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

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

О.А.Кравченко

Основы алгоритмизации и программирования

КУРС ЛЕКЦИЙ
по одноименной дисциплине для студентов специальности
1-40 01 02 «Информационные системы и технологии»

Гомель 2011

СОДЕРЖАНИЕ

1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ АЛГОРИТМИЗАЦИИ И ОБЬЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

1.1 Свойства модуля

Модульное программирование основано на понятии *модуля* — логически взаимосвязанной совокупности функциональных элементов, оформленных в виде отдельного программного элемента.

Модуль характеризуют:

- один вход и один выход — на входе программный модуль получает определенный набор исходных данных, выполняет содержательную обработку и возвращает один набор результатов, т. е. реализуется стандартный принцип IPO (Input — Process — Output) — *вход-процесс-выход;*
- функциональная завершенность — модуль выполняет перечень регламентированных операций для реализации каждой отдельной функции в полном составе, достаточных для завершения начатой обработки;
- логическая независимость — результат работы программного модуля зависит только от исходных данных, но не зависит от работы других модулей;
- слабые информационные связи с другими программными модулями — обмен информацией между модулями должен быть по возможности минимизирован;
- обозримый по размеру и сложности программный элемент.

Таким образом, модули содержат определение доступных для обработки данных, операции обработки данных, схемы взаимосвязи с другими модулями.

Каждый модуль состоит из спецификации и тела. Спецификации определяют правила использования модуля, а тело — способ реализации процесса обработки.

1.2 Модульная структура программных продуктов

Принципы модульного программирования программных продуктов во многом сходны с принципами нисходящего проектирования. Сначала определяются состав и подчиненность функций, а затем — набор программных модулей, реализующих эти функции.

Однотипные функции реализуются одними и теми же модулями. Функция верхнего уровня обеспечивается главным модулем; он управляет выполнением нижестоящих функций, которым соответствуют подчиненные модули.

При определении набора модулей, реализующих функции конкретного алгоритма, необходимо учитывать следующее:

- каждый модуль вызывается на выполнение вышестоящим модулем и, закончив работу, возвращает управление вызвавшему его модулю;

- принятие основных решений в алгоритме выносится на максимально "высокий" по иерархии уровень;
- для использования одной и той же функции в разных местах алгоритма создается один модуль, который вызывается на выполнение по мере необходимости.

В результате дальнейшей детализации алгоритма создается *функционально-модульная схема* (ФМС) алгоритма приложения, которая является основой для программирования (рисунок 1.1).

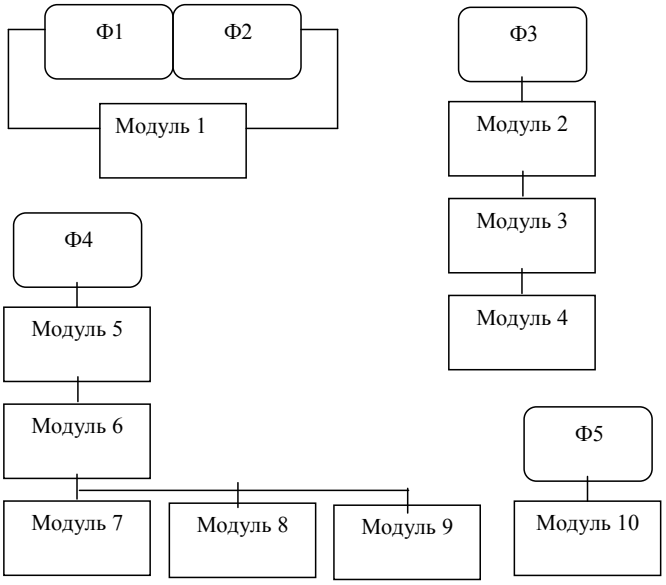


Рисунок 1.1 – Функционально-модульная структура приложения

Рисунок 1.1 демонстрирует следующие особенности распределения функций по модулям:

- некоторые функции могут выполняться с помощью одного и того же программного модуля (например, функции Ф1 и Ф2);
- функция Ф3 реализуется в виде последовательности выполнения программных модулей;
- функция Ф4 реализуется с помощью иерархии связанных модулей;
- модуль 6 управляет выбором на выполнение подчиненных модулей;
- функция Ф5 реализуется одним программным модулем.

Состав и вид программных модулей, их назначение и характер использования в программе в значительной степени определяются

инструментальными средствами, т. е. системой программирования, в которой эти модули должны реализовываться.

1.3 Основные понятия объектно-ориентированного программирования

Метод *объектно-ориентированного проектирования* основывается на:

- модели построения системы как совокупности объектов абстрактного типа данных;
- модульной структуре программ;
- нисходящем проектировании, используемом при выделении объектов.

Объектно-ориентированный подход использует следующие базовые понятия:

- объект;
- свойство объекта;
- метод обработки;
- событие;
- класс объектов.

Объект — совокупность свойств (параметров) определенных сущностей и методов их обработки (программных средств).

Объект содержит *инструкции* (программный код), определяющие действия, которые может выполнять объект, и обрабатываемые *данные*.

Свойство — характеристика объекта, его параметр. Все объекты наделены определенными свойствами, которые в совокупности выделяют объект из множества других объектов.

Объект обладает *качественной* определенностью, что позволяет выделить его из множества других объектов и обусловливает независимость создания и обработки от других объектов.

Например, объект можно представить перечислением присущих ему свойств:

ОБЪЕКТ_А (свойство-1, свойство-2,...., свойство-k).

Свойства объектов различных классов могут "пересекаться", т.е. возможны объекты, обладающие одинаковыми свойствами:

ОБЪЕКТ_В (...свойство - n, свойство-m,...свойство - r,...)

ОБЪЕКТ_С (...свойство - n, , свойство - r,...).

Одним из свойств объекта являются метод его обработки.

Метод — программа действий над объектом или его свойствами.

Метод рассматривается как программный код, связанный с определенным объектом; осуществляет преобразование свойств, изменяет поведение объекта.

Объект может обладать набором заранее определенных встроенных методов обработки, либо созданных пользователем или заимствованных в стандартных библиотеках, которые выполняются при наступлении *заранее определенных событий*, например, однократное нажатие левой кнопки мыши, вход в поле ввода, выход из поля ввода, нажатие определенной клавиши и т.п.

По мере развития систем обработки данных создаются *стандартные* библиотеки методов, в состав которых включаются типизированные методы обработки объектов определенного класса (аналог — стандартные подпрограммы обработки данных при структурном подходе), которые можно заимствовать для различных объектов.

Событие — изменение состояния объекта

Класс — совокупность объектов, характеризующихся общностью применяемых методов обработки или свойств.

Внешние события генерируются пользователем (например, клавиатурный ввод или нажатие кнопки мыши, выбор пункта меню, запуск макроса); *внутренние события* генерируются системой.

Объекты могут объединяться в классы (группы или наборы — в различных программных системах возможна другая терминология).

Один объект может выступать объединением вложенных в него по иерархии других объектов.

Схематично связь основных понятий объектно-ориентированного программирования представим следующим образом (рисунок 1.2):

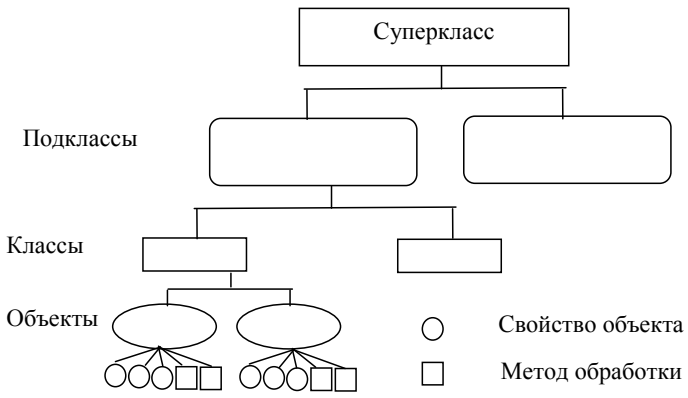


Рисунок 1.2 – Соотношение основных понятий объектно-ориентированного подхода

В объектно-ориентированном программировании используется следующий формат записи работы с объектами:

ОБЪЕКТ. МЕТОД

ОБЪЕКТ.СВОЙСТВО.МЕТОД

Программный продукт, созданный с помощью инструментальных средств объектно-ориентированного программирования, содержит объекты с их характерными свойствами, для которых разработан графический интерфейс пользователя. Как правило, работа с программным продуктом осуществляется с помощью экранной формы, с объектами управления, которые содержат методы обработки, вызываемые при наступлении определенных событий. Экранные формы также используются для выполнения заданий и перехода от одного компонента программного продукта к другому. Каждый объект управления обладает определенными свойствами, значения которых могут изменяться. Для объектов управления уточняется перечень событий и создаются пользовательские методы обработки — программный код на языке программирования в виде событийных процедур.

1.4 Понятие об алгоритме

Решение любой задачи осуществляется по определенному плану, называемому алгоритмом.

Алгоритм - понятное и точное предписание исполнителю совершить последовательность действий, направленных на достижение указанной цели или на решение поставленной задачи.

Для раскрытия приведенного определения составим простую классификацию всех окружающих нас правил. В своей жизни мы постоянно имеем дело с различными правилами – разрешавшими, запрещающими предписывающими.

Примерами разрешающих правил могут быть следующие: "Каждый студент может получать повышенную стипендию", "Стоянка разрешена", "На дискотеку вход свободный". Никакое разрешающее правило не является алгоритмом. Это же утверждение справедливо и для запрещающих правил, примерами которых являются: "При плохой успеваемости студент не может получать повышенную стипендию", "Во всех корпусах университета курение запрещено", "Стоянка запрещена".

К алгоритмам относятся только предписывающие правила. Например, кулинарный рецепт приготовления какого-либо блюда является алгоритмом. Алгоритмом является также инструкция по включению какого-либо прибора, например, компьютера, составленная из предписывающих правил.

Но не всякое предписывающее правило можно считать алгоритмом. Так, например, предписывающее правило "Взвейтесь кострами, синие ночи!" не является алгоритмом, т.к. предполагаемый исполнитель – "синие ночи" желаемого действия (взвейтесь кострами) не в состоянии выполнить.

Таким образом, алгоритм – последовательность предписывавших правил понятно и точно указывающих исполнителю, какую последовательность

действий он должен исполнить по переработке исходных данных в искомые результаты.

1.5 Свойства алгоритма

Рассмотрим несколько свойств алгоритма.

Точность определяется как свойство, согласно которому исполнителю точно известно, какая команда должна выполняться следующей.

Для иллюстрации важности этого свойства рассмотрим предписывающее правило: "Уходя, гасите свет". Является ли это правило алгоритмом? С одной стороны рассматриваемое правило предписывает вполне определенному исполнителю (человеку, уходящему из данного помещения) выполнить в определенный момент (в момент ухода) вполне определенное действие (выключить свет). Исходя из этого, можно считать рассматриваемое правило алгоритмом. Но, с другой стороны, для человека, "страдающего" излишним формализмом, указанное правило не вполне точно определяет последовательность необходимых действия, т.к. для этого человека остается неясным, должен ли он, покидая помещение, выключить свет, если в помещении остаются люди. Анализируемый пример показывает на существование серьезной проблемы –то, что однозначно (точно) понимается одним исполнителем, может совсем не так восприниматься другим.

Понятность – это свойство состоит в том, что каждая команда алгоритма долина входить в систему команд исполнителя. Система команд исполнителя – это совокупность команд, которые могут быть выполнены исполнителем. Например, система команд микрокалькулятора описывается в его паспорте, в неё входят такие команды, как четыре арифметические операции, запись числа в ячейку памяти, вычисление элементарных функции и т.д.

Дискретность алгоритма заключается в том, что предписание представляет собой последовательность четко выраженных отдельных команд. Выполнив одну команду, исполнитель выполняет следующую команду и г.д. Бессмысленно ставить вопрос – что делает исполнитель между выполнением двух последовательных команд – таких моментов просто нет.

Массовость – для каждого алгоритма существует класс объектов, допустимых в качестве исходных данных. Так, например, алгоритм нахождения наибольшего общего делителя двух целых чисел применим для любой пары целых чисел. Из двух алгоритмов, разработанных для решения одной задачи, более ценным является тот, у которого шире класс допустимых исходных данных. Можно сказать, что у такого алгоритма массовость больше.

Результативность – это свойство состоит в том, что результат выполнения алгоритма исполнитель должен получить, выполнив конечное число действий. Если для каких-то допустимых исходных данных исполнитель не может получить результат выполнения алгоритма за конечное число шагов, то

говорят, что алгоритм не применим для этих исходных данных. Так, например, алгоритм получения точного значения частного при делении уголком не применим для чисел 20 и 3. При этом, класс допустимых исходных данных для указанного алгоритма – пара целых чисел, второе из которых не равно 0.

1.6 Средства записи алгоритма

Для записи алгоритмов используются различные способы – словесная, графическая схема алгоритма (блок-схема), алгоритмический язык, языки программирования. Рассмотрим причины указанного разнообразия и области употребления каждого способа записи.

Алгоритм разрабатывается и записывается для какого-то конкретного исполнителя. Можно выделить два класса исполнителей – люди и вычислительные устройства. К последним относятся роботы, микрокалькуляторы, компьютеры и т.д. Так как разные исполнители имеют различные системы команд, то в записях алгоритмов должны использоваться команды, зависящие от конкретного исполнителя (свойство понятности:). Кроме наличия различных систем команд каждый исполнитель отличается своим способом восприятия команд алгоритма – человек может прочитать или выслушать данные ему для исполнения команды, микрокалькулятор воспринимает команды, вводимые в него нажатием клавиши, в компьютеры команды можно вводить нажатием клавиш на клавиатуре или с помощью указателя мыши.

Итак, причина разнообразия средства записи алгоритмов – различия в системах команд исполнителей и в способах восприятия исполнителем команд алгоритма.

Язык программирования – это способ записи алгоритма, ориентированный на исполнение его системой программирования компьютера. Дня записи алгоритмов, предназначенных для безмашинного исполнения, можно использовать естественный язык (например, русский), дополненный математическими символами – это, так называемая, словесная запись алгоритма.

Наиболее часто при разработке и документирования программы, лежащие в ее основе алгоритмы изображаются в виде графических схем алгоритма или блок-схем.

Алгоритм большой сложности обычно представляется с помощью схем двух видов:

- *обобщенной схемы алгоритма* — раскрывает общий принцип функционирования алгоритма и основные логические связи между отдельными модулями на уровне обработки информации (ввод и редактирование данных, вычисления, печать результатов и т.п.);

- *детальной схемы алгоритма* — представляет содержание каждого элемента обобщенной схемы с использованием управляющих структур в блок-схемах алгоритма, псевдокода либо алгоритмических языков высокого уровня.

1.7 Графические схемы алгоритмов

Перед тем как записать алгоритм в виде программы, его, как правило, представляют в виде схемы алгоритма. **А не наоборот, как пытаются сделать многие начинающие!** Схема алгоритма, если она правильно составлена, способствует правильному и более быстрому написанию программы!

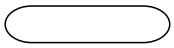
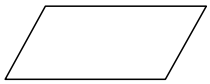
Правила выполнения схем алгоритмов регламентированы ГОСТ 19.701 – 90 (ИСО 5807 – 85) [1], входящего в единую систему программной документации (ЕСПД) под названием "Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения". Согласно этому стандарту схема алгоритма – это графическое представление метода решения задачи, в котором используются символы для отображения операций, данных, потока, оборудования и т.д.

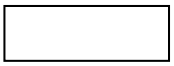
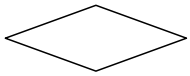
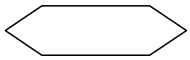
Схема алгоритма состоит из:

1) символов процесса, указывающих фактические операции обработки данных (включая символы, определяющие путь, которого следует придерживаться с учетом логических условий);
2) линейных символов, указывающих поток управления;
3) специальных символов, используемых для облегчения написания и чтения схемы.

Обозначение наиболее часто употребляемых символов и описание отображаемых ими действий приведено в таблице 1.1.

Таблица 1.1 – Символы схемы алгоритма

Название символа	**Обозначение**	**Значение**
Терминатор		Начало или конец схемы алгоритма
Данные		Ввод или вывод данных; носитель данных не определен

Процесс		Обработка данных любого вида, приводящая к изменению значения, формы или размещения информации
Предопределенный процесс		Использование подпрограммы (или модуля)
Решение		Проверка условия и выбор одного из нескольких альтернативных выходов
Подготовка		Модификация команды, группы команд, индексного регистра (создание цикла)
Линия		Отображает поток данных и управления. При необходимости могут быть добавлены стрелки - указатели
Соединитель		Обрыв линии и продолжение ее в другом месте. Соответствующие символы должны иметь одно и то же уникальное обозначение.
Комментарий		Пояснения к выполняемым действиям. Располагается около ограничивающей фигуры (символа или блока символов, обведенных пунктирной линией)

Символы могут быть вычерчены в любой ориентации, но, по - возможности, предпочтительной является горизонтальная ориентация.

Текст, необходимый для понимания функции данного символа, следует помещать внутри данного символа и записывать слева направо и сверху вниз независимо от направления потока. Если объем текста превышает размеры символа, нужно использовать символ комментария. В схемах может использоваться идентификатор символов (например, в виде номера), которым можно воспользоваться в справочных целях в других элементах документации (или для ссылки на символ). Идентификатор символа должен располагаться слева над символом.

Правила выполнения соединений. Потоки данных или потоки управления в схемах показываются линиями. Направление потока слева направо и сверху вниз считается стандартным. Если необходимо внести большую ясность в схему (например, при соединениях), на линиях используются стрелки. Если направление потока отличается от стандартного, то стрелки должны указывать это направление.

В схемах следует избегать пересечения линий. Пересекающиеся линии не имеют логической связи между собой, поэтому изменения направления потока в точках пересечения не допускаются. Две или более входящие линии могут объединяться в одну исходящую.

Линии в схемах должны подходить к символу либо слева, либо сверху, а исходить либо справа, либо снизу. Линии должны быть направлены к центру символа. При необходимости линии в схемах нужно разрывать во избежание излишних пересечений или слишком длинных линий, а также, если схема состоит из нескольких страниц. Соединитель в начале разрыва называется внешним соединителем, а соединитель в конце разрыва – внутренним соединителем. Совместно с символом комментария можно указать, с какой страницы, или на какую страницу схемы совершается переход.

Примеры соединителей приведены на рисунке 1.3.

Рисунок 1.3 – Виды соединителей

Несколько выходов из символа можно показывать:

- несколькими линиями от данного символа к другим символам;
- одной линией от данного символа, которая затем разветвляется в соответствующее число линий.

Примеры изображения выходов из символа приведены на рисунке 1.4.:

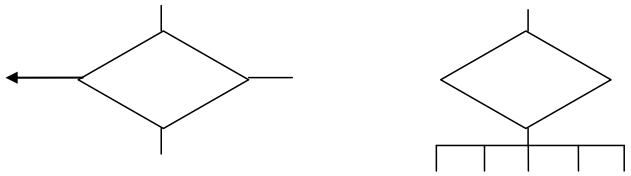


Рисунок 1.4 – Изображение выходов из символа

Каждый выход из символа нужно пометить значениями соответствующих условий, чтобы показать логический путь, который он представляет, с тем, чтобы эти условия и соответствующие ссылки были идентифицированы.

Примеры идентификации ссылок приведены на рисунке 1.5.

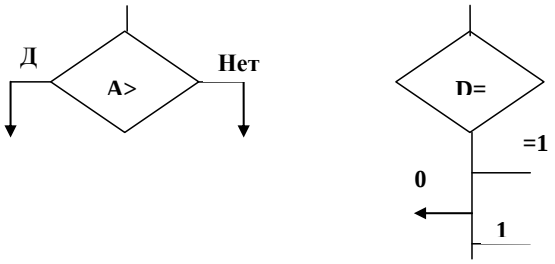


Рисунок 1.5 – примеры идентификации ссылок

При всем разнообразии структур алгоритмов можно выделить четыре типовых структуры (рисунок 1.6), из которых как из кирпичиков, можно построить здание алгоритма любой сложности: **следование**, **ветвление**, **цикл** и **выбор**.

Типовые структуры хороши тем, что все они имеют одну точку входа и одну точку выхода. Это упрощает просмотр и понимание алгоритма и сокращает количество возможных ошибок, сделанных при составлении схемы и время на их поиск. Типовые структуры могут быть вложенными друг в друга, например, символ, входящий в структуру *следование* может включать в себя структуры *ветвление* или *цикл*, а каждая из ветвей структур *ветвление* или *выбор* может содержать структуры *следование*, *цикл* или еще одно *ветвление* и *выбор*.

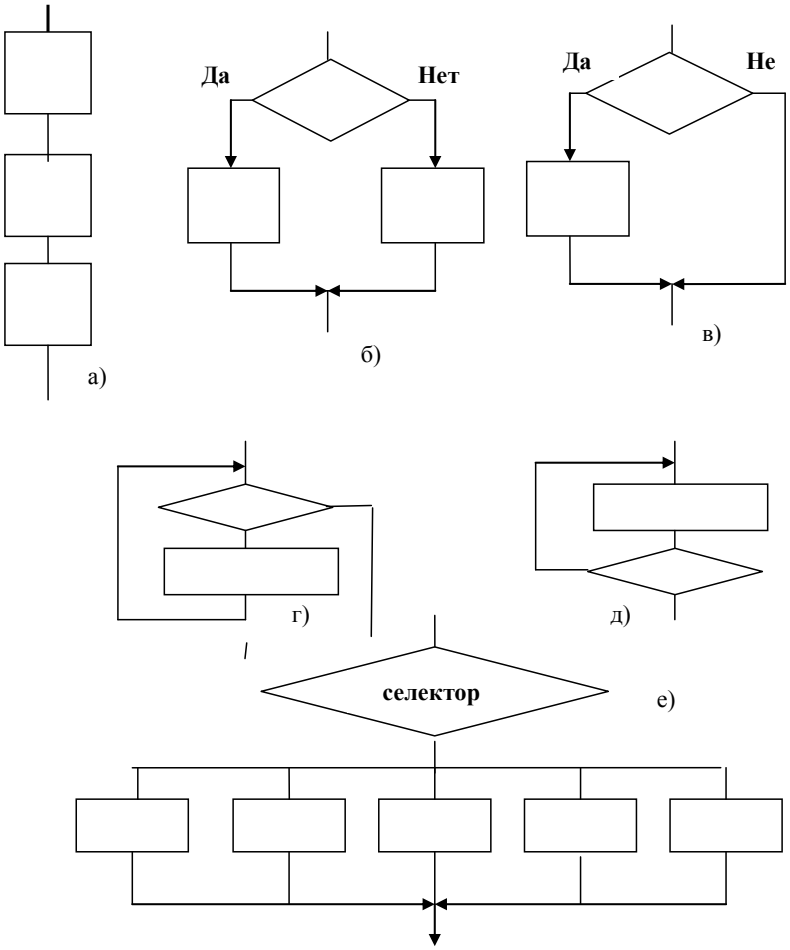


Рисунок1.6 – типовые структуры алгоритмов: а) - следование; б, в) – ветвление (полное и неполное), г) – цикл с предусловием; д) – цикл с постусловием, е) — выбор

1.8 Типы алгоритмов

На основе перечисленных в подразделе 1.7 структур строятся следующие типы алгоритмов:

- **линейный** (на основе структуры следование); характеризуется тем, что все действия, определяемые символами, входящими в схему, выполняются последовательно, в порядке их написания;

- **разветвляющийся** (на основе структур ветвление и выбор): характеризуется тем, что в ходе выполнения, решение задачи идет только по одному из имеющихся направлений, выбор которого зависит от выполнения заданного условия:
- **циклический** (на основе структуры цикл): характеризуется многократным повторением определенной группы действий.

В последующих разделах на конкретных примерах рассматриваются проекты в системе программирования С, основанные на всех вышеперечисленных типах алгоритмов.

2 СТРУКТУРА ПРОГРАММЫ

Программа на языке Си состоит из директив препроцессора, объявлений глобальных переменных, комментариев, одной главной функции (main) и, возможно, ряда неглавных функций.

Следующий короткий пример иллюстрирует структуру программы на Си.

```
/*Пример программы на языке Си*/
#include <stdio.h>                    // директива препроцессора
main ()                               // заголовок функции main
{                                     // начало тела функции main
      char name[20];                  // объявление переменной
      printf(“Введите Ваше имя: ”);   // вывод сообщения
      scanf(“%s”,name);               // ввод переменной
      printf("Здравствуйте, %s!\n",name);
      return(0);
}                                     // конец тела функции
```

Комментарии предназначены для пояснения назначения переменных и действий программы. Комментарий является частью кода, который игнорируется компилятором. Могут вставляться в любое место программы, где допускаются пробелы. Существует два способа вставки комментариев в программу:

а) с помощью открывающей скобки комментария “/*” и закрывающей скобки комментария “*/”. Текст, который находится между парой таких скобок является комментарием. Оформленный таким образом комментарий может занимать более одной строки;

б) с помощью пары символов “//”. Текст, записанный после данной пары символов до конца строки является комментарием. С помощью этого способа можно вставлять только однострочные комментарии.

Директива #include сообщает компилятору, что необходимо включить в текст программы содержимое файла, имя которого указано в угловых скобках либо в кавычках. Такой файл называется заголовочным, имеет расширение .h (от англ. header – заголовок). Фактически с помощью директивы #include к программе подключаются необходимые библиотеки функций. Например, в заголовочном файле stdio.h определяются некоторые макросы и переменные, используемые библиотекой ввода-вывода.

Существует большое количество стандартных функций. Они объединены в группы и хранятся в отдельных библиотеках по тематическому принципу (функции для математических расчетов, ввода-вывода, обработки строк и т.д.). Некоторые стандартные библиотеки подключаются автоматически. Подключение других библиотек нужно делать вручную, с помощью директивы #include <имя файла>.

Программа на языке Си состоит из отдельных подпрограмм (функций). В общем виде функция имеет вид:

```
тип_значения имя_функции (параметры)
{
    //тело функции
}
```

В рассматриваемом простом примере программа состоит из одной функции main(), не имеющей параметров. Функция main () присутствует в каждой программе, это основная функция, с нее начинается выполнение программы.

Функция main() по умолчанию должна возвращать целое значение. Значение, возвращаемое функцией, записывается с помощью оператора return. В примере возвращается целое значение 0, сигнализирующее операционной системе о нормальном завершении программы.

Тело функции – это блок, заключенный в фигурные скобки. Он может содержать объявления данных и операторы обработки данных. Каждый оператор заканчивается символом “;”.

3ТИПЫ ДАННЫХ

Основная цель любой программы состоит в обработке данных. Данные различного типа хранятся и обрабатываются по-разному. В любом алгоритмическом языке каждая константа, переменная, результат вычисления выражения или функции должны иметь определенный тип. Тип данных определяет:

• *множество значений,* которые могут принимать величины этого типа;

• *внутреннее представление* данных в памяти компьютера;

• *операции и функции,* которые можно применять к величинам этого типа.

Исходя из этих характеристик, программист выбирает тип каждой величины, используемой в программе для представления реальных объектов. **Обязательное описание типа** позволяет компилятору производить проверку допустимости различных конструкций программы. От типа величины зависят машинные команды, которые будут использоваться для обработки данных. Все типы языка С++ можно разделить на *основные* и ***составные***. В языке С++ определено шесть основных типов данных для представления *целых, вещественных, символьных* и *логических* величин. На основе этих типов программист может вводить описание составных типов. К ним относятся *массивы, перечисления, функции, структуры, ссылки, указатели, объединения* и *классы*.

Основные ***(стандартные)*** типы данных часто **называют *арифметическими,*** поскольку их можно использовать в арифметических операциях. Для описания основных типов определены следующие ключевые слова:

int (целый); ***char*** (символьный); ***wchar_t*** (расширенный символьный); ***bool*** (логический); ***float*** (вещественный); ***double*** (вещественный с двойной точностью).

Первые четыре типа называют целочисленными *(целыми),* последние два - *типами с плавающей точкой.* Код, который формирует компилятор для обработки целых величин, отличается от кода для величин с плавающей точкой.

Существует четыре ***спецификатора типа***, уточняющих внутреннее представление и диапазон значений стандартных типов:***short*** (короткий);***long*** (длинный);***signed*** (знаковый);***unsigned*** (беззнаковый).Диапазоны значение и объем занимаемой памяти для простых типов приведены в таблице 3.1.

Таблица 3.1 – Диапазоны значений простых типов данных для IBM PC

Тип	**Диапазон значений**	**Размер(байт)**
bool	true и false	1
signed char	-128 ... 127	1
unsigned char	0 ... 255	1
signed short int	-32 768 ... 32 767	2
unsigned short int	0 ... 65 535	2
signed long int	-2 147 483 648 ... 2 147 483 647	4
unsigned long int	0 ... 4 294 967 295	4
float	3.4e-38 ... 3.4e+38	4
double	1.7e-308 ... 1.7e+308	8
long double	3.4e-4932 ... 3.4e+4932	10

3.2 Целый тип (int)

Размер типа int не определяется стандартом, а зависит от компьютера и компилятора. Для 16-разрядного процессора под величины этого типа отводится 2 байта, для 32-разрядного — 4 байта. Спецификатор *short* перед именем типа указывает компилятору, что под число требуется отвести 2 байта независимо от разрядности процессора. Спецификатор *long* означает, что целая величина будет занимать 4 байта. Таким образом, на 16-разрядном компьютере эквиваленты *int* и *short int*, а на 32-разрядном — *int* и *long int*. Внутреннее представление величины целого типа — целое число в двоичном коде. При использовании спецификатора *signed* старший бит числа интерпретируется как знаковый (0 - положительное число, 1 - отрицательное). Спецификатор *unsigned* позволяет представлять только положительные числа, поскольку старший разряд рассматривается как часть кода числа. Таким образом, диапазон значений типа *int* зависит от спецификаторов. Диапазоны значений величин целого типа с различными спецификаторами для IBM PC-совместимых компьютеров приведены в таблице. По умолчанию все целочисленные типы считаются знаковыми, то есть спецификатор *signed* можно опускать. Константам, встречающимся в программе, приписывается тот или иной

тип в соответствии с их видом. Если этот тип по каким-либо причинам не устраивает программиста, он может явно указать требуемый тип с помощью суффиксов *L, l* (*long*) и *U, u* (*unsigned*). Например, константа *32L* будет иметь тип *long* и занимать 4 байта. Можно использовать суффиксы *L* и *U* одновременно, например, *OX22UL* или *05Lu*.

Примечание. Типы *short int*, *long int*, *signed int* и *unsigned int* можно сокращать до *short*, *long*, *signed* и *unsigned* соответственно.

3.3 Символьный тип (char)

Под величину символьного типа отводится количество байт, достаточное для размещения любого символа из набора символов для данного компьютера, что и обусловило название типа. Как правило, это 1 байт. Тип *char*, как и другие целые типы, может быть со знаком или без знака. В величинах со знаком можно хранить значения в диапазоне от -128 до 127. При использовании спецификатора *unsigned* значения могут находиться в пределах от 0 до 255. Этого достаточно для хранения любого символа из 256-символьного набора ASCII. Величины типа *char* применяются также для хранения целых чисел, не превышающих границы указанных диапазонов.

3.4 Расширенный символьный тип (wchar_t)

Тип *wchar_t* предназначен для работы *с* набором символов, для кодировки которых недостаточно 1 байта, например, Unicode. Размер этого типа зависит от реализации; как правило, он соответствует типу *short*. Строковые константы типа *wchar_t* записываются с префиксом L, например, L "Gates".

3.5 Логический тип (bool)

Величины логического типа могут принимать только значения *true* и *false*, являющиеся зарезервированными словами. Внутренняя форма представления значения *false* — 0 (нуль). Любое другое значение интерпретируется как *true*. При преобразовании к целому типу *true* имеет значение 1.

3.6 Типы с плавающей точкой (float, double и long double)

Стандарт C++ определяет три типа данных для хранения вещественных значений: *float, double* и *long double*.Типы данных с плавающей точкой хранятся в памяти компьютера иначе, чем целочисленные. Внутреннее представление вещественного числа состоит из двух частей — мантиссы и порядка. В IBM PC-совместимых компьютерах величины типа float занимают 4 байта, из которых один двоичный разряд отводится под знак мантиссы, 8 разрядов под порядок и 23 под мантиссу. Мантисса — это число, большее 1.0, но меньшее 2.0. Поскольку старшая цифра мантиссы всегда равна 1, она не хранится. Для величин типа *double*, занимающих 8 байт, под порядок и мантиссу отводится 11 и 52 разряда соответственно. Длина мантиссы определяет точность числа, а длина порядка — его диапазон. Как можно видеть из табл., при одинаковом количестве байт, отводимом под величины типа float и long int, диапазоны их допустимых значений сильно различаются из-за внутренней формы представления.

Спецификатор *long* перед именем типа *double* указывает, что под величину отводится 10 байт. Константы с плавающей точкой имеют по умолчанию тип *double*. Можно явно указать тип константы с помощью суффиксов *F, f (float)* и *L, l (long).* Например, константа *2E+6L* будет иметь тип *long double*, а константа *1.82f* — тип *float*.Для вещественных типов в таблице приведены абсолютные величины минимальных и максимальных значений. Для написания переносимых на различные платформы программ нельзя делать предположений о размере типа int. Для его получения необходимо пользоваться операцией *sizeof*, результатом которой является размер типа/в байтах. Например, для операционной системы MS-DOS *sizeof (int*) даст в результате 2, а для Windows 9X или OS/2 результатом будет 4. В стандарте ANSI диапазоны значений для основных типов не задаются, определяются только соотношения между их размерами, например:

sizeof(float) ≤ sizeof(double) ≤ sizeof(long double)

sizeof(char) ≤ sizeof(short) ≤ sizeof(int) ≤ sizeof(long)

Различные виды целых и вещественных типов, различающиеся диапазоном и точностью представления данных, введены для того, чтобы дать программисту возможность наиболее эффективно использовать возможности конкретной аппаратуры, поскольку от выбора типа зависит скорость вычислений и объем памяти. Но оптимизированная для компьютеров какого-либо одного типа программа может стать не переносимой на другие платформы, поэтому в общем случае следует избегать зависимостей от конкретных характеристик типов данных.

3.7 Тип Void

К основным типам языка относится тип также *void*. Множество значений этого типа *пусто*. Он используется:

- для определения функций, которые не возвращают значения,
- для указания пустого списка аргументов функции.

3.8 Переменные и константы

Переменная - это величина, которая имеет имя и которая может изменять свои значения в процессе выполнения программы.

Примеры объявления переменных:

```
int q, w, r;
float t1, u;
shot b, j;
long r, u, p, g;
char s34;
unsigned w234;
double b_1, G6;
enum seasons{spring, summer, autumn, winter} m. n. i;
```

Переменные можно инициализировать, т.е. определять их значения при объявлении с помощью оператора объявления:

имя_типа имя_переменной = значение;

Например:

```
float a=-2.4e4;
int b=456;
```

Константа – это величина, которая не изменяет своего значения в процессе выполнения программы. Константы могут быть неименованными и именованными.

Неименованные константы - это обычные константы, представленные в выражении своими значениями, например, числами (-3.1, 456, 2.123e-2), строкой символов ("Поздравляю с поступлением в университет!").

Имя и значение ***именованной константы*** определяются с помощью спецификатора ***const*** . Форма объявления именованной константы :

const тип имя_константы = выражение;

Например:

```
const float k1=2.48567,
```

```
        k2= 1/k1;
    const char c='*';
    const char *f="Маша+Ваня";
    const char s[]="г. Гомель";
```

Типы констант:

вещественные;

целые;

длинные целые, в конце записи которых добавляется буква L, например, 4567382L;

беззнаковые, в конце записи которых добавляется буква U;

восьмеричные, в которых перед первой значащей цифрой записывается нуль (0), например, 071;

шестнадцатеричные, в которых перед первой значащей цифрой записывается пара символов нуль-икс (0X), например, 0X2D1;

символьные, в которых единственный символ заключен в одинарные кавычки, например, '2', '&', 'y';

строковые, представляющие собой последовательность символов, заключенные в двойные кавычки, например, "Гомельский Государственный технический университет им. П.О.Сухого".

4 ОПЕРАЦИИ И ВЫРАЖЕНИЯ

4.1 Основные понятия

Операции осуществляют пересылку и преобразование данных в программах. Обозначаются операции одним или несколькими символами, например +, -, >> и т.д. Данные, участвующие в операции, называются операндами. В качестве операндов могут использоваться переменные, константы, выражения.

Выражение – это строка, составленная по определенным правилам из операндов, операций и предназначенная для получения значения. Например 3+6/2*d. ***Выражение*** называется ***константным***, если его операндами являются только константы, например, 67-234.

По числу операндов, участвующих в операции, различают ***унарные*** (один операнд) и ***бинарные*** (два операнда) операции. Пример унарной операции – унарный минус (изменение знака числа); пример бинарной операции -*(умножение).

По типу выполняемой операции различают *арифметические, сдвига, поразрядные логические, логические, операции отношения* и др.

4.2 Операция присваивания и оператор присваивания

Строку, содержащую знак “=” (присвоить) называют *оператором присваивания*.

Общий вид оператора присваивания:

P = V;,

где V- выражение, P – переменная.

Оператор присваивания выполняется следующим образом: вычисляется значение выражения V; это значение присваивается переменной P.

Например оператор a = a+1; означает: взять текущее значение переменной a, прибавить к нему единицу и полученный результат сохранить в переменной a.

Язык Си допускает множественное присваивание – присваивание одного значения набору переменных в одном операторе. При этом операции выполняются справа налево

Пример:

a = b = c = d = 1;

Это эквивалентно следующей последовательности операторов присваивания:

d = 1;
c = d;
b = c;
a = b.

Также в языке Си существует операция комбинированного присваивания, которая записывается в виде:

V1 OP = V, что эквивалентно V1 = V1 OP V,

где V1 – переменная, V – выражение, OP – одна из операций, задаваемых знаками *, /, +, -, %, >>, <<, &, ^.

Пример:

a += b – 6 эквивалентно a = a + (b – 6).

4.3 Арифметические операции

Арифметические операции приведены в таблице 5.1.

Таблица 5.1 – Арифметические операции

Знак операции	**Название**	**Описание**
-	унарный минус	изменение знака числа
*, /, %	умножение, деление, остаток от деления	умножение, деление, остаток от деления одного числа на другое
+, -	сложение, вычитание	сложение, вычитание чисел
++, --	инкремент и декремент	увеличение и уменьшение на единицу

Операции записаны в порядке уменьшения приоритета. Сначала выполняются операции с более высоким приоритетом, затем с более низким. Операции с одинаковым приоритетом выполняются слева направо. Порядок выполнения операций можно изменять с помощью скобок.

Пример:

2 + 2 * 2 = 6;

(2 + 2) * 2 = 8.

Специфическими для Си являются операции *определения остатка*(%), *инкремента* (++) и *декремента* (--).

Операция % (деление по модулю) бинарная, применима к целочисленным операндам. Результат – остаток от деления целых чисел.

Примеры:

6/4 → 1; 3/4 → 0; 3%4 → 3;

6%4 → 2; 9%2 → 1; 9%3 → 0;

Операции инкремента (++) и декремента (--) унарные, прибавляют или вычитают единицу из значения своего операнда. Могут быть использованы перед или после своего операнда (в префиксной и постфиксной форме соответственно).

Префиксные и постфиксные операции выполняются по разному в выражениях.

Например, выражение b = ++a означает выполнение двух операций в следующем порядке: a = a+1; b = a, а выражение b = a++ означает выполнение тех же операций, но в другом порядке: b = a; a = a+1.

Операция декремента выполняется аналогично.

4.4 Математические функции

Прототипы функций хранятся в заголовочном файле *math.h*.

Для того, чтобы получить данный перечень, необходимо ввести в тексте программы имя библиотеки *math* и нажать Ctrl+F1. Появится перечень слов с выделенным именем библиотеки. Нажать *Enter*.

Для того чтобы прочитать описание конкретной функции, необходимо выделить имя функции и нажать *Enter* или выполнить двойной щелчок мышью на имени функции.

Таблица 4.2 – Описание некоторых математических функций

Обращение к функции	**Тип аргументов**	**Тип значения**	**Описание**
abs(x)	int	int	Нахождение |x|
fabs(x)	double	double	-//-

labs(x)	long int	long int	-//-
acos(x)	double	double	arccos(x), $-1 \le x \le 1$, $0 \le z \le pi$
asin(x)	double	double	arcsin(x), $-1 \le x \le 1$, $-pi/2 \le z \le pi/2$
atan(x)	double	double	arctg(x), $-pi/2 < z < pi/2$
atan2(y,x)	double, double	double	arctg(y/x), $x \neq 0$, $-pi < z < pi$
ceil(x)	double	double	Нахождение наименьшего целого $\ge x$
floor(x)	double	double	Нахождение наибольшего целого $\le x$
cos(x)	double	double	Нахождение cos(x), x-в радианах
sin(x)	double	double	Нахождение sin(x), x-в радианах
tan(x)	double	double	Нахождение tg(x), x-в радианах
cosh(x)	double	double	Нахождение ch(x)
sinh(x)	double	double	Нахождение sh(x)
tanh(x)	double	double	Нахождение th(x)
exp(x)	double	double	Нахождение e^x
fmod(x,y)	double	double	Деление x по модулю y
log(x)	double	double	Вычисление ln(x), x>0
log10(x)	double	double	Вычисление lg(x), x>0
modf(x,&i)	double, double *i	double	Определение целой i и дробной modf(x,&i) частей числа x
pow(x,y)	double, double	double	Вычисление x^y, x>0. Если x=y=0=>1
sqrt(x)	double	double	Вычисление корня квадратного, $x \ge 0$
pow10(x)	int	double	Вычисление 10^x

4.5 Операции сдвига и дополнения

Язык Си обеспечивает средства для обработки данных, представленных как набор бит. Это побитовые операции. Они дают возможность непосредственно управлять битовыми операциями, формирующими символьные или целые значения.

Операции “<<” (левый сдвиг) и “>>” (правый сдвиг) являются бинарными. Имеют следующий синтаксис:

выражение1 << выражение2;

выражение1 >> выражение2,

где выражение1 – строка бит, сдвигаемая влево или вправо на число бит, которое определяется величиной выражение2.

Типы операндов: выражение1 – символьный или целый, выражение2 – константа или константное выражение (выражение, состоящее из констант).

При левом сдвиге, освобождающиеся биты справа заполняются нулями. При правом сдвиге освобождающиеся биты слева заполняются нулями.

Сдвиг операнда влево на 1, 2, 3… бита соответствует его умножению на 2, 4, 8… соответственно. Сдвиг операнда вправо на 1, 2, 3… бита соответствует его целочисленному делению на 2, 4, 8… соответственно.

4.6 Логические операции и операции отношения

Логические операции и операции отношения используются при формировании логических выражений, имеющих только два значения: 1, если логическое выражение ИСТИННО; 0, если логическое выражение ЛОЖНО.

Си поддерживает следующие логические операции:

&& - логическое И;

| | - логическое ИЛИ;

! - логическое НЕ.

Таблица 4.3 – Таблица истинности логических операций

Операнд1	**Операнд2**	**&&**	**||**
ЛОЖЬ	ЛОЖЬ	ЛОЖЬ	ЛОЖЬ
ЛОЖЬ	ИСТИНА	ЛОЖЬ	ИСТИНА

ИСТИНА	ЛОЖЬ	ЛОЖЬ	ИСТИНА
ИСТИНА	ИСТИНА	ИСТИНА	ИСТИНА

Операция “!” унарная, инвертирует операнд справа (меняет ИСТИНА на ЛОЖЬ и наоборот).

Операции отношения:

> (больше) – дает результат ИСТИНА, если операнд слева от знака больше операнда справа от знака; в противном случае дает ЛОЖЬ;

< (меньше) – дает результат ИСТИНА, если операнд слева от знака меньше операнда справа от знака; в противном случае дает ЛОЖЬ;

>= (больше или равно) – дает результат ИСТИНА, если операнд слева от знака больше или равен операнду справа от знака; в противном случае дает ЛОЖЬ;

<= (меньше или равно) – дает результат ИСТИНА, если операнд слева от знака меньше или равен операнду справа от знака; в противном случае дает ЛОЖЬ;

= = (равно) – дает результат ИСТИНА, если операнд слева от знака равен операнду справа от знака; в противном случае дает ЛОЖЬ;

!= (не равно) – дает результат ИСТИНА, если операнд слева от знака не равен операнду справа от знака; в противном случае дает ЛОЖЬ.

Примечание. Строго говоря, в Си значению ИСТИНА соответствует не только значение 1, но и любое другое ненулевое значение.

4.7 Смешанные операнды и преобразования типов

Если в некотором арифметическом выражении встречаются операнды разных типов, то один из операндов подвергается преобразованию типов так, чтобы он соответствовал типу другого операнда. Операнд для преобразования выбирается по следующему правилу. Все основные типы имеют следующий порядок старшинства:

символьные<целые<длинные<плавающие<двойной точности

Типы, стоящие справа, превышают по старшинству все типы, стоящие слева.

Преобразование типов происходит и при выполнении операции присваивания: тип выражения справа от “=” преобразуется в тип переменной, стоящей слева от “=”. При этом может появиться необходимость повышения

ранга всех значений данных в выражении до некоторого высшего типа или понижение ранга значения до типа переменной, стоящей слева от “=”.

4.8 Операция sizeof

Результатом операции sizeof является размер в байтах типа или объявленной переменной. Применение операции к массиву возвращает число байтов, необходимое для размещения всех элементов массива.

5 УКАЗАТЕЛИ

5.1 Понятие указателя

В языке Си существует два способа доступа к переменной: ссылка на переменную по имени и использование механизма указателей. Механизм указателей в Си – одна из наиболее привлекательных для профессиональных программистов особенностей языка.

Указатель-переменная (или просто указатель) – это переменная, предназначенная для хранения адреса в памяти.

Указатель-константа – это значение адреса оперативной памяти. В языке Си определены две специальные операции для доступа к переменным через указатели: операция “&” и операция “*”. Результатом операции “&” является адрес объекта, к которому эта операция применяется. Например, &var1 дает адрес, по которому var1 хранится в памяти (точнее адрес первого байта var1). Операция “*” – это операция обращения к содержимому памяти по адресу, хранимому в переменной-указателе или равному указателю-константе.

Признаком переменной-указателя для компилятора является наличие в описании переменной двух компонентов:

1) типа объекта данных, для доступа к которому используется указатель (или, как часто говорят, на который ссылается указатель);

2) символа “*” перед именем переменной.

В совокупности тип и “*” воспринимаются компилятором как особый тип данных – “указатель на что-либо”.

Таким образом, описание *int var1, *ptr* приводит к появлению переменной var1 и указателя-переменной ptr. Указатель ptr имеет тип *int, т.е. тип “указатель на целое”. Место, выделяемое под такой тип компилятором, зависит от модели памяти.

Операцию “*”, пытаясь выразить словами смысл выражения, можно заменить фразой “взять содержимое по адресу, равному значению указателя”. Например, оператор присваивания **ptr = *ptr2 + 4* можно интерпретировать так: взять содержимое памяти по адресу, равному значению указателя ptr2, прибавить к этому содержимому 4, а результат поместить по адресу, равному значению указателя ptr. Число байтов, извлекаемых из памяти и участвующих в операции, определяется компилятором исходя из типа, на который “указывает” указатель.

5.2 Операции над указателями

Для указателей-переменных разрешены некоторые операции: присваивание, инкремент, декремент, сложение, вычитание, сравнение.

Язык Си разрешает операцию сравнения указателей одинакового типа. При выполнении присваивания значение указателя в правой части выражения пересылается в ячейку памяти, отведенную для указателя в левой части.

Важной особенностью арифметических операций с указателями является то, что физическое увеличение или уменьшение его значения зависит от типа указателя, т.е. от размера того объекта, на который указатель ссылается. Если к указателю, описанному как *type *ptr* прибавляется или отнимается константа N, значение *ptr* изменяется на N*sizeof(type). Разность двух указателей *type *ptr1, *ptr2* – это разность их значений, поделенная на sizeof(type).

Такие правила арифметических операций с указателями вытекают из того, что указатель в Си неявно рассматривается как указатель на начало массива однотипных элементов. Продвижение указателя вперед или назад совпадает с увеличением или уменьшением индекса элемента.

6 КОНСОЛЬНЫЙ ВВОД–ВЫВОД

6.1 Основные функции ввода-вывода

В языке Си отсутствуют специальные операторы ввода-вывода. Их заменяют стандартные функции, собранные в "библиотечный файл" *stdio.h*. Программа, содержащая обращения к функциям ввода-вывода, должна содержать строку подключения этого файла: *#include <stdio.h>*.

Рассмотрим шесть наиболее распространенных функций ввода информации с клавиатуры и ее вывода на экран (консольный ввод-вывод): *getchar*, *putchar* (для ввода-вывода символа); *gets*, *puts* (для ввода-вывода строки); *scanf*, *printf* (для форматированного ввода-вывода); и функцию *fflush* очистки буфера *stdin*.

Функция *getchar* предназначена для ввода символа, не имеет параметров, возвращает целое число - код введенного символа. Обращение имеет вид: *getchar()***.** Выполняя эту функцию, программа приостанавливает свою работу и ждет от пользователя ввода символа и/или нажатия клавиши Enter.

Функция *putchar* предназначена для вывода символа. Имеет один параметр типа *int* (код выводимого символа) или char. Обращение имеет вид: *putchar(i)*, где i – выражение, определяющее символ. После вывода символа курсор не переходит к началу новой строки.

Пример 1.

```
/* Ввод-вывод символа */
#include <stdio.h>
main()
{
    int ch;                          // Код вводимого символа
    puts("Введите любой символ");    // Вывод строки - посказки
    ch=getchar();                    /* Ввод символа и присваивание
        его кода переменной ch*/
    puts("Вы ввели символ");         // Вывод строки - сообщения
    putchar(ch);                     /* Вывод символа, определенного
        кодом ch*/
    printf("\n");                    // Перевод курсора к началу новой
        строки
```

```
    putchar('A');                   /* Вывод символа 'A', заданного
        константой*/
    printf("\n");
    putchar(65);                    /* Вывод символа 'A', заданного
        кодом символа*/
    fflush(stdin);                  // Очистка буфера ввода
    getchar();                      /* Ввод символа и/или нажатия
        Enter для организации задержки смены окна Output на окно
        редактора текста*/
    return(0);
}
```

Как видно из примера, функция *getchar()* может использоваться в операторе присваивания. В этом случае код введенного символа присваивается переменной типа *int* (в примере - переменной ch).

Функции *putchar('A')* и *putchar(65)* выводят один и тот же символ *'A'*, символьной константой и кодом символа (65).

Функция gets предназначена для ввода строки. Имеет один параметр, задающий адрес области памяти, в которую помещаются символы вводимой строки. В языке Си имя переменной, имеющей строковый тип, является этим адресом. Обращение имеет вид: *gets(name)),* где name – переменная строкового типа - имя вводимой строки. Выполняя эту функцию, программа приостанавливает свою работу и ждет от пользователя ввода последовательности символов и/или нажатия клавиши Enter.

Функция puts предназначена для вывода строки. Имеет один параметр, задающий адрес области памяти, из которой на экран выводятся символы. Как уже отмечалось, имя переменной, имеющей строковый тип, является этим адресом. Обращение имеет вид: *puts(name)),* где name – переменная-строка - имя выводимой строки или строка символов, заключенная в кавычки. После вывода строки курсор перемещается к началу новой строки экрана.

Пример 2.

```
/* Ввод-вывод с использованием функций getchar, putchar, gets, puts */
#include <stdio.h>
#include <conio.h>
main()
{
```

```
    char nf[40];
    int ch;
    clrscr();
    puts("**************************");
    puts("Введите Ваше имя и фамилию");
    gets(nf);
    puts("Вас зовут");
    puts(nf);
    puts("Введите любой символ");
    ch=getchar();
    puts("Вы ввели символ");
    putchar(ch);
    puts("**************************");
    return(0);
}
```

Функция *printf* предназначена для вывода форматированной последовательности данных. Может иметь один или несколько параметров, первым из которых является строка, называемая форматной строкой. За форматной строкой следует список вывода, который может содержать переменные, константы, выражения разных типов. Форматная строка задает способ преобразования и представления на печати элементов списка вывода, а также определяет, сколько элементов содержит список вывода и какого они типа. Обращение имеет вид:

printf(fs,a1,a2,...)),

где *fs* – форматная строка, *a1, a2, ...* - список вывода.

Например, printf("Получены значения %d %s\n",g,p). Здесь "Получены значения %d %s\n" - форматная строка, g,p - список вывода.

Или *printf("Вы ввели число").*Здесь нет списка вывода.

Форматная строка может содержать символы, которые будут выведены в том виде, в каком они есть (например: *Получены значения*), *спецификации преобразования*, которые начинаются со знака % (например: *%d, %s*), *последовательности переключения кода* (escape sequences)(например: *n*, что означает перевод курсора на следующую строку).

Спецификация преобразования имеет следующий вид:

%[флаг][ширина][.точность][размер]тип

Каждая спецификация заставляет функцию *printf,* имеющую список вывода, искать следующий элемент списка вывода, который затем преобразуется и выводится в соответствии с заданным преобразованием. Поэтому число элементов списка вывода должно соответствовать числу спецификаций форматной строки в вызове *printf*. Если *printf* содержит в качестве аргумента только форматную строку, то просто выводит эту строку.

Примечание. Те символы, которые должны быть выведены без преобразования (в том виде, в котором они записаны в форматной строке) до выводимого значения, записываются в форматной строке до спецификации формата, а те, которые должны быть выведены после значения, должны быть записаны после соответствующей спецификации преобразования.

В таблице 6.1 приведены спецификации преобразования.

Таблица 6.1 – Спецификации преобразования для функции *printf*

Элемент специфи-кации	**Значение**
флаг (необязательный элемент)	
-	Прижать число при выводе к левому краю выделенного поля
0	Заполнить лишнее пространство нулями вместо пробелов
+	Всегда выводить знак числа (+ или -)
пробел	Пробел на месте знака, если значение положительно
#	Выводить 0 перед восьмеричным или 0x перед шестнадцатеричным значением
ширина (необязательный элемент)	
n	Минимальная ширина поля в n символов
точность (необязательный элемент)	
.n	Не более n знаков после точки для числа в форме e, E, f

размер (необязательный элемент)	
h	Короткое целое (short int)
l	Длинное целое (long int)
тип (тип преобразования)	
d	Десятичное целое число со знаком
i	То же, что и d
o	Восьмеричное целое число без знака
u	Десятичное целое число без знака
x	Шестнадцатеричное целое число без знака, цифры в нижнем регистре
X	То же, что x, но цифры в верхнем регистре
f	Число со знаком в форме [-]dddd.dddd
e	Число со знаком в форме [-]d.ddde[+/-]ddd
g	Число со знаком в форме e или f
E	Число со знаком в форме [-]d.ddddE[+/-]ddd
G	Число со знаком в форме E или F
c	Один символ
s	Строка

6.2. Escape-последовательности

Обратная косая черта (\) имеет в языке Си специальное значение. Ее называют *escape - символом* и применяют для представления символов или чисел, которые нельзя непосредственно ввести с клавиатуры. *Escape-последовательность* - это *escape – символом*, за которым следует *escape – код*. В таблице 2 приведены допустимые в Си *escape-последовательности*. Escape-последовательности записываются в форматной строке операторов ввода-вывода и могут произвольно перемешиваться с любыми символами и спецификациями.

Например, оператор *printf("A\nBC\nDEF\n")* выведет на экран символы A,B,C,D,E,F в виде:

A
BC
DEF

Это определяется наличием в форматной строке escape-последовательности \n - переход к началу новой строки.

Таблица 6.2 – Escape-последовательности

Последовательность	**Название**	**Значение**
\n	Новая строка	Переход к началу новой строки
\t	Табуляция	Переход к следующей позиции табуляции
\b	Backspase	Возврат на шаг
\r	Возврат каретки	Возврат к началу текущей строки
\f	Перевод страницы	Начало нового экрана
\v	Вертикальная табуляция	Перевод курсора вниз на несколько строк
\\	Обратная косая черта	Вывод обратной косой черты
\'	Апостроф	Вывод апострофа
\0	Нуль (пусто)	
\"	Двойная кавычка	Вывод двойной кавычки
\a		Подача звукового сигнала

Функция scanf предназначена для ввода данных в заданном формате. Обращение имеет вид:

scanf(nf,&a1,&a2,...)

Здесь *nf* - *форматная* строка; *&a1,&a2,...* - список ввода - указатели на значения вводимых переменных *a1, a2, ...*

Примечание. Указатель на значение переменной - это адрес этой переменной, а не ее текущее значение. Чтобы указать на адрес переменной, надо перед именем переменной поставить символ операции &. Например, &dозначает адрес переменной d, а не значение, которое эта переменная имеет в данный момент. Т. о., если надо ввести значения переменных, например, a, b, c, то в списке ввода следует записать &a, &b, &c - указатели на значения переменных.

Выполняя функцию *scanf(nf,&a1,&a2,...)*, программа приостанавливает свою работу и ждет от пользователя ввода последовательности символов. После ввода запрашиваемой информации следует нажать клавишу Enter (курсор перейдет к началу следующей строки экрана). Функция *scanf* прекращает прием символов во внутренний буфер и переходит к обработке ввода в соответствии с форматной строкой. При этом, по спецификации %s функция *scanf* передает в программу все символы до первого разделителя, в том числе и пробела. Оставшиеся в буфере символы будут поступать в программу при следующих обращениях к функции *scanf*. Поэтому перед очередным обращением к функции *scanf* выполняется *очистка буфера stdin* фукцией *ffluch(stdin).*

Пример 3. Ввести целое положительное двузначное число i1 и целое отрицательное трехзначное число i2. Вывести в одну строку i1, i2 без разделителя, затем "пробел" и число i2. В следующую строку вывести i2, прижатое к левому краю поля в 7 позиций, "пробел", i2, прижатое к правому краю поля в 7 позиций, "пробел", i1 в поле из 5 позиций с выводом нулейв левые лишние позиции поля, "пробел", i1 в поле из 5 позицийсо знаком "+", "пробел", i2 в поле длиной 5 с пробелом вместо знака "+", "пробел", i2 в одну позицию, "пробел", i1 - в 4 позиции.

```
/* Форматированный ввод-вывод целых чисел */
#include <stdio.h>
#include <conio.h>
main()
{
      int i1,i2;
      clrscr();
      printf("Введите положительное и отрицательное целые числа\n");
      scanf("%i %d",&i1,&i2);
      printf("Числа %d и %d в разных форматах:\n",i1,i2);
```

```
    printf("*********\n");
    printf("%d %i %d\n%-7i %7d %05d %+5d % 5d %1d %4i\n",
    i1,i2,i2, i2,  i2,  i1,  i1,  i1, i2,i1);
    printf("******* ******* ***** ***** ***** * ****\n");
    fflush(stdin);
    getchar();
    return(0);
}
```

Ниже приводится вид экрана (окна Output) после выполнения программы (серым цветом выделены данные, введенные пользователем):

```
Введите положительное и отрицательное целые числа
12 -567
Числа 12 и -567 в разных форматах:
*********
12 -567 -567
-567          -567  00012    +12      12 -567     12
******* ******* ***** ***** ***** * ****
```

6.3 Функции atoi, atol, atof

Часто программисты избегают пользоваться функцией scanf. Если данные, прочитанные с помощью scanf, не соответствует форматной строке, то функция может вести себя непредсказуемо. Можно вводить данные с помощью функции gets. Данные представляются в виде строки. Затем нужно вызвать функцию *atoi, atol* или *atof* для преобразования строки в целое, длинное целое или вещественное число, соответственно. Указанные функции содержатся в файле *stdlib.h*

Пример 4.

```
/* Пример использования функций atoi и atof */
#include <conio.h>
#include <stdlib.h>
main()
{
    char s1[10],        // Строка для целого числа
         s2[20];        // Строка для вещественного числа
```

```
    int i;
    float f;
    printf("Введите целое число ");
    gets(s1);
    printf("Введите вещественное число ");
    gets(s2);
    i=atoi(s1);    // Преобразование строки в целое число
    f=atof(s2);    // Преобразование строки в вещественное число
    printf("i=%d f=%f\n",i,f);
    fflush(stdin);
    getchar();
    return(0);
}
```

Результат выполнения программы (серым цветом выделены введенные пользователем строки):

```
Введите целое число -231
Введите вещественное число 0.035647
i=-231 f=0.035647
```

7 РАЗРАБОТКА И ОТЛАДКА ЛИНЕЙНЫХ АЛГОРИТМОВ

Линейными называются алгоритмы, в которых выполняются все команды последовательно, одна за другой. При программировании таких алгоритмов используются операторы ввода и вывода данных, операторы присваивания, арифметические выражения. Арифметические выражения составляются из констант, переменных, обращений к математическим функциям с помощью знаков операций и скобок. В линейных алгоритмах арифметические выражения могут использоваться в операторах присваивания, операторах вывода. Например,

```
r = sqrt(d/(0.56*c-a));
μ = y-b*pow(fabs(r-d),1./3));
α = μ+r;
printf("r = %.2f c = %.3f знач.выражения: %.4f\n",r,c,r*sin(c));
```

Сразу же оговоримся, что в рассмотренных ниже примерах исходные данные задаются корректно, т.е. например, если в расчете функции используется ln(x), то значение x будет задано большим, чем 0, т.к. в противном случае произойдет ошибка вычисления (логарифм для $x \le 0$ не существует). Поэтому схема алгоритма проверки условия $x>0$ не предусматривает. Хотя, конечно, в серьезных задачах нужно обязательно проверять все вводимые значения на предмет: допустимы ли они или нет.

Пример составления и отладки линейной программы.

Составить программу вычисления y и z по формулам

$$y = \ln\left|\alpha - \sqrt{|x|}\right|; \qquad z = x - \frac{b}{\alpha + \frac{x^2}{4}}.$$

Решение.

Исходными данными являются значения α, b и x.

Таблица 7.1 – Таблица соответствия переменных:

Переменные в задаче	Имя на языке Си	Тип	Комментарий
α	a	float	исх. данное

b	b	float	исх. данное
x	x	float	исх. данное
y	y	float	результат
z	z	float	результат

Графическая схема алгоритма приведена на рисунке 7.1. В данном случае она слишком тривиальна, поэтому необязательна.

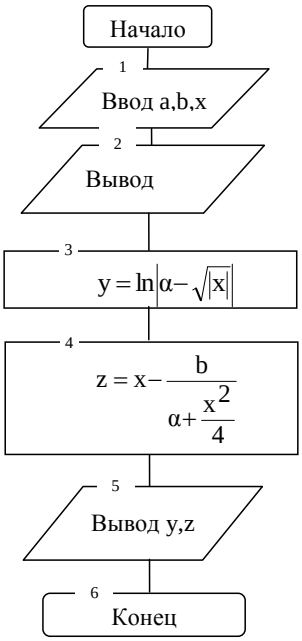


Рисунок 7.1 – Графическая схема алгоритма

Ниже приведен текст программы.

```
/* Пример линейной программы */
#include <stdio.h>
#include <math.h>
main()
{
  float x,y,z,a,b;                    // Описание переменных
```

```
  printf("Введите x,a,b\n");          // Вывод сообщения
  scanf("%f %f %f",&x,&a,&b);     // Ввод исх. Данных
// Вывод исх. данных
  printf("Исходные данные:\nx=%7.3f a=%7.3fb=%7.3f\n",x,a,b);
// Вычисление значения y
  y=log(fabs(a-sqrt(fabs(x))));
// Вычисление значения z
  z=x-b/(a+x*x/4);
// Вывод результатов
  printf("Результаты\n y=%f  z=%f\n",y, z);
  fflush(stdin);                          // Очистка буфера
  getchar();                      // Ожидание ввода символа
  return(0);
}
```

Для проверки правильности составленной программы выполняется ее отладка с помощью ПК следующим образом. Программа запускается на выполнение с вводом исходных данных, для которых ответ известен (указанный набор исходных данных с соответствующими результатами называется ***тестом***). Результаты выполнения программы сравниваются с известными. Ниже приводится тест и результаты выполнения программы. Значения y и z для теста подсчитаны в MachCad.

Тест:	Результаты выполнения программы:
α=1.2, b=5, x=0.3	Исходные данные:
y=-0.42729, z=-3.78998	x= 0.300 a= 1.200 b= 5.000
	Результаты:
	y=-0.427285 z=-3.78997

Видим, что результаты выполнения программы совпали с тестом.

8 ПРОГРАММИРОВАНИЕ РАЗВЕТЛЯЮЩИХСЯ АЛГОРИТМОВ

8.1 Понятие разветвляющегося алгоритма и программы

Разветвляющимся называется алгоритм, в котором последовательность и количество выполняемых команд зависит от выполнения или невыполнения некоторых условий. В разветвляющихся алгоритмах используются команды ветвления и выбора из большого количества вариантов. Графическое изображение команд ветвления представлено на рисунке 9.1.

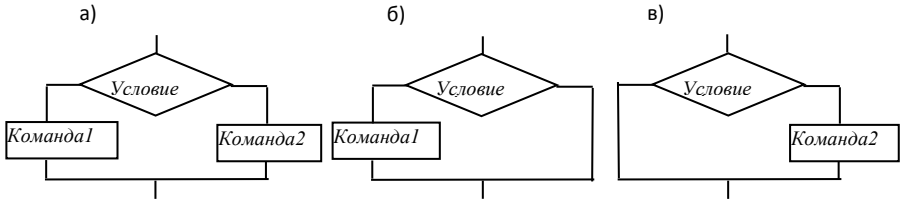


Рисунок 8.1 – Команды ветвления: а) – в полной форме; б) , в) – в сокращенной–форме

Для программирования ветвлений используется оператор ***if***, а для программирования выбора – операторы ***switch*** и ***break***.

8.2 Логические выражения

Логическое выражение (условие)- выражение, которое содержит знаки операций отношения и/или знаки логических операций. Значением логического выражения может быть только 1, если логическое выражение есть ИСТИНА (true), или 0, если логическое выражение ЛОЖЬ (false).

Операции отношения являются бинарными и обозначаются следующим образом (приведены в порядке уменьшения приоритета):
< (меньше); <= (меньше или равно); > (больше); >= (больше или равно); == (равно); **!=** (не равно).

Примеры: a<b, x!=5, y==c, t>=8.1, d<>(w-c).

Логические операции – это операции : **&&** - логическое И (бинарная операция); || - логическое ИЛИ (бинарная операция); **!** - логическое НЕ (унарная операция). Имеют более низкий приоритет, чем операции отношения. Результаты выполнения логических операций приведены в таблицах 1.1-1.3.

Таблица 8.1

x	y	x && y
1	1	1
0	1	0
1	0	0
0	0	0

Таблица 8.2

x	y	x || y
1	1	1
0	1	1
1	0	1
0	0	0

Таблица 8.3

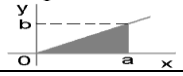
x	!x
1	0
0	1

Например, логическое выражение a>3 && c<7 при a=5, c=6 будет иметь значение ИСТИНА, а при a=3, c=8 – ЛОЖЬ; логическое выражение a>3 || c<7 при a=5, c=6 будет иметь значение ИСТИНА, при a=2, c=6 – ЛОЖЬ; при a=7 логическое выражение !a>2 будет иметь значение ЛОЖЬ.

При составлении логических выражений следует учитывать приоритет выполнения операций (табл. 1.4).

Примеры записи логических выражений приведены в таблице 89.5.

Таблица 8.5 – Примеры записи логических выражений

Условие	Логическое выражение
$x \in [a,b]$	x>=a && x<=b
$x \notin [a,b]$	x<a || x>b
$x \in [a,b]$ или $x \in [c,d]$	x>=a && x<=b || x>=c && x<=d
Хотя бы одно из чисел x, y положительное	x>0 || y>0
Только одно из чисел x, y положительное	(x>0 && !y>0) || (y>0 && !x>0)
Ни одно из чисел x, y не является четным	x % 2 !=0 && y % 2 !=0
Точка (x,y) принадлежит заштрихованной области 	x>=0 && x<=a && y>=0 && y<=b/a*x

8.3 Оператор *if*

Оператор ***if*** называется ***условным*** оператором и используется для программирования ветвлений. Имеет две формы записи – сокращенную и полную.

Сокращенная форма записи оператора ***if*** имеет вид:

if (выражение) оператор1;

Например, *if (a>b) y=2*x;*

Оператор ***if*** в полной форме называется оператором ***if- else*** и имеет следующую форму записи:

if (выражение) оператор1; else опертор2;

Например, *if (x>0 && x!=10) y=2*x; else y=x*x;*

Здесь ***выражение*** – выражение, которое может иметь арифметический тип или тип указателя; ***оператор1***, ***оператор2*** – простые или составные операторы языка. Простой оператор – это один оператор. Составной оператор (блок операторов) – это последовательность из нескольких любых операторов, в том числе операторов описания, заключенных в фигурные скобки.

Например, *if (x<=5.6) { float a=2.1; y=a*sin(x); printf("%.4f\n",y) ;}*

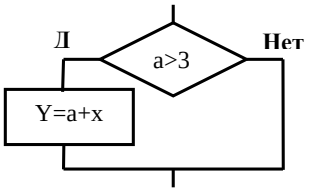
Следует учитывать, что переменная, описанная в блоке, не существует вне блока.

Оператор ***if*** называется вложенным, если хотя бы один из операторов ***оператор1***, ***оператор2*** содержит условные операторы, и используется для программирования вложенных ветвлений.

При выполнении оператора ***if*** и ***if- else*** сначала вычисляется значение выражения ***выражение***. Если полученное значение не равно нулю (имеет значение ***true***), то выполняется ***оператор1***, иначе: для оператора ***if*** - управление передаются на оператор, следующий за условным; для оператора ***if- else*** - выполняется ***оператор2,*** а затем управление передаются на оператор, следующий за условным.

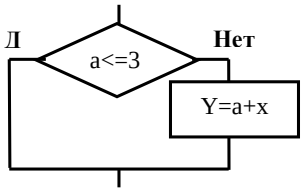
Ниже приведены примеры программирования ветвлений.

Пример 1.



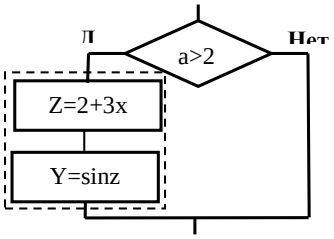
If (a>3) y=a+x;

Пример 2.



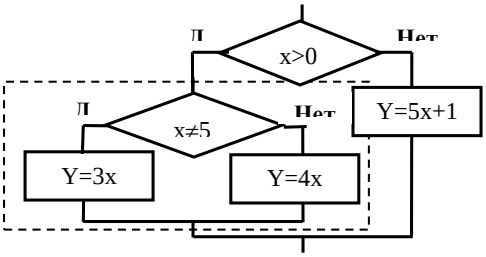
if (a<=3) { ; } else y=a+x;

Пример 3.



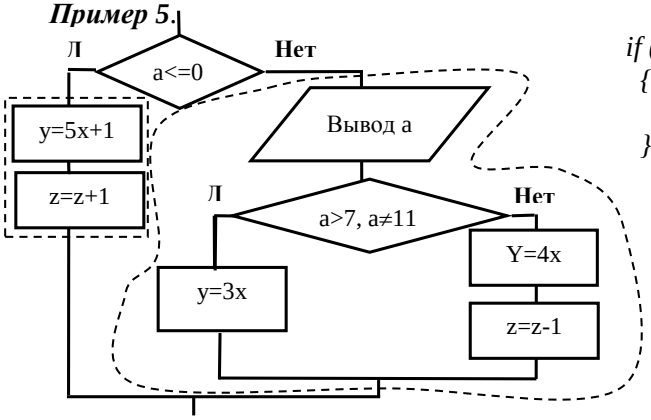
*if (a>2) { z=2+3*x; y=sin(z);}*

Пример 4.



*if (x>0) {if (x!=5) y=3*x; else y= 4*x;}*
*else y=5*x+1;*

Пример 5.




```
if (a<=0)
 { y=5*x+1;
    z++;
 }
    else{ cout << " a=" << a;
      if (a>7 && a<> 11
          y=3*x;
          else{ y=4*x; z--;
           }
     }
```

В примерах 1 и 2 оператор *y=a+x;* является простым оператором. В примере 3 оператор *{ z=2+3*x; y=sin(z);}* является составным. Кроме того, примеры 1 и 2 показывают, что неполное ветвление, в котором отсутствуют команды на ветви "Да", может быть приведено к другой форме неполного ветвления путем изменения условия на противоположное (условие, противоположное условию a>3 есть условие a<=3, и наоборот). Из примера 2 видно, что отсутствующие команды ветви "Да" программируются с помощью пустого оператора {;}.

Пример 4 иллюстрирует программирование вложенного ветвления. Фигурные скобки в данном случае не обязательны, так как компилятор относит часть *else* к ближайшему *if*.

В 5-ом примере на ветви "Да" внешнего ветвления имеется группа команд, которой соответствует составной оператор *{ y=5*x+1; z++;}*. На ветви "Нет" этого ветвления имеется группа команд, содержащая другое ветвление. Этой группе команд соответствует составной оператор *{ cout << " a=" << a; if (a>7 && a<> 11) y=3*x; { y=4*x; z--; } }*. Вложенное ветвление на ветви "Да" содержит одну команду (соответствующий оператор *y=3*x;*), а на ветви "Нет" – две команды, которым соответствует составной оператор *{ y=4*x; z--; }* .

8.4 Примеры разветвляющихся алгоритмов

Задача. Даны два вещественных числа *x* и *y* – координаты точки (*x*,*y*) на плоскости и область *D*, заданная графически (рисунок 9.2). Вычислить значение z, заданное формулой

$$z = \begin{cases} 1 + xy, \text{ если } (x, y) \in D; \\ 5, \text{ если } (x, y) \notin D. \end{cases} \qquad (1)$$

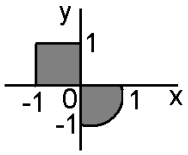


Рисунок 8.2 – *D* – заштрихованная область

Решение. Из рис.9.2 видно, что область *D* является объединением двух областей – квадрата со стороной, равной 1, и части круга с радиусом, равным 1. Следовательно, условие $(x,y)\in D$ состоит в том, что точка с координатами (x,y) принадлежит квадрату или сегмента круга. Условие принадлежности точки квадрату, изображенному на рис. 8.2, можно записать в виде логического выражения

$x>=-1 \&\& x<=0 \&\& y<=1 \&\& y>=0,$ (2)

а условие принадлежности точки сегменту – в виде логического выражения

$x>=0 \&\& x<=1 \&\& y>=-1 \&\& y<=0 \&\& x*x+y*y<=1.$ (3)

Объединив выражения (2) и (3) знаком логической операции || (ИЛИ), получим логическое выражение, соответствующее условию $(x,y)\in D$, которое затем используем в операторе ***if- else***.

Графическая схема алгоритма решения задачи изображена на рисунке 9.3. Алгоритм содержит ветвление, на каждой ветви которого имеется лишь по одной команде. Программируется такое ветвление с помощью оператора ***if- else***.

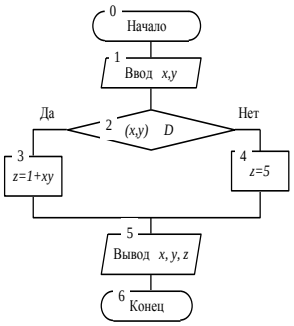


Рисунок 8.3 – Решение задачи 1.1.

```
/* Текст программы*/
#include <stdio.h>
#include <math.h>
main()
{
   float x,y,z;
   puts("Введите координаты точки (x,y)");
   scanf("%f %f",&x,&y);
   if ( x>=-1 && x<=0 && y<=1 && y>=0
      ||x>=0 && x<=1 && y>=-1 && y<=0 &&
x*x+y*y<=1 )       z=1+x*y;
      else z=5;
   printf("Для точки (%.2f,%.2f) z=%.4f",x,y,z);
   fflush(stdin);  getchar();
  return(0);
 }
```

Задача 8.2. Вычислить значение функции

$$y = \begin{cases} x^2, & \text{если} \quad x > 5 \text{ и } x \neq 10; & //1 \\ \sin x, & \text{если} \quad 0 \le x \le 5; & //2 \\ 2x - & \text{в остальных случаях.} & //3 \end{cases}$$

Решение. Необходимо задать значение x и проанализировать, по какой из трех формул, необходимо вычислить y . При этом придется задать один или два вопроса. Например, для x=6, зададим вопрос x>5 и x≠10? Ответ будет "Да", следовательно, вычислять y надо по первой формуле. Пусть x=3,14159. На вопрос x>5 и x≠10? ответ будет "Нет", т.е. вычислять y надо по второй или третьей формуле. Для выяснения, по какой конкретно, необходимо задать еще один вопрос, например, $0 \le x \le 5$? В этом случае ответ будет "Да". Следовательно, y должно вычисляться по второй формуле. Рассуждая аналогично, придем к выводу, что для x=-5 необходимо задать два вопроса x>5 и x≠10? и $0 \le x \le 5$? На каждый из вопросов будет получен ответ "Нет", следовательно, y надо вычислять по третьей формуле. Таким образом, приходим к разветвляющемуся алгоритму, изображенному на рис. 1.4. В алгоритме предусмотрен вывод исходных данных, формулы и номера формулы, по которой производится вычисление y.

Сведем в таблицу соответствия (табл. 1.6) обозначения переменных в задаче и в программе.

Таблица 8.6 – Таблица соответствия переменных:

Обозначения в задаче	Имена в алгоритме и программе	Тип данных	Комментарий
x	x	вещественный	исходное данное – аргумент функции
y	y	-//-	результат – значение функции
	n	целочисленный	номер формулы
	z	строка	вид формулы

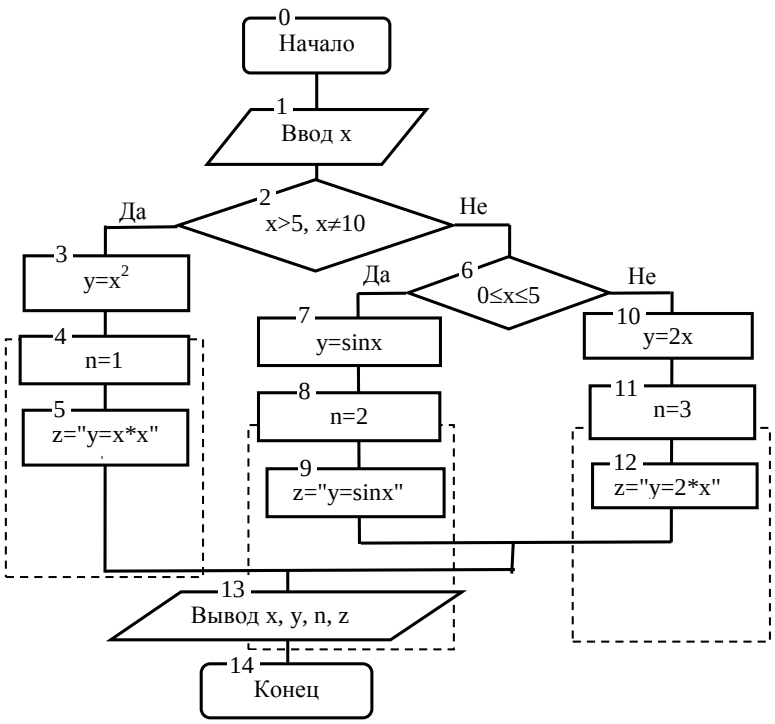


Рисунок 8.4 – Алгоритм решения задачи 9.2

```
/* Программа задачи 9.2*/
#include <stdio.h>
#include <math.h>
main()
{
  float x,y;
  int n;
  char *z;
  puts("Введите x");
  scanf("%f",&x);
  if(x>5 && x!=10)
 {
            y=x*x;
            n=1;
            z="y=x*x";
```

```
        }
    else
        if(x>=0 && x<=5)
          {
            y=sin(x);
            n=2;
            z="y=sinx";
           }
                 else
             {
               y=2*x;
               n=3;
               z="y=2x";
              }
 printf("При x=%.2f y=%.4f (формула %d:%s)\n",x,y,n,z);
 fflush(stdin);
 getchar();
 return(0);
}
```

Для проверки правильности разработанного алгоритма и программы достаточно подготовить семь тестов. Сделаем это с помощью рисунка 9.5. На рисунке 9.5 на числовой оси отмечены промежутки значений *x* с указанием формул для вычисления *y* для каждого промежутка и каждой граничной точки. В таблице 9.7 приведены тесты.

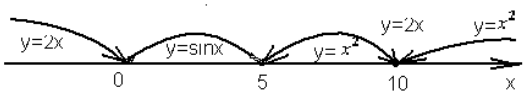


Рисунок 8.6 – Промежутки значений *x*

Таблица 8.7 – Тесты к задаче 8.2

Тест №	Значение *x*	Значение *y*	Формула для вычисления *y*
Тест 1	-7.2	-14.4	*y=2x*
Тест 2	0	0	*y=sinx*
Тест 3	1.5707968	≈1	*y=sinx*
Тест 4	5	-0.9589	*y=sinx*
Тест 5	6	36	$y=x^2$
Тест 6	10	20	*y=2x*
Тест 7	11	121	$y=x^2$

8.5. Команда выбора. Операторы *switch* и *break*

Выбор из многих вариантов можно сделать с помощью вложенных операторов ***if- else***. Более удобный способ – использование операторов ***switch*** и ***break***. Общий вид оператора выбора :

```
switch (i)
 {
  case k1 : op1;
  [ break;]
  case k2 : op2;
  [ break;]
  …
  case kn : opn;
  [ break;]
  [ default : opn+1;]
 }
```

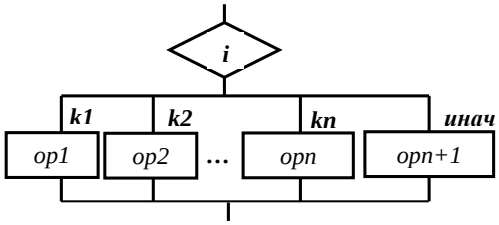


Рисунок 8.7 – Структура команды выбора, соответствующая оператору ***switch*** с оператором ***break*** после каждого ***case***

Здесь ***i*** – любое выражение, дающее целое значение (в том числе – символьное);

k1, k2, ..., kn – константы или константные выражения - возможные значения ***i*** (например, 2*g, 'a'-'b', '1'') – записываются после слова ***case***;

op1, op2, ..., opn, opn+1 – простые или составные операторы языка.

Выполнение оператора switch:

1. Вычисляется значение выражения ***i.***
2. Значение ***i*** сравнивается с *k1, k2, ..., kn.*
3. При совпадении ***i*** со значением *ki* выполняется оператор *opi*. Затем управление передается оператору, стоящему после следующего ***case***, если после *opi* нет оператора ***break.*** Если после оператора *opi* есть оператор ***break***, то управление передается на оператор, следующий после оператора ***switch***.
4. Если ***i*** не совпало ни с одним *k1, k2, ..., kn*, то выполняется оператор *opn+1*, стоящий после ***default***, а затем оператор, следующий после оператора ***switch***. При отсутствии ***default*** выполняется следующий после ***switch*** оператор.

Рассмотрим примеры решения задач с использованием оператора выбора.

Задача9.3. Задан номер текущего дня недели. Вывести названия дней, оставшихся до конца недели, включая текущий день.

Решение. Возможные значения номера дней недели (обозначим их через *n*) – это числа 1-7. При вводе числа 1 программа должна выводить названия "Понедельник ", "Вторник ", …, "Воскресенье", при вводе числа 2 – названия "Вторник ", …, "Воскресенье". И т.д. При вводе числа 7 – только название "Воскресенье". При вводе любого другого числа – сообщение "Неправильно введен номер дня недели". Реализация выбора из такого множества вариантов

может быть осуществлена с помощью оператора ***switch (k)***. Ниже приведен текст программы с оператором ***switch(k)***, в котором только после последнего ***case*** есть оператор ***break***;, по которому и происходит выход из ***switch*** и переход на следующий оператор программы (*fflush(stdin);*) для *k*=1-7. Если k≠1-7, то выполнится оператор *puts ("Неправильно введен номер дня недели");*, стоящий после *default*, затем следующий оператор программы (*fflush(stdin);*).

```
/*  Программа задачи 8.3 */
        #include <stdio.h>
        #include <math.h>
        main()
        {
          int k;                        // Номер дня недели
          puts("Введите номер дня недели");
          scanf("%d",&k);
          printf("До конца недели:\n");
          switch (k)
          {
            case 1: puts("Понедельник ");
            case 2: puts ("Вторник ");
            case 3: puts ("Среда ");
            case 4: puts ("Четверг ");
            case 5: puts ("Пятница ");
            case 6: puts ("Суббота ");
            case 7: puts ("Воскресенье \n");
            break;
            default: puts ("Неправильно введен номер дня недели");
  }
  fflush(stdin);  getchar();
  return(0);
}
```

Следующий пример показывает, что после ***case*** *ki* : может не быть оператора.

Задача 8.4. Ввести символ и определить, является ли он цифрой.

Решение. Пусть k – вводимый символ. В программе использованы операторы ввода-вывода в стиле Си.

В задаче 1.5 используется оператор выбора с оператором ***break*** после каждого ***case*** *ki* :

Задача 9.5. Дано целое число в диапазоне 1 – 5. Вывести строку — словесное описание соответствующей оценки (1 — "плохо", 2 — "неудовлетворительно", 3 — "удовлетворительно", 4 — "хорошо", 5 — "отлично").

Решение. В программе N- число, которое вводится, а затем анализируется. Если ввести, например 4, то программа выведет слово "Хорошо", а если ввести 7, то

программа выведет строку "Нет оценки" и т.д. В программе использованы операторы ввода-вывода в стиле Си++.

Программа задачи 8.4

```
#include <stdio.h>
main()
{
  char k;
  puts("Введите символ");
  k=getchar();
  switch (k)
  {
    case '1':
    case '2':
    case '3':
    case '4':
    case '5':
    case '6':
    case '7':
    case '8':
    case '9':
    case '0':printf("Это число %c\n",k);
    break;
    default: printf("%c-не число\n",k);
  }
  fflush(stdin);
  getchar();
  return(0);
}
```

Программа задачи 8.5

```
#include <stdio.h>
#include <iostream.h>
main()
{
  int N;
  cout << "\nВведите число ";
  cin >> N;
  switch (N)
  {
    case 1: cout << "\nПлохо";
    break;
    case 2: cout << "\nНеуд.";
    break;
    case 3: cout << "\nУдовл.";
    break;
    case 4: cout << "\nХорошо";
    break;
    case 5: cout << "\nОтлично";
    break;
    default: cout << "\nНет  оценки ";
  }
  fflush(stdin);
  getchar();
  return(0);
}
```

9 ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

9.1 Понятие цикла

При составлении алгоритмов решения практических задач нередко приходится некоторые действия выполнять многократно, каждый раз с новыми значениями для величин, входящих в повторяющуюся группу действий.

Многократно повторяемые участки вычислений называют **циклами**. Один проход цикла называется **итерацией** или **повторением**. Величины, определяющие количество повторений цикла, называются **параметрами циклов.** Параметрами цикла могут быть константы и переменные, причем среди них обязательно должна быть хотя бы одна переменная, значение которой изменяется при выполнении цикла. Целочисленные параметры цикла, изменяющиеся с постоянным шагом на каждой итерации, называются **счетчиками цикла**. Вычислительные процессы, содержащие циклы, называют **циклическими**.

Циклы состоят из следующих частей:

подготовки к выполнению цикла (присваивания параметрам цикла начальных значений);

определения момента завершения цикла (условия повторения цикла);

тела цикла, включающего:

- рабочую часть цикла, содержащую действия, непосредственно связанные с решением задачи, для которой разработан алгоритм;
- подготовку данных для очередного шага цикла (изменение параметров цикла).

Различают циклы с предусловием и циклы с постусловием. В циклах с предусловием условие повторения цикла проверяется раньше, чем выполняется тело цикла, поэтому возможно, что цикл не выполнится ни разу. В циклах с постусловием условие повторения цикла проверяется после выполнения тела цикла и подготовки данных для очередного шага цикла, поэтому цикл выполняется всегда хотя бы один раз. В общем виде циклы с предусловием и постусловием графически можно изобразить, как показано на рисунке 10.1.

ци

В языке Си циклы программируются с помощью трех разных операторов ***for***, ***while***, ***do-while*** . Возможно принудительное завершение текущей итерации и цикла в целом. Для этого служат операторы ***break***, ***continue***, ***return***, ***goto***. Передавать управление извне внутрь цикла не рекомендуется.

Использование оператора ***goto*** приводит к созданию запутанного, трудно читаемого кода, прозванного программистами "***спагетти***". Использование оператора ***goto*** среди программистов считается верхом неприличия. Чтобы избежать оператора ***goto***, используют сложные условия повторения цикла.

а) б)

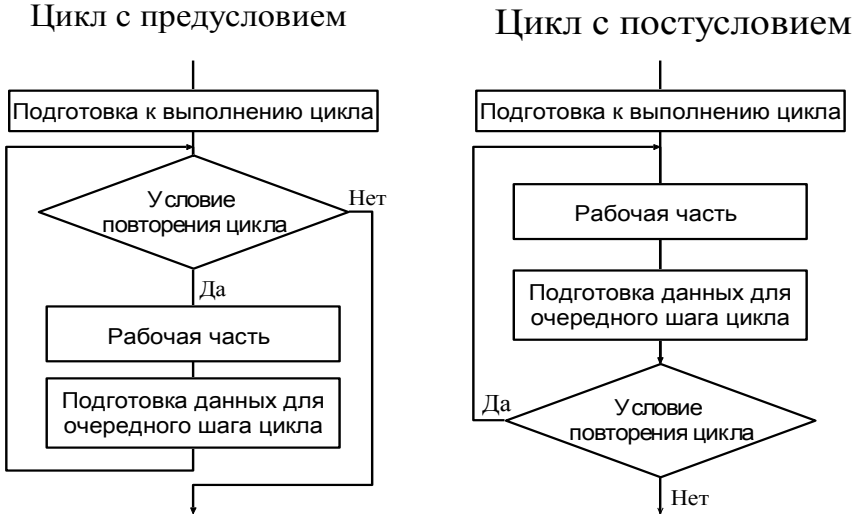


Рисунок 9.1 – Графические схемы циклов а) – с предусловием и б) – с постусловием

9.2 Операторы циклов

9.2.1. Оператор цикла for

Оператор цикла **for** – оператор цикла с параметром и с предусловием. Используется, если заранее известно количество повторений тела цикла, начальные значения параметров цикла, условие повторения цикла, как изменяются параметры цикла. Общий вид оператора цикла **for**:

for(p1; p2; p3) S;

Здесь

for(p1; p2; p3) – заголовок цикла.

S – простой или составной оператор языка Си – тело или рабочая часть цикла.

p1 – список операторов, разделенных запятой, инициирующих начальные значения (как правило) параметров цикла. Эти операторы выполняются один раз до начала выполнения рабочей части цикла. Областью действия переменных, объявленных в этой части цикла, является цикл.

p2 – список операторов и выражений, определяющих условие повторения цикла. Выполняется перед каждым выполнением тела цикла. При этом, если

значение последнего выражения списка **p2** истинно (!=0), то тело цикла выполняется, а если ложно (==0), происходит выход из цикла к следующему оператору программы.

p3 – список операторов и выражений изменения параметров цикла (подготовка данных для очередного шага цикла). Выполняется после каждого выполнения тела цикла.

Пример (вычисление и вывод квадратов чисел от 1 до 10).

for (i=1; i<=10; i++) printf("i*i=%d\n",i*i);

Здесь ***p1*** – оператор ***i=1*** (инициирует начальное значение параметра цикла ***i***); ***p2*** **–** выражение **i<=10** (определяет условие повторения цикла); ***p3*** - оператор ***i++*** (оператор изменения параметра цикла)**.**

S **–** оператор ***printf("i*i=%d\n",i*i);*** (рабочая часть цикла).

Пример (вычисление суммы чисел от 1 до 15)

for (int i=1,q=0; i<=10; i++) q+=i;

Здесь ***p1*** **-** список операторов, разделенных запятой. Первый из этих операторов (***int i=1***) объявляет параметр цикла ***i*** и инициирует его начальное значение и объявляет и инициирует начальное значение вычисляемой суммы ***q***. Как и в первом примере ***p2*** **–** выражение ***i*<=10**; ***p3*** - оператор ***i++***. ***S*** **–** оператор ***q+=i;***

Основные правила и порядок выполнения оператора цикла ***for***:

1) До начала выполнения тела цикла выполняется один раз список ***p1***.
2) Выполняются операторы и выражения списка ***p2***; производится анализ полученного значения последнего выражения из ***p2*** (в дальнейшем ***p2***):
   a) если ***p2*** истинно (!=0), то тело цикла выполняется;
   b) если ***p2*** ложно (==0), то тело цикл завершается;
   c) если ***p2*** ложно (==0) до первого выполнения тела цикла, то тело цикла не выполняется ни разу.
3) После выполнения рабочей части цикла выполняются операторы и выражения ***p3*** и осуществляется переход к п. 2) данных правил.
4) Появление в любом месте тела цикла оператора ***continue*** вызывает переход к выполнению списка ***p3***, п. 3) данных правил.
5) Появление в любом месте тела цикла оператора ***break*** вызывает переход к оператору, следующему после оператора цикла, т. е. осуществляется выход из цикла. При этом параметр цикла сохраняет то значение, при котором произошло завершение выполнения цикла.
6) После нормального завершения цикла значение параметра цикла равно значению, которое привело к завершению цикла.

Пример:

for (i=1; i<5; i++);
printf ("i=%d\n",i);

Приведен цикл, не содержащий тела цикла, так как сразу после заголовка цикла стоит символ **";"** (тело цикла – пустой оператор), и оператор вывода

значения ***i***. Выполнение цикла завершится при ***i***=5. Значение ***5*** и будет выведено.

7) Если в заголовке цикла отсутствует список ***p1*** и/или ***p2*** и/или ***p3***, то символы "**;**" должны остаться. Например,

```
for ( ; ; );          //Бесконечный цикл
for ( i=1; ; i++ );   //Бесконечный цикл
for ( i=1; i<5 ; );   //Бесконечный цикл
```

Примечание. *Использование в заголовке цикла посторонних вычислений считается плохим стилем. Следует использовать заголовок цикла for для операций по управлению циклом.*

Пример.1.2. Вычислить и вывести степени двойки – значения 2^n для n=1, *2, …, k (k>1).*

*Решение. Пусть x – искомые значения. Положим x=1. Тогда первая степень двойки x равна x•2, что равно 2^1. Если положить x=x*2, то вторая степень двойки есть x•2, что равно 2^2. И так далее. Т. о., степени двойки от первой до к-ой можно вычислить, выполнив к раз рекуррентное соотношение x=x•2, положив предварительно x=1. Реализовать такой алгоритм можно с использованием цикла с параметром и с предусловием. Графическая схема алгоритма, программа на языке Си, результаты выполнения программы при к=15 изображены на рис. 2.1. Параметр цикла* ***i*** *(степень двойки) принимает значения 1, 2, …, к, следовательно, является счетчиком.*

9.2.2. Оператор цикла while

Оператор цикла **while** – оператор цикла итеративного типа с предусловием. Используется, когда количество повторений операторов тела цикла заранее неизвестно и определяется в процессе выполнения цикла. В этом операторе анализ конца цикла производится до выполнения операторов тела цикла.

Общий вид оператора цикла **while**:

```
while ( b ) S;
```

Здесь **b** – выражение любого типа, например, логическое, приводимое к арифметическому типу, определяющее условие повторения цикла; **S** – простой или составной оператор - рабочая часть цикла. Он должен включать операторы тела цикла и операторы изменения операндов выражения **b** (подготовки данных для очередного шага цикла).

Основные правила использования и порядок выполнения оператора цикла ***while***:

1) До оператора ***while*** в программе должны содержаться операторы подготовки к выполнению цикла.

2). Выполнение оператора ***while*** начинается с вычисления значения выражения ***b*** (выражение ***b*** вычисляется перед каждой итерацией цикла). Производится анализ полученного значения ***b***:

a) если ***b*** истинно (!=0), то выполняется ***S***;
b) если ***b*** ложно (==0), то тело цикл завершается;
c) если ***b*** ложно (==0) до первого выполнения тела цикла, то тело цикла не выполняется ни разу.

3). После выполнения оператора ***S*** вычисляется значение выражения ***b*** и осуществляется переход к п. 1) данных правил.

4). После нормального завершения цикла значения параметров цикла равно значениям, которые привели к завершению цикла.

Цикл с предусловием ***while***, т.о., реализуется по схеме, изображенной на рис. 9.1.

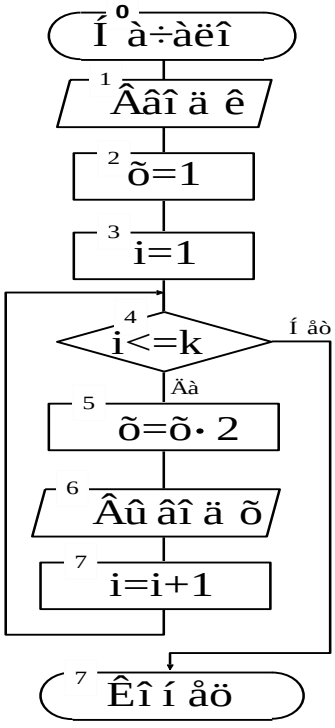


Рисунок 10.1 – Графическая схема алгоритма вычислени степеней двойки, текст программы и результаты выполнения программы при *к*=15

```
#include <stdio.h>
 main()
  {
int i,k;
long x;
puts("Введите k-степень двойки ");
scanf("%d",&k);
x=1;
for(i=1; i<=k; i++)  //Заголовок цикла
  {
x*=2;                                   Тело цикла
printf("%5ld ",x);
if(i%5==0)
printf("\n");
}
printf("\nНажмите любую клавишу\n");
fflush(stdin);
getchar();
return(0);
}
```

Результаты выполнения программы:

2 4 8 16 32

64 128 256 512 1024

2048 4096 8192 16384 32768

Примеры.

Пример 2.2.1.

while(true); - бесконечный цикл

Пример 2.2.2. Какое значение y будет напечатано после выполнения операторов?

```
x=2;
while(x<0) y=x•3;
y=x-1;
printf("y=%d",y);
```

Условие x<0 в операторе цикла является ложным, поэтому цикл не выполнится ни разу. Значит, y=2-1=1. Будет выведено значение 1.

Пример 2.2.3. Чему будет равно j после завершения цикла?

```
j=3; while(j<=7) j=j+2;
```

После завершения цикла j будет иметь значение 9.

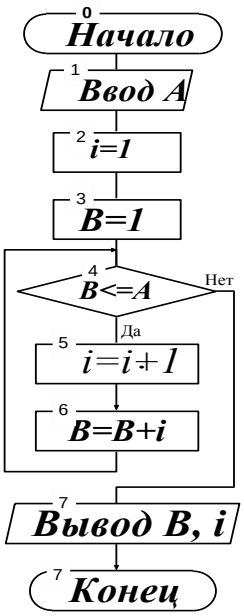
Пример 2.2.4. Дано вещественное число A>0. Числа B_i образуются по закону: $B_i = \sum_{k=1}^{i} k, \quad i = 1,2,\ldots$. Найти среди B_i первое число, большее A.

Решение. Очевидно, что B_1=1. Каждое следующее значение B_i может быть получено по рекуррентному соотношению $B_i = B_{i-1}+i$, где i=2, 3, … , что нетрудно проверить. В самом деле B_2=1+2= B_1+2=1+2=3; B_3=1+2+3=B_2+3=3+3=6; B_4=1+2+3+4=B_3+4=6+4=10; и т. д. При этом заранее неизвестно, сколько раз соотношение $B_i = B_{i-1}+i$ будет выполнено, нет необходимости сохранять в памяти компьютера все полученные значения B_i. Поэтому тело цикла будет состоять в выполнении операторов i=i+1 и B= B+1. Условие повторения - неравенство B<=A. Параметром цикла будет являться B. Подготовка данных для очередной итерации цикла будет заключаться в увеличении параметра B на величину i . Подготовка к выполнению цикла будет состоять в присвоении переменным B и i начальных значений: B =1; i=1. Т.о., получим графическую схему, изображенную на рис. 2.2.4а. Текст программы представлен на рис. 2.2.4б. В программе для организации цикла использован оператор **while**, поскольку число итераций цикла заранее неизвестно.

Результаты выполнения программы для разных значений A:

```
b=6  i=3 для a=4.00
b=21 i=6 для a=16.00
b=45 i=9 для a=40.00
```

а) Графическая схема



б) Текст программы

```
#include <stdio.h>
main()
{
float a;
int i,b;
puts("Введите число a>0 ");
scanf("%f",&a);
b=i=1;
while(b<=a)
{
  i++;  // i=i+1;
  b+=i; // b=b+i;
}
printf("b=%d i=%d для a=%.2f\n",b,i,a);
fflush(stdin);
getchar();
return(0);
}
```

Рисунок 9.2. Решение примера 2.2.4.

Пример 2.2.5. Табулирование функции.

Постановка задачи:

Вычислить значение функции $y = x^2$ для x, изменяющегося от $x_н$ до $x_к$ с шагом Δx.

Решение

Исходные данные:

- левая граница интервала
- правая граница интервала
- шаг табулирования

Результаты:

таблица значений заданной функции на интервале ($x_н, x_к$).

Таблица 9.1 – Таблица соответствия переменных

Имя переменной в условии	Имя переменной в программе	Тип переменной	Комментарий
$x_н$	x_n	float	Левая граница интервала
$x_к$	x_k	float	Правая граница интервала
Δx	h	float	Шаг табулирования
y	y	float	Значение функции
x	x	float	Аргумент функции

Таблица 10.2 – Тесты

Тест 1		*Тест 2*	
$x_н$ **=2,** $x_к$ **=6,** Δx **=1**		$x_н$ **=4,** $x_к$ **=14,** Δx **=2**	
x	***y***	***x***	***y***
2	4	4	16
3	9	6	36
4	16	8	64
5	25	10	100
6	36	12	144
		14	196

Текст программы

```
/*      подключение стандартных библиотек      */
#include<stdio.h>
#include<conio.h>
main()
{
 /*      описание используемых переменных       */
 float x_n,x_k,h,x,y;
 clrscr;
 /*ввод исходных данных*/
 puts("введите начавльное значение интервала");
 scanf("%f",&x_n);
 puts("введите конечное значение интервала");
 scanf("%f",&x_k);
 puts("введите шаг табулирования");
 scanf("%f",&h);
```

```
/*      формирование шапки таблицы        */
puts(" ┌──────────┬──────────┐ ");
puts(" │    x     │    Y     │ ");
puts(" ├──────────┼──────────┤ ");

/*      рассчет таблицы значений функции y=x*x на заданном интервале        */
x=xn;
while (x<=xk)
 {
   y=x*x;
   printf(" │%7.1f│%7.1f│\n",x,y);
   x=x+h ;
 }
 puts(" └──────────┴──────────┘ ");
}
```

Графическая схема алгоритма

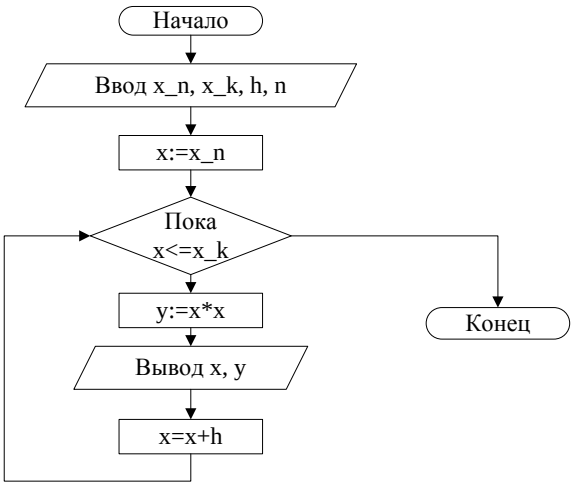


Рисунок 9.3 – Табулирование функции

Пример 2.2.7. Определить для заданного натурального числа N, составляют ли цифры этого числа неубывающую последовательность?

Решение. Рассмотрим, например числа 113479 и 64801. Цифры первого числа (1, 1, 3, 4, 7, 9) составляют неубывающую последовательность, а числа второго числа (6, 4, 8, 0, 1) – не составляют неубывающую последовательность.

Для ответа на поставленный вопрос задачи будем выделять цифры числа, начиная с младшей (правой) до старшей (левой) цифры. Если при этом для

каждой пары соседних цифр выполняется неравенство C1>=C2, где C1 – младшая из цифр в паре, C2 – старшая из цифр в паре, то цифры заданного числа составляют неубывающую последовательность. Если для очередной пары цифр это неравенство не выполняется, то цифры заданного числа не составляют неубывающую последовательность. Например, для первого числа, C1=9; C2=7. Так как 9>7, то рассматривается следующая пара C1=C2=7; C2=4. Так как 7>4, то рассматривается следующая пара C1=C2=4; C2=3. Так как 4>3, то рассматривается следующая пара C1=C2=3; C2=1. Так как 3>1, то рассматривается следующая пара, последняя пара, C1=C2=1; C2=1, для которой C1=C2. Для второго числа уже на второй итерации выяляется, что C1<C2 (0<8). Процесс следует остановить и сделать вывод о том, что цифры числа не составляют неубывающую последовательность.

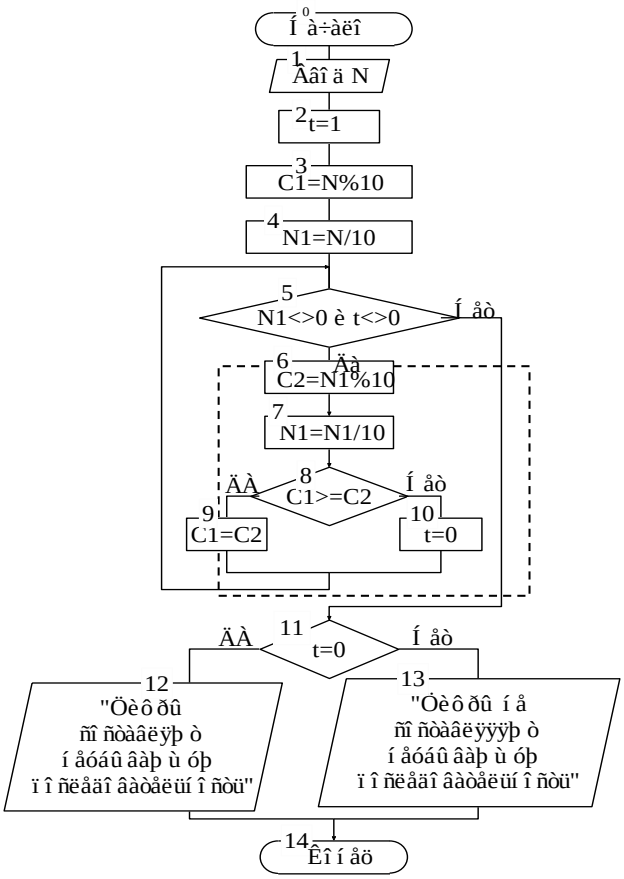
Цифры числа будем определять как остаток от деления числа на 10. Младшая цифра – это N%10 (% - операция определения остатка от деления целых чисел). Следующая цифра – это остаток от деления числа N1 на 10, где N1 – число, получаемое из N отбрасыванием его младшей цифры и т. д. Процесс выделения цифр заканчивается, когда N1 становится равным нулю.

Если последовательность цифр числа не является неубывающей (не выполняется для очередной пары цифр неравенство C1>=C2), то процесс лучше всего прекратить. Для этой цели используется переменная t, принимающая значение 1, если не найдена пара цифр такая, что C1<C2, и 0 в противном случае.

Таким образом, описанный процесс является циклическим. Он будет иметь два параметра цикла ***N1*** и ***t***, в качестве условия повторения будет иметь сложное условие ***N1<>0 и t<>0*** (не закончились цифры и не установлено условие неупорядоченности цифр по не убыванию), телом его цикла будут команды, выделенные пунктирной линией на рис. 2.?. Начальные значения параметров ***N1=N****/10* и ***t=1***.

Графическая схема алгоритма и программа изображены на рисунке 10.4.

В программе для организации цикла использован оператор ***while*** , т. к. число повторений цикла заранее неизвестно.




```
include <stdio.h>

main()
{ long N, N1;
  int C1, C2, t=1;
  puts("Введите N");
  scanf("%ld",&N);
  C1=(N%10);
  N1=N/10;
  while (N1!=0 && t!=0)
   {
     C2=(N1%10);
     N1=N1/10;
     if(C1>=C2) C1=C2;
        else t=0;
   }
  if(t==0) printf("N");
       else printf("Y");
  printf("   N=%ld\n",N);
  fflush(stdin);
  getchar();
  return(0);
}
```

Рисунок 94. – Решение задачи 2.2.7

9.2.3. Оператор цикла do-while

Оператор цикла ***do-while*** является оператором цикла с постусловием, так как в нем анализ конца цикла производится после операторов тела цикла. Он используется, как и оператор ***while***, когда количество итераций цикла заранее неизвестно и определяется в процессе выполнения цикла.

Особенностью оператора является выполнение тела цикла хотя бы один раз.

Общий вид оператора:

do S while (b);

Здесь ***S*** – простой или составной оператор – тело цикла. Он должен включать рабочую часть цикла и операторы изменения операндов выражения ***b*** (подготовки данных для очередного шага цикла); ***b*** – выражение любого типа, например, логическое, приводимое к арифметическому типу, определяющее условие повторения цикла.

Оператор цикла ***do-while*** выполняется по схеме цикла с постусловием, изображенной на рис. 10.1б.

Пример. Найти сумму первых *N* членов ряда $\frac{x}{2}+\frac{x^3}{4}+\frac{x^5}{8}+\frac{x^7}{16}+\ldots$.

Решение. Введем обозначения: C – сумма ряда, U – произвольный член ряда. Сумму вычислим в цикле как нарастающую сумму: $C = C + U$. Для вычисления значения очередного члена ряда достаточно значение предыдущего члена ряда умножить на $\frac{x^2}{2}$, т.е. $U = U \cdot \frac{x^2}{2}$.

Полученная формула называется рекуррентной. Она позволяет вычислить любой член ряда, если известен первый член ряда.

К моменту исполнения первой итерации цикла значения C и U должны быть определены: $C = 0;\ U = \frac{x}{2}$. Параметром цикла пусть будет переменная *k*. Параметр *k* должен изменяться от 1 до *N* с шагом 1. Выражение $C = C + U$ должно быть выполнено хотя бы один раз (для N=1 – один раз; для N=2 – два раза и т. д.), поэтому для вычисления суммы следует использовать цикл с постусловием. Графическая схема алгоритма приведена на рисунке 10.5.

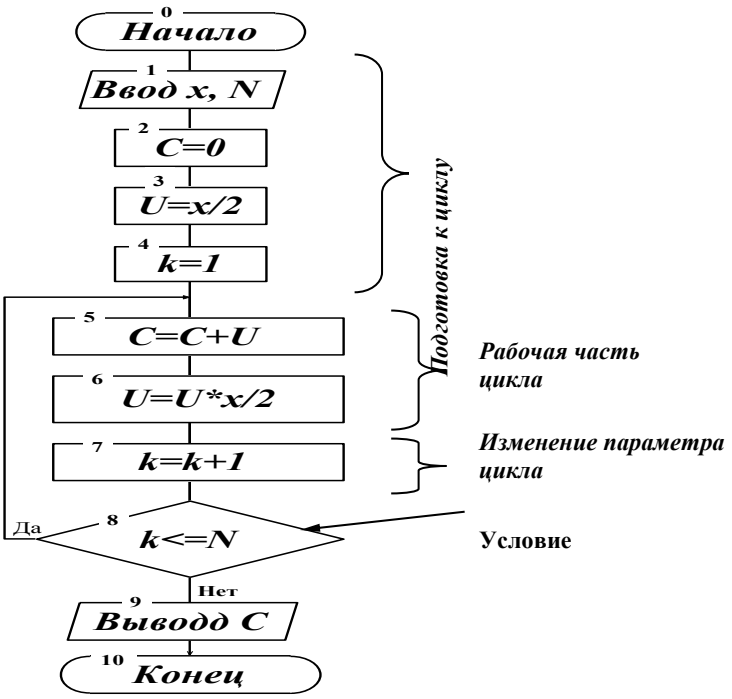


Рисунок 9.5 – Графическая схема алгоритма

9.3. Вложенные циклы

В теле цикла может располагаться еще один цикл. В этом случае говорят, что алгоритм или программа содержат цикл в цикле или вложенные циклы. Общий вид графической схемы цикла в цикле изображен на рис. 3.1. Здесь внешний цикл – цикл с предусловием, внутренний цикл –цикл с постусловием. Рабочая часть внешнего цикла выделена пунктирной линией. Число выполнений команд рабочей части внутреннего цикла может быть найдено как произведение числа повторений внутреннего цикла на число повторений внешнего цикла.

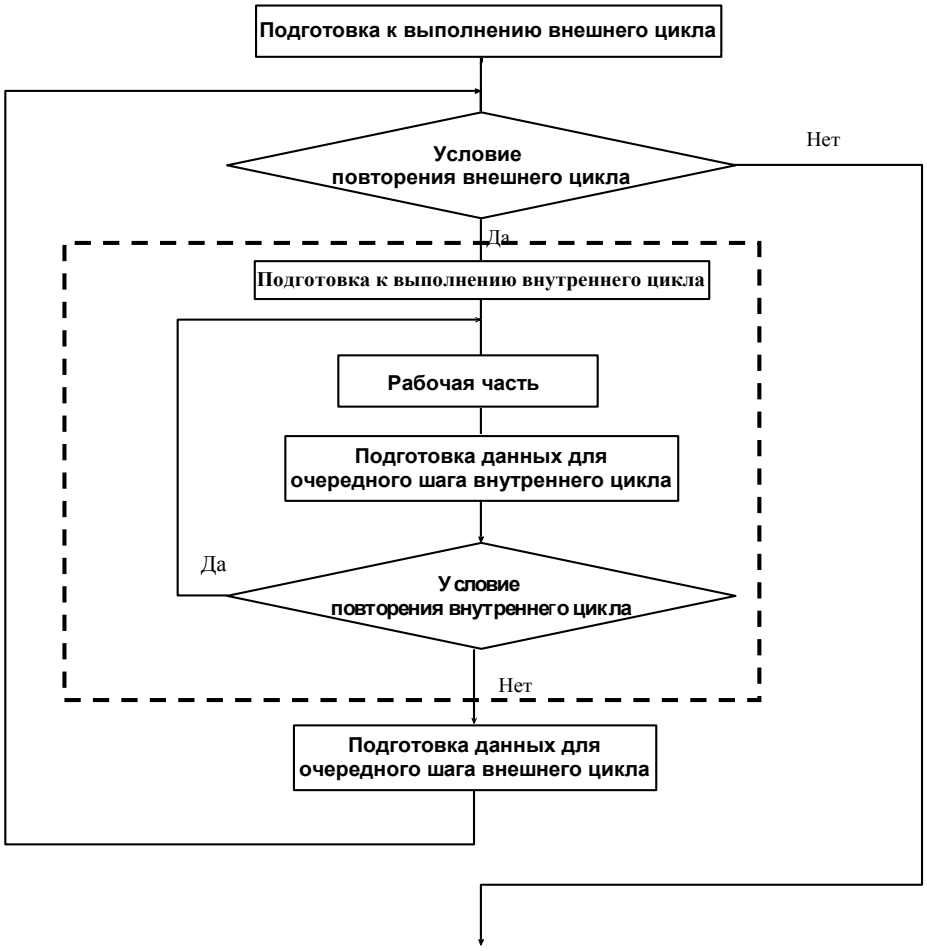


Рисунок 9.6 – Схема цикла с постусловием в цикле с предусловием

Кратность вложения циклов может равной не только двум, но и трем, четырем и т.д. Следующий пример демонстрирует трехкратное вложение циклов.

Пример 3.2. Определить количество трехзначных, натуральных чисел, сумма цифр которых равна n.

Решение. Пусть l – искомое количество натуральных трехзначных чисел; i – старшая цифра числа; j – средняя цифра; k – младшая цифра. Тогда сумма цифр трехзначного числа есть i+j+k, где i=1, 2, …, 9; j=0, 1, 2, …, 9; k=0, 1, 2, …, 9 (старшая цифра трехзначного числа не может быть нулем). Начальное значение l=0. Для вычисления всех возможных сумм i+j+k организуем вложенные циклы (кратность вложения равна трем) с предусловием с параметрами i, j, k (соответственно). Например, цикл по k в цикле по j, цикл по j в цикле по i (то есть цикл в цикле и еще раз в цикле). В самом внутреннем цикле значение l будем увеличивать на единицу, если выполнится условие i+j+k=n. Графическая схема изложенного алгоритма приведена на рис. 3.2. Рабочие части циклов выделены пунктирными линиями. Рабочей частью самого внутреннего цикла является ветвление с логическим блоком 9. Это ветвление выполнится 900 раз (10*10*9), что равно произведению числа повторений внутреннего, среднего и внешнего циклов. При этом для каждого из значений i=1, 2, …, 9 j принимает значения 0, 1, 2, …, 9. k принимает значения 0, 1, 2, …, 9 для каждого значения j.

Текст программы

```
#include <stdio.h>
#include <iostream.h>
main()
{
  int L,i,j,k,n;
  L=0;
  cout << "\nВведите целое число ";
  cin >> n;
  for(i=1; i<=9; i++)
     for(j=0; j<=9; j++)
          for(k=0; k<=9; k++)
            if (i+j+k==n) {L=L+1; cout<< "\n" <<i<<j<<k;}
  cout<<"\nКоличество трехзначных чисел, сумма цифр которого равна "<<n<<"
есть "<<L;
  cout << "\n Нажмите любую клавишу \n";
  fflush(stdin);
  getchar();
  return(0);
}
```

Графическаясхема алгоритма

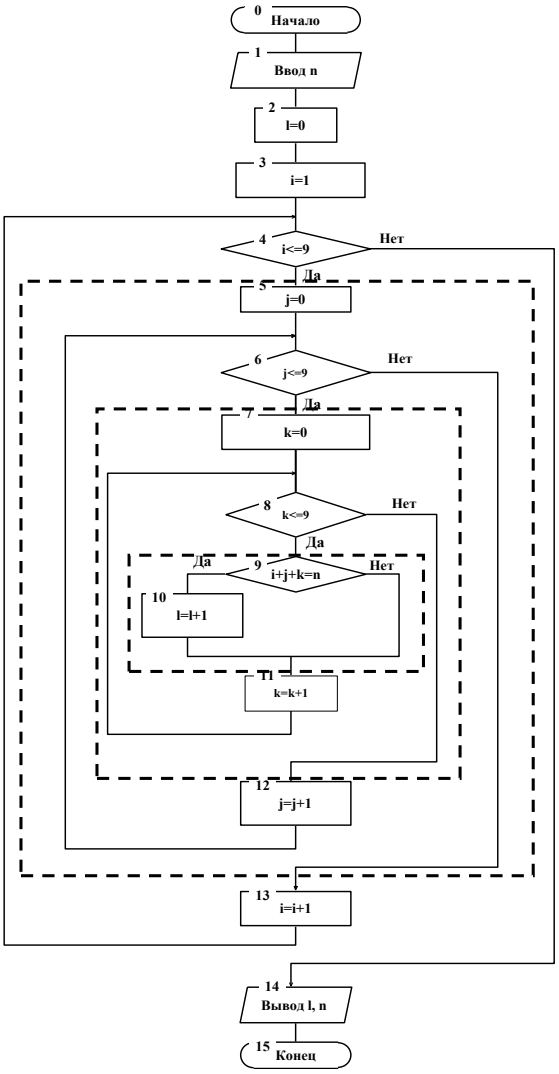


Рисунок 9.7 – Графическая схема алгоритма задачи 3.2

11 МАССИВЫ В ЯЗЫКЕ С

11.1 Понятие массива

Массив – это совокупность элементов одного типа, имеющих одно имя и расположенных в памяти ПК вплотную друг к другу. Массивы могут состоять из арифметических данных, символов, строк, структур, указателей. Доступ к отдельным элементам массива осуществляется по имени массива и индексу (порядковому номеру) элемента.

При объявлении массива в программе определяется ***имя массива***, ***тип его элементов***, ***размерность*** и ***размер***. ***Размерность*** или количество измерений массива определяется количеством индексов при обращении к элементам массива. Массивы бывают одномерные, двухмерные, трехмерные и т.д. . ***Размер массива*** – это количество его элементов по соответствующим размерностям. Общий вид объявления массива:

<имя_типа> <имя_массива> [k1] [k2] … [kn];

где ***k1, k2, …, kn*** – количество элементов массива – константы или константные выражения по *1, 2, ...,* ***n*** измерениям. Причем значения индексов могут изменяться от *0* до ***ki*** *– 1*.

Такое объявление массива называют ***статическим***, поскольку предельное количество его элементов известно заранее и оно уже не может быть изменено в ходе выполнения программы. При работе с массивами необходимо следовать следующим правилам:

- современные трансляторы языка Си не контролируют допустимость значений индексов, это должен делать программист*;*
- количество измерений массива не ограничено;
- в памяти элементы массива располагаются так, что при переходе от элемента к элементу наиболее быстро меняется самый правый индекс массива, т.е. матрица, например, располагается в памяти по строкам;
- имя массива является указателем – константой на первый элемент массива;
- операций над массивами в Си нет, поэтому пересылка элементов одного массива в другой может быть реализована только поэлементно с помощью цикла;
- над элементами массива допускаются те же операции что и над простыми переменными того же типа;
- ввод/вывод значений элементов массива можно производить только поэлементно;
- начальные значения элементам массива можно присвоить при объявлении массива.

Примеры объявления массивов:

int A [10]; //одномерный массив из 10 целочисленных величин
float X [20]; //одномерный массив из 20 вещественных величин
int a[5]={1, 2, 3, 4, 5}; //массив с инициализацией его элементов
int c[]={–1 , 2, 0, –4, 5, –3, –5, –6, 1}; // массив размерность которого бопределяется числом инициализирующих элементов

Обращения к элементам одномерного массива могут иметь вид: *A[0], A[1], A[2],…A[9], A[2*3]*.

В Си нет массивов с переменными границами. Но, если количество элементов массива известно до выполнения программы, можно определить его как константу с помощью директивы ***#define***, а затем использовать ее в качестве границы массива, например,

```
#define n 10;
Main ( )
{ int a[n], b[n]; // Объявление 2–х одномерных массивов
```

Если количество элементов массива определяется в процессе выполнения программы, используют ***динамическое*** выделение оперативной памяти компьютера.

11.2. Динамические массивы

Если до начала работы программы неизвестно, сколько в массиве элементов, в программе используют динамические массивы. Память под них выделяется с помощью оператора ***new*** во время выполнения программы. Адрес начала массива хранится в переменной, называемой ***указателем***. Например.

```
int n=20;
int *a = new int[n];
```

Здесь описан указатель ***a*** на целую величину, которому присваивается адрес начала непрерывной области динамической памяти, выделенной с помощью оператора ***new***. Выделяется столько памяти, сколько необходимо для хранения ***n*** величин типа ***int***. Величина ***n*** может быть переменной.

Примечание: Обнуление памяти при ее выделении не происходит. Инициализировать динамический массив нельзя.

Обращение к элементу динамического массива осуществляется так же, как и к элементам обычного массива. Например: a*[0], a[1], …, a[9]*.
Можно обратиться к элементу массива другим способом: *(a+1), …,*(a+9), *(a+i),* т.к. в переменной – указателе ***a*** хранится адрес начала массива. Для получения адреса, например, 9 – го его элемента к этому адресу прибавляется 9·***sizeof(int)*** (9 умножить на·длину элемента типа *int*), т.е. к начальному адресу ***a***

прибавляется смещение 9. Затем с помощью операции *****(разадресации) выполняется выборка значения из указанной области памяти.

После использования массива выделенная динамическая память освобождается с помощью опереатора:

delete [ ] имя массива.

Так например, для одномерного массива a:

delete [ ] a; .

Время "жизни" динамического массива определяется с момента выделения динамической памяти до момента ее освобождения.

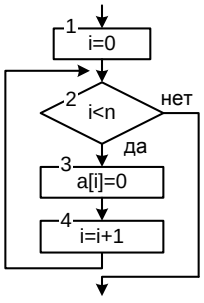
11.3 Алгоритмы обработки одномерных массивов

11.3.1 Инициализация массива

Инициализация массива – это присваивание элементам массива начальных значений. Инициализацию массива можно выполнить на этапе описания массива, как это показано в п.11.1. Но в том случае, когда начальные значения получают лишь некоторые элементы массива, а остальные вычисляются в процессе выполнения программы, в программе записывают операторы присваивания. Например:

a[0]= –1; a[1]=1.1;

Присваивание всем элементам массива одного и того же значения осуществляется в цикле. Например, чтобы всем элементам массива ***a*** присвоить значение 0, можно воспользоватся алгоритмом изображенный на рис. 11.1.



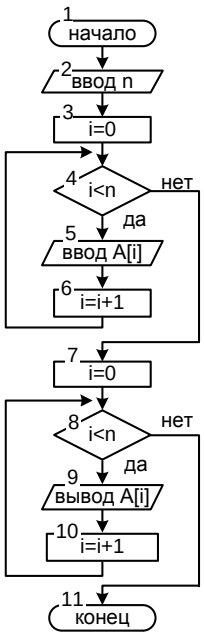

```
   for(i=0;i<n;i++)
      a[i]=0;
// или с помощью цикла while
   i=0;
   while (i<n)
   {
       a[i]=0;
        i=i+1;
   }
```

Рисунок 11.1 – Алгоритм и фрагмент программы инициализации массива

В представленном алгоритме все элементы массива в цикле последовательно инициализируются значением – 0.

11.3.2 Ввод – вывод одномерного массива

Для ввода ***n*** элементов одномерного массива, назовем его ***A***, требуется организовать цикл, для ввода каждого ***i*** – го элемента, где *i=0,1,2, …, n–1*. Аналогичный цикл требуется организовать и для вывода элементов массива. На рисунке 11.2 изображена графическая схема ввода и вывода элементов массива.




```
/* Ввод – вывод статического массива*/
#include <stdio.h>
#define n 50;
 void main()
{
  int n,i;
  float A[n];
  puts("Введите число элементов массива ");
  scanf("%d",&n);
// Ввод массива
  for (i=0; i<n; i++)
    {  printf("Введите число A[%2d]=",i);
       scanf("%f",&A[i]);
    }
  // Вывод массива
  puts("Массив A");
  for(i=0;i<n;i++)
    printf("%6.3f ",A[i]);
  printf("\n");
}
```

Рисунок 11.2 – Алгоритм и программа ввода – вывода статического массива

Ввод–вывод динамического массива осуществляется по тому же алгоритму. Из приведенного ниже примера программы ввода и вывода динамического массива видно, что отличие заключается лишь в описании массива.

```
/* Ввод – вывод динамического  массива*/
#include <stdio.h>
void main()
{
  int n,i;
  puts("Введите число элементов массива a");
```

```
 scanf("%d",&n);
 float *a=new float[n]; // Описание динамического массива
// Ввод массива
 for (i=0;i<n;i++)
   { printf("Введите число a[%2d]=",i);
     scanf("%f",a+i);      // или  scanf("%f",&a[i]);
   }
// Вывод массива
 puts("Массив a");
 for(i=0;i<n;i++)
   printf("%.3f ",*(a+i)); // или   printf("%.3f ",a[i]);
 printf("\n");
 delete[] a;            // Освобождение памяти выделенной под массив
}
```

11.3.3 Перестановка двух элементов массива

Для перестановки двух элементов массива ***x[]*** с индексами ***k*** и ***m***, необходимо использование дополнительной переменной (***tmp***), для хранения копии одного из элементов (рисунок 11.3 а), но можно обойтись и без использования дополнительной переменной ***tmp***. В этом случаи алгоритм перестановки имеет следующий вид (рисунок 11.3 б).

В большинстве случаев предпочтительнее использовать первый способ, поскольку он не содержит дополнительных вычислений, что особенно важно при перестановке вещественных чисел.

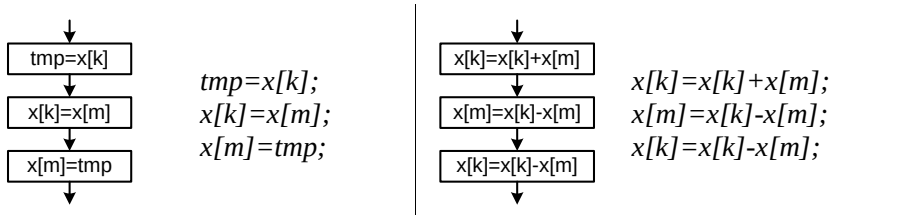


Рисунок 11.3 Алгоритм и фрагмент программы перестановки двух элементов массива с использованием дополнительной переменной (а) и без нее (б)

Пример 11.1

Переставить первый и последний элемент массива ***x[]*** местами. Количество элементов массива ***n***.

Решение

В С нумерация элементов массива начинается с нуля, поэтому номер последнего элемента массива (***n***–1) .

1 способ*: tmp=x[0]; x[0]=x[n-1]; x[n-1]=tmp;*

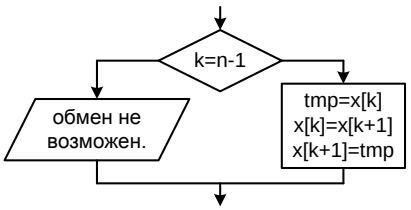
2 способ: *x[0]=x[0]+x[n-1]; x[n-1]=x[0]-x[n-1]; x[0]=x[0]-x[n-1];*

Пример 11.2

Поменять местами заданный элемент массива ***x[k]*** с последующим.

Решение

При решении этой задачи необходимо учитывать, что если заданный элемент массива ***x[k]*** является последним, то обмен выполнить не возможно, поскольку последующий элемент отсутствует.




```
if(k == n-1)
  puts("Обмен не возможен.");
else
{
  tmp=x[k];
  x[k]=x[k+1];
  x[k+1]=tmp;
}
```

Рисунок 11.4 Алгоритм и фрагмент программы перестановки заданного элемент массива ***x[k]*** с последующим

При перестановке с предыдущим элементом, обмен невозможен если заданный элемент является первым (***k=0***).

11.3.4 Вычисление суммы элементов массива

Часто возникают задачи, требующие вычислить сумму всех или некоторых элементов массива, например, сумму элементов, стоящих в массиве на заданных местах, или сумму элементов, удовлетворяющих некоторому условию (сумму только положительных элементов, сумму ненулевых элементов второй половины массива и т.д.).

Пусть ***a[]*** – заданный массив из ***n*** элементов. Сумма всех его элементов в математической форме выглядит следующим образом:

$$s = a_0 + a_1 + ... + a_{n-1} = \sum_{i=0}^{n-1} a_i \qquad (11.1)$$

Для вычисления суммы элементов части массива, например, с ***in***–го до ***ik***–го. Следует использовать формулу:

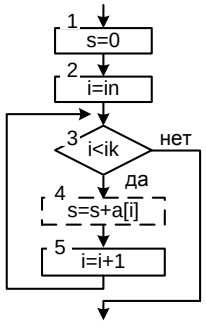
$$s = a_{in} + a_{in+1} + ... + a_{ik} = \sum_{i=in}^{ik} a_i \qquad (11.2)$$

Очевидно, что формула (11.2) получается из формулы (11.1) при ***in=0*** и ***ik***=***n***–*1*.

Алгоритм вычисления суммы состоит в следующем:

1. установить значение переменой для накопления суммы (***s***) в нулевое значение (***s=0***);
2. в цикле изменяя ***i*** от ***in*** до ***ik*** вычислить суммуэлементов массива по выражениию ***s=s+a$_i$***.

При первой итерации цикла (***i=in***) получим ***s=s+a$_{in}$= 0+ a$_{in}$***. На второй (***i=in+1***) – ***s=s+a$_{in+1}$= a$_{in}$ + a$_{in+1}$*** и т. д. На последней итерации цикла будем иметь ***s=s+a$_{ik}$= a$_{in}$ + a$_{in+1}$+…+ a$_{ik}$***. Т.е. в цикле по параметру ***i*** "старое" значение ***s***, содержащее накопленную сумму на предыдущей итерации, изменяется на значение ***a$_i$***. На рисунке 11.5 представлен алгоритм и фрагменты программ вычисления суммы элементов массива.

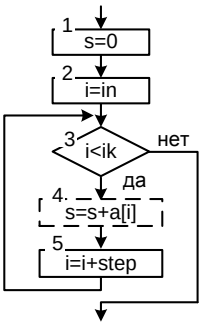



```
*/вычисление суммы элементов массива с in по ik
s=0;
for(i=in;i<ik;i++)
  s=s+a[i];     // или  s+=a[i]; или s+=*(a+i);

*/вычисление суммы всех элементов массива
s=0;
for(i=0;i<n;i++)
   s+=a[i];
```

Рисунок 11.5 Графическая схема и фрагмент программы вычисления суммы элементов массива

Если в алгоритме (рисунок 11.5) в блоке 2 записать i=0, а блоке 3 – (***i<n)***, то получим алгоритм вычисления суммы всех элементов массива.

Рассмотренный алгоритм вычисления суммы, можно применить для ***вычисления суммы элементов, стоящих в массиве на заданных местах*** (рисунок 2.6). В этом случаи шаг изменения параметра цикла определяется переменной ***step***.



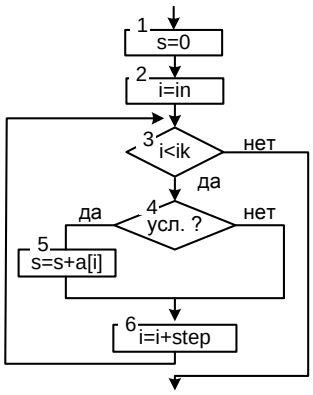

```
/* с помощью цикла  for */
s=0;
for(i=in;i<ik;i=i+step)
   s+=a[i];  // или s=s+a[i];
/* с помощью цикла  while */
s=0;   i=in;
while (i<ik)
{
   s+=a[i];
    i=i+step;
}
```

Рисунок 11.6 Графическая схема и фрагмент программы вычисления суммы элементов массива стоящих на заданных местах

Например, чтобы вычислить сумму элементов, стоящих в массиве на ***четных*** местах, необходимо "заставить" ***i*** принимать значения *1, 3, 5, …* (поскольку нумерация элементов массива в С начинается с нуля т.е. элемент массива с индексом ***a[0]*** – первый элемент массива). Для этого достаточно в блоке 2 записать ***i=1***, в блоке 3 – (***i<n)***, а в блоке 5 записать ***i=i+2*** (***step=2***). В программе на языке С соответствующий фрагмент будет выглядеть следующим образом:

```
s=0;
for(i=1;i<n;i=i+2)          // или for(i=1;i<n;i+=2)
   s+=a[i];                 // или s=s+a[i];
```

Для ***вычисления суммы только тех элементов, которые удовлетворяют некоторому условию***, необходимо в алгоритме вычисления суммы (рисунок 11.6) блок 4 заменить на ветвление, которое обеспечивает выполнение команды $s=s+a_i$ только тогда, когда условие выполнено для рассматриваемого элемента массива a_i. В этом случаи алгоритм вычисления суммы примет следующий вид (рисунок 11.7).




```
/* с помощью цикла  for */
s=0;
for(i=in;i<ik;i+=step)
   if( условие )  s+=a[i];  // или s=s+a[i];
/* с помощью цикла  while */
s=0;   i=in;
while (i<ik)
{
   if( условие )  s+=a[i];
    i=i+step;
}
```

Рисунок 11.7 Графическая схема и фрагмент программы вычисления суммы элементов массива стоящих на заданных местах

Применим полученный алгоритм для вычисления суммы положительных элементов массива стоящих на нечетных местах. Для этого в блоке 2 запишем ***i=0***, в блоке 3 (***i<n)***, в 4 условие – (***a[i]>0***), а в блоке 6 изменение параметра цикла (***step=2***) ***i=i+2***. Тогда соответствующий фрагмент программы можно записать в виде:

```
s=0;
for(i=0;i<n;i+=2)
   if( a[i]>0 )  s+=a[i];  // или s=s+a[i];
```

Рассмотрим примеры использования рассмотренных алгоритмов.

Пример 11.3.

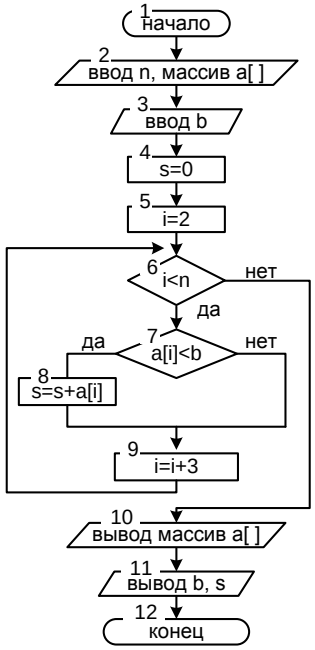
В одномерном массиве ***a*** размерностью ***n***, вычислить сумму элементов массива, меньших заданного значения ***B*** и стоящих на местах, кратных трем.

Решение.

Объединим алгоритмы ввода – вывода массива (рисунок 11.2) и вычисления суммы (рисунок 2.7). Для сокращения записи графической схемы алгоритма ввода и вывода массива, здесь и в дальнейшем используем простые блоки вида:

В алгоритме для вычисления искомой суммы рассматриваются только те элементы, которые в массиве стоят на местах, кратных трем, при этом необходимо учитывать что нумерация элементов массива в С начинается с нуля т.е. элемент массива с индексом ***a[0]*** это первый элемент массива. Таким образом, элементы стоящие на местах кратных трем – a_2, a_5, a_8, …, индекс элемента

массива (он же – параметр цикла) должен последовательно принимать значения 2, 5, 8, …, т.е. изменяться от 2 с шагом 3, что и достигается изменениями в блоках 2 и 6 алгоритма вычисления суммы (рисунок 2.7). Так в блоке 2 запишем ***i=2***, в блоке 3 (***i<n)***, а в блоке 6 – (***step=3***) ***i=i+3***. Для суммирования из рассмотренных элементов только тех, которые меньше заданного ***B***, используется ветвление с условием a_i***<B*** (блок 4). Окончательный алгоритм вычисления суммы заданных элементов примет, следующий вид (рисунок 11.8). В задаче будем использовать динамический способ задания массива. В данном примере для обращения к элементам массива используются указатели. Как уже отмечалось в разделе 1.1, имя массива является указателем на его первый элемент.



Используемые переменные:

n – число элементов массива;
a[] – динамический массив;
s – сумма элементов массива;
B – заданное число;
i – параметр цикла;

```
#include <stdio.h>
main()
{
  int n,i;
  float s, B;

  puts("Введите число элементов
массива a");
  scanf("%d",&n);

  float *a=new float[n];

  for (i=0;i<n;i++)
   { printf("Введите число a[%2d]=",i);
     scanf("%f",a+i);
   }
  puts("Введите B");
 scanf("%f",&B);
  s=0;
  for(i=2;i<n;i+=3)
      if(*(a+i)<B) s+=*(a+i);
  puts("Массив a");
  for(i=0;i<n;i++)
    printf("%.1f ",*(a+i));
  printf("\n");
  printf("Сумма чисел, меньших %.1f,
стоящих на местах, кратных 3, равна
%.2f\n",B,s);
  delete[] a; // освобождение памяти
  return(0);
}
```

Рисунок 11.8 Графическая схема и программа примера 11.3

11.3.5 Подсчет количества элементов массива, удовлетворяющих заданному условию

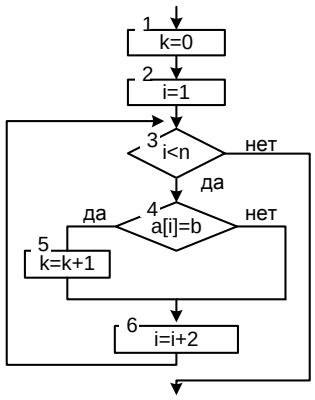
Подсчет количества элементов массива, удовлетворяющих заданному условию, производится по алгоритмам, аналогичным вычислению суммы. Отличие заключается в том, что вместо добавления элемента массива к сумме, переменная – счетчик (***k***) увеличивается на единицу (***k=k+1***). Таким образом, если в графических схемах алгоритмов, рисунок 2.5–2.7, вместо ***s=0*** и $s=s+a_i$ записать ***k=0*** и ***k=k+1***, то получим алгоритмы подсчета количества элементов массива.

Пример 11.4.

В одномерном массиве ***a*** размерностью ***n***, вычислить количество элементов равных заданному числу B и стоящих на четных местах.

Решение.

Графическая схема алгоритма решения задачи и фрагмент программы изображена на рисунке. 11.9.




```
/* с помощью цикла  for */
k=0;
for(i=1;i<n;i+=2)
   if(a[i]==b ) k++;  // или k=k+1;

/* с помощью цикла  while */
k=0;   i=1;
while (i<n)
{
   if(a[i]==b)  k++;
    i=i+2;
}
```

Рисунок 11.9 Графическая схема и программа для примера 11.4

Следующий пример показывает, как в одном алгоритме находить сумму и количество элементов, удовлетворяющих заданному условию.

Пример 11.5.

В одномерном массиве ***a*** размерностью ***n***, вычислить среднее арифметическое положительных элементов второй половины массива, стоящих на нечетных местах.

Решение

Среднее арифметическое чисел (***sr***) – частное от деления их суммы (***s***) на их количество (***k***): ***sr=s/k***, где ***k≠0***. Таким образом, задача сводится к нахождению суммы и количества положительных элементов второй половины массива, стоящих на нечетных местах. Для решения данной задачи применим алгоритм, приведенный на рисунке 11.7, в соответствии с которым можно найти сумму и количество части элементов массива (второй половины), удовлетворяющих заданному условию (положительных элементов).

Определим номер ***in*** того элемента, с которого будем просматривать элементы второй половины массива. Поскольку по условию задачи обрабатываются элементы стоящие на нечетных местах то начальное значение ***in*** тоже должно быть четным (поскольку нумерация элементов массива начинается с нуля). Значение переменной ***in*** можно определить по формуле (11.3) где пара квадратных скобок ***[ ]*** определяет операцию вычисления целой части числа:

$$in = \begin{cases} \left[\frac{n}{2}\right] + 1, если \left[\frac{n}{2}\right] - нечетное\ число; \\ \left[\frac{n}{2}\right], если \left[\frac{n}{2}\right] - четное\ число;. \end{cases} \qquad (11.3)$$

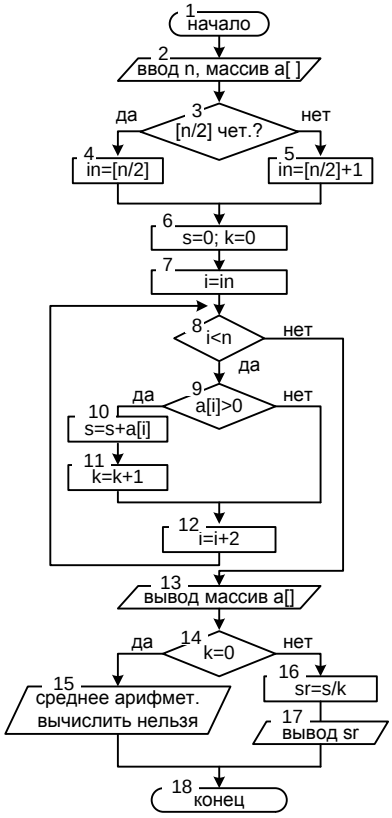
Значение ***ik*** совпадает с ***n***, поскольку массив надо просматривать до конца. Параметр цикла ***i*** (номер элемента массива) необходимо изменять с шагом 2, чтобы обеспечить четные значения индекса (значения номеров элементов массива), т.е. ***i=i+2***. Вычисление суммы и количества будем вычислять в одном цикле по параметру ***i***, что значительно сократит алгоритм.

При вычислении среднего арифметического ***sr=s/k*** необходимо избежать возможного деления на ноль, поскольку если во второй половине массива на нечетных местах не окажется положительных элементов то ***k*** будет равным нулю. Например, для массива ***X***={*2, 0, –6, 7, 9, 0, –14, –5, 0, –4, –32*}; *n=11*; *in=[n/2]+1=6*, в цикле по ***i*** будут просматриваться только выделенные элементы, а среди них нет положительных чисел, поэтому переменная ***k*** останется равной нулю. В графической схеме (рисунок 11.10) для этой цели используется ветвление, в котором проверяется условие ***k=0***. Если это условие выполнено, то выводится сообщение о том, что среднее значение не может быть вычислено, в противном случае вычисляется и выводится значение ***sr***.

В приведенном ниже фрагменте программы четность [*n/2*] определяется по остатку от деления ***n*** на 2 с помощью операции ***%*** – определения остатка целочисленного деления (**примечание:** результат операции целочисленный, т.к. ***n*** и 2 – целочисленные операнды).

Используемые переменные:	*#include <stdio.h>*
n – число элементов массива;	*main()*
a[] – статический массив;	*{*

in – первый четный номер второй половины массива;
s – сумма элементов массива удовлетворяющих условию;
k – количество элементов массива удовлетворяющих условию;
sr – среднее значение элементов массива удовлетворяющих условию;
i – параметр цикла;




```
 float a[20];
 int n, i , in, k;
 float s, sr;

 puts("Введите число элементов массива
a");
 scanf("%d",&n);

 for (i=0;i<n;i++)
  { printf("Введите число a[%2d]=",i);
    scanf("%f",&a[i]);
  }


 if ((n/2)%2==0) in=n/2;
 else in=n/2+1;

 s=0; k=0;
 for(i=in;i<n;i+=2)
    if(a[i]>0) { s+=a[i]; k++;}

 puts("Массив a");
 for(i=0;i<n;i++)
  printf("a[%2d]=%6.2f \n", i, a[i]);

 if (k==0)
  puts("Среднее арифметическое
вычислить нельзя!");
  else
  {
    sr=s/k;
    printf("Среднее ариф. положж.
элементов на нечетных. местах второй
полов. массива =%6.3f\n", sr);
  }
 return(0);
}
```

Рисунок 11.10 Графическая схема и программа для примера 11.5

11.3.6 Вычисление произведения элементов массива

Формулы, по которым вычисляется произведение элементов массива, аналогичны формулам вычисления сумм:

$$p = a_0 \cdot a_1 \cdot \ldots \cdot a_{n-1} = \prod_{i=0}^{n-1} a_i , \qquad (11.4)$$

$$p = a_{in} \cdot a_{in+1} \cdot \ldots \cdot a_{ik} = \prod_{i=in}^{ik} a_i . \qquad (11.5)$$

Поэтому вычисление произведения элементов массива выполнятся по алгоритмам аналогичным вычислению суммы. Отличие заключается в том, что начальное значение произведения ***p*** должно быть равным *1*, а в цикле по параметру ***i*** надо вычислять $p=p*a_i$. Таким образом, если в графических схемах алгоритмов, рисунок 2.5 – 2.7 вместо ***s=0*** и $s=s+a_i$ записать ***p=1*** и $p=p*a_i$, то получим алгоритмы вычисления произведения элементов массива.

Пример 11.6.

В одномерном массиве ***a*** размерностью ***n***, вычислить среднее геометрическое ненулевых элементов массива.

Решение

Среднее геометрическое ***k*** элементов массива – это корень степени ***k*** из произведения этих элементов. Таким образом, сначала необходимо вычислить произведение ***P*** ненулевых элементов массива и их количество ***k***, а затем среднее геометрическое ***Sg*** по формуле:

$$Sg = \sqrt[k]{P} = P^{\frac{1}{k}} . \qquad (11.6)$$

Например, если элементы массива равны ***A****= {1, 0, 2, 4, 0}* то –

$P = 8, \quad k = 3, \quad SG = \sqrt[3]{P} = 2.$

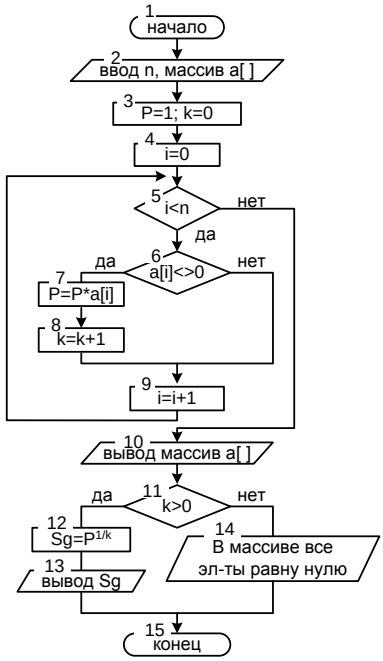
Графическая схема алгоритма решения задачи изображена на рисунке 11.11. В приведенном алгоритме в цикле по ***i*** (блоки 5 – 9) помимо вычисления произведения вычисляется и количество ненулевых элементов массива. После цикла с помощью ветвления, проверяется, есть ли в массиве ненулевые элементы (***k>0*** – условие наличия в массиве ненулевых элементов), в этом случае вычисляется и выводится среднее геометрическое. В противном случае выводится сообщение "*В массиве все элементы равны нулю*". В программе переменные ***P*** и ***Sg*** имеют вещественный тип двойной точности (***double***), т.к. произведение вещественных чисел может быть очень большим числом.

Используемые переменные:

n – число элементов массива;
a[] – статический массив;

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

main()
```

P – произведение не нулевых элементов массива;
k – количество не нулевых элементов массива;
Sg – среднее геометрическое элементов массива;
i – параметр цикла;




```
{
 float a[20];
 int n, i , k;
 double P, Sg;
 puts("Введите число элементов массива
a");
 scanf("%d",&n);
 for (i=0;i<n;i++)
  { printf("Введите число a[%2d]=",i);
    scanf("%f",&a[i]);
  }
 P=1; k=0;
 for(i=0;i<n;i++)
    if(a[i]!=0) {P*=a[i]; k++;}
 puts("Массив a");
 for(i=0;i<n;i++)
  printf("a[%2d]=%6.2f \n", i, a[i]);
 if(k>0)
 {
     if(P>0) Sg=powl(P,1.0/k);
      else Sg= –powl(fabs(P),1.0/k);
      printf("Среднее геометрическое
ненулевых  элементов массива =%.4lf
\n", Sg);
     printf("P=%.4lf  k=%d \n", P, k);
  }
  else puts("В массиве все элементы
равны нулю! ");
 return(0);
}
```

Рисунок 11.11 Графическая схема и программа для примера 11.6

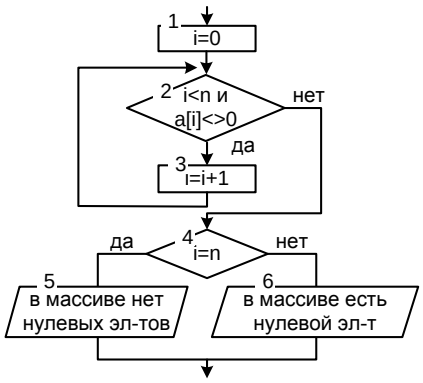
В программе для возведения ***P*** в степень ***1/k*** используется функция ***powl(основание, степень)***, первый аргумент которой может быть только положительным числом. Поэтому для отрицательного ***P*** использовано выражение $-|P|^{1/k}$, запись которого на языке С имеет вид: ***–powl(fabs(P), 1.0/k)***.

11.3.7 Поиск элементов, обладающих заданным свойством

При поиске элементов, обладающих заданным свойством, необязательно просматривать все элементы массива. Например, требуется определить, есть ли в

массиве хотя бы один нулевой элемент. Для ответа на этот вопрос, достаточно в цикле просматривать элементы массива до тех пор, пока не закончится массив или не встретится равный нулю элемент. Если, например, уже третий элемент равен нулю, то остальные элементы просматривать нет необходимости.

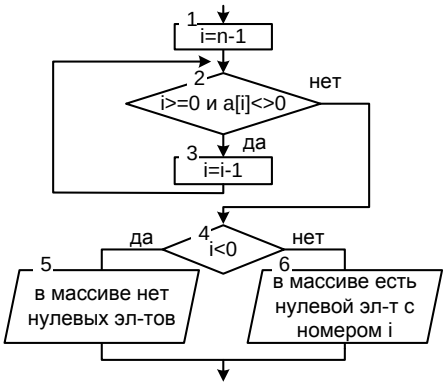
В таких случаях для просмотра массива обычно используется оператор цикла ***while*** со сложным условием. Графическая схема для рассматриваемого примера изображена на рисунке 11.12. После цикла достаточно проверить, чему равно ***i***. Если окажется, что ***i=n***, т.е. были просмотрены все элементы, то в массиве нет нулевых элементов.




```
i=0;
while(i<n && a[i]!=0) i=i+1;

If(i == n)
    puts("В массиве нет нулевых
элементов");
else
   puts("В массиве есть нулевой
элемент");
```

Рисунок 11.12 Графическая схема и фрагмент программы поиска нулевого элемента в массиве

Встречаются задачи, в которых требуется не только определить, есть ли элемент, обладающим заданным свойством в массиве, но и номер (индекс) такого элемента. Например, найти максимальный элемент в части массива, находящейся после последнего нуля. Решение задачи следует начать с вычисления индекса последнего нулевого элемента. Для определения индекса самого правого элемента, обладающего заданным свойством, массив следует просматривать с конца до тех пор, пока не закончатся элементы и текущий элемент не равен нулю (рисунок 11.13).



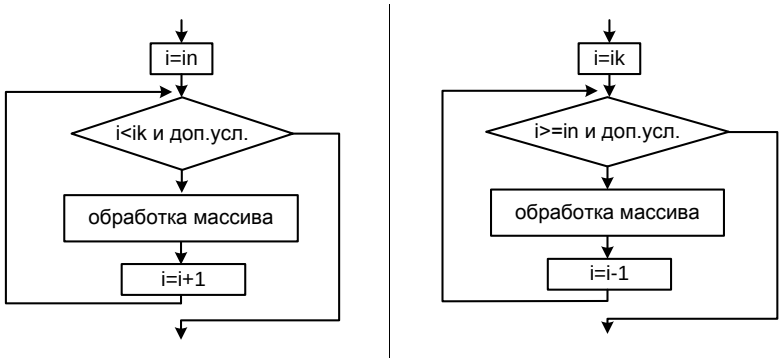

```
i=n–1;
while(i>=0 && a[i]!=0) i=i–1;

If(i<0)
   puts("В массиве нет нулевых
элементов");
 else
    printf("Индекс последнего нуля  –
%d \n", i);
```

Рисунок 11.13 Графическая схема и фрагмент программы поиска номера последнего нулевого элемента в массиве

Номер (индекс) первого встретившегося нулевого элемента можно узнать по значению параметра цикла ***i***. Этот номер можно использовать в дальнейших вычислениях например как номер начального элемента для поиска максимума.

11.3.8 Поиск в упорядоченном массиве

Упорядоченность элементов массива позволяет значительно увеличить скорость его обработки, за счет снижения числа проверяемых элементов массива. В таких алгоритмах массив проверяется пока выполняется (или не выполняется) дополнительное условие, определяющее досрочный выход из цикла. Также при составлении алгоритма необходимо учитывать возрастающим или убывающим является проверяемый массив, что оказывает влияние на то, как удобнее обрабатывать массив с начала или с конца. В общем случае алгоритм обработки упорядоченного массива имеет следующий вид – рисунок 11.14.



(а) **(б)**

Рисунок 11.14. Графический алгоритм обработки

упорядоченного массива с перебором с начала (а), с конца (б)

Как видно из блок–схемы, дополнительное условие управляет досрочным выходом из цикла. Пока дополнительное условие истина и не конец массива ***i<ik*** цикл выполняется, как только одно из условий будет не выполнено происходит выход из цикла.

Пример 11.7

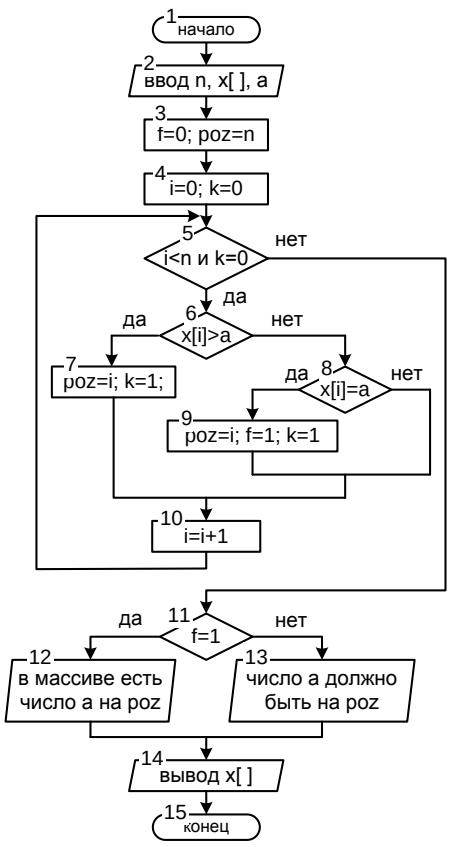
В возрастающем одномерном массиве ***X*** с количеством элементов ***n***, определить есть ли число, равное ***A,*** и на какой позиции оно находится, если числа ***A*** нет, определить место, на котором оно должно находиться, чтобы не нарушить упорядоченность массива.

Решение

В данной задаче обработку массива будем проводить с начала. Выход из цикла по дополнительному условию будет выполнен, если в массиве найден элемент больший либо равный ***A*** (***k=1***). Для индикации наличия в массиве элемента равного ***A***, введем вспомогательную переменную ***f*** с начальным значением ***f=0***. При обнаружении элемента ***A***, переменная ***f=1***. Для определения номера позиции числа ***A*** в массиве введем дополнительную переменную ***poz*** с начальным значением ***n***, т.е. предполагая, что все элементы массива меньше ***A***. При обнаружении в массиве числа большего или равного ***A*** в переменной ***poz*** сохраняется его индекс – ***i***. После выхода из цикла, по значению переменной ***f*** определяется наличие и место переменной ***A*** в массиве. Описанный алгоритм поиска и программа представлены на рисунке 11.15.

Используемые переменные:
n – число элементов массива;
a[] – статический массив;
k – переменная для досрочного выхода из цикла при нахождении элемента большего или равного a;
f – вспомогательная переменная для индикации наличия в массиве числа равного a;
poz – номер элемента массива на котором должно находится число a;
i – параметр цикла;

```
#include <stdio.h>

main()
{
  int f, k, n, poz, i, x[100], a;
  puts("Введите число элементов
массива:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
   printf("x[%2d]=",i);
   scanf("%d",&x[i]);
  }
  puts("Введите число a:");
  scanf("%d",&a);
  f=0; poz=n; k=0;
  for(i=0;i<n&&k==0;i++)
  {
```




```
   if(x[i]>a) { poz=i;k=1;}
   else
   {
    if(x[i]==a)
      {poz=i; f=1; k=1;}
   }
  }
  if(f==1)
    printf("В массиве есть число
=%d, на позиции-%d\n", a, poz);
  else
    printf("Число %d должно
находиться на позиции-%d\n" ,a,
poz);
  for(i=0;i<n;i++)
  printf("x[%d]=%d\n",i,x[i]);
   return 0;
}
```

Рисунок 11.15. Графический алгоритм и программа для примера 11.7

Описанный алгоритм можно дополнить предварительным сравнением последнего элемента массива ***X[n-1]*** с числом ***A***, если ***X[n-1]=A –*** то заданное число находится на последнем месте, а в случае выполнения ***X[n-1]>A –*** то, число ***A*** должно находится в массиве на позиции ***n***. Если ни одно из этих условий не выполнено, то это означает, что необходимо выполнить поиск числа ***A*** в массиве.

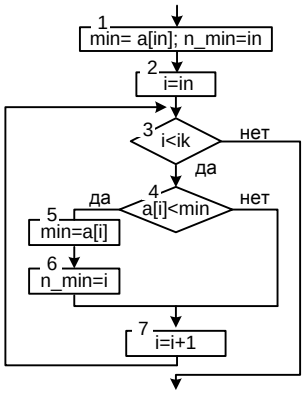
11.3.9 Поиск минимального и максимального элемента массива и его порядкового номера (индекса)

Пусть требуется найти минимальный элемент (***min***) и его индекс (***n_min***) во всем массиве (***in=0*** *и* ***ik=n***) или какой то его части (с ***in*** – го по ***ik*** – ый), в этом случаи алгоритм решения задачи можно записать так:

1. в качестве начального значения переменной ***min*** выберем любой из рассматриваемых элементов (обычно выбирают первый). Тогда ***min=a_{in}, n_min= in;***

2. затем в цикле по параметру ***i*** начиная со следующего элемента (***i=in+1, …, ik***) будем сравнивать элементы массива a_i текущим минимальным ***min***. Если окажется, что текущий (***i – ый***) элемент массива меньше минимального ($a_i <$ ***min***), то переменная ***min*** принимает значение a_i, а ***n_min*** – на ***i***: ***min***=a_i, ***n_min***= **i**.

Графическая схема алгоритма и фрагмент программы поиска минимального элемента в массиве приведены на рисунке 11.16.



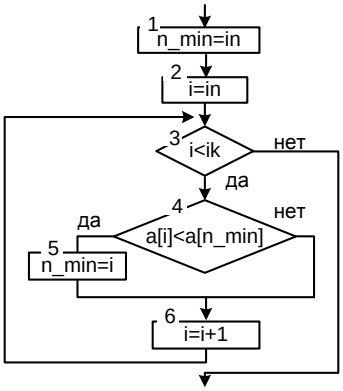

```
min=a[in];
n_min=in;
for(i=in+1; i<ik; i++)
  if(a[i]<min)
  {
     min=a[i];
     n_min=i;
  }
```

Рисунок 11.16. Графический алгоритм и фрагмент программы поиска минимального элемента в массиве

Заметим, что при наличии в массиве нескольких минимальных элементов, найден будет первый из них (самый левый минимальный элемент) при просмотре массива слева направо. Если в неравенстве $a_i <$ ***min*** знак > поменять на знак ≥, то будет найден последний из них (самый правый минимальный элемент).

Для поиска максимального элемента ***max*** и его индекса ***n_max*** используется аналогичный алгоритм, в котором сначала надо принять ***max*** $=a_{in}$, ***n_ max*** = ***in***, вместо неравенства $a_i <$ ***min*** используется неравенство $a_i >$ ***max***. В случаи выполнения условия $a_i >$ ***max*** записать в ***max*** $=a_i$ и в ***n_ max = i***.

Для поиска в массиве экстремума можно не использовать вспомогательную переменную ***min*** (***max***). В этом случаи минимальный элемент массива определяется только по его индексу ***n_min*** (***n_max***) (рисунок 11.17).



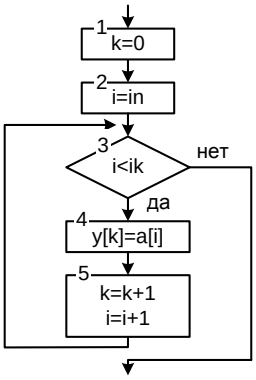

```
/*поиск минимального элемента*/
  n_min=in;
  for(i=in+1; i<ik; i++)
    if(a[i]<a[n_min])
      n_min=i;
/*поиск максимального элемента*/
  n_max=in;
  for(i=in+1; i<ik; i++)
    if(a[i]>a[n_max])
      n_max=i;
```

Рисунок 11.17. Графический алгоритм и фрагмент программы поиска минимального элемента в массиве по его индексу

Пример использования рассмотренных алгоритмов представлен в приложении 2.

11.3.10 Копирование массивов

В ряде задач для организации дополнительных или промежуточных вычислений, требуется создание копии всего массива или части его элементов. Для этого можно воспользоваться алгоритмом представленным на рисунке 11.18.




```
k=0;
for(i=in;i<ik;i++)
{
  y[k]=a[i];
  k++;
}
```

Рисунок 11.18 Алгоритм и фрагмент программы создания копии массива

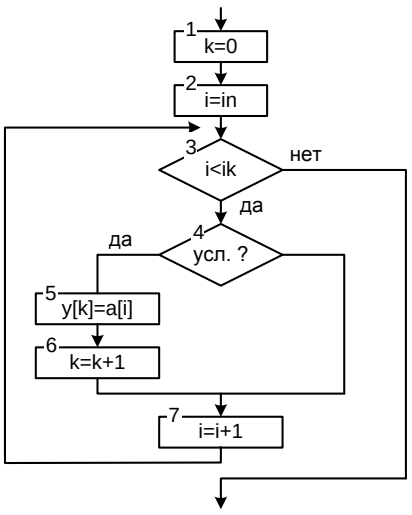
В зависимости от параметров ***in*** и ***ik***, в массив ***y[ ]*** копируются элементы из исходного массива ***a[ ]***. Так для копирования всех элементов массива ***a[ ]*** необходимо задать ***in*=*0***, ***ik*=*n*** (***n*** – количество элементов массива ***a[ ]***). При

копировании части массива, например с 3 по 9, принимаем ***in=2*** (поскольку нумерация элементов массива в С++, начинается с нуля) и ***ik=9***.

11.3.11 Формирование нового массива

В задачах формирования нового массива требуется создать массив из элементов существующего массива (массивов) удовлетворяющих заданному условию. В новый массив элементы заносятся, последовательно начиная с нулевого индекса. Максимально число элементов в формируемом массиве может достигать количества элементов в исходном массиве (массивах), минимальное значение равняется нулю. В этом случаи считается, что новый массив не сформирован.

При формировании новых массивов удобно использовать динамические массивы, поскольку число его элементов заранее не известно. Алгоритм создания нового массива схож с алгоритмом копирования (рисунок 11.19).




```
k=0;
for(i=in;i<ik;i++)
{
 if (условие)
 {
   y[k]=a[i];
   k++;
 }
}
```

Рисунок 11.19 Алгоритм и фрагмент программы формирования нового массива

Для последовательной записи элементов в новый массив используется дополнительная переменная ***k*** – *счетчик элементов в новом массиве*. Начальное значение этой переменной принимается равной *нулю*, т.е считается что в новом массиве нет элементов. При обнаружении в исходном массиве элемента удовлетворяющего заданному условию, его значение заносится в новый массив на позицию ***k***, а после счетчик элементов увеличивается на единицу (***k=k+1***). Таким образом, после обработки всего исходного массива по значению счетчика ***k*** можно определить, сформирован новый массив (***k>0***) и сколько в нем элементов (***k***).

Пример 11.8

Даны два одномерных массива ***X*** и ***Y***. Необходимо сформировать массив ***Z*** из положительных элементов массива ***X*** стоящих на четных местах и элементов массива ***Y*** больших первого элемента массива ***X***.

Решение

Если число элементов массива ***X*** – ***n***, а массива ***Y*** – ***m***, то с учетом того, что из первого массива выбираются элементы стоящие только на четных местах, максимальное число элементов в новом массиве ***Z*** может достигать ***m+n/2*** элементов. Поэтому для массива ***Z*** с помощью оператора динамического выделения памяти (***new***) выделим ***m+[n/2]*** ячейки памяти (***[n/2]*** – целая часть от деления). Начальное значение счетчика элементов нового массива ***k*** принимается равным *нулю*.

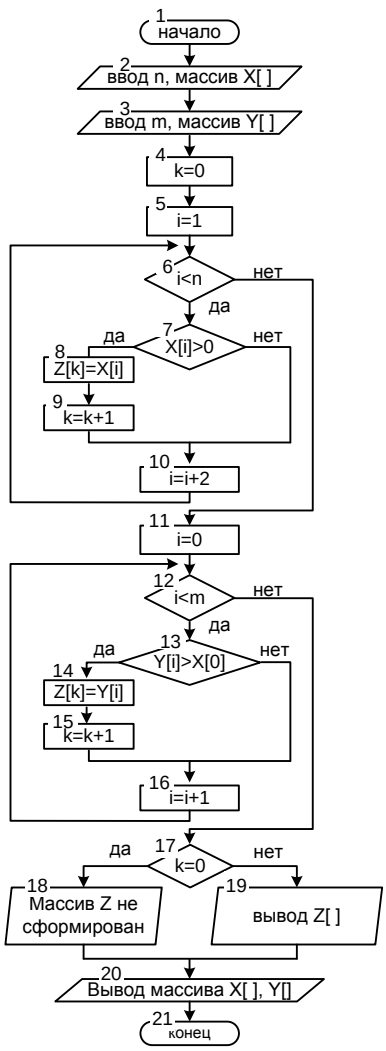
При обработке массива ***X*** необходимо проверять только элементы, стоящие на четных местах, т.е. параметр цикла ***i*** изменяется от ***in=1*** до ***ik=n*** с шагом ***2***. Условие отбора элементов из первого массива ***X[i]>0***. При обработке массива ***Y*** учитываются все его элементы, т.е. параметр цикла ***i*** изменяется от ***in=0*** до ***ik=m*** с шагом ***1***. Условие отбора элементов из второго массива – ***Y[i]> X[0]***.

Описанный алгоритм формирования нового массива и программа представлены на рисунке 11.20.

Используемые переменные:

x[] – статический (исходный) массив;
n – число элементов массива X;
y[] – статический (исходный) массив;
m – число элементов массива;
z[] – динамический (формируемый) массив
k – счетчик элементов нового массива Z;
i – параметр цикла;

```
#include <stdio.h>
main()
{
  int  k, n, m, i, x[10], y[10];
  puts("Введите число элементов
массива X:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
    printf("x[%2d]=",i);
    scanf("%d",&x[i]);
  }
  puts("Введите число элементов
массива Y:");
  scanf("%d",&m);
  for(i=0;i<m;i++)
  {
    printf("y[%2d]=",i);
    scanf("%d",&y[i]);
  }
  int *z=new int[15]; // выделение
памяти под массив Z
```




```
k=0;
for(i=1;i<n;i+=2)
{
 if(x[i]>0)
 {
  z[k]=x[i];
  k++;
 }
}
for(i=0;i<m;i++)
{
 if(y[i]>x[0])
 {
  z[k]=y[i];
  k++;
 }
}
puts("Массив X:");
for(i=0;i<n;i++)
 printf("x[%d]=%d\n",i,x[i]);
puts("Массив Y:");
for(i=0;i<m;i++)
 printf("y[%d]=%d\n",i,y[i]);
if(k==0)
  puts("Массив Z не сформиро-ван.");
else
{
   puts("Массив Z:");
   for(i=0;i<k;i++)
   printf("z[%d]=%d\n",i,z[i]);
 }
 delete[] z; // освобождение памяти
}
```

Рисунок 11.20. Графический алгоритм и программа для примера 2.8

11.4. Примеры решения задач по обработке одномерных массивов

Задача 1

В одномерном массиве ***A*** размерностью ***n***, найти количество чисел, меньших заданного ***X***, и произведение отрицательных чисел, стоящих на четных местах.

Решение

Таблица 11.1 – Таблица соответствия переменных

Переменные в задаче	Имя на языке Си	Тип	Комментарий
A[]	*A[]*	*float*	Одномерный статический массив
n	*n*	*int*	Количество элементов массива
P	*P*	*float*	Произведение отрицательных чисел
X	*x*	*float*	Заданное число
k	*k*	*int*	Количество чисел меньших *X*
–	*k1*	*int*	Количество отрицательных чисел
–	*i*	*int*	Номер элемента массива; параметр цикла

Графическая схема алгоритма

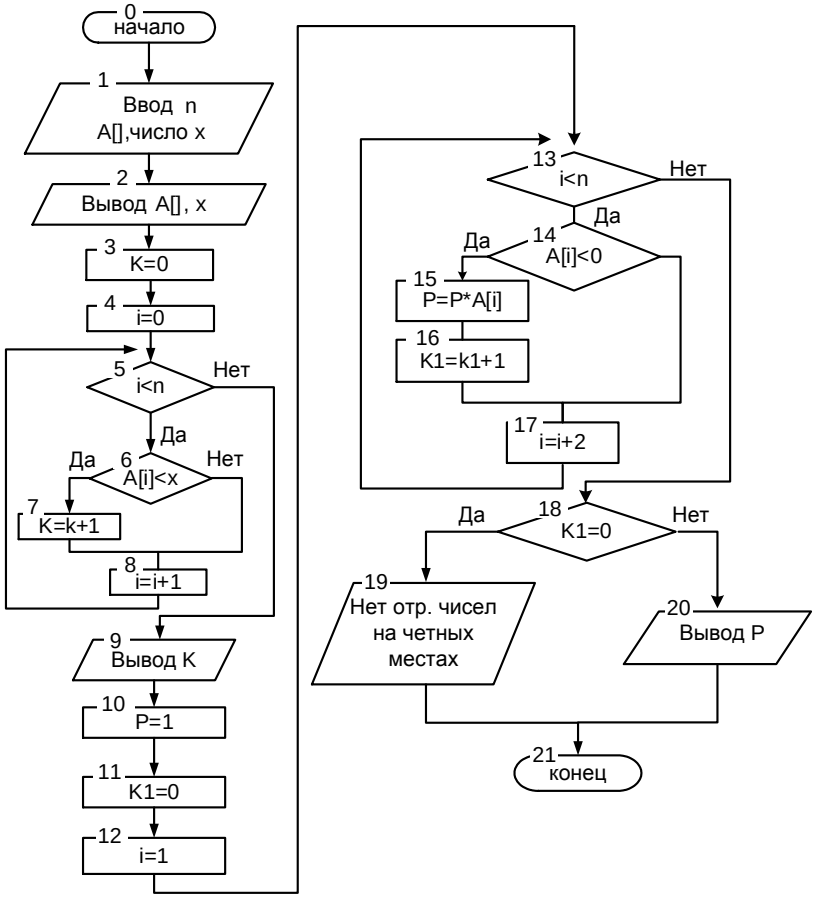


Рисунок 11.21 – Графическая схема алгоритма

Описание алгоритма

1. блоки 1 – 2 вводятся исходные данные;
2. блоки 3 – 9 вычисление и вывод количества ***K*** меньших заданного ***X***;
3. блоки 10 – 18 вычисление количества ***K1*** и произведения ***P*** отрицательных элементов массива стоящих на четных местах;
4. блоки 18 – 20 проверка на наличие в массиве отрицательных чисел на четных местах и в случаи выполнения проверки вывод их произведения ***P***.

Листинг программы

```
#include <stdio.h>
#include <math.h>
main ()
{
 float P, A[10], x;                  //Описание переменных
 int i, k1, k, n;
 puts (" Введите число x");
 scanf ("%f", &x);                   // Ввод x
 puts (" Введите число элементов массива A");
 scanf ("%d", &n);
 for (i=0; i<n; i++)                 // Ввод массива
 {
  printf("Введите число A[%d]=", i);
  scanf("%f", &A[i]);
 }
 puts("Массив A");                   // Вывод массива
 for (i=0; i<n; i++)
  printf("%.2f ", A[i]);
 printf("\n");
 printf(" x=%.3f \n", x);            // Вывод x
 k=0;
 for (i=0; i<n; i++)
  if (A[i]<x) k=k+1;                 // Определение кол-ва чисел <x
 printf(" k=%d\n" ,k);
 P=1;
 k1=0;
 for (i=1; i<n; i=i+2)
   if (A[i]<0)
   {                                 // Вычисление произведения
    P=P*A[i];                        // отрицательных чисел
    k1=k1+1;                         // стоящих на четных местах
    }
 if (k1==0)
   printf("В массиве на чётных местах нет отрицательных чисел \n");
 else
 printf(" P=%6.2f \n" ,P);
 }
```

Тесты

1)

Массив A

3.00 6.00 10.00 -7.00 -3.00 -7.00 -5.00 17.00 6.00 10.00
x=10.000
k=7
P= 49.00

2)

Массив A
2.00 9.00 4.00 6.00 7.00 8.00
x=0.000
k=0
В массиве на чётных местах нет отрицательных чисел

Задача.2.

В одномерном массиве ***A*** размерностью ***n***, найти максимальный элемент, расположенный между первым и последним нулевыми элементами массива.

Решение

Таблица 11.2 – Таблица соответствия переменных

Переменные в задаче	Имя на языке Си	Тип	Комментарий
max	*max*	*int*	Максимальное число
A[]	*A[]*	*int*	Одномерный динамический массив
n	*n*	*int*	Количество элементов
–	*t1*	*int*	Индекс первого нулевого элемента
–	*t2*	*int*	Индекс последнего нулевого элемента
–	*i*	*int*	Номер элемента массива; параметр цикла

Описание алгоритма

1. блоки 1 – 2 ввод/вывод исходного массива ***A[]***;
2. блоки 3 – 5 поиск первого нулевого элемента в массиве;
3. блоки 6 – 7 проверка на наличие в массиве хотя бы одного нулевого элемента и в случаи их отсутствия вывод сообщения «*В массиве нет нулей*» и переход на конец алгоритма;
4. блоки 8 – 12 поиск последнего нулевого элемента в массиве и сохранение позиции первого нуля в ***t1***, а последнего в ***t2***;
5. блоки 13 – 14 проверка расположения нулевых элементов в массиве и в случаи ошибки вывод сообщения «*В массиве только один ноль или они располагаются друг за другом*» » и переход на конец алгоритма;
6. блоки 15 – 21 в случаи выполнения проверки (блок 13) поиск максимального элемента между крайними нулевыми элементами и вывод полученного результата.

Графическая схема алгоритма

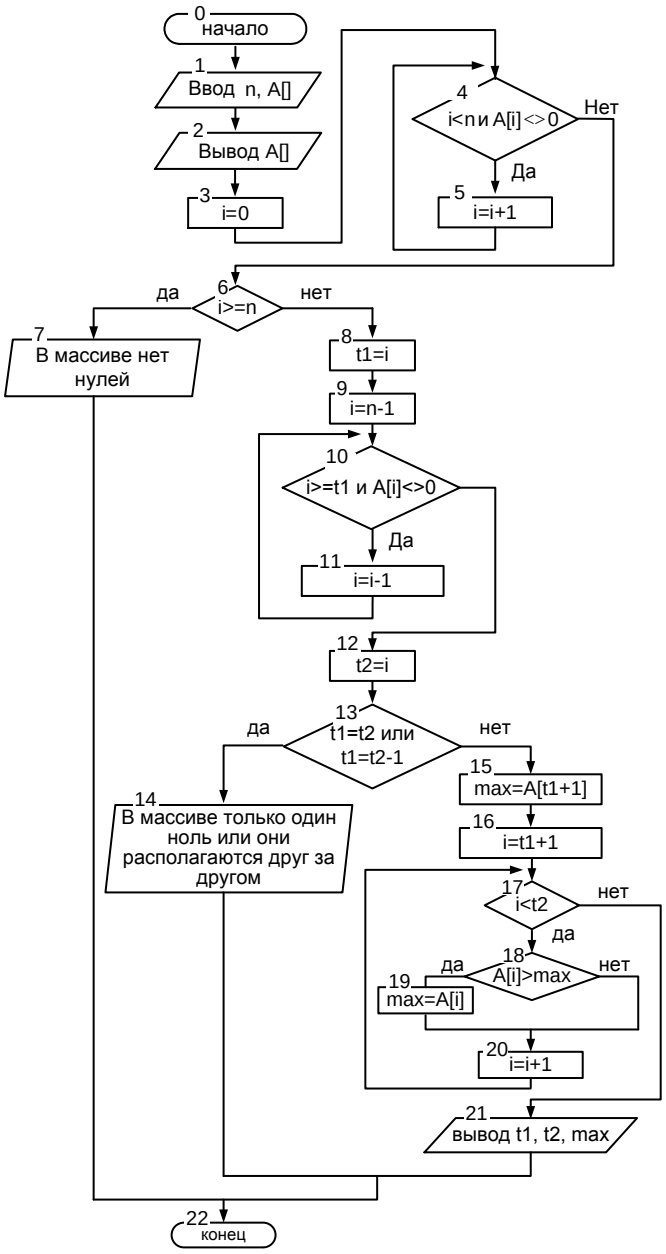


Рисунок 11.22 – Графическая схема алгоритма

Листинг программы

```
#include <stdio.h>
main()
{
  int i,t1,t2,n,max;
  puts("Введите число элементов массива A");
  scanf ("%d", &n);
  int*A=new int[n];   //выделение памяти под массив
  for(i=0;i<n;i++)    //ввод массива
  {
   printf("a[%2d]=",i);
   scanf("%d",&A[i]);
  }
  puts("Массив A:");
  for(i=0;i<n;i++)    //вывод массива
   printf("a[%d]=%d\n",i,A[i]);
  i=0;
  while(i<n && A[i]!=0) i=i+1;  //поиск позиции первого нуля
  if(i>=n)
   printf("В массиве нет нулей \n");
  else
  {
   t1=i;
   i=n-1;
   while(i>=t1 && A[i]!=0) i=i-1; //поиск позиции последнего нуля
   t2=i;
   if(t1==t2 || t1==t2-1)
      printf("В массиве только один ноль или они располагаются друг за
другом\n");
    else
    {
      max=A[t1+1];
      for(i=t1+1;i<t2;i++)
       if(A[i]>max) max=A[i];   //поиск максимального элемента
      printf("t1=%d t2=%d max=%d \n",t1,t2,max);
    }
  }
 delete[] A;          // освобождение динамической памяти
}
```

Тесты

1)
Массив A:
a[0]=1
a[1]=3
a[2]=0
a[3]=x6
a[4]=-3
a[5]=7
a[6]=4
a[7]=0
a[8]=1
t1=2 t2=7 max=7

2)
Массив A:
a[0]=0
a[1]=1
a[2]=-2
a[3]=4
a[4]=0
t1=0 t2=4 max=4

3)
Массив A:
a[0]=1
a[1]=2
a[2]=0
a[3]=5
a[4]=0
a[5]=3
a[6]=8
a[7]=0
a[8]=3
a[9]=1
t1=2 t2=7 max=8

4)
Массив A:
a[0]=1
a[1]=2
a[2]=3
a[3]=4
В массиве нет нулей

5)
Массив A:
a[0]=1
a[1]=0
a[2]=2
a[3]=3
a[4]=4
В массиве только один ноль или они располагаются друг за другом
t1=1 t2=1 max=0

6)
Массив A:
a[0]=1
a[1]=2
a[2]=0
a[3]=0
a[4]=5
a[5]=2
В массиве только один ноль или они располагаются друг за другом

11.5 Двумерные массивы

Массив - это совокупность элементов одного типа, расположенных в памяти вплотную друг за другом. Каждый массив имеет имя. Массивы могут состоять из арифметических данных, символов, строк, структур, указателей файлов. Доступ к отдельным элементам массива осуществляется по имени массива и индексу (порядковому номеру) элемента.

При объявлении массива в программе определяется ***имя массива***, ***тип его элементов***, ***размерность*** и ***размер***. ***Размерность*** или количество измерений массива определяется количеством индексов при обращении к элементам массива. Массивы бывают одномерные, двухмерные, трехмерные и т.д. .

Размер массива - это количество его элементов.

Общий вид объявления массива:
<имя_типа> <имя_массива> [k1] [k2] … [kn] ;
где k1, k2, …, kn - количество элементов массива - константы или константные выражения по 1, 2, …, n измерениям. Причем значения индексов могут изменяться от 0 до ki-1.

Примеры объявления массивов:
) int C[10] [20] [15]; //трёхмерный целочисленный массив
Примеры обращение к элементам:
float B[50][20]; //двумерный вещественный массив из 50 строк и 20 столбцов.
Примеры обращения к его элементам :
B[0] [0], B[0] [1] ,.., B[i] [j], …, B[4] [19], …

C[i] [j] [k], ..., C[2*i] [0] [3*k+1], …

При работе с массивами надо пользоваться следующими правилами:

- Современные трансляторы языка Си не контролируют допустимость значений индексов. Это должен делать программист.
- Кол-во измерений массива не ограничено.
- В памяти элементы массива располагаются так, что при переходе от элемента к элементу наиболее быстро меняется самый правый индекс массива, т.е. матрица, например, располагается в памяти по строкам.
- Операций над массивами в Си нет, поэтому пересылка элементов одного массива в другой может быть реализована только поэлементно с помощью цикла.
- Над элементами массива допускаются те же операции что и над простыми переменными того же типа.
- Ввод-вывод значений элементов массива можно производить только поэлементно.
- Начальные значения элементам массива можно присвоить при объявлении массива.

Например:
1) int a[10]={1,2,3,4,5,6,7,8,9,10};
2) int c[ ]={-1,2,0,-4,5,-3,-5,-6,1};

Если размер одномерного массива не указан (даны только по [ ]), то размер определяется количеством инициированных значений, т.е. кол-вом значений в { }

```
3) int  w[3] [4] ={{1,2,3,4},  // элем. 0-ой строки
                  {-3,2,-1,1},  // эл-ты 1-ой строки
                  {2,10,1,-6}};  // эл-ты 2-ой строки
```

При этом если инициализирующих значений меньше, чем элементов массива, то остаток массива обнуляется; если больше - лишние значения не используются.

- В Си нет массивов с переменными границами. Но, если количество элементов массива известно до выполнения программы, можно

определить его как константу с помощью директивы #define , а затем использовать ее в качестве границы массива, например,

```
#define  n  10;
 Main  ( )
    { int a[n], b[n];     // Объявление 2-х одномерных массивов
      …
```

Если количество элементов массива определяется в процессе выполнения программы, используют ***динамическое*** выделение оперативной памяти компьютера (ОП).

11.6 Динамические массивы

Если до начала работы программы неизвестно, сколько в массиве элементов, в программе используют динамические массивы. Память под них выделяется с помощью операции ***new*** или функции ***malloc*** во время выполнения программы. Адрес начала массива хранится в переменной, называемой ***указателем***. Например.

Примечание. *Обнуление памяти при ее выделении не происходит.*

*Инициализировать динамический массив нельзя***.**

Обращение к элементу динамического массива осуществляется также, как и к элементам обычного массива.

Например, a[0], a[1], …, a[19], c[2][5] и т.д.

Другой способ обращения к элементу b[i][j] двумерного массива имеет вид:

((b+i)+j).

Поясним это. Двумерный массив представляет собой массив массивов, т.е., это массив, каждый элемент которого является массивом. Имя двумерного массива также является константным указателем на начало массива. Например, int b[5][10] является массивом, состоящим из 5-ти массивов. Для обращения к b[i][j] сначала требуется обратиться к i-ой строке массива, т.е., к одномерному массиву b[i]. Для этого надо к адресу начала массива b прибавить смещение, равное номеру строки i: b+i (при сложении указателя b с i учитывается длина адресуемого элемента, т.е., **i·(n·sizeof(int))**, т.к. элементом массива b[i] является строка, состоящая из n элементов типа int). Затем требуется выполнить разадресацию: *(b+i). Получим массив из 10 элементов. Далее требуется обратиться к j-му элементу полученного массива Для получения его адреса опять применяется сложение указателя с j: ***(b+i)+j** (на самом деле прибавляется **j·sizeof(int)**). Затем применяется операция разадресации: ***(*(b+i)+j)**. Т.о., получаем формулу для вычисления адреса элемента b[i][j]:

$$b+k\cdot i\cdot n+k\cdot j=b+k\cdot(i\cdot n+j),$$

где k – длина в байтах одного элемента массива, b – адрес начала массива. Эта формула может быть использована в дальнейшем, например для

организации передачи в подпрограмму двумерного массива переменной размерности.

Приведем универсальный способ выделения памяти под двумерный массив, когда обе его размерности задаются на этапе выполнения программы:

```
int i, m, n;   //i – номер строки, m, n – количество строк и столбцов
puts("Введите m и n");
scanf("%d %d",&m, &n);   // Ввод m, n
int **b=new int *[m];              //1
for (i=0; i<m; i++)                 //2
     b[i]=new int[n];               //3
```

Здесь в операторе //1 объявляется переменная типа "указатель на указатель на int" и выделяется память под массив указателей на строки массива (m строк). В операторе //2 организуется цикл для выделения памяти под каждую строку массива. В операторе //3 выделяется память под каждую строку массива и i-му элементу массива указателей на строки присваивается адрес начала участка памяти, выделенного под строку двумерного массива. Каждая строка массива состоит из n элементов типа int.

Примечание. Для выделения динамической памяти для вещественного двумерного массива достаточно в приведенном фрагменте программы в стоках //1, //3 имя типа int поменять на имя типа float.

Освобождение выделенной динамической памяти.

Динамическая память освобождается с помощью операции ***delete[] имя массива***, например, для одномерного массива, ***delete[]*a**; Освобождение памяти, выделенной для двумерного массива ***b***, выглядит следующим образом:

```
for (i=0; i<n; i++)
    delete [ ] b[i];
 delete [ ] b;
```

Время "жизни" динамического массива определяется с момента выделения динамической памяти до момента ее освобождения.

1. Алгоритмы обработки двумерных массивов

Ввод и вывод матрицы

```
//Ввод матрицы
float  x[10][10];
puts("Введите m,n");
scanf("%d %d",&m,&n);
for(i=0;i<m;i++)
  for(j=0;j<n;j++)
   {
     printf("Введите x[%d][%d]=",i,j);
     scanf("%f",&x[i][j]);
   }

// Вывод матрицы
  puts("Матрица x:");
  for(i=0;i<m;i++)
    { for(j=0;j<n;j++)
          printf("%8.2f",x[i][j]);
      printf("\n");
    }
```

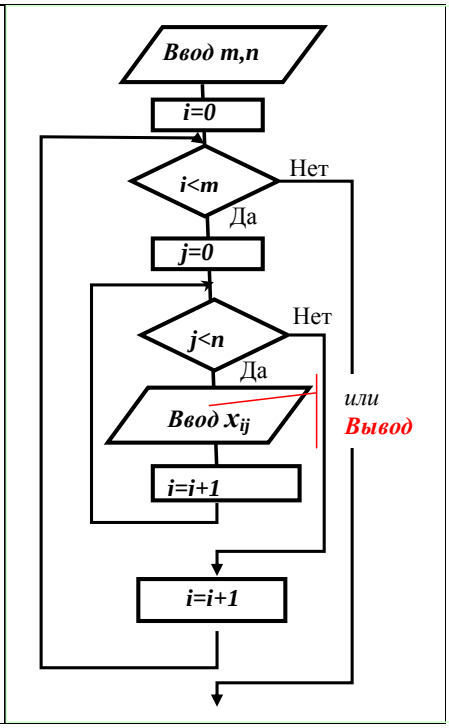


Рисунок 11.23 – Графическая схема алгоритма

Примечание. ***Ввод-вывод динамической матрицы отличается от ввода-вывода статической матрицы лишь описанием матрицы***

2.2 Определить количество элементов, больших заданного A и расположенных в строках с нечетными номерами

Графическая схема алгоритма

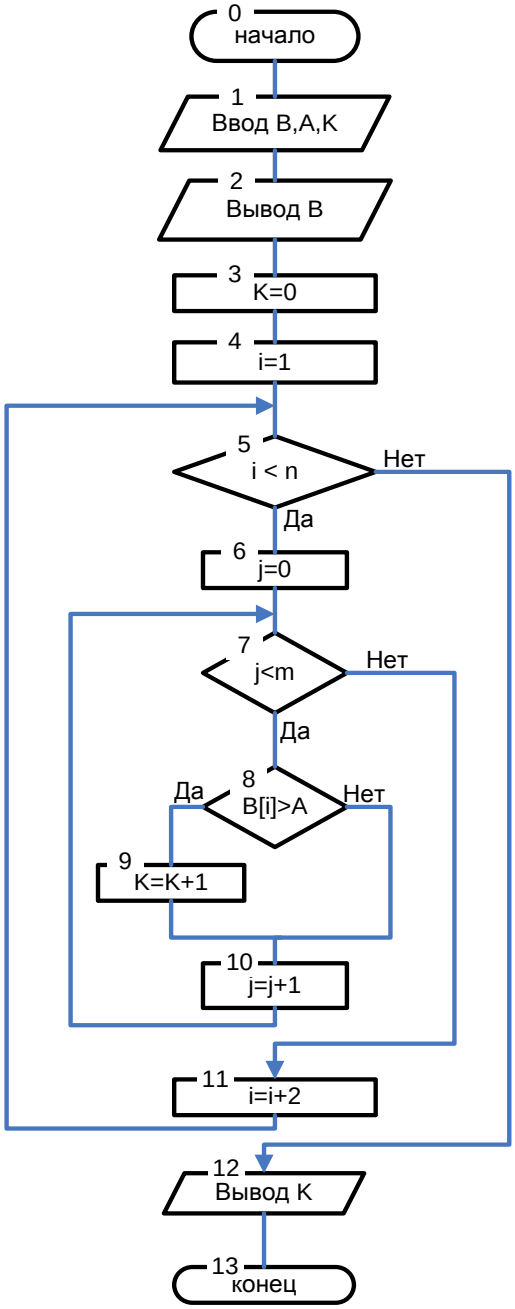


Рисунок 11.24 – Графическая схема алгоритма

Таблица 11.3 – Таблица соответствия:

Переменные в задаче	Имя на языке Си	Тип	Комментарий
K	K	int	Искомое количество элементов
B	B	float	Двумерный статический массив
A	A	float	Заданное число
-	i	int	Номер строки
-	j	int	Номер столбца

```
/* Определение количества элементов, больших заданного A и
расположенных в строках с нечетными номерами */
 # include <stdio.h>
 # include <math.h>
 void main ( )
{
  int i, j, m, n, K;
  float B [10][10];
  float A; //Описание переменных
  printf (“ Введите число строк и столбцов”);
  scanf (“%d %d”, &m, &n);
  for(i=0; i<m; i++)
    for(j=0; j<n; j++)
    {
      printf(“Введите B[%d, %d]=”, i, j);
      scanf(“%f”, &B[i] [j]);
    }

 puts("Матрица B:");
 for(i=0;i<m;i++)
   { for(j=0;j<n;j++)
         printf("%8.2f",B[i][j]);
     printf("\n");
   }

   puts (“ введите число A”);
   scanf (“%f”, &A);
   K=0;
   for ( i=1; i<m; i=i+2)
     for ( j=0; j<n; j++)
       if( B[i][j]>A) K=K+1;
   printf(“%d \n”, K);
  }//конец main
```

Тесты:

1) B= -3; 8; -2; 10; 7; 82; -4
-9; 0; -3; 85; 3; 40; -9
-1; 8; 7; -95; 4; -5; 67
A= 4; K=7.

2) B= -4; -7; 8; 9; 2; -8; -5
-2; 10; 0; 9; -8; 7; -5
-7; 6; -9; 7; 0; -3; 7
A=10; K=0.

2.3 Поиск в матрице строки с максимальной суммой

Графическая схема алгоритма

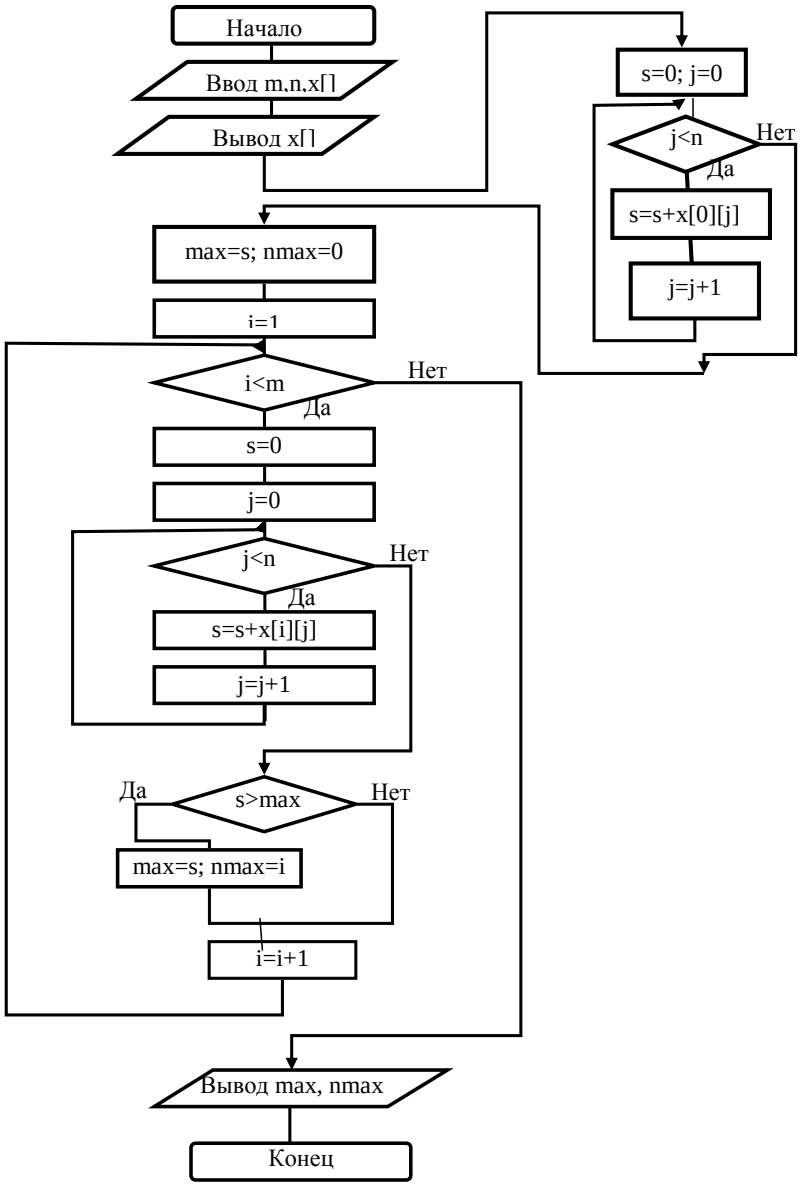


Рисунок 11.25 – Графическая схема алгоритма

Текст программы

```
/*  Строка с максимальной суммой*/
#include <stdio.h>
main()
{
  int m, n, nmax, i, j;
  float max, s, x[10][10];
  puts("Введите m, n");
  scanf("%d %d", &m, &n);

  for(i=0;i<m;i++)
    for(j=0;j<n; j++)
        { printf ("Введите x[%d][%d]=", i, j);
          scanf ("%f", &x[i][j]);
        }

  puts("Матрица x:");
  for(i=0;i<m; i++)
   { for(j=0;j<n; j++)
         printf("%8.2f",x[i][j]);
     printf("\n");
   }

  s=0;
  for(j=0; j<n; j++)
    s =s+x[0][j]; //Нач. знач. max – сумма элем. 0-ой стр.

  max=s;  nmax=0;
  for(i=0;i<m;i++)
    { s=0;
      for(j=0;j<n;j++)  s+=x[i][j];    // Сумма элем. строк
      if(max<s)
         { max=s;   nmax=i;}
    }

  printf("Максимальная сумма %.2f в строке %d\n",max,nmax);
  fflush(stdin);  getchar();
  return(0);
}
```

2.4 Пример обработки динамического двумерного массива

Задача. Определить количество строк матрицы, в которых суммы всех элементов отрицательные. Массив объявить как динамический.

Решение.

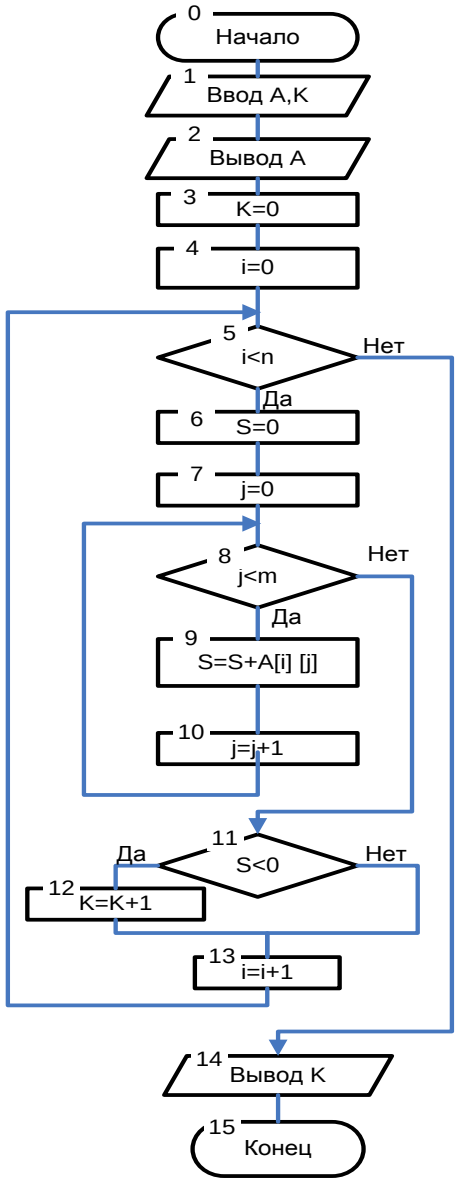


Рисунок 11.26 – Графическая схема алгоритма

Таблица 11.4 – Таблица соответствия.

Переменные в задаче	Имя на языке Си	Тип	Комментарий
S	S	float	Сумма элементов i-той строки
A	A	float	Двумерный динамический массив
K	K	float	Количество искомых строк
-	n	int	Количество всех строк в матрице
-	m	int	Количество столбцов
-	i	int	Номер строки
-	j	int	Номер столбца

```
/*Пример обработки двумерного динамического массива*/
 # include <stdio.h>
 # include <math.h>
 void main ( )
{
  int i, j, m, n;
  float K, S; //Описание переменных
  puts (“ введите n, m”); // Вывод сообщения
  scanf (“%d %d”, &n, &m); // Ввод исх.числа строк и столбцов
  float**A=new float*[n];
  for(i=0; i<n; i++)
    A[i]=new float[m];
  for(i=0; i<n; i++)
     for(j=0; j<m; j++)
       {
         printf(“Введите A[%d; %d]=”, i, j);
         scanf(“%f”, &A[i] [j]);
       }
   K=0;
   for(i=0; i<n; i++)
     {
         S=0;
         for(j=0; j<m; j++)
            S=S+A[i] [j];
         if(S<0) K=K+1;
     }
   printf(“%f \n”, K);
   for(i=0; i<n; i++) //Освобождение динамической памяти
     delete[]A[i];
```

```
 delete[]A;
}
```

Тесты:

1) A= $\begin{pmatrix} -3 & -2 & 2 & 6 & -3 \\ 6 & 7 & -1 & 20 & -4 \\ -4 & -2 & -3 & 6 & -1 \end{pmatrix}$ K=2.

2) A= $\begin{pmatrix} -4 & -2 & 4 & 6 & 4 \\ 3 & 5 & 7 & 2 & 0 \\ 5 & 0 & -2 & 9 & 0 \end{pmatrix}$ K=0.

2.5 Определить, есть ли в матрице столбец, содержащий хотя бы один нулевой элемент

Решение.

Введем две вспомогательные переменные t и w:

$$t = \begin{cases} 1, & \text{если столбец не найден;} \\ 0, & \text{если столбец найден .} \end{cases}$$

$$w = \begin{cases} 1, & \text{åñëè â ñòîëáöå íå íàéäåí ýëåìåíò ðàâíûé íóëþ;} \\ 0, & \text{åñëè â ñòîëáöå íàéäåí ýëåìåíò ðàâíûé íóëþ .} \end{cases}$$

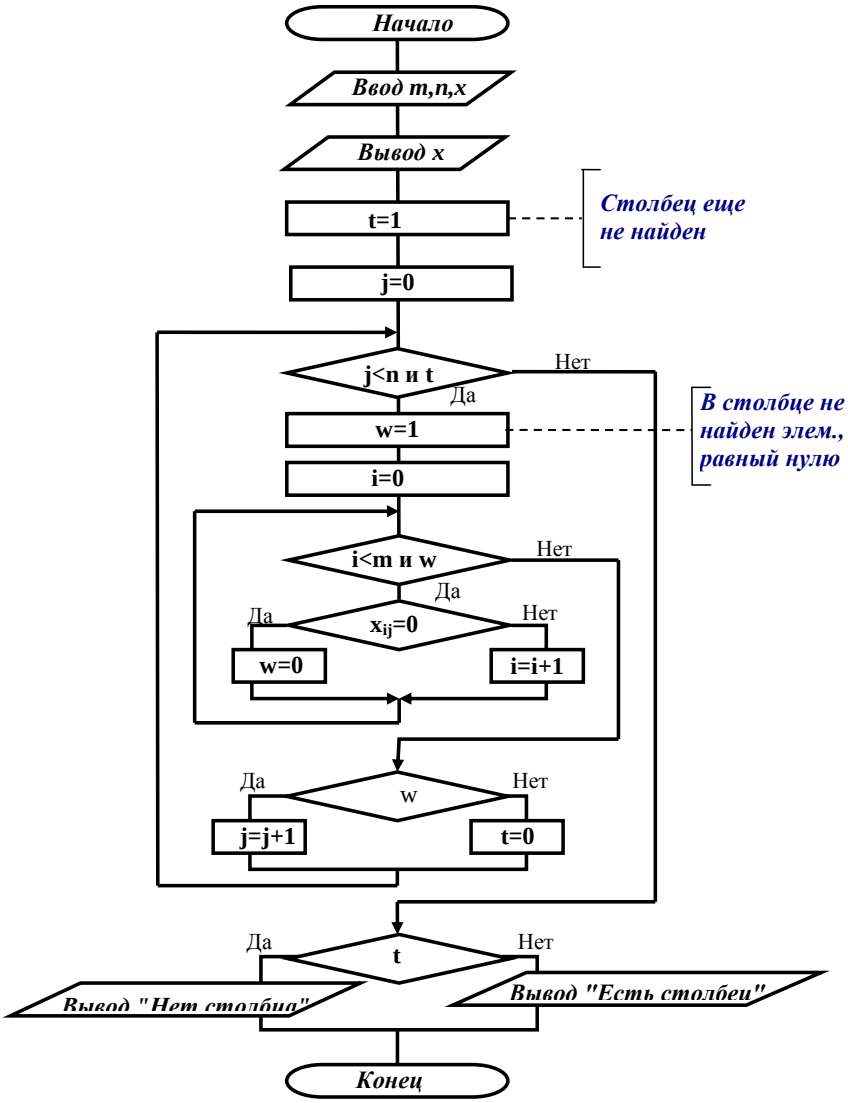


Рисунок 11.27 – Графическая схема алгоритма

Текст программы

```
#include <stdio.h>
#include <conio.h>
main()
{
  int m,n,i,j,t,w;
  float  x[10][10];
  puts("Введите m,n");
  scanf("%d %d",&m,&n);
  for(i=0;i<m;i++)
     for(j=0;j<n;j++)
          {
           printf("Введите x[%d][%d]=",i, j);
            scanf("%f", &x[i][j]);
          }
  puts("Матрица x:");
  for(i=0;i<m;i++)
    {
     for(j=0;j<n;j++)
           printf("%8.2f",x[i][j]);
     printf("\n");
    }
  t=1; j=0;    //Столбец еще не найден
  while(j<n && t)
   {
    w=1; i=0;   //В столбце не найден элем., равный нулю
    while(i<m && w)
          if(x[i][j]==0) w=0;
               else    i++;
    if(w)   j++;  // В столбце нет равного 0 элем.
      else  t=0;  // В столбце есть равный 0 элем.
   }
  if(t)  printf("Нет\n");
    else printf("Есть %d %d\n",i,j);
  getch();
  return(0);
}
```

2.5 Обработка элементов квадратных матриц относительно главной и побочной диагоналей

```
/* "Разминка". Фомирование матрицы */
#include <stdio.h>
main()
{
  const m=10;
  int i,j,N;
  int  x[m][m];
  /* Обнуление матрицы  */
  for(i=0;i<m;i++)
     for(j=0;j<m;j++)
          x[i][j]=0;
  puts("Номер варианта?");
  scanf("%d",&N);
  printf("Вариант %d:\n",N);
  switch(N)
   { case 1:{    /*   Вариант 1        */
              for(i=0;i<m;i++)
                  for(j=0;j<i+1;j++)
                     x[i][j]=i-j+1;
              break;}
     case 2:{    /*   Вариант 2        */
              for(j=0;j<m;j++)
                  for(i=j;i<m;i++)
                     x[i][j]=j+1;
              break; }
    case 3: {    /*   Вариант 3        */
              for(i=0;i<m;i++)
                  for(j=i;j<m;j++)
                     x[i][j]=i+1;
              break; }
    case 4: {    /*   Вариант 4        */
              for(i=0;i<m;i++)
                  for(j=i;j<m;j++)
                     x[i][j]=j-i+1;
              break; }
    case 5:{     /*   Вариант 5        */
              for(i=0;i<m;i++)
                  for(j=0;j<m;j++)
                     x[i][j]=i+1;
              break; }
```

```
 case 6:{     /*   Вариант 6        */
          for(i=0;i<m;i++)
              for(j=i;j<m;j++)
                if (i==j) x[i][j]=1;
                     else x[i][j]=2;
          break; }
  case 7:{   /*   Вариант 7        */
          for(i=0;i<m;i++)
              for(j=0;j<i+1;j++)
                x[i][j]=1;
          break; }
case  8:{      /*   Вариант 8        */
          for(i=0;i<m;i++)
              for(j=0;j<i+1;j++)
                x[i][j]=(i-j+1)*(i-j+1);
          break; }
 case 9:{      /*   Вариант 9        */
          for(j=0;j<m;j++)
              for(i=j;i<m;i++)
                x[i][j]=(j+1)*(j+1);
          break; }
 case 10:{     /*   Вариант 10       */
          for(j=0;j<m;j++)
              for(i=0;i<m;i++)
                if((i+j)%2==0) x[i][j]=1;
                          else x[i][j]=0;
          break; }
 case  11:{    /*   Вариант 11       */
          for(i=0;i<m;i++)
              for(j=m-i-1;j<m;j++)
                x[i][j]=1;
          break; }
 case 12:{     /*   Вариант 12       */
          for(i=0;i<m;i++)
              for(j=0;j<m;j++)
                x[i][j]=(i+1)*(i+1);
          break; }
 case 13:{     /*   Вариант 13       */
          for(i=0;i<m;i++)
              for(j=m-i-1;j<m;j++)
                if (i+j==m-1) x[i][j]=1;
                          else x[i][j]=2;
          break; }
 case 14:{     /*   Вариант 14       */
```

```
            for(i=0;i<m;i++)
                for(j=i;j<m;j++)
                  x[i][j]=(j-i+1)*(j-i+1);
            break;}
  }    // Вывод матрицы
 for(i=0;i<m;i++)
  { for(j=0;j<m;j++)
        printf("%4d",x[i][j]);
    printf("\n");
  }
 fflush(stdin);   getchar();
 return(0);
}
```

12 СТРОКИ И СИМВОЛЫ

12.1 Работа с символами

Для хранения отдельных символов используются переменные типа ***char***. Пример описания: ***char t, p;*** // Описаны 2 переменные символьного типа.

Возможными значениями символьной переменной могут быть различные символы. Такими символами могут быть почти все знаки из кодовой таблицы ПК за исключением специальных управляющих знаков, не имеющих графического изображения. Например, можно записать присваивания: **t='Q'; p='p';** Здесь **t** и **p** – переменные, которые могут принимать разные значения, а **'Q'** и **'p'** – конкретные значения, присваиваемые этим переменным. После этого в ячейке памяти, которая зарезервирована для переменной **t**, будет записан код символа '**Q**', а для переменной **p** – код символа '**p**'. ***Переменная p и значение 'p' – это не одно и то же!***

Код символа - это целое число. Занимает в памяти 1 байт. Коды всех символов сведены в кодовую таблицу, имеющую 16 строк и 16 столбцов, пронумерованных 16-ричными цифрами: 0, 1, …, 9, A, B, C, D, E, F. Т.о., кодовая таблица хранит коды 256 символов. Из кодовой таблицы можно определить код символа в 16-ричной системе счисления – это последовательность 16-ричных цифр, составленная из номера столбца и номера строки. Например, символ, стоящий на пересечении столбца 9 и строки F, имеет 16-ричный код $9F_{16}$. Можно перевести код в 10-тичную систему счисления: $9F_{16}=9*16+15=159_{10}$.

Существуют специальные программы вывода таблицы кодов символов на экран и вставки символов в текст программы. Например, ***ascii.com***.

Таблица ASCII кодов, применяемая в системе программирования Borland C++ 3.1 for DOS приведена на рис. 1.

Хотя в языке Си можно одновременно использовать переменные типов ***int*** и ***char***, все же отметим, что если объявлено ***int a; char b;*** и назначено ***a=5; b='5';***, то ***a*** и ***b*** - это не одно и то же: переменная ***a*** имеет числовое значение ***5***, а ***b*** – значение символа ***'5'***. Число ***5*** в двоичном коде равно ***101***, а двоичный код символа ***'5'*** есть ***110101*** ($110101_2= 35_{16}=53_{10}$).

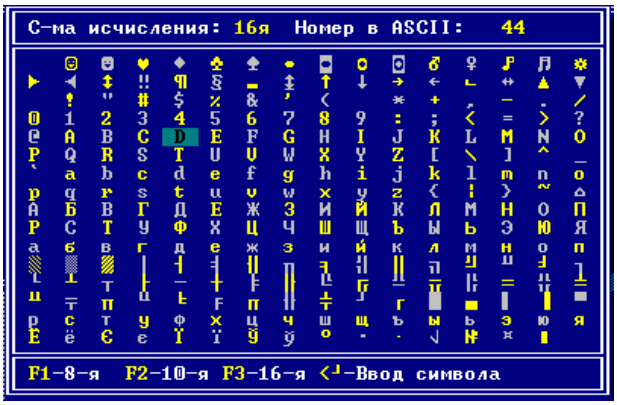


Рисунок 12.1 – Пример кодовой таблицы.

Ввод-вывод символов.

Рассмотрим наиболее распространенные функции ввода символа с клавиатуры и его вывода на экран дисплея (консольный ввод-вывод): *getchar*, *putchar* (для ввода-вывода символа); *gets*, *puts* (для ввода-вывода строки); *scanf*, *printf* (для форматированного ввода-вывода); и функцию *fflush* очистки буфера *stdin*.

Функция *getchar* предназначена для ввода символа, не имеет параметров, возвращает целое число - код введенного символа. Если символо не может быть прочитан, то возвращает значени EOF. Обращение к функции может иметь вид: *getchar()*. Функция может использоваться в выражении, например, ch=getchar();.

Выполняя эту функцию, программа приостанавливает свою работу и ждет от пользователя ввода символа и/или нажатия клавиши Enter, если буфер клавиатуры (*stdin*) пуст. Если буфер клавиатуры не пуст, то введенным считается символ, находящийся в буфере. Поэтому, перед использованием функции *getchar()* рекомендуется очищать буфер клавиатуры обращением к функции *fflush*: ***fflush(stdin);***.

Функция *putchar* предназначена для вывода символа на стандартное устройство вывода. Имеет один параметр типа *int* (код выводимого символа) или char. Если вывод успешен, возвращает значение кода символа, иначе возвращает значение EOF. После вывода символа на экран дисплея, курсор не переходит к началу новой строки.

Пример ввода и вывода символа.

```
/* Ввод-вывод символа */
#include <stdio.h>
main()
{
 int ch;                                  // Код вводимого символа
 puts("Введите любой символ");            // Вывод строки - подсказки
```

```
 ch=getchar();                          /* Ввод символа и присваивание его
кода переменной ch */
 puts("Вы ввели символ");               // Вывод строки - сообщения
 putchar(ch);              /* Вывод символа, определенного кодом ch */
 printf("\n");                          /* Перевод курсора к началу новой
строки */
 putchar('A');            /* Вывод символа 'A', заданного константой */
 printf("\n");
 putchar(65);             /* Вывод символа 'A', заданного кодом символа */
 fflush(stdin);             // Очистка буфера ввода
 getchar();   /* Ввод символа и/или нажатия Enter для организации задержки
смены окна Output на окно редактора текста*/
 return(0);
}
```

Как видно из примера, функция* getchar() *может использоваться в операторе присваивания. В этом случае код введенного символа присваивается переменной типа* int *(в примере - переменной ch).

Функции* putchar('A') *и* putchar(65) *выводят один и тот же символ* 'A', *символьной константой и кодом символа (65).

Если требуется ввести в одном операторе данные разных типов (в том числе и символ), то удобнее использовать *функцию* scanf ***(для ввода символа используется форматная спецификация %c).***

Если требуется вывести в одном операторе данные разных типов (в том числе и символ), то удобнее использовать *функцию* printf***. При этом, для вывода символа используется форматная спецификация %c; для вывода кода символа в десятичной, восьмеричной или шестнадцатеричной системе счисления используются спецификации %d, %o или %X, соответственно.***

Пример*. Ввести значение символьной переменной, вывести значение в виде символа и десятичного, восьмеричного и шестнадцатеричного кодов.*

Текст программы

```
/* Ввод символьной переменной и вывод в различных формах*/
#include<stdio.h>
#include<conio.h>
main()
{
  char b;                          //Объявление символьной переменной
```

```
 puts("Введите один символ");
 b=getchar();                    //Ввод значения переменной b
 printf("\tb=");
 putchar(b);                     //Вывод значения переменной b
 printf("\nДесятичный код b=%d\n",b); //Вывод десятичного кода b
 printf("\tВосьмеричный код b=%o\n",b); //Вывод восьмеричного кода
 printf("\t\tШестнадцатеричный код b=%X\n",b);
 printf("b='%c'\n",b);
 getch();
 return(0);
}
```

Ниже приведены результаты выполнения программы (серым цветом выделены символы, которые вводил с клавиатуры пользователь):

```
Введите один символ
*
    b=*
Десятичный код b=42
    Восьмеричный код b=52
        Шестнадцатеричный код b=2A
b='*'

Введите один символ
{
    b={
Десятичный код b=123
    Восьмеричный код b=173
        Шестнадцатеричный код b=7B
b='{'
Введите один символ
#
    b=#
Десятичный код b=35
    Восьмеричный код b=43
        Шестнадцатеричный код b=23
b='#'


Введите один символ
5
    b=5
Десятичный код b=53
    Восьмеричный код b=65
        Шестнадцатеричный код b=35
```

```
b='5'
```

Определение принадлежности символа какому-либо множеству

В библиотеке ***ctype*** определен целый ряд функций, проверяющих принадлежность символа какому-либо множеству: множеству букв (***isalpha***), букв или цифр (***isalnum***), разделителей (***isspace***), знаков пунктуации (***ispunct***), цифр (***isdigit***), видимых символов (***isgraph***), букв верхнего регистра (***isuper***), букв нижнего регистра (***islower***), печатаемых символов (***isprint***) и т.д.

Аргументом каждой из этих функций является символ. Функция возвращает значение ***true,*** если символ принадлежит конкретному множеству символов, или ***false*** в противном случае.

Пример. *Ввести символ с клавиатуры и проверить, является ли он знаком пунктуации.*

```
//Текст программы
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
void main()
{
  char a;
  puts("Введите один символ");
  a=getchar();
  if(ispunct(a))
    printf("Символ %c явл. знаком пунктуации\n",a);
  else
    printf("Символ %c не явл. знаком пунктуации\n",a);
  getch();
}
```

В приведенном тексте программы использовалась функция ispunct(a) для проверки принадлежности символа a множеству знаков пунктуации.

Ниже приведены результаты выполнения программы **(**серым цветом выделены символы, которые вводил с клавиатуры пользователь):

```
Введите один символ
.
Символ . явл. знаком пунктуации
Введите один символ
^
Символ ^ явл. знаком пунктуации
Введите один символ
e
```

```
Символ e не явл. знаком пунктуации
Введите один символ
#
Символ # явл. знаком пунктуации
Введите один символ
[
Символ [ явл. знаком пунктуации
Введите один символ
-
Символ - явл. знаком пунктуации
Введите один символ
n
Символ n не явл. знаком пунктуации
```

Следующий ***пример*** показывает, что символьная переменная может быть определена символом или кодом символа.

```
#include<stdio.h>
void main()
{
  char c,c1;
  c='A';          // символьная переменная определена символом
  c1=65;          // символьная переменная определена кодом символа
  printf("%c %d\n", c, c);          //1
  printf("%d %c\n", c1, c1);        //2
  if(c1=='A' && c== 65)
     puts("Обе переменные определяют символ  A");    //3
   else puts("****");
}
```

Результат выполнения программы показывают, что значением переменных символьного типа c, c1 на печати будет символ, если выводить значение по форматной спецификации %c, и код символа, если выводить значение по форматной спецификации %d:

```
A 65          //результат выполнения оператора //1
65 A          //  результат выполнения оператора //2
Обе переменные определяют символ  A//рез. выполнения оп. //3
```

Пример использования функции ***char*** для определения кода символа. Функция ***char*** имеет один аргумент – символ. Возвращает код своего аргумента.

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

```
#include<conio.h>
void main()
{
  int cod;        //код символа
  char S;         //символ
  puts("Введите один символ");
  S=getchar();
  cod=char(S);
  putchar(S);       //1
  printf("\t\n");
  printf("%c %d %d\n",S, S, cod);    //2
  getch();
}
```

Результаты выполнения программы:

```
Введите один символ
2
2                    // результат выполнения оператора //1

        2 50 50      //результат выполнения оператора //2

Введите один символ
%
%
              % 37 37
Введите один символ
D
D
              D 68 68
```

12.2 Строки

1.2. 1 Понятие и описание строки

В языке Си отсутствует специальный строковый тип. ***Строка в Си – это массив символов.*** В памяти ЭВМ строка представляется как массив элементов типа ***char***, в конце которого помещается символ **'\0'** - ***завершающий нуль-символ***.

Строковая константа – это последовательность символов, заключенная в кавычки ("). Например, " Задача имеет множество решений". В памяти представляется как массив элементов типа ***char*** с завершающим нуль-символом (**'\0').**

Общий вид описания строки:

char <имя строки>**[**<длина строки с учетом нуль-символа>**]**;

Например, char S[100];
Или
const int N=100; //Удобно задавать длину строки с помощью
char S[N]; // именованной константы

В такой строке S можно хранить 99 символов и завершающий нуль-символ.

При описании строки можно выполнять ее инициализацию. Например,

char S1[100]= "Исходный массив";

При этом нуль-символ автоматически формируется за последним символом символьной константы.

Если строка при описании инициализируется, то можно опускать длину строки. Например,

char S1[]= "Исходный массив"; //16 символов

Примечание. Имя строки, как и любого массива – это указатель-константа. Поэтому ошибкой будет попытка использовать имя строки в некоторых операциях адресной арифметики. Например, нельзя выполнить такую "пересылку" строки символов в массив:

char a[20]; //***Так***
a="Строковая константа"; //***нельзя***

Описание динамической строки

Для размещения строки в динамической памяти, необходимо описать указатель на Char, а затем выделить память с помощью ***new*** или malloc (new предпочтительнее).

Например,
char *S3=new char[m]; //m должно быть уже определено
Или
char *S3;
S3=new char[m];

Примечание. Динамические строки, как и другие массивы, нельзя инициализировать.

Например, оператор,
char *S4="На нуль делить нельзя";

Создает не строковую переменную, а указатель на строковую константу, изменить которую невозможно. S4 – адрес первого символа строковой константы.

1.2.2 Ввод-вывод строк

Рассмотрим ввод-вывод строк с помощью функций, унаследованных из бибдиотеки Си: ***gets*** и ***puts***, ***scanf*** и ***printf***.

Функции ***gets*** и ***puts*** используются, если работа производится только со строками. Функции ***scanf*** и ***printf*** удобнее использовать в том случае, если в одном операторе требуется ввести или вывести данные разных типов.

Функция *gets* предназначена для ввода строки. Имеет один параметр, задающий адрес области памяти, в которую помещаются символы вводимой строки. В языке Си имя переменной, имеющей строковый тип, является этим адресом. Обращение имеет вид: **gets***(name),* где *name* – переменная строкового типа - имя вводимой строки. Выполняя эту функцию, программа приостанавливает свою работу и ждет от пользователя ввода последовательности символов и/или нажатия клавиши Enter. Символ новой строки в строку не включается, вместо него в строку заносится нуль-символ. Функция возвращает указатель на строку s, а в случае возникновения ошибки или конца файла – NULL.

Функция *puts* предназначена для вывода строки на стандартное устройство вывода. Имеет один параметр, задающий адрес области памяти, из которой на экран выводятся символы. Как уже отмечалось, имя переменной, имеющей строковый тип, является этим адресом. Обращение имеет вид: *puts(name),* где name – переменная-строка - имя выводимой строки или строка символов, заключенная в кавычки. После вывода строки курсор перемещается к началу новой строки экрана, т.е. завершающий нуль-символ строки заменяется на символ новой строки. Возвращает неотричательное значение при успехе или EOF при ошибке.

Функция *printf* предназначена для вывода форматированной последовательности данны*х*

Функция scanf ***предназначена для ввода данных в заданном формате. Обращение имеет вид:***

scanf(nf,&a1,&a2,...)

Здесь *nf - форматная* строка; *&a1,&a2,...* - список ввода - указатели на значения вводимых переменных *a1, a2, ...*

Подробно эти функции рассматривались в разделе "Консольный ввод-вывод"[].

1.2.3 Операции над строками

1.2.3.1 Реализация операции присваивания

Поскольку строка является массивом, а не специальным типом данных, то для строк не определена операция присваивания. Присваивание можно выполнить посимвольно, т.е. "вручную", или с помощью стандартных функций ***strcpy*** и ***strncpy***.

Рассмотрим первый способ. Пример:

s[0]='В'; s[1]='в'; s[2]='о'; s[3]='д';

Рассмотрим второй способ.

Для использования функций ***strcpy*** и ***strncpy*** к программе следует подключить заголовочный файл <***string.h***> предложением #include <***string.h***>

Обращение к **функции *strcpy*** имеет вид:

strcpy(s1,s2);

Функция ***strcpy*** копирует все символы строки ***s2***, включая завершающий нуль-символ, в строку ***s1*** и возвращает ***s1***.

Пример 1.

```
/* Копирование строки s2 в строку s1 с помощью strcpy  */
#include<stdio.h>
#include<string.h>
#include<conio.h>
main()
{
 int n;                    // Длина строки
 char s2[20]="Скоро сессия!";  //Исходная строка
 n=strlen(s2)+1;           //Длина s1
 char *s1=new char[n];     //Строка - копия s1
 // char s1[20];
 strcpy(s1,s2);            //Копирование s2 в s1
 puts(s2);                 // Вывод s2
 puts(s1);                 //Вывод s1
 puts(strcpy(s1,s2));      //strcpy возвращает s1
 getch();
 return(0);
}
```

Результат выполнения программы – три строки:

```
Скоро сессия!
Скоро сессия!
Скоро сессия!
```

Обращение к **функции *strncpy*** имеет вид:

strncpy(s1,s2,n);

Функция ***strncpy*** копирует не более ***n*** символов из строки ***s2*** в строку ***s1*** и возвращает ***s1***. При этом если длина исходной строки ***s2*** превышает или равна ***n***, нуль-символ в конец строки ***s1*** не добавляется. В противном случае строка ***s1*** дополняется нуль-символами до n-го символа. Если строки перекрываются, поведение не определено.

Пример 2.

```
/* Копирование строки s2 в строку s1 с помощью strncpy  */
#include<stdio.h>
#include<string.h>
```

```
#include<conio.h>
main()
{
  int n;                        // Длина строки
  char s2[20]="Скоро сессия!";     //Исходная строка
  n=strlen(s2)+1;                 //Длина s1
  //char *s1=new char[n];          //Строка - копия s1
  char s1[20];
  strncpy(s1,s2,n);               //Копирование s2 в s1
  puts(s2);                      // Вывод s2
  puts(s1);                      //Вывод s1
  puts(strncpy(s1,s2,n));          //strncpy возврващает s1
  getch();
return(0);
}
```

Результат выполнения программы такой же, как и в примере 1.

Примечание 1. В рассмотренных примерах использована функция strlen(s), возвращающая фактическую длину строки s, не включая нуль-символ.

Примечание 2. Программист должен сам заботиться о том, чтобы в строке-приемнике (s1) хватило места для строки-источника (s2), и о том, чтобы строка всегда имела нуль-символ.

Выход за пределы строки и отсутствие нуль-символа является распространенными причинами ошибок в программах обработки строк.

1.2.3.2 Преобразование строки в число

Преобразование строк в числа можно выполнить с помощью функций ***atoi, atol, atof.*** Обратные преобразования - с помощью функций ***itoa,*** .

Функция *atoi(s)* преобразует строку, содержащую символьное представление целого числа в соответствующее целое число. Признаком конца числа служит первый символ строки, который не может быть интерпретирован как принадлежащий целому числу. Если преобразование не удалось, возвращает 0.

Функция *atol(s)* преобразует строку, содержащую символьное представление длинного целого числа в соответствующее целое число.

Функция *atof(s)* преобразует строку, содержащую символьное представление вещественного числа в соответствующее вещественное число двойной точности.

Для использования функций ***atoi(s) atol(s)*** и ***atof(s)*** к программе следует подключить заголовочный файл <***stdlib.h***> предложением #include < ***stdlib.h***>

Пример. Данные об участнике соревнований (номер участника, рост и вес) содержатся в строке символов. Вывести номер, рост и вес участника соревнований, зная, что номер записан в самом начале строки, рост, начиная с позиции 11, вес - с позиции 26.

```
/* Преобразование строки в число */
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<conio.h>
main()
{
 char s[]="10) Рост – 162 см., вес – 63.4кг";     //Строка
 int n;                                  // Целое число
 long h;                                 // Длинное целое
 double w;                               //Вещественное число
 n=atoi(s);                              //Преобразование в целое
 h=atol(&s[11]);                         //Преобразование в длинное целое
 w=atof(&s[26]);            //Преобразование в вещественное число
 printf("Участник # %d. Его рост %ld см и вес %lf кг\n",n,h,w);
 puts("Исходная строка");
 puts(s);
 puts("Первая подстрока");
 puts(&s[11]);
 puts("Вторая подстрока");
 puts(&s[26]);
 getch();
 return(0);
}
```

Результат выполнения программы:

```
Участник # 10.  Его рост 162 см и вес 63.400000 кг
Исходная строка
10) Рост – 162 см., вес – 63.4кг
Первая подстрока
162 см., вес – 63.4кг
Вторая подстрока
63.4кг
```

Заметим, что строка определяется адресом ее нулевого символа. Для всей строки s таким адресом является имя строки ***s***. Число **10** в символьном представлении находится в строке ***s*** в самом ее начале. Поэтому аргументом

функции ***atoi*** является строка ***s***. Число **162** в символьном представлении находится в строке ***s***, начиная с позиции **11**. Поэтому аргументом функции ***atol*** является подстрока строки ***s***, определяемая адресом **&s[11]**. Число **63.4** в символьном представлении находится в строке ***s***, начиная с позиции **26**. Поэтому аргументом функции ***atof*** является подстрока строки ***s***, определяемая адресом **&s[26]**.

1.2.3.3 Поиск подстроки в строке

Функция *strstr(s1,s2)* выполняет поиск подстроки ***s2*** в строке ***s1*** (первого вхождения подстроки ***s2*** в строку ***s1***). Обе строки должны завершаться нуль-символами. В случае успешного поиска функция возвращает указатель на найденную подстроку. В случае неудачи – NULL.

Пример. Определить, содержится ли строка ***s2*** в строке ***s1*** в качестве подстроки.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
main()
{
  const int n=81;
  char s1[n],s2[n];
  char *p;
  clrscr();
  puts("Введите строку s1");
  gets(s1);
  puts("Введите строку s2");
  gets(s2);
  p=strstr(s1,s2);
  if(p)
   {
    printf("Подстрока '%s'\n начинается в строке: %s\n",s2,s1);
    printf("символом '%c'\nэтим символом начинается подстрока: %s\n",*p,p);
   }
     else puts("NO");
  getch();
 return(0);
}
```

Заметим, что ****p*** – символ в строке ***s1***, с которого начинается ***s2***; ***p*** – подстрока, начиная с символа ****p*** до конца строки ***s1***.

Пример выполнения программы.

Введите строку s1
Скоро ли сессия? Так хочется сдать экзамен по ОАиП!
Введите строку s2
экзамен
Подстрока ' экзамен '
начинается в строке: Скоро ли сессия? Так хочется сдать экзамен по ОАиП!
символом 'э'
этим символом начинается подстрока: экзамен по ОАиП!

1.2.3.4 Сцепление двух строк (конкатенация)

Функция *strcat(s1,s2)* присоединяет строку ***s2*** в конец строки ***s1*** и возвращает указатель на строку, совпадающий с первым аргументом. При этом сначала из строки ***s1*** удаляется завершающий нуль-символ. В конце новой строки ***S1*** помещается *'\0'*.

Программист должен сам обеспечить достаточную длину строки s1.

Примеры.

1)

```
char s1[40]="Скоро Новый Год!";
char s2[]="Скоро первая сессия!";
printf(" %s\n", strcat(s1,s2));
```

Будет получена строка, полученная присоединением строки ***s2*** в конец строки ***s1***:

Скоро Новый Год! Скоро первая сессия!

2) Вторым параметром функции ***strcat*** является текстовая константа:

```
char s3[50]="Успешной сдачи";
printf("%s\n\t%s\n", strcat (s3," 1-ой сессии!!!"), s3);
```

Будет получено две строки:

Успешной сдачи 1-ой сессии!!!
Успешной сдачи 1-ой сессии!!!

Видно, что строка, возвращенная функцией ***strcat,*** совпадает со строкой ***s3.***

3)

```
char s4[35]="Успехов и здоровья";
char s5[]=" в Новом Году!";
char *p;
p=strcat(s4, s5);
pintf("%s\n%s\n",s4,p);
```

Будет получено две строки:

Успехов и здоровья в Новом Году!
Успехов и здоровья в Новом Году!

В рассмотренном примере описан указатель на char: ***char *p;***. Указатель ***p*** определяет строку, возвращаемую функцией: ***p=strcat(s4, s5);***. Строка ***s4*** совпадает со строкой, определяемой указателем ***p***.

Функция *strncat(s1,s2,n)* присоединяет ***n*** символов строки, на начало которой указывает ***s2,*** в конец строки, на начало которой указывает ***s1*** и возвращает указатель на строку, совпадающий с первым аргументом. Сформированная строка ***s1*** ограничивается ***'\0'***.

Если n больше длины строки ***s2***, то выполняется простая конкатенация.

Если длина строки ***s2*** меньше ***n***, то все символы *s2* присоединяются к строке ***s1***.

1.2.3.5. Определение позиции первого вхождения символа из заданного набора символов

Функция ***strcspn(s1,s2***) сопоставляет каждый символ строки ***s1*** со всеми символами строки, на начало которой указывает ***s2,*** и возвращает позицию первого вхождения символа строки ***s2*** в строке ***s1***. Символ '/0' в сравнении не участвует.

Если строка ***s1*** начинается с символа, встречающегося в строке ***s2***, то функция возвращает значение нуль. Если строка ***s1*** не содержит ни одного символа строки ***s2***, то возвращаемое функцией значение совпадает с длиной строки ***s1***.

Примеры.

```
 printf("%d %d %d\n",
strcspn("asdf",  "hjyars"),    //s1 начанается с 'a'
strcspn("asdf", "ghtdu"),    //в s1 'd' в позиции № 2
strcspn("asrt", "hj"));       // в s1 не символов s2
```

Получим:

0 2 4

Здесь 0 и 2 – позиции в строке s1, 4 – длина строки s1.

1.2.3.6 Сравнение двух строк

Сравнение строк производится посимвольно слева направо. Большей считается та строка, в которой первый несовпадающий символ имеет больший код в кодовой таблице.

Функция *strcmp(s1,s2)* сравнивает строки ***s1*** и ***s2*** . Возвращает отрицательное значение, если ***s1<s2***, нулевое, если ***s1=s2*** или положительное значение, если ***s1>s2***.

Функция ***strncmp(s1,s2, n)*** сравнивает строку ***s1*** и первые ***n*** символов строки ***s2*** . Возвращает отрицательное значение, если ***s1< чем первые n символов s2***, нулевое, если ***s1=первым n символам s2*** или положительное значение, если ***s1> чем первые n символов s2***.

Пример. Ввести две строки и вывести их в лексикографическом порядке.

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
main()
{
  char s1[20];
  char s2[20];
  puts("Введите 1-ую строку ");
  gets(s1);
  puts("Введите 2-ую строку ");
  gets(s2);
  if(strcmp(s1,s2)<0)
      {
       puts(s1);
       puts(s2);
      }
   else
    if(strcmp(s1,s2)>0)
      {
        puts(s2);
        puts(s1);
      }
     else puts("Строки совпадают");
  getch();
  return(0);
}
```

Примеры результатов выполнения программы:

```
Введите 1-ую строку
abcdefg
Введите 2-ую строку
abcdefg
Строки совпадают
```

```
Введите 1-ую строку
bcdert
Введите 2-ую строку
bccertyu
bccertyu
bcdert

Введите 1-ую строку
rtyu
Введите 2-ую строку
rtdh
rtdh
rtyu
```

1.3. Примеры решения задач по обработке строк

Пример 1. Сколько раз подстрока s1 содержится в строке s?

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
main()
{
  char s[20];
  char s1[20];
  puts("Введите строку ");
  gets(s);
  puts("Введите подстроку ");
  gets(s1);
  int k=0, i=0;
  char *p=s;
  int n=strlen(s1);
  printf("n=%d %c\n",n,*p);
  while(strstr(p,s1) && i<n)
   {
     k=k+1;
     p=p+n+1; printf("%c\n",*p);
   }
  puts("В строке:");
  puts(s);
  printf("%d раз встречается %s\n",k,s1);
  getch();
  return(0);
```

```
}
```
Пример
мама мыла, мама стирала, мама варила, будила студента – строка s
мама – подстрока s1

Эта же задача с использованием srtncmp

```
/*Сколько раз подстрока s1 встречается в строке s? */
#include<stdio.h>
#include<string.h>
#include<conio.h>
main()
{
  char s[20];
  char s1[20];
  puts("Введите строку ");  gets(s);
  puts("Введите подстроку ");  gets(s1);
  int k=0,i=0;
  int m=strlen(s);
  int n=strlen(s1);
  printf("m=%d n=%d\n",m,n);
  while(i<m)
   {
     if(strncmp(s1,&s[i],n)==0)
                {
                  k=k+1; i=i+n;
                 }
     else i=i+1;
    }
  puts("В строке:");
  puts(s);
  printf("%d раз(а) встречается %s\n", k, s1);
  getch();
 return(0);

}
/*Сколько раз слово s1 встречается в строке s? */
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<ctype.h>
void main()
{
  char s[20];
```

```
  char s1[20];
  puts("Введите строку ");  gets(s);
  puts("Введите слово ");  gets(s1);
  int k=0,i=0;
  int m=strlen(s);
  int n=strlen(s1);
  printf("m=%d n=%d\n",m,n);
  while(i<m)

   {
     if(strncmp(s1,&s[i],n)==0)
          { if(i==0)
             {
               if(isspace(s[i+n]) || ispunct(s[i+n]) || (s[i+n]=='\0'))
                  { k++; i=i+n; }
                  else i++;
             }
            else
             if((isspace(s[i-1])||ispunct(s[i-1])) &&
               (isspace(s[i+n])|| ispunct(s[i+n]))||(s[i+n]=='\0'))
               { k++; i=i+n; }
              else i++;
            }
     else i++;
   }
  puts("В строке:");
  puts(s);
  printf("%d раз(а) встречается %s\n",k,s1);
  getch();

}

/*Сколько раз слово s1 встречается в строке s? */
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<ctype.h>
void main()
{
  char s[20];
  char s1[20];
  puts("Введите строку ");  gets(s);
  puts("Введите слово ");  gets(s1);
  int k=0,i=0;
```

```
  int m=strlen(s);
  int n=strlen(s1);
  printf("m=%d n=%d\n",m,n);
  //Добавление пробела в начало слова
  for(i=m;i>=0;i--)
    s[i+1]=s[i];
  s[0]=' ';
  puts(s);
  //Поиск в строке
  while(i<m+1)
     if(strncmp(s1,&s[i],n)==0)
           {
               if((isspace(s[i+n]) || ispunct(s[i+n]) || (s[i+n]=='\0')) &&
               (isspace(s[i-1])||ispunct(s[i-1])))
                  { k++; i=i+n; }
                  else i++;
            }
     else i++;
  //Удаление из строки символа s[0]
  for(i=1;i<=m+1;i++)
     s[i-1]=s[i];
  puts("В строке:");
  puts(s);
  printf("%d раз(а) встречается %s\n",k,s1);
  getch();

}
/*Сколько букв в третьем слове строки s? */
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<ctype.h>
void main()
{
  char s[80];
  int k,k3,i,n;
  puts("Введите строку ");  gets(s);
  n=strlen(s);
  k=0;//Кол. слов
  //Поиск в строке
  i=0; //Номер символа
  while(i<n && k!=3)
    {
      if(isalpha(s[i]))   //Слово началось
```

```
          {
            k++;
            k3=0; //Кол. букв в слове
            while(isalpha(s[i]))
              {
                k3++; i++;
              }
          }
      i++;
    }
  if(k==3)
    {
      printf("В третьем слове строки:");
      puts(s);
      printf(" %d символ(ов,а) \n",k3);
    }                                                    2
   else
     {
       puts("В строке:");
       puts(s);
       puts("меньше трех слов");
     }
  getch();
}

/*Вставить пробел в начало строки (с использ. указателей) */
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<ctype.h>
void main()
{
  char *s,*p;
  puts("Введите строку ");  gets(s);
  int m=strlen(s);
  p=s-1;
  *p=' ';
  puts(s);
  puts(p);
  s=p;

  puts(s);
  getch();
```

```
}

/*  Вставить пробел перед'.'  */
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<ctype.h>
void main()
{
  char *s,*p,*c;
  puts("Введите строку ");  gets(s);
  puts(s);
  int m=strlen(s);
  p=strchr(s,'.');//Позиция .
  puts(p);
  strncpy(c,s,p-s); //Подстрока до .
  puts(c);
  strcat(c," ");
  strcat(c,p);
  s=c;
  puts(s);
  puts(p);
  puts(c);
  getch();

}
```

13 ЗАПИСИ (СТРУКТУРЫ)

13.1 Определение записи (структуры)

Запись (record) – структурированный (составной) тип данных, состоящий из фиксированного числа компонентов разного типа, называемых ***полями*** (Helds) записи.
Запись связывает элементы разных типов в один объект. Этим запись отличается от массива (в массиве все элементы имеют одинаковый тип). Чтобы можно было ссылаться на тот или иной компонент записи, поля именуются, т.е. каждому полю записи присваивается имя. Например, такую информацию о студенте, как фамилия, имя, отчество, адрес, возраст, экзаменационные оценки, средняя успеваемость можно объединить в одну запись, имеющую пять полей. Имена и типы полей приведены в таблице 2.1.
Таблица 13.1

	Поля записи				
Запись	Fio	Adress	Age	Oc	Sr
Типы полей	Строка	Строка	Целое	Массив целочис.	Вещ.

Поля записи могут иметь любой допустимый в Си тип данных, в том числе поля сами могут быть структурами (записями). Но поле не может иметь тип этой же структуры, но может быть указателем на нее.
Описание структурной переменной состоит из двух этапов:
1. Описание шаблона структуры
2. Описание структурной переменной

Синтаксис описания шаблона структуры:

```
struct  <имя_ шаблона>
        {
          <тип1>   <имя_поля1>;
          <тип2>   <имя_поля2>;
          …
          <типN>   <имя_поляN>;
        };
```

где <тип1>, <тип2>, … ,<типN> - любые основные типы (int, char, float, и т. д.), массив, указатель, структура, объединение.

Пример:

```
struct  Student
        {
         char *fio;              // Фамилия       - указатель на char
```

```
  char   Adress[40];   // Адрес          - строка
  int    Age;          // Возраст        - целое
  int    oc[4];        // Оценки         - целочис. массив
  float  sr;           // Средний балл - вещественное
} ;
```

Шаблон структуры определяет новый тип, имя которого можно использовать наряду со стандартными типами

Синтаксис описания структурной переменной (записи):

struct <имя_шаблона> <имя_переменной>;

или

struct <имя_шаблона> <список имен переменных>;

Ключевое слово *struct* можно опускать в таком описании.

Примеры:

struct Student S;

Student S, S1, S2;

Компилятор выделяет под структурную переменную число байтов, не всегда равное сумме длин отдельных полей из-за влияния дополнительного фактора внутреннего представления структурных переменных, называемого выравниванием. Точное число выделенных байтов определяется с помощью стандартной функции ***sizeof***:

sizeof(*struct*<имя_шаблона>);

Например, *sizeof* (*struct* Student); возвращает 56 байт.

Можно совмещать описание шаблона структуры и структурной переменной в одном предложении. Разрешается выполнять инициализацию полей структурной переменной при ее описании. Для инициализации структурной переменной значения ее полей перечисляются в фигурных скобках в порядке, соответствующему описанию полей.

Пример совмещения описания шаблона, описания структурных переменных и инициализации полей переменный в одном предложении:

```
struct Student                  // Описание шаблона структуры
  { char *fio;                  // Фамилия       - указатель на char
    char   Adress[40];          // Адрес          - строка
    int    Age;                 // Возраст        - целое
    int    oc[4];               // Оценки         - целочис. массив
    float  sr;                  // Средний балл - вещественное
  }
 S, S1,                         // Описание переменных S,S1,S2
S2={ "Иванов",                  // и инициализация полей S2
     "г.  Жлобин, ул Цветочная, д.1, кв.43 ",
```

```
    1991,
    {9,6,8,5},        //Инициализация поля - массива  (int oc[4]; )
    0
  };
```

Для переменных одного итого же структурного типа определена ***операция присваивания***. Например, можно записать S1=S2; После выполнения такого оператора присваивания значения полей структуры S1 будут совпадать со значениями соответствующих полей структуры S2.

Вложенные структуры.

Структуру, имеющую поле, являющееся структурой, называют вложенной структурой. Шаблон вкладываемой структуры должен быть известен до описания вложенной структуры. Например,

```
struct fios                    // Описание вкладываемой структуры
      { char *f,*im,*ot; };

typedef struct Student      // Описание собственного типа MY_Student
        { struct fios fio;        // поле  - структура
          char Adress[40];
          int  Age;
          int  oc[4];
          float sr;
        } MY_Student;
/* Описание структурной переменной S с ее инициализаций */
 MY_Student S={  {"Иванов", "Петр", "Сидорович"},
                  "Река Сож",
                   1988,
                   {5,4,5,4},
                    0
                };
```

Доступ к отдельным полям структурной переменной.

Для доступа к полям структуры используется операция выбора . (точка) , с помощью которой конструируется ***составное имя***:

< имя_структурной_переменной >.<имя поля> - *составное имя*, где . - операция *выбора* или *ссылки на поле* (обычная точка).

Например, S.Age, S.oc[0], S.oc[i]

Доступ к компонентам поля, являющимсяся структурой, осуществляется через две операции выбора. Например,

S.fio.f, S.fio.im, S.fio.ot

Структурные переменные и указатели

Синтаксис описания указателя:
<тип>*<имя_переменной>;
Таким образом, для описания указателя на структуру должен быть создан новый тип.
-> - операция доступа к полям структурной переменной через указатель (минус больше).
Пример:

```
typedef struct Student
        { char   fio[30];
          char   Adress[40];
          int    Age;
          int    oc[4];
          float  sr;
        } MY_Student;  //имя нового типа
MY_Student *S3          // Указатель на структуру
```

Примеры обращения к полям:

S3->Age, S3->oc[0], S3->oc[i], S3->fio, S3-> sr

Пример программ работы со структурами

```
#include <stdio.h>
main()
{
      /*              Описание шаблона структуры                    */
 struct Student
      { char *fio;          // Фамилия      - указатель на char
        char  Adress[40];   // Адрес        - строка
        int   Age;          // Возраст      - целое
        int   oc[4];        // Оценки       - целочис. массив
          float  sr;        // Средний балл - вещественное
      } ;
 struct Student S;          // Описание структурной переменной S
      int i;
 float sr;
 S.fio="Петушков";      //Присваивание полю значения
 printf("Введите адрес студента %sa ",S.fio);
 gets(S.Adress);          //Ввод значения поля
 S.Age=1987;              //Присваивание полю значения
 S.oc[0]=3;  S.oc[1]=5; S.oc[2]=4; S.oc[3]= S.oc[2];
 sr=0;                    //Вычисление среднего балла
 for(i=0;i<=3;i++)
```

```
      sr=sr+S.oc[i];
  sr=sr/4;
  S.sr=sr;                    //Присваивание полю вычисленного значения
/*     Вывод полей записи   S */
  printf(" Средний балл студента %sa", S.fio);
  printf(" %d года рождения,\n проживающего по адресу: %s,",
                          S.Age,S.Adress);
  printf(" равен %5.2f\n", S.sr);
  fflush(stdin);   getchar();
  return(0);
}
```

Из примера видно, что имя поля структуры и имя переменной могут совпадать, поскольку у них разные области видимости. Более того, поля разных структур могут совпадать. Но делать так не рекомендуется, поскольку себя запутать легче, чем компилятор.

Вид экрана после выполнения приведенной программы:

Введите адрес студента Петушкова ул. Солнечная, д.1, кв. 4
Средний балл студента Петушкова 1987 года рождения,
проживающего по адресу: ул. Солнечная, д.1, кв. 4, равен 4.00

Здесь серым цветом выделен текст, который пользователь набрал на клавиатуре.

Массивы структур (записей)

Описание массива структур не отличается от описания массива обычных переменных.
Пример:

```
Struct Man
        {
          char   fio[31];          // ФИО
           int   year;             // Год рождения
          float  pay;              // Оклад
        };
Man d[3], *p=d;     // массив структур из трех элементов и
                            // указатель, инициализированный адресом
                            //первого элемента массива
```

Примеры обращения к полям:
d[i].year, (*(d+i)).year, (p+i)->year

```
/*  Пример 6. Поиск в массиве структур, вводимых с клавиатуры  */
/*   Фамилии сотрудников, имеющих оклад выше среднего  */

#include <stdio.h>
#include <string.h>
#include <iostream.h>

void main()
{
  const int lfio=20, lpay=5, lo=7,     //длины полей фио, г.рожд., оклада
              ldb=10;
  struct Man
     {  char   fio[lfio+1];     // фио
        int    year;           // год рожд.
        float  pay;            // оклад
     };
  Man db[ldb];                  // массив структур
  int i, n;
  float s;  // Средний оклад
  puts("Число записей?(1<n<=10)");   cin>>n;
 /*Ввод массива записей */
  for(i=0; i<n; i++)
    {
      puts("Фамилия? ");          cin>>db[i].fio;
      puts("Год рождения? ");   cin>>db[i].year;
      puts("Оклад? ");              cin>>db[i].pay;
    }
/* Вывод массива записей в форме таблицы*/
  puts(" Список сотрудников");
  puts(" ┌──────────────────┬──────┬─────────┐ ");
  puts(" │  ФИО             │  г.р.│ Оклад   │ ");
  puts(" ├──────────────────┼──────┼─────────┤ ");
  for(i=0; i<n; i++)
    printf("|%-20s|% 5d | % 7.2f |\n", db[i].fio, db[i].year, db[i].pay);
  puts(" └──────────────────┴──────┴─────────┘ ");
  puts("Фамилии сотрудников, имеющих оклад выше среднего");
//Вычисление среднего оклада
  s=0;
  for(i=0; i<n; i++)
      s+=db[i].pay; // s=s+db[i].pay;
  s/=n;   //s=s/n;

 // Определение и вывод фамилий
```

```
  for(i=0; i<n; i++)
    if(db[i].pay>s)  printf("%-s\n", db[i].fio);
  fflush(stdin); getchar();
}
```

Пример. Описать структуру с именем Regrz, содержащую следующие поля:

1. ФИО – студента
2. Номер группы
3. Название предмета
4. Дата поступления работы
5. ФИО - преподавателя, проверяющего работу.
6. Оценка о зачете (зачет, незачет)

Написать программу (Регистрация контрольных работ заочников), выполняющую следующие действия:

- ввод с клавиатуры данных в массив;
- добавлять записи в массив;
- выводить на экран все записи в виде таблицы;
- удалять из массива запись с заданным номером;
- осуществлять поиск в соответствии с запросами:
  - удалять все записи по конкретной группе;
  - изменять фамилию заданного студента;
  - вывести все данные о зачтенных контрольных работах конкретного студента;
- результаты поиска выводить на экран в виде таблицы.

Решение.

Выполнил студент
Группы ИТ-11
Иваненко Н.И.
Проверил преподаватель
Кравченко О.А.

Для упрощения основного алгоритма, можно его разделить на несколько подзадач:

- добавление записей (input);
- вывод записей в виде таблицы (output);
- удаление записи по номеру (delnum);
- удаление записей по конкретной группе (delgr);
- замена фамилии студента (change);
- вывод всех зачтенных работ конкретного лица (outstud).

Реализуем первую подзадачу. Для этого разработаем алгоритм ввода/добавления записей. Его графическая схема изображена на рисунке 13.1.

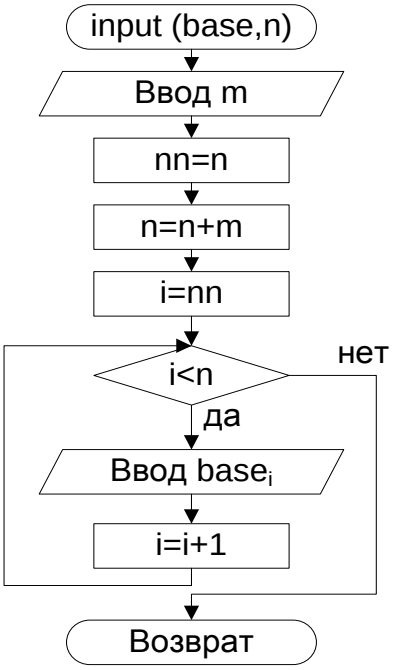


Рисунок 13.1 – Алгоритм ввода/добавления записей в массив записей.

В таблице 13.2. указаны типы используемых в алгоритме переменных.

Таблица 13.2

Имя перем. в условии	Имя перем. на С	Тип	Комментарий
	base	zaochn	Формальный параметр – указатель на структуру
	n	int	Формальный параметр - количество записей в массиве
	i	int	Параметр цикла
	m	int	Количество добавляемых записей
	nn	int	Начальное
	k	int	Параметр для ввода по полям

Реализуем вторую подзадачу. Разработаем алгоритм вывода записей. Его графическая схема изображена на рисунке 13.2.

В таблице 13.3. указаны типы используемых в алгоритме переменных.

Таблица 13.3.

Имя перем. в	Имя перем. на С	Тип	Комментарий

условии			
	base	zaochn	Формальный параметр – указатель на структуру
	n	int	Количество записей во всей структуре
	i	int	Параметр цикла
	i_nach	int	Номер начальной записи
	i_kon	int	Номер конечной записи
	vvod	int	Параметр для организации прокрутки
	koef	int	Параметр для вычисления количества записей на страницу

Реализуем третью подзадачу. Разработаем алгоритм удаления записи по номеру. Его графическая схема изображена на рисунке 13.2.
В таблице 13.4 указаны типы используемых в алгоритме переменных.
Таблица 13.4.

Имя перем. в условии	Имя перем. на С	Тип	Комментарий
	base	zaochn	Формальный параметр – указатель на структуру
	n	int	Количество записей во всей структуре
	i	int	Параметр цикла
	j	int	Параметр цикла
	k	int	Номер удаляемой записи

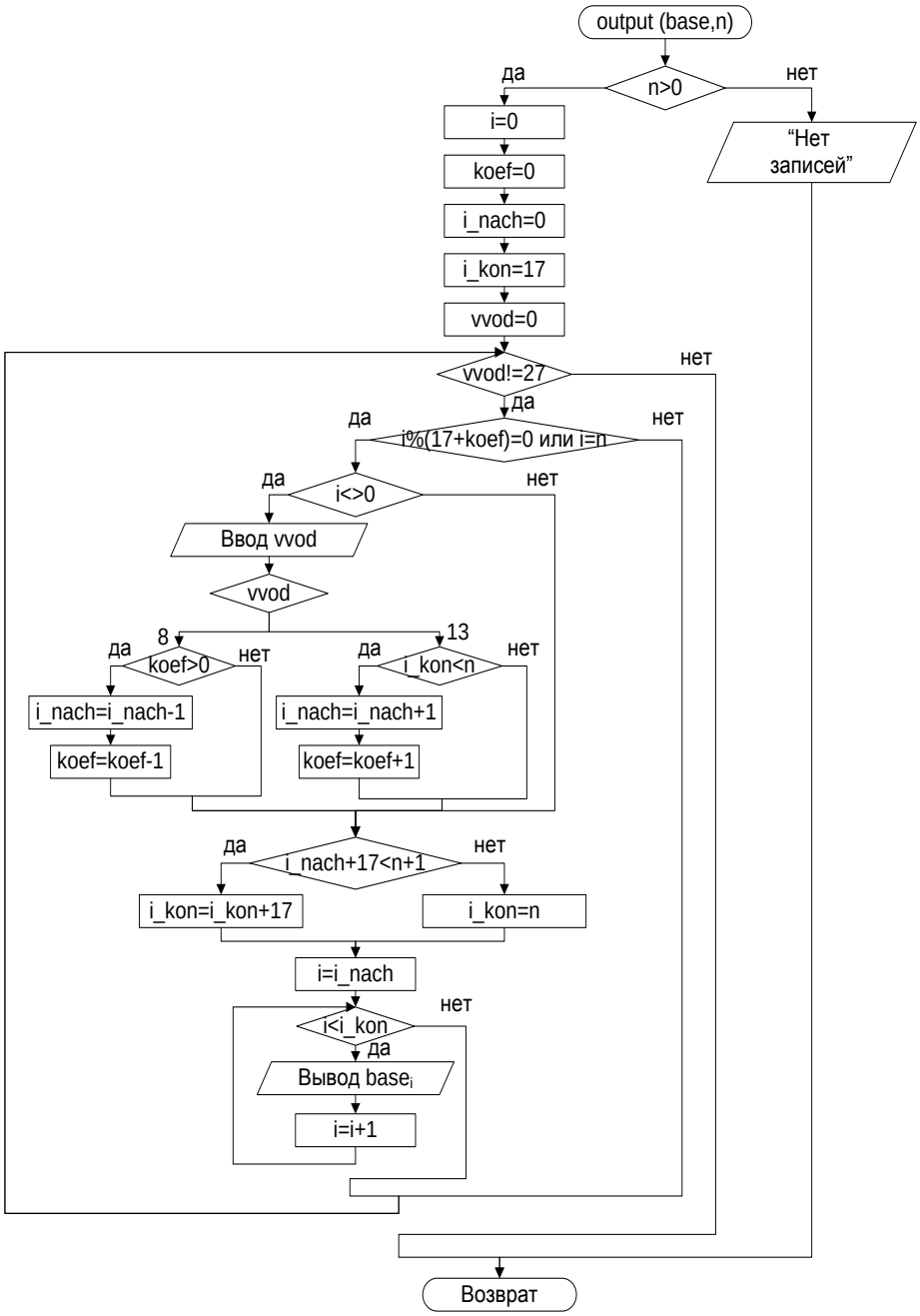


Рисунок 13.2 – Алгоритм вывода записей.

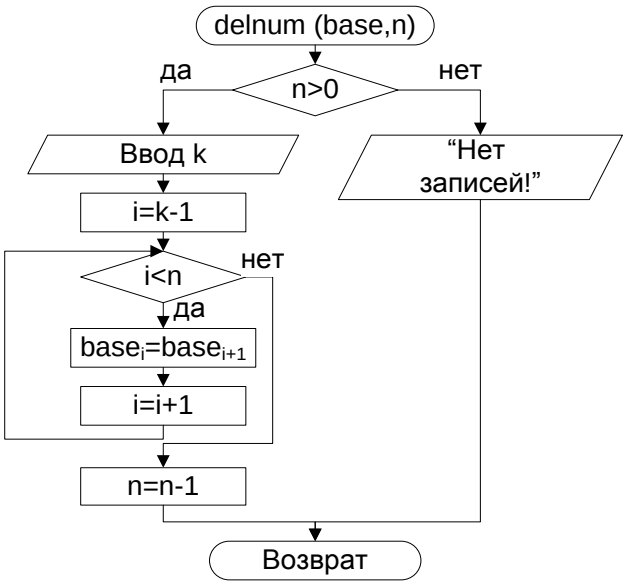


Рисунок 13.3. – Алгоритм удаления записи по номеру.

Реализуем четвертую подзадачу. Разработаем алгоритм удаления записи по группе. Его графическая схема изображена на рисунке 13.4.
В таблице 13.5 указаны типы используемых в алгоритме переменных.

Таблица 13.5

Имя перем. в условии	Имя перем. на С	Тип	Комментарий
	base	zaochn	Формальный параметр – указатель на структуру
	n	int	Количество записей во всей структуре
	i	int	Параметр цикла
	j	int	Параметр цикла
	k	int	Номер удаляемой записи
	grp	char	Номер группы

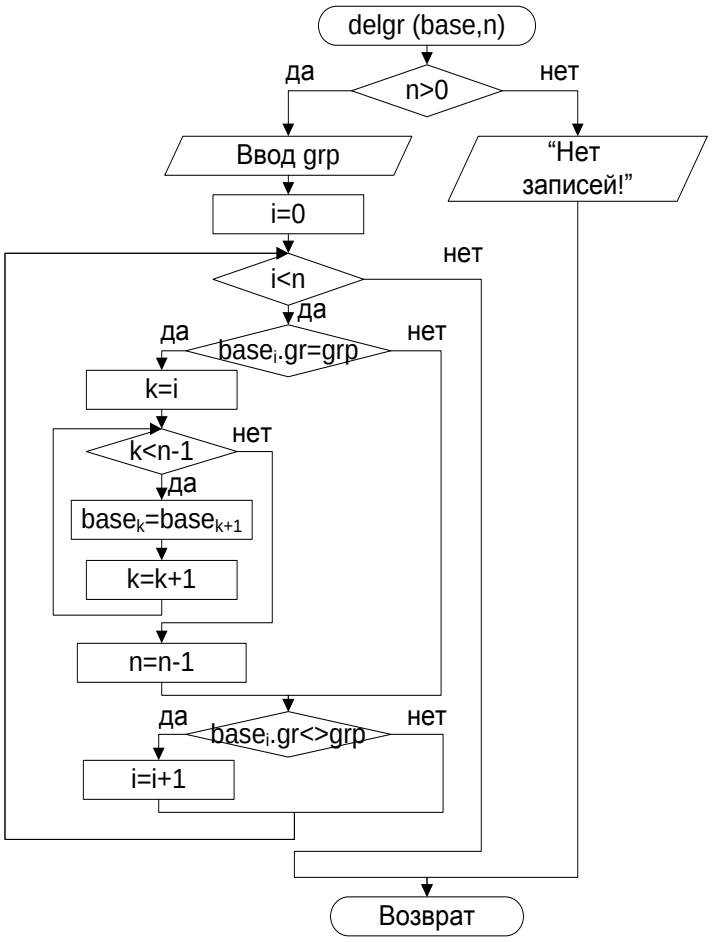


Рисунок 13.4 – Алгоритм удаления записи по группе

Реализуем пятую подзадачу. Разработаем алгоритм смены фамилии конкретного студента. Его графическая схема изображена на рисунке 13.5. В таблице 13.6 указаны типы используемых в алгоритме переменных.

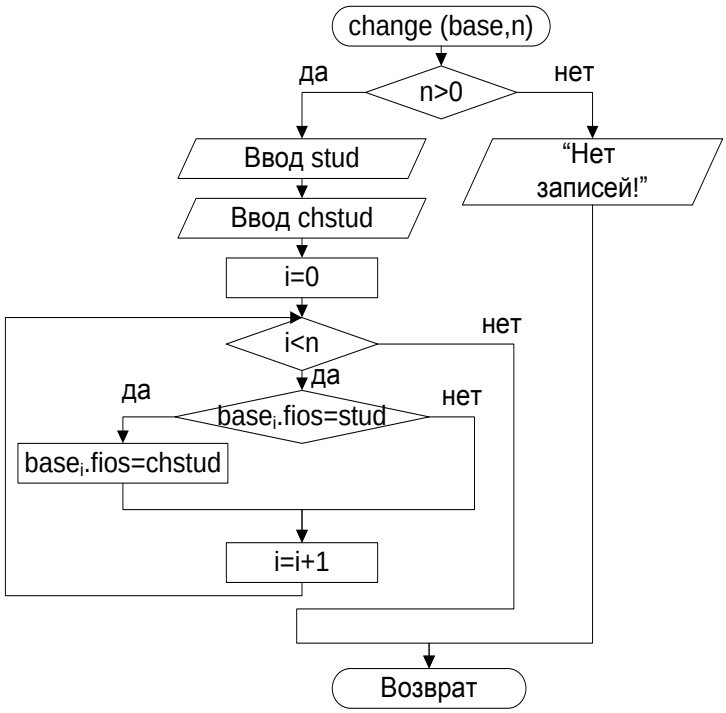


Рисунок 13.5 – Алгоритм смены фамилии конкретного студента.

Таблица 13.6

Имя перем. в условии	Имя перем. на С	Тип	Комментарий
	base	zaochn	Формальный параметр – указатель на структуру
	n	int	Количество записей во всей структуре
	i	int	Параметр цикла
	j	int	Параметр цикла
	stud	char	Начальная фамилия студента
	chstud	char	Конечная фамилия студента

Реализуем шестую подзадачу. Разработаем алгоритм вывода зачтенных работ конкретного студента. Его графическая схема изображена на рисунке 13.6. В таблице 13.7 указаны типы используемых в алгоритме переменных.

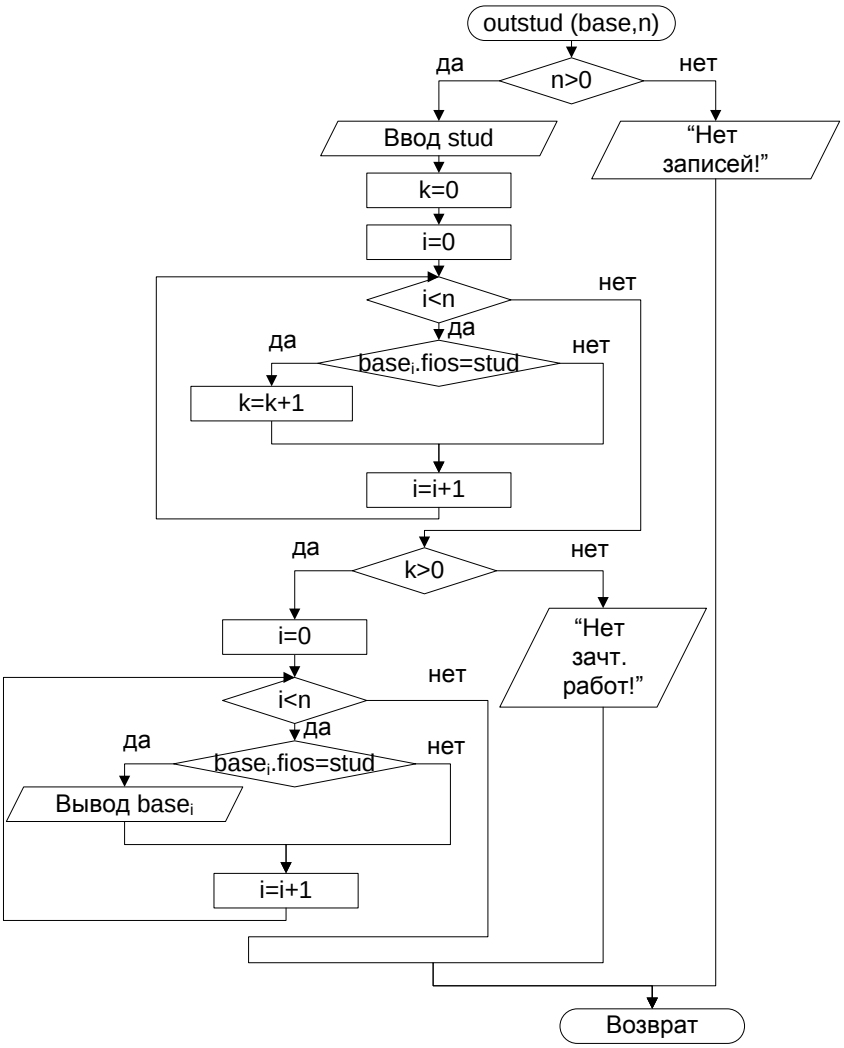


Рисунок 13.6. – Алгоритм вывода зачтенных работ конкретного студента.

Таблица 13.7

Имя перем. в условии	Имя перем. на С	Тип	Комментарий
	base	zaochn	Формальный параметр – указатель на структуру
	n	int	Количество записей во всей структуре
	i	int	Параметр цикла

	k	int	Параметр для подсчета зачтенных работ
	stud	char	Фамилия студента

Основной алгоритм выглядит следующим образом.

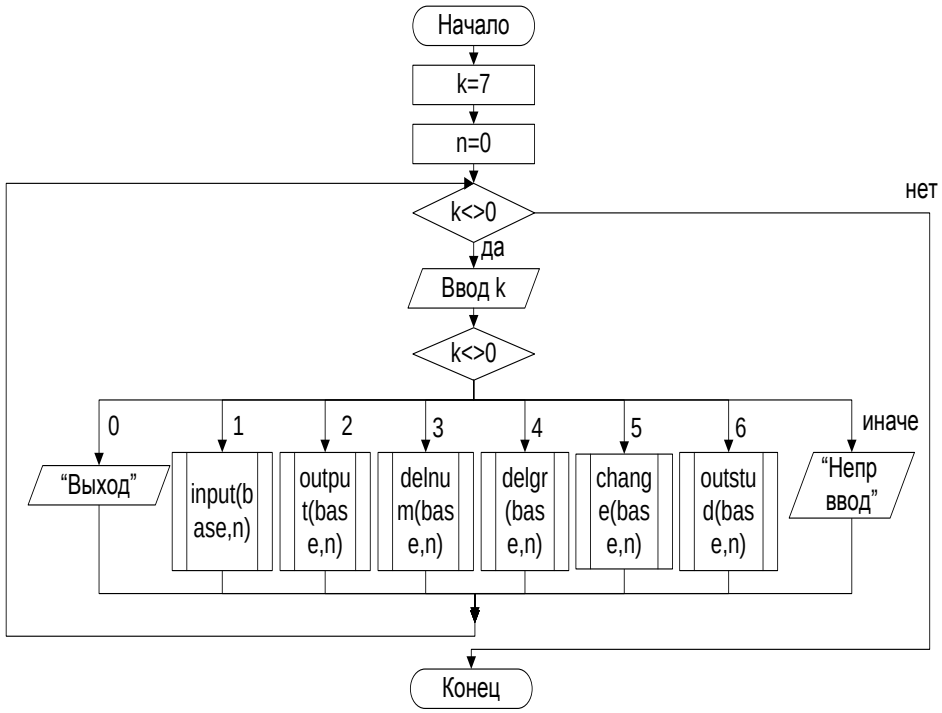


Рисунок 13.7 – Основной алгоритм (к- параметр для организации меню)

Листинг программы:

```
/* Выполнил студент группы ИТ-11 Иваненко Никита Игоревич */
#include <stdio.h>
#include <conio.h>
#include <string.h>
typedef struct Regrz
                {
                 char fios[21];
                 char gr[6];
                 char subj[7];
                 char date[10];
                 char fiop[21];
```

```
                char zach[8];
               } zaochn;
void input (zaochn *base, int *n);
void output (zaochn *base, int n);
void delnum (zaochn *base, int *n);
void delgr (zaochn *base, int *n);
void change (zaochn *base, int n);
void outstud (zaochn *base, int n);
main()
{
 int k,n;
 zaochn base[100];
 k=7;
 n=0;
 while (k!=0)
        {
         textcolor(11);
         textbackground(1);
         clrscr();
         puts("\t\t╔═══════════════════════════════════════════════════╗");
         puts("\t\t│                Главное меню.                      ");
         puts("\t\t│              Выберите действие:                   ");
         puts("\t\t├──────┬────────────────────────────────────────────┤");
         puts("\t\t│  1   │       Добавление записи(-сей)              ");
         puts("\t\t├──────┼────────────────────────────────────────────┤");
         puts("\t\t│  2   │   Вывод записей на экран в виде таблицы    ");
         puts("\t\t├──────┼────────────────────────────────────────────┤");
         puts("\t\t│  3   │       Удаление записи по номеру            ");
         puts("\t\t├──────┼────────────────────────────────────────────┤");
         puts("\t\t│  4   │Удаление записи(-сей) по конкретной группе  ");
         puts("\t\t├──────┼────────────────────────────────────────────┤");
         puts("\t\t│  5   │       Замена фамилии студента              ");
         puts("\t\t├──────┼────────────────────────────────────────────┤");
         puts("\t\t│  6   │ Вывод всех зачтенных работ конкретного лица");
puts("\t\t╟──────┼────────────────────────────────────────────╢");
         puts("\t\t│  0   │       Завершение работы программы          ");
         puts("\t\t╚═══════════════════════════════════════════════════╝");
         printf ("\t\tВаш выбор(0-6):");
         scanf ("%d",&k);
         switch (k)
                {
                 case 1 : {
```

```
                input(base,&n);
                puts("Добавление записи(-сей) окончено. Нажмите любую
клавишу для продолжения...");
                getch();
                }
        break;
        case 2 : {
                output(base,n);
                puts("Вывод записи(-сей) окончен. Нажмите любую
клавишу для продолжения...");
                getch();
                }
        break;
        case 3 : {
                delnum(base,&n);
                puts("Удаление записи окончено. Нажмите любую
клавишу для продолжения...");
                getch();
                }
        break;
        case 4 : {
                delgr(base,&n);
                puts("Удаление записи(-сей) окончено. Нажмите любую
клавишу для продолжения...");
                getch();
                }
        break;
        case 5 : {
                change(base,n);
                puts("Изменение записи(-сей) окончено. Нажмите любую
клавишу для продолжения...");
                getch();
                }
        break;
        case 6 : {
                outstud(base,n);
                puts("Вывод записи(-сей) окончен. Нажмите любую
клавишу для продолжения...");
                getch();
                }
        break;
        case 0 : {
                textcolor(14);
                clrscr();
```

```
                        puts("Выход... Нажмите любую клавишу для
продолжения...");
                        getch();
                       }
                break;
                default :  {
                        textcolor(12);
                        clrscr();
                        puts("Неправильный ввод! Вводите число от 0 до 6!");
                        getch();
                       }
                break;
               }
        }
 return(0);
}

void input (zaochn *base, int *n)
{               //Функция для ввода/добавления записей в структуру
 int i,m,nn,k;
 textcolor(10);
 clrscr();
 puts("Введите кол-во записей:");
 scanf("%d",&m);
 nn=*n;
 *n+=m;
 for (i=nn;i<*n;i++)
          {
          if (i%21==0||nn!=0&&i==nn)
                  {
                   clrscr();
                   k=4;

puts("┌────┬──────────────────────────┬──────┬─────────┬──────────
──────┬──────────────────────────┬─────────┐");
                   puts("│ № │    ФИО студента    │ Гр. │Предм.│Дата п.р.│
ФИО преподавателя │  Оц.  │");

puts("├────┼──────────────────────────┼──────┼─────────┼──────────
──────┼──────────────────────────┼─────────┤");
                  }
          printf("│%3d│",i+1);
          scanf("%s",&base[i].fios);
          gotoxy(26,k);
```

```
            printf("│");
            scanf ("%s",&base[i].gr);
            gotoxy(32,k);
            printf("│");
            scanf ("%s",&base[i].subj);
            gotoxy(39,k);
            printf("│");
            scanf ("%s",&base[i].date);
            gotoxy(49,k);
            printf("│");
            scanf("%s",&base[i].fiop);
            gotoxy(70,k);
            printf("│");
            scanf ("%s",&base[i].zach);
            gotoxy(78,k);
            printf("│\n");
            k++;
           }

puts("└─────┴──────────────────────────────┴────────┴─────────┴──────
──────┴──────────────────────────────┴─────────┘");
}

void output (zaochn *base, int n)
{               //Функция для вывода записей в виде таблицы
 int i,i_nach,i_kon,vvod,koef;
 clrscr();
 if (n>0)
     {
      i=0;
      koef=0;
      i_nach=0;
      i_kon=17;
      vvod=0;
      while (vvod!=27)
                {
                 if (i%(17+koef)==0||i==n)
                         {
                          if (i!=0)
                              {

        puts("└─────┴──────────────────────────────┴────────┴─────────┴──
──────┴──────────────────────────────┴─────────┘");
```

```
                                        if (i_nach>0) printf("Нажмите \"Backspace\" для
прокрутки текста вверх.\n");
                                        if (i<n) printf("Нажмите \"Enter\" для прокрутки текста
вниз.\n");
                                        printf("Нажмите \"Esc\" для выхода.\n");
                                        vvod=getch();
                                        switch (vvod)
                                              {
                                               case 8 : if (koef>0)
                                                           {
                                                            i_nach=i_nach-1;
                                                            koef--;
                                                           }
                                               break;
                                               case 13 : if (i_kon<n)
                                                           {
                                                            i_nach=i_nach+1;
                                                            koef++;
                                                           }
                                               break;
                                              }
                                     }
                                clrscr();

puts("┌─────┬─────────────────────────┬───────┬─────────┬──────────
──────────┬────────────────────────────┬──────────┐");
                                puts("│ № │    ФИО студента      │ Гр. │Предм.│Дата п.р.│
ФИО преподавателя │  Оц.   │");

puts("├─────┼─────────────────────────┼───────┼─────────┼──────────
──────────┼────────────────────────────┼──────────┤");
                               }
                     if (i_nach+17<n+1) i_kon=i_nach+17;
                          else i_kon=n;
                     for(i=i_nach;i<i_kon;i++) printf("│%3d│%-20s│%-5s│%-6s│%-
9s│%-20s│%-7s│\n",i+1,base[i].fios, base[i].gr, base[i].subj, base[i].date,
base[i].fiop, base[i].zach);
                     if (vvod==27)
puts("└─────┴─────────────────────────┴───────┴─────────┴──────────
──────────┴────────────────────────────┴──────────┘");
                    }
          }
             else puts ("Нет записей для вывода!");
}
```

```
void delnum (zaochn *base, int *n)
{           //Функция для удаление записи по номеру
 int i,j,k;
 textcolor(13);
 clrscr();
 if (*n>0)
      {
        puts("Введите № удаляемой записи:");
        scanf("%d",&k);
        for (i=k-1;i<*n;i++)
                for(j=0;j<22;j++)
                        {
                         if (j<22)
                                {
                                 base[i].fios[j]=base[i+1].fios[j];
                                 base[i].fiop[j]=base[i+1].fiop[j];
                                }
                         if (j<11) base[i].date[j]=base[i+1].date[j];
                         if (j<9)  base[i].zach[j]=base[i+1].zach[j];
                         if (j<8)  base[i].subj[j]=base[i+1].subj[j];
                         if (j<7)  base[i].gr[j]=base[i+1].gr[j];
                        }
        *n=*n-1;
      }
    else puts ("Нет записей для удаления!");
}

void delgr (zaochn *base, int *n)
{          //Функция для удаления записей по конкретной группе
 int i,j,k;
 char grp[6];
 textcolor(11);
 clrscr();
 if (*n>0)
     {
      puts("Введите название группы для удаления записи(-сей):");
      scanf("%s",&grp);
      i=0;
      while (i<*n)
              {
                if (strcmp(base[i].gr,grp)==0)
                                {
                                  for (k=i;k<*n-1;k++)
```

```
                                for (j=0;j<22;j++)
                                        {
                                         if (j<22)
                                              {
                                                base[k].fios[j]=base[k+1].fios[j];
                                                base[k].fiop[j]=base[k+1].fiop[j];
                                              }
                                         if (j<11) base[k].date[j]=base[k+1].date[j];
                                         if (j<9)  base[k].zach[j]=base[k+1].zach[j];
                                         if (j<8)  base[k].subj[j]=base[k+1].subj[j];
                                         if (j<7)  base[k].gr[j]=base[k+1].gr[j];
                                        }
                            *n=*n-1;
                           }
                if (strcmp(base[i].gr,grp)!=0) i++;
              }
     }
        else puts ("Нет записей для удаления!");
}

void change (zaochn *base, int n)
{             //Функция для смены фамилии студента с заданной на заданную
 int i,j;
 char stud[21],chstud[21];
 textcolor(12);
 clrscr();
 if (n>0)
      {
       puts("Введите начальную фамилию студента:");
       scanf("%s",&stud);
       puts("Введите фамилию студента для смены:");
       scanf("%s",&chstud);
       for (i=0;i<n;i++) if (strcmp(base[i].fios,stud)==0) for (j=0;j<22;j++) if
(strlen(chstud)+1>j) base[i].fios[j]=chstud[j];
                                                                        else
base[i].fios[strlen(chstud)+1]='\0';
      }
     else puts ("Нет записей для изменения!");
}

void outstud (zaochn *base, int n)
{           //Функция для вывода зачтенных работ конкретного лица
 int i,k;
 char stud[21];
```

```
 clrscr();
 if (n>0)
     {
      puts("Введите фамилию студента для вывода записи(-сей):");
      scanf("%s",&stud);
      k=0;
      for (i=0;i<n;i++) if
(strcmp(base[i].fios,stud)==0&&strcmp(base[i].zach,"зачет")==0) k++;
      if (k>0)
            {

puts("┌────┬──────────────────────┬──────┬─────────┬──────────┬──────────────────────┬──────────┐");
            puts("│ № │     ФИО студента     │ Гр. │Предм.│Дата п.р.│   ФИО преподавателя │   Оц.   │");

puts("├────┼──────────────────────┼──────┼─────────┼──────────┼──────────────────────┼──────────┤");
            k=1;
            for (i=0;i<n;i++)
if (strcmp(base[i].fios,stud)==0&&strcmp(base[i].zach,"зачет")==0)
printf("│%3d │%-20s │%-5s │%-6s │%-9s │%-20s │%-7s │\n",k,base[i].fios,
base[i].gr, base[i].subj, base[i].date, base[i].fiop, base[i].zach);

puts("└────┴──────────────────────┴──────┴─────────┴──────────┴──────────────────────┴──────────┘");
            }
           else printf ("У студента %s нет зачтенных работ!\n",stud);
     }
    else puts ("Нет записей для вывода!");
}
```

Тесты:
Был создан массив из 25 записей:

№	ФИО студента	Гр.	Предм.	Дата п.р.	ФИО преподавателя	Оц.
1	Петров	ОС	мат	11.11.2006	Авакян	зачет
2	Петров	ОС	физ	12.11.2006	Курбатова	зачет
3	Петров	ОС	физ	13.12.2006	Кравченко	незачет
4	Петров	ОС	инф	14.01.2007	Кротенок	зачет
5	Петров	ОС	физ	16.01.2007	Дробышевский	зачет
6	Петров	ОС	инф	17.02.2007	Ковалев	незачет
7	Петров	ОС	физ	18.02.2007	Ярчак	зачет
8	Петров	ОС	мат	21.03.2007	Авакян	незачет
9	Сидоров	ПЭ	физ	11.10.2006	Курбатова	зачет
10	Сидоров	ПЭ	физ	12.10.2006	Кравченко	зачет
11	Сидоров	ПЭ	инф	13.10.2006	Кротенок	незачет
12	Сидоров	ПЭ	физ	14.10.2007	Дробышевский	зачет
13	Сидоров	ПЭ	инф	11.11.2006	Ковалев	незачет
14	Сидоров	ПЭ	физ	12.11.2006	Ярчак	незачет
15	Сидоров	ПЭ	мат	13.12.2006	Авакян	незачет
16	Иванов	МК	физ	14.01.2007	Курбатова	зачет
17	Иванов	МК	физ	16.01.2007	Кравченко	зачет
18	Иванов	МК	инф	17.02.2007	Кротенок	зачет
19	Иванов	МК	физ	18.02.2007	Дробышевский	зачет
20	Иванов	МК	инф	21.03.2007	Ковалев	незачет
21	Иванов	МК	физ	22.03.2007	Ярчак	незачет
22	Иванов	МК	мат	23.03.2007	Авакян	незачет
23	Иванов	МК	физ	01.04.2007	Курбатова	зачет
24	Рыбин	ОС	инф	16.01.2007	Кравченко	зачет
25	Рыбин	ОС	инф	18.02.2007	Кротенок	незачет

1. Удаление записи по номеру
   k=20

№	ФИО студента	Гр.	Предм.	Дата п.р.	ФИО преподавателя	Оц.
1	Петров	ОС	мат	11.11.2006	Авакян	зачет
2	Петров	ОС	физ	12.11.2006	Курбатова	зачет
3	Петров	ОС	физ	13.12.2006	Кравченко	незачет
4	Петров	ОС	инф	14.01.2007	Кротенок	зачет
5	Петров	ОС	физ	16.01.2007	Дробышевский	зачет
6	Петров	ОС	инф	17.02.2007	Ковалев	незачет
7	Петров	ОС	физ	18.02.2007	Ярчак	зачет
8	Петров	ОС	мат	21.03.2007	Авакян	незачет
9	Сидоров	ПЭ	физ	11.10.2006	Курбатова	зачет
10	Сидоров	ПЭ	физ	12.10.2006	Кравченко	зачет
11	Сидоров	ПЭ	инф	13.10.2006	Кротенок	незачет
12	Сидоров	ПЭ	физ	14.10.2007	Дробышевский	зачет
13	Сидоров	ПЭ	инф	11.11.2006	Ковалев	незачет
14	Сидоров	ПЭ	физ	12.11.2006	Ярчак	незачет
15	Сидоров	ПЭ	мат	13.12.2006	Авакян	незачет
16	Иванов	МК	физ	14.01.2007	Курбатова	зачет
17	Иванов	МК	физ	16.01.2007	Кравченко	зачет
18	Иванов	МК	инф	17.02.2007	Кротенок	зачет
19	Иванов	МК	инф	21.03.2007	Ковалев	незачет
20	Иванов	МК	физ	22.03.2007	Ярчак	незачет
21	Иванов	МК	мат	23.03.2007	Авакян	незачет
22	Иванов	МК	физ	01.04.2007	Курбатова	зачет
23	Рыбин	ОС	инф	16.01.2007	Кравченко	зачет
24	Рыбин	ОС	инф	18.02.2007	Кротенок	незачет

2. Вывод зачтенных работ конкретного лица.
   stud=’Петров’

№	ФИО студента	Гр.	Предм.	Дата п.р.	ФИО преподавателя	Оц.
1	Петров	ОС	мат	11.11.2006	Авакян	зачет
2	Петров	ОС	физ	12.11.2006	Курбатова	зачет
3	Петров	ОС	инф	14.01.2007	Кротенок	зачет
4	Петров	ОС	физ	16.01.2007	Дробышевский	зачет
5	Петров	ОС	физ	18.02.2007	Ярчак	зачет

3. Удаление записей по конкретной группе.
   grp='ОС'

№	ФИО студента	Гр.	Предм.	Дата п.р.	ФИО преподавателя	Оц.
1	Сидоров	ПЭ	физ	11.10.2006	Курбатова	зачет
2	Сидоров	ПЭ	физ	12.10.2006	Кравченко	зачет
3	Сидоров	ПЭ	инф	13.10.2006	Кротенок	незачет
4	Сидоров	ПЭ	физ	14.10.2007	Дробышевский	зачет
5	Сидоров	ПЭ	инф	11.11.2006	Ковалев	незачет
6	Сидоров	ПЭ	физ	12.11.2006	Ярчак	незачет
7	Сидоров	ПЭ	мат	13.12.2006	Авакян	незачет
8	Иванов	МК	физ	14.01.2007	Курбатова	зачет
9	Иванов	МК	физ	16.01.2007	Кравченко	зачет
10	Иванов	МК	инф	17.02.2007	Кротенок	зачет
11	Иванов	МК	физ	18.02.2007	Дробышевский	зачет
12	Иванов	МК	инф	21.03.2007	Ковалев	незачет
13	Иванов	МК	физ	22.03.2007	Ярчак	незачет
14	Иванов	МК	мат	23.03.2007	Авакян	незачет
15	Иванов	МК	физ	01.04.2007	Курбатова	зачет

4. Смена фамилии студента.

stud='Иванов'; chstud='Петренко'

№	ФИО студента	Гр.	Предм.	Дата п.р.	ФИО преподавателя	Оц.
1	Сидоров	ПЭ	физ	11.10.2006	Курбатова	зачет
2	Сидоров	ПЭ	физ	12.10.2006	Кравченко	зачет
3	Сидоров	ПЭ	инф	13.10.2006	Кротенок	незачет
4	Сидоров	ПЭ	физ	14.10.2007	Дробышевский	зачет
5	Сидоров	ПЭ	инф	11.11.2006	Ковалев	незачет
6	Сидоров	ПЭ	физ	12.11.2006	Ярчак	незачет
7	Сидоров	ПЭ	мат	13.12.2006	Авакян	незачет
8	Петренко	МК	физ	14.01.2007	Курбатова	зачет
9	Петренко	МК	физ	16.01.2007	Кравченко	зачет
10	Петренко	МК	инф	17.02.2007	Кротенок	зачет
11	Петренко	МК	физ	18.02.2007	Дробышевский	зачет
12	Петренко	МК	инф	21.03.2007	Ковалев	незачет
13	Петренко	МК	физ	22.03.2007	Ярчак	незачет
14	Петренко	МК	мат	23.03.2007	Авакян	незачет
15	Петренко	МК	физ	01.04.2007	Курбатова	зачет

14 СОРТИРОВКА

. В основу этого анализа закладывается понятие эффективности алгоритма. При изучении дисциплины активно используются сведения, полученные студентами из курсов высшей математики, теории вероятностей, информатики, программирования на языках высокого уровня.

14.1 Общие сведения

Сортировка – это операция, упорядочивающая последовательность (массив) элементов по ключам. **Ключ** – это некоторое числовое свойство элемента массива. Т. е. каждому элементу массива ставится в соответствие некоторое число, называемое ключом элемента.

Если мы имеем числовой массив, то ключ элемента – это сам элемент. В этом случае сортировка массива – это упорядочивание массива (перестановка его элементов) таким образом, чтобы получилась неубывающая или невозрастающая последовательность.

Если каждый элемент исходного массива представляет собой слова, составленные из букв русского алфавита, то ключ, по которому может быть упорядочен массив, связывается с порядковым номером буквы в алфавите. По этому принципу упорядочиваются слова в словарях – из двух слов первым помещается то слово, ключ которого меньше. Здесь принимается, что отсутствие буквы (т. е. пустая строка) имеет меньший ключ, чем любая другая буква. Так слово "студент" помещается в словаре перед словом "студентка".

Если слова состоят из букв разных алфавитов и цифр, как, например, имена файлов и папок, то любая цифра имеет меньший ключ, чем любая буква, а любая латинская буква имеет меньший ключ по сравнению с любой русской буквой. При сортировке таких имен в качестве ключа используется код символа в некоторой кодовой таблицы. По этому принципу, например, отсортированы имена встроенных функций MS Excel в категории "Полный алфавитный перечень". Этот принцип упорядочивания еще называют лексикографическим порядком или расширенным алфавитом.

Разработкой различных алгоритмов сортировки информации, хранящейся в оперативной памяти компьютера или на его жестком диске, программисты занимаются уже давно. Интерес к этой проблеме обусловлен тем, что по мнению специалистов 25% всего времени обработки информации расходуется на сортировку данных.

Ясно, что с отсортированными данными работать легче, чем с произвольно расположенными. Когда элементы отсортированы, то проще найти нужный элемент или установить, что его нет.

Уточним терминологию.

Если элементы массива связаны отношениями $a_0 < a_1 < \cdots < a_{n-1}$, то говорят, что массив упорядочен по **возрастанию**. Такая упорядоченность предполагает, что в массиве нет одинаковых элементов.

Если элементы массива связаны отношениями $a_0 \le a_1 \le \cdots \le a_{n-1}$, то говорят, что массив упорядочен по **неубыванию**. Такая упорядоченность не исключает наличие в массиве одинаковых элементов.

Если элементы массива связаны отношениями $a_0 > a_1 > \cdots > a_{n-1}$, то говорят, что массив упорядочен по **убыванию.** Такая упорядоченность предполагает, что в массиве нет одинаковых элементов.

Если элементы массива связаны отношениями $a_0 \ge a_1 \ge \cdots \ge a_{n-1}$, то говорят, что массив упорядочен по **невозрастанию**. Такая упорядоченность не исключает наличие в массиве одинаковых элементов.

14.2 Классификация методов сортировки

Все методы сортировки можно разделить на пять групп:

1) методы извлечения;
2) методы включения;
3) методы обменов;
4) методы слияния;
5) методы распределения.

Общая концепция **методов извлечения** заключается в следующем: из исходного массива извлекается минимальный элемент и меняется местами с первым элементом массива, затем извлекается минимальный элемент из части массива, начиная со второго элемента, и меняется местами со вторым элементом и т. д. Последний раз минимальный элемент выбирается из двух последних элементов массива. В результате получится массив, упорядоченный по неубыванию.

Различные методы извлечения отличаются объектом извлечения (минимальный или максимальный элемент) и, соответственно, объектами перестановки (первый или последний элемент), а также условием окончания процесса сортировки.

В качестве примера рассмотрим следующий массив:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
8	1	3	0	4	8	0	3

Выберем минимальный элемент из всех элементов массива: a_3=0 и переставим его с первым элементом массива. Получим следующий массив:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	1	3	8	4	8	0	3

Теперь выберем минимальный элемент из элементов массива a_1, a_2,…, a_7 (это элемент a_6=0) и переставим его с элементом a_1. Получим:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	0	3	8	4	8	1	3

Дальнейшие шаги по сортировке массива приведены ниже:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	0	1	8	4	8	3	3
0	0	1	3	4	8	8	3
0	0	1	3	3	8	8	4
0	0	1	3	3	4	8	8
0	0	1	3	3	4	8	8

Идея **методов включения** состоит в том, что сначала первый элемент массива рассматривается как упорядоченный массив и в этот массив включается следующий элемент исходного массива так, чтобы получился упорядоченный по неубыванию массив из двух элементов. Затем в полученный упорядоченный массив включается третий элемент массива так, чтобы опять-таки получился упорядоченный массив. Процесс продолжается до тех пор, пока не будет включен последний элемент.

Различные алгоритмы включения отличаются способами выбора элемента для включения, способами определения места включения и методами самого включения.

В качестве примера рассмотрим тот же массив:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
8	1	3	0	4	8	0	3

После включения элемента a_1=1 в массив, состоящий из одного элемента a_0=8, получится следующий массив:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	8	3	0	4	8	0	3

Далее включаем элемент a_2=3 в массив a_0, a_1. Получим:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	3	8	0	4	8	0	3

Продолжая этот процесс, мы последовательно будем получать следующие состояния массива:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	1	3	8	4	8	0	3
0	1	3	4	8	8	0	3
0	0	1	3	4	8	8	3
0	0	1	3	3	4	8	8

Идея **методов обменов** состоит в следующем: в исходном массиве выбирается пара элементов, и они сравниваются между собой. Если их положение не удовлетворяет требованию упорядоченности, то элементы переставляются. Затем выбирается следующая пара элементов и так до тех пор, пока не получим упорядоченный массив.

Различные алгоритмы обменов отличаются способами выбора пары элементов для сравнения и перестановки, а также условиями окончания процесса сортировки.

В качестве примера рассмотрим сортировку по неубыванию того же массива:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
8	1	3	0	4	8	0	3

Сравним первые два элемента и переставим их в соответствии с требование неубывания:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	8	3	0	4	8	0	3

Теперь проделаем ту же процедуру для элементов a_1 и a_2. Получим:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	3	8	0	4	8	0	3

Продолжая этот процесс, мы последовательно будем получать следующие состояния массива:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	3	8	0	4	8	0	3
1	3	0	8	4	8	0	3
1	3	0	4	8	8	0	3
1	3	0	4	8	8	0	3
1	3	0	4	8	0	8	3
1	3	0	4	8	0	3	8

Закончился первый просмотр массива. В результате массив не упорядочился, но максимальный элемент массива занял свое окончательное место – в конце массива. Теперь повторим процесс для элементов a_0, a_1,…, a_6.

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	3	0	4	8	0	3	8
1	0	3	4	8	0	3	8
1	0	3	4	0	8	3	8
1	0	3	4	0	3	8	8

После окончания второго просмотра массива два последних элемента заняли свое окончательное место. Проведем третий просмотр массива без рассмотрения двух последних элементов.

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	1	3	4	0	3	8	8
0	1	3	0	4	3	8	8
0	1	3	0	3	4	8	8

Уже три последних элемента заняли свои окончательные места. Повторим процесс:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	1	3	0	3	4	8	8
0	1	0	3	3	4	8	8

Последний шаг процесса сортировки:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	0	1	3	3	4	8	8

Метод слияния применяется в том случае, когда имеются два (или больше) упорядоченных массива и требуется соединить исходные массивы в один общий упорядоченный массив.

Пусть, например, имеются два массива:

A_0	A_1	A_2	A_3	A_4
0	0	1	3	7

B_0	B_1	B_2	B_3
9	3	3	0

Требуется получить общий массив, упорядоченный по неубыванию.

Ясно, что для решения задачи надо сравнивать элементы двух массивов, причем в первом массиве надо выбирать элементы слева направо, а во втором массиве – справа налево.

Получим следующий массив:

C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
0	0	0	1	3	3	3	7	9

Метод распределения употребим в тех случаях, когда в исходном массиве имеется заданное, известное заранее, количество различных ключей (значений). Например, имеется список студентов с оценками по пятибалльной системе, полученными на экзамене. Нам известно заранее, что оценки могут быть 5, 4, 3 и 2. Поэтому для упорядочения массива по невозрастанию можно сначала выбрать все записи с оценками 5, затем с оценками 4, потом с оценками 3 и, наконец, с оценками 2.

14.3 Алгоритм сортировки методом извлечения

Упорядочим заданный целочисленный массив по **неубыванию** на основе алгоритма **извлечения минимального** элемента.

Разработаем **алгоритм** сортировки на основе извлечения минимального элемента. Сначала запишем необходимые действия в **словесной форме**:

1) найдем минимальный элемент среди всех элементов массива $a_0, a_1, \cdots, a_{n-1}$ и определим его номер. Пусть это будет элемент a_m;
2) поменяем местами элементы a_0 и a_m. Таким образом, минимальный элемент массива окажется на своем окончательном месте;
3) теперь найдем минимальный элемент среди элементов массива, начиная со второго $a_1, a_2, \cdots, a_{n-1}$ и определим его номер. Опять обозначим этот элемент a_m;
4) поменяем местами элементы a_1 и a_m. В результате два первых элемента массива окажутся на своих окончательных местах;
5) на заключительном этапе этого процесса надо выбрать минимальный элемент из двух последних элементов массива a_{n-2} и a_{n-1}. После чего этот элемент должен быть поставлен на предпоследнее место.

Теперь обобщим представленные шаги алгоритма следующим образом: нахождение минимального элемента a_m среди элементов $a_k, a_{k+1}, \cdots, a_{n-1}$ и

последующая перестановка элементов a_m и a_k, при этом указанный процесс должен повторяться при изменении *k* от *0* до *n-2*.

Сформулированному алгоритму соответствует **графическая схема алгоритма**, приведенная на рис. 14.1.

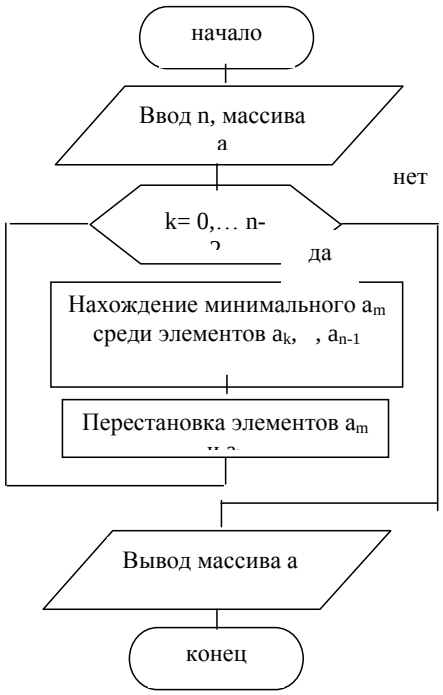


Рисунок. 14.1 – Схема алгоритма сортировки массива по неубыванию методом извлечения

Мы не будем детализировать блок нахождения минимального элемента в отрезке массива $a_k, a_{k+1},\cdots, a_{n-1}$, т. к. в ранее это было реализовано.

Отладку алгоритма следует провести для следующих тестов:

1) массив содержит все одинаковые элементы;
2) исходный массив упорядочен по неубыванию;
3) исходный массив упорядочен по невозрастанию;
4) элементы массива неупорядочены, причем в массиве несколько минимальных элементов.

14.4 Алгоритм сортировки методом обменов

Упорядочим заданный целочисленный массив по **неубыванию** на основе алгоритма "**пузырька**", относящегося к алгоритмам **обменов.**

Сначала запишем необходимые действия в **словесной форме**:

1) сравним элементы a_0 и a_1. Если не выполняется условие, $a_0 \le a_1$,то меняем местами эти элементы и сравниваем элементы a_1 и a_2. Так сравниваем и при необходимости меняем местами все пары исходного массива. Последней рассматривается пара a_{n-2} и a_{n-1}. Так заканчивается первый просмотр массива, при котором максимальный элемент окажется на последнем месте. Это напоминает процесс вскипания воды: первым всплывает самый большой пузырек. Именно поэтому рассматриваемый способ сортировки называется методом "пузырька";
2) второй просмотр массива опять-таки начинается со сравнения элементов a_0 и a_1. Последней рассматривается пара a_{n-3} и a_{n-2}. В результате на месте элемента a_{n-2} окажется второй по величине элемент массива (всплыл второй по величине пузырек);
3) третий просмотр массива начинается с проверки пары a_0 и a_1, заканчивается проверкой пары a_{n-4} и a_{n-3}, на месте элемента a_{n-3} окажется третий по величине элемент массива;
4) при последнем просмотре будут сравниваться только элементы a_0 и a_1.

Теперь обобщим представленные шаги алгоритма следующим образом: просмотр массива состоит в проверке условия $a_i \le a_{i+1}$ и перестановке этих элементов при невыполнении условия неубывания, при этом значение переменной *i* изменяется от 0 до некоторого *k*, т. е. последнее проверяемое условие будет $a_k \le a_{k+1}$. Первый просмотр происходит при *k=n-2*, следующий при *k=n-3*, затем при *k=n-4* и т. д. до *k=0*.

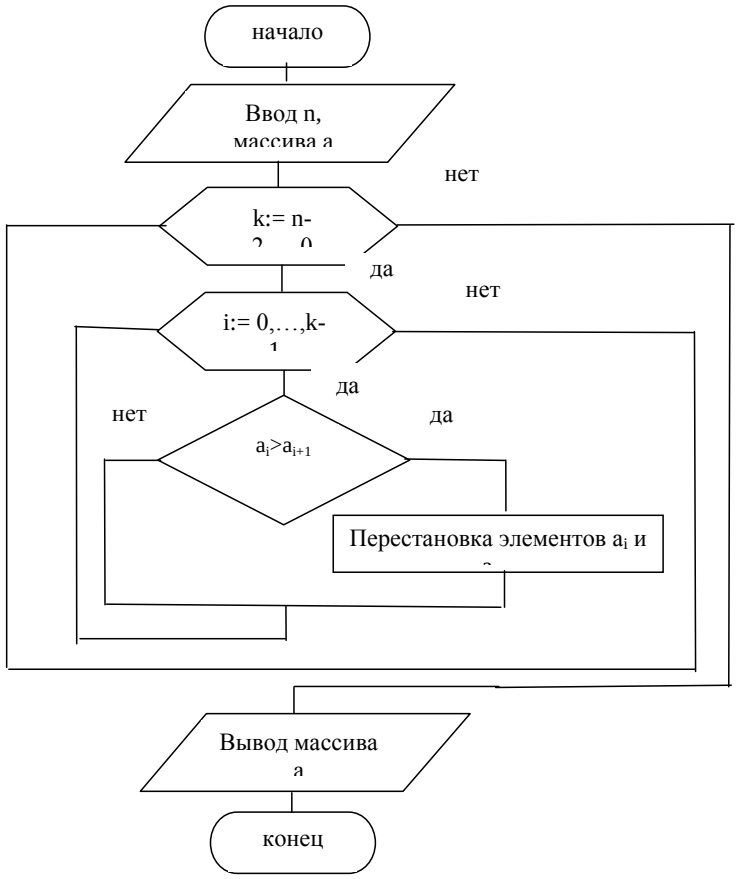


Рисунок 14.2 – Схема алгоритма сортировки массива по неубыванию методом "пузырька"

Сформулированному алгоритму соответствует **графическая схема алгоритма**, приведенная на рис. 14.2.

Отладку алгоритма следует провести для тех же тестов, что и в предыдущем проекте.

14.5 Минимизация числа просмотров при сортировке методом "пузырька"

При сортировке методом "пузырька" часто встречается ситуация, когда массив уже отсортирован, а просмотры массива продолжаются. Чтобы вовремя прекратить процесс сортировки, будем фиксировать факт перестановки в какой-нибудь переменной.

Для этой цели лучше всего подходит переменная логического типа. Она принимает одно из двух значений: ***True*** или ***False***.

С целью ускорения сортировки пузырьковым методом будем присваивать переменной ***W*** значение ***True*** всякий раз, когда после проверки очередной пары происходила перестановка значений сравниваемых элементов. Очередной оборот цикла для организации нового просмотра (цикл по переменной ***k***) будем выполнять только в том случае, когда при предыдущем просмотре была сделана хотя бы одна перестановка. Перед началом цикла проверки упорядоченности (цикл по переменной ***i***) надо переменной ***W*** присвоить значение ***False***, признак того, что пока перестановок не было.

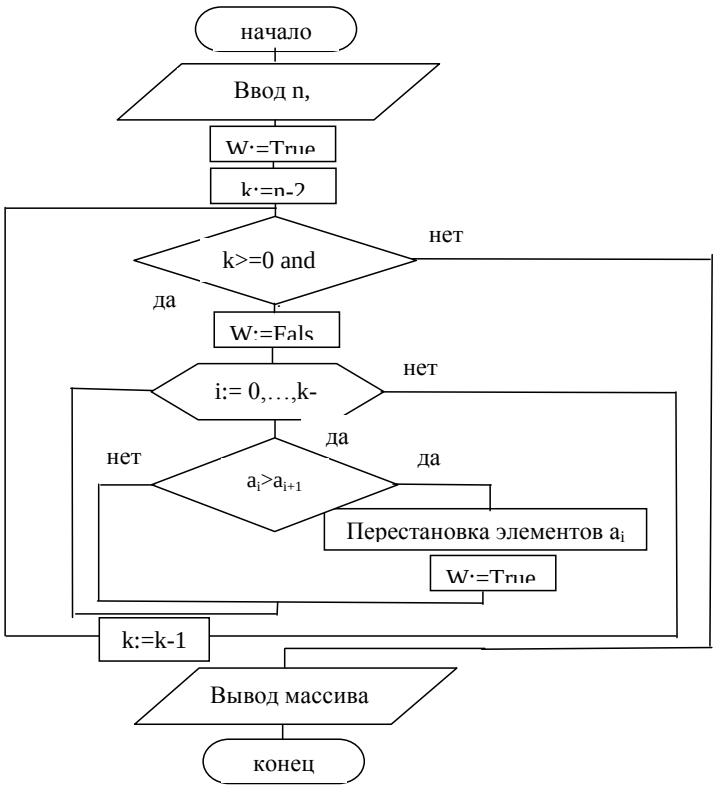


Рисунок 14.3 – Схема алгоритма сортировки массива по неубыванию методом "пузырька" с минимизацией числа просмотров

Схема алгоритма приведена на рис. 14.3.

Обратите внимание на блок ***W:= True*** перед циклом по переменной ***k***. Такой блок обеспечивает выполнение цикла первый раз.

14.6 Сортировка методом обменов за один просмотр "с возвращением"

Рассмотрим еще один алгоритм сортировки методом обменов. Как и в алгоритме методом "пузырька", мы будем последовательно проверять на

упорядоченность по неубыванию пары, начиная с $a_0 \le a_1$ и до $a_{n-2} \le a_{n-1}$. Однако после перестановки элементов, например a_k и a_{k+1}, мы не будем сразу продолжать просмотр слева направо, а будем устанавливать правильное местоположение элемента a_{k+1}, проверяя пары элементов справа налево. Таким образом, идея рассматриваемого алгоритма состоит в том, что после нахождения пары, не удовлетворяющей условию неубывания, т. е. пары $a_{k-1} > a_k$, мы в упорядоченной части массива $a_0,\ldots,a_{k-1}$ отыскиваем такое место для элемента a_k, чтобы отсортированной оказалась часть массива $a_0,\ldots,a_k$. После этого можно продолжать просмотр массива слева направо, т. е. можно переходить к проверке условия $a_k > a_{k+1}$.

Схема описанного алгоритма приведена на рис. 14.4.

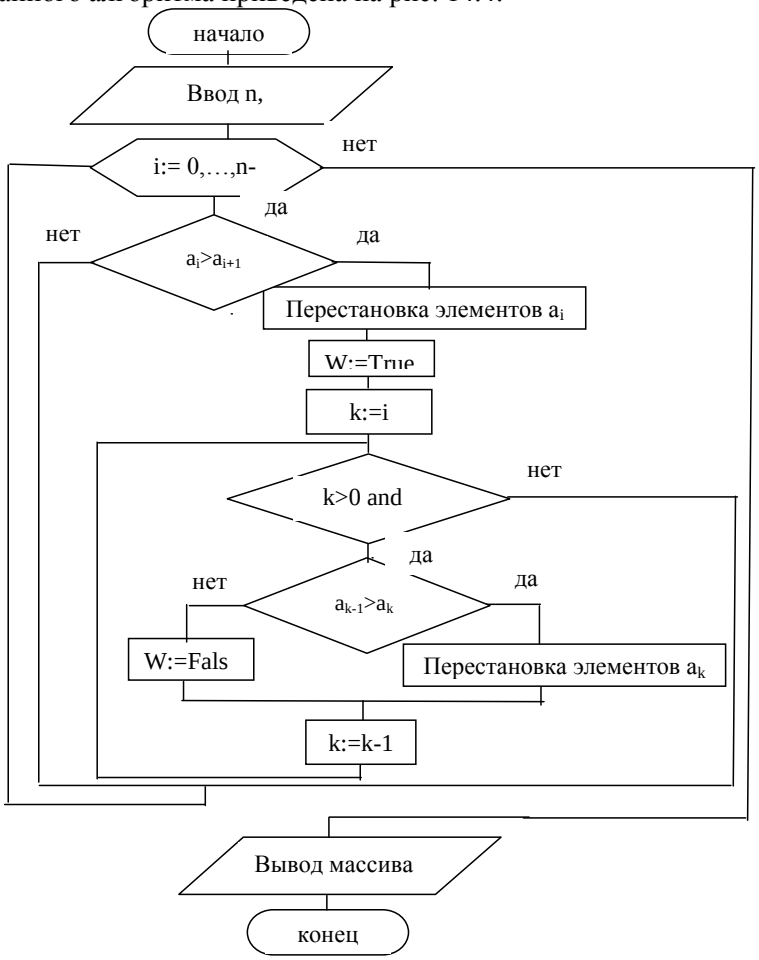


Рисунок 14.4 – Схема алгоритма сортировки массива по неубыванию методом обменов за один просмотр "с возвращением"

В этом алгоритме внутренний цикл обеспечивает возврат от найденной неупорядоченной пары $a_i > a_{i+1}$ к началу массива. Перед началом этого цикла переменной ***W*** присвоено значение ***True***. Данный цикл заканчивается при выполнении одного из двух условий:

1) ***k=0*** – проверены и переставлены все пары элементов, предшествующие элементу a_i;

2) ***W=False*** – нашлась упорядоченная пара $a_{k-1} \le a_k$.

14.7 Алгоритм сортировки методом слияния

Рассмотрим алгоритм создания упорядоченного по неубыванию массива методом слияния двух массивов, один из которых упорядочен по невозрастанию, а другой - по неубыванию.

Имеем массив A из n элементов и массив B из m элементов:

$a_0 \le a_1 \le \cdots \le a_{n-1}$, - неубывание;

$b_0 \ge b_1 \ge \cdots \ge b_{m-1}$ - невозрастание.

Требуется получить массив c из n+m элементов:

$c_0 \ge c_1 \ge \cdots \ge c_{n+m-1}$ - неубывание.

План решения задачи.

Пусть i - номер очередного элемента массива C. Тогда i=0,1,2,…,n+m-1.

Пусть k - номер максимального элемента среди оставшихся в массиве A. Сначала k=n-1. По мере переписывания элементов из массива A в массив C k уменьшается на единицу (k=k-1).

Пусть *l* - номер максимального элемента среди оставшихся в массиве B. Сначала l=0. По мере переписывания элементов из массива B в массив C *l* увеличивается на единицу (*l*=*l*+1).

Если не исчерпаны элементы в массивах A и B, то $c_i = \max\{a_k, b_l\}$. Если же исчерпаны элементы в массиве A, то $c_i = b_l$, затем *l*=*l*+1. Если же исчерпаны элементы в массиве B, то $c_i = a_k$, затем k=k-1.

Схема описанного алгоритма приведена на рисунке 3.6.

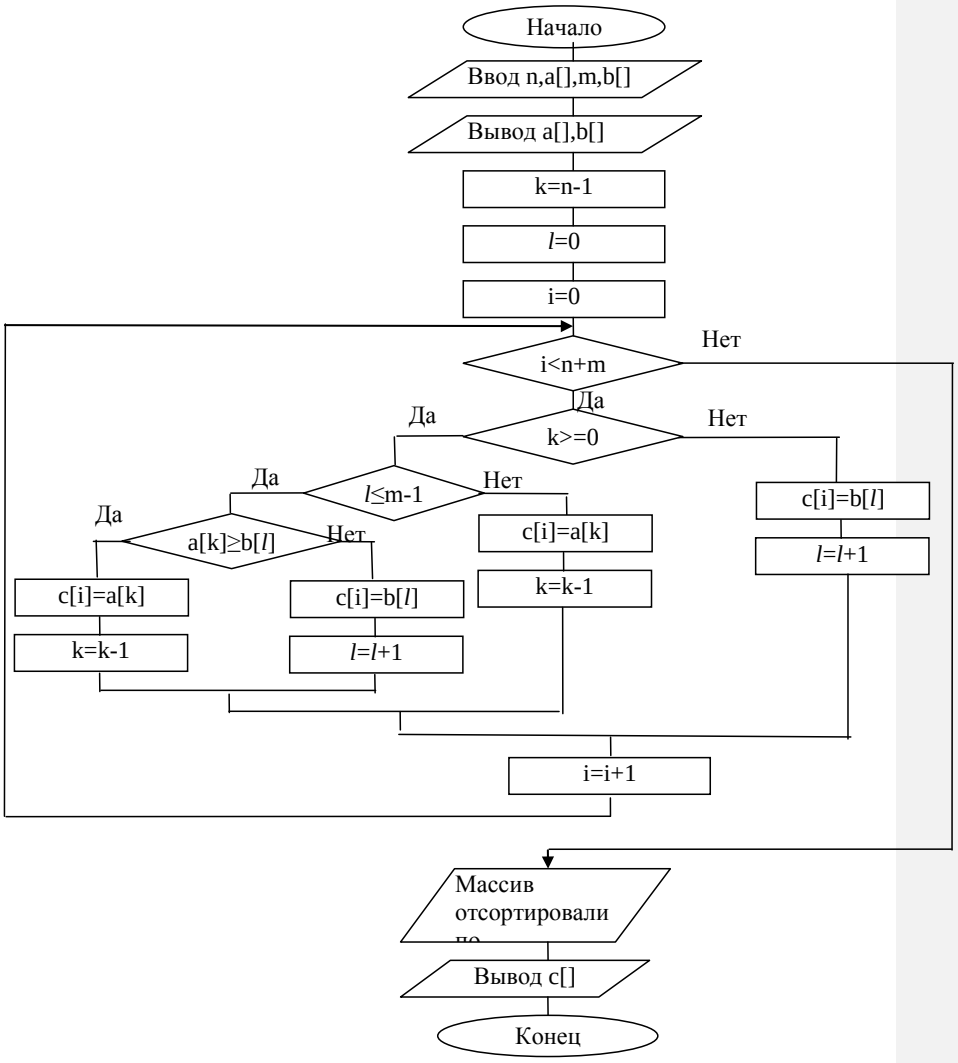


Рисунок 14.5 – Схема алгоритма сортировки массива слиянием

Таблица 14.1 – Таблица соответствия:

Имя в задаче	Имя наСи	Тип	Комментарий
a	a	float	исходные данные (массив)
b	b	float	исходные данные (массив)
n	n	int	количество элементов в массиве A
m	m	int	количество элементов в массиве B

k	k	int	счетчик
l	*l*	int	счетчик
i	i	int	счетчик

```
#include<stdio.h>
#include<conio.h>
void main()
{
  clrscr();
  float a[50],b[50];
  int n,m,k,l,i;
  puts("Введите количество элементов массива A, упорядоченного по
неубыванию");
  scanf("%d",&n);
  for(i=0;i<n;i++)
         {
          printf("Введите a[%d]\n",i);
          scanf("%f",&a[i]);
         }
  puts("Введите количество элементов массива B, упорядоченного по
невозрастанию");
  scanf("%d",&m);
  for(i=0;i<m;i++)
         {
          printf("Введите b[%d]\n",i);
          scanf("%f",&b[i]);
         }
  clrscr();
  puts("Исходные данные");
  puts("Массив A");
  for(i=0;i<n;i++) printf("%5.1f",a[i]);
  printf("\n");
  puts("Массив B");
  for(i=0;i<m;i++) printf("%5.1f",b[i]);
  printf("\n");
  float *c=new float[m+n];
  k=0; l=m-1;
  for(i=0;i<n+m;i++)
   {
        if(k<n)
         {if(l>=0)
           {if(a[k]<b[l])  { c[i]=a[k]; k++;}
                 else { c[i]=b[l]; l--;}
```

```
          }
         else { c[i]=a[k]; k++;}
        }
       else { c[i]=b[l]; l--;}
  }
 puts("Массив отсортировали по неубыванию");
 for(i=0;i<n+m;i++)  printf("%5.1f",c[i]);
 delete[]c;
}
```

Тесты:

Данные: Результат:

1. a[5]={1,2,4,6,8}; b[4]={11,9,7,-3} c[9]={-3,1,2,4,6,7,8,9,11}
2. a[5]={1,1,1,1,1}; b[3]={2,2,2} c[8]={1,1,1,1,1,2,2,2}
3. a[4]={3,6,7,10}; b[5]={22,21,18,17,10} c[9]={3,6,7,10,10,17,18,21,22}

14.8 Алгоритм сортировки распределением

Сортировку распределением в отличие от ранее рассмотренных методов нельзя применять для упорядочивания произвольного массива. Данный метод употребим в том случае, когда в исходном числовом массиве имеется небольшое количество различных значений (ключей) и множество этих значений известно заранее.

Отсортировать массив по невозрастанию методом распределения по массиву ключей, упорядоченному по невозрастанию.

Таблица 14.2 – Таблица соответствия:

Имя в задаче	Имя наСи	Тип	Комментарий
a	a	float	исходные данные (массив)
n	n	int	количество элементов в массиве
b	b	float	массив ключей
m	m	int	количество элементов в массиве ключей
i	i	int	счетчик
j	j	int	счетчик
k	k	int	количество совпадений элементов массива с ключом

Тесты:

Данные:	Результат:
1. a[9]={10,10,8,8,7,6,2,2,2}; b[5]={10,8,7,6,2} a[9]={10,10,8,8,7,6,2,2,2}	
2. a[9]={2,2,2,6,7,8,8,10,10}; b[5]={10,8,7,6,2} a[9]={10,10,8,8,7,6,2,2,2}	
3. a[9]={10,2,8,6,8,10,2,2,7}; b[5]={10,8,7,6,2} a[9]={10,10,8,8,7,6,2,2,2}	
4. a[4]={1,1,1,1}; b[1]={1}	a[4]={1,1,1,1}

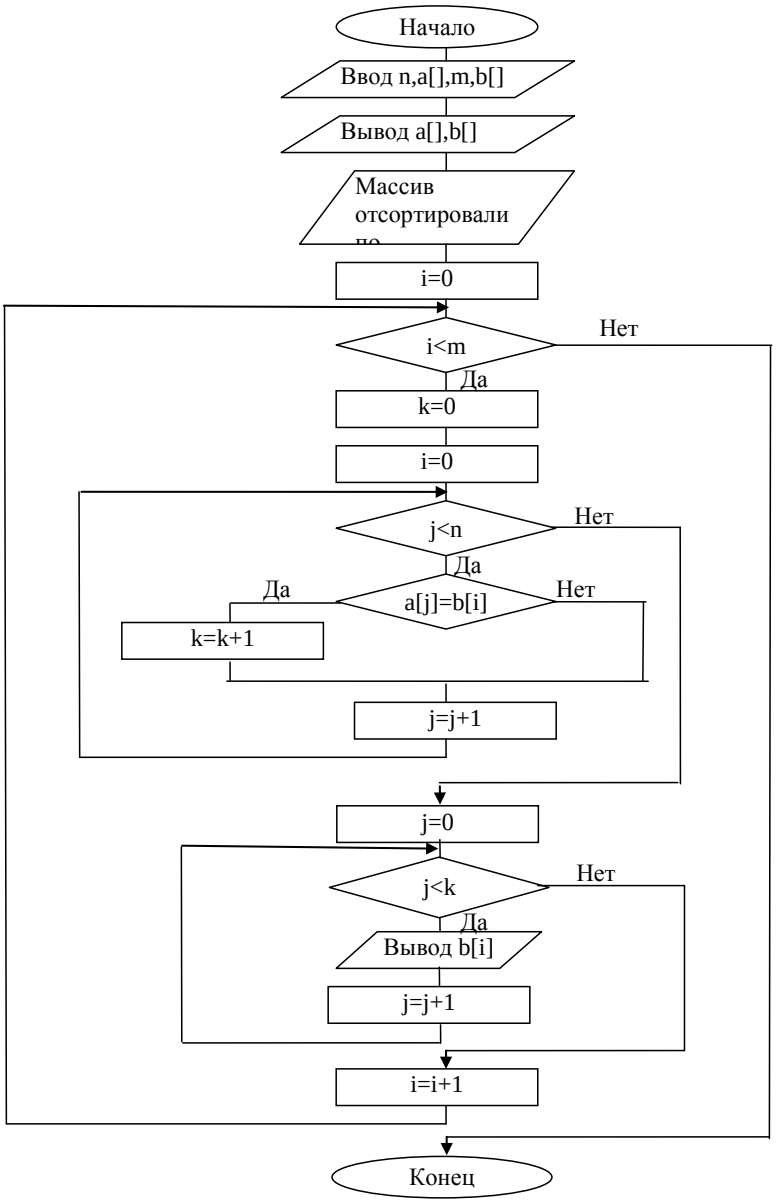


Рисунок 14.6 – Схема сортировки массива методом распределения

Текст программы

```
#include<stdio.h>
#include<conio.h>
```

```
void main()
{
  clrscr();
  float a[50],b[20];
  int n,k,j,i,m;
  puts("Введите количество элементов массива");
  scanf("%d",&n);
  for(i=0;i<n;i++)
         {
          printf("Введите a[%d]\n",i);
          scanf("%f",&a[i]);
         }
  clrscr();
  puts("Исходный массив");
  for(i=0;i<n;i++) printf("%5.1f",a[i]);
  printf("\n");
  puts("Введите количество ключей");
  scanf("%d",&m);
  for(i=0;i<m;i++)
         {
          printf("Введите ключ b[%d]\n",i);
          scanf("%f",&b[i]);
         }
  printf("\n");
  puts("Массив ключей");
  for(i=0;i<m;i++) printf("%5.1f",b[i]);
  printf("\n");
  puts("Массив отсортировали по невозрастанию");
  for(i=0;i<m;i++)
   {k=0;
    for(j=0;j<n;j++)
      if(a[j]==b[i]) k++;  //Кол-во совпавших элементов
    for(j=0;j<k;j++)
     printf("%5.1f",b[i]);
   }
}
```

14.9 Пример перестановки строк матрицы в порядке невозрастания сумм ее строк

В матрице mxn переставить строки таким образом, чтобы получилась последовательность
F1 > F2 >...> Fm, где F - сумма всех элементов i-ой строки.

Таблица 14.3 – Таблица соответствия:

Имя в задаче	Имя наСи	Тип	Комментарий
x	x	float	исходные данные (массив)
n	n	int	количество столбцов
m	m	int	количество строк
t	t	float	вспомогательная переменная
a	a	int	вспомогательная переменная
j	j	int	счетчик
i	i	int	счетчик
s1	s1	float	сумма элементов i-ой строки
s2	s2	float	сумма элементов a-ой строки

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int x[20][20],t;
 int n,m,i,a,j,s1,s2;
 clrscr();
 puts("Введите количество строк");
 scanf("%d",&m);
 puts("Введите количество столбцов");
 scanf("%d",&n);
 for(i=0;i<m;i++)      //Ввод матрицы
        for(j=0;j<n;j++)
          {
           printf("Введите x[%d][%d] : ",i,j);
           scanf("%d",&x[i][j]);
          }
 clrscr();
 puts("Исходный массив:");
 for(i=0;i<m;i++)    //Вывод матрицы
  {
        for(j=0;j<n;j++)
          printf(" %4d ",x[i][j]);
        printf("\n");
        }
 for(i=0;i<m;i++)
  {
        s1=0;
        for(j=0;j<n;j++)     //находим хар-ку i строки
          s1=s1+x[i][j];
```

```
        a=i+1;
        while(a<m)   //сравниваем i строку со всеми последующими
         {          //и ставим на ее место строку с наибольшей S
          s2=0;
          for(j=0;j<n;j++)
            s2=s2+x[a][j];
          if(s1<s2)
           {s1=s2;  //хар-ка i строки становится равной s2
                 for (j=0;j<n;j++)
                  {
                   t=x[i][j];       //меняем местами элементы
                   x[i][j]=x[a][j];
                   x[a][j]=t;
                  }
           }
          a++;
         }
   }
  printf("\n");
  puts("Искомый массив:");
  for(i=0;i<m;i++)    //Вывод результата
   {
        for(j=0;j<n;j++)
          printf(" %4d ",x[i][j]);
        printf("\n");
        }
}
```

Тесты:

	Данные:	Результат:
1.	7 4 6 3 1 10 3 2 12 0 0 1 -4 4 6 1	. 7 4 6 3 1 10 3 2 12 0 0 1 -4 4 6 1
2.	2 5 -5 0 12 0 -3 5 14 6 2 3 11 3 4 0	14 6 2 3 11 3 4 0 12 0 -3 5 2 5 -5 0
3.	0 0 -4 1 2 -1 5 2 3	2 4 11 5 2 3 1 2 -1

2 4 11 0 0 -4

14.10 Применение алгоритма сортировки методом "пузырька" для расположения записей в массиве записей в лексикографическом порядке

Дан массив записей, содержащий данные о студентах университета. Полями записей являются: ФИО, название факультета, группа, средний балл.

Требуется отсортировать и вывести записи массива в лексикографическом порядке названий факультета.

Для сортировки использовался метод пузырька, графическая схема которого изображена на рисунке 3.2.

Поскольку в алгоритме требуется сравнивать величины не числовые, а строковые (поля Факультет), то для сравнения строк использована функция strcmp(str1,str2), которая подробно рассматривалась в разделе 1.2.3.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<conio.h>
//  Объявления шаблона структурной переменной
struct student
  {
   char fio[25];
   char facultet[10];
   char grup[11];
   double sr;
  }stud[10],p;

main()
{
  int i,j,n,k;
  i=0;
  puts(“Введите количество записей”);
  scanf(“%d”,&n);
  //Ввод массива записей с консоли
  for (i=0;i<n; i++)
   {
     puts(“Введите ФИО”);
     scanf(“%s”,&stud[i].fio)
     puts(“Введите факультет”);
     scanf(“%s”,&stud[i].facultet);
     puts(“Введите группу”);
     scanf(“%s”,&stud[i].grup);
     puts(“Введите средний балл”);
```

```
    scanf(“%f”,&stud[i].sr)
   }

//Сортировка массива записей по  полю факультет в //алфавитном порядке
  for (j=0;j<n-1;j++)
    for(k=n-1;k>j;k--)
      if (strcmp(stud[k].facultet,stud[k-1].facultet)<0)
          {
       // поменять записи местами
            p=stud[k];
            stud[k]=stud[k-1];
            stud[k-1]=p;
          }
//Вывод на экран данных в виде таблицы
printf("-----------------------------------------------------\n");
  for (j=0;j<i;j++)
    printf("|%22s|%10s|%10s|%6.2f|\n",stud[j].fio, stud[j].facultet, stud[j].grup,
stud[j].sr);
printf("-----------------------------------------------------\n");

getch();
return(0);
}
```

14.11 Использование встроенной функции qsort из библиотеки stdlib для сортировки массива записей

Дан массив записей, состоящий из полей: имя, фамилия, год рождения. Вывести в алфавитном порядке записи по полю фамилия, используя встроенную функцию сортировки qsort.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//Шаблон структуры студенты
typedef struct{
  char name[20];
  char family[20];
  int year;
} TStudent;
```

```
//Как сравнить 2-х студентов по фамилии. Эта процедура будет являться
параметром //функции сортировки
int sort_cmp( const void *a, const void *b)
  {
     TStudent *x = (TStudent *)a; //x - pointer to the first student
     TStudent *y = (TStudent *)b; //y - pointer to the second
      return( strcmp( x->family, y->family ) ); //compare by family

     /*
       Если хотите сравнить по году рождения
       if( x->year < y->year ) return -1;
           else if( x->year == y->year ) return 0;
                    else return 1;
     */
 }

int main(void)
{
  int  i;

  //Ввод данных
  const int StudCount = 4;
  TStudent students[StudCount] = {
        {"Alexander", "Farberov", 1987},
        {"Dmitry", "Hrabrov", 1988},
        {"Denis", "Shulga", 1988},
        {"Boris", "Abramovich", 2666}
  };

  //Сортировка
  qsort( (void *)students, StudCount, sizeof(TStudent), sort_cmp );

 //Вывод
  puts("The students are:");
  for (i = 0; i < StudCount; i++)
    printf("%d. %s %s %d\n", i+1,
        students[i].family, students[i].name, students[i].year);
  return 0;
}
```

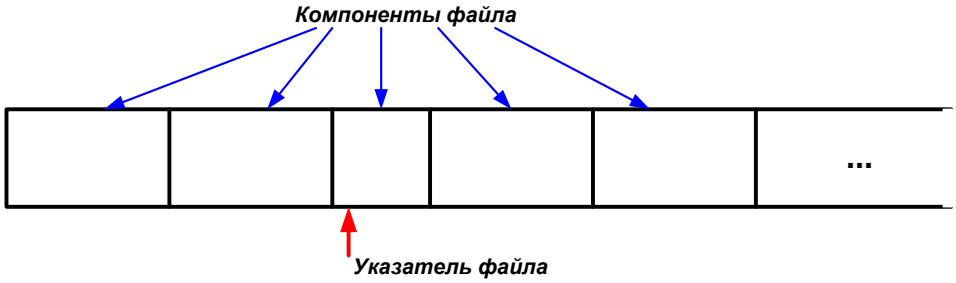
15 ФАЙЛЫ В ЯЗЫКЕ С

15.1. Общие понятия

Под файлом понимается поименованная область внешней памяти ПК (жесткого диска, гибкой дискеты, электронного "виртуального диска", …), хранящая данные. Под файлом понимается также логическое устройство – потенциальный источник или приемник информации.

Любой файл имеет следующие характеристики (атрибуты):

- ***Имя файла***. Составляется по правилам составления идентификаторов в рассматриваемой ОС, например, ***C:\MCDOC\d.txt***).
- ***Тип компонентов***. Например, файл может представлять собой последовательность строк или последовательность байтов.
- ***Длина файла***. Это число компонент файла.
- ***Указатель файла***. Это переменная специального типа, предназначенная для указания на компонент (позицию) файла. Значение указателя файла изменяется после каждого выполнения операции чтения или записи данных.



Логические устройства.

Стандартные аппаратные устройства ПК, такие как клавиатура, экран дисплея, печатающее устройство (принтер) и коммуникационные каналы ввода-вывода, определяются специальными именами, которые называются ***логическими устройствами***:

- ***con*** – логическое имя, которое определяет консоль (клавиатуру или экран дисплея);
- ***prn*** – логическое имя принтера. Если к ПК подключено несколько принтеров, доступ к ним осуществляется по логическим именам ***LPT1***, ***LPT2***, ***LPT3***. Первоначально ***prn*** и ***LPT1***– синонимы.
- ***aux*** – логическое имя коммуникационного канала, который используется для связи ПК с двумя машинами. Коммуникационный канал может осуществлять передачу и прием данных. Как правило,

имеется 2 коммуникационных канала: ***com1*** и ***com2***. Первоначально ***aux*** и ***com1*** – синонимы.

- ***NUL –*** логическое имя "пустого" устройства. Чаще всего используется в отладочном режиме как устройство-приемник информации неограниченной емкости. При обращении к ***NUL*** как к источнику информации, выдается признак конца файла (***EOF***).

Понятие потока.

При вводе/выводе данные рассматриваются как поток байтов.

Поток – это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику, т.о., физически ***поток*** в С– это файл или устройство.

Чтение данных из потока называется ***извлечением***, вывод в поток – ***помещением*** или ***включением.***

Обмен с потоками для увеличения скорости передачи данных производится, как правило, через специальную область ОП – ***буфер***. При выводе фактическая передача данных происходит после заполнения буфера. При вводе фактическая передача данных происходит, если буфер исчерпан.

По направлению обмена потоки можно разделить на ***входные*** (данные вводятся в память), ***выходные*** (данные выводятся из памяти) и ***двунаправленные*** (происходит как извлечение, так и включение).

По виду устройств, с которыми работает поток, можно разделит потоки на ***стандартные***, ***файловые*** и ***строковые.***

Стандартные потоки предназначены для передачи данных от клавиатуры в память ПК, или из памяти ПК на экран дисплея и принтер.

Файловые потоки предназначены для обмена информацией с файлами на внешних носителях данных.

Строковые потоки – для работы с массивами символов в ОП.

По структуре данных поток может быть ***текстовым*** или ***бинарным*** (двоичным).

Текстовый поток – это последовательность строк, каждая из которых имеет нуль или более ASCII-символов и заканчивается символом "\n" – конца строки и перехода к следующей строке. Текстовые файлы могут быть просмотрены и отредактированы с клавиатуры любым текстовым редактором.

Бинарный (***двоичный***) поток - это последовательность байтов без учета разделения на строки. Каждая программа для своих бинарных файлов определяет собственную структуру.

Стандартные библиотеки для работы с потоками

Для работы с потоками существуют стандартные библиотеки ***stdio***, ***string***, ***stdlib***, ***io***, которые становятся доступными из функций пользователя после использования директивы препроцессора ***#include***, например,

#include <stdio.h>

Вызов подсказки для получения списка функций библиотеки:

1) В среде С выполнить команду HELP→Contents и нажать Enter;
2) Выбрать раздел Header Files и нажать Enter;
3) Выбрать название библиотеки и нажать Enter;
4) Выбрать функцию.

15.2. Работа с файлами (потоками)

15.2.1 Этапы работы с файлами (потоками)

Файл в программе на языке С – это переменная–указатель на тип ***FILE***, называемая ***файловой переменной***.
Работа с файлами состоит их трех этапов:

1. Открытие файла;
2. Обработка файла;
3. Закрытие файла.

Открытие файла (потока)

Поток можно открыть для **чтения**, **записи**, и для ***чтения и записи*** с помощью стандартной функции ***fopen,*** прототип которой имеет вид:

FILE *fopen (const char *filename, const char*mode);

Здесь первый параметр функции (***filename***) – имя открываемого файла в виде строки символов, второй (***mode***) - режим открытия файла (тип доступа к файлу)– строка из одного и более символов:

"***r***" – файл открывается для чтения;

"***w***" – открывается пустой файл для записи (если файл существует, то его содержимое теряется);

"***a***" – файл открывается для добавления информации в конец файла; если файла нет, он создается;

"***r+***" – файл открывается для чтения и записи (файл должен существовать);

"***w+***" – открывается пустой файл для записи и чтения (если файл существует, то его содержимое теряется);

"***a+***" – файл открывается для чтения и добавления данных в конец файла.

"***t***" – открытие файла в текстовом режиме;

"***b***" – открытие файла в двоичном режиме.

Примечание.

По умолчанию (когда в режиме открытия файла отсутствует *t* или ***b***) файл открывается в ***текстовом режиме***.

Возможны следующие режимы доступа: ***"w+b", "wb", "rw+", "w+t", "rt+"*** и др.

При успешном открытии потока функция ***fopen*** возвращает указатель на структуру типа ***FILE***. Эта структура связана с физическим файлом и содержит всю необходимую информацию для работы с ним (указатель на текущую позицию в файле, тип доступа и др.).

Возвращаемое функцией значение нужно сохранить и использовать для ссылки на открытый файл.

Если произошла ошибка при открытии файла, то возвращается **NULL**.

Т.о., для работы с функцией ***fopen*** в программе на языке *C* необходимо:

1. Объявить переменную-указатель на структуру типа ***FILE***, например:

FILE* f;

Эта переменная (***f***) называется файловой переменной или файлом;

2. Файловой переменной присвоить значение, возвращаемое функцией ***fopen***, например:

f=fopen("c:\mdoc\d.txt", "r+");

Примечание. Описание и инициализация файловой переменной (например, ***f***) может быть осуществлено в одном операторе, например:

FILE*f = fopen ("c:\mdoc\d.txt","r+");

После открытия файла все действия над файловой переменной будут отождествляться с действиями над внешним файлом (физическим файлом), т.е. с файловой переменной связывается определенный физический поток. При этом структуре типа ***FILE*** выделяется область **ОП**, адрес которой возвращает функция ***fopen***. Поток связывается со структурой типа ***FILE***.

Структура типа ***FILE*** используется функциями ввода/вывода для хранения информации, связанной с устройством или файлом. Указатель на тип ***FILE*** используется для операций с файлом с помощью библиотечных

функций. Его передают библиотечным функциям в качестве параметра. Такие библиотечные функции называются функциями ввода/вывода.

Примечания.

1. При открытии потока с ним связывается область памяти – буфер.
2. При аварийном завершении программы выходной буфер может быть не выгружен, поэтому возможна потеря данных.
3. С помощью функций ***setbuf*** и ***setvbuf*** можно управлять размерами и наличием буферов.
4. Перед началом работы программы операционная система автоматически открывает пять стандартных потоков:
   1) ***stdin*** – стандартный ввод;
   2) ***stdout*** – стандартный вывод;
   3) ***stdprn*** – стандартная печать;
   4) ***stderr*** – стандартный вывод сообщений об ошибках;
   5) ***stdaux*** – стандартный дополнительный поток (стандартные порты).

Потоки 1), 2), 4) относятся к консоли (***con***), 3) – к ***prn***, 5) – к ***aux*** (коммуникационный канал), т.е., в начале работы программы автоматически открываются потоки следующими операторами:

1) ***FILE *stdin=fopen***("***con***", "***r***");
2) ***FILE *stdout=fopen***("***con***", "***w***");
3) ***FILE *stdprn=fopen("prn", "w");***
4) ***FILE *stderr=fopen***("***con***", "***w***");
5) ***FILE *stdaux=fopen***("***aux***", "***wb***");

Файлы ***stdin*** и ***stdout*** можно переназначать при запуске exe-программы или в программе с помощью функции ***freopen***. Ее прототип:

File ****freopen***(***const char *filename***, ***const char *mode***, ***FILE *stream***);

где ***freopen*** – имя функции ;
filename – имя файла ;
mode – режим работы файла;
stream - поток.
Функция ***freopen*** работает аналогично функции ***fopen***, но предварительно закрывает поток ***stream***, если тот был ранее открыт.

Закрытие файла (потока)

Для закрытия потоков, используемых в программе, применяются стандартные функции ***fclose*** и ***fcloseall.***

Один заданный поток закрывается функцией ***fclose***. Ее прототип:

int fclose (***FILE *stream***);

где ***stream –*** поток**.**

Пример обращения к функции ***fclose***:

fclose (***f***);

Если поток успешно закрыт, функция ***fclose*** возвращает значение 0. Если при закрытии потока произошла ошибка – значение ***EOF*** (-1).

Функция ***fcloseall*** закрывает все потоки, открытые с помощью ***fopen***, кроме ***stdin***, ***stdout***, ***stderr***. Прототип функции ***fcloseall***:

int fcloseall();

Функция ***fcloseall*** возвращает количество закрытых потоков или значение ***EOF*** (-1), если при закрытии произошла ошибка.

Примечание. Функция ***perror*** позволяет вывести сообщение об ошибке при неуспешном закрытии файла. Ее прототип:

void perror(***const char *s***);

Пример.

```
FILE * fp;
fp=fopen("d.dat", "r");
if(!fp) perror("Нельзя открыть файл для чтения");
   else fclose ("d.dat");
```

Удаление файла

Удалить файл можно с помощью функции ***remove***. Ее прототип:

int remove (***const char *filename***);

где ***filename*** – указатель на строку с именем физического файла (имя файла).

Функция ***remove*** возвращает значение 0 при успешном удалении файла и не нуль в противном случае. Открытый файл необходимо предварительно закрыть.

Пример открытия и закрытия файла.

В текстовый файл с именем ***rez.txt*** записываются результаты выполнения программы в начало нового файла, если файл не существует, или в конец существующего файла.

```
#include <stdio.h>
void main ()
{
  int n;  //номер теста
  float y,x;
  FILE *f;          //Описание файловой переменной f (файла)
  puts(“Введите x”);
  scanf(“%f”, &x);
  y=2*x;
  f=fopen("rez.txt", "a");  //Открытие файла
  puts("Номер теста ?");
```

```
  scanf("%d",&n);
  fprintf(f,"Тест N %d\n",n);  //Запись в файл f значения
  fprintf(f,"%f  %f\n",x,y);
  fclose(f);                   //Закрытие файла
 fflush(stdin);  getchar();
}
```

В результате первого выполнения программы в текущем каталоге будет создан файл с именем ***rez.txt***. В его начало запишутся значения ***n, x, y.*** При последующих запусках программы на выполнение значения ***n, x, y*** будут записываться в конец файла ***rez.txt***, на что указывает режим "***a***" открытия файла в функции fopen. Отсутствие символа "***t***" или "***b***" в режиме открытия файла "говорит" о работе с текстовым файлом (но не расширение ***txt*** в имени файла).

15.3 Ввод/вывод в поток

15.3.1 Основные понятия

Ввод/вывод в поток можно осуществить разными способами:

- в виде последовательности байтов;
- в виде символов и строк;
- с использованием форматных преобразований.

Операции ввода/вывода выполняются, начиная с текущей позиции потока. Текущая позиция определяется положением ***указателя потока***. Указатель потока устанавливается на начало или конец файла при открытии в соответствии с режимом открытия. После каждой операции ввода/вывода указатель потока автоматически изменяется.

15.3.2 Позиционирование в файле

15.3.2.1 Функции получения текущего положения указателя потока

ftell и ***fgetpos***

Прототип функции ***ftell*** (из <stdio.h>):

long int ftell(File *f);

Функция ***ftell*** возвращает текущую позицию в файле, связанном с потоком ***f***, как длинное целое. В случае ошибки возвращает число (-1).

Прототип функции ***fgetpos*** (из <stdio.h>):

long int fgetpos (File *f, fpos_t *pos);

Функция ***fgetpos*** возвращает текущую позицию в файле, связанном с потоком ***f***, и копирует значение текущей позиции по адресу ***pos***. Это

значение позднее может использоваться функцией ***fsetpos***. Возвращаемое значение имеет тип ***fpos_t***, который используется функциями ***fgetpos*** и ***fsetpos*** для хранения текущей позиции файла:

typedef long **fpos_t***;*

15.3.2.2 Функции задания положения указателя ***fseek*** и ***fsetpos, rewind***

Прототип функции ***fseek*** :

int fseek (File *f, long off, int org);

Функция перемещает текущую позицию в файле, связанном с потоком ***f***, на позицию ***off***, отсчитываемую от значения ***org***, которое должно быть равно одной из констант, определенных в <***stdio.h***>:

SEEK_CUR – от текущей позиции указателя (1);

SEEK_END – от конца файла (2);

SEEK_SET - от начала файла (0).

Параметр ***off*** задает количество байтов, на которое необходимо сместить указатель соответственно параметру ***org***. Величина смещения может быть как положительной, так и отрицательной, но нельзя сместиться за пределы начала файла.

Такой доступ к данным в файле называется произвольным.

Прототип функции ***fsetpos*** :

int fsetpos (File *f, const fpos_t *pos);

Функция перемещает текущую позицию в файле, связанном с потоком ***f***, на позицию *pos, предварительно полученную с помощью функции ***fgetpos***.

Примечание. Функции ***fseek*** и ***fsetpos*** нельзя использовать для стандартных потоков.

Функция ***rewind*** очищает флаги ошибок при работе с потоками и устанавливает указатель на начало файла. Ее прототип:

void rewind(File *f);

15.3.3 Функции чтения и записи потока байтов ***fread*** и ***fwrite***

Прототип функции ***fread***:

size_t fread (***void *buffer***, ***size_t size***, ***size_t count***, ***FILE *stream***);

Функция ***fread*** считывает ***count*** элементов длиной ***size*** байтов в область, заданную указателем ***buffer***, из потока ***stream***. Возвращает количество прочитанных элементов, которое может быть меньше ***count***, если при чтении произошла ошибка или встретился конец файла.

Прототип функции ***fwrite***:

size_t fwrite (const ***void *p***, ***size_t size***, ***size_t n***, ***FILE *f***);

Функция записывает ***n*** элементов длиной ***size*** байтов из буфера, заданного указателем ***p***, в поток ***f***. Возвращает число записанных элементов.

15.3.4 Функции чтения символа из потока (*getc, fgetc, getchar*)

Функция ***getc***, имеющая прототип:
int getc (FILE *f);

возвращает очередной символ в формате ***int*** из стандартного потока ***f***. Если символ не может быть прочитан, то возвращает значение ***EOF***.

Функция ***fgetc***, имеющая прототип:

int fgetc (FILE *f);

возвращает очередной символ в формате ***int*** из потока ***f***. Если символ не может быть прочитан, то возвращает значение ***EOF***.

Функция ***getchar***, имеющая прототип:

int getchar (void);

возвращает очередной символ в формате ***int*** из стандартного потока (*stdin*). Если символ не может быть прочитан, то возвращает значение ***EOF***.

15.3.5 Функции записи символа в поток (***putc***, ***fputc***, ***putchar***)

Функция ***putc*** записывает символ ***ch*** в поток ***f***. При ошибке возвращает значение ***EOF***. Ее прототип:
int putc (***int ch***, ***FILE *f***);

Выполняется ***fputc*** аналогично функции ***putc***. Ее прототип:

int fputc (***int ch***, ***FILE *f***);

Функция ***putchar()*** выводит символ ***ch*** на стандартное устройство вывода, добавляя в конце символ новой строки. Возвращает неотрицательное значение при успехе или EOF – при ошибке.

15.3.6 Функции чтения строки из потока (***fgets***, ***gets***)

Прототип функции ***fgets***:

char *fgets (***char *s***, ***int n***, ***FILE *f***);

Функция читает не более ***n-1*** байт из потока ***f*** и помещает прочитанные байты в строку ***s***, прекращая чтение при обнаружении символа новой строки или конца файла. Символ новой (**'\n'**) строки при чтении не отбрасывается, помещается в конец строки ***s***. Прочитанная строка дополняется ограничителем строки (**'\0'**). При обнаружении ошибки или конца файла возвращает ***NULL***, в противном случае – указатель на строку ***s***.

Функция ***gets*** читает символы с клавиатуры до появления символа новой строки и помещает их в строку ***s***. Сам символ новой строки в строку не включается. Возвращает указатель на ***char*** . Прототип функции ***gets***:

char *gets (***char *s***);

15.3.7 Функции записи строки в поток (***fputs***, ***puts***)

Прототип функции ***fputs***:

int fputs (***const char * s***, ***FILE *f***);

Функция ***fputs*** записывает строку символов ***s*** в поток ***f***. Символ конца строки в файл не записывается. При ошибке возвращает значение ***EOF***, иначе – неотрицательное число.

Функция ***puts*** выводит строку ***s*** на стандартное устройство вывода, добавляя в конце символ новой строки. Возвращает неотрицательное значение при успехе или ***EOF*** – при ошибке. Прототип функции ***puts***:
int puts (***char *s***);

15.3.8 Функции форматированного ввода (чтения) из потока (***fscanf*** , ***scanf***, ***sscanf***)

Прототип функции ***fscanf***:

int fscanf (***FILE *f***, ***const char *fmt*** [, ***par1***, ***par2***, …]);

Эта функция вводит (читает) строку параметров ***par1***, ***par2***, … в формате, определенном строкой ***fmt***, из потока (файла) ***f***. Возвращает число переменных, которым присвоено значение.

Прототип функции ***scanf***:

int scanf (***const char *fmt*** [, ***par1***, ***par2***, …]);

Функция ***scanf*** вводит (читает) строку параметров ***par1***, ***par2***, … в формате, определенном строкой ***fmt***, со стандартного устройства ввода (обычно с клавиатуры (***stdin***)). Возвращает число переменных, которым присвоено значение.

Прототип функции ***sscanf***:

int sscanf (***const char *buf***, ***const char *fmt*** [, ***par1***, ***par2***, …]);

Функция ***sscanf*** вводит данные аналогично функции ***scanf***, но не с клавиатуры, а из строки символов, переданной ей первым параметром. Аргумент ***buf*** – строка символов, из которой вводятся значения, ***fmt*** - строка формата, в соответствии с которой происходит преобразование данных. Многоточие указывает на наличие необязательных аргументов, соответствующих адресам вводимых значений.

15.3.9 Функции форматированного вывода в поток (***fprintf***, ***printf***, ***sprintf***)

Прототип функции ***fprintf***:

int fprintf (***FILE *f***, ***const char *fmt***, **…**);

Функция записывает в поток ***f*** значения переменных, список которых обозначен многоточием (**…**), в формате, указанном строкой ***fmt***. Возвращает число записанных значений.

Прототип функции ***printf***:

int printf (***const char *fmt***, **…**);

Функция выводит на стандартное устройство вывода (***stdout***) значения переменных, перечисленных в списке, обозначенном многоточием (**…**), в формате, определенном строкой ***fmt***. Возвращает число выведенных значений.

Прототип функции ***sprintf***:

int sprintf (***char *buf***, ***const char *fmt***, **…**);

Функция выводит в строку ***buf*** значения переменных, перечисленных в списке, обозначенном многоточием (**…**), в формате, определенном строкой ***fmt***.
Примечание. Спецификации формата ***fmt*** были рассмотрены в разделе "Консольный ввод-вывод".

15.4 Обработка ошибок

Функции работы с потоками возвращают значения, которые рекомендуется анализировать в программе и обрабатывать ошибочные ситуации, возникающие, например, при открытии файлов или чтении из потока. При работе с файлами часто используют функции ***feof*** и ***ferror***.

Прототип функции ***feof***:
int feof (***FILE *f***);

Функция возвращает ***EOF*** или значение, отличное от 0, если достигается конец файла; в противном случае 0.

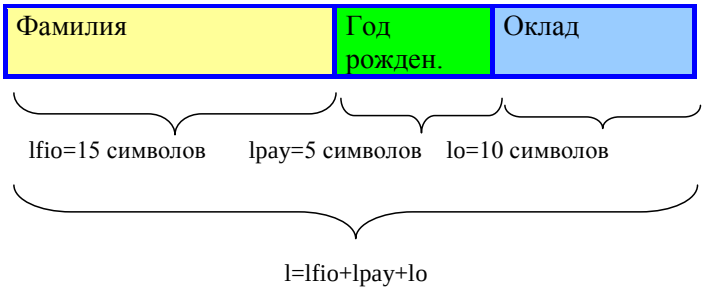
Прототип функции ***ferror***:
int ferror (***FILE *f***);

Возвращает не равное нулю целое значение, означающее код ошибки, если обнаружена ошибка ввода/вывода; 0 – в противном случае.

15.5 Пример обработки текстового файла

В текстовом файле "dbase.txt" хранятся данные о сотрудниках фирмы. В каждой строке файла указана фамилия, год рождения, оклад сотрудника. Для простоты обработки файла данные записаны единообразно: первые 15 символов занимает фамилия, следующие 5 – год рождения, последние 10 – оклад.
Требуется, интерпретируя структурами строки файла, вывести на экран или принтер содержимое файла в виде таблицы, создать из строк файла с данными о сотрудниках фирмы, моложе 20 лет, массив структур. Вывести на

экран или в текстовый файл полученный массив или вывести сообщение о том, что такие молодые сотрудники не работают в фирме.
Структура записи файла изображена на рисунке.



Записи файла могут иметь вид:

Иванов И.П. **1982 453120**
Авчинникова **1999 578320**
Васильков **1955 456780**
Чернов **1967 1345600**

Требуется на экран, на принтер или в текстовый файл вывести список в форме таблицы. Например:

СПИСОК СОТРУДНИКОВ ФИРМЫ

Фамилия	Год рожд.	Оклад
Иванов И.П.	**1982**	**453120**
Авчинникова	**1999**	**578320**
Васильков	**1955**	**456780**
Чернов	**1967**	**1345600**

```
/*   Поиск в массиве структур, читаемых из текстового файла */
#include <stdio.h>    // с записью рез-тов в новый текстовый файл
#include <string.h>
#include <stdlib.h>
// #include <windows.h>
main()
{
  const int lfio=15,            //длина поля фио,
      lpay=5,                   // длина поля г.рожд
      lo=10,                    //длина поля оклада
      l=lfio+lpay+lo;           //длина записи в файле
  struct Man
      {  char   fio[lfio];      // фио
```

```
        int    year;          // год рожд.
        float  pay;           // оклад
    };
 Man db;
 char  s[l];                  //строка для записи файла
 FILE *fin,                   // Исх. файл
     *fo,*f1 ;                // Вых. Файлы
 fin =fopen("dbase.txt", "r");
 if  (fin==NULL)
        puts("Ошибка открытия файла\n");
 else
{
 puts("Файл открыт");
 fo=fopen("dbout.txt","w");
 f1=fopen("dbout1.txt","w");
 while (!feof(fin))
  { fgets(s,l,fin);   puts(s);
    strncpy(db.fio,s,lfio-1);
    db.fio[lfio-1]='\0';
    db.year=atoi(&s[15]);
    db.pay=atof(&s[20]);
    if(db.year>2000)
      fprintf(fo,"%-15s %10.1f\n", db.fio, db.pay);
    fputs(s,f1);
  }
 fclose(fin); fclose(fo); fclose(f1);//
}
 fflush(stdin); getchar();
 return 0;
}
```

15.6 Пример обработки текстового и бинарного файла

/* Построчное считывание данных из текстового файла "***dbase.txt***" в буферную переменную ***s***, формирование из них структуры ***db*** и запись ее в двоичном режиме в выходной файл "***dbout.dat***" .
Считывание из двоичного файла записи с номером i и вывод ее на экран.
Считывание из двоичного файла записей и вывод на экран только тех записей, для которых фамилия есть "***ivanoff*** ". */

Отформатировано: Русский (Россия)

Отформатировано: Русский (Россия)

Таблица 15.1 – Таблица соответствия

№	Идентификатор	Тип	Комментарий
1	lfio	const int	Длина поля фио

2	lyear	const int	Длина поля г. рожд.
3	lo	const int	Длина поля оклада
4	l	const int	Длина записи
5	db	Запись	Запись
6	s	строка	Строка с содерж. записи
7	fin	Текстовый файл	Исходный текстовый файл
8	fo	Двоичный файл	Двоичный файл, получ. из fin
9	kol	int	Кол-во записей файла fin
10	i	int	Номер записи файла fin

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream.h>
main()
{
 const int lfio=15, lpay=5, lo=10,//длины полей: фио, г.рожд., оклада в т. файле
 l=lfio+lpay+lo;    //длина записи в т. файле
 struct Man
     {    char   fio[lfio];    // фио
          int    year;         // год рожд.
          float  pay;          // оклад
     };
 Man db;
 char  s[l];         //строка для записи в файл
 FILE *fin,          // Исх. файл
       *fo ;       // Вых. файлы
 if ((fin=fopen("dbase.txt", "r"))==NULL)
         {
          puts("Ошибка открытия файла\n");
          fflush(stdin); getchar();  return (1);
         }
 puts("Файл открыт");
 fo=fopen("dbout.dat","w+b");
 int kol=0;  //кол. зап. в текст. файле
 while (!feof(fin))    // пока не конец файла
  {
   fgets(s,l,fin);      // читается строка
   puts(s);
   strncpy(db.fio,s,lfio-1);    //из строки формир. структура db
   db.fio[lfio-1]='\0';
   db.year=atoi(&s[lfio]);
```

```
    db.pay=atof(&s[lfio+lpay]);
    fwrite(&db, sizeof (db) ,1, fo);     // структура зап. в бинарный файл
    kol++;
   }
  fclose(fin);
  // чтение зап. с ном. i из бин. файла и вывод ее на экран
  int i;
  printf("Введите номер записи (0-%d)", kol-1);
  cin>>i;
  if(i>=kol || i<0) { cout<<"Запись не существует"; fclose(fo); return (1);}
  fseek(fo, sizeof (db)*i, SEEK_SET);   //установ. указ. на зап. с ном. i
  fread(&db, sizeof(db), 1, fo);        //чтение зап. из файла
  cout << db.fio << " "<< db.year<<" " << db.pay; //вывод зап. на экран

// Вывод записей с фам. ivanoff
  fseek(fo,0,SEEK_SET);  //указ. на нач. файла
  for(i=0; i<kol;i++)
    {  printf("\n");
       fread(&db,sizeof (db),1,fo);
       if(!strcmp (    db.fio," ivanoff      "))
              printf("% -15s % 5d %/.0f \n",  db.fio, db.year, db.pay);
    }
  fclose(fo);
  fflush(stdin); getchar();
  return (0);
}
```

16 ПОРОЗРЯДНЫЕ ЛОГИЧЕСКИЕ ОПЕРАЦИИ

16.1 Размещение данных в памяти ПЭВМ

Данные и программы во время работы ПЭВМ размещаются в оперативной памяти, которая представляет собой последовательность пронумерованных ячеек. По указанному номеру процессор находит нужную ячейку, поэтому ***номер ячейки называется ее адресом***. Минимальная адресованная ячейка состоит из 8 двоичных позиций, т.е. в каждую позицию может быть записан 0 или 1.

Для удобства работы введены следующие термины, обозначающие совокупности двоичных разрядов. Эти термины обычно используются в качестве единиц измерения объемов информации, хранимой или обрабатываемой в ЭВМ.

Объем информации, который помещается в одну двоичную позицию, называется ***битом***.

Объем информации, равный 8 битам, называется ***байтом***.

В одной ячейке из 8 двоичных разрядов помещается объем информации в 1 байт, поэтому объем памяти принято оценивать количеством байт:

2^{10} байт = 1024 байт = 1 Кб
2^{10} Кб = 1048576 байт = 1 Мб
2^{10} Мб=1Гб
2^{10} Гб=1Тб.

При размещении данных производится их запись с помощью нулей и единиц - ***кодирование***, при котором каждый символ заменяется последовательностью из 8 двоичных разрядов в соответствии со стандартной кодовой таблицей (ASCII). Например символ *D* (код - 68) имеет двоичный код 01000100; символ *F* (код - 70) имеет двоичный код 00100110; символ 4 (код - 52) имеет двоичный код 00110100.

При кодировании числа (коды) преобразуются в двоичное представление, например,

$2 = 1*2^1 + 0*2^0 = 10_2$;
$5 = 1*2^2 + 0*2^1 + 1*2^0 = 101_2$;
$256 = 1*2^8 = 100000000_2$.

С увеличением числа количество разрядов для его представления в двоичной системе резко возрастает, поэтому для размещения большого числа выделяется несколько подряд расположенных байт. В этом случае ***адресом ячейки является адрес первого байта***, один бит которого выделяется под знак числа.

Последовательность нескольких битов или байтов часто называют ***полем*** данных. ***Биты в числе (в слове, в поле и т.п.) нумеруются справа налево, начиная с 0-го разряда.***

В ПК могут обрабатываться поля постоянной и переменной длины.

Поля постоянной длины:

слово – 2 байта;

полуслово – 1 байт;
слово длиной 10 байт – 10 байт;
двойное слово – 4 байта;
расширенное слово – 8 байт.

Числа с фиксированной запятой чаще всего имеют формат слова и полуслова, числа с плавающей запятой – формат двойного и расширенного слова.

Поля переменной длины могут иметь любой размер от 0 до 256 байт, но обязательно равный целому числу байтов.

16.2 Побитовые операции

Побитовые или поразрядные логические операции предназначены для обработки значений на машинном (битовом или двоичном) уровне их представления и позволяют манипулировать индивидуальным битом в интегрированном примитивном типе данных (данным целого типа). Побитовые операции выполняются по законам Булевой алгебры с битами как с самостоятельными величинами. Обычно побитовые операции используются в программах, реализующих доступ к аппаратуре, для логической обработки битовых данных, требующих операций над битами: их выделения из значения, анализа, замены, сдвига и т.п. В таблице приведены побитовые операции в порядке убывания их приоритета. Чтобы правильно использовать эти операции, надо знать машинное (битовое) представление обрабатываемых значений.

Побитовые операции языка С

Операции	Результат	Оператор	Результат
~	побитовое унарное отрицание (обращение) (NOT)		
>>	сдвиг вправо	>> =	сдвиг вправо с присваиванием
<<	сдвиг влево	<<=	сдвиг влево с присваиванием
&	побитовое И (AND) – поразрядное логическое умножение	&=	побитовое И (AND) с присваиванием
^	побитовое исключающее ИЛИ (XOR) – сложение разрядов по модулю 2	^=	побитовое исключающее ИЛИ (XOR) с присваиванием
\| | побитовое ИЛИ (OR) | \|= | побитовое ИЛИ (OR) с присваиванием |

Побитовые операции выполняются только над скалярными типами данных т.е над данными, принимающими только целочисленные значения: char, short int, int, long int, а также их signed и unsigned модификации.
Побитовые операции можно комбинировать со знаком = для соединения побитовой операции и операции присваивания: **&=**, |= и ^= являются допустимыми. Так как ~ - это унарная операция, то она не может комбинироваться со знаком =.
Результаты побитовых операций **&**, ^, |, ~ идентичны результатам соответствующих им логических операций. Результаты побитовых операций **&**, ^, |, ~ приведены в таблице истинности.

Таблица 16.1 – Таблица истинности побитовых операций **&**, ^, |, ~

A	B	A**&**B	A^B	A|B	~ A
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	0	1	0

Побитовые операции &, ^, | сравнивают каждый бит своего первого операнда с соответствующим битом второго операнда и в зависимости от их комбинации устанавливают соответствующий бит результата в 0 или 1.
Операция ~ изменяет каждый бит своего операнда на противоположное значение: 0 – на 1, 1 – на 0.
Примеры:

1) ~

```
int a=2;    //0000…0010 – двоичное представление числа 2
int r=~a;   //1111…1101  – результат ~
```

2) &

```
int a=10;      //0000…01010
int b=12;      //0000…01100
int r=a&b;    //0000…01000 – число 8
```

3) ^

```
int a=10;     //0000…01010
int b= 12;    //0000…01100
int r=a^b;    //0000…00110 – число 6
```

4) |

```
int a=10;     //0000…01010
int b= 12;    //0000…01100
int r=a|b;      //0000…01110 – число 14
```

Операция << объединяет два операнда (например, a<<b) и выполняет сдвиг влево всех битов своего левого операнда (операнд a) на число позиций,

заданное правым операндом (операнд b). При этом часть битов в левых разрядах операнда a выходит за границы и теряется, а соответствующие правые позиции заполняются нулями.
Операция >> также объединяет два операнда (a>>b) и означает сдвиг вправо. Она перемещает все биты своего левого операнда вправо на число позиций, заданное правым операндом. Когда биты левого операнда выдвигаются за самую правую позицию слова (16 бит), они теряются. При сдвиге вправо освобождающиеся старшие (левые) разряды сдвигаемого числа заполняются предыдущим содержимым знакового разряда. Такое поведение называют ***расширением знакового разряда***.
Результат операции сдвига не определен, если второй операнд отрицательный. Результат операции теряется, если результат сдвига не может быть представлен типом первого операнда после преобразования.

Пример сдвига влево.

0000000010001010_2 << 2 = 0000001000101000_2 (<< 2 эквивалентно умножению на 4).

Покажем, что сдвиг влево на два бита эквивалентен умножению на 4:
$0000000010001010_2=2^7+2^3+2^1 = 128+8+2 = 138_{10}$;
$0000001000101000_2=2^9+2^5+2^3 =512+32+8=552_{10}$.
138*4=552.

Пример сдвига вправо.

0000000010001010_2 >> 2 = 0000000000100010_2 (>> 2 эквивалентно целочисленному делению на 4).
Покажем, что сдвиг вправо на два бита эквивалентен делению на 4:
$0000000010001010_2=2^7+2^3+2^1 = 128+8+2 = 138_{10}$;
$0000000000100010_2 =2^5+2^1=32+2=34_{10}$;
[138/4]=34.
Операции сдвига могут применяться для деления или умножения целого значения на число, равное степени 2:
x << n – эквивалентно умножению числа x на 2^n.
x >> n - эквивалентно целочисленному делению числа x на 2^n.
Например, чтобы разделить целое число x на 8 ($8=2^3$), можно записать оператор:
x=x>>3;

16.3 Примеры работы с битами

В приведенных ниже примерах x – целое число (2 байта): bitno – номер бита; maska – число, которое может быть записано в виде двоичного числа.

1. **Установка заданного бита или группы битов (**значение бита или группы битов сделать равным 1**):**
x|= (1 << bitno) или x |= maska //1
Строку //1 можно записать:
x=x|(1 << bitno) или x= x|maska
Например, надо в числе x установить 5-ый бит. Для этого запишем оператор **x=x|(1<<5);**
Покажем на числовом примере, что это верно;
$x=15_{10}$ $=0000000000001111_2$
1_{10} $=0000000000000001_2$
1<<5 $=0000000000100000_2$
x|(1<<5) $=0000000000101111_2$
Например, в числе требуется установить каждый четный бит, тогда используем «маску» maska =0b0101010101010101;
Если x=0b0110010011110001; (серым цветом выделены биты, которые требуется установить), то после выполнения оператора x=x|maska; получим значение x, равное
0111010111110101
В самом деле:
0110010011110001
0101010101010101
После | получаем
0111010111110101
Здесь цветом выделены установленные биты. Остальные биты остались без изменения.
Если надо сформировать значение результата как совокупности всех двоичных единиц операндов, можно использовать операцию |. Например,
int x=10; //0000…1010_2
int y= 12; //0000…1100_2
int z=x|y; //0000…1110_2 – число 14_{10}
Если в качестве результата надо сохранить только те разряды, в которых только одна из слагаемых есть единица, надо применить операцию ^. Например,
int x=3; //0000…000011_2
int y= 26; //0000…011010_2
int z=x^y; // 0000…011001_2 – число 25

2. **Сброс (очистка) заданного бита или группы битов (**значение бита или группы битов сделать равным 0**):**
x &= ~(1 << bitno) или x &= ~maska //2
Строку //2 можно записать:
x =x &(~(1 << bitno)) или x =x&(~maska)

Например, надо в числе x сбросить 5-ый бит. Для этого запишем оператор **x=x&(~(1<<5));**
Покажем на числовом примере, что это верно;
x = 0000000011110011_2
1 = 0000000000000001_2
1<<5 = 0000000000100000_2
~(1<<5) = 1111111111011111_2
x&(~(1<<5)) = 0000000011010011_2
Например, в числе требуется сбросить биты 2-ой, 10-ый и 13-ый, тогда используем "маску"
maska =0b0010010000000100;
Если x= 0b0110010011110001; (серым цветом выделены биты, которые требуется сбросить), то после выполнения оператора
x =x&(~maska) ;
получим последовательно:
x = 0110010011110001_2
maska = 0010010000000100_2
~maska = 1101101111111011_2
x&(~maska) = 0100000011110001_2
В программе, т.о., надо записать два оператора:
maska =0b0011010000000100; // Задание маски в виде двоичного числа
x =x&(~maska) ;

3. **Инвертирование заданного бита**:
x ^= (1 << bitno) //3
Строку //3 можно записать:
x=x ^ (1 << bitno)
Например, требуется инвертировать 5-ой бит числа x. В программе запишем оператор
x=x ^ (1 <<5);
Покажем на числовом примере, что это так для x=0101010000001001_2.
1 = 0000000000000001_2
1<<5 = 0000000000100000_2
x = 0101010000001001_2.
x ^ (1 <<5) = 0101010000101001_2.

4. **Для выделения заданных битов используюя «маски», которые обнуляют ненужные биты**:
Дано число A. Выделить из числа 1, 3 10 и 12 биты:
A&0b0001010000001010.
Здесь маска записана в двоичной системе счисления, число A имеет тип int.
Таким образом, чтобы "замаскировать" или выделить для обработки заданные разряды (биты) двоичного числа, надо использовать в константе

выделения ("маске") значения: для выделения разрядов – единицы, а для "маскировки" разрядов – нули.

ЗАДАЧИ.

Даны два целых положительных числа A и B.:

1. Определить значение каждого бита числа A.
2. Обнулить все четные биты числа A.
3. Выполнить проверку заданного (вводится с клавиатуры) бита числа B и если заданный бит имеет значение «ЛОЖЬ», т.е. равен 0, установить его.
4. Найти $A\&B$, $A|B$, $A\hat{}B$, $\overline{A}$.
5. Рассчитать $A>>1$, $A>>2$, $A>>3$, $A>>8$, $B<<1$, $B<<2$, $B<<3$, $B<<8$ и показать, что результаты выполнения указанных операций совпадают с делением или умножением на степень двойки, соответствующий количеству сдвигаемых разрядов вправо или влево.

Переменные A и B должны иметь тип unsigned int.
Исходные данные и результат вывести в десятичной, шестнадцатеричной системах счисления и в виде двоичного числа.

Решение задачи 1

```
/* Определение каждого бита целого числа A */
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
{
 int A,  // Заданное целое число
    A1,  // Рабочее целое число
    B,   // Маска
    C,   // Значение бита числа
    i;   //параметр цикла

 puts(" Введите целое число");
 scanf("%d",&A);
 printf("  A=%d\n",A);
 B=1;  // в двоичной сист. 0000000000000001
 A1=A;
 puts(" Двоичное представление числа A:");
 for(i=1;i<=16;i++)
   {
    C=A1&B;// Определение i-1- го бита
    printf("  бит %2d равен  %d\n",i-1,C);
    A1=A1>>1;
```

```
  }
 printf("\n");
 getch();
 return(0);
}
```

Запустим программу на выполнение (BIT1.EXE)

Решение задачи 2

```
/* Обнуление четных битов целого числа A */
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
{
 int A,  // Заданное целое число
    A1,  // Рабочее челое число
    B,   // Маска
    C,C1,   // Значение бита числа
    i;   //параметр цикла
 int y1[16],y2[16];
 puts(" Введите целое число");
 scanf("%d",&A);
 printf("  A=%d\n",A);
 B=1;  // в двоичной сист. 0000000000000001
 A1=A;
 /* Двоичное представление исходного  A: */
 for(i=0;i<=15;i++)
   {
    C=A1&B;
    y1[i]=C;
    A1=A1>>1;
   }
 printf("\n");
 /* Обнуление четных битов числа A  */
 for(i=0;i<=15;i=i+2)
   {
    B=~(1<<i);
    A=A&B;
   }
 /* Двоичное представление изменного числа  A:*/
 B=1;A1=A;
 for(i=0;i<=15;i++)
   {
```

```
    C=A1&B;
    y2[i]=C;
    A1=A1>>1;
  }
 printf(" Исходное и измененное число A\n ");
 for(i=15;i>=0;i--)
  printf(" %d ",y1[i]);
 printf("\n ");

 for(i=15;i>=0;i--)
  printf(" %d ",y2[i]);
 printf("\n");
 getch();
 return(0);
}
```

Решение задачи 3

```
/* Найти A&B, A|B, A^B, ~A для целых A и B */
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
{
 int A,B,  // Заданные числа
             A1,  // Рабочее челое число
   C,C1,C2,C3,C4,   // Значение бита числа  и результаты
   i,    //параметр цикла
   k;   // номер бита
 int y1[16],y2[16],y[16];
 puts(" Введите два целых числа");
 scanf("%d%d",&A,&B);
 printf("  A=%d B=%d\n",A,B);
 getch();
 //1 -  в двоичной сист. 0000000000000001
 A1=A;
 /* Двоичное представление исходного  A: */
 for(i=0;i<=15;i++)
  {
    C=A1&1;
    y1[i]=C;
    A1=A1>>1;
  }
 printf("\n");
```

```
A1=B;
/* Двоичное представление исходного  B: */
for(i=0;i<=15;i++)
  {
    C=A1&1;
    y2[i]=C;
    A1=A1>>1;
  }
printf("\n");
C1=A&B;
/* Двоичное представление A&B:*/
A1=C1;
for(i=0;i<=15;i++)
  {
    C=A1&1;
    y[i]=C;
    A1=A1>>1;
  }
printf(" A:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y1[i]);
printf("\n ");
printf(" B:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y2[i]);
printf("\n");
printf(" A&B:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y[i]);
printf("\n");
 /* Двоичное представление A|B:*/
C2=A|B;
A1=C2;
for(i=0;i<=15;i++)
  {
    C=A1&1;
    y[i]=C;
    A1=A1>>1;
  }
printf(" A|B:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y[i]);
printf("\n");
/* Двоичное представление A^B:*/
```

```
 C3=A^B;
 A1=C3;
 for(i=0;i<=15;i++)
   {
     C=A1&1;
     y[i]=C;
     A1=A1>>1;
   }
 printf(" A^B:\n ");
 for(i=15;i>=0;i--)
   printf(" %d ",y[i]);
 printf("\n");
 /* Двоичное представление ~A:*/
 C4=~A;
 A1=C4;
 for(i=0;i<=15;i++)
   {
     C=A1&1;
     y[i]=C;
     A1=A1>>1;
   }
 printf(" ~A:\n ");
 for(i=15;i>=0;i--)
   printf(" %d ",y[i]);
 printf("\n");
 getch();
 return(0);
}
```

Решение задачи 4

```
/* Если в A установлены 1 2 5 8 биты в B установлены 3 4 9 биты в C 1 бит
то вычислить 2A&B/4+4C
Если в A установлены 6 9  биты в B установлены 13 15 биты в C 2 бит
то вычислить (A|B/2)C */
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
{
 unsigned int A,B,  // Заданные числа
              AR,BR,R,  // Рабочее челое число
              s,   //цифра числа
```

```
   A1,A2,A5,A8,B3,B4,B9,A6,A9,B13,B15;   // Значение бита числа  и
результаты
    char C,CR,C1,C2;
  int i,    //параметр цикла
   k;   // номер бита
 int y1[16],y2[16],y[16];
 puts(" Введите три целых числа");
 scanf("%d%d%d",&A,&B,&C);
 printf("  A=%d B=%d C=%d\n",A,B,C);
 getch();
 //1 -  в двоичной сист. 0000000000000001
 AR=A;
 /* Двоичное представление исходного  A: */
 for(i=0;i<=15;i++)
   {
     s=AR&1;
     y1[i]=s;
     AR=AR>>1;
   }
 printf("\n");
 BR=B;
 /* Двоичное представление исходного  B: */
 for(i=0;i<=15;i++)
   {
     s=BR&1;
     y2[i]=s;
     BR=BR>>1;
   }
 printf("\n");
 /* Двоичное представление C:*/
 CR=C;
 for(i=0;i<=15;i++)
   {
     s=CR&1;
     y[i]=s;
     CR=CR>>1;
   }
 printf(" A:\n ");
 for(i=15;i>=0;i--)
   printf(" %d ",y1[i]);
 printf("\n ");
 printf(" B:\n ");
 for(i=15;i>=0;i--)
   printf(" %d ",y2[i]);
```

```
printf("\n");
printf(" C:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y[i]);
printf("\n");

/* Определение значений битов 1 2 5 8 числа A
 битов 3 4 9 числа B и бита 1 числа C */
 AR=A;  A1=(AR>>1)&1;
 AR=A;  A2=(AR>>2)&1;
 AR=A;  A5=(AR>>5)&1;
 AR=A;  A8=(AR>>8)&1;
 BR=B;  B3=(BR>>3)&1;
 BR=B;  B4=(BR>>4)&1;
 BR=B;  B9=(BR>>9)&1;
 CR=C;  C1=(CR>>1)&1;
 if(A1&&A2&&A5&&A8&&B3&&B4&&B9&&C1)
   {
    puts("ДА1");
    AR=A; BR=B; CR=C;
    R=(AR<<1)&(BR>>2)+CR<<2;
    printf("R=%d\n",R);
   }
    /* Определение значений битов 6 9 числа A
 битов 13 15 числа B и бита 2 числа C */
  else
   {
    AR=A;  A6=(AR>>6)&1;
    AR=A;  A9=(AR>>9)&1;
    BR=B;  B13=(BR>>13)&1;
    BR=B;  B15=(BR>>15)&1;
    CR=C;  C2=(CR>>2)&1;
    if(A6&&A9&&B13&&B15&&C2)
     {
        puts("Да2");
        AR=A; BR=B; CR=R;
        R=(AR|(BR>>1))<<3;
        printf("R=%d\n",R);
     }
    else
     {
        puts("Очистка A, B, C");
        A=A&0; B=B&0;  C=C&0;
        printf("A=%d B=%d C=%d\n",A,B,C);
```

```
    }
  }
R=0z12;
printf("R=%z\n",R);

getch();
return(0);
}
```

Литература

1. Информатика. Базовый курс /Симонович С.В. и др.: Питер, 2001. – 640 с.
2. Касаткин А.И., Вальвачев А.Н. Профессиональное программирование на языке СИ: от Turbo C к Borland C++: Справ.пособие. – Мн.: Выш.шк., 1992. – 240 с.
3. Бруно Б. Просто и ясно. Borland C++: Пер. с англ.- М.: БИНОМ.- 400с.
4. Крячков А.В., Сухина И.В., Томшин В.К. Программирование на С и С++. Практикум: Учебн. Пособие для вузов. – М.: Горячая лининия – Телеком, 2000 – 344 с.
5. С/С++. Программирование на языке высокого урвня / Т.А. Павловская – СПБ.:Питер, 2002. – 464с.
6. Информатика: Учебник /Под ред. Проф. Н.В.Макаровой. –М.: Финансы и статистика, 1998.
7. Страуструп Б. Язык программирования Си++: Пер. с англ.- М.: Радио и связь, 1991. – 352 с.
8. Морис С. Объектно-ориентированное программирование. – Ростов-на-Дону: Феникс, 1997. – 952 с.
9. Топп У., Форд У. Структуры данных в С++: Пер. с англ.-М.: БИНОМ, 1994. – 816 с.
10. Хэнпок Л., Кригер М. Введение в программирование на языке Си: Пер.с англ.-М.: Радио и связь, 1986. – 192 с.
11. М/ук №1051 Мовшович С.М. к л/з по теме «Основы алгоритмизации». - Гомель: ГПИ, 1998. - 25 с.
12. М/ук № Основы алгоритмизации: практ. Пособие к лаб. и контрол. работам по курсу "Информатика" и "основы информатики и вычислительной техники" для студентов всех специальностей днев. И заоч. Отд-ний / авт.-сост.: Н.В. Водополова, В.И. Мисюткин, С.А. Чабуркина. – Гомель: ГГТУ им. П.О.Сухого,2005. – 32 с.
13. М/ук № 3089. Программирование ввода-вывода данных и линейных вычислительных алгоритмов на языке С: практ. пособие к выполнению лаб. и контрольных работ по дисциплине "Вычислительная техника и программирование" для студентов техн. специальностей дневн. и заочн. форм обучения /авт.-сост.: О.А.Кравченко, А.М. Мартыненко. – Гомель: ГГТУ им. П.О. Сухого, 2005. – 33 с.
14. Мовшович С.М. М/ук №1909 к л/з по теме «Методы сортировок» - Гомель: ГПИ, 1995. - 29 с.
15. М/ук № . Программирование разветвляющихся и циклических алгоритмов на языке С:
16. М/ук № Массивы в языке С. пособие по выполнению лабораторных и контрольных работ по дисциплине "Вычислительная техника и программирование"для студентов техн. специальностей дневн. и заочн. форм обучения /авт.-сост.: О.А.Кравченко, Д.А. Литвинов. - Гомель: ГГТУ им. П.О. Сухого, 2005. – 51 с.

17. Обработка символьной информации. Электронная версия МУК /авт.-сост.: О.А.Кравченко.
18. Записи. Электронная версия МУК /авт.-сост.: О.А.Кравченко.
19. Подпрограммы (функции). Электронная версия МУК /авт.-сост.: О.А.Кравченко.
20. Работа с файлами в языке С. Электронная версия МУК /авт.-сост.: О.А.Кравченко.