем присвоить всему выражению "i,j" значение "j".

Пример 3. float sl[m][n][l]; /* m,n,l - константы, которые должны быть предварительно описаны с помощью инструкции препроцессора define */

Многомерные массивы хранятся в памяти слева направо по правилу "строки-столбцы", то есть быстрее всего изменяется последний индекс.

Например, имеем следующее описание двухмерного массива:

int a[3][2];

Элементы этого массива хранятся в памяти в следующем порядке:

a[0][0]; a[0][1]; a[1][0]; a[1][1]; a[2][0]; a[2][1].

В языке Си под массив всегда выделяется непрерывное место в оперативной памяти, и нет проверки выхода индекса за пределы массива. Это означает, что если, например, для массива ,описанного как "int a[100]", в программе указать a[200], то сообщения об ошибке не будет, а в качестве значения элемента a[200] будет выдано некоторое число, занимающее соответствующие 2 байта от начала массива.

Имя массива без индексов в Си фактически является константой - адресом начала массива. Поэтому присвоить значение одного массива другому "целиком", как в Паскале, нельзя. Например, для двух массивом mas1 и mas2, описанных как

int mas1[10],mas2[10];

записать оператор присваивания " mas2=mas1 " нельзя.

Пример 4.

```
/* Некоторые типовые задачи по обработке одномерного массива  */
#include <stdio.h>
 int main()
 {
  int a[20],          /* Определение массива a  */
      n,              /* Фактическое число элементов массива */
      i,              /* Индекс массива */
      maxa,           /* Значение максимального элемента массива */
      k;              /* Количество положительных элементов массива */

/* Ввод одномерного целого массива */
  do {
       printf("Введите количество элементов массива (не более 20): ");
       scanf("%d",&n);
  }while (n<1||n>20);
  printf("Введите %d целых элементов массива: \n",n);
  for (i=0;i<n;i++)
       scanf("%d",&a[i]);

/* Определение значения наибольшего элемента массива */
  maxa=a[0];
  for (i=1;i<n;i++)
      if (maxa<a[i])
           maxa=a[i];
  printf("Значение максимального элемента массива: %d\n",maxa);

/* Определение количества положительных элементов массива */
  k=0;
  for (i=0;i<n;i++)
      if (a[i]>0)
              k++;
  printf("Количество положительных элементов массива: %d\n",k);

/* Вывод одномерного целого массива */
  for (i=0;i<n;i++)
         printf("%d ",a[i]);
}
```

Пример 5.

```
/* Ввод двумерных массивов  a  и  b  и их матричное умножение : c=a*b   */
#include <stdio.h>
 int main()
 {
  int a[3][4], b[4][2], c[3][2],          /* Описание массивов  */
      i,j,k;
  printf("Введите массив a(3,4) построчно: \n");    /* Ввод массива a */
    for (i=0; i<3; i++)
          for (j=0; j<4; j++)
             scanf("%d",&a[i][j]);
  printf("Введите массив b(4,2) построчно: \n");    /* Ввод массива b */
    for (i=0; i<4; i++)
          for (j=0; j<2; j++)
             scanf("%d",&b[i][j]);
```

```
    for (i=0; i<3; i++)                    /* Умножение: c=a*b  */
          {
          for (j=0; j<2; j++)
             {
              c[i][j]=0;
              for (k=0; k<4; k++)
                    c[i][j] += a[i][k]*b[k][j];
              }
          }

    for (i=0; i<3; i++)                    /* Вывод матрицы  c      */
          {
          for (j=0; j<2; j++)
             printf("c[%d][%d]=%d ",i,j,c[i][j]);
          printf("\n");
          }
}
```

Пример 6.

```
/* Различные способы заполнения двумерного символьного массива пробелами (или любым другим символом-знаком)  */
#define RM 7
#define CM 5
#include <stdio.h>
 int main()
 {
  char letter[RM][CM];          /* Описание массива  */
  int  r,c;
/* 1-ыи способ заполнения массива пробелами */
    for (r=0; r<RM; r++)
          for (c=0; c<CM; c++)
             letter[r][c]=' ';
    /* Вывод массива          */
    printf("\nЗполнение массива пробелами \n");
    for (r=0; r<RM; r++)
          {
          for (c=0; c<CM; c++)
             printf("c[%d][%d]=\'%c\' ",r,c,letter[r][c]);
          printf("\n");
          }
/* 2-ои способ заполнения массива знаком  '+'  */
    for (r=0; r<RM; r++)
          for  (c=0;  c<CM;  letter[r][c++]='+'); /* Использование ++c вместо c++ в данном случае не дало бы требуемого результата  */
    /* Вывод массива          */
    printf("\nЗаполнение массива знаком \'+\' \n");
    for (r=0; r<RM; r++)
          {
          for (c=0; c<CM; c++)
             printf("c[%d][%d]=\'%c\' ",r,c,letter[r][c]);
          printf("\n");
          }
 }
```

3.2 СТРУКТУРЫ

Структура ("запись" в терминах языка Паскаль) - это составной объект, в который входят компоненты любых типов, за исключением функций.

3.2.1 Определение структуры

Существует 3 основных способа (варианта) определения структур в программе. Возможно 3-ий способ модифицировать, тогда, по сути, появляется еще и 4-ый способ.

1 способ.

```
struct {
  <список описаний>
} <описатели>;
```

где <список описаний> - это описание компонентов (полей, элементов) структуры (должен быть указан хотя бы один компонент);
<описатели> - это обычно имена переменных, массивов, указателей и функций.

```
Пример 1. struct {          /* Переменные a и b определяются как структуры, каждая из */
                double x,y; /* которых состоит из двух компонентов x и y*/
              } a,b,c[9];   /* Переменная c определяется как массив из 9 таких структур */
Пример 2. struct {          /* Каждая из двух переменных-структур date1 и date2 */
                int year    /* состоит из 3 компонент     */
                short mont,day;
              } date1,date2;
```

2 способ.

Можно явно задать имя типа структуры с помощью ключевого слова typedef, а затем это имя использовать для определения переменных. Общий вид описания типа структуры:

```
typedef struct {
  <список описаний>
} <имя типа>
```

```
Пример 3. typedef struct {      /* Описан тип структуры */
                char name[30];  /* с именем empl        */
                int d;
              } empl;
              empl e1,e2;   /* Переменные e1,e2 - это структуры типа empl */
```

3 способ.

Обоснован на применении меток, или шаблонов, структуры (аналогично меткам перечисляемого типа).

Метка структуры описывается следующим образом:

```
struct <метка> {
  <список описаний>
};
```

где <метка> - это идентификатор.

Структуры определяются с помощью меток структур следующим образом :

```
  struct  <метка>  <список идентификаторов>;
```

```
Пример 4. struct  student  {    /* student - метка структуры */
                char name[23];
                 int id,age;
                 char pol;
              };
```

```
struct student s1,s2;  /* s1,s2 - это структуры, тип которых задан меткой student */
struct student studgrup[30];  /* studgrup - это массив структур шаблона student */
```

4 способ.

Задание шаблона структуры и объявление переменных можно производить одновременно:

```
struct  student  {      /* student - метка структуры */
  char name[23];
   int id,age;
   char pol;
} s1, s2, studgrup[30];
```

Использование меток структуры необходимо для описания рекурсивных структур.

3.2.2 Доступ к компонентам структуры

Доступ к компонентам структуры имеет вид:
<имя структуры-переменной>.<имя компоненты>

Например (здесь и далее см. пример 4 выше): s1.id, s2.pol.

Следует иметь в виду, что запись
studgrup[20].age.
дает доступ к полю age 21-го элемента массива studgrup, а запись

```
s1.name[5]
```

дает доступ к 6-му элементу поля name переменной s1.

Если объявлены две переменные типа структуры с одним и тем же именем типа или шаблона, то их можно присвоить одна другой; например:

```
s1=s2;
```

Переменные типа структуры нельзя сравнивать на "=" или " ≠" .

К структуре, как к любой переменной, можно применить операцию & для вычисления ее адреса.

3.2.3 Пример работы со структурой

```
/*  Дан список автомобилей, каждая строка которого содержит: марку автомобиля, год выпуска, цену. Распечатать список тех автомобилей, год выпуска которых не ранее некоторого заданного года, а цена не превышает некоторой заданной цены */
#include <stdio.h>
 int main()
 {
  struct{
   char marka[10]; /* Марка авто  */
   int year;           /* Год выпуска*/
   float money;     /* Цена */
         } avto[20];      /* Массив структур - список данных по автомобилям */
   int    n,        /* Количество элементов в массиве avto */
         i,        /* Индекс массива avto */
         y;        /* Контрольный год */
   float m;       /* Контрольная цена */
printf("\n Введите количество автомобилей n (n<20): ");
scanf("%d",&n);
printf("Введите список из %d автомобилей:\n",n);
```

```
for (i=0;i<n;i++)
        {fflush(stdin); gets(avto[i].marka);
         fflush(stdin); scanf("%d",&avto[i].year);
         fflush(stdin); scanf("%f",&avto[i].money);   }
printf("Введите контрольный год и цену: ");
scanf("%d%f",&y,&m);
for (i=0;i<n;i++)
        if (avto[i].year>=y&&avto[i].money<=m)
          printf(\n%12s  %4d  %7.3f", avto[i].marka, avto[i].year, avto[i].money);
}
```

3.3 ОБЪЕДИНЕНИЯ

Объединение подобно структуре, однако в каждый момент времени может использоваться (или являться активным) только один из компонентов.

Как и структуру, объединение можно определить 3 основными способами (см. п.3.2.1.).

Согласно 1-му способу можно записать:

```
union {
 <описание компонента 1>
 <описание компонента 2>
 ... ... ... ... ... ...
 <описание компонента n>
    } <описатели>;
```

Для каждого из этих компонентов 1,2,...n выделяется одна и та же область памяти, то есть они перекрываются. Причем места в памяти выделяется ровно столько, сколько надо тому элементу объединения, который имеет наибольший размер в байтах.

Объединения применяются для:

1)минимизации объема памяти, если в каждый момент времени только один объект из многих является активным;

2)интерпретации основного представления объекта одного типа, как если бы этому объекту был присвоен другой тип.

Доступ к компонентам объединения осуществляется тем же способом, что и к компонентам структур.

Пример.

```
    union {
       float radius;   /* Окружность */
       float a[2];     /* Прямоугольник */
       int b[3];       /* Треугольник */
       position p;     /* Точка.  position - тип, описанный пользователем */
         } geom_fig
```

Имеет смысл обрабатывать лишь активный компонент, который последним получил свое значение. Так, после присваивания значения компоненту radius не имеет смысла обращаться к массиву b[3].

3.4 ПЕРЕМЕННЫЕ СТРУКТУРЫ

Структуры, описанные в п.3.2. - это постоянные структуры с фиксированными компонентами. Они имеют в различных ситуациях строго определенную форму.

Иногда необходимо, чтобы некоторые компоненты структуры в различных ситуациях были бы различны. Для реализации такой возможности предназначены переменные структуры, которые представляют собой комбинацию структуры и объединения.

Переменная структура содержит набор общих компонентов плюс набор меняющихся компонентов.

В общем случае переменные структуры состоят из трех частей: набора общих компонентов, метки активного компонента и части с меняющимися компонентами.

Общая форма переменной структуры:

```
struct {
  <общие компоненты>;
  <метка активного компонента>;
  union {
    <описание компонента 1>;
    <описание компонента 2>;
     ... ... ... ... ... ...
    < описание компонента n>;
   } <идентификатор для union>;
 } <описатели>;
```

Текущее значение метки активного компонента (например, 1,2,...n) указывает, какой из переменных компонентов объединения является активным в данный момент. В принципе же, метка активного компонента не обязательна.

Пример. Рассмотрим представление фигур: окружности, прямоугольника, треугольника. Пусть каждая фигура характеризуется тремя параметрами: площадь, периметр, размер. Площадь и периметр - это общая информация для этих фигур; информация о размере различна в зависимости от формы фигуры: для круга это - размер радиуса, для прямоугольника - размер двух сторон, для треугольника - размер трех сторон. Рассмотрим соответствующую структуру fig и переменную f типа fig:

```
typedef struct {
  float area, perimetr;  /* Общие компоненты: площадь и периметр */
  int type;               /* Метка активного компонента */
  union {                 /* Переменный компонент */
        float r;              /* радиус окружности */
        float a[2];           /* две стороны прямоугольника */
        float b[3];          /* три стороны треугольника */
         } geomfig;    /* Имя объединения - переменного компонента */
} fig;
fig f;                      /* Определение переменной  f  типа  fig  */
```

Каждая переменная типа fig состоит из трех постоянных компонентов: area, perimetr, type. Компонент type (метка активного компонента) используется для указания, какой из компонентов объединения geomfig является активным в данный момент. Например:

type=1 - считаем, что активна первая компонента r

type=2 - считаем, что активна вторая компонента a[2];

type=3 - считаем, что активна третья компонента b[3].

По соглашению, обычно перед присваиванием значения одному из компонентов структуры соответствующее значение присваивается также переменной f.type для указания активного компонента.

Например: f.type=1; f.geomfig.r=5.0;

Аналогично перед обращением к компоненту объединения необходимо проверить, является ли этот компонент активным. Например:

```
switch (f.type) {
  case 1: <обработка окружности>; break;
  case 2: <обработка прямоугольника>; break;
  case 3: <обработка треугольника>; break;
  default:<ошибка>;
 }
```

Метку активного компонента type можно для наглядности описать типа fclass, где fclass - это перечисляемый тип:

typedef enum {circle, rect, triangle} fclass;

В этом случае для работы с треугольником необходимо написать:

f.type=triangle;

Использование перечисляемого типа позволит компилятору предупредить программиста о потенциально ошибочных присваиваниях вида:

f.type=44;

3.5 ИНИЦИАЛИЗАЦИЯ

Инициализация - это присваивание начальных значений при объявлении переменных. Для практического программирования инициализация бывает очень удобна. Подробно эта тема обсуждается в п. 7.5, здесь она рассматривается кратко.

Инициализировать можно простые переменные и сложные, такие как массивы и структуры.

Пример 1. Инициализация простых переменных:

```
int  i=0, j=1;
```

Пример 2. Инициализация одномерного массива:

```
int days[7]={1,2,3,4,5,6,7};
```

Пример 3. Инициализация двумерного массива:

```
int a[2][3]={1,2,3,4,5,6};
```

Эта инициализация равноценна следующему набору операторов присваивания:

```
a[0][0]=1; a[0][1]=2; a[0][2]=3;   a[1][0]=4; a[1][1]=5; a[1][2]=6;
```

Пример 4. Для двумерных, и, вообще, многомерных массивов часто используется вложенная инициализация. Так, инициализацию массива, приведенную в примере 3, можно записать следующим образом:

```
int a[2][3]={ {1,2,3}, {4,5,6}};
```

Пример 5. Символьный массив можно инициализировать как обычный массив:

```
char str[15]={'T','u','r','b','o',' ','C'};
```

а можно - как строку символов: `char str[15]={"Turbo C"};`

Следует иметь в виду, что результат в двух этих случаях будет разный ! Во втором случае в конец строки автоматически добавляется нулевой знак.

Как видно из примера 5, количество инициализаторов (начальных значений) не обязано совпадать с количеством элементов массива. Если инициализаторов меньше, то оставшиеся значения элементов массива не определены.

3.6 ЛАБОРАТОРНАЯ РАБОТА №3 "МАССИВЫ И СТРУКТУРЫ"

Массив - это упорядоченная по индексам совокупность элементов одного типа. Массив - одно из основных понятий программирования.

Структуры позволяют хранить и обрабатывать совокупности данных различного типа. Различают постоянные структуры с фиксированными компонентами и переменные структуры, в которых есть компоненты, различные по типу в зависимости от конкретной ситуации. С точки зрения языка, переменная структура - это структура, одним из компонентов которой является объединение.

Цель лабораторной работы "Массивы и структуры" - научиться писать на языке Си программы, предполагающие обработку массивов, причем не только массивов из простых элементов, но и массивов структур (постоянных и переменных).

Задание к лабораторной работе включает в себя три следующие задачи.

1. Написать программу по обработке двумерного целочисленного массива.

Вариант задания взять из списка заданий в п.3.6.1. Программа должна быть организована таким образом, чтобы при тестировании распечатывались результаты всех этапов

формирования и обработки массива. Полученный "протокол" работы программы позволит легко проанализировать ее правильность.

2. Написать программу по обработке массива структур. Вариант конкретного задания взять из списка заданий в п.3.6.2. В программе должны быть предусмотрены три отдельных последовательных блока: ввод исходного массива структур, обработка массива структур, вывод всех полученных результатов.

3. Написать программу по обработке массива переменных структур. Вариант конкретного задания взять из списка заданий в п.3.6.3. В программе должны быть предусмотрены три отдельных последовательных блока: ввод исходного массива переменных структур, обработка массива переменных структур, вывод всех полученных результатов.

При выполнении второго и третьего задания можно использовать стандартные функции обработки строк, которые подробно описаны в п.5.2.7 ниже.

3.6.1 Массивы

ЗАДАНИЕ. Написать программу для обработки двумерного целочисленного массива.

№1.

Дана целая матрица, в которой имеется несколько построенных по спирали участков (спираль - по часовой стрелке, движение - по горизонтали-вертикали). Центром каждого такого участка является 1, а каждый следующий элемент на 1 больше предыдущего. Найти число таких участков, а также число составляющих их элементов.

Пример матрицы с одним циклическим участком длиной 12:

*	*	*	*	*	*	*
*	*	7	8	9	10	*
*	*	6	1	2	11	*
*	*	5	4	3	12	*
*	*	*	*	*	*	*

(* - любое число, не принадлежащее спирали)

№2.

Дана целая матрица. Найти последовательность целых чисел, получающихся при чтении данной матрицы по спирали (спираль - по часовой стрелке, порядок развертывания спирали ясен из примера ниже). Начало спирали (координаты) и ее длину задает пользователь.

Пример: координаты начала спирали (2, 2); длина 7; матрица имеет вид

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

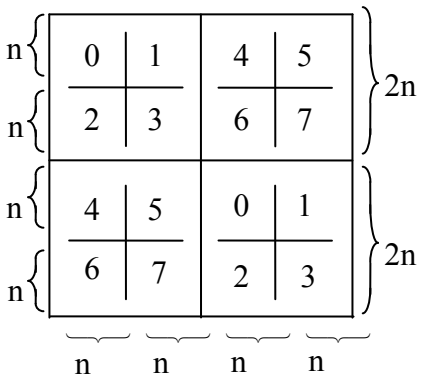
Результат: элементы 7, 2, 3, 8, 13, 12, 11

Если для заданных условий получить спираль требуемой длины не удается, то найти спираль из указанного центра максимально возможной длины. Например, если для приведенной выше матрицы указано начало спирали (2, 2) и длина 15, то результатом будет последовательность из 9 элементов вида:

7, 2, 3, 8, 13, 12, 11, 6, 1.

№3.

Построить квадратную матрицу порядка (4 * n), то есть порядок кратен 4, следующего общего вида (матрица условно разбита на 16 квадрантов, и все элементы определенного квадранта должны быть равны указанной ниже цифре):



Примеры для матриц размером (4×4) и (8×8) :

$$\begin{pmatrix} 0 & 1 & 4 & 5 \\ 2 & 3 & 6 & 7 \\ 4 & 5 & 0 & 1 \\ 6 & 7 & 2 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 4 & 4 & 5 & 5 \\ 0 & 0 & 1 & 1 & 4 & 4 & 5 & 5 \\ 2 & 2 & 3 & 3 & 6 & 6 & 7 & 7 \\ 2 & 2 & 3 & 3 & 6 & 6 & 7 & 7 \\ 4 & 4 & 5 & 5 & 0 & 0 & 1 & 1 \\ 4 & 4 & 5 & 5 & 0 & 0 & 1 & 1 \\ 6 & 6 & 7 & 7 & 2 & 2 & 3 & 3 \\ 6 & 6 & 7 & 7 & 2 & 2 & 3 & 3 \end{pmatrix}$$

№4.

Предположим, что целая матрица - это пруд с водой. В любое место пруда (в любую позицию матрицы) бросают камушек. Место попадания обозначим цифрой 0. От камушка (от 0) по воде расходятся круги в виде цифр 1, 2,3, ...

Написать программу для формирования поверхности пруда после попадания камушка в любую точку пруда.

Пример: матрица (5×5), камушек попал в позицию (3, 2). В результате поверхность пруда должна быть представлена следующим образом:

2 2 2 2 3
1 1 1 2 3
1 0 1 2 3
1 1 1 2 3
2 2 2 2 3

№5.

Дана целая матрица, не обязательно квадратная, количество строк которой равно 2×n (n - любое натуральное число).

Переставить в ней строки следующим образом (ниже даны номера строк исходной матрицы):

1; 2×n; 2; 2×n-1; 3; . . . n; n+1.

Пример: в матрице 6 строк, их номера: 1, 2, 3, 4, 5, 6.

После перестановки строки должны расположиться следующим образом:

1, 6, 2, 5, 3, 4.

№6.

Дана целая матрица, не обязательно квадратная, количество столбцов которой равно 2×n (n - любое натуральное число).

Переставить в ней столбцы следующим образом (ниже даны номера столбцов исходной матрицы):

1; n+1; 2; n+2; 3; n+3; . . . n; 2×n.

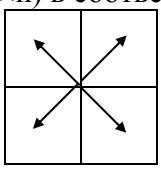
Пример: в матрице 6 столбцов, их номера: 1, 2, 3, 4, 5, 6.

После перестановки столбцы должны расположиться следующим образом:

1, 4, 2, 5, 3, 6.

№7.

Дана целая квадратная матрица порядка 2×n, то есть ее порядок кратен 2.
Переставить ее блоки размера (n×n) в соответствии со следующим рисунком:

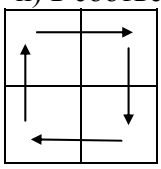


Пример для матрицы (2×2):

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \rightarrow \begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix}$$

№8.

Дана целая квадратная матрица порядка 2×n, то есть ее порядок кратен 2.
Переставить ее блоки размера (n×n) в соответствии со следующим рисунком:



Пример для матрицы (2×2):

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \rightarrow \begin{pmatrix} 3 & 1 \\ 4 & 2 \end{pmatrix}$$

№9.

Заполнить матрицу (n×n) целыми числами 1, 2, . . . n×n в следующем порядке:

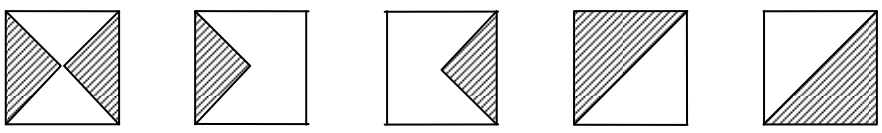
1	2	6	7	.
3	5	8	.	.
4	9	.	.	.
10	.		.	.
.	.	.	.	.

Пример заполнения матрицы (4×4):

1	2	6	7
3	5	8	13
4	9	12	14
10	11	15	16

№10.

Дана целая квадратная матрица порядка n. Найти наименьшее из значений элементов, расположенных в заштрихованной части матрицы:

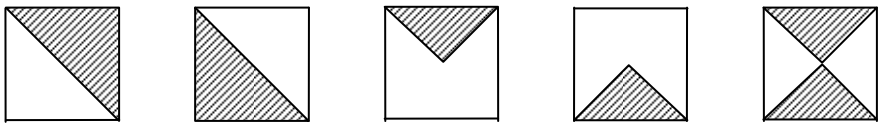


а *б* *в* *г* *д*

Выбор варианта (а, б, в, г, д) осуществляет пользователь. Предварительно программа должна представить эти картинки пользователю для выбора.

№11.

Дана целая квадратная матрица порядка n. Найти наибольшее из значений элементов, расположенных в заштрихованной части матрицы:



а *б* *в* *г* *д*

Выбор варианта (а, б, в, г, д) осуществляет пользователь. Предварительно программа должна представить эти картинки пользователю для выбора.

№12.

Назовем допустимым преобразованием матрицы перестановку двух строк или двух столбцов. Дана целая квадратная матрица порядка n, все элементы которой различны. С помощью допустимых преобразований добиться того, чтобы:

а) один из элементов матрицы, обладающий наибольшим значением, располагался в левом верхнем углу матрицы;

а) один из элементов матрицы, обладающий наименьшим значением, располагался в левом нижнем углу матрицы.

Пользователь должен указать программе, что делать: а) или б).

Тест должен быть организован так, чтобы можно было проследить последовательность преобразований.

№13.

Дана матрица. Напечатать те ее элементы, все соседи которых по горизонтали, вертикали и диагонали:

а) больше этого элемента,

б) меньше этого элемента.

Пользователь должен указать программе, какой вариант (а или б) следует реализовать.

№14.

Дана целая матрица (m×n), m, n ≤20.

В каждой строке матрицы переставить элементы следующим образом: число элементов в строке разделить пополам и элементы из “второй” половины переставить вперед (см. пример).

Например:

было: (1, 2, 3, 4, 5, 6) → стало: (1, 4, 2, 5, 3, 6);

было: (1, 2, 3, 4, 5, 6, 7) → стало: (1, 5, 2, 6, 3, 7, 4).

При перестановке элементов в строке вспомогательный массив не использовать.

№15.

Дана целая матрица (m×n), m, n ≤20.

В строках с четными номерами переставить все элементы, по модулю не превышающие 10, в начало строки, а в строках с нечетными номерами переставить такие элементы в конец сроки. Для перестановки элементов в строке вспомогательные массивы не использовать.

Например:

первая строка: (1, 100, 2, 4, 50, 15, 1) → (100, 50, 15, 1, 2, 4, 1);

вторая строка: (1, 100, 2, 4, 50, 15, 1) → (1, 2, 4, 1, 100, 50, 15).

Порядок следования элементов внутри строки не менять.

3.6.2 Массив структур

ЗАДАНИЕ. Написать программу для ввода и обработки массива структур.

№1.

Имеется таблица T1, каждая строка которой содержит следующие сведения: вид (номер) изделия; количество изделий этого вида; технические характеристики изделия (не более 3), каждая характеристика - это целое положительное число.

Имеется таблица T2, содержащая сведения о том, на сколько уменьшилось или увеличилось количество изделий по некоторым видам. Таблица T2 может содержать несколько сообщений по изделию одного вида или не содержать ни одного такого сообщения.

Обновить таблицу T1 на основе таблицы T2 и упорядочить полученную таблицу T1 по убыванию номеров изделия.

№2.

Имеется таблица, каждая строка которой содержит следующие сведения: название изделия; количество изделий этого вида; технические характеристики изделия (не более 4), каждая характеристика - это целое число.

Упорядочить таблицу по названиям изделий и определить те изделия, количество которых меньше некоторой заданной величины.

№3.

Имеется таблица, каждая строка которой содержит следующие сведения: номер изделия; количество изделий этого номера; технические характеристики изделия (не более 5), каждая характеристика - это целое положительное число.

Упорядочить таблицу по убыванию количества изделий и определить те изделия, вторая характеристика которых лежит в заданном диапазоне.

№4.

Имеется таблица, каждая строка которой содержит данные за год об одном вкладчике банка: номер вклада (6 цифр), ФИО; размер первоначального вклада; для каждого из 12 месяцев - сумму поступления.

Получить массив, куда переписать данные тех вкладчиков, которые в указанный месяц внесли сумму, большую некоторой заданной величины, и упорядочить массив по возрастанию сумм вкладов в этом месяце.

№5.

Имеется таблица “Экспортируемые товары предприятия”. Каждая строка таблицы содержит: инвентарный номер товара; количество стран, покупающих товар; объем продаваемой партии (в штуках) для каждой страны.

Упорядочить таблицу по возрастанию инвентарных номеров и найти номера тех товаров, общий объем продажи которых превышает некоторую заданную величину.

№6.

Имеется список товаров, продаваемых в магазине. Каждая строка этого списка содержит: название товара; количество видов этого товара; цену каждого вида.

Упорядочить список по названиям товаров и определить, есть ли в магазине товар заданного наименования, цена на который - в пределах заданного диапазона.

№7.

Имеется список товаров, хранящихся на базе. Каждая строка этого списка содержит: инвентарный номер товара; количество видов этого товара; срок хранения (до какого года) для каждого вида.

Упорядочить список по инвентарным номерам и найти те товары (номер товара и номер вида), срок хранения которых истекает в заданном году.

№8.

Имеется таблица сведений о продаваемом товаре. Каждая строка таблицы содержит: инвентарный номер товара; количество предприятий, покупающих товар; объем продаваемых партий товара (в штуках) каждому предприятию.

Упорядочить таблицу по возрастанию объема самой большой партии одного вида товара.

№9.

Имеется таблица “Автомобили”. Каждая строка содержит следующие сведения: марка, номер, фамилия владельца, технические характеристики (не более 3), каждая характеристика - целое число.

Упорядочить таблицу по фамилиям владельцев и найти фамилию владельца и номера автомобилей заданной марки.

№10.

Имеется таблица сведений об автомобилях. Каждая строка содержит следующие сведения: номер, фамилия владельца, марка, технические характеристики (не более 4), каждая характеристика - целое положительное число.

Упорядочить таблицу по номерам автомобилей и найти количество автомобилей каждой марки.

№11.

Имеется таблица “Успеваемость студентов за год”. Одна строка таблицы соответствует одному студенту и содержит следующие сведения: ФИО студента, номер группы, экзаменационные оценки за первый семестр, экзаменационные оценки за второй семестр.

Упорядочить строки таблицы по номерам групп и найти студентов, у которых средний балл за второй семестр ниже, чем за первый.

№12.

Имеется таблица “Успеваемость студентов за семестр”. Одна строка таблицы соответствует одному студенту и содержит следующие сведения: ФИО студента, номер группы, экзаменационные оценки (не более 5).

Упорядочить строки таблицы по фамилиям студентов и найти всех студентов, которые учатся только на “хорошо” и "отлично".

№13.

Дано расписание самолетов, каждая строка расписания содержит: пункт назначения; номера рейсов до данного пункта; время вылета, соответствующее каждому номеру рейса.

Упорядочить расписание по пунктам назначения (они должны идти по алфавиту) и определить все рейсы до двух заданных пунктов, время вылета для которых расположено в заданном интервале.

№14.

Имеется таблица “Багаж пассажира”. Одна строка таблицы соответствует одному пассажиру и содержит следующие сведения: номер рейса, фамилию пассажира, количество вещей (не более 5), вес вещей (каждой отдельно).

Упорядочить таблицу по номерам рейсов и найти тех пассажиров, общий вес багажа которых превышает некоторую заданную величину.

№15.

Имеется таблица, содержащая сведения о багаже пассажиров заданного рейса. Каждая строка содержит: номер кресла (кресло может пустовать), общее количество вещей пассажира, занимающего это кресло (не более 4-х вещей), вес каждой вещи.

Упорядочить таблицу по убыванию веса самой тяжелой вещи одного пассажира.

№16.

Имеется таблица сведений о багаже пассажиров заданного рейса. Каждая строка содержит сведения о багаже одного пассажира: количество вещей (не более 7), вес вещей (каждой отдельно).

Упорядочить таблицу по убыванию общего веса багажа одного пассажира.

№17.

Имеется таблица, содержащая сведения о возрасте жильцов одного дома. Каждая строка таблицы содержит: номер квартиры; количество жильцов в квартире; возраст каждого жильца (количество жильцов - не более 10-ти человек).

Упорядочить таблицу по возрастанию возраста старейшего жильца каждой квартиры.

№18.

Имеется таблица, содержащая сведения о жильцах одного дома. Каждая строка содержит следующие сведения: номер квартиры; фамилию квартиросъемщика; общее количество жильцов в квартире (не более 10-ти человек); инициалы (из 3-х букв) каждого жильца.

Упорядочить таблицу по номерам квартир и найти квартиры, в которых проживает более 5-ти человек.

№19.

Имеется таблица сведений о детях сотрудников одного отдела. Каждая строка содержит: номер сотрудника, количество детей у него, возраст каждого ребенка (полное число лет).

Упорядочить таблицу по убыванию количества несовершеннолетних детей у каждого сотрудника.

№20.

Имеется таблица, содержащая сведения о детях сотрудников предприятия. Каждая строка содержит: ФИО сотрудника, количество детей у него, возраст каждого ребенка (полное число лет).

Упорядочить таблицу по фамилиям сотрудников и найти тех сотрудников, которые имеют 3-х и более несовершеннолетних детей.

№21.

Имеется таблица “Шарики”. Каждая строка содержит сведения об одном шарике: инвентарный номер шарика, его цвет (7 цветов радуги) и материал (металл, пластмасса, дерево).

Упорядочить таблицу по убыванию инвентарных номеров и найти количество шариков для каждого вида материалов.

№22.

Имеется таблица “Кубики”. Каждая строка содержит сведения об одном кубике: размерность кубика (длина ребра в см.), его цвет (7 цветов радуги) и материал (металлический, деревянный, картонный).

Упорядочить таблицу по возрастанию размерности кубика и найти количество кубиков каждого из перечисленных цветов.

№23.

Имеется список членов семьи. Каждая строка списка содержит: имя; день, месяц и год рождения.

Упорядочить список по датам (день и месяц без учета года) и найти самого старшего члена семьи.

№24.

Дана таблица, каждая строка которой содержит данные об определенном рейсе междугороднего автобуса: номер рейса; пункт назначения; массив из 10 элементов, каждый элемент которого содержит дату отправления (то есть собираются сведения о 10 ближайших датах отправления этого рейса) и признаки для 30 мест автобуса, продан билет на это место или нет.

Указана определенная дата.

Получить массив, элемент которого содержит: номер рейса, отправляющегося в эту дату; количество свободных мест для этого рейса; массив данных о 30 местах. Полученный массив упорядочить по убыванию количества свободных мест.

№25.

Таблица содержит данные о ВУЗах. В каждой строке информация об одном ВУЗе: код (5 символов); количество факультетов (не более 10); для каждого факультета указывается общий набор на первый курс и количество специальностей. Получить таблицу ВУЗов, в которых не более 3 факультетов; строка этой таблицы содержит: код ВУЗа; количество факультетов; общий объем приема на первый курс ВУЗа. Упорядочить таблицу по убыванию общего объема приема.

№26.

Дана таблица, каждый элемент которой содержит данные об одной машине: номерной знак; ФИО. владельца; марку; количество нарушений (не больше 5); о каждом нарушении - номер нарушенного правила и дату нарушения.

Указан номер определенного правила.

Получить массив, элемент которого содержит: номер машины; общее количество нарушений для этой машины; количество нарушений указанного правила. Полученный массив упорядочить по количеству нарушений указанного правила.

3.6.3 Массив переменных структур

ЗАДАНИЕ. Написать программу для ввода и обработки массива переменных структур.

№1.

Имеется набор геометрических фигур: круг, прямоугольник, треугольник. Для каждой фигуры дан ее размер (для круга - радиус; для прямоугольника - длины двух сторон, для треугольника - длины трех сторон) и цвет (7 цветов радуги). Если площадь фигуры меньше некоторой заданной величины ξ, то эта фигура в списке фигур должна быть заменена на некоторую стандартную фигуру с тем же цветом (вместо размера должно быть указано слово “стандарт”).

№2.

Имеется набор простейших геометрических фигур: круг, прямоугольник, треугольник. Для каждой фигуры задан размер (для круга - радиус; для прямоугольника - длины двух сторон, для треугольника - длины трех сторон) и цвет (допустимы 7 цветов радуги).

Найти площадь каждой фигуры и упорядочить список фигур по возрастанию площадей.

№3.

Имеется список изделий, принадлежащих двум группам А и Б. Для каждого изделия указано:

1) инвентарный номер изделия;

2) группа (А и Б);

3) для изделий группы А - материал (дерево, пластмасса) и вес, для изделий группы Б - три размера (высота, ширина, глубина).

Составить список всех изделий группы А, изготовленных из дерева и имеющих вес не больше некоторого заданного.

№4.

Имеется список изделий трех видов. Для каждого изделия указано:

1) номер изделия;

2) вид;

3) для первого вида - цвет (допустимо 7 цветов радуги);
для второго вида - вес;
для третьего вида - три размера (высота, ширина, глубина).

Составить отдельный список изделий второго вида, упорядоченный по весу.

№5.

Есть список художественных произведений, опубликованных или в виде книги, или в журнале. Каждая строка списка содержит следующие сведения:

1) автор;

2) название произведения;

3) признак вида публикации (книга или журнал);

4) для книги - год издания, для журнала - название, год издания и номер.

Получить список всех журнальных произведений за определенный год. Список должен быть упорядочен по названию произведений.

№6.

Имеется список произведений, опубликованных в виде отдельной книги или в журналах. Каждая строка списка содержит следующие сведения:

1) автор;

2) название произведения;

3) признак вида публикации (книга или журнал);

4) для книги - город, издательство, год издания;
для журнала - название журнала, год издания и номер.

Составить отдельный список всех журнальных произведений. Список должен быть упорядочен по фамилиям авторов.

№7.

Список имеющегося транспорта некоторого автопредприятия имеет вид:

1) номер машины и марка;

2) для грузовой - тоннаж, для пассажирского транспорта - количество посадочных мест, для легкового - наименование арендующего предприятия.

Составить список легковых машин заданной марки. Список упорядочить по названию арендующих предприятий.

№8.

Дана анкета, каждая строка которой содержит следующую информацию:

1) ФИО;
2) возраст (полное число лет);
3) если возраст меньше 16, то ФИО отца и ФИО матери;
   если возраст больше либо равен 16, то номер паспорта.

Составить список граждан не моложе 16 лет, упорядоченный по номеру паспорта.

№9.

Имеется телефонный справочник. Если телефон личный, то строка справочника содержит: 1) номер телефона, 2) фамилию владельца, 3) адрес владельца; если телефон служебный - 1) номер телефона, 2) наименование учреждения, 3) номер отдела.

Определить все номера телефонов, владельцы которых имеют заданную фамилию, и упорядочить их по адресам.

№10.

Имеется телефонный справочник. Если телефон личный, то строка справочника содержит: 1) номер телефона, 2) ФИО владельца; если телефон служебный - 1) номер телефона, 2) наименование учреждения, 3) наименование подразделения.

Определить все телефоны заданного учреждения и упорядочить их по номерам.

№11.

Дана анкета, каждая строка которой содержит информацию:

1) фамилия;
2) пол;
3) если пол женский, то год рождения; если пол мужской, то указать семейное положение (холост, женат) и количество детей.

Составить упорядоченный по фамилиям список всех холостых мужчин, не имеющих детей.

№12.

Дана анкета, каждая строка которой содержит информацию:

1) ФИО;
2) пол;
3) если пол женский, то количество детей; если пол мужской, то признак, военнообязанный или нет, и домашний адрес.

Составить список всех женщин, упорядоченный по убыванию количества детей.

№13.

Имеется список, каждая строка которого содержит следующую информацию:

1) вид жилья: государственная квартира или частный дом;
2) общая жилая площадь;
3) для государственной квартиры - год сдачи ее в эксплуатацию, для частного дома - фамилия владельца.

Составить упорядоченный по фамилиям список владельцев частных домов, общая жилая площадь которых не меньше заданной.

№14.

Имеется список, каждая строка которого содержит следующую информацию:

1) вид жилья: государственная квартира, ведомственная квартира или личный дом;
2) общая жилая площадь;

3) для государственной квартиры - ФИО квартиросъемщика, для личного дома - ФИО владельца, для ведомственной квартиры - название учреждения и номер его банковского счета.

Составить упорядоченный по фамилиям список граждан, проживающих в государственных квартирах.

№15.

Имеется список сотрудников предприятия в виде:

1) фамилия;
2) образование (среднее или высшее);
3) если образование высшее - год окончания и название высшего учебного заведения; если среднее - возраст сотрудника.

Составить упорядоченный по возрасту список сотрудников, не имеющих высшего образования.

№16.

Имеется список, каждая строка которого содержит следующую информацию:

1) ФИО;
2) место работы;
3) для работающих - номер банковского счета предприятия, для неработающих - домашний адрес.

Выделить отдельно список работающих граждан, упорядоченный по номеру банковского счета предприятий.

№17.

Имеется анкета, каждая строка которой содержит следующие сведения:

1) ФИО;
2) год рождения;
3) пол;
4) семейное положение:
    а) женат (замужем);
    б) разведен (а);
    в) в браке не состоял(а) ни разу;
5) если пол мужской, то признак, военнообязанный или нет; если пол женский, то домашний адрес.

Составить упорядоченный по году рождения список женщин, которые состоят в браке.

№18.

Имеется анкета, каждая строка которой содержит следующую информацию:

1) ФИО;
2) пол;
3) семейное положение: состоит в браке или нет;
4) знание иностранных языков;
5) если есть знание иностранных языков, то сколько языков и какие; если нет знания иностранных языков, то год рождения.

Составить упорядоченный по фамилиям список холостых мужчин без знания иностранного языка.

№19.

Список сведений о сотрудниках предприятия имеет вид:

1) фамилия;
2) пол;
3) количество дней за последний год, проведенных на "больничном";

4) если количество "больничных" дней больше 21, то номер и адрес поликлиники, обслуживающей предприятие; иначе указано семейное положение: состоит в браке или нет.

Составить упорядоченный по фамилиям список холостых сотрудников-мужчин, которые провели за последний год на "больничном" не более 21 дня.

№20.

Имеется список сведений о сотрудниках института в виде:

1) ФИО;

2) женат (замужем), разведен и ли в браке не состоял ни разу.

3) если женат (замужем), то ФИО жены (мужа);

если разведен - год развода,

если в браке вообще не состоял - никакой информации.

Составить отдельный упорядоченный по фамилиям список семейных сотрудников.

№21.

Имеется анкета, каждая строка которой содержит следующую информацию:

1) ФИО;

2) возраст (полное число лет);

3) если возраст меньше 18, то ФИО отца и ФИО матери;

если возраст больше либо равен 18, то количество детей.

Определить всех граждан не моложе 18 лет и упорядочить их в порядке убывания количества детей.

3.7 КОНТРОЛЬНЫЕ ВОПРОСЫ ПО МОДУЛЮ 3

3.7.1 Определение массивов в Си. Обработка одномерных и двумерных массивов.

3.7.2 Понятие структуры. Три способа определения структуры в Си.

3.7.3 Понятие объединения. Три способа определения объединения в Си.

3.7.4 Структуры переменные и постоянные.

3.7.5 Инициализация простых переменных и массивов.

4 УКАЗАТЕЛИ

Цель изучения данного модуля: познакомиться с понятием указателя и изучить одно из основных применений указателей - динамическое распределение памяти.

4.1 ПОНЯТИЕ УКАЗАТЕЛЯ

Описание переменной означает, что ей должна быть выделена память. При этом в памяти машины отводится ячейка для хранения значения этой переменной. Размер ячейки (количество байт) зависит от типа переменной.

Каждая переменная имеет значение и адрес. Адрес - это место в памяти, где расположена переменная. Он определяется при выделении памяти и не может быть изменён программой. Значение - это содержимое отведенного под переменную участка памяти. Значение определяется программой и может многократно изменяться в ходе её работы. Указатель - это тип переменной, значением которой является адрес. Указатель, как всякая переменная, имеет собственный адрес, а значение указателя - это адрес какой-то другой переменной. Как и любая переменная, указатель в программе на Си должен быть описан. Определение указателя имеет вид

<тип данных> *id;

где <тип> данных - это базовый тип указателя id.

Это определение означает: тип переменной id определен как тип указателя на базовый тип данных. Переменная id служит ссылкой (указателем) на объект базового типа и может хранить адрес переменной базового типа.

Пример 1. Ниже приведены примеры определения указателей.

```
int *pi, *qi;  /* Определены  pi, qi -  указатели на целые переменные */
char *q;       /* Определен  q  -  указатель на символьную переменную */
struct {int x,y;} *p;  /* Определен p - указатель на структуру с компонентами x,y* /
complex *x;            /* Определен x - указатель на объект определенного пользователем
                          типа complex */
```

Итак, знак "*" перед именем переменной в описании показывает, что описывается указатель, а тип, указанный перед "*", означает тип переменной, адрес которой содержит переменная - указатель. Так, для примера выше значением переменных pi, qi могут быть адреса переменных типа int, а значением q - адрес переменной типа char.

Пример 2. Два описания вида:

```
int  i;         /* i - хранит целые значения */
int  *prt;      /* prt - хранит адрес величины целого типа */
```

можно компактно записать следующим образом:

```
int  i,*prt;
```

Область применения указателей в Си достаточно широкая. В первую очередь, это динамическое распределение памяти (см. подраздел 4.4) и работа с массивами и строками (см. раздел 5). Кроме того, указатели используются для указания на статические переменные (см. подраздел 4.5), для указания на произвольные ячейки памяти (см. подраздел 4.6), для работы со структурами (см. подраздел 4.7) и др.

4.2 АДРЕСНЫЕ ОПЕРАЦИИ

С использованием указателя связаны две специальные адресные операции:

1) операция определения адреса &;

2) операция обращения по адресу (операция ссылки) *.

Если a - переменная любого типа, то &a - адрес этой переменной. Значение адреса можно присвоить указателю.

Если p - указатель, то *p - это значение, хранящееся по адресу, который является значением указателя (например, если msg является указателем на тип char, то *msg является символом, на который указывает msg).

Пример1.

```
#include <stdio.h>
   int main()
   {int a,*p;   /* a - целая переменная, p - указатель на целое */

    p=&a;
    a=3;
    printf("%d",*p);  /* Печатается 3  */
    a++;
    printf("%d",*p);  /* Печатается 4  */
    (*p)--;
    printf("%d",*p);  /* Печатается 3  */
    (*p)=8;
    printf("%d",*p);  /* Печатается 8  */
   }
```

Пример2.

```
#include <stdio.h>
   int main()
   {int ivar,*iptp;   /* ivar - целая переменная, iptr - указатель на целое */

    iptr=&ivar;       /*  iptr=<адрес ivar>  */
    ivar=421;         /*  ivar=421           */
    printf("Размещение ivar: %p\n",&ivar);
    printf("Содержимое ivar: %d\n", ivar);
    printf("Содержимое iptr: %p\n", iptr);
    printf("Адресуемое значение: %d\n",*iptr);
   }
   /*  Результат работы программы (предполагается, что адрес ivar - 166E):
       Размещение ivar: 166E
       Содержимое ivar: 421
       Содержимое iptr: 166E
       Адресуемое значение: 421     */
```

Замечание. Для вывода значения адреса обычно используется спецификация %p. В принципе, допустимы спецификации %x и %d (в последнем случае значение адреса распечатается в десятичной системе счисления).

Пример 3. Программа ниже - небольшая вариация предыдущей программы из примера 2, дает совершенно тот же результат. Отличие по тексту - в записи одного оператора: вместо " ivar=421;" записано "*iptr=421;". В данном случае оба эти оператора дают один

и тот же результат. Оператор "*iptr=421;" означает: запомнить 421 в переменной типа int, расположенной по адресу, хранящемуся в переменной iptr.

```
#include <stdio.h>
   int main()
   {int ivar,*iptp;   /* ivar - целая переменная, iptr - указатель на целое */

    iptr=&ivar;      /*  iptr=<адрес ivar>  */
    *iptr=421;       /*  ivar=421           */
    printf("Размещение ivar: %p\n",&ivar);
    printf("Содержимое ivar: %d\n", ivar);
    printf("Содержимое iptr: %p\n", iptr);
    printf("Адресуемое значение: %d\n",*iptr);
   }
```

4.3 АДРЕСНАЯ АРИФМЕТИКА

Адресную арифметику иногда называют арифметикой указателей. Разрешено прибавлять к указателям и вычитать из них целые числа. Результат такой операции - указатель; при этом учитывается, на какой тип ссылается указатель, то есть сложение и вычитание производится с учетом длины этого типа.

Предположим, в памяти расположены подряд несколько значений типа int, значением указателя p является адрес одного из них. Тогда p-1 указывает на предыдущее значение, p+1 - на последующее, p+2 - на значение, хранящееся через одно от исходного.

При этом совершенно не важно, сколько байт памяти занимает значение типа int. Этот размер может различаться на разных ЭВМ, но p и p+1 всегда будут указывать на соседние в памяти целые значения. Пример использования адресной арифметики будет приведен ниже - см. пример 3 в п.4.4.2.

4.4 ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ПАМЯТИ

4.4.1 Динамические переменные

Различают статические (заранее определенные) переменные и динамические переменные. Статические переменные создаются при определении переменных (в начале программы) и обозначаются идентификаторами. За каждой такой переменной транслятор закрепляет определенное число ячеек памяти на все время работы программы. Динамические переменные - это "невидимки" в программе; идентификаторами в программе они не обозначаются, транслятор им места в памяти не отводит. Память под такие переменные резервируется и освобождается динамически в процессе выполнения программы по мере необходимости. Динамические переменные не имеют имен, и ссылки на них, то есть работа с ними, выполняются с помощью указателей.

Использование указателей позволяет создавать и обрабатывать новые динамические переменные в процессе выполнения программы. Программа пользователя может запрашивать некоторое количество памяти (в байтах), при этом в программу возвращается адрес выделенной области, который можно запомнить в указателе. Этот прием называется "динамическое распределение памяти". Используя его, программа может приспосабливаться к любому доступному объему памяти, в зависимости от того, как много (или мало) памяти доступно компьютеру.

Замечание. Значение 0 может быть присвоено указателям любого типа. Это значение показывает, что данный указатель не содержит ссылки на какую-либо переменную.

Попытка использовать это значение для обращения к динамической переменной приведет к ошибке. По соглашению, для обозначения константы с нулевым значением используется идентификатор NULL. Его описание "#define NULL 0" содержится среди описаний файла стандартного пакета ввода-вывода stdio.h.

4.4.2 Создание динамических переменных

Для создания динамических переменных служат функции malloc и calloc. Они описаны в библиотеке alloc.h (при использовании функций эту библиотеку надо подключить по include). В некоторых системах, например, в MinGW, библиотека называется malloc.

Вид обращения к функциям:

malloc(size),

где size (типа unsigned) - объем памяти, который необходимо выделить;

calloc(nelem,elsize),

где nelem (типа unsigned) - число элементов, для которых надо выделить память;

elsize (типа unsigned) - объем памяти, который необходимо выделить для каждого элемента.

Обе функции выделяют основную память и возвращают указатель на тип void, содержащий адрес выделенной области памяти. Если памяти недостаточно, функции возвращают NULL.

Функция calloc, кроме отведения памяти, присваивает каждому элементу значение 0, что очень удобно для автоматического создания массивов.

То, что функции имеют тип void (и, значит, возвращают результат типа void), означает, что они возвращают нетипизированный указатель, значение которого затем можно присваивать указателю любого типа данных без предварительного преобразования типов. Но это справедливо лишь в Турбо-СИ, поэтому для переносимости программного обеспечения лучше всегда использовать явное преобразование типа.

Для определения необходимого объема памяти можно использовать унарную операцию sizeof (размер). Эта операция используется в двух формах:

1) sizeof(<выражение>)

2) sizeof(<тип>)

Частный случай выражение - константа или переменная. Результат выполнения операции - количество байтов, занимаемых операндом, то есть объем памяти в байтах, необходимый для хранения:

1) значения выражения;

2) значения заданного типа.

Пример 1.

```
/* Пример иллюстрирует результат выполнения операции sizeof */
#include <stdio.h>
int main ()
{int b;   float d[500]; /*Предполагаем, что int занимает 2 байта, float – 4 байта */
  printf("\nРазмер памяти под целое  %d",sizeof(int));            /* Результат: 2 */
  printf("\nРазмер памяти под переменную b  %d",sizeof(b)); /* Результат: 2 */
  printf("\nРазмер памяти под  d  %d", sizeof d);                 /* Результат: 2000 */
}
```

Пример 2.

```
/* Пример динамического распределения памяти */
/* Динамическое создание одной переменной типа int */
#include <stdio.h>
#include <alloc.h>
int main()
```

```
{int *iptr;    /*  iptr - указатель на целое */
  iptr=(int *) malloc(sizeof(int));
  *iptr=421; /* *iptr - это имя динамической переменнной. Ее значение равно 421.*/
  printf("Содержимое iptr: %p\n",iptr);           /* Вывод: Содержимое iptr: <адрес> */
  printf("Адресуемое значение: %d\n",*iptr); /* Вывод: Адресуемое значение: 421 */
}
```

В примере выше оператор "iptr=(int*) malloc(sizeof(int));" выполняет следующие действия:

1) выражение sizeof(int) возвращает количество байтов, требуемых для хранения переменной типа int (для компилятора Турбо-Си на IBM PC это значение равно 2);

2) функция malloc(n) резервирует n последовательных байтов доступной (свободной) памяти в компьютере, и возвращает начальный адрес размещения в памяти этой последовательности байтов;

3) выражение (int *) указывает, что этот начальный адрес есть указатель на данные типа int. Это выражение явного приведения типа (см. п.1.4.3). Для Турбо-Си это приведение необязательно, но для других компиляторов Си является обязательным. Из соображения переносимости программного обеспечения лучше всегда предусматривать явное приведение типов в своих программах;

4) адрес, полученный с помощью функции malloc , запоминается в iptr. Таким образом, получена динамически созданная целая переменная, к которой можно обращаться при помощи имени "*iptr".

Краткая формулировка действий: "выделить в памяти компьютера некоторый участок для переменной int, затем присвоить начальный адрес этого участка переменной iptr, являющейся указателем на переменную типа int".

Делать присваивание "*iptr=421" без предварительного присвоения адреса в iptr нельзя, так как нет гарантии, что iptr указывает на свободный участок памяти. Вообще, правило использования указателей: указатель всегда должен иметь адрес до своего использования в программе.

Пример 3.

```
/* Динамическое распределение памяти */
/* Адресная арифметика */
/* Динамическое создание трех переменных типа int */
#include <stdio.h>
#include <alloc.h>
int main()
{
#define n 3
  int  *list, i;
list=(int *) calloc(n,sizeof(int));
        *list=421;
        *(list+1)=53;
        *(list+2)=1806;

        printf("\nСписок адресов :");
        for (i=0;i<n;i++)
              printf("%4p ",(list+i));
        printf("\nСписок значений :");
        for (i=0;i<n;i++)
              printf("%4d", *(list+i));
        printf("\n");
}
```

```
/*      list указывает на участок памяти размером 3*2=6 байтов, достаточный для хранения
            3-х элементов типа int. */
/*      list+i представляет адрес памяти, определяемый выражением  list+(i*sizeof(int)). */
/*      Вывод (конкретные адреса могут быть, конечно, другие):
                Список адресов: 163A 163C 163E
                Список значений: 421 53 1806
        Заметьте, что адреса различаются в 2 байта      */
```

Пример 4. В примере указатель инициализируется в момент описания.

```
#include <alloc.h>
main()
{ int *y=(int *)malloc(sizeof(int));   /*  Указатель  y  инициализируется в момент описания;
то есть здесь  y  описан как указатель на целое, начальное значение которого равно
адресу, возвращаемому функцией  malloc*/
```

4.4.3 Доступ к динамическим переменным

Присваивание значения динамической переменной, ссылка на которую задана указателем iptr, выполняется с помощью имени "*iptr", например:

```
*iptr=421; (см. пример 2  из п.9.4.1.2.)
```

Одно и тоже значение может быть присвоено более чем одной переменной-указателю. Таким образом, можно ссылаться на динамическую переменную с помощью более чем одного указателя. Про переменную, к которой можно обращаться с использованием более чем одного указателя, говорят, что она имеет псевдоимена (alias). Например в результате присваивания

```
jptr=iptr;
```

и jptr , и iptr указывают на одну и ту же переменную, то есть они являются псевдоименами.

Неуправляемое использование псевдоимен делает текст программы труднопонимаемым.

4.4.4 Освобождение выделенной памяти

Память, выделенная динамически, должна быть освобождена явным указанием, если она необходима для других целей. В противном случае эта память может быть потеряна, так как станет невозможным ее повторное использование. Явное освобождение памяти, распределенной динамически, осуществляется с помощью функции free (описание в библиотеке alloc.h).

Вид обращения к функции:

```
free(ptr)          где ptr - указатель на void, содержащей адрес
                          1-го байта освобождаемой области памяти.
```

При работе с функцией free надо следить за тем, чтобы не было ссылок на динамические переменные, память для которых уже освобождена (называется "проблема висящей ссылки", приводит к ошибке).

Пример.

```
#include <alloc.h>
int main()
{ int *ptr;
#define n <число>;
   ptr=(int *)calloc(n, sizeof(int));
    . . .
```

```
    free(ptr);
     . . .
}
```

4.5 УКАЗАНИЕ НА СТАТИЧЕСКИЕ ПЕРЕМЕННЫЕ

Указатели могут обеспечивать ссылку на заранее определенные (статические) переменные. Адрес такой переменной определяется с помощью операции определения адреса & (см. п.4.2).

Пример. int i, *pi;
pi=&i;

После этого имена "i" и "*pi" - псевдоимена, то есть на переменную с именем i теперь можно ссылаться с помощью указателя pi, используя имя "*pi" (см. примеры из п.4.2.)

4.6 УКАЗАНИЕ НА ПРОИЗВОЛЬНУЮ ЯЧЕЙКУ ПАМЯТИ

С помощью явных преобразований можно получить указатель на произвольную ячейку памяти. Например, предположим, что pt является указателем типа T. Тогда указатель на ячейку памяти 0777000 можно получить с помощью следующей записи: pt=(T *)0777000;

Обращение к конкретным ячейкам памяти часто бывает необходимо в программах, взаимодействующих с оборудованием, например, в драйверах устройств, когда для управления устройствами надо иметь доступ к таким ячейкам памяти, как регистры состояния или ячейки буфера устройства.

Использовать такие возможности следует осторожно. Большинство операционных систем запрещают обычным пользователям доступ к абсолютным адресам памяти для сохранения целостности системных средств и защиты других пользователей.

4.7 УКАЗАТЕЛИ И СТРУКТУРЫ

Указатели на структуры описываются точно также, как и на другие типы данных.

Указатели на структуры часто используются для создания связных списков и других динамических структур данных, элементами которых, в свою очередь, являются структуры данных.

Пример 1. Предположим, имеются следующие объявления:

```
struct  student {              /*student - метка структуры */
        char name[25];
        int id, age;
        char sex;
};
struct  student  *st;          /*st - это указатель на структуру student */
```

Пусть память уже выделена так, что st указывает на конкретную структуру student. Тогда на компоненты этой структуры можно ссылаться следующим образом:

```
(*st).name
(*st).id
(*st).sex
```

Указатели на структуры так часто используются в Си, что существует специальная операция для ссылки на элементы структуры, адресованной указателем. Эта операция

обозначается стрелкой вправо "->". У этой операции - высший приоритет (такой же, как у круглых и квадратных скобок).

Пример 2. Ссылки на компоненты структуры, описанной выше в примере 1, можно записать так:

```
st->name
st->id
st->sex
```

4.8 УКАЗАТЕЛЬ НА ПУСТОЙ ТИП void

Указатель должен быть объявлен как указатель на некоторый конкретный тип, даже если этот тип - void.

Указатель на void не будет указателем «на ничто», а будет указателем на любой тип данных, причем конкретный тип этих данных знать необязательно. Вы можете присвоить любой указатель указателю типа void и наоборот без приведения типов (справедливо только в Турбо-Си). Но нельзя применять операцию косвенной адресации "*" к указателю типа void, так как используемый тип не определен.

Тип void был введен в язык Си сравнительно недавно. В некоторых вариантах Си такой тип отсутствует и вместо него часто используется обычный тип int.

4.9 ЛАБОРАТОРНАЯ РАБОТА №4 "ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ПАМЯТИ"

Одно из основных применений указателей в Си - это динамическое распределение памяти. Суть подхода «динамическое распределение»:

- в процессе работы программа может запросить у системы некоторый объем памяти;
- система выделяет, если возможно, память и возвращает программе начальный адрес выделенной области;
- этот адрес записывается в указатель, и через этот указатель программа работает с выделенной памятью;
- когда память больше не нужна, программа может ее освободить, и система в дальнейшем имеет возможность использовать эту область в процессе работы.

Таким образом, динамическое распределение памяти позволяет экономно расходовать память в процессе работы программы и дает возможность программе "приспособиться" к любому доступному объему памяти.

Цель лабораторной работы "Динамическое распределение памяти" - научиться писать на языке Си программы, в которых реализуется динамическое распределение памяти, а также усвоить на примере практической задачи понятие указателя, которое рассматривается в данном модуле 4.

Задание к лабораторной работе: написать программу, в которой реализуется динамическое распределение памяти. Это означает, что все основные используемые переменные, в первую очередь массивы, должны быть динамические. В программе необходимо предусмотреть печать исходных, промежуточных и полученых данных, причем каждый раз следует распечатывать два компонента динамической переменной: значение и адрес, по которому расположено это значение. Вариант конкретного задания выбрать из списка заданий в п. 4.9.1. Примеры программ с динамическим распределением памяти см. в подразделе 4.4. При выполнении данной работы необходимо создать (ввести с клавиатуры) динамический массив. Существует много вариантов того, как это можно сделать. Ниже приведено несколько примеров создания динамических массивов.

Пример 1. Фрагмент программы, приведенный ниже, позволяет создать и распечатать динамический одномерный символьный массив.

```
        . . .
int m,   /* Количество элементов массива */
    i;   /* Индекс массива */
char  *mp;  /* mp - указатель, который позволит работать с динамическим массивом*/
printf("\nВведите количество элементов массива: ");
scanf("%d",&m);
mp=(char *)calloc(m,sizeof(char)); /* Выделение памяти под массив */
if (mp==NULL)
        {puts(“Нет памяти. Конец работы.”);
         return 1;
        }
fflush(stdin);
printf("\nВведите массив из  %d  элементов: \n",m);
for(i=0;i<m;i++)
  scanf("%c",mp+i); /*  Этот оператор можно записать в виде:   scanf("%c",&mp[i]); */
printf("\nЭлементы массива и их адреса: \n");
for(i=0;i<m;i++)
  printf("%2c  %p",*(mp+i),mp+i);
        . . .
```

Пример 2. Фрагмент программы, приведенный ниже, позволяет создать динамическую целочисленную матрицу (двумерный массив).

```
        . . .
int n,m,   /* n - число строк, m - число столбцов матрицы */
    i,j;   /* Индексы массива */
int *mp;  /* mp - указатель, который позволит работать с динамическим массивом*/
printf("\nВведите число строк и столбцов матрицы: ");
scanf("%d%d",&n,&m);
mp=(int *)calloc(n*m,sizeof(int)); /* Выделение памяти под массив */
if (mp==NULL)
        {puts(“Нет памяти. Конец работы.”);
         return 1;
        }
printf("\nВведите целую матрицу  (%d*%d)  построчно: \n",n,m);
/*  Первый способ ввода матрицы  */
for(i=0;i<n;i++)
 for(j=0;j<m;j++)
  scanf("%d",mp+i*m+j); /* Этот оператор можно записать: scanf("%d",&mp[i*m+j]); */
/*  Второй способ ввода матрицы  */
for(i=0;i<n*m;scanf("%d",mp+i++);
        . . .
```

4.9.1 Динамическое распределение памяти и указатели

ЗАДАНИЕ. Написать программу, в которой все основные используемые переменные - динамические.

№1.

Дана динамическая символьная матрица и динамическая целая переменная h. Если h<1, то построить динамический одномерный массив, каждый элемент которого равен разности в одной строке числа элементов исходной матрицы, равных ‘a’ и равных ‘b’;

если h≥1, то вычислить количество элементов, равных ‘a’ и ‘b’ в исходной матрице, и записать это число как динамическую переменную.

Освободить память от матрицы.

Все исходные данные, результаты и соответствующие адреса распечатать.

№2.

Дана динамическая целочисленная матрица. Построить динамический одномерный массив, каждый элемент которого равен сумме элементов одной строки исходной матрицы.

Освободить память от матрицы, найти произведение элементов одномерного массива и записать его как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса распечатать.

№3.

Дана динамическая целочисленная матрица. Построить динамический одномерный массив, каждый элемент которого равен максимуму элементов одной строки исходной матрицы.

Освободить память от матрицы. Найти сумму абсолютных значений элементов одномерного массива и записать ее как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса распечатать.

№4.

Дана динамическая целочисленная матрица. Построить динамический одномерный массив, каждый элемент которого равен сумме четных элементов одной строки исходной матрицы.

Освободить память от матрицы. Найти максимальный элемент одномерного массива и записать его как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса распечатать.

№5.

Дана динамическая целочисленная (n×m) матрица. Построить динамический одномерный символьный массив из n элементов, каждый элемент которого равен’2’, если в соответствующей строке исходной матрицы количество четных элементов больше количества нечетных, и ‘1’ - в противном случае.

Освободить память от матрицы. Найти количество ‘2’ и ‘1’ в одномерном массиве и эти два числа записать как динамические переменные.

Все исходные данные, результаты и соответствующие адреса распечатать.

№6.

Дан динамический символьный массив $c_1, c_2, \ldots c_n$ и динамическая целая переменная h. Если h<1, преобразовать исходный массив к виду $c_n, c_{n-1}, \ldots c_2, c_1$; иначе - не менять массив.

Определить номер первого знака ‘+’ в последовательности, освободить память от массива и записать этот номер как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса распечатать.

№7.

Дан динамический символьный массив $c_1, c_2, \ldots c_k$ и динамическая целая переменная a. Если a<0, преобразовать исходный массив к виду $c_3, c_4, \ldots c_k, c_1 c_2$; если a≥0, то к виду $c_k, c_{k-1}, c_1 c_2 \ldots c_{k-2,}$.

Определить символьную переменную ch по следующему правилу: ch равно последнему элементу преобразованного массива; освободить память от массива и записать ch как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса распечатать.

№8.

Дан динамический символьный массив c_1, c_2, ... c_m и динамическая целая переменная a. Если a≠0, преобразовать исходный массив к виду $c_2, c_3, ... c_m, c_1$; если a=0, то к виду $c_m, c_1, ... c_{m-1,}$.

Определить значение новой целой переменной t как номер первого символа ‘1’ в массиве; освободить память от массива и записать t как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса распечатать.

№9.

Дан динамический целочисленный массив y_1, y_2, ... y_n. Проверить, являются ли первый и последний член массива четными.

Освободить память от массива и создать новый динамический символьный массив из 3-х переменных, каждая из которых равна '+' в случае успешной проверки, и '-' - в противном случае.

Все исходные данные, результаты и соответствующие адреса распечатать.

№10.

Дан динамический символьный массив c_1, c_2, ... c_m. Подсчитать n - количество знаков '+' и '-' среди c_1, c_2, ... c_m.

Освободить память от массива. Создать новый динамический целочисленный массив из n элементов, если n>m/2 или из n/2 элементов - в противном случае. Элементы нового массива равны 1.

Все исходные данные, результаты и соответствующие адреса распечатать.

№11.

Дан динамический символьный массив c_1, c_2, ... c_n, элементы которого - символы ‘f’ и ‘i’. Подсчитать количество знаков ‘f’ и количество знаков ‘i’, освободить память от массива и создать новый динамический массив из n элементов. Элементы нового массива-вещественные числа 0.1, если знаков ‘f’ больше знаков ‘i’, и целые числа 1 - в противном случае.

Все исходные данные, результаты и соответствующие адреса распечатать.

№12.

Дан динамический целочисленный массив x_1, x_2, … $x_{n.}$ Создать динамическую символьную переменную h по следующему правилу: h=’1’, если количество нечетных элементов массива больше четных, и h=’2’ - в противном случае.

Освободить память от массива и создать новый динамический символьный массив из n элементов, каждый элемент которого равен h.

Все исходные данные, результаты и соответствующие адреса распечатать.

№13.

Дан динамический целочисленный массив y_1, y_2, … $y_{n.}$ и динамическая символьная переменная h. Если значение h равно ‘+’, преобразовать исходный массив к виду $y_n, y_{n-1}, … , y_2, y_1$; иначе - не менять порядок элементов.

Определить значение первого положительного элемента массива, освободить память от массива и записать номер этого элемента как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса памяти распечатать.

№14.

Дан динамический целый массив x_1, x_2, … x_n и динамическая символьная переменная ch. Если значение ch равно ’+’, найти максимальный элемент массива, иначе - минимальный.

Освободить память от массива и записать найденное значение как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса распечатать.

№15.

Дан динамический целочисленный массив $x_1, x_2, \dots x_n$ и динамическая символьная переменная ch. Если значение ch равно ’-’, преобразовать исходный массив в массив $x_1-x_n, x_2-x_n, \dots x_n-x_n$., если ‘+’ - в массив $x_1+x_n, x_2+x_n, \dots x_n+x_n$.

Найти сумму элементов полученного массива, освободить память от массива и записать сумму как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса распечатать.

№16.

Дан динамический целочисленный массив $x_1, x_2, \dots x_n$ и динамическая символьная переменная ch. Если значение ch равно ’+’, найти сумму элементов массива; если ch равно ‘*’ - произведение.

Освободить память от массива и полученное значение записать как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса распечатать.

№17.

Дан динамический целочисленный массив $x_1, x_2, \dots x_n$ и динамическая символьная переменная ch. Если значение ch равно ’a’, преобразовать исходный массив к виду: $x_2, x_3, \dots x_n, x_1$; если ch равно ’z’, к виду: $x_n, x_1, x_{2, \dots} x_{n-1}$.

В полученном массиве найти m - номер первого положительного элемента. Освободить память от массива и записать m как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса памяти распечатать.

№18.

Дан динамический целочисленный массив $x_1, x_2, \dots x_n$ и динамическая символьная переменная c. Если значение c равно ’<’, упорядочить исходный массив по убыванию; если c равно ‘>‘ - по возрастанию.

В полученном массиве найти k - номер первого элемента, на котором произошла смена знака. Освободить память от массива и записать k как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса распечатать.

№19.

Дан динамический целочисленный массив $x_1, x_2, \dots x_n$ и динамическая символьная переменная c. Если значение c равно ’+’, получить новый динамический массив $x_1+x_n, x_2+x_n, \dots x_n+x_n$; если c равно ‘-‘, получить динамический массив $x_1-x_n, x_2-x_n, \dots x_n-x_n$

В полученном массиве определить количество отрицательных элементов k. Освободить память от первого массива и записать k как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса распечатать.

№20.

Дан динамический целочисленный массив $x_1, x_2, \dots x_n$ и динамический символьный массив $c_1, c_2, \dots c_m$. Для первого массива найти S - сумму положительных элементов, второй массив преобразовать к виду $c_m, c_1, c_2, \dots c_{m-1}$.

Освободить память от первого массива и записать S как динамическую переменную.

Все исходные данные, результаты и соответствующие адреса распечатать.

№21.

Дан динамический массив целых чисел x_1, x_2, … x_n. Создать новый динамический массив, элементы которого $x_1 \cdot x_n$, $x_2 \cdot x_{n-1}$, … $x_n \cdot x_1$. Вычислить сумму элементов полученного массива и записать ее в динамическую переменную.

Освободить память от обоих массивов.

Все исходные данные, результаты и соответствующие адреса распечатать.

№22.

Дан динамический массив целых чисел x_1, x_2, … x_n и динамический символьный массив c_1, c_2, … c_m.

Найти $p=(x_1+x_n)\cdot(x_2+x_{n-1})\cdot\ldots\cdot(x_n+x_1)$, освободить память от первого массива и записать p в динамическую переменную.

Все исходные данные, результаты и соответствующие адреса памяти распечатать.

№23.

Дан динамический символьный массив c_1, c_2, … c_n. Создать новый динамический массив, элементы которого - это элементы исходного массива, записанные в обратном порядке.

Освободить память от первого массива. Определить в новом массиве номер первого символа ‘1’, и записать этот номер в динамическую переменную.

Освободить память от второго массива.

Все исходные данные, результаты и соответствующие адреса памяти распечатать.

4.10 КОНТРОЛЬНЫЕ ВОПРОСЫ ПО МОДУЛЮ 4

4.10.1 Понятие указателя.
4.10.2 Адресные операции "&" и "*".
4.10.3 Адресная арифметика.
4.10.4 Динамическое распределение памяти - общее понятие.
4.10.5 Создание динамических переменных. Стандартные функции malloc() и calloc().
4.10.6 Операция определения размера sizeof.
4.10.7 Доступ к динамическим переменным.
4.10.8 Освобождение выделенной памяти.
4.10.9 Указатели на структуры.

5 МАССИВЫ, СТРОКИ И УКАЗАТЕЛИ

Цель изучения данного модуля: рассмотреть связь таких трех основных понятий, как указатели, массивы и строки, и освоить основные возможности по обработке текстовой информации, а именно строк, в Си.

5.1 СВЯЗЬ МЕЖДУ УКАЗАТЕЛЯМИ И МАССИВАМИ

В языке Си массивы и указатели тесно связаны. Имя массива можно рассматривать как константу-указатель на начальный элемент; можно, наоборот, определить указатель как массив.

Имя массива, указанное без индекса, является указателем на начальный элемент массива (адрес начального элемента). Если, например, a - одномерный массив, то запись "a" означает адрес нулевого элемента массива, то есть то же самое, что и запись "&a[0]". Тогда, в соответствии с арифметикой указателей, "a+i" - это адрес i-го элемента массива, а запись "*(a+i)" равносильна записи "a[i]".

Таким образом, имеем следующие тождества:

```
   a    = = &(a[0])
 (a+i) = = &(a[i])
*(a+i) = = a[i]
```

Выражения слева и справа тождественны, и часто можно использовать одно вместо другого, не принимая во внимание, описано "a" как указатель, или как массив.

Основное отличие между описанием "a" как указателя или как массива состоит в размещении самого массива. Если "a" описан как массив, то программа автоматически выделяет требуемый объем памяти. Если же переменную "a" описали как указатель, то обязательно необходимо выделить память под массив, используя для этого функцию calloc (или сходную с ней), или присвоить этой переменной адрес некоторого сегмента памяти, который был ранее определен.

Пример 1.

/* Ниже приведены две программы: слева - с использованием массива list[n], справа - с использованием указателя *listt. Обе программы дают один и тот же результат. */

```
 int main()
 {
 #define n 3
   int list[n],i;

   list[0]=421;
   list[1]=53;
   list[2]=1800;
 print("Список адресов: ");
 for (i=0; i<n; i++)
      printf("%4p ", &list[i]);
 printf("\n Список значений :");
 for(i=0; i<n; i++)
       printf("%4d ", list[i]);
 printf("\n");
 }
```

```
int main()
{
#define n 3
   int *listt,i;
  listt=(int *)calloc(n,sizeof(int));
  *listt=421;
  *(listt+1)=53;
  *(listt+2)=1800;
print("Список адресов: ");
for (i=0; i<n; i++)
      printf("%4p ", (listt+i) );
printf("\n Список значений :");
for(i=0; i<n; i++)
       printf("%4d ", *(listt+i) );
printf("\n");
}
```

Замечание 1. Доступ к данным массивов list и listt можно выполнять одинаково. Чтобы получить третье значение в массиве list, надо указать list[2]. Для получения третьего значения в listt можно указать listt[2].

Замечание 2. Можно рассмотреть еще третий вариант программы из примера 1 выше, который дает тот же самый результат:

```
int main()
{
#define n 3
  int *listt, list[n], i;
listt=list;
*listt=421; /* То же, что list[0]=421;  */
*(listt+1)=53; /* То же, что list[1]=53;  */
*(listt+2)=1800; /* То же, что list[2]=1800;*/
. . . /*Далее все так же, как в примере выше*/
```

Обратите внимание на оператор "listt=list", в результате которого указателю listt присваивается адрес начала статического массива list. Записать наоборот "list=listt" нельзя. Это приведет к ошибке, так как означает попытку изменить адрес-константу статического массива list.

Пример 2. Имя массива всегда трактуется как ссылка (адрес). Если сделать копию такой ссылки в указателе, то, увеличивая или уменьшая значение ссылки (значение указателя), можно ссылаться на отдельные элементы массива.

```
  int  g[5], *gptr, s;
     gptr=g;   /*  Сейчас  gptr  указывает на  g[0]  */
     gptr++;   /*  Сейчас  gptr  указывает на  g[1]  */
     gptr++;   /*  Сейчас  gptr  указывает на  g[2]  */
  gptr=g;   /* Сейчас gptr  указывает на начало массива g, т. е. на g[0]  */
  for(s=0;s<5;s++) *gptr++=0;   /*  Обнуление элементов массива  g  */
```

Замечание 1. Рассмотрим выполнение оператора "*gptr++=0;". Вначале операция обращения по адресу "*" выдает значение по адресу gptr. Это значение становится равным нулю. Так обнуляется элемент g[0] в первой итерации цикла. Далее адрес gptr согласно операции "++" увеличивается на 1.

Замечание 2. Увеличивать или уменьшать с помощью операций "++" или "—" само g нельзя, потому что g является константой-указателем (изменение g означало бы изменение адреса, или положения, статического массива в памяти, а это недопустимо). Можно использовать операции, подобные g+1, для идентификации следующего элемента массива. Но выражение вида "g++" не разрешается, так как операцию увеличения можно использовать с именами переменных, а не констант.

Замечание 3. В данном примере для обнуления элементов массива в цикле for записано " *gptr++=0 ". Запись вида " *++gptr=0 " привела бы к ошибке, так как при этом обнуляется массив "со сдвигом", то есть нулевой элемент не обнуляется, но зато обнуляется следующий за массивом элемент, то есть "портится" чужая память.

Пример 3.

```
#include <stdio.h>
#include <alloc.h>
int main()
{ int  *data=(int *)malloc(20*sizeof(int));
  int  i;
   for(i=0;i<20;i++)
       data[i]=10*i;     /* Итак:  *data=data[0]=0, data[1]=10 и т.д. */
   printf("\n ++*data=%d", ++*data);   /* Вывод: ++*data=1 ,  так как выражение "++*data"
              увеличивает значение *data на единицу до его использования, и в результате
              на печати будет  1 */
   printf("\n *++data=%d", *++data);   /*  Вывод:  *++data=10 ,  так как вначале адрес  data
              увеличивается на единицу, а затем выбирается значение по этому адресу, оно
              равно  10  */
```

```
}
```

Наиболее серьезные ошибки при работе с массивами и указателями:
1) использование неинициализированного указателя (см. пример 4 ниже);
2) выход за границу массива (см. пример 5 ниже).

Пример 4.

```
/* В программе есть ошибка "Неинициализированный указатель" */
#include <stdio.h>
int main()
{ int  *x;
    *x=16;
    printf("\n Значение указателя x=%p",x);
    printf("\n Значение по такому адресу =%d",*x);
}
/* Предполагаемый вывод:  Значение указателя x=1486
                          Значение по такому адресу = 16       */
```

Замечание. Программа может работать, а может привести к зависанию ЭВМ вплоть до перезагрузки. Ошибка в том, что указатель x - не инициализирован. Описание "int *x;" приводит к тому, что компилятор резервирует память для хранения адреса (2 байта обычно). Но при этом не отводит 2 байта для размещения целого числа по этому адресу. Кроме того, случайное начальное значение адреса x может не соответствовать допустимому адресу, например, может совпасть с адресом операционной системы. Для того, чтобы исправить ситуацию, надо записать:

```
x=(int*) malloc(siteof(int));
```

до оператора "*x=16", а также добавить выше строчку "#include <alloc.h>".

Пример 5.

```
/* В программе есть ошибка "Выход за границу массива */
#include <stdio.h>
#include <alloc.h>
int main()
{int  *data, size,i;
        printf("Размер массива :  ");
        scanf("%d,&size);
        data=(int *) malloc(size *sizeof(int));
        for (i=0; i<=size; i++)
             data[i]=100*i;
}
/* Ошибка! В цикле for делается попытка присвоить значение 100*size неопределенному элементу data[size], память под который не выделялась. Программа может работать, а может зависнуть. */
```

5.2 МАССИВЫ, СТРОКИ, УКАЗАТЕЛИ

5.2.1 Понятие строки

Строки - это массивы знаков, то есть массивы элементов типа char. При этом количество элементов массива на единицу больше числа символов в строке: в конце

строки должен быть специальный нулевой символ с кодом 0. (не путать с символом '0', код которого отнюдь не 0).

Пример 1. Строка "СИ" представляет собой массив из 3-х элементов. Если a - имя этого массива, то:

элемент a[0]='С'
элемент a[1]='И'
элемент a[2]=0 (или можно записать, что a[2]='\0').

Итак, элемент a[2] равен знаку '\0' (но не знаку '0' !) или просто 0, так как в СИ разрешается использовать числа (коды) в качестве значений для типа char.

Если строка задана константой (например, строка формата в операторе printf), то нулевой знак в конце строки добавляется автоматически. Во всех остальных случаях явного формирования строки в программе нулевой знак в конец строки должен добавить программист.

Итак, строка - это массив символов. Имя массива фактически является указателем на первый элемент массива, поэтому строку в программе можно описать и как массив символов, и как указатель на символ.

Пример 2. char msg[30],*msg1;

Для строки, описанной как "char msg[30]", память резервируется автоматически. Итак, msg - это массив из 30 знаков, и можно сказать, что это строка, которая содержит максимум 29 "реальных знаков".

Для строки, описанной как знаковый указатель "char *msg1" память должна быть выделена явно (с помощью функции malloc и т. п.), или ей должен быть присвоен адрес уже существующей строки. Заметим, что память должна быть также выделена или зарезервирована для нулевого знака. Итак, msg1 - это указатель на char. Можно сказать, что это массив или строка с начальным адресом msg1 и пока неопределенным числом элементов.

Итак, строки могут рассматриваться двояко - и как массивы, и как указатели - и все в одной программе. Это особенно важно, когда строки передаются как аргументы функции. Вызывающая программа может рассматривать строку как массив знаков, а вызываемая функция может рассматривать её как знаковый указатель.

Если длина строки непостоянна, то использование знаковых указателей для строк имеет определенные преимущества. Например, для размещения строк разной длины может быть создан массив знаковых указателей; альтернативное решение с использованием двухмерного массива знаков в общем случае будет использовать память неэффективно, так как в этом случае потребовалось бы сделать число столбцов равным числу знаков в строке наибольшей возможной длины.

5.2.2 Определение строки

Способы определения строки-константы, то есть строковой константы:

1) непосредственно по тексту: если компилятор в тексте программы встречает что-то, заключенное в двойные кавычки, он воспринимает это как строковую константу; нулевой знак в конце строки добавляется автоматически; например:

```
puts(“Привет!”); /* Слово “Привет” в кавычках – это стоковая константа */
```

2) с помощью директивы препроцессора define, например

```
#define msg "Привет!"
⋮
puts(msg);      /* Вывод строки "Привет!" на экран */
```

Способы определения строки - переменной:

1) использование символьного массива;
2) использование указателя на символ.

Пример 1.

```
/*Использование указателя на символ для определения строки */
#include <stdio.h>
int main()
{
 char *msg; /* msg является указателем на символ, то есть msg может хранить адрес
            некоторого символа.  При этом компилятор не выделяет никакого пространства
            для размещения символов и не инициализирует msg каким-нибудь конкретным
            значением */
 msg="Hello";
 puts(msg);
}
```

Выполнение оператора "msg="Hello";"

– создает строку "Hello", сопровождаемую нулевым знаком (\0), в некотором месте файла объектного кода);

– присваивает начальный адрес этой строки - адрес символа "H" - переменной msg.

Пример 2.

```
/* Определение строки как символьного массива */
#include <string.h>
#include <stdio.h>
int main()
{
 char msg[30]; /* Выделяет память для массива из 29 символов. Одно знакоместо - для
                  нулевого символа */
 strcpy(msg, "Hello");
 puts(msg);
}
```

Выполнение оператора "strcpy(msg, "Hello");":

– создает строку "Hello", сопровождаемую нулевым знаком (\0), в некотором месте файла объектного кода;

- стандартная подпрограмма strcpy() копирует символы из этой строки по одному в участок памяти, указываемой переменной msg. Это делается до тех пор, пока не будет скопирован нулевой символ в конце строки "Hello". То есть функция strcpy() дает посимвольное копирование из строковой константы "Hello" в массив msg. Запись msg="Hello" недопустима, так как имя массива msg является константой-указателем. Значение этой константы, то есть начального адреса массива, определяется компилятором в момент описания массива, и изменить его в программе (менять на адрес строки "Hello") нельзя, так как изменять константы в программе нельзя.

Пример 3.

```
/* Комбинированное определение строк */
/* Явное посимвольное формирование строки */
/* Различные способы печати строки */
#include <stdio.h>
#define  n  80
int main()
{
  char  mess[n], *m;   /* Комбинированное определение строки */
```

```
        mess[0]='H';
        mess[1]='e';
        mess[2]='l';        } Явное посимвольное
        mess[3]='l';        формирование строки
        mess[4]='o';        mess
        mess[5]='\0';
         m=mess;        /* Формирование строки m */
        /*  1-ый вариант печати строки  */
   while (*m!=NULL) putch(*m++);
        /*  2-ой вариант печати строки */
   printf("%s\n ",m);
        /*3-ий вариант печати строки*/
   puts(mess);    /* или */    puts(m);
}
```

5.2.3 Ввод строк

Пример 1.

```
#include <stdio.h>
int main()
{
 char  name[30];
 printf("\n Как Вас зовут ? ");
 scanf("%s", name);
 printf("Привет, %s\n", name);
}
```

Замечание 1. Так как name является массивом символов, то значение name - это адрес самого массива. Поэтому в операторе scanf перед именем name не используется операция получения адреса &.

Замечание 2. Использован массив символов (char name[30]), а не указатель на символ (char *name). Причина: объявление массива резервирует память для хранения его элементов, а при объявлении указателя этого не происходит. В случае объявления "char *name" пришлось бы явным образом резервировать память для хранения переменной *name.

Замечание 3. Если на запрос программы "Как Вас зовут ?" пользователь введет одно имя, например "Ольга", то программа выдаст ответное сообщение в виде "Привет, Ольга". Если же на запрос программы пользователь введет имя и фамилию через пробел, например "Ольга Иванова", то ответное сообщение будут таким же: "Привет, Ольга". Дело в том, что введенный после имени пробел сигнализирует функции scanf о конце вводимой строки, и в переменную name записывается только имя. Возможные решения этой проблемы:

1) ввести две отдельные строки: строка с именем и строка с фамилией (см. пример 2 ниже);
2) ввести одну строку с именем и фамилией, используя для ввода функцию gets, которая пробелы во входной строке воспринимает как реальные знаки (см. пример 3 ниже).

Пример 2.

```
/* Ввод двух строк с помощью функции scanf() */
#include <stdio.h>
int main()
{
 char  name[20], fam[20];
```

```
 printf("\n Как Вас зовут (имя и фамилия) ? ");
 scanf("%s%s", name,fam);
 printf("Здравствуйте, %s %s\n", name,fam);
}
```

Замечание. Для данного случая в функции scanf спецификации %s можно записывать подряд друг за другом: "%s%s", а можно разделять пробелом: "%s %s", - это не имеет никакого значения. При вводе с клавиатуры имя и фамилию можно разделить или пробелом, или клавишей "Ввод" - это также не имеет значения.

Пример 3.

```
/* Ввод строки с помощью функции gets */
#include <stdio.h>
int main()
{
 char name[40];
 printf("\n Как Вас зовут (имя и фамилия через пробел) ? ");
 gets(name);
 printf("Здравствуйте, %s\n", name);
}
```

Замечание 1. Функция gets читает все, что набирает пользователь, до тех пор, пока он не нажмет клавишу "Ввод". Код клавиши "Ввод" не помещается в строку, однако в конец строки добавляется нулевой символ '\0' */

Замечание 2. Если при вводе строки количество вводимых знаков превысит размер строки, установленный при ее описании, то это, скорее всего, приведет к сбою работы программы, так как лишние знаки запишутся в память в "чужую" область, не зарезервированную под строку.

5.2.4 Строки-резюме

5.2.4.1 Строку можно описать как массив символов, например:

```
char strarr[n];   /*  n - константа */,
```

или как указатель на символ, например:

```
char *strptr;
```

Первое описание выделит n байтов для строки и запишет адрес-константу в strarr.

Второе описание только выделит несколько байт для указателя strptr, который указывает на char.

Длина строки не хранится. Вместо этого используется маркировка конца строки с помощью специального разделителя - нулевого символа, обозначаемого '\0' и имеющего код 0.

Строка начинается с элемента strarr[0] и может содержать только (n-1) "реальных " символов, так как один байт должен быть отведен для разделителя - нулевого символа.

5.2.4.2 В первом случае определения строки в виде ”char strarr[n];” идентификатор strarr не является фактической последовательностью байт (strarr - это константа-адрес нулевого элемента массива), и strarr нельзя прямо присвоить строку литералов. Вместо этого надо использовать стандартную подпрограмму strcpy() (или подобные) для побайтной передачи одной строки в другую. Например:

```
strcpy(strarr,"Здравствуй, друг !");
```

Однако можно напрямую читать в strarr, используя функции scanf и gets.

5.2.4.3 Во втором случае определения строки в виде "char *strptr;" пространство для содержимого строки не распределяется.

Можно прямо присвоить strptr значение строки литералов. Литералы создаются как область с их объектным кодом, поэтому при таком присвоении адрес этой области присваивается переменной strptr. Например:

strptr = "Здравствуй, друг !";

Можно присвоить strptr значение strarr. В этом случае обе переменные будут указывать на одну и ту же строку. Например:

strptr = strarr;

То же самое произойдет, если присвоить strptr значение другого указателя на строку.

Если требуется, чтобы strptr указывал именно на свою строку, то надо явно выделить пространство. Например:

strptr = (char *)malloc(n);

В результате выделится n байт имеющейся памяти, и strptr присвоится адрес этой области. Затем можно использовать strcpy для переписывания строк (литералов или переменных) в эти выделенные байты.

5.2.5 Типичные ошибки при работе со строками

Пример 1.

```
/* Внимание !  Программа содержит ошибки */
#include <stdio.h>
int main()
{
 char  *name;
 char  msg[10];

 printf("\n Ваше имя ? ");
 scanf("%s", name);
 msg="Здравствуйте, ";
 printf("%s%s\n", msg,name);
}
```

Ошибка 1 - оператор scanf("%s", name);

Память для name не резервируется; введенная строка запишется по какому-то случайному адресу, который окажется в name. Эта ситуация не приведет к обязательному сбою в выполнении программы, так как строка может быть сохранена. Поэтому компилятор выдаст лишь предупреждающее сообщение вида:

"Possible use of 'name' before definition"

(Возможно использование name до его определения)

Ошибка 2 - msg="Здравствуйте, ";

Компилятор считает, что Вы пытаетесь заменить значение msg на адрес строковой константы "Здравствуйте, ". Это сделать невозможно, так как имена массивов являются константами и не могут быть модифицированы (как, например, число 7 является константой и нельзя записать "7=i"). Компилятор выдаст сообщение об ошибке вида:

"Lvalue required"

(Необходим адрес переменной, т.е. использование константы недопустимо)

Возможно два варианта решения проблемы - см. примеры 2 и 3 ниже.

Пример 2.

```
/* 1-ый вариант исправления программы из примера 1 - изменить способ описания
переменных name и msg */
#include <stdio.h>
int main()
{
 char  name[10];
```

```
 char  *msg;

 printf("\n Ваше имя ? ");
 scanf("%s", name);
 msg="Здравствуйте, ";
 printf("%s%s\n", msg,name);
}
```

Пример 3.

```
/* 2-ый вариант исправления программы из примера 1 с сохранением первоначального описания переменных name и msg */
#include <stdio.h>
int main()
{
 char  *name;
 char  msg[10];

 name=(char *)malloc(10);
 printf("\n Ваше имя ? ");
 scanf("%s", name);
 strcpy(msg,"Здравствуйте, ");
 printf("%s%s\n", msg,name);
}
```

5.2.6 Примеры работы со строками

Пример 1.

```
/* Определение длины строки без использования стандартной подпрограммы  strlen */
#include <stdio.h>
int main()
{
 char  s[30];   /*  s - исходная строка */
 int  i;            /*  i - длина строки s   */

 printf("\n Введите строку \n");
 gets(s);
 for(i=0;s[i]!=0;i++) ;   /* В цикле for нет операторов - пустой оператор */
 printf("Длина введенной строки равна %d", i);
}
```

Пример 2.

```
/* Присваивание строк побайтно , то есть  t=s, где  t,s - строки */
#include <stdio.h>
int main()
{
 char  s[30], t[30], *s1, *t1, *t2;
 int  i;
/* 1-ый способ с использованием массивов s, t */
 printf("\n Введите строку\n");
 gets(s);
 i=0;
 do
    {t[i]=s[i];}
```

```
  while (s[i++]!=0);
  puts(t);
/* Конец 1-го способа */
/* 2-ой способ с использованием указателей  s1, t1 */
  s1=(char *)malloc(30);
  printf("\n Введите строку\n");
  gets(s1);
  t1=(char *)malloc(30);
  t2=t1;   /* Запоминаем в t2 начальный адрес t1 */
  while (( *t1++ = *s1++ ) != 0);
  puts(t2);
/* Конец 2-го способа */
}
```

Замечание 1. В операторе "while ((*t1++ = *s1++) != 0);" при вычислении выражения " *t1++ = *s1++" значение по адресу t1 (то есть *t1) становится равным значению по адресу s1 (то есть *s1), после чего адреса увеличиваются. Если был скопирован ненулевой символ, то при следующем выполнении цикла будет скопирован следующий символ. После копирования концевого нулевого символа цикл прекращает работу.

Замечание 2. Запись цикла while во 2-ом способе возможно сократить. Это сокращение основано на том, что проверка в Си означает сравнение с нулем. Поэтому оператор

```
while (( *t1++ = *s1++ ) != 0);
```

можно кратко записать в виде

```
while ( *t1++ = *s1++ );
```

Пример 3.

```
/* Вывод строки - разные способы */
#include <stdio.h>
int main()
{
char  s[20], *str;
  printf("\n Введите строку\n");
  gets(s);
  str=s;     /* В str запоминаем начальный адрес строки s */
  puts(s);        /* 1-ый вариант печати строки s */
  puts(str);      /* 2-ой вариант печати строки s */
  while (*str!=NULL) putchar(*str++);     /* 3-ий вариант печати строки s */
  str=s;                  /* В str восстанавливаем начальный адрес строки s */
  while (*str++!=NULL) putchar(*str);     /* Этот вариант печати строки s содержит ошибку!
          Не распечатается начальный элемент строки  s  (так как адрес  str  увеличится на
          1 до того, как будет распечатано соответствующее значение *str, а не после того)
          и в конце распечатается лишний пробел  после  последнего  реального  знака
          строки  s  (это будет печать нулевого знака  putchar(0) ). */
}
```

Пример 4.

```
/* Вывод строки - разные способы */
#include <stdio.h>
#include <alloc.h>
int main()
{char *c,*c1;     int i,n;
  c=(char *)calloc(10,sizeof(char));
  printf("\n Введите строку\n");
  gets(c);
```

```
 c1=c;     /* В c1 запоминаем начальный адрес строки c */
 n=strlen(c);

       for (i=0;i<n;i++) putchar(*(c+i));   /* 1-ый способ печати строки */
       for (i=0;i<n;i++) putchar(*c++);     /* 2-ой способ печати строки */
c=c1; for (i=0;i<n;i++) putchar(*(c++));  /* 3-ий способ печати - точно такой же, как 2-ой */
c=c1; for (i=0;i<n;i++) putchar((*c)++);  /* Этот способ печати строки содержит ошибку !
                  Вначале печатается (*c), то есть начальный элемент строки, затем(*c)
                  увеличивается на 1 и печатается опять и т.д. То есть все время печатается
                  начальный элемент строки, увеличиваемый на 1 */
       for (i=0;i<strlen(c);i++) putchar(*(c+i)); /* 4-ый способ печати строки */
       for (i=0;i<strlen(c);i++) putchar(*c++)); /* Этот способ печати строки содержит
                  ошибку !  Сравните со способом  2-ым !  В данном случае ошибка
                  произойдет потому, что когда адрес c увеличивается на 1, то длина
                  строки, определяемая по strlen(), уменьшается на 1. Поэтому
                  распечатается только половина строки. */
}
```

5.2.7 Стандартные функции для работы со строками

Стандартные подпрограммы обработки строк (функции семейства str...) описаны в библиотеке string.h.

Ниже описание этих функций приводится в виде:

<тип результата> <имя функции>(<формальные параметры>);

Предполагается, что существуют следующие описания:

```
char  c, ch;
char  *string1, *string2;
int  m;
unsigned  n;
```

Функции манипулирования со строками позволяют решать различные типы задач:

-связывание,
-изменение,
-сравнение,
-преобразование,
-копирование,
-поиск.

Ниже дается описание некоторых функций семейства str..., разделенных по типам решаемых задач.

5.2.7.1 Связывание(конкатенация)

strcat char *strcat(string1, string2);

Добавляет копию строки string2 в конец строки string1.

strncat char *strncat(string1, string2, m);

Добавляет копию m символов строки string2 в конец строки string1, и затем добавляет нулевой символ.

Обе функции возвращают ссылку на результат, то есть на строку string1. Эти функции не проверяют, умещается ли вторая строка в первой. Если Вы ошиблись при выделении памяти для первого массива, у Вас возникнут проблемы.

5.2.7.2 Сравнение

strcmp int strcmp(string1, string2);

Сравнивает string1 и string2

stricmp int stricmp(string1, string2);

Сравнивает string1 и string2, не делая отличия между прописными и строчными буквами.

strncmp int strncmp(string1, string2, m);

Производит такое же сравнение, как и stricmp, но просматривает не более, чем m символов.

strnicmp int strnicmp (string1, string2, n);

Сравнивает string1 и string2 (но не более, чем n байтов), не делая отличия между прописными и строчными буквами.

Все функции сравнения возвращают целое значение, меньшее, равное или большее нуля, в зависимости от результата лексикографического сравнения string1 со string2.

5.2.7.3 Копирование

strcpy char *strcpy(string1, string2);

Копирует string2 в строку string1 и останавливается после пересылки нулевого символа.

strncpy char *strncpy(string1, string2, m);

Копирует точно m символов из string2 в string1, усекая или заполняя нулевыми знаками string1, если это необходимо. Если длина строки string2 больше или равна m, результирующая строка не заканчивается нулевым знаком.

Обе функции возвращают ссылку на результат, то есть string1. Функции не проверяют, умещается ли вторая строка в первой.

5.2.7.4 Поиск

strlen int strlen(string1);

Возвращает число знаков в строке string1; нулевой знак, завершающий строку, не включается в это число.

strchr char *strchr(string1, c);

strrchr char *strrchr(string1, c);

Функция strchr(strrchr) возвращает указатель на первое (последнее) вхождение символа "c" в строку string1; если знак "c" не обнаружен в string1, то функция возвращает NULL. Нулевой знак, завершающий строку, является частью строки.

strpbrk char *strpbrk (string1, string2);

Функция возвращает указатель на первый встретившийся в строке string1 знак строки string2 или NULL, если в строке string1 не встретилось ни одного знака строки string2.

strspn int strspn(string1, string2);

strcspn int strcspn(string1, string2);

Функция strspn(strcspn) возвращает значение длины начального сегмента строки string1, целиком состоящего из знаков строки string2 (целиком состоящего из знаков, отсутствующих в строке string2).

strtok char *strtok(string1, string2);

Функция strtok рассматривает строку string1 как последовательность текстовых элементов (их число должно быть больше либо равно нулю), разделенных одним или более знаками из строки string2, выступающей в роли источника разделителей. Первый вызов функции (с заданным значением указателя string1 в качестве первого аргумента) возвращает указатель на первый знак первого элемента и записывает нулевой знак в строку string1 сразу после этого элемента. Функция хранит информацию, необходимую для последующих вызовов (которые должны выполняться со значением первого аргумента NULL), обеспечивающих обработку строки string1 со знака, непосредственно следующего за элементом строки, обработанным во время предыдущего вызова функции. Строка-разделитель string2 меняется от вызова к вызову. После того, как исчерпаны все элементы строки string1, функция возвращает значение указателя NULL.

5.2.7.5. Изменение

strlwr char *strlwr(string1);

Преобразует буквы верхнего регистра строки string1 в буквы нижнего регистра.

strupr char *strupr(string1);

Преобразует буквы нижнего регистра строки string1 в буквы верхнего регистра.

strset char *strset(string1, ch);

Устанавливает все символы строки string1 в символы ch.

strnset char *strnset(string1, ch, n);

Заменяет первые n байт строки string1 на символы ch. Если n>strlen(string1), то strlen(string1) принимает значение n.

5.2.8 Примеры использования строковых стандартных функций

Пример 1.

```
/* strlen()  - определяет длину строки;
   strcpy() - копирует вторую строку в первую;
   strcat()  - добавляет копию второй строки в конец первой;
   puts()  - выводит строку на экран и завершает вывод символом '\n'. */
#include <stdio.h>
#include <string.h>
#define s 19
int main()
{ char *msg;
     msg="Вы изучили язык Си хорошо";   /* Можно было выделить область в msg и затем:
                    strcpy (msg, "Вы изучили язык Си хорошо");  */
     puts(msg);
     if (strlen(msg)>s)  *(msg+s)='\0';  puts(msg);
     strcat(msg, "плохо   ");              puts(msg);
}
/*  Программа выдает:
                Вы изучили язык Си хорошо
                Вы изучили язык Си
                Вы изучили язык Си плохо
```

При выполнении оператора if, если длина строки msg больше 19, то в 20-ый символ (вместо буквы 'x') помещается символ '\0'. Остаток массива остается на прежнем месте, но последующая функция puts(msg) прекращает вывод при встрече первого нулевого символа и игнорирует остаток массива. */

/* При объединении двух строк с помощью функции strcat() не проверяется размер первой строки, что, в принципе, может привести к ошибке, так как strcat() не проверяет первую строку на переполнение. */

Пример 2.

```
/* Объединение двух строк с помощью функции strcat().
   Предварительно проверяется размер первой строки с помощью функции strlen(). */
#include<stdio.h>
#include<string.h>
#define size 30
int main()
{char  s1[size], *s2;
  s2=" - прекрасный язык!";
  puts("Ваш любимый язык программирования?");
  gets(s1);    /* Ввод строки s1 */
  if ( strlen(s1)+strlen(s2)+1<size )
```

```
        strcat(s1,s2);
  puts(s1);
}
/* Ввод - вывод программы:
        Вывод:          Ваш любимый язык программирования?
        Ввод:  СИ
        Вывод СИ - прекрасный язык!                    */
/* При проверке условия в операторе  if  цифра  "1"  добавлена к объединенной длине для размещения нулевого символа  */
```

Пример 3.

```
/* Функция  strcmp()  сравнивает две строки.
   Функция  strcmp()  использует два указателя строк в качестве аргументов и возвращает значение  0 , если эти две строки одинаковы. Точнее: функция передвигается вдоль строк до тех пор, пока не находит первую пару несовпадающих символов; затем она возвращает разницу в кодах ASCII  */
#include<stdio.h>
#include<string.h>
int main()
{printf("%d\n",strcmp("A","A"));      /* Вывод:   0  */
 printf("%d\n",strcmp("A","B"));      /* Вывод: -1  */
 printf("%d\n",strcmp("B","A"));      /* Вывод:   1  */
 printf("%d\n",strcmp("C","A"));      /* Вывод:   2  */
 printf("%d\n",strcmp("apples","apple")); /* Вывод:  115. Первая  пара  несовпадающих символов: 's' (код 115) и '\0' (код 0),- поэтому вывод: 115-0=115*/
}
```

Пример 4.

```
/* Функция  strcmp()  сравнивает две строки */
#include<stdio.h>
#include<string.h>
#define answer "Ритчи"
int main()
{char try[40];
  puts("Кто разработал язык СИ?");
  gets(try);
  while(strcmp(try,answer)!=0);
        {
         puts("Неверно. Попытайтесь еще раз.");
         gets(try);
        }
  puts("Правильно !");
}
/*  Так как ненулевые значения интерпретируются всегда как истина, можно сократить в программе запись условия в операторе  while  до   "while(strcmp(try,answer))...  ".
   Запись вида                  while(try!=answer)...
вместо                      while(strcmp(try,answer))...
приведет к ошибке, так как  try  и  answer  - указатели, и сравнение
                          "try != answer"
является сравнением не самих строк, а их адресов;  try  и  answer  запоминаются в разных ячейках и, следовательно, программа в этом случае "зацикливается", постоянно сообщая о том, что ответ неверен. */
```

/* Хорошо, что strcmp() сравнивает строки, а не массивы. Поэтому, хотя массив try занимает 40 ячеек памяти, а "Ритчи" - только 6 (не забывайте об одной ячейке, занятой нулевым символом), сравнение выполняется только с частью try до первого нулевого символа. */

Пример 5.

```
/*  Функция strtok().  Пример применения ее при разборе даты.
        Дата может быть представлена в различных форматах:
                12/3/1991
                12-March-91
                March 12,1991 и т.д.
        Определим строку ограничителей следующим образом: ". ,-/" , то есть: точка,
        пробел, запятая, тире, слэш.  */
#include<stdio.h>
#include<string.h>
int main()
{char *ptr;
  ptr=strtok("March 14, 1991",". ,-/");
  printf("ptr=%s\n", ptr);
  ptr=strtok(NULL,". ,- /");
  printf("ptr=%s\n", ptr);
}
/*  Вывод программы:
        ptr=March
        ptr=14          */
```

5.2.9 Массивы указателей. Массивы символьных строк

Пример 1. Фрагмент программы, предназначенной для выдачи пользователю различных сообщений об ошибках.

```
    ⋮
  char *error[30];     /*error  - массив из 30 ссылок на char */
    ⋮
  error[0] ="not enough arguments";
  error[1] ="too enough arguments";
  error[2] ="invalid argument";
    ⋮
  printf("***%s***\n",error[i]);      /*выдача сообщения об i-ой ошибке */
```

Пример 2. Можно явно задать размер строк символов, используя, например, следующее описание: char error[30][25];

В данном случае с помощью второго индекса задан прямоугольный массив из 30 строк, в котором все строки имеют одинаковую длину 25 (24 «реальных» знака в строке).

Предыдущее же описание "char *error[30];" определяет "рваный" массив, в котором длина каждой строки определяется длиной присваиваемой строки. "Рваный" массив не тратит память напрасно.

Пример 3. char (*lptr[26])[7][5];

Здесь lptr - это массив из 26 элементов. Каждый элемент - это ссылка, указывающая на массив символов. Размеры этого массива символов - (7*5).

5.3 МНОГОМЕРНЫЕ МАССИВЫ И УКАЗАТЕЛИ

5.3.1 Одномерные массивы и указатели

Выше в пунктах 5.1 и 5.2 рассматривали связь одномерных массивов и указателей. Напомним: имя массива - это адрес памяти, начиная с которогорасположен нулевой элемент массива.

Пример.

```
int p[10], *ptr;
ptr=p;          }  Эти два оператора дают один и тот
ptr=&p[0];      }  же результат
p[5]=6;         }  Эти операторы дают один и тот же
*(p+5)=6;       }  результат: значение 6-го элемента
ptr[5]=6;       }  массива становится равным 6.
*(ptr+5)=6;
```

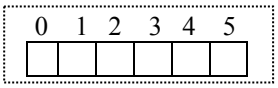
Рекомендации по использованию. При работе с подряд идущими элементами массива арифметические операции над указателями выполняются быстрее, чем над элементами с индексами. Если же элементы массива выбираются для обработки случайным образом, то быстрее и нагляднее работа с индексами.

5.3.2 Двумерные массивы и указатели

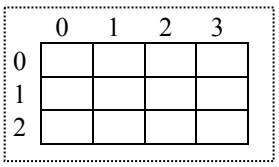
Массив занимает непрерывную область памяти и хранится в памяти по строкам. Двумерный массив - это одномерный массив, каждым элементом которого является строка.

Пусть имеем:

int a[6];



int d[3][4];



a - указатель на 0-ой элемент массива;
(a+1) - указатель на 1-ый элемент;
(a+2) - указатель на 2-ый элемент;
Выражение a[2] эквивалентно выражению *(a+2)

d - указатель на 0-ю строку массива;
(d+1) - указатель на 1-ю строку;
(d+2) - указатель на 2-ю строку;
Выражение d[2][3] эквивалентно выражению *(d[2]+3) или выражению *(*(d+2)+3)

Итак:

1) для двумерного массива d выражение d[row][col] переводится компилятором в эквивалентное выражение: *(*(d+row)+col) ;

2) для двумерного массива d выражение:

d+2 - это адрес 2-ой строки, текущая "единица измерения" - строка;
*(d+2) - это адрес 0-го элемента 2-ой строки, текущая "единица измерения" - элемент;
d[2] - это адрес 0-го элемента 2-ой строки, текущая "единица измерения" - элемент.

Пример. Двухмерные массивы и указатели.

```
int main()
{int i,j,n,x[2][3];
 n=1;
 for(i=0;i<2;i++)
   for(j=0;j<3;j++)     {x[i][j]=i+j+n;n++;}
for(i=0;i<2;i++) {for(j=0;j<3;j++) printf("%d",x[i][j]);
                  printf("\n");     /*  Получится массив:      1 3 5
                                                               5 7 9 */.
```

Предположим, что начальный адрес массива равен FFE8. Ниже приведены операторы вывода, которые распечатывают значение элемента массива или адрес элемента ; и даны пояснения по полученным результатам.

Операция	Значение	Пояснения

printf("%p",&x[0][0]);
printf("%p",x);
printf("%p",*x);
	FFE8
	FFE8
	FFE8
		Это адрес начала массива, т.е. нулевой строки, т.е. элемента x[0][0]

printf("%p",x+1);	FFEE	Это адрес первой строки, т.е. элемента x[1][0]
printf("%p",*x+1);	FFEA	Это адрес элемента x[0][1]
printf("%p",*(x+1));	FFEE	Это адрес нулевого элемента первой строки, т.е. x[1][0]
printf("%p",x[0]);	FFE8	x[0] - это адрес нулевой строки, т.е. элемента x[0][0]
printf("%p",x[1]);	FFEE	x[1] - это адрес первой строки, т.е. элемента x[1][0]
printf("%d",*(*x)); printf("%d",*x[0]);	1 1	Это значение элемента x[0][0]
printf("%d",*x[1]);	5	Это значение элемента x[1][0]
printf("%d",*(x[1]+1)); printf("%d",*(*(x+1)+1));	7 7	Это значение элемента x[1][1]

Объяснения.

Один адрес:
- x - адрес начала массива, т.е. нулевой строки
- x[0] - адрес нулевого элемента 0-ой строки, т.е. &x[0][0]
- *x - адрес нулевого элемента 0-ой строки, т.е. &x[0][0]

Один адрес:
- x+1 - адрес первой строки
- x[1] - адрес нулевого элемента 1-ой строки, т.е. &x[1][0]
- *(x+1) - адрес нулевого элемента 1-ой строки, т.е. &x[1][0]

*x+1 - это адрес первого элемента нулевой строки, т.е. адрес x[0][1]

Различие x и *x (или x[0]): для x адресная арифметика производится в единицах измерения "строка", то есть x+1 - это адрес первой строки; для *x адресная арифметика производится в единицах измерения "элемент", то есть *x+1 - это адрес первого элемента в нулевой строке.

Способ запоминания: если для двухмерного массива x в записи есть одна * - это адрес; две * - это значение элемента; одна * и одни [ ] - значение элемента; две [ ] - значение элемента; одна [ ] - адрес.

5.3.3 Многомерные массивы и указатели

Пусть x - двумерный массив. Тогда x[i] является ссылкой на i-ю строку массива x; x[i] - это адрес нулевого элемента этой i-ой строки, то есть *(x+i). Элементы каждой строки занимают непрерывную область памяти, так как массивы хранятся записанными по строкам (то есть при записи элементов массива в памяти быстрее всех изменяется последний индекс).

Пусть y - это n-мерный массив (n>1). Тогда y[i] - это адрес нулевого элемента (n-1)-мерного подмассива массива y, то есть *(y+i). Все элементы этого (n-1)-мерного массива занимают непрерывную область памяти.

Итак, указывая только первые p индексов, можно ссылаться на (k-p)-мерный подмассив k-мерного массива ($p \le k$).

5.4 ЛАБОРАТОРНАЯ РАБОТА № 5 "РАБОТА СО СТРОКАМИ"

Цель лабораторной работы "Работа со строками" - получить практический опыт написания программ по обработке текстовой информации и использования стандартных функций для обработки строк.

Задание к лабораторной работе: написать две программы по обработке строк. В программах следует использовать стандартные строковые функции. Варианты конкретных заданий выбрать из списка заданий в п.5.4.1. Примеры программ с использованием стандартных функций по работе со строками см. в п.5.2.8.

В большинстве заданий используется понятие "текст" и предполагается, что это строка. Фактически текст - это набор слов, разделенных пробелами и знаками препинания (при окончательной постановке задачи программист должен указать все допустимые для его программы знаки-разделители). Слова состоят из букв английского и/или русского алфавита и цифр (некоторые специальные случаи оговариваются отдельно). В некоторых заданиях указано, что различие прописных и строчных букв во внимание не принимать. Это значит, что, например, слова "Anna" и "anna" - одинаковы.

5.4.1 Обработка текстовой информации. Работа со строками

ЗАДАНИЕ. Выполнить на ЭВМ программу обработки символьных данных (текста) в соответствии с одним из указанных ниже вариантов.

N1.

Дан текст. Определить в нем наиболее часто встречающийся символ; затем определить все слова, в которых доля этого символа максимальна.

Пример. Текст: "Veni, scripsi, vixi" ("Пришел, написал, прожил"). Наиболее часто встречающийся символ - "i". Слова, в которых доля этого символа максимальна: "vixi" (0.5).

N2.

Дан текст. Определить в нем все слова, в которых доля заданного символа максимальна.

Пример. Текст: "Veni, vidi, vici" ("Пришел, увидел, победил"). Символ: "i".

Результат: слова "vidi", "vici", доля символа i равна 0.5.

N3.

Дан текст. Определить, является ли он "перевертышем".

Пример текста-перевертыша: "А роза упала на лапу Азора". Различие строчных и прописных букв во внимание не принимать, пробелы не учитывать.

N4.

Дан текст. Выделить из текста все слова-"перевертыши". Различие строчных и прописных букв во внимание не принимать.

Например: "top apple pot". Top — pot - слова "перевертыши".

N5.

Дан текст. Выделить из него все слова, которые не содержат одинаковых символов. Различие строчных и прописных букв во внимание не принимать.

Пример. Текст: "Няня Нину мылом мыла". Слова: "мыла".

N6.

Дан текст. Найти в нем все симметричные слова. Различие строчных и прописных букв во внимание не принимать.

Примеры симметричных слов: Anna, anpna.

N7.

Дан текст. Найти в нем слова, в которых некоторый заданный символ встречается наибольшее число раз.

Пример. Текст: "Anna and Dasha want to take this apple". Символ: "a".
Результат: слова "Anna", "Dasha" содержат символ "a" 2 раза.

N8.

Дан текст. Найти все слова, в которые заданный символ входит не менее 2 раз.

N9.

Дан текст (английский). Найти все слова, содержащие наибольшее количество гласных латинских букв. (a,e,i,o,u).

Пример. $\underbrace{\text{Anna}}_{2}$, $\underbrace{\text{Table}}_{2}$.

N10.

Дан текст. Найти все слова, которые содержат символ «t» и встречаются в тексте не менее 2 раз.

Пример. Текст: "to be or not to be". Слова: "to".

N11.

Дан текст. Найти такие слова, которые начинаются и кончаются одним символом.

N12.

Дан текст. Найти номер первого по порядку слова, которое начинается с заданного символа.

N13.

Дан текст. Найти все слова, которые оканчиваются тем же символом, что и первое слово.

N14.

Дан текст. Выделить из него все слова, не содержащие букв из последнего слова.

N15.

Дан текст. Определить в нем все слова, содержащие сочетание "ab", и заменить его на "ba".

N16.

Дан текст. Заменить все вхождения подстроки "del" на "Insert".

Пример. Исходный текст: "Нажмите клавишу del"
Полученный текст: "Нажмите клавишу Insert".

N17.

Дан текст. В тех словах, которые оканчиваются сочетанием букв "ing", заменить это окончание на "ed".

N18.

Дан текст. Во всех словах заменить буквосочетание "abc" на "def".

N19.

Дан текст. Для каждого слова указать, сколько раз оно встречается в тексте. Различие строчных и прописных букв во внимание не принимать.

N20.

Дан текст. Напечатать все различные слова. Различие строчных и прописных букв во внимание не принимать.

N21.

Дан текст, состоящий из слов, разделенных пробелами. Слова - английские. Напечатать все слова в алфавитном порядке. Одинаковые слова печатать один раз.

N22.

Дан текст. Удвоить каждое вхождение определенного символа ch. Предусмотреть повторение этой процедуры несколько раз для разных значений ch.

Пример. Исходная строка: "Талин - красивый город".
Результат: ch= "л" $\rightarrow$ Таллин - красивый город;
ch= "н" $\rightarrow$ Таллинн -красивый город.

N23.

Из заданного текста выбрать и распечатать только те символы, которые встречаются в тексте только один раз (в том порядке, в котором они встречаются в тексте). Затем распечатать слова, которые состоят только из этих символов.

N24.

Дан текст. Распечатать те слова, в которых буквы упорядочены по алфавиту. Различие строчных и прописных букв во внимание не принимать.

Пример. Исходный текст: "Bell is my friend". Слова: Bell, is, my.

N25.

Дан текст. Найти в нем самое короткое и самое длинное слово.

N26.

Дан текст. Напечатать все слова, которые не содержат определенных символов $c_1, c_2, \ldots c_n$. Эти символы вводятся с клавиатуры.

Пример. Исходный текст: "Карфаген должен быть разрушен". Символы: "о", "б".
Получим слова: Карфаген, разрушен.

N27.

Дан текст. После каждого символа ch вставить строку str. Значение ch и str вводятся с клавиатуры.

N28.

Дан текст. Для каждого слова, длина которого - нечетное число, удалить среднюю букву.

Пример. Исходный текст: "Наша кошка пьет молоко".
Полученный текст: "Наша кока пьет молоко".

N29.

Дан текст. Каждое слово преобразовать следующим образом: оставить в слове только первые вхождения каждой буквы. Различие строчных и прописных букв во внимание не принимать.

Пример. Исходная строка: "Никто не обнимет необъятного".

Полученная строка: "Никто не обнимет необъятг".

N30.

Дан текст. Каждое слово преобразовать следующим образом: удалить из слова все предыдущие вхождения последней буквы. Различие строчных и прописных букв во внимание не принимать.

Пример. Исходный текст: "На седьмом небе".
Полученный текст: "На седьом нбе".

N31.

Дан текст. Исключить из него все символы, расположенные между круглыми скобками. Сами скобки также исключить.

N32.

Дан текст. Проверить, имеется ли в нем баланс операторных скобок "begin ...end".

N33.

Дан текст. Проверить, имеется ли в нем баланс открывающих и закрывающих круглых скобок.

N34.

Дан текст. Удалить из каждой группы идущих подряд цифр, в которой более двух цифр и которой предшествует точка без пробела, все цифры, начиная с третьей.

Пример. Исходный текст: "Рассмотрим выражение ab+0.1973-1.1".
Преобразованный текст: "Рассмотрим выражение ab+0.19-1.1".

N35.

Дан текст. Найти сумму всех чисел текста.

Пример. Исходный текст: " Если к 12 прибавить 2, то получится 14".
Результат: 28.
Пояснение: 12+2+14=28.

N36.

Дан текст. Подсчитать сумму всех цифр, входящих в текст.

Пример. Исходный текст: "Если к 12 прибавить 2, то получим 14".
Результат: 10.
Пояснение: 1+2+2+1+4=10.

N37.

Дан текст, состоящий из нескольких предложений. Предложения оканчиваются точкой, восклицательным или вопросительным знаками. Сколько раз в каждом предложении встречается какой-то заданный символ?

N38.

Дан текст, состоящий из нескольких предложений. Предложения оканчиваются точкой, восклицательным или вопросительным знаками. Из каждого предложения удалить средний символ, если длина предложения нечетна, и два средних символа, если длина предложения четна.

N39.

Дан текст, состоящий из нескольких предложений. Предложения оканчиваются точкой, восклицательным или вопросительным знаками. Подсчитать количество слов и вывести только те из них, которые начинаются с буквы "a" (для слов, начинающих предложения, с буквы "A").

N40.

Дан текст. В каждом предложении символ "пробел" заменить на символ ",". Если несколько символов "пробел" встречаются подряд, то заменить их на одну запятую. Пробелы, отделяющие друг от друга предложения (каждое предложение кончается точкой), на запятую не менять.

N41.

Дан текст (английский). Первую букву первого слова каждого предложения заменить на прописную ("маленькую") букву.

5.5 КОНТРОЛЬНЫЕ ВОПРОСЫ ПО МОДУЛЮ 5

5.5.1 Связь указателей и массивов в Си.
5.5.2 Связь указателей, массивов и строк.
5.5.3 Определение, ввод, обработка и вывод строк.
5.5.4 Стандартные функции для обработки строк.
5.5.5 Двумерные массивы и указатели.

6 ФУНКЦИИ

Цель изучения данного модуля - познакомиться с основными правилами оформления функций (подпрограмм) в Си, а также приобрести практический опыт работы с подпрограммами.

6.1 ОПРЕДЕЛЕНИЕ И ОПИСАНИЕ ФУНКЦИИ

В Си все подпрограммы рассматриваются как функции, которые делятся на две группы:

1) функции, возвращающие значение в вызывающую программу;

2) функции, не возвращающие значение в вызывающую программу (аналогично процедуре в Паскале).

Функцию можно описывать и определять. Описание функции дает всем остальным программам (включая главный модуль main) информацию о том, каким образом должно осуществляться обращение к этой функции. При определении функции указывается, какие конкретно действия она выполняет.

6.1.1 Определение функции

Определение функции имеет вид:

```
[static]  [<тип результата>]  <имя функции>( [<формальные параметры>] )
  [ < описания формальных параметров > ]
  {
    < тело функции >
  }
```

Ключевое слово static (статический) - это класс памяти. Классы памяти подробно изучаются в 7 разделе. Функция с классом памяти static доступна только внутри файла, содержащего ее. Если ключевое слово static опущено, то по умолчанию функция имеет класс памяти extern (внешний). Внешние функции доступны из любых файлов.

Если опущен тип результата, то по умолчанию предполагается int. Для ясности рекомендуется тип результата всегда указывать явно. Для функций, не возвращающих значения, в качестве типа результата должен быть указан тип void (пустой). В качестве результата функция не может возвращать массив или функцию, но может возвращать указатель на массив или функцию.

Описания формальных параметров аналогичны описаниям обычных переменных, за исключением того, что в таких описаниях явно можно указать только один класс памяти register. В описании формальных параметров-массивов может быть опущено максимальное значение первого индекса массива.

Пример допустимых описаний параметров-массивов:

```
int  a[ ], b[4], d[ ][4], e[2][4];
```

Пример неверного описания параметра-массива (не указано максимальное значение второго индекса):

```
int c[ ][ ];
```

Выполнение функции, возвращающей значение, обязано завершаться оператором return (возврат) вида:

```
return < выражение >;
```

При выполнении этого оператора работа функции прекращается, значение входящего в оператор return выражения становится результатом функции. Подобных операторов return в теле функции может быть несколько.

Выполнение функции, не возвращающей значения, завершается, если выполнено ее тело или выполнен оператор return вида:

```
return;
```

Функция может содержать несколько таких операторов.

Пример 1. Определение функции, находящей максимум из двух своих параметров.

```
int max(a,b)      /* В конце заголовка функции нет знака ";"  */
int a,b;          /* Формальные параметры функции описываются до знака "{",  */
{                 /*  определяющего начало тела функции   */
  return (a>b) ? a: b;
}
```

При определении функции можно использовать два стиля программирования: классический и современный (современный стиль применим в конструкции расширенной версии СИ, предложенной ANSI).

Классический формат определения функции дан выше.

При использовании современного стиля в круглых скобках после имени функции записываются не только имена формальных параметров, но сразу же указывается их тип. В этом случае отдельное описание формальных параметров не требуется.

Пример 2. Ниже дано определение той же функции, что и в примере 1, но в современном стиле.

```
int max(int a, int b)
{
  return (a>b) ? a:b;
}
```

6.1.2 Описание функции

Функция может быть вызвана или передана как параметр до того, как ее определение встретится в тексте программы при условии, что до обращения к ней в тексте программы встретилось ее описание.

Описание функции необходимо для генерации кода и ограниченной проверки ошибок.

Описание функции имеет вид:

1) функция, возвращающая результат:

```
[static/extern]  <тип результата>  <имя функции>();
```

2) функция, не возвращающая результат:

```
[static/extern]  void  <имя функции>();
```

Если в описании не указан класс памяти, то по умолчанию предполагается extern.

Заметьте, что после описания функции ставится точка с запятой, а после заголовка в определении функции - нет.

Примеры описания функций:

```
int max();
extern char *strcpy();
char *mal();
```

При описании функций, как и при определении, может использоваться два стиля программирования.

Выше дано описание функций в соответствии с классическим стилем.

При описании функций в современном стиле используются специальные средства языка, известные под названием "прототип функции". Описание функции с использованием прототипа содержит дополнительно информацию о ее параметрах. Эта информация записывается в круглых скобках после имени функции и для каждого параметра имеет один из следующих форматов:

1) < тип >

2) < тип > < имя параметра >

3) ...

Другими словами, при использовании прототипа функции должен быть описан тип каждого формального параметра и, возможно, указано его имя. Если функция использует переменный список параметров, то после указания последнего фиксированного параметра функции в описании необходимо использовать эллипсис - многоточие ... .

При описании функций классическим стилем отсутствует контроль над ошибками при использовании параметров, поэтому предпочтительнее описывать функции с помощью прототипов. В этом случае компилятор производит проверку на соответствие количества и типа параметров при каждом обращении к функции, а также выполняет преобразование аргументов к требуемому типу.

Пример. Рассмотрим фрагмент программы:

```
 long  lmax(long v1,long v2);    /* Описание функции lmax с помощью прототипа */
void  main(void) /*Главная программа, в которой есть вызов lmax */
 { int limit=32;
   char ch='A';
   long mval;
    mval=lmax(limit,ch);
 }
```

Используя прототип функции lmax, программа будет преобразовывать параметры limit и ch к типу long прежде, чем они будут помещены в стек для обращения к lmax.

При отсутствии прототипа функции параметры limit и ch были бы помещены в стек соответственно как целое значение и символ. В этом случае в lmax передавались бы параметры, не совпадающие по размеру и содержанию с ожидаемыми, что привело бы к возникновению проблем.

Замечания по оформлению описания параметров в прототипе:

1) описания переменных отделяются друг от друга запятой, а не точкой с запятой;

2) нельзя задавать список нескольких имен для одного типа, то есть записать, например, "long v1,v2" нельзя.

Прототип функции с заключенным в скобки единственным словом void означает, что функция совсем не имеет параметров. Например:

```
 int f(void);   /* Функция возвращает значение типа  int; параметры не передаются */
```

Резюме.

1) Си-компилятор не производит проверку ошибок при вызове функции: он не проверяет передаваемые функции параметры, их типы и количество. С одной стороны, это дает некоторую свободу, так как можно вызывать функцию до того, как она будет определена, с другой стороны - это часто приводит к ошибкам. Для избежания возможных ошибок рекомендуется описывать функцию до ее использования.

2) Многие компиляторы поддерживает и классический, и современный стиль при описании и определении функций. Однако сейчас просматривается тенденция к использованию современного стиля, его и рекомендуется использовать.

6.2 УПРАВЛЕНИЕ ВИДИМОСТЬЮ ФУНКЦИЙ

В языке Си вложенные функции не допускаются. Следовательно, невозможно определить функции, которые были бы невидимы вне функции, их содержащей. Однако эту ситуацию можно исправить с помощью класса памяти static.

Пример. Пусть функции “a” и “b” определены в отдельном файле следующим образом:

```
static int a(...)
{ ...
}
int b(...)
```

```
{ ...
}
```

В этом случае функция “a” может быть вызвана из функции “b”, но никакая другая функция, физически размещенная в другом файле, не сможет вызвать функцию “a”.

6.3 ВЫЗОВ ФУНКЦИЙ

Вызов функции, возвращающей значение, обычно является частью оператора и имеет следующую форму:

<имя функции>(<список фактических параметров>);

где фактические параметры могут быть выражениями. Например:

```
a=max(a,5);
b=max(max(b,5),e)+c;
ch=getch();
fs=fopen(argv[1],"r");
```

Функции, не возвращающие значения, вызываются аналогично, но их вызовы преобразуются в следующие операторы:

<имя функции>(<список формальных параметров>);

Например:

```
delay(0.5);
partition(a,1,u,&i,&j);
```

Замечание. Описанное различие в вызове функций в Си достаточно условно. Во-первых, все функции в Си возвращают значение. Даже если реальное значение не возвращается, говорят, что возвращается результат типа void. Во-вторых, совершенно не обязательно обращаться по первому способу к функции, возвращающей значение. Например, к функции getch() можно обратиться в виде

```
c=getch();
```

если нужно результат ее работы записать в переменную c; и просто в виде

```
getch();
```

если результат в виде введенного знака не нужен.

6.4 ПЕРЕДАЧА ПАРАМЕТРОВ

Передача параметров в языках программирования обычно реализуется двумя способами:

- передача по значению (аналог в Паскале - параметры-значения);
- передача по ссылке (аналог в Паскале - параметры-переменные).

В языке Си параметры передаются только одним способом - по значению, то есть все параметры в Си - только входные. Это означает, что значения фактических параметров становятся начальными значениями формальных. Все последующие изменения формальных параметров никак не сказываются на фактических параметрах.

Пример 1.

```
/*  Программа  "сокращение дробей" :  4/6  →  2/3.    */
/*  Изменения "x" и "y" в функции  ff()  никак не влияют на значение "a" и "b" в функции
                    main() */
#include <stdio.h>
     int  ff(int x, int y);   /* Описание функции  ff  с помощью прототипа  */
  /* Основная программа "Сокращение дробей" */
int main()
    { int a,b;
```

```
        printf("\nВведите числитель и знаменатель: ");
        scanf("%d %d",&a,&b);
        x=ff(a,b); /* Вызов функции  ff  для нахождения наибольшего общего делителя x */
        printf("\n%d/%d=%d/%d",a,b,a/x,b/x);
    }
/* Определение функции ff "Нахождение наибольшего общего делителя - НОД" */
    int ff(int x, int y)
    { while (x!=y)
          if (x>y)
            x-=y;
          else
            y-=x;
        return x;
    }
```

Передача параметров по значению удобна, когда функция возвращает значение без изменения своих фактических параметров. На практике, однако, часто возникает необходимость в выходных параметрах, то есть надо, чтобы функция изменяла значения фактических параметров. Передача параметров по значению также неэффективна в том случае, если она требует копирования большого объема данных.

В языке Си реализация выходных параметров моделируется передачей в качестве параметра указателя на соответствующий объект, то есть передачей параметров по ссылке. То, что все параметры входные, в случае параметра-указателя означает, что нельзя изменить адрес, на который он указывает. Но это не запрещает изменить значение, хранящееся по этому адресу.

Для использования переменной как выходного параметра при вызове функции указывается не сама переменная, а ее адрес. Так, например, происходит при обращении к функции scanf(). При выполнении scanf() переменным присваиваются значения, то есть они являются выходными параметрами. Следовательно, при вызове функции необходимо указывать адреса переменных, поэтому мы и ставим перед именем знак &.

Пример 2. Функция swap переставляет значения двух своих параметров.

```
 #include <stdio.h>
      void swap(int *a, int *b); /* Описание функции swap с помощью прототипа */
      int main()
      { int i, j;
        i=421;
        j=53;
        printf("До обращения: i=%4d, j=%4d\n",i,j);
        swap(&i,&j);   /* Вызов функции swap */
        printf("После обращения: i=%4d, j=%4d\n",i,j ");
      }
    void swap(int *a,int *b)  /* Определение функции swap */
    { int temp;
       temp=(*a); *a=(*b); *b=temp;
    }
```

Функция swap объявляет два формальных параметра a и b, как указатели на тип данных int. Это означает, что функция swap работает непосредственно с адресами целых переменных. Поэтому при вызове функции swap в качестве фактических параметров указывались два адреса. Функция swap не может изменить эти два адреса, но может изменить значения, хранящиеся по этим адресам.

6.5 ПЕРЕДАЧА МАССИВОВ В КАЧЕСТВЕ ПАРАМЕТРОВ

Правило: в Си все параметры передаются по значению.

Исключение: имя массива - это указатель на массив, то есть адрес первого элемента массива, откуда следует, что передача массива в качестве параметра выполняется по ссылке.

Пример 1.

```
/* Передача в качестве параметра одномерного массива */
#include <stdio.h>
#define s 20    /*Максимальный размер массива */
void setrand(int list[], int size);       /* Описание функции setrand() с помощью прототипа */

int main()       /* Основная программа */
{int r,i.v[s];    /* v - массив, r - его размер */
 printf("Введите количество элементов массива "); scanf("%d",&r);
 setrand(v,r)   /* Вызов функции setrand() для ввода массива v */
 for (i=0; i<r; i++)
        printf ("v[%2d] = %6d\n",i,v[i]);
}

void setrand(int list[ ],int size)          /* Определение функции setrand() */
{                                          /* Функция вводит одномерный массив */
 int i;
 printf("Введите %d целых элементов: ",size);
 for(i=0;i<size; i++)   scanf("%d", &list[i]);
}
```

Замечание 1. Запись "int list[ ]" в списке формальных параметров функции setrand() - это просто фиксация типа формального параметра. При этом не происходит никакого выделения памяти под массив. Компилятор считает list[ ] начальным адресом массива и не заботится о том, где его конец, поэтому размер по первой размерности (у нас list - одномерный массив) можно не задавать.

То, что в функции setrand() передается начальный адрес массива "v" означает, что если произойдет какие-то изменения массива list в функции setrand(), то те же изменения будут произведены и в массиве v.

Замечание 2. При классическом стиле оформления подпрограмм описание функции setrand() должно выглядеть так:

```
        int setrand ();
```

а определение функции должно выглядеть так:

```
        void setrand(list,size)
        int list[],size;
```

Пример 2.

```
/* Передача в качестве параметра многомерного (двумерного) массива */
/* Пример2 - это модифицированный для двухмерного массива пример 1 */
#include <stdio.h>
#define s1 20
#define s2 10  /*Глобальная константа, определяющая второе измерение массива */
void setrand(int list[][s2], int rs); /* Описание функции setrand() с помощью прототипа */
int main()
{int  i,j,r,v[s1][s2];
 printf("Введите первое измерение массива: ");
 scanf ("%d", &r);
```

```
 setrand(v,r);
 for (i=0; i<r; i++)
        { for (j=0;j<s2, j++)
                printf("%6d", v[i][j]);
                printf("\n");
        }
}
void setrand(int list[][s2], int size)   /* Определение функции setrand() */
{int  i,j;
 for (i=0; i<size; i+j)
        for (j=0; j<s2; j++)
                scanf("%d", &list[i][j]);
}
```

Замечание 1. Размер по второй размерности в описании массива пропускать нельзя, так как иначе нельзя будет индексировать имя-ссылку.

Замечание 2. Если оставить предыдущее описание и определение функции setrand(), при котором заголовок выглядит как

setrand (int list[], int size),

то при работе с двухмерным массивом v[r][s2] обращение к этой функции setrand() будет иметь вид

setrand(v,r*s2);

Функция setrand будет рассматривать массив v как одномерный массив, содержащий r*s2 элементов.

Организовать в подпрограмме (ПП) работу с матрицей, передаваемой в ПП в качестве параметра, можно двумя основными способами.

Способ 1. Этот способ рассмотрен выше в примере 2. В обращении к ПП указывается имя матрицы (то есть адрес начального элемента). В определении ПП соответствующий параметр описан как матрица и с ним работают как с матрицей.

Способ 2. В обращении к ПП указывается имя матрицы (то есть адрес начального элемента). В определении ПП соответствующий параметр описан как указатель. Тогда и в тексте ПП с этим параметром надо работать как с указателем, используя адресную арифметику.

Способ 1 - нагляднее и понятнее, способ 2 - более гибкий. Во избежание ошибок лучше эти способы не комбинировать.

Пример 3. Оформить в виде 2-х отдельных функций:

1) ввод матрицы A(n*m);

2) определение значения максимального элемента матрицы.

Все результаты передавать через список параметров и распечатывать в главной программе. Решим эту задачу, используя вначале способ 1, а потом - способ 2.

Способ 1.

```
#include <stdio.h>
/* Функция ввода матрицы  A(n*m)  */
void in(int n,int m,int A[][20])
{int i,j;
  for (i=0;i<n;i++)
    for (j=0;j<m;j++)
      scanf("%d",&A[i][j]);
}
/*Функция определения значения наибольшего элемента матрицы A(n*m)*/
void maxin(int n,int m,int A[][20],int *max)
{int i,j;
  *max=A[0][0];
```

```
  for (i=0;i<n;i++)
    for (j=0;j<m;j++)
      if (*max<A[i][j]) *max=A[i][j];
}
/* Главная программа */
int main()
{int n, m, A[20][20], maxel;
  printf('\nВведите размерность матрицы A(n*m): ");
  scanf("%d%d",&n,&m);
  printf('\nВведите матрицу построчно:\n");
  in(n,m,A);    /* При вызове пишем "A" – имя двумерного массива без индексов.
                   То есть указываем адрес этого массива как двойной указатель. */
  maxin(n,m,A,&maxel);
  printf('\nЗначение максимального элемента: %d",maxel);
}
```

Способ 2.

```
#include <stdio.h>
/* Функция ввода матрицы  A(n*m)  */
void in(int n,int m,int *a)
{int i,j;
  for (i=0;i<n;i++)
    for (j=0;j<m;j++)
      scanf("%d",a++); /* При выходе из функции в главной программе указатель a
останется прежним, он будет указывать на начало массива */
}
/*Функция определения значения наибольшего элемента матрицы A(n*m)*/
void maxin(int n,int m,int *a,int *max)
{int i,j;
  *max=*a;
  for (i=0;i<n;i++)
    for (j=0;j<m;j++)
      if (*(a+i*m+j)>*max) *max=*(a+i*m+j);
}
/* Главная программа */
int main()
{int n, m, A[20][20], maxel;
  printf('\nВведите размерность матрицы A(n*m): ");
  scanf("%d%d",&n,&m);
  printf('\nВведите матрицу построчно:\n");
  in(n,m,&A[0][0]);    /* Здесь при вызове надо писать "&A[0][0]". Можно это же записать
        как: "*A" или "(int *)A", то есть передаем адрес как простой указатель */
  maxin(n,m,&A[0][0],&maxel);
  printf('\nЗначение максимального элемента: %d",maxel);
}
```

Замечание 1. Обратите внимание, как в случае способа 2 в функции maxin() получают значение очередного элемента:

*(a+i*m+j)

Эта запись основана на том, что в памяти элементы располагаются подряд друг за другом построчно. Например, если ввод массива A(2*3) был организован по способу 2, то в памяти элементы располагаются следующим образом: a[0][0] -> по адресу a+0,

a[0][1] -> по адресу a+1,
a[0][2] -> по адресу a+2,
a[1][0] -> по адресу a+3,
a[1][1] -> по адресу a+4,

a[1][2] -> по адресу a+5.

Замечание 2. Распространенная ошибка: ввод массива по способу 1 (при этом память под массив размером (2*3) заполнена "с дырами", так как в описании формального параметра-массива второй размер равен 20), а дальнейшая обработка, например, нахождения максимального, проводится по способу 2, который предполагает непрерывное заполнение памяти элементами массива. Для того, чтобы избежать ошибки в этом случае, в функции maxin() для получения значения очередного элемента надо писать: *(a+i*20+j), а не *(a+i*m+j).

6.6 УКАЗАТЕЛЬ НА ФУНКЦИЮ. ПЕРЕДАЧА ФУНКЦИЙ В КАЧЕСТВЕ ПАРАМЕТРОВ

6.6.1 В Си можно создать указатель на функцию. Например, так выглядит объявление указателя на функцию, возвращающую значение типа int и имеющую 2 параметра типов type1, type2:

```
int (*ptrpp)(type1 t1,type2 t2);
```

Указатель на функцию содержит адрес первого байта или слова выполняемого кода функции. Над указателями на функции запрещены арифметические операции.

Пример 1.

```
/*       Указатель на функцию       */
#include <stdio.h>
#include <math.h>
double f(double x);    /* Прототип функции f */
int main()
{ double (*ptrpp)(double x); /* ptrpp - указатель на функцию,
                                возвращающую значение типа double и
                                имеющую один параметр типа double */
  double z=1,y,y1;
  ptrpp=f;             /* Инициализация указателя */
  y=(*ptrpp)(z); /* Вызов функции через указатель - 1-ый способ */
  y1=ptrpp(z);   /* Вызов функции через указатель - 2-ой способ (используется реже)*/
  printf("z=%f, y=%f, y1=%f\n",z,y,y1);
  ptrpp=sin;           /* В качестве функции можно использовать
                                стандартные библиотечные функции  */
  y=(*ptrpp)(z); /* Вызов функции через указатель - 1-ый способ */
  y1=ptrpp(z);   /* Вызов функции через указатель - 2-ой способ */
  printf("z=%f, y=%f, y1=%f\n",z,y,y1);
}
/* Определение функции f */
double f(double x)
{puts("Вы - в функции f"); return x+2; }
```

Замечание. Первый способ вызова функции через указатель используется чаще, чем второй, потому что при втором способе указатель ptrpp очень похож на имя функции, что может ввести в недоумение программиста, читающего тест программы. Первоначально в Си вообще существовал только первый способ.

6.6.2 Можно использовать массив указателей на функцию. Пусть, например, имеем:

```
int  f0(void);   /*  Прототип функции f0  */
int  f1(void);   /*  Прототип функции f1  */
int  f2(void);   /*  Прототип функции f2  */
int  (*arrptrf[])(void)={f0,f1,f2};   /*  arrptrf - массив из трех указателей на функцию.
```

```
                                        Массив проинициализирован                    */
```
Соответствующую функцию вызываем так:

```
        v=(*arrptrf[i])();
```

или

```
        v=arrptrf[i]();
```

6.6.3 Функции могут передаваться в качестве параметров. Как и в случае массивов, фактически передаваемое значение является адресом функции.

Пример 2.

```
/* Передача функции в качестве параметра другой функции */
/* Примитивный калькулятор */
#include <stdio.h>
int sum(int a,int b); /*Прототип функции для сложения двух целых чисел */
int raz(int a,int b);  /*Прототип функции для вычитания двух целых чисел */
int calcul(int a,int b,int (*fp)(int a,int b)); /*Прототип функции "калькулятор".
                        Здесь fp - указатель на функцию, возвращающую целое */

int main()       /* Основная программа */
{int a,b,d;          /* a,b - исходные числа, d - результат */
  printf("\nВведите два целых числа: ");
  scanf("%d%d",&a,&b);
   d=calcul(a,b,sum); /* Функция sum передается в качестве параметра
                         функции calcul*/
   printf("%d + %d = %d\n",a,b,d);
   d=calcul(a,b,raz); /* Функция raz передается в качестве параметра
                         функции calcul*/
   printf("%d - %d = %d\n",a,b,d);
}
/* Определение функции calcul */
  int calcul(int a,int b,int (*fp)(int a,int b))
  {int rez;
   rez=(*fp)(a,b);   /* Можно писать : rez=fp(a,b); */
   return rez;
  }
/* Определение функции sum */
  int sum(int a,int b)
  {return a+b;}
/* Определение функции raz */
  int raz(int a,int b)
  {return a-b;}
```

Пример 3.

```
/* Передача функции в качестве параметра другой функции */
/* Вычисление по разным формулам и для разных аргументов */
/* Основное назначение этого примера - показать работу с функцией func(), одним из
параметров которой является имя какой-то другой функции. */
#include <stdio.h>
#include <conio.h>
int f1(int x); /*Прототип функции  */
int f2(int x); /*Прототип функции  */
int func(int x,int (*fp)(int x)); /*Прототип функции.
  Здесь fp - указатель на функцию, возвращающую целое */

int main()          /* Основная программа */
```

```
{int n,x,y;          /* x - исходное число, y - результат */
  do {
  printf("\nВведите аргумент (целое число) и номер формулы: ");
  scanf("%d%d",&x,&n);
   if (n==1) y=func(x,f1); else y=func(x,f2);
   printf("x=%d, y=%d, n=%d\n",x,y,n);
   printf("Будете еще работать (да - 'y')? ");
     } while (getch()=='y') ;
}
/* Определение функции func() */
  int func(int x,int (*fp)(int x))
  {int rez;
   rez=(*fp)(x);   /* Можно писать : rez=fp(x); */
   return rez;
  }
/* Определение функции f1() */
  int f1(int x)
  {return x+1;}
/* Определение функции f2() */
  int f2(int x)
  {return x+2;}
```

6.7 СВЯЗЬ ФУНКЦИЙ ИЗ РАЗНЫХ ФАЙЛОВ

Для связи функций из разных файлов можно использовать директиву препроцессора include.

Пример. Пусть в текущем каталоге имеем два файла: max1 и mainmax1.

В файле max1 содержится определение функции max1() в виде:

```
int max1(int a,int b)
{return (a>b)?a:b; }
```

В файле mainmax1 содержится текст основной программы, которая использует функцию max1(), в виде:

```
#include <stdio.h>
#include "max1.c"
int main()
{int a,b,c;
 printf("\nВведите два целых числа: ");
 scanf("%d%d",&a,&b);
 c=max1(a,b);
 printf("c=%d",c);
}
```

6.8 ЛОКАЛЬНЫЕ И ГЛОБАЛЬНЫЕ ДАННЫЕ

Переменные, константы и типы данных, описанные в теле функции, называются локальными (внутренними). Они могут использоваться только в той функции, где описаны. В разных функциях могут оказаться локальные переменные с одинаковыми именами. Это разные переменные, никак не связанные между собой.

Параметры функции также локальны.

Переменные, константы и типы, описанные вне функций, включая main, считаются глобальными (внешними) ниже точки описания. Их можно использовать во всех функциях, определенных после этих описаний.

Обычно глобальные данные используются при программировании сложных задач, когда программа состоит из большого числа функций. С помощью глобальных переменных описывают основные объекты такой программы.

Лексической областью действия идентификатора называется область программы, в которой его описание или определение имеет силу.

Лексическая область действия:

- внешнего идентификатора - располагается от его описания или определения до конца файл, его содержащего;
- формального параметра или функции - сама эта функция;
- идентификатора, определенного в блоке - этот блок;
- метки - функция, в которой она описана.

Локальное определение имеет больший приоритет, чем глобальное.

6.9 ЛАБОРАТОРНАЯ РАБОТА №6 "ПОДПРОГРАММЫ В СИ"

Подпрограммы, или функции, представляют собой важный инструмент языка Си. Они позволяют писать хорошо структурированные модульные программы.

Цель лабораторной работы "Подпрограммы в Си" - научиться разбивать программу на отдельные модули и работать на языке Си с подпрограммами.

Задание к лабораторной работе включает в себя три следующие задачи.

1. Организация функций - см. задание в п.6.9.1.1.

2. Организация функций и передача функции в качестве параметра - см. задание в п.6.9.1.2.

3. Функции; передача массивов в качестве параметров и организация выходных параметров - см. задание в п.6.9.2.1.

Задание 1 и 2 следует выполнять строго последовательно: вначале задание 1, потом задание 2.

6.9.1 Организация функций

6.9.1.1 Задание 1

1. Написать функцию 1, которая для любого заданного X вычисляет с заданной точностью E значение У как сумму членов бесконечного ряда. Вычисления провести по формуле 1, которую следует выбрать из списка заданий в п.6.9.1.3.

2. Написать функцию 2, которая для любого заданного X вычисляет с заданной точностью E значение Y как сумму членов бесконечного ряда. Вычисления провести по формуле 2, которую следует выбрать из списка заданий в п.6.9.1.3.

3. Написать функцию 3, которая распечатывает три заданных величины: X, У, E.

4. Написать основную программу main, которая

а) в режиме диалога вводит исходные данные X, E и N - номер функции, по которой следует вычислить Y;

б) в зависимости от указанного номера формулы N вызывает функцию 1 или функцию 2;

в) вызывает функцию 3 для печати результатов;

г) выполняет действия, описанные в пунктах а), б), в), столько раз, сколько потребует пользователь.

Замечание 1. Функции 1 и 2 должны возвращать результат через оператор return.

Замечание 2. При выполнении большинства заданий рекомендуется указывать значение X из диапазона [0,1] или [-1,1], так как на этих интервалах сходятся функции практически для всех заданий.

Замечание 3. При тестировании рекомендуется первоначально указать достаточно "большое" E (например, E = 0,1), а затем, при получении решения за удовлетворительное время, постепенно уменьшать точность E .

6.9.1.2 Задание 2

Программу, написанную при выполнении задания 1, следует изменить так, чтобы в зависимости от номера варианта N вызывались не просто две разные функции, а вызывалась одна функция 4 "Вычисление Y для заданного X с заданной точностью E". Этой функции в качестве одного из параметров следует передать имя функции 1 или имя функции 2 (в зависимости от номера N). Образец написания подобной программы см. в п.6.6.3 в примере 3.

6.9.1.3 Вычисление суммы ряда с заданной точностью

ЗАДАНИЕ. Написать функцию, которая для заданного аргумента x и заданной точности e вычисляет значение y=f(x) как сумму членов бесконечного ряда.

ПОЯСНЕНИЕ. Сумма вычисляется с точностью до члена ряда, меньшего e, то есть суммирование следует закончить, как только будет получено очередное слагаемое, по модулю не превышающее e.

N1.

$$Y(x)=\sqrt[3]{x}=1+\frac{1}{3}(x-1)-\frac{1\cdot 2}{3\cdot 6}(x-1)^2+\frac{1\cdot 2\cdot 5}{3\cdot 6\cdot 9}(x-1)^3-\frac{1\cdot 2\cdot 5\cdot 8}{3\cdot 6\cdot 9\cdot 12}(x-1)^4\ \ldots$$

N2.

$$Y(x)=\arccos ec\, x\quad=\frac{1}{x}+\frac{1}{2\cdot 3\cdot x^3}+\frac{1\cdot 3}{2\cdot 4\cdot 5\cdot x^5}+\frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6\cdot 7\cdot x^7}+\quad\ldots$$

N3.

$$Y(x)=\arcsin x\quad=x+\frac{x^3}{2\cdot 3}+\frac{1\cdot 3\cdot x^5}{2\cdot 4\cdot 5}+\frac{1\cdot 3\cdot 5\cdot x^7}{2\cdot 4\cdot 6\cdot 7}\quad\ldots$$

N4.

$$Y(x)=\mathrm{E}(x)=\frac{\pi}{2}\left(1-\frac{1}{2^2}x^2+\frac{1^2\cdot 3^2}{2^2\cdot 4^2}x^4-\frac{1^2\cdot 3^2\cdot 5^2}{2^2\cdot 4^2\cdot 6^2}x^6\quad\ldots\right)$$

N5.

$$Y(x)=\frac{\pi}{2}\left(1-\frac{1^2}{2^2}x^2+\frac{1^2\cdot 3^2}{2^2\cdot 4^2}x^4-\frac{1^2\cdot 3^2\cdot 5^2}{2^2\cdot 4^2\cdot 6^2}x^6\quad\ldots\right)$$

N6.

$$Y(x)=\Gamma_2(x)=\frac{x^2}{2^2\cdot 2!}-\frac{x^4}{2^4\cdot 1!\cdot 3!}+\frac{x^6}{2^6\cdot 2!\cdot 4!}-\frac{x^8}{2^8\cdot 3!\cdot 5!}\quad\ldots$$

N7.

$$Y(x) = erf(x) = \frac{2}{\sqrt{\pi}}\left(x - \frac{x^3}{3} + \frac{1}{2!}\cdot\frac{x^5}{5} - \frac{1}{3!}\cdot\frac{x^7}{7} \quad \ldots \right)$$

N8.

$$Y(x) = bei(x) = \frac{\left(x/2\right)^2}{(1!)^2} - \frac{\left(x/2\right)^6}{(3!)^2} + \frac{\left(x/2\right)^{10}}{(5!)^2} \quad \ldots$$

N9.

$$Y(x) = \Gamma(x) = 1 - \left(x/2\right)^2 + \frac{\left(x/2\right)^4}{1^2\cdot 2^2} - \frac{\left(x/2\right)^6}{1^2\cdot 2^2\cdot 3^2} + \frac{\left(x/2\right)^8}{1^2\cdot 2^2\cdot 3^2\cdot 4^2} \quad \ldots$$

N10.

$$Y(x) = F(x) = \ln|\ln x| = \ln x + \frac{(\ln x)^2}{2\cdot 2!} + \frac{(\ln x)^3}{3\cdot 3!} \quad \ldots$$

N11.

$$Y(x) = Si(x) = x - \frac{1}{3!}\cdot\frac{x^3}{3} + \frac{1}{5!}\cdot\frac{x^5}{5} \quad \ldots$$

N12.

$$Y(x) = \cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} \quad \ldots$$

N13.

$$Y(x) = 1 + \frac{x^2}{2!} - \frac{3\cdot x^4}{4!} + \frac{5\cdot x^6}{6!} - \quad \ldots$$

N14.

$$Y(x) = \sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \quad \ldots$$

N15.

$$Y(x) = -\frac{(2x)^2}{2!} + \frac{(2x)^4}{4!} - \frac{(2x)^6}{6!} + \frac{(2x)^8}{8!} - \quad \ldots$$

N16.

$$Y(x) = sh(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} \quad \ldots$$

N17.

$$Y(x) = \frac{x^3}{3} - \frac{x^5}{15} + \ldots + (-1)^{n+1}\frac{x^{2n+1}}{4n^2-1} + \quad \ldots$$

N18.

$$Y(x) = \ln|\sin x| = -\ln 2 - \cos 2x - \frac{\cos 4x}{2} - \frac{\cos 6x}{3} \quad \ldots$$

N19.

$$Y(x) = \ln(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} \quad \ldots$$

N20.

$$Y(x) = \frac{x-1}{x} + \frac{(x-1)^2}{2x^2} + \frac{(x-1)^3}{3x^3} + \quad \ldots; \quad x > 1/2$$

N21.

$$Y(x) = Arth\ x = x + \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} \quad \ldots$$

N22.

$$Y(x) = cth\ x = 1 + 2\left(e^{-2x} + e^{-4x} + e^{-6x} + \quad \ldots\right)$$

N23.

$$Y(x) = th\ x = 1 - 2e^{-2x} + 2e^{-4x} - 2e^{-6x} \quad \ldots$$

N24.

$$Y(x) = F(x) = x - \frac{x^3}{3^2} + \frac{x^5}{5^2} - \frac{x^7}{7^2} \quad \ldots$$

N25.

$$Y(x) = 1 + \frac{1}{2^x} + \frac{1}{3^x} + \frac{1}{4^x} \quad \ldots; \quad x > 0$$

N26.

$$Y(x) = arctg\ x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} \quad \ldots; \quad |x| < 1$$

N27.

$$Y(x) = arctg\ x = \frac{1}{x} - \frac{1}{3x^3} + \frac{1}{5x^5} - \frac{1}{7x^7} + \frac{1}{9x^9} \quad \ldots$$

N28.

$$Y(x) = \frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \quad \ldots; \quad x > 1$$

N29.

$$Y(x) = \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \quad \ldots; \quad |x| < 1$$

N30.

$$Y(x) = x + \frac{x^2}{2} + \frac{x^3}{3} + \quad \ldots; \quad |x| < 1$$

6.9.2 Передача массивов в качестве параметров

6.9.2.1 Задание

Оформить в виде отдельных функций:

1) ввод двухмерной целочисленной матрицы;

2) вывод двухмерной целочисленной матрицы;

3) обработка матрицы по индивидуальному заданию, которое следует выбрать из списка заданий в п. 6.9.2.2; в каждом задании есть какое-то преобразование матрицы и получение при этом не менее двух выходных значения.

Написать основную программу main для проверки работы этих функций.

Замечание 1. Все данные передавать в функции и обратно как параметры, а не как глобальные данные.

Замечание 2. Все результаты (простые переменные, массивы), полученные в ходе обработки матрицы, возвращать в основную программу и только там распечатывать (то есть обязательно должна быть организована передача выходных параметров).

Замечание 3. Минимальный шаблон основной программы для всех заданий:

1) вызов функции 1 для ввода матрицы;

2) вызов функции 2 для вывода введенной матрицы;

3) вызов функции 3 для обработки матрицы;

4) вызов функции 2 для вывода обработанной матрицы;

5) печать результатов обработки матрицы, полученных после выполнения п.3).

При выполнении конкретного задания может потребоваться определение еще и других функций в дополнении к данному "минимальному" набору.

6.9.2.2 Функции. Передача массивов в качестве параметров

ЗАДАНИЕ. Написать программу, состоящую из нескольких функций. Связь отдельных функций - только через список параметров.

№ 1.

Дана целочисленная квадратная матрица.

Найти наименьшее из значений элементов побочной диагонали, а также наименьшее значение для каждой из двух соседних с ней линий.

Все элементы на побочной диагонали заменить на 0.

№ 2.

Для заданной целочисленной квадратной матрицы порядка n построить двухмерную матрицу ($k \times 2$), состоящую из индексов элементов с наибольшим значением, где k - количество наибольших элементов.

Определить количество наибольших элементов исходной матрицы, которые лежат:

а) выше главной диагонали;

б) ниже главной диагонали;

в) на главной диагонали.

№ 3.

Дана целочисленная квадратная матрица.

Найти количество положительных элементов выше побочной диагонали и количество отрицательных элементов ниже побочной диагонали.

Все отрицательные элементы выше побочной диагонали заменить на 0.

№ 4.

Дана квадратная целочисленная матрица.

Найти значение наибольшего элемента среди элементов, лежащих выше побочной диагонали, и значение наименьшего элемента среди элементов, лежащих ниже побочной диагонали..

Все элементы побочной диагонали заменить на среднее арифметическое двух найденных значений.

№ 5.

В двумерной целочисленной матрице заменить все элементы, большие 7, на 7, а меньшие -7, на -7. Подсчитать число замен в обоих случаях.

№ 6.

Дана целочисленная квадратная матрица.

Найти наибольшее значение элементов главной диагонали и наименьшее значение элементов побочной диагонали.

Все элементы главной диагонали заменить на найденный наибольший элемент.

№ 7.

Дана целочисленная квадратная матрица A.

Найти значение наибольшего по модулю элемента матрицы и количество таких элементов.

Получить матрицу, составленную из индексов таких элементов.

Пример.

Дано: $A = \begin{pmatrix} -5 & 5 & 4 \\ 3 & -1 & -5 \\ 5 & 5 & 2 \end{pmatrix}$.

Получим: наибольших по модулю элементов - 5 штук. Индексы: $\begin{pmatrix} 00 \\ 01 \\ 12 \\ 20 \\ 21 \end{pmatrix}$.

№ 8.

В двумерной матрице (квадратной) упорядочить каждую строку по возрастанию элементов строки.

Затем из элементов главной диагонали найти наибольший и наименьший элемент.

№ 9.

В целочисленной квадратной матрице упорядочить строки по возрастанию значений наибольших элементов строк.

В полученной матрице найти сумму элементов первого столбца и сумму элементов последнего столбца.

№ 10.

Дана целочисленная матрица A(n×m).

Определить значение наибольшего и наименьшего элементов.

Составить вектор *b*, каждый элемент которого - это сумма элементов строки исходной матрицы, содержащей наибольший элемент.

Пример.

Дано: $A = \begin{pmatrix} 1 & 5 & 2 \\ 5 & 5 & 5 \\ 1 & 1 & 1 \\ 5 & 1 & 1 \end{pmatrix}$. Получим: *max* = 5; *min* = 1; $b = \begin{pmatrix} 8 \\ 15 \\ 7 \end{pmatrix}$.

№ 11.

Дана целочисленная матрица (n×m).

Для каждой строки определить разность наибольшего и наименьшего элементов.

В одномерном массиве, составленном из этих разностей, найти наибольший и наименьший элемент.

№ 12.

Дана целочисленная матрица (n×m).

Построить вектор, состоящий из наименьших значений элементов строк этой матрицы.

Найти сумму и произведение элементов этого вектора.

№ 13.

Дана целочисленная матрица (n×m).

Построить вектор, состоящий из наименьших значений в каждом столбце исходной матрицы.

Для получения вектора найти наибольшее и наименьшее значения.

№ 14.

Дана целочисленная матрица A (n×m).

Посчитать число отрицательных элементов в каждом столбце и из этих чисел сформировать вектор B.

Найти наибольшее и наименьшее значения для этого вектора.

Пример.

Дано: $A = \begin{pmatrix} 1 & 5 & -1 & -2 \\ -1 & 7 & 1 & -2 \\ 2 & 6 & 1 & -2 \\ -2 & 4 & 1 & 0 \end{pmatrix}$. Получим: B = (2, 0, 1, 3), $B_{max} = 3$, $B_{min} = 0$.

№ 15.

В двумерной целочисленной матрице найти среднее арифметическое каждого столбца. Затем найти наибольшее и наименьшее значения из этих среднеарифметических.

№ 16.

В двумерной целочисленной матрице определить:

1) *a* - количество простых чисел;

2) *b* - количество простых чисел, не превосходящих 101.

Все простые числа, не превосходящие 101, заменить на 101.

№ 17.

В двумерной целочисленной матрице определить долю (в процентах) чисел, больших 10, и долю (в процентах) чисел, меньших 10.

Все числа, большие 10, заменить на 10; меньшие 10 - на -10.

№ 18.

В двухмерной целочисленной матрице найти значения наибольшего и наименьшего элементов.

Заменить все элементы матрицы, меньшие некоторой заданной величины "*a*", на значение наименьшего элемента, а большие некоторой заданной величины "*b*" - на наибольший элемент.

№ 19.

В двумерной целочисленной матрице найти среднее арифметическое чисел, лежащих в диапазоне [-5, 5]. Затем все элементы из этого диапазона заменить на 1.

№ 20.

В двумерной целочисленной матрице найти количество:

1) четных чисел;
2) чисел, делящихся на 4.

Все четные элементы заменить на 2.

№ 21.

Дана целочисленная матрица (n×m).

Найти среднее арифметическое набольшего и наименьшего значений ее элементов.

Все элементы, имеющие наибольшее и наименьшее значение, заменить на это среднее арифметическое.

№ 22.

В двумерной целочисленной матрице определить количество отрицательных и количество положительных элементов.

Все элементы из диапазона [-10, 10] заменить на 0.

№ 23.

В двумерной целочисленной матрице найти сумму и произведение всех элементов, значение которых попадает в диапазоне [-5, 5].

Затем все элементы из этого диапазона заменить на 0.

6.10 КОНТРОЛЬНЫЕ ВОПРОСЫ ПО МОДУЛЮ 6

6.10.1 Определение функции.
6.10.2 Описание функции.
6.10.3 Вызов функции.
6.10.4 Передача входных и выходных параметров.
6.10.5 Передача массивов в качестве параметров.
6.10.6 Указатель на функцию.
6.10.7 Передача функции в качестве параметра другой функции.
6.10.8 Связь функций из разных файлов.

7 ОПРЕДЕЛЕНИЯ И ОПИСАНИЯ - ОБЩАЯ ФОРМА

Переменная состоит из двух компонентов: объекта (это область памяти - ОП) и имени этого объекта (идентификатора). Каждая переменная должна быть определена или описана. Это не одно и тоже .

Определение - это указание свойств переменной и назначение для нее области памяти.

Описание - это указание свойств переменной без назначения для нее области памяти. Описания нужны для того, чтобы обеспечить возможность ссылки.

Следует отметить, что понятия "описание" и "определение" применимы не только к переменным, но и к функциям. Общая форма определений и описаний:

<класс памяти> <тип данных> <описатели>;

Класс памяти (КП) и тип данных (ТД) могут быть опущены; если они заданы, то относятся к каждому описателю.

Описатель - это в простейшем случае просто идентификатор.

За каждым описателем может следовать инициализатор, задающий начальное значение соответствующей переменной.

Цель изучения данного модуля - рассмотреть вопросы, связанные с понятиями "описание" и "определение", и подробно разобрать материал по классам памяти.

7.1 ТИПЫ ДАННЫХ

Тип данных, указанный в описании или определении, чаще всего один из нижеследующих (существуют и другие идентификаторы типа):

char,
unsigned char,
int,
short int (или просто short),
long int (или просто long),
unsigned int (или просто unsigned),
float,
double (или long float),
void,
struct < метка >,
union < метка >,
enum < метка >,
<typedef-имя>,

где <метка> и <typedef-имя> должны быть предварительно описаны.

По умолчанию (если в определении или в описании тип данных не указан) предполагается тип int. Все указанные типы, кроме unsigned char, уже были рассмотрены. В следующем подразделе рассматривается тип unsigned char.

7.1.1 Тип unsigned char

Наличие ключевого слова "unsigned" ("беззнаковое") перед словом "char" предотвращает интерпретацию самого левого бита кода символа в качестве знакового бита. Такой знаковый бит был бы перенесен влево при преобразовании кода символа в тип "int" ("целый"), а иногда это нежелательно.

Например, имеем :

unsigned char k = 0xC0 ;

В двоичном представлении: k = 11000000 .

Операция сдвига вправо на 4 позиции

k >> 4

дает результат 0x0c ("00001100" - в двоичном представлении). Обычно это и нужно получить в большинстве приложений. Использование unsiqned в этом случае приведет при выполнении операции сдвига к расширению байта

11000000

до целого

00000000 11000000,

что после сдвига вправо дает правильное значение

00000000 00001100 .

Предположим, что k было бы типа char , а не unsigned char. Перед сдвигом значение k должно преобразоваться в int (так выполняется операция сдвига). При этом знаковый бит, равный 1, был бы использован для "расширения" байта

11000000

до двухбайтового целого со знаком

11111111 11000000

и после сдвига вправо на 4 позиции будет получено в результате

0000111111111100.

Теперь значение правых 8 бит равно 0xFC.

7.1.2 Директива typedef

Описание типа осуществляется с помощью директивы typedef . Эта директива позволяет связать нужный тип данных с некоторым именем (имя типа), а затем объявить переменные этого типа.

Общий вид использования директивы typedef:

typedef <спецификатор-типа> <описатели>;

Здесь: 1) спецификатор типа - это основной, производный тип или тип,
ранее определенный программистом;
2) идентификаторы в описателях - это описываемые имена типов.

Примеры описания типов с помощью директивы typedef:

```
typedef float miles,speed;  /* имена miles , speed описаны
                               как синонимы для имени float */
typedef float a[5], *pf;    /* тип "a" описан как массив из 5 компонентов типа float; тип "pf"
                               описан как указатель на объекты типа float */
typedef enum {mon, tue, wed} days; /* тип "days" описан как перечисляемый тип */
typedef struct {
        char fio[30] ;
        days dd;
        int progul ;
} student;      /* тип "student" - это тип структуры */
typedef student class[100];
                /* тип "class" - это массив из 100 компонентов типа "student" */
```

С помощью описанных таким образом типов можно определять переменные так же, как, например, с помощью стандартных типов float и int. Например:

```
class p1,p2,*p;   /* p1 ,p2 - структуры типа class , p - указатель на структуру типа class */
```

7.2 ОПИСАТЕЛИ В ОПРЕДЕЛЕНИЯХ И ОПИСАНИЯХ

Каждый описатель содержит только один идентификатор, который является именем определяемой или описываемой переменной. Описатели отделяются друг от друга запятыми. Пусть имеем:

T <описатель>;

где T- тип данных.

В качестве описателя может быть:

1) идентификатор - тогда определяется переменная с заданным именем типа T; например, "int a;" ;
2) (описатель) - то же самое, что описатель без скобок; например, "int (a);";
3) *описатель - определяется некоторый описатель с типом данных "указатель на T"; например, "int *a;" ;
4) описатель( ) - определяется некоторый описатель с типом данных "функция, возвращающая значение типа T"; например, "int a();" ;
5) описатель[N] - определяется некоторый описатель с типом данных "массив с N элементами типа T"; N- выражение с постоянным значением, элементы нумеруются от 0 до N-1; например, "int a[N]".

Описатели используются с некоторыми ограничениями. Функции не могут возвращать массивы или функции как значения (однако допустимы указатели на них). Кроме того, не могут быть описаны или определены массивы функций, не могут быть функции и компонентами структуры или объединения.

При использовании в описаниях и определениях, операция косвенной ссылки "*" имеет меньший приоритет по сравнению с операциями "[ ]" (для обозначения массива) и "( )" (для обозначения функций).

7.3 КЛАССЫ ПАМЯТИ

Время жизни (продолжительность существования в памяти) и область действия переменной определяется ее классом памяти. Класс памяти устанавливается при описании (или определении) переменной с помощью соответствующего ключевого слова: auto, extern, static, register.

Класс памяти	Ключевое слово	Продолжительность существования	Область действия	Примечание
Автоматический	auto	Временно	Локальная	Это локальные переменные, память для которых выделяется при входе в блок (т.е. составной оператор) и освобождается при выходе из него.
Регистровый	register	Временно	Локальная	Регистровые переменные подобны автоматическим переменным. Значения регистровых переменных должны, если возможно, помещаться в регистрах ЭВМ для обеспечения быстрого доступа к данным.
Статический	static	Постоянно	Локальная	Это локальные переменные, существующие в процессе всех выполнений блока. Память для них выделяется только один раз - в начале выполнения программы, и

				они существуют, пока программа выполняется.
Внешний	extern	Постоянно	Глобальная (все файлы)	Внешние переменные используются для связи между функциями, в том числе независимо скомпилированными функциями, которые находятся в разных файлах. Память, выделенная под внешнюю переменную, является постоянной, однако её содержимое может меняться.
Внешний статический	static	Постоянно	Глобальная (один файл)	Обычная внешняя переменная (типа extern) может использоваться функциями в любом файле, в то время как внешняя статическая переменная может использоваться только функциями того файла, в котором она определена.

Для переменной, определяемой вне функции, можно указать ключевое слово extern или static или вообще опустить указание класса памяти (по умолчанию будет extern). Для переменной, определяемой внутри функции, можно указать любое ключевое слово: auto, static, extern, register, - или вообще его не указывать (по умолчанию - auto).

Таким образом, если класс памяти не указан явно с помощью ключевого слова, то он задается по умолчанию по следующему правилу: если переменная определяется внутри функции, то у нее класс памяти auto, в остальных случаях имеем класс памяти extern.

7.3.1 Автоматические переменные

По умолчанию переменные, описанные внутри функции, являются автоматическими. Можно подчеркнуть это явно с помощью ключевого слова auto (для показа, что определение переменной не надо искать вне функции).

```
Пример.  main()
         {auto  int  pl1;
          . . .
```

Область действия автоматической переменной - только тот блок, в котором она определена. Рекомендуется описывать все переменные в начале тела функции, так чтобы областью действия их являлась вся функция. Однако, в принципе, можно описывать переменную внутри подблока, и тогда переменная будет известна только в этой части функции.

7.3.2 Регистровые переменные

Обычные переменные хранятся в памяти машины. Регистровые переменные запоминаются в регистрах центрального процессора, где доступ к ним и работа с ними выполняются гораздо быстрее, чем в памяти. В остальном регистровые переменные аналогичны автоматическим переменным.

```
Пример.  main()
         {register  int  qk;
          . . .
```

Объявление переменной как регистровой не всегда означает, что она действительно будет регистровой. Если компилятор не обнаружит свободных регистров, то переменная становится простой автоматической переменной.

Описание переменной как регистровой может увеличить скорости выполнения программы только в случае использования компилятора Си, не оптимизирующего программу, или с незначительной оптимизацией.

Иногда описание переменных как регистровых может даже замедлить выполнение программы. Так, предположим, что малоиспользуемая переменная описана с классом памяти register. Тогда выделение регистра для такой переменной может замедлить выполнение программы, если это будет мешать лучшему использованию регистров.

Ограничения, накладываемые на использование регистровых переменных:

1) невозможно определение адреса переменной, размещенной в регистре, с помощью адресной операции &;

2) в регистрах могут храниться только значения простых типов, таких как int и char, а также значения указателей.

В настоящее время во многих компиляторах Си класс register можно применять к переменным любых типов. Это означает «доступ к объекту так быстро, как возможно».

7.3.3 Статические переменные (локальные)

Название "статические" в данном случае не означает, что переменные не могут изменяться. Здесь слово "статические" означает, что переменные остаются в работе.

Статические переменные имеют ту же область действия, что и автоматические, но они не исчезают, когда содержащая их функция закончит свою работу. Компилятор хранит их значения от одного вызова функции до другого.

Часто статические переменные используются в следующей ситуации: необходимо подсчитать, сколько раз выполнилась некоторая функция, причем счетчик должен быть локализован в этой функции (должен принадлежать ей). В этом случае счетчик следует описать как статическую переменную.

Пример.

```
/* Пример использования статической переменной */
#include <stdio h>
void provstat();
int main()              /* Основная программа */
{int i;
  for (i=1;i<=3;i++)
    {
      printf("Итерация %d:\n",i);
      provstat();
    }
}
void provstat()   /* Подпрограмма */
{ int atx=1;           /* atx-автоматическая переменная */
  static int stx=1;   /* stx-статическая переменная */
    printf(" atx=%d, stx=%d\n",atx++,stx++);
}
```

Результат работы программы:

```
Итерация 1:
  atx=1,stx=1
Итерация 2:
  atx=1,stx=2
```

Итерация 3:
atx=1,stx=3

Обратите внимание на разницу в инициализации: автоматическая переменная atx инициализируется каждый раз, когда вызывается функция provstat(), а статическая переменная stx инициализируется только один раз при компиляции функции provstat().

7.3.4 Глобальные переменные

7.3.4.1 Внешние переменные и связь по данным функций из разных файлов

Для связи по данным функций из разных файлов используются внешние переменные (класс памяти extern). Если функция ссылается на внешнюю переменную, то в файле, содержащем эту функцию, должно быть описание или определение этого идентификатора (при определении память выделяется, при описании - нет.)

Когда класс памяти extern указан явно, то это значит, что имеем описание внешней переменной, которое необходимо для проверки типа и генерации кода. Описание внешнего идентификатора может встретиться во многих файлах, но только один файл должен содержать определение этого идентификатора. При определении класс памяти не указывается явно.

Итак, область действия внешних идентификаторов не ограничивается файлом, содержащим их определение, а включает также файлы с соответствующими описаниями (с классом памяти extern). Однако определение внешних переменных как статических (с классом памяти static) ограничивает их область действия файлом, содержащим эти переменные.

Пример. Рассмотрим описание и определение переменных x и i вне функций в двух файлах a.c, b.c.

Файл a.c содержит определения:	Файл b.c содержит описание для x и определение для i:
int x;	extern int x;
static int i;	static int i;
….	…

Память для переменной x выделяется в определении в файле a.c, и она может использоваться для связи между функциями в этих двух файлах.

С другой стороны, каждый файл имеет собственную локальную переменную i; память для нее выделяется в каждом из двух определений переменной i. Каждая переменная i доступна только функциям из файла, содержащего ее определение.

Замечание. При описании внешних массивов максимальное значение первого индекса указывать необязательно, так как оно будет получено из соответствующего определения.

7.3.4.2. Внешние переменные в рамках единого файла

Переменная, определенная вне функции, является внешней. Внешнюю переменную можно также описать внутри функции, которая использует ее, при помощи ключевого слова extern.

Пример 1.

```
int a ;  /* Две переменные a и c, определенные вне функции */
char c;
int main()
{
  extern int  a;   /* Указано, что 2 переменные a и c */
  extern char c;   /* являются внешними */
    . . .
```

Группу extern-описаний (то есть "exrtern int a; extern char c;") можно вообще опустить, если исходные определения переменных появляются в том же файле и перед функцией, которая их использует.

Включение описаний с ключевым словом extern в тело функции позволяет этой функции использовать внешнюю переменную, даже если она определяется позже в этом или другом файле (оба файла должны быть, конечно, скомпилированы, связаны или собраны в одно и то же время).

Если переменная с некоторым именем определена вне функции, а затем переменная с таким же именем определена внутри функции без слова extern, то под этим именем внутри функции создается новая автоматическая переменная. Эту новую автоматическую переменную можно пометить словом auto, чтобы показать, что это намерение, а не оплошность.

Примеры, приведенные ниже, демонстрируют возможные комбинации описаний.

Пример 2.

```
/* Внешняя переменная fx известна двум функциям main() и magic(), в которых она
описана с помощью ключевого слова extern  */
int fx;              /* fx определена как внешняя переменная */
int main()
  { extern int fx;  /* fx описана как внешняя переменная */
    . . . }
magic ()
  { extern int fx;   /*fx описана как внешняя переменная */
    . . . }
```

Пример 3.

```
/* Внешняя переменная fx известна двум функциям main() и magic(). В main() переменная
fx описана с помощью ключевого слова extern,  magic() она известна по умолчанию.  */
int fx;     /* fx-определена как внешняя переменная */
int main()
  {extern int fx; /* fx описана как внешняя переменная */
    . . . }
magic()
  { /* f(x) не описана совсем */
    . . . }
```

Пример 4.

```
/* Созданы три разные переменные с одинаковым именем fx.  Переменная fx в функции
main() является автоматической по умолчанию и локальной для  main(); в функции magic()
переменная fx явно определена автоматической и известна только для magic().
```

Внешняя переменная fx не известна ни main(),ни magic(), но обычно известна любой другой функции в файле, которая не имеет своей собственной локальной переменной fx */

```
int fx;  /* fx-определена как внешняя переменная */
int main()
  { int fx;  /* fx определена и является автоматической по умолчанию */
    . . . }
magic()
  { auto int fx; /* fx определена как автоматическая переменная */
    . . . }
```

7.3.5 Выбор класса памяти

Всегда следует стараться использовать автоматический класс памяти. Может показаться, что использование внешних переменных удобно: если описать все переменные как внешние, то не будет никаких забот при использовании аргументов и указателей для связи между функциями в прямом и обратном направлениях.

Но такое широкое использование внешних переменных является плохим стилем программирования, так как возможно, что одна функция изменит значения переменных в другой функции, когда это совсем не требуется.

Следует организовывать работу каждой функции по возможности автономно, а глобальные переменные использовать только в случае действительной необходимости.

Иногда полезны и другие классы памяти.

Пример 1. Необходимо подсчитать, сколько раз выполнилась некоторая функция, причем счетчик должен принадлежать ей. В этом случае счетчик следует описать как локальную статическую переменную (см. п.7.3.3).

Пример 2. Две или более функций работают с содержимым некоторой таблицы. Каждой функции необходим доступ к таблице и относящимся к ней ссылкам (указателям). При этом ни одна другая функция программы не должна иметь возможности обратиться к таблице. Это требование можно удовлетворить, если внутри одного файла пользоваться данными, описанными как глобальные статические. Эти данные будут доступны всем функциям из рассматриваемого файла и не доступны для функций из других файлов.

7.4 СИНТАКСИЧЕСКИЕ ОТЛИЧИЯ ОПРЕДЕЛЕНИЙ И ОПИСАНИЙ

Пример 1. Имеем внутри функции следующие определения и описания:

```
int  i, *ip, f( ), *fip( ), (*pf)( );
```

Здесь объекты i, ip, pf определяются (для них назначается область памяти), а объекты f и fip - описываются (для них не назначается область памяти):

```
i   -целая переменная;
ip  -указатель на целую переменную;
f   -функция, возвращающая целое значение;
fip -функция, возвращающая указатель на целое значение;
pf  -указатель на функцию, возвращающую целое значение.
```

По умолчанию класс памяти всех этих переменных - auto.

Пример 2.

```
static emp *db[m];
```

Здесь db определен как массив указателей на элементы типа emp. Элементы массива db нумеруются от 0 до m-1.

Синтаксические отличия описания объекта от его определения:

1) наличие ключевого слова extern указывает, что переменная описывается, а не определяется;

2) отсутствие параметров функции и соответствующего имени тела функции означает, что функция описывается, а не определяется; знак “;” в конце заголовка функции – также признак описания функции;

3) параметры функции описываются, но не определяются.

Пример 3.

```
/* описания */
        char   *str( ), *ind( );
        extern int max( ), n;
        extern float a[ ];
   /* определения */
        char  c;
        point x, y;
```

Пример 4.

```
int  *(*(*x)[6])( );
```

Здесь переменная x определяется как указатель на массив из 6 элементов, каждый из которых является указателем на функцию, возвращающую указатель на целый объект.

Пример 5. char (*(*y( ))[ ])( );

Здесь переменная y описывается как функция, которая возвращает указатель на массив указателей на функции, возвращающие знаковые значения.

Рекомендации, помогающие понять описания и определения.

1. Операция косвенной ссылки - префиксная (записывается перед идентификатором), остальные операции - постфиксные (записываются после идентификатора); смотрите примеры 4, 5 выше.

2. Основные конструкции в описаниях:

```
int x;     /* x-целого типа */
int *x;    /* x-указатель на целое */
int x[ ]; /* x-массив из целых элементов */
int x( ); /* x-функция, возвращающая целое */
```

3. Операция косвенной ссылки "*" имеет приоритет меньше, чем операции [ ] и ( ), используемые при задании массивов и функций, соответственно.

Пример 6.

"char *a[ ];" - воспринимается как "char *(a[ ]);", то есть a-массив указателей;

"char (*a)[ ];" - здесь a - это указатель на массив.

7.5 ИНИЦИАЛИЗАТОРЫ

Для присваивания начальных значений переменным при их определении используются инициализаторы. Они имеют одну из следующих двух форм:

```
=<значение>
={<список значений>}
```

Значение - это выражение с постоянным значением.

Список значений - это значения, отделенные друг от друга запятыми.

Сложное значение - это список значений, заключенный в фигурные скобки.

Статическим и внешним переменным по умолчанию присваиваются нулевые значения, в то время как автоматическим переменным никаких значений по умолчанию не присваивается. Более того, для многих компиляторов нельзя задать начальные сложные значения типа auto, то есть начальные значения для массивов и структур можно задать, только если они статические или внешние.

Пример 1.

```
static float eps=0.0001;
int i=0,j=0;
```

Пример 2.

```
int days[7]={1,2,3,4,5,6,7,};  /* Массив days - внешний */
main ( )
{ . . .
}
```

Пример 3.

```
int days[ ]={1,2,3,4,5,6,7};  /* Указывать размерность массива нет необходимости, так как
                                 компилятор сам определит ее из начальных значений */
int main( )
{int i ;
  extern int days[ ];   /* Необязательное описание */
```

```
  for (i=0; i< sizeof days/(sigeof(int));i++)
     printf (" День %d -  %d\n",i+1, days[i]);
  . . .}
```
Здесь подсчет количества элементов массива осуществлен с помощью выражения
```
        sizeof days/(sizeof(int)),
```
то есть

<размер всего массива в байтах> / <размер одного элемента массива в байтах>

Пример 4.
```
/* Пример вложенной инициализации */
int  m[5][3]={ {1, 1, 1 },
               {2, 2, 2 },
               {3, 3, 3 },
               {4, 4, 4 },
               {5, 5, 5 } };
```
Здесь элементам строки i присваивается значение i.

Пример 5.
```
 char strok1[9]="Helloy !";
/* stork1 -внешний массив (строка). Длина строки равна 9 - один элемент резервируется
          для признака конца строки. Размер 9 в квадратных скобках можно не
          указывать.
   strok1 - это краткая форма инициализации строки. */
 char strok2[ ]={'H','e','l','l','o','y',' ','!','\0'};
/* strok2 - это стандартная форма инициализации массива-строки. Без последнего знака
          имели бы просто массив знаков, а не строку. */
```

Пример 6.
```
typedef struct {
      float x,y;
} point;            /* Описание типа структуры point */
point p={0.0,0.1}; /* Инициализация переменной p типа point */
          int  main()
            { ... }
```

Пример 7.
```
typedef struct {
      char tifle[40];
      char authar [30];
      float value ;
} book ; /* Описание типа структуры book */
int main()
{ static book b={
       "программирование на языке СИ",
       "Д.Джехани",
       1.30
       }; /* Инициализация статической переменной b типа book*/
 . . .
}
```

Пример 8.
```
/* Печать таблиц */
int main ()
{
```

```
 static char *z[]={
   "------------",         /* z[0] */
   "|   x  |  y   |",      /* z[1] */
   "------------",         /* z[2] */
   "|      |      |"     };  /* z[3] */
   . . .
    for (i=0;i<=3 ;i++)     /* Печать шапки таблицы */
      printf("%s\n", z[i]);
      . . .
/* Для печати через строчку можно использовать z[3]; для печати конца таблицы - z[0] или
                z[2] */
```

Пример 9.

```
/* Примеры инициализации символьных строк */
#unclude <stdio.h>
#define  MSG  "Это - константа символьной строки"
char  m1[]=" Строка 1"; /* Инициализация внешнего символьного массива */
char  *m2 =" Строка 2"; /* Инициализация указателя внешнего символьного массива */
   int main()
{char *m3=" Строка 3"; /* Инициализация указателя символьного массива */
 static  char  *mytal[3]={"Стр1", "Стр22", "Стр333"};     /* Инициализация массива строк;
      mytal - массив из 3 указателей на символьные строки */
```

7.6 ЛАБОРАТОРНАЯ РАБОТА №7 "ИГРА. РАЗРАБОТКА ДИАЛОГОВОЙ ПРОГРАММЫ"

Цель лабораторной работы - организация диалоговой программы и закрепление в процессе ее создания всех основных элементов программирования на языке Си, рассмотренных в этом и предыдущих разделах.

Задание к лабораторной работе: написать программу, реализующую диалоговый режим игры. Конкретное задание выбрать из списка заданий в п.7.6.1. Программа должна иметь модульную структуру и, по возможности, использовать переменные разных классов памяти.

Замечание 1. Программа должна иметь модульную структуру, то есть состоять из отдельных функций. Так, обычно в виде отдельной функции реализуется основной алгоритм игры. Например, при угадывании слова по одной букве в виде отдельной функции можно оформить проверку, содержится ли введенная буква в заданном слове.

Замечание 2. На примере этой лабораторной работы рекомендуется апробировать использование переменных разных классов памяти. Так, в одной из функций можно использовать статическую локальную переменную; например, для подсчета числа ходов.

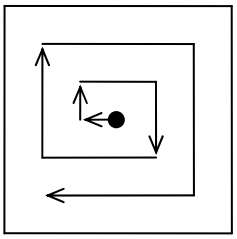
Замечание 3. При реализации программы следует разработать приемлемый для диалоговой программы интерфейс. Работа программы должна начинаться с выдачи на экран дисплея "заставки", где дается название игры и основные правила. Должны быть предусмотрены простейшие защиты от неправильного ввода. Примеры неправильного ввода: надо ввести цифру, а пользователь ввел букву; надо ввести не более трех знаков, а пользователь ввел больше и т.д. Должна быть предусмотрена возможность прерывания текущей игры в любой момент по желанию пользователя.. Игроку при этом следует сообщить о его текущих результатах, например, о количестве сделанных ходов или о количестве накопленных очков, и дать возможность, по желанию, повторить игру.

Замечание 4. Для реализации хорошего диалогового интерфейса полезно воспользоваться дополнительной литературой, список которой приведен в конце учебника.

7.6.1 Игровые программы

ЗАДАНИЕ. Написать диалоговую программу, реализующую один из предложенных ниже алгоритмов игры.

№1 Игра "прыжок кенгуру".

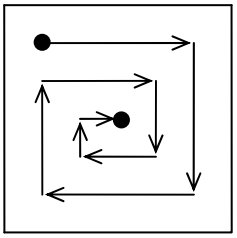


В центре поля 5×5 позиций (можно взять поле другого нечетного размера) стартует кенгуру. Она прыгает по спирали из центра в левый нижний угол либо в соседнюю клетку, либо через одну (случайным образом).

После каждого прыжка кенгуру охотник может поставить ловушку, общее число ловушек - не более 3-х. Устанавливая ловушку, охотник указывает ее координаты (в нижний левый угол ставить ловушку нельзя).

Если кенгуру при очередном прыжке попала в ловушку, то она поймана; если благополучно добралась до финиша, то "1:0" в пользу общества охраны животных.

№2 Игра "поймай зайца".

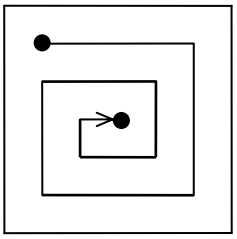


В левом верхнем углу поля 5×5 позиций (можно взять поле другого нечетного размера) стартует заяц. Он прыгает по спирали до центра. Прыгать заяц может либо в соседнюю клетку, либо через одну (случайным образом).

После каждого прыжка зайца охотнику предлагается поставить ловушку. Всего ловушек может быть не более 3-х. Устанавливая ловушку, охотник указывает ее координаты (в центр поля ставить ловушку нельзя).

Если заяц при очередном прыжке попал в ловушку, то он пойман; если благополучно добралась до центра, то охотник остался без зайца.

№3 Игра "кто вперед".



Два игрока стартуют в левом верхнем углу поля 9×9 и движутся к центру поля. Победит тот, кто придет первым. Положение одного игрока можно отмечать “+”, второго “*”, если оба в одной клетке - “0”.

Игроки могут ходить на одну или две клетки по спирали к центру (игрок вводит, соответственно, 1 или 2).

На пути есть 5 особых клеток, расставленных программой случайным образом:

1) назад на 3 клетки;
2) вперед на две клетки;
3) пропустить ход;
4) сделать дополнительный ход;
5) встать в одну клетку с противником.

Клетки невидимы до тех пор, пока один из игроков не попадет на них.

№ 4 Игра “найти невидимку” (диагональная).

На поле 4 × 4 позиции (можно взять поле другого размера) в одной из клеток стоит невидимка, которого должен найти игрок.

Игрок указывает координаты невидимки. Если угадал, то нашел невидимку. Если не угадал, то невидимка передвигается в другую позицию, а игроку сообщается, где невидимка был в момент хода.

Невидимка может двигаться только по диагонали.

№ 5 Игра “найти невидимку” (вертикальная).

На поле 5 × 5 позиции (можно взять поле другого размера) в одной из клеток стоит невидимка, которого должен найти игрок.

Игрок указывает координаты невидимки. Если угадал, то нашел невидимку. Если не угадал, то невидимка передвигается в другую позицию, а игроку сообщается, где невидимка был в момент хода.

Невидимка может двигаться только по горизонтали или вертикали.

№ 6 Игра в кости.

Играющий называет любое число в диапазоне от 2 до 12 и ставку, которую он делает в этот ход. Программа с помощью датчика случайных чисел дважды выбирает числа от 1 до 6 (“бросает кубик”, на гранях которого цифры от 1 до 6). Если сумма выпавших цифр меньше 7 и играющий задумал число меньшее 7, он выигрывает сделанную ставку. Если сумма выпавших цифр больше 7 и играющий задумал число большее 7, он также выигрывает сделанную ставку. Если играющий угадал сумму цифр, он получает в 4 раза больше очков, чем сделанная ставка. Ставка проиграна, если не имеет места ни одна из описанных ситуаций.

В начальный момент у играющего 100 очков.

№ 7 Игра “морской бой” (два игрока).

На поле 4 × 4 клетки игроки устанавливают три корабля по 1 клетке (у каждого игрока - свое поле).

Программа “запоминает” положение кораблей.

Затем игроки начинают “поражать” корабли противника, по очереди указывая координаты предполагаемого корабля. Результат попадания (попал или мимо) отмечается на поле (например, “*” - попал, “+” - мимо).

№ 8 Игра “морской бой” (один игрок).

На поле 4 × 4 клетки стоят три вражеских корабля по 1 клетке (программа выставляет их случайным образом). Для игрока корабли невидимы.

Игрок должен “поразить” корабли. Во время очередного хода игрок указывает позицию предполагаемого корабля. Если игрок угадал, клетка отмечается “*”, иначе - “+”.

№ 9 Игра “коровы и быки” для слов.

Два игрока загадывают по четырехбуквенному слову (вводят без отображения на экране).

Затем каждый игрок должен угадать слово противника. Игроки ходят по очереди. На каждом шаге игрок называет четырехбуквенное слово, а программа сообщает, сколько букв угадано (быки) и сколько букв угадано и стоит на своем месте (коровы).

№ 10 Игра “цифры рядом”.

Программа загадывает целое четырехзначное положительное число. Игрок должен отгадать число. Он называет две цифры и получает один из 3-х ответов:

1) есть, если совпала одна или обе цифры с цифрами задуманного числа;

2) есть рядом, если совпали обе цифры, причем в числе они стоят рядом в любом порядке;

3) нет - в противных случаях.

Например: исходное число 1277.

“13”, “17” - есть; “12”, “72” - есть рядом.

Когда игрок готов назвать все число, он сообщает все 4 цифры в определенном порядке. Если совпало с заданным числом - выигрыш, нет - проигрыш.

№ 11 Игра “слово”.

Ведущий вводит слово (без отображения его на экране). На экране высвечивается столько звездочек, сколько букв в слове. Предпочтительнее, чтобы слово вводил не ведущий, а сама программа случайным образом выбирала его из словаря слов.

Играют двое. Выигрывает тот, кто первым отгадает слово. Играют по очереди, указывая по одной букве. Если в слове есть эта буква, она высвечивается вместо звездочки. Игрок может в случае своего хода пойти на риск: попытаться сразу угадать целиком все слово. Если не угадает, то пропускает один ход; если угадает, то игра окончена в его пользу.

№ 12 Игра “угадай слово”.

Играют: программа, ведущий и игрок. Роль ведущего может выполнять программа.

Ведущий загадывает слово (без отображения его на экране) и указывает подсказку об этом слове (с отображением на экране). На экране высвечивается столько звездочек, сколько букв в слове, и подсказка об этом слове. Предпочтительнее, чтобы слово вводил не ведущий, а сама программа случайным образом выбирала его из словаря слов..

Игрок отгадывает слово по одной букве. Если в слове есть эта буква, она высвечивается вместо звездочки. Игроку всегда сообщается о номере его очередного хода. В любой момент игрок может отказаться от игры.

По окончании игры сообщается, за сколько ходов игрок угадал слово или сколько было сделано ходов до прерывания игры, и какое слово было загадано, если игрок отказался от игры.

№ 13 Игра “угадай число”.

Ведущий загадывает целое положительное число (без отображения его на экране). На экране высвечивается столько звездочек, сколько цифр в числе. Предпочтительнее, чтобы число вводил не ведущий, а сама программа случайным образом формировала его.

Игрок отгадывает число по одной цифре. Если угадал цифру, она высвечивается вместо звездочки. Игроку всегда сообщается о номере его очередного хода. В любой момент игрок может отказаться от игры.

По окончании игры сообщается, за сколько ходов игрок угадал число или сколько было сделано ходов до прерывания игры, и какое число было загадано, если игрок отказался от игры.

№ 14 Игра “два числа”.

Игрок должен угадать два числа “a” и “b”, задуманных программой.

Игрок задает число и получает один из ответов:

1) число лежит левее интервала (a, b);

2) число лежит правее интервала (a, b);

3) число лежит внутри интервала (a, b);

4) число совпало с одним из задуманных чисел a, b.

№ 15 Игра “больше-меньше числа”.

Программа задумывает натуральное число. Игрок должен угадать его.

Игрок задает свое число и получает один из 3-х ответов:

1) больше заданного;

2) меньше заданного;

3) совпало с заданным; игра окончена.

В зависимости от размера исходного числа игроку дается определенное количество попыток.

№ 16 Игра “двадцать одно” (человек - программа).

Играют: человек и программа.

Игроки по очереди называют цифры от 1 до 9. Эти цифры суммируются.

Проигрывает тот, кто первым дойдет до числа 21 или более.

Например:

Номер хода	Программа	Человек	Сумма
1	9		9
2		9	18
3	2		20
		проиграл	

Программа выбирает цифры случайным образом.

№ 17 Игра “двадцать одно” (человек - человек).

Два игрока по очереди называют цифры от 1 до 9. Программа суммирует эти цифры.

Проигрывает тот, кто первым дойдет до числа 21 или более.

Например:

Номер хода	Игрок 1	Игрок 2	Сумма
1	9		9
2		9	18
3	2		20
4		Проиграл	

№ 18 Игра “Тренировка памяти - числа” (2 игрока).

Играют два игрока. Каждому по очереди на определенное время высвечивается несколько чисел, полученных с помощью датчика случайных чисел. Размер чисел ограничен (например, не более 3-х цифр).

Игроки должны воспроизвести числа. Каждому игроку дается определенное число шагов (игроки указывают это число в начале игры). Время запоминания также указывается игроками в начале игры.

Игроки могут играть в 2-х режимах:

а) просто воспроизвести числа;

б) воспроизвести числа в том же порядке.

№ 19 Игра “Тренировка памяти - слова” (1 игрок).

Программа на определенное время высвечивает несколько слов - существительных. Слова выбираются из специального файла случайным образом (или из специальной таблицы).

Игрок должен воспроизвести слова.

Замечания:

1) время для запоминания может быть различным (например, в программе предусматривается три временных режима или это время задает игрок в начале игры);

2) число слов может быть различным (см. пункт 1) - аналогично);

3) игрок может играть в 2-х режимах:

а) просто воспроизвести слова;

б) воспроизвести слова в определенном порядке.

№ 20 Игра “устный счет”

На каждом шаге играющему предлагается два числа и арифметическое действие, которое надо выполнить.

Если игрок отвечает неверно, программа сообщает правильный ответ и дает следующее задание с тем же арифметическим действием.

Размер чисел и максимальное время ответа устанавливаются по желанию игрока в начале игры.

7.7 КОНТРОЛЬНЫЕ ВОПРОСЫ ПО МОДУЛЮ 7

7.7.1 Понятия "определение" и "описание" и их синтаксическое отличие.
7.7.2 Типы данных. Директива typedef.
7.7.3 Понятие "классы памяти".
7.7.4 Автоматический класс.
7.7.5 Регистровый класс.
7.7.6 Статический локальный класс.
7.7.7 Внешний класс.
7.7.8 Внешний статический класс.
7.7.9 Инициализаторы.

8 ФАЙЛЫ

Цель изучения данного модуля - познакомиться с организацией работы с файлами в языке Си, а также приобрести практический опыт создания программ, использующих файлы, в частности таких программ, для которых исходные данные поступают из файла, а результаты работы записываются в файл.

Файл - это специальным образом оформленная совокупность логически связанной информации. Примеры файлов:

1) текст исходного модуля;
2) готовая к выполнению программа;
3) совокупность исходных данных для программы.

Обычно файлу ставится в соответствие носитель с информацией - участок памяти на диске или устройство, выполняющее операции обмена (ввода-вывода).

Ввод-вывод в языке Си организован посредством библиотечных функций.

Система Borland C++ поддерживает ввод-вывод:

1) по стандарту ANSI (то есть ввод-вывод "чисто Си") - называется буферизованный, или форматированный, или потоковый, ввод-вывод;
2) Unix-подобный метод ввода-вывода - называется небуферизованный, или неформатированный, ввод-вывод;
3) собственный ввод-вывод языка C++.

Мы рассмотрим ввод-вывод по первому методу - по стандарту ANSI.

Следует различать два понятия: файл (file) и поток (stream). Система Си поддерживает ввод-вывод, не зависящий от того, какое используется реальное устройство ввода-вывода. Вот такое абстрактное, логическое устройство ввода-вывода называется потоком. Конкретное же физическое устройство (диск, терминал) с соответствующим способом хранения на нем информации называется файлом. Таким образом, все потоки одинаковы, чего нельзя сказать о файлах. В итоге, например, так как поток не зависит от физического устройства, то одна и та же стандартная подпрограмма записи информации в поток может записывать информацию и на диск, и на магнитную ленту, и выводить на экран.

Специальный оператор открытия файла (стандартная функция fopen) связывает поток с файлом. Как только файл открыт, то информация может передаваться между ним и Вашей программой. Каждый поток, связанный с файлом, имеет управляющую структуру, называемую FILE. Она описана в заголовочном файле stdio.h.

В Си различают две категории файлов (потоков): текстовые (text) и двоичные (binary). Текстовые потоки состоят из строк символов и предназначены для чтения человеком. Однако может не быть взаимооднозначного соответствия между символами, которые передаются в потоке и выводятся на экран. Среди символов пара может соответствовать возврату каретки или символу табуляции. Двоичный поток - это последовательность байтов, которые взаимооднозначно соответствуют тому, что находится на внешнем устройстве. Для пользователя текстовый файл от файла данных отличается в основном признаком: t - текстовый, b - двоичный, - который может указываться при определении файлов.

8.1 ПОТОКИ (stream)

Итак, большинство программ читают вводимые данные из специального файла - потока (stream). В поток записываются и выводимые данные. Поток - это файл, ассоциированный с буфером, позволяющим эффективный ввод и вывод на уровне пользователя. Буфер - это выделенный участок памяти. При буферном вводе информация с носителя вначале поступает в буфер, а затем оттуда передается прикладной программе. Аналогично и при

буферном выводе - информация из программы вначале поступает в буфер, а затем - на носитель. Вообще, при буферном вводе-выводе передача информации в (из) файла осуществляется блоками из pbs знаков, где pbs (physical block size) - это обозначение размера физического блока, связанного с периферийным устройством, на котором хранится файл.

Если для ввода и вывода в программе используются потоки, то для программиста буферизация выполняется автоматически; в противном случае для эффективной работы программы буферизация ввода-вывода должна быть организована самим программистом. Поэтому обычно используется ввод-вывод потоков, а функции ввода вывода нижнего уровня используются лишь тогда, когда программист хочет управлять вводом и выводом на уровне детализации, недоступном с помощью потоков.

В дальнейшем всегда будем, говоря файл, иметь в виду поток.

8.2 ОПРЕДЕЛЕНИЕ ПОТОКА

Для управления потоками используются потоковые указатели (stream pointer). Потоковый указатель является ссылкой (указателем) на структуру, содержащую информацию о соответствующем файле.

Потоки (потоковые указатели) определяются с помощью заранее определенного типа структуры (с именем типа) FILE следующем образом:

FILE *<имя потокового указателя>,

где имя потокового указателя, или просто имя потока - это имя указателя на структуру FILE.

Описание типа FILE дается в файле stdio.h; поэтому для того, чтобы объявлять файлы, в программе должно быть указано включение файла stdio.h:

#include <stdio.h>

Структура определения файла для потока, определенная в stdio.h:

```
typedef struct
 {
    short level;        В определении типа FILE указывается, например,
    usisigned flag;   адрес буфера,  с которым будет работать данный
    . . .                  файл; адрес текущего символа, который будет
 } FILE;                  передаваться (приниматься) в прикладную программу.
```

Вообще, в файле stdio.h даны описания, используемые для стандартного ввода-вывода. Отметим среди этих описаний еще следующие:

```
#define EOF -1 /*Константа, возвращаемая при обнаружении конца файла*/
                  /*Для текстовых файлов EOF есть код символа [Ctr/Z] */
#define NULL 0 /*Константа с нулевым значением, часто используется как
                значение пустого указателя. Не путать со знаком ‘0’*/
```

8.3 СТАНДАРТНЫЕ ПОТОКИ

Для любой программы в системах с Си всегда доступны три стандартных потока:

stdin - файл стандартного ввода, или просто стандартное входное устройство;

stdout - файл стандартного вывода, или просто стандартное выходное устройство;

stderr - файл стандартных сообщений об ошибках (в этом файле записываются сообщения об ошибках), или просто стандартное устройство для вывода ошибок;

stdprn - стандартный принтер (в Турбо-Си).

Эти файлы всегда связываются (отождествляются) с пользовательским терминалом (дисплеем и клавиатурой).

Определения стандартных потоков включены ввод файл stdio.h и имеют вид:

FILE *stdin, *stdout, *stderr;

Стандартные файлы автоматически открываются при входе в программу и закрываются при выходе из нее.

Для чтения и записи данных из (в) стандартных потоков можно использовать следующие стандартные функции (прототипы - в stdio.h):

getchar (putchar) - чтение (запись) знака из stdin (в stdout);

gets (puts) - чтение (запись) строки из stdin (в stdout);

scanf (printf) - форматный ввод (вывод) из stdin (в stdout).

8.4 НЕСТАНДАРТНЫЕ ПОТОКИ

Нестандартные потоки - это потоки, определяемые программистом. То есть если пользователь хочет использовать нестандартные потоки, то определять, открывать и закрывать их он должен сам в своей программе.

Потоки определяются с использованием типа FILE - см. выше п.8.2. Например:

FILE *fptr;

Чтобы открыть файл как поток и тем самым организовать буфер для этого файл, используется функция fopen.

Замечание. Возможен и другой вариант открытия файла: сначала файл можно открыть с помощью, например, функции open, а затем преобразовать его в поток, используя функцию fdopen.

Для чтения и записи данных из (в) потока используются следующие стандартные функции и макросы (прототип - в stdio.h):

fgetc,fputc
getc,putc } - чтение (запись) знака из (в) указанный поток;

fgets, fputs - чтение (запись) строки из (в) указанный поток;

fscanf, fprintf - форматированный ввод (вывод) из (в) указанный поток.

Указатель файла отмечает положение в потоке следующего элемента данных ввода или вывода. Для установки указателя файла на заданную позицию используется функция fseek; для установки указателя на начало файла - функция rewind.

После обработки файл должен быть закрыт вызовом функции fclose. После завершения работы программы все файлы закрываются автоматически.

Перечисленные выше стандартные функции подробно рассматриваются далее, однако существуют и другие функции для работы с файлами (их прототипы также даны в заголовочном файле stdio.h). Среди них отметим функцию feof(), которая служит для определения конца файла, и функцию remove(), которая уничтожает указанный файл. Прототипы этих функций:

```
int feof(FILE *fptr); //Возвращает 1 (истина), если достигнут конец файла,
                      // и 0 – в противном случае
int remove(char *filename);
```

8.4.1 Открытие файла (потока)

Для открытия файла используется функция fopen(). Прототип функции:

FILE *fopen(char *fname, char *type);

Эта функция открывает файл с именем, заданным строкой символов fname. Параметр type указывает на строку, определяющую режим использования открываемого файла. Функция возвращает указатель на открытый файл. Этот указатель должен быть предварительно описан как указатель на структуру FILE.

В случае ошибки (например, при попытке открыть несуществующий файл для чтения) функция возвращает NULL. Поэтому сразу же после обращения к fopen() надо с помощью оператора if проверять, не имеет ли ссылка на файл значение NULL.

Пример. Открытия файла с именем rezult.txt для записи.

```
FILE *fptr;      /* Это - определение потока. fptr - указатель на FILE */
fptr = fopen("rezult.txt","w");
if (fptr = =NULL)
        { /* Файл не открылся. Напечатать его имя и режим обработки и аварийно
        завершить программу */
        ⋮
        }
else
        { /* Файл открылся. Провести обработку файла */
        ⋮
        }
```

8.4.1.1 Имя файла может быть указано с помощью одного из следующих 3-х способов.

1. Имя файла записывается в виде строковой константы.

Пример 1. `fptr = fopen("prim.c", "r");`

↑

имя файла

2. В программе может быть определен массив типа char. Тогда имя открываемого файла может быть прочитано с терминала и записано в этот массив.

Пример 2.

```
char  imf [81];
printf("\n Введите имя файла с данными для ввода:");
gets(inf);
fptr = fopen(imf,"r+");
```

3. Пользователь может указать имя файла непосредственно в команде на выполнение программы, указав его в качестве аргумента командной строки.

Рассмотрим подробнее 3-ий способ. Пусть, например, мы имеем готовую для выполнения программу wwod, которая записана в подкаталоге powt01 каталога USER. Требуется, чтобы программа wwod работала с двумя файлами F1.ww и F2.ww.

Пусть система работает в корневом каталоге. Тогда для выполнения программы wwod укажем в командной строке MS DOS команду:

```
\USER\powt01  wwod  F1.ww  F2.ww <ВК>
                          ↑
                  Это аргументы командной строки, которые
                  передаются выполняемой программе.
```

Для того, чтобы в программе получить доступ к аргументам командной строки, надо определить главную функцию main следующим образом:

```
 int main (int  karg, char  *arg[ ])
  /* karg - количество аргументов, т.е. строк, разделенных пробелом */
  /* arg - адреса аргументов */
  { . . . }
```

Система все параметры командной строки выделяет в отдельные строки символов. Количество строк, то есть количество аргументов, присваивается параметру karg. В нашем случае karg=3.

Устанавливается массив указателей arg на эти строки; то есть первый параметр “wwod” запишется в память, а адрес присвоится элементу arg[0]; строка “F1.ww” запишется в память, а ее адрес присвоится элементу arg[1] и т.д. Таким образом, массив arg[ ] указывает на следующее:

arg[0] → "wwod"
arg[1] → "F1.ww"
arg[2] → "F2.ww"
arg[3] → (null)

В некоторых версиях (MS DOS версии 3.x) arg[0] содержит имя выполняемой программы, в некоторых версиях (MS DOS версии 2.x) arg[0] указывает на пустую строку (""). Arg[4] также указывает на пустую строку (содержит ноль).

Если имена файлов были переданы через командную строку, то в программе можем, например, записать.

```
fptr1 = fopen (arg[1], "r");
fptr2 = fopen (arg[2], "w");
```

Пример 3.

```
/* Программа считывает содержимое файла , преобразует строчные буквы в прописные и записывает результат в отдельный файл с одновременной выдачей на экран дисплея . Имена исходного и полученного файлов передаются через командную строку при вызове программы. */
#include<stdio.h>
int main (argc, argv)  /* Классический стиль оформления заголовка main */
   int    argc;
   char  *argv[];
{FILE *fin, *fout;
 int ch;

     fin = fopen( *(++argv), "r");
     fout= fopen( *(++argv), "w");
     while ( (ch = getc(fin)) != EOF)
         { if ( ('a'<=ch) && (ch<='z') )
           ch = ch+'A'-'a';
           putc(ch,fout);
           putchar(ch);
         }
}
```

8.4.1.2. Режим использования файла (параметр type в функции fopen) - это одна из следующих строк:

"r" - чтение (открыть файл для чтения);
"w" - запись (создать файл для записи);
"a" - добавление (открыть для добавления в существующий файл);
"r+" - чтение с модификацией (открыть файл для чтения и записи);
"w+" - запись с модификацией (создать файл для чтения и записи);;
"a+" - добавление с модификацией (открыть для добавления в существующий файл или создать для чтения и записи).

В режиме чтения "r":

1) существующий файл открывается для ввода;

2) при попытке открыть для чтения несуществующий файл fopen выдает значение NULL.

В режиме записи "w":

1) если файл существовал, то старая копия теряется, и запись будет осуществляться с начала файла;

2) если файл не существовал, то создается новый файл, в который можно осуществлять запись.

В режиме добавления "a":
можно осуществлять запись в конец существующего файла или в начало нового.

Режимы с модификацией при открытии файлов идентичны соответствующим режимам без модификации, однако в дальнейшем они позволяют переходить с ввода на вывод, и наоборот, перемежая вызов функций ввода и вывода. Однако:

а) вывод не может сразу следовать за вводом без использования функций fseek() (установка указателя файла) или rewind() (установка указателя на начало файла);

б) ввод не может сразу следовать за выводом без использования функций fseek() или rewind() или операции ввода, которая встречает конец файла.

В режиме добавления "a' или "a+" при записи выходных данных в файл текущее значение указателя файла игнорируется. Все выходные данные записываются в конец файла, а указатель файла устанавливается на конец выведенных данных.

Для задания того, что файл открыт или создан в текстовом режиме следует в записи режима указать символ "t" (например, "rt", "w+t" и т.д.); аналогично для задания двоичного режима следует добавить "b" (например, "wb", "a+b" и т.д.)

Если "t" или "b" не указано, то по умолчанию обычно принимается "t".

8.4.2 Закрытие потока

После обработки поток должен быть закрыт вызовом функции fclose(). После завершения работы программы все файлы закрываются автоматически.

Прототип функции fclose:

```
int  fclose(FILE *stream);
```

Функция закрывает поток stream. Перед закрытием все буферы, связанные с потоком stream, очищаются. Возвращаемое значение: 0 - при успешном завершении, EOF - если обнаружена ошибка.

Пример.

```
  ⋮
FILE  *ved;
  ⋮
ved = fopen("ved.dat", "r");
if (ved = NULL)
        { printf ("/n Ошибка при открытии файла ved.dat");
          exit(1);    /* Завершение программы с кодом 1 */
        }
<обработка файла>
fclose(ved);
  ⋮
```

8.4.3 Очистка потока

Прототип функции fflush() (в stdio.h):

```
int  fflush(FILE *stream);
```

Функция fflush() вызывает запись в stream содержимого буферов, связанных с открытыми потоками вывода, и чистит содержимое буфера, если stream - открытый поток ввода (чаще всего для очистки буфера ввода и используется). Поток stream остается открытым.

Пример.

```
scanf(“%d”, &n);
fflush(stdin);     /* Очистка вх. потока stdin в случае неправильного ввода */
scanf (“%d”, &i);
```

fflush(stdin); /* Очистка вх. потока stdin в случае неправильного ввода и для последующего нормального выполнения функции gets (надо "убрать" символ конец строки, оставшийся от scanf, чтобы gets нормально проработала). */
gets (str);

8.4.4 Обработка (чтение и запись) нестандартных текстовых файлов

Для чтения и записи данных из (в) потока используются следующие стандартные функции (прототип - в stdio.h):

fgetc,fputc
getc,putc
} - чтение (запись) знака из (в) указанный поток; getc, putc - макросы, а не функции;

fgets, fputs - чтение (запись) строки из (в) указанный поток;
fscanf, fprintf - форматированный ввод (вывод) из (в) указанный поток.

8.4.4.1 Чтение (запись) символа из (в) потока

Чтение. Прототип функции:

```
int     fgetc(FILE *stream);
```

Прототип макроса:

```
int     getc(FILE *stream);
```

Вводит следующий символ из потока, указанного в stream.

В случае успеха fgetc() и getc() возвращают прочитанный символ после преобразования его в целые без знака. При конце файла или ошибке возвращают EOF.

Запись. Прототип функции:

```
int     fputc(int ch, FILE *sream);
```

Прототип макроса:

```
int     putc(int ch, FILE *stream);
```

Выводит заданный символ ch в заданный поток stream. В случае успеха fputc() и putc() возвращают символ ch, в случае ошибки - EOF.

По действию соответствующие функции и макросы эквивалентны. Функция работает медленнее, чем макрос, но на каждый вызов требует меньше памяти. Кроме того, существуют ситуации (например, при работе с указателями), когда необходимо использовать именно функцию, а не макрос. Так, нельзя использовать функцию, указывающую на макрос.

Пример. Оформить функцию, которая заданный файл копирует на экран, то есть посимвольно из указанного файла передает информацию на экран.

```
#include<stdio.h>
/* Функция filecopy() копирует заданный файл на экран.
Входные параметры.
   fp - указатель заданного файла.
   Возможны два варианта:
      1) файл fp создан так, что в него включены символы перехода на
       новую строку '\n' (редакторы текстов обрабатывают перед записью тексты исходных
      модулей, добавляя в конец каждой строки '\n');
       2)  файл fp не содержит символов '\n', но известна длина строки.
   Установим: признак k:  k=0, если имеет место вариант 1;
   k=<число символов в строке>, если  имеет место вариант 2.
Выходные параметры - нет                                          */
int filecopy(fp,k)
```

```
  FILE *fp;  /* указатель заданного файла - входной параметр */
  int   k;       /* признак - входной параметр                */
{ int i;    /* номер следующего печатаемого символа в строке */
  int ch; /* печатаемый символ */
    i = 1;
    while ((ch=getc(fp)) != EOF)
        {if ((k!=0) && (i > k ))
           {putchar('\n'); i=1; };
         putchar(ch);  i++;          }
}
 int main()   /* Основная программа для печати заданного файла */
     { FILE *copy;
       char in[21];
       printf("\n Input name of file: ");   gets(in);
       copy = fopen(in,"r");
       if (copy != NULL)
          { filecopy(copy,0);
            fclose(copy);   }
       else
          printf(" File %s not open for reading\n",in);
     }
```

8.4.4.2 Форматированный ввод (вывод) в (из) потока

Ввод. Прототип функции:

```
int       fscanf(FILE *stream, char * format [, argument, . . .] );
```

Вывод. Прототип функции:

```
int       fprintf(FILE *stream, char * format [, argument, . . .] );
```

Функция fscanf() читает данные из заданного входного потока. Функция fprintf() записывает данные в заданный выходной поток. Имя потока задано параметром stream. Параметр format задает строку формата, в соответствии с которой производится форматированный ввод или вывод переменного числа величин, заданных параметром “argument . . .

Функция fscanf() возвращает количество успешно введенных и сохраненных входных аргументов. Если при чтении входного потока функция встретила конец файла, то возвращает EOF.

Функция fprintf() возвращает количество выведенных байтов. В случае ошибки возвращает EOF.

Эти функции используются, если в файле должны храниться значения арифметических данных.

Пример 1.

```
/* Исходная матрица размерностью (m*n) хранится в файле mat1.dat.
  Вычислить сумму элементов в каждой строке, упорядочить строки в порядке
возрастания этих сумм и записать полученную матрицу в новый файл mat2.dat */
#include <stdio.h>
#include <conio.h>
FILE *ved;

/* Функция создания файла mat1.dat, содержащего исходную матрицу */
int sozdmat1()
{int n,m;   /* n - количество строк, m - количество столбцов матрицы */
 int i,j,x; /* i,j,x - вспомогательные переменные */
  ved=fopen("mat1.dat","w"); /* Открытие файла mat1.dat для записи */
```

```
   if (ved==NULL) return -1;   /* Файл не открылся */
   do {printf("\nВведите размеры матрицы (не более 10): ");
       scanf("%d%d",&n,&m); }
   while (n>10||n<1||m>10||m<1);
   fprintf(ved,"%d  %d ",n,m); // Между спецификациями формата стоит пробел,
                               // иначе два числа  в файле «сольются» в одно число
   printf("Введите целую матрицу (%d*%d) построчно: \n",n,m);
   for(i=0;i<n;i++)
    for(j=0;j<m;j++)
     {scanf("%d",&x);
      fprintf(ved,"%d  ",x);} // После спецификации формата стоит пробел,
                              // иначе числа  в файле «сольются» в одно число
   fclose(ved);
   return 0;
}

/* Функция чтения файла mat1.dat, содержащего исходную матрицу,
   обработки этой матрицы и записи новой матрицы в файл mat2.dat */
int sozdmat2()
{int n,m;     /* n - количество строк, m - количество столбцов матрицы */
 int mat[10][10], /* Матрица */
     sum[10];     /* Сумма элементов каждой строки исходной матрицы */
 int i,j,k,x; /*i,j,k,x - вспомогательные переменные */
/* Чтение исходной матрицы из файла mat1.dat */
   ved=fopen("mat1.dat","r"); /* Открытие файла mat1.dat для чтения */
   if (ved==NULL) return -1;  /* Файл не открылся */
   fscanf(ved,"%d %d ",&n,&m);
   for(i=0;i<n;i++)
    for(j=0;j<m;j++)
     fscanf(ved,"%d",&mat[i][j]);
   fclose(ved);
/* Вывод исходной матрицы на экран */
   printf("Исходная матрица: \n");
   for(i=0;i<n;i++)
    {for(j=0;j<m;j++) printf("%d ",mat[i][j]);
     printf("\n"); }
/* Определение суммы элементов в каждой строке матрицы */
   for(i=0;i<n;i++)
    {sum[i]=0;
     for(j=0;j<m;j++) sum[i]+=mat[i][j]; }
   printf("Сумма элементов в каждой строке исходной матрицы: \n");
   for(i=0;i<n;i++) printf("%d ",sum[i]);  printf("\n");
/* Упорядочивание строк матрица по возрастанию сумм элементов */
   for(k=0;k<n-1;k++)
    for(i=0;i<n-k-1;i++)
     if (sum[i]>sum[i+1])
      {x=sum[i]; sum[i]=sum[i+1]; sum[i+1]=x;
       for(j=0;j<m;j++)
           {x=mat[i][j]; mat[i][j]=mat[i+1][j]; mat[i+1][j]=x;}
      }
/* Вывод матрицы после упорядочивания строк на экран */
   printf("Полученная матрица: \n");
   for(i=0;i<n;i++)
    {for(j=0;j<m;j++) printf("%d ",mat[i][j]);
```

```
    printf("\n"); }
/* Запись полученной матрицы в файл mat2.dat */
  ved=fopen("mat2.dat","w"); /* Открытие файла mat2.dat для записи */
  if (ved==NULL) return -1;  /* Файл не открылся */
  fprintf(ved,"%d %d ",n,m);
  for(i=0;i<n;i++)
   for(j=0;j<m;j++)
    fprintf(ved,"%d ",mat[i][j]);
  fclose(ved);
  return 1;
}

int main()  /* Основная программа */
{
 clrscr();
 switch (sozdmat1())
  {case -1: printf("\nФайл mat1.dat не создан. Конец работы."); exit();
   case  0: printf("\nФайл mat1.dat создан. \n"); break;
  }
 switch (sozdmat2())
  {case -1: printf("\nФайл mat1.dat не открыт. Конец работы."); exit();
   case  0: printf("\nФайл mat2.dat не создан. Конец работы."); exit();
   case  1: printf("\nФайл mat2.dat создан. "); break;
   }
}

/* Ниже приведена копия экрана с результатами работы программы.

Введите размеры матрицы (не более 10): 3 4
Введите целую матрицу (3*4) построчно:
5 6 7 8
1 2 3 4
9 9 9 9

Файл mat1.dat создан.
Исходная матрица:
5 6 7 8
1 2 3 4
9 9 9 9
Сумма элементов в каждой строке исходной матрицы:
26 10 36
Полученная матрица:
1 2 3 4
5 6 7 8
9 9 9 9

Файл mat2.dat создан.

*/
```

Пример 2. Анализ успеваемости студентов.

Для анализа успеваемости студентов при сдаче экзаменов создадим файл ved.dat. В файл занесем для каждой группы ее код, количество студентов и количество экзаменов. Напишем функцию sozdved() для создания требуемого файла. Будем предполагать, что

данная функция - часть сложной системы, все модули которой работают с файлом ved.dat, поэтому определение указателя на него сделаем глобальным.

При работе с файлом ved.dat очень часто будет нужно найти данные, относящиеся к одной группе. Чтобы не просматривать весь файл ved.dat от начала, надо организовать “прямой” доступ к информации указанной группы. Для этого создадим дополнительный файл tkg.dat, куда будем писать только коды групп. Указатель на этот файл, также как и на файл ved.dat, сделаем глобальным. Напишем функцию sozdtkg() для создания файла tkg.dat.

На данном этапе следует подробно проанализировать функцию sozdved(). Функции sozdtkg() и poisc() используют стандартные подпрограммы fputs, fgets и fseek, которые рассматриваются в следующих подразделах.

```
/* СИ . Работа с файлами . Пример. Описание программы .
 1. Функция sozdved() - создание файла ved.dat .
   В этот файл записывается информация о студенческих группах ;
   для каждой группы  : код(kg,7 байтов) , количество студентов
   (kst,2 байта) , кол-во экзаменов (kex,1 байт) . Итого: запишем в файл
   10 байтов на одну группу . Данные вводятся с клавиатуры в виде :
   <код>  <кол-во студентов>  <кол-во экзаменов> <ВК>
          .    .    .
   В конце ввода информации надо ввести EOF - нажать клавиши
   <Ctrl/Z> <ВК> .
```

Замечание. Вместо комбинации [Ctrl/Z] можно нажать клавишу [F6], код у них одинаковый.

```
 2. Функция sozdtkg() - создание файла tkg.dat .
   В этот файл записываются коды групп (код одной группы - 7 байтов).
   Просматривается уже имеющийся файл ved.dat , из него коды групп
   переписываются в отдельный файл tkg.dat (в том же порядке, как они
   записаны в ved.dat) .
 3. Функция  poisc() - для заданного кода группы находит и
   распечатывает информацию. Вначале код группы ищется в файле
   tkg.dat , затем с помощью функции fseek() "позицируется" на
   соответствующее место в файле ved.dat , откуда вся информация
   о требуемой группе распечатывается на экране.
 4. Основная функция main по очереди вызывает:
      1) функцию sozdved() ;
      2) функцию sozdtkg() ;
      3) функцию poisc() ;
   и обрабатывает коды возврата .
 5. Порядок работы с программой :
      Программа :     Input :
      Пользователь:  powt12 10 1<ВК>
                     powt13 20 2<ВК>
                     powt14 30 3<ВК>
                     <Ctrl/Z><ВК>
      Программа :     File "ved.dat" is created .
                     File "tkg.dat" is created .
                     Input name group :
      Пользователь:  powt13<ВК>
      Программа :     code : powt13 , kst : 20 , kex :2 .
                     Happy end!
        Рекомендуется посмотреть из редактора Турбо-Си созданные
     файлы ved.dat и tkg.dat. В нашем случае будет :
     ved.dat :    powt12 101powt13 202powt14 303
     tkg.dat :     powt12 powt13 powt14 .                          */
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
FILE *ved;
FILE *tkg;
/* Функция sozdved - создание файла "ved.dat"    */
int sozdved()
{ char kg[8];   /* Код группы */
 int kst,kex;  /*kst - кол-во студентов , kex - кол-во экзаменов */
 ved = fopen("ved.dat","w");   /* Открытие файла ved.dat для записи */
 if(ved==NULL) return -1;      /* Файл не открылся */
 printf("\n Input:\n");
 while(scanf("%s %d %d\n",kg,&kst,&kex) != EOF)
     {fprintf(ved,"%-7.7s%2d%1d",kg,kst,kex); }
 fclose(ved);
 return 0;
 }
 /* Функция sozdtkg - создание файла "tkg.dat" */
 int sozdtkg()
 { char kg[8];    /* Код группы */
  int kst,kex, i,d;
 tkg = fopen("tkg.dat","w");  if(tkg==NULL) return 1;
 ved = fopen("ved.dat","r");  if(ved==NULL) return 2;
 while(fscanf(ved,"%7s%2d%1d",kg,&kst,&kex) != EOF)
        {d=strlen(kg); for(i=d ; i<7 ; i++) kg[i]=' '; kg[7]='\0';
      fputs(kg,tkg);    }
      fclose(ved);   fclose(tkg);
      return 0;
      }
/*Функция poisc      */
int poisc()
{int i,kst,kex;
 long l;
 char zkg[20],kg[8];
 printf("\nInput name group: ");  fflush(stdin);  gets(zkg);
 l=strlen(zkg);
 for(i=l ; i<7 ; i++)   zkg[i]=' '; zkg[7]='\0';
 tkg = fopen("tkg.dat","r");  if(tkg==NULL) return 1;
 ved = fopen("ved.dat","r");  if(ved==NULL) return 2;
 for(i=0 ; (fgets(kg,8,tkg) != NULL) && (strcmp(zkg,kg) !=0) ; i++);
 l = i*10;
 if (fseek(ved,l,0)==0)
      {fscanf(ved,"%7s%2d%1d",kg,&kst,&kex);
   printf("\ncode: %7s, kst: %2d, kex: %1d",kg,kst,kex);
   return 0;}
 else return 3;
}
/* Главная программа  */
int main()
{
switch( sozdved() )
 {case -1: printf("\n File\"ved.dat\" not created . Abort ."); exit(1);
  case  0: printf("\n File\"ved.dat\" is created ."); break;  }
```

```
 switch( sozdtkg() )
  {case  1: printf("\n File\"tkg.dat\" not created . Abort ."); exit(1);
   case  2: printf("\n File\"ved.dat\" not opened . Abort ."); exit(1);
   case  0: printf("\n File\"tkg.dat\" is created ."); break;  }
  switch( poisc() )
  {case  1: printf("\n File\"tkg.dat\" not opened . Abort ."); exit(1);
   case  2: printf("\n File\"ved.dat\" not opened . Abort ."); exit(1);
   case  3: printf("\n Bad with funution \"fseek\"  "); exit(1);
   case  0:printf("\nHappy end !"); break;  }
 }
```

8.4.4.3 Чтение (запись) строки из (в) потока

Чтение. Прототип функции:

```
char     *fgets(char *string, int n, FILE *stream);
```

Запись. Прототип функции:

```
char     *fputs(char *string, FILE *stream);
```

Функция fgets() читает символы из потока stream в строку string. Функция заканчивает чтение, когда она либо прочтет n-1 символ, либо встретит символ новой строки. Последним символом, записанным в string, будет нулевой символ. Возвращаемое значение: при успехе - строка string, переданная как аргумент, а при ошибке или конце файла - NULL.

Функция fputs() копирует строку string, оканчивающуюся нулевым символом, в выходной поток stream; символ новой строки не добавляется. Возвращаемое значение: при успехе - последний записанный символ; в противном случае - EOF.

Пример. См. пример 2 из предыдущего подраздела 8.4.4.2. Следует подробно проанализировать работу функции sozdtkg().

8.4.4.4. Установка указателя файла в потоке (обработка файла с указанной позиции)

Прототипы функций:

```
int      fseek(FILE *stream, long offset, int fromwhere);
void     rewind(FILE *stream);
long     ftell(FILE *stream);
```

Функция fseek() устанавливает указатель файла, связанного со stream, на новую позицию, которая отстоит от места в файле, заданного параметром fromwhere, на количество байт, указанных в offset (с учетом знака).

Параметр fromwhere должен быть одной из величин: 0, 1, 2. Эти величины можно представлять тремя константами, определенными в stdio.h (см. таблицу ниже).

Fromwhere		Размещение в файле
Имя константы	значение	
SEEK_SET	0	Начало файла
SEEK_CUR	1	Текущее положение указателя файла
SEEK_END	2	Конец файла

Функция rewind() устанавливает указатель на начало файла, то есть вызов функции rewind(stream) эквивалентен по своему действию вызову функции fseek(stream, 0L, 0).

После вызова функции fseek() или rewind() следующей операцией с файлом, открытым для модификации, может быть как ввод, так и вывод.

Возвращаемое значение: fseek() и rewind() возвращают 0, если указатель успешно перемещен, и ненулевое значение при ошибке.

Функция ftell() возвращает текущее положение (смещение) указателя файла, указанного в stream. Смещение измеряется в байтах, считая от начала файла.

Пример 1.

```
 #include <stdio.h>
/* Функция filesize возвращает число байт в потоке */
long filesize(FILE *stream)
{long c,length;
  c=ftell(stream); /* Запоминаем текущее положение указателя в  c */
  fseek(stream,0L,SEEK_END); /*Устанавливаем указатель на конец файла */
  length=ftell(stream);
  fseek(stream,c,SEEK_SET);   /* Возвращаем текущее положение указателя "на место", которое сохранилось в  c  */
  return (length);
}
  int main()
{FILE *stream;
 char in[20];
 printf("\n Введите имя файла: ");   gets(in);
 stream=fopen(in,"r");
 if (stream != NULL)
    printf("Размер файла - %Ld байт",filesize(stream));
 else printf ("ошибка");
}
```

Пример 2. См. пример 2 из п.8.4.4.2. Следует подробно проанализировать работу функции poisc().

Пример 3.

```
/* Пример чередования печати в прямом и обратном направлении */
# include <stdio.h>
 int main()
{ FILE *fp;
   long offset = 0L;
  if ((fp = fopen ("f1.dat", "r")) = =0)
      printf("\nНе могу открыть файл ");
  else
    { while(fseek(fp, offset++, 0) = = 0)
        {putchar (getc(fp));
        if(fseek(fp, -(offset+3), 2) = = 0)   /* В зависимости от системы к offset
                        может понадобиться прибавлять не 3, а др. величину */
          putchar (getc(fp));       }
      fclose (fp);
    }
}
```

Использование функций ftell() и seek() для потока, открытого в текстовом режиме, может привести к неожиданным результатам из-за преобразования символов “возврат каретки - перевод строки”. Значение указателя, возвращаемое ftell(), может не отражать физического смещения в байтах текущей позиции потока. Правильное выполнение функции fseek() гарантируется только в случаях:

а) указатель устанавливается относительно базовых значений 0, 1, 2 со смещением 0;

б) указатель устанавливается относительно начала потока со значением смещения, возвращаемого функцией ftell().

8.5 ЛАБОРАТОРНАЯ РАБОТА №8 "ФАЙЛЫ"

Цель лабораторной работы "Файлы" - получить навыки проектирования структуры файлов и разработки программ, использующих файлы, а именно таких программ, которые читают данные из файла и выводят данные в файл.

Задание к лабораторной работе включает в себя оформление двух отдельных функций создания файла и обработки файла и основной программы, которая последовательно вызывает эти две функции. При необходимости набор функций может быть увеличен, что диктуется требованиями конкретной задачи.

8.5.1 Файлы. Список заданий

ЗАДАНИЕ. Написать программу создания и обработки файла.

N1.

Имеется таблица выигрышей международной лотереи журналистов (следует создать соответствующий файл). Проверить, является ли билет выигрышным.

Таблица состоит из чисел, размер чисел - от3 до 8 цифр. Билеты лотереи имеют номера из 8 цифр. Выигрышным является такой билет, для которого совпали все 8, или 7, или 6, или 5, или 4, или 3 последних цифры. Возможно, что на один билет выпадет сразу несколько выигрышей.

Пример. Имеем следующую таблицу:

87345103 - автомобиль
7123 - пылесос
103 - 50 руб.

Билет с номером 87345103 выиграл автомобиль и 50 руб.

N2.

Создать файл f, компоненты которого являются целыми числами. Никакая из компонент файла не равна 0. Файла f содержит равное число отрицательных и положительных чисел. Число компонентов файла f делится на 4.

Используя вспомогательный файл, переписать компоненты файла f в файл g так, чтобы в файле g числа шли в следующем порядке: ++ -- ++ -- … , то есть два положительных числа, два отрицательных числа и т.д.

N3.

Дан файл (создать его), в котором количество положительных чисел равно количеству отрицательных чисел и нет чисел, равных 0.

Создать новый файл, в котором знаки чисел чередуются: положительное число, отрицательное число и т.д. Допустимо использовать вспомогательный файл.

N4.

Дан файл f (создать его), компоненты которого являются целыми числами. Никакая из компонент файла не равна 0. Файла f содержит равное число отрицательных и положительных чисел.

Используя вспомогательный файл, переписать компоненты файла f в другой файл g так, чтобы в файле g числа шли сначала все положительные числа, а потом все отрицательные числа.

N5.

Создать файл f, компоненты которого являются целыми числами. Никакая из компонент файла не равна 0. Числа в файле записаны в следующем порядке: 5 положительных, 5 отрицательных и т.д. Число компонентов файла f делится на 20.

Переписать компоненты файла f в файл g так, чтобы в файле g числа шли в следующем порядке: ++ -- ++ -- … , то есть два положительных числа, два отрицательных числа и т.д. Допустимо использование вспомогательного файла.

N6.

Дан файл f (создать его), компонентами которого являются целые числа.

Получить новый файл из данного исключением повторных вхождений одного и того же числа.

N7.

Дан файл F (создать его), компонентами которого являются целые числа, не равные 0. Числа в файле записаны в следующем порядке: ++ -- ++ -- …, то есть два положительных числа, два отрицательных числа и т.д. Число компонентов файла F делится на 12.

Переписать компоненты файла F в файл H так, чтобы в файле H числа шли в следующем порядке:+++ --- +++ ..., то есть три положительных числа, три отрицательных чичла и т.д. Допустимо использование вспомогательного файла.

N8.

Дан файл F (создать его), компоненты которого - целые числа . Число их кратно 5.

Записать в новый файл G наибольшее из первых 5 компонент файла F, затем - наибольшее из следующих 5 компонент и т. д.

N9.

Дан файл F (создать его), компоненты которого являются целыми числами.

Записать в файл G все четные числа файла F , а в файл H - все нечетные числа файла F. Порядок следования чисел сохраняется.

N10.

Дан символьный файл (создать его), содержащий слова. Слова в тексте разделяются пробелами и знаками препинания.

Получить 2 наиболее часто встречающихся слова и число их повторений. Если таких слов более 2, выдать соответствующую информацию для каждого из них.

N11.

Создать символьный файл s, содержащий слова, разделенные пробелами.

Удалить из файла все однобуквенные слова и лишние пробелы. Полученный файл записать в новый файл f.

Пример.

Файл s: "стол и стул".

Файл f: "стол стул".

N12.

Дан символьный файл (создать его).

Найти самое длинное слово среди слов, вторая буква которых есть 'a'. Если таких слов с наибольшей длиной несколько, то найти все. Если таких слов нет, то сообщить об этом.

N13.

Создать символьный файл.

Подсчитать число вхождений в файл сочетаний 'ab' и 'cd'. Если таких сочетаний нет, то сообщить об этом.

N14.

Дан текстовый файл, содержащий программу на языке ПАСКАЛЬ (создать его).

Проверить эту программу на несоответствие числа открывающихся и закрывающихся круглых и операторных скобок.

N15.

Дан символьный файл (создать его). Файл состоит из слов, разделенных пробелами или знаками препинания.

Определить количество слов в файле и распечатать первое и последнее слово.

N16.

Даны (создать) два файла - упорядоченные по номерам телефонные справочники. Номер представляет собой 5-значное число, начинающееся с цифры 1 или с цифры 2. Формат записи справочника: Телефон ФИО Адрес .

Написать программу, строящую третий файл - общий упорядоченный справочник.

N17.

Дан файл "Товар" (создать его), содержащий сведения об экспортируемых товарах: указывается наименование товара; страна, экспортирующая товар; объем поставляемой партии товара в штуках.

Составить и записать в новый файл список стран, в которые экспортируется определенный товар, и найти общий объем экспорта этого товара. Исходный файл и полученный список распечатать.

N18.

Создать два файла f1 и f2. Файл f1 - это инвентарный файл, содержащий сведения о том, сколько изделий каких видов продукции хранится на складе. Файл f2 - это вспомогательный файл, содержащий сведения о том, насколько уменьшилось или увеличилось количество изделий по некоторым видам продукции. Файл F2 может содержать несколько сообщений по продукции одного вида или не содержать ни одного сообщения по продукции какого-то вида.

Обновить инвентарный файл f1 на основе вспомогательного файла f2, образовав новый файл f3. Все три файла распечатать.

N19.

Багаж каждого пассажира характеризуется количеством его вещей и общим весом этих вещей. Создать файл "Багаж", содержащий сведения о багаже нескольких пассажиров.

Упорядочить сведения о багаже, записанные в файле "Багаж", по убыванию общего веса багажа одного пассажира. Предполагается, что число пассажиров равно N и не слишком велико, то есть упорядочивание можно сделать в оперативной памяти.

N20.

Багаж каждого пассажира характеризуется количеством его вещей и общим весом этих вещей. Создать файл "Багаж", содержащий сведения о багаже нескольких пассажиров.

Найти число пассажиров, имеющих более 2-х вещей, и число пассажиров, количество вещей у которых превосходит среднее число вещей.

N21.

Создать файл "Кубики", содержащий сведения о кубиках: размер (длина ребра), цвет (допустимы 7 цветов радуги) и материал (дерево, металл, картон).

Найти количество кубиков каждого из возможных цветов и их суммарный объем.

N22.

Для каждого авиарейса в файле записано: номер рейса, название порта назначения, время отправления и прибытия, номера дней вылета. Указан порт назначения.

Напечатать и записать в другой файл номера рейсов и дни отправления рейсов до указанного порта.

N23.

Сведения о студентах (ФИО, группа, оценки за контрольные работы по пятибалльной системе) содержатся в файле (создать его).

Собрать в новый файл сведения о лучших студентах, не имеющих за все контрольные работы оценок ниже четырех баллов.

8.6 КОНТРОЛЬНЫЕ ВОПРОСЫ ПО МОДУЛЮ 8

8.6.1 Понятия "файл" и "поток".
8.6.2 Определение потока.
8.6.3 Стандартные потоки.
8.6.4 Нестандартные потоки. Основные этапы работы с ними.
8.6.5 Открытие потока.
8.6.6 Закрытие потока
8.6.7 Очистка потока.
8.6.8 Чтение (запись) символа из (в) потока.
8.6.9 Форматированный ввод (вывод) в(из) поток.
8.6.10 Чтение (запись) строки из (в) потока.
8.6.11 Обработка потока с определенной позиции.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Абрамов С.А., Зима Е.В. Начала информатики. – М.: Наука, 1989. – 256 с.
2. Абрамов С.А., Гнездилов Г.Г., Капустина Е.Н., Селюн М.И. Задачи по программированию – М.: Наука, 1988. – 224 с.
3. Березин Б.И., Березин С.Б. Начальный курс С и С++. - М.: ДИАЛОГ-МИФИ, 1996. - 288 с.
4. Гуденко Д.А., Петроченко Д.В. Сборник задач по программированию. – СПб.: Питер, 2003. – 475 с.
5. Егорова Е.В. Программирование на языке высокого уровня: учеб. пособие / Е.В.Егорова. - Барнаул: Изд-во АлтГТУ, 2008. - 165 с.
6. Егорова Е.В. Основные этапы решения задач на ЭВМ: Методические указания к лабораторным работам по курсу «Программирование на языке высокого уровня» / Алт.гос.техн.ун-т им.И.И.Ползунова. – Барнаул: Изд-во АлтГТУ, 2010. – 9 с.
7. Егорова Е.В. Программировании е на языке высокого уровня: учеб. пособие / Е.В.Егорова. - [Ч.1]. - Барнаул: Изд-во АлтГТУ, 2010. - 209с.
8. Касаткин А.И. Профессиональное программирование на языке Си. Управление ресурсами: Справ. пособ. - Мн.: Выш. шк., 1992. - 432с.
9. Керниган Б., Ритчи Д., Фьюэр А. Язык программирования Си. Задачи по языку Си. -М.: Финансы и статистика, 1989. - 279с.
10. Павловская Т.А. С/С++. Программирование на языке высокого уровня: [для вузов] / Т.А.Павловская. - СПб. [и др.]: Питер, 2010. - 460 с. - 25 экз.
11. Подбельский В.В. Программирование на языке Си: учеб. пособие для вузов / В.В.Подбельский, С.С.Фомин. - М.: Финансы и статистика, 2003. - 600с. - 51 экз.
12. Подбельский В.В. Язык Си++: учеб. пособие для вузов / В.В.Подбельский. - М.: Финансы и статистика, 2004. - 560с. - 40 экз.
13. Романовская Л.М., Русс Т.В., Свитковский С.Г. Программирование в среде Си для ПЭВМ ЕС. - М.: Финансы и статистика, 1991. - 350 с.
14. Страуструп Б. Язык программирования Си++. - М.: Радио и связь, 1991. - 352 с.
15. Шилдт Г. Полный справочник по С. – М.: Издательский дом "Вильямс", 2002. - 704 с.