ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Рубцовский индустриальный институт
ГОУ ВПО «Алтайский государственный технический университет им. И.И. Ползунова»

Л.А. Попова, И.Б. Шульман

ИНФОРМАТИКА–ПРАКТИКУМ
ЧАСТЬ 1

Учебное пособие
для студентов технических специальностей 1-го курса

Рубцовск 2009

УДК 681.3.06
Попова, Л.А., Шульман, И.Б. Информатика–практикум. Часть 1: Учебное пособие для студентов технических специальностей 1-го курса / Рубцовский индустриальный институт. – Рубцовск, 2009. – 75 с.

Пособие содержит краткий справочный материал по каждой теме, примеры с пояснениями по их выполнению, задания для самостоятельной работы студентов. Часть 1 состоит из двух разделов: Основы работы на компьютере и Основы программирования. Пособие рекомендуется использовать при проведении лабораторных работах и выполнении домашних заданий.

Рассмотрено и одобрено
на заседании кафедры ПМ
Рубцовского индустриального
института
Протокол № 2 от 06.10.08.

Рецензент: к.ф.-м.н., доцент кафедры
естественно-научных дисциплин РИИ И.О. Кох

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данное учебное пособие предназначено для студентов первого курса технических специальностей. Содержит краткий справочный материал по каждой теме, примеры с пояснениями, задания для самостоятельных, лабораторных и домашних работ, а также задачи повышенной сложности.

Учебное пособие состоит из двух частей. Первая часть содержит разделы "Основы работы на компьютере" и "Основы программирования".

Задания для самостоятельной работы позволят углубить понимание рассматриваемых на занятиях тем, а также проявить творческую активность студентов. В комплект к методическому пособию входят программы на языке Паскаль, полностью или частично подготовленные для запуска на выполнение.

Данное пособие содержит практические задания и вопросы по теории, для решения которых требуется самостоятельная работа студентов с дополнительной литературой и другими учебными пособиями (так как пособие не содержит достаточное количество теоретического материала, а является дополнением к лекционным занятиям).

Цель практикума – научить студентов использовать технические и программные средства для решения задач и выполнения инженерных расчетов.

В пособии используются следующие условные обозначения:

 – справочный материал;

 – примеры выполненных заданий;

 – письменное задание;

? – устные вопросы;

 – задания для работы на компьютере.

РАЗДЕЛ 1. ОСНОВЫ РАБОТЫ НА КОМПЬЮТЕРЕ

Для работы на компьютере нужны две составляющие: аппаратные и программные средства, между которыми должно быть соответствие.

📖 *Аппаратные средства* состоят из устройств: ввода (клавиатура, мышь, сканер и др.), вывода (монитор, принтер и др.) и обработки информации (электронные элементы, расположенные в системном блоке).

Программные средства делятся на системные, прикладные и средства программирования (для создания новых программ).

Найти две системы с одинаковыми аппаратными и программными конфигурациями сложно, и поэтому для эффективной эксплуатации вычислительной техники от специалистов требуется достаточно широкий уровень знаний и практических навыков.

Основу взаимодействия человека с компьютером образуют два принципа: соглашения и умолчания.

Соглашение определяет набор функций, которые будут выполняться конкретной программой, и варианты исполнения каждой функции (то, что можно реализовать с помощью данной программы).

Умолчание устанавливает конкретный вариант исполнения той или иной функции (при наличии альтернатив!), если пользователь явным образом не потребовал иного.

1.1. ОСНОВНЫЕ ОБЪЕКТЫ ФАЙЛОВОЙ СТРУКТУРЫ

При хранении информации на ЭВМ решаются две проблемы: сохранение данных в наиболее компактном виде и обеспечение к ним удобного и быстрого доступа (если доступ не обеспечен, то это не хранение). Для обеспечения доступа необходимо, чтобы данные имели упорядоченную структуру. В качестве единицы хранения данных принят объект переменной длины, называемый *файлом*. Проще всего представить себе файл в виде безразмерного канцелярского досье, в которое можно по желанию добавлять содержимое или извлекать его оттуда.

Папки (каталоги) являются важными элементами иерархической структуры, необходимыми для обеспечения удобного доступа к файлам (если файлов на носителе слишком много).

✍ *1.1. Закончить предложения или ответить на вопросы.*

1. Файловая структура – это …
2. Носители данных – это … Они делятся на …
3. Файл – это …
4. Что такое расширение файла, для чего оно предназначено?
5. По каким признакам и на какие группы классифицируются файлы?
6. Папка (каталог) – это …
7. По каким признакам и на какие группы классифицируются папки?
8. Путь доступа к объекту состоит из …
9. Полное имя файла образуется из …

10. Какая папка называется текущей?
11. Сколько уровней вложенности папок приведено на рис. 1?
12. Указать папку верхнего уровня, нижнего уровня, родительскую для папки Вопросы (рис.1).

Рис. 1. Пример файловой структуры

1.2. ОПЕРАЦИОННАЯ СИСТЕМА WINDOWS XP

Процесс работы компьютера в определенном смысле сводится к обмену данными между устройствами.

🕮 Вся работа на компьютере выполняется под руководством операционной системы (ОС), которая обеспечивает совместное функционирование всех устройств компьютера и предоставляет пользователю доступ к его ресурсам. В операционной системе имеются программные модули, управляющие файловой системой. В состав ОС входит специальная программа – командный процессор, которая запрашивает у пользователя команды и выполняет их. Основная функция всех операционных систем – посредническая.

В настоящее время, лидирующее место занимает операционная система Windows XP, которая обладает наибольшей универсальностью, имеет самое широкое распространение и, соответственно, имеет особую поддержку со стороны производителей аппаратного и программного обеспечения.

Все элементы файловой структуры (папки, документы, программы, ярлыки) в ОС Windows XP называются *объектами*. Каждый объект имеет имя и графическое изображение, что делает работу пользователя с ним более наглядной и удобной.

Преимущество ОС Windows в стандартизации интерфейса пользователя (раскрытые папки и программы – окна, имеют примерно одинаковый набор средств диалога).

После запуска ОС Windows на экране появляется *Рабочий стол* (рис. 2), на котором в удобном для работы порядке располагаются *значки объектов* (пиктограммы) и *элементы управления*.

Рис. 2. Рабочий стол Windows XP

Содержимое файловой структуры можно просмотреть с помощью системы окон *Мой компьютер*, последовательно открывая окна папок (например, Мой компьютер → Локальный диск (D:) → № гр).

1.2. Закончить предложения или ответить на вопросы.

1. Панель задач – это…
2. На ней расположены … (см. рис. 2)
3. Контекстное меню – это …
4. Как вызывается справочная система? Для чего она предназначена?
5. Скопировать объект – значит …
6. При перемещении объектов выполняется …
7. Корзина используется для … Из окна Корзины можно выполнять …
8. Что такое Буфер обмена? Какие команды используются для работы с ним?

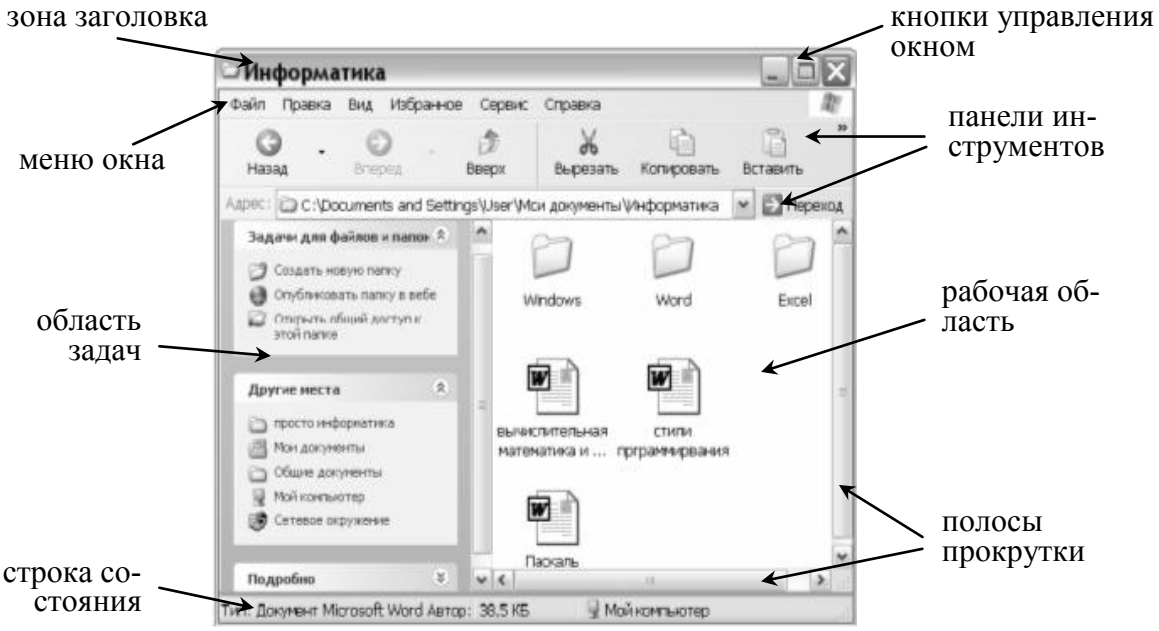


Рис. 3. Окно папки системы *Мой компьютер* и его элементы

9. Окно папки имеет следующие элементы: … (*перечислить, указать назначение, см. рис. 3*)
10. Каким образом можно настроить внешний вид окна?
11. Диалоговое окно предназначено для …
12. Может иметь следующие элементы (*см. рис. 4*).

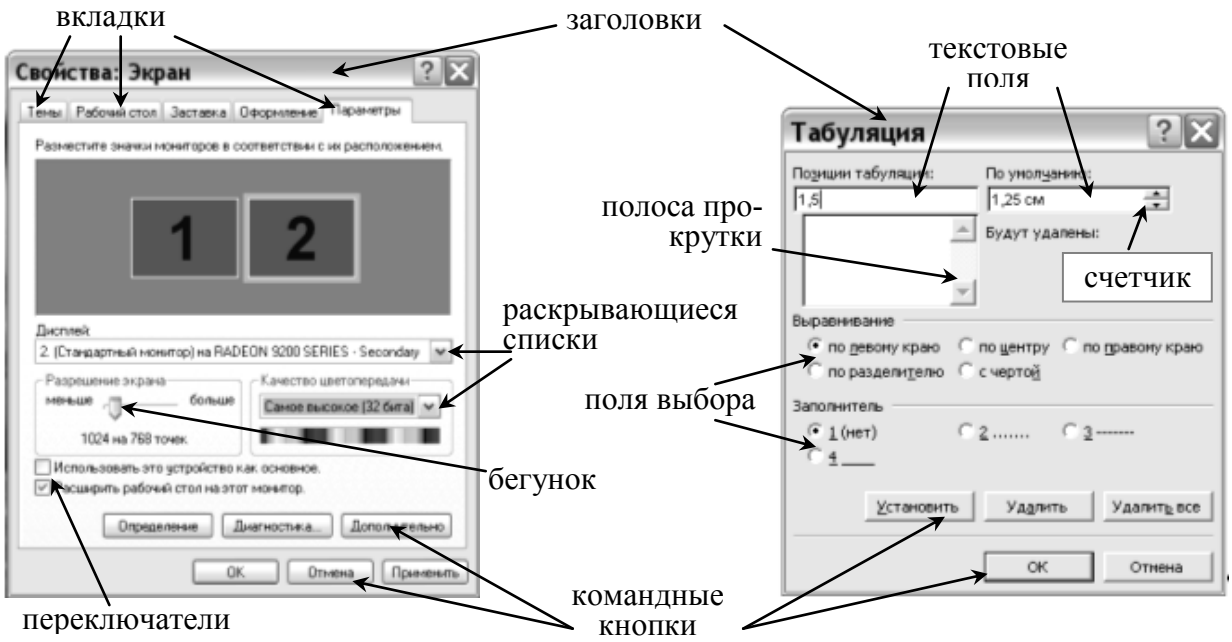


Рис. 4. Диалоговые окна и их элементы

📖 В Windows XP большую часть команд можно выполнять с помощью мыши, с которой связан активный элемент управления – *указатель мыши*.

Основными приемами управления с помощью мыши являются:

1. *щелчок левой кнопкой* мыши (ЛКМ) используется для выделения объекта, выбора пункта меню, перевода курсора. Для выделения группы объектов щелчок ЛКМ сочетается с одновременным нажатием клавиш *Shift* (для смежных объектов) и *Ctrl* (для несмежных объектов);
2. *двойной щелчок* ЛКМ используется для открытия папки или файла, запуска программы на выполнение;
3. *щелчок правой кнопкой* мыши (ПКМ) используется для вызова контекстного меню;
4. *перетаскивание* (*drag-and-drop*) левой кнопкой мыши используется для перемещения или копирования выделенных объектов, на которых установлен указатель, или для перемещения окон, изменения их размеров;
   примечание: это действие можно выполнять при одновременном нажатии клавиш *Shift* или *Ctrl*;
5. *протягивание мыши* (*drag*) используется для выделения объектов, находящихся в прямоугольной области, или изменения размеров окна;
6. *специальное перетаскивание* выполняется правой кнопкой мыши, при этом появляется контекстно-зависимое меню объектов;
7. *зависание* используется для вызова *всплывающей подсказки*, кратко характеризующей назначение или свойства объекта;

8. *прокрутка* выполняется после установки указателя на полосу прокрутки и нажатия ЛКМ либо с помощью скроллинга ("колесика" мыши), используется для быстрого перемещения по рабочей области окна.

💻 *Задание 1. Выполнить с помощью мыши все перечисленные выше действия.*

? Почему не всегда удается изменить размеры окна или переместить объекты в определенное место в окне папки?
Какие команды содержатся в контекстном меню панели задач?

💻 *Задание 2. Открыть окна папок Мои документы и Корзина, выполнить различные варианты сортировки объектов и изменить их представление. Расположить окна различными способами (каскадом, сверху вниз, слева направо).*

📖 Прежде чем задавать какие-либо команды, нужно определить и *выделить* те объекты, над которыми их следует выполнять.
Выполнение действий под руководством ОС Windows XP осуществляются с помощью:
- ✓ меню окна,
- ✓ контекстного меню рабочей области окна или объекта,
- ✓ кнопок на панели инструментов,
- ✓ клавиатуры.

? Как определить сочетания клавиш, с помощью которых можно выполнить какую-либо команду?
Как определить назначение кнопок на панелях инструментов?

Далее перечислены основные действия над объектами файловой структуры и способы их выполнения.

1. *Создать папку*:
   - ✓ открыть окно папки, в которой следует создать новую папку;
   - ✓ выполнить команду *Создать→папку* (из пункта меню *Файл* или из контекстного меню рабочей области окна);
   - ✓ ввести имя папки.

2. *Создать документ*:

1 способ
   - ✓ открыть окно папки, в которой следует создать документ;
   - ✓ выполнить команду *Создать→документ определенного типа* (из пункта *Файл* или из контекстного меню рабочей области окна);
   - ✓ ввести имя документа.

2 способ
   - ✓ запустить программу *Пуск→Программы→…*(*название программы*);
   - ✓ выполнить команду *Файл→Сохранить как…*;
   - ✓ в диалоговом окне (рис. 5) указать путь доступа и имя документа; тип документа задается приложением по умолчанию.

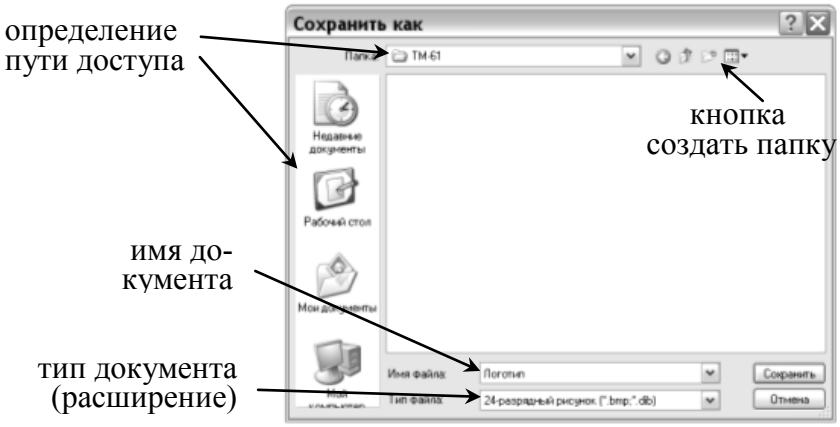


Рис. 5. Диалоговое окно сохранения документа

Примечание: папку можно создать из диалогового окна сохранения документа, нажав кнопку *Создать папку.*

3. Сохранить изменения в документе:

1 способ

- ✓ открыть документ из окна папки, ввести данные;
- ✓ выполнить команду *Файл→Сохранить*;

2 способ

- ✓ запустить программу;
- ✓ открыть документ (диалоговое окно открытия документа аналогично окну сохранения документа, рис. 5);
- ✓ отредактировать или отформатировать данные;
- ✓ нажать кнопку *закрыть* и в появившемся окне сообщения (рис. 6) выбрать кнопку *Да*.

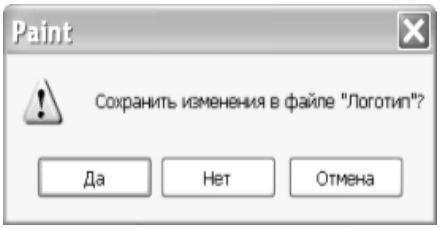


Рис. 6. Окно сообщения об особой ситуации

4. Создать ярлык для объекта на Рабочем столе:

1 способ

- ✓ открыть окно папки, содержащей объект;
- ✓ выделить объект;
- ✓ выполнить команду *Отправить→Рабочий стол (создать ярлык)* (из пункта *Файл* или из контекстного меню)*;*

2 способ

- ✓ вызвать контекстное меню *Рабочего стола* и выполнить команду *Создать→Ярлык*;
- ✓ в диалоговом окне указать имя объекта (при необходимости с помощью кнопки *Обзор*… можно определить путь доступа);
- ✓ далее ввести имя ярлыка;

3 способ

- ✓ открыть окно папки и выделить объект;
- ✓ перетащить ПКМ объект на *Рабочий стол* и из контекстного меню выполнить команду *Создать ярлыки*.

5. *Скопировать объекты*:

1 способ (с помощью буфера обмена)

- ✓ открыть окно папки-источника (в которой содержится объект);
- ✓ выделить объекты;
- ✓ выполнить команду *Копировать* (из пункта меню *Правка* или из контекстного меню) или нажать кнопку  на панели инструментов;
- ✓ открыть окно папки-приемника (в которой будет создана копия объекта);
- ✓ выполнить команду *Вставить* (из пункта меню *Правка* или из контекстного меню) или нажать кнопку  на панели инструментов.

Примечание: буфер обмена – это область оперативной памяти компьютера, в которой временно (до выключения или перезагрузки) можно хранить объекты файловой структуры или текстовые фрагменты для дальнейшего копирования или перемещения.

2 способ (с помощью мыши, минуя буфер обмена)

- ✓ открыть два окна папок, участвующих в копировании;
- ✓ перетащить мышью выделенные объекты из окна папки-источника в окно папки-приемника. При перетаскивании с одного диска на другой автоматически выполняется копирование; при перетаскивании в пределах одного диска следует дополнительно удерживать нажатой клавишу *Ctrl*.

6. *Переместить объекты* (действия выполняются аналогично копированию):

1 способ

- ✓ открыть окно папки-источника;
- ✓ выделить объекты;
- ✓ выполнить команду *Вырезать* (но не Копировать!) или нажать кнопку  на панели инструментов;
- ✓ открыть окно папки-приемника;
- ✓ выполнить команду *Вставить* или нажать кнопку  на панели инструментов;

2 способ

- ✓ открыть два окна папок, участвующих в перемещении;
- ✓ перетащить мышью выделенные объекты из окна папки-источника в окно папки-приемника. При перетаскивании объектов в пределах одного диска автоматически выполняется перемещение; при перетаскивании с одного диска на другой следует дополнительно удерживать нажатой клавишу *Shift*.

Примечание: при перетаскивании объектов правой клавишей мыши появляется контекстное меню, в котором можно выбрать действие (копировать, переместить или отменить). Поэтому этот прием *универсальный*, подходит как для копирования, так и для перемещения объектов внутри одного диска или с одного диска на другой диск.

7. Переименовать объект:

1 способ

- ✓ выделить объект;
- ✓ выполнить команду *Переименовать* (из пункта *Файл* или из контекстного меню объекта);
- ✓ ввести с клавиатуры новое имя и нажать клавишу *Enter*;

2 способ

- ✓ выделить объект;
- ✓ щелкнуть ЛКМ на имени объекта;
- ✓ ввести с клавиатуры новое имя и нажать клавишу *Enter*;

3 способ

- ✓ выделить объект;
- ✓ нажать функциональную клавишу *F2*;
- ✓ ввести с клавиатуры новое имя и нажать клавишу *Enter*.

8. Удалить объекты:

1 способ

- ✓ выделить объекты;
- ✓ выполнить команду *Удалить* (из пункта меню *Файл* или из контекстного меню одного из объектов);
- ✓ в появившемся диалоговом окне подтвердить удаление;

2 способ

- ✓ выделить объекты;
- ✓ нажать на клавишу *Delete*;
- ✓ в появившемся диалоговом окне подтвердить удаление;

3 способ

- ✓ выделить объекты;
- ✓ перетащить их мышью на значок *Корзины*.

Примечание: при удалении объектов с жесткого диска они помещаются в *Корзину* (специальную папку на жестком диске, в которой временно хранятся удаленные из файловой структуры объекты), из которой их можно восстановить в исходную папку. Если выполнять удаление объекта при дополнительно нажатой клавише *Shift*, то он удаляется с диска, минуя *Корзину*.

💻 *Задание 3. Выполнить действия.*

I вариант

1. Создать папку с *номером* вашей группы на диске D: и в ней подпапку с *фамилией* пользователя.

2. Создать два документа: текстовый документ *Текст1* в подпапке с фамилией и точечный рисунок *Логотип* в папке с номером группы.
3. Сохранить изменения в документах.
4. Создать ярлыки для своих документов и программы *Проводник* на *Рабочем столе* разными способами.
5. *Скопировать объекты*: текстовый документ в папку с номером группы; документы из окна папки с номером группы в папку *Мои документы*.
6. *Переместить объекты*: *Логотип* из папки с номером группы в папку с фамилией; документы *Текст1* и *Логотип* из папки *Мои документы* на *Рабочий стол*.
7. *Переименовать* документы и ярлыки документов на *Рабочем столе*.

II вариант

1. Открыть окно *Мой компьютер* и создать систему папок *D:\№группы\Тексты\Документы*.
2. В папке *Документы* создать два документа OpenDocument с именами *Док1.odt* и *Док2.odt*.
3. В папке *Тексты* создать два текстовых документа с именами *Текст1.txt* и *Текст2.txt*.
4. *Скопировать* документы OpenDocument в папку второго уровня. *Переместить* текстовые документы в папку первого уровня.

📖 Для навигации по файловой структуре компьютера и ее обслуживания используется служебная программа *Проводник*, которая относится к категории диспетчеров файлов. Проводник интегрирован в операционную систему Windows. Пользователь работает с ним даже тогда, когда его не видит. Если по щелчку ПКМ на каком-либо объекте получает контекстное меню, либо при перетаскивании объектов из одного окна в другое выполняется копирование или перемещение, то это результат невидимой (заочной) работы Проводника. Однако с ним можно работать и напрямую, запустив: *Пуск→Программы→Стандартные→Проводник* (или другим способом).

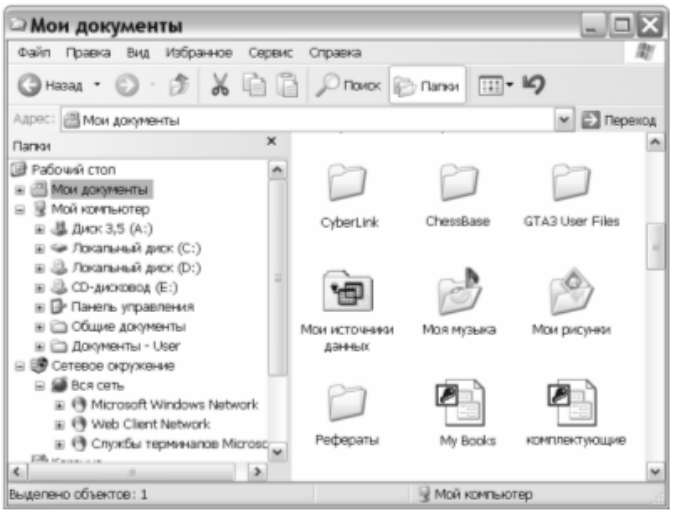


Рис. 7. Окно программы Проводник

Окно программы Проводник (рис. 7) имеет две рабочие области: слева панель папок (для отображения файловой структуры) и справа панель содержимого (для отображения содержимого текущей папки). Кнопка *Папки* на панели инструментов *Обычные кнопки* позволяет перейти из окна папки в окно Проводника и обратно.

? Что такое навигация по файловой структуре?

С помощью каких программ можно осуществлять навигацию?

На что указывают значки "+" и "–" рядом с именами папок в окне программы Проводник?

Какие преимущества у программы Проводник перед системой окон Мой компьютер?

Для чего предназначена поисковая система? Как она вызывается? По каким параметрам можно выполнять поиск?

? Вопросы для повторения.

1. Назначение операционной системы.
2. Объекты файловой структуры – …
3. Перечислить элементы интерфейса ОС Windows XP.
4. Дать определение окна, диалогового окна. Перечислить их элементы.
5. Что такое буфер обмена?
6. Основные приемы управления с помощью мыши.
7. Какие действия можно выполнять с объектами файловой структуры? Способы выполнения этих действий.
8. Что такое иерархия папок?
9. Какая папка называется: корневой, I уровня, родительской, текущей?
10. Чем отличаются действия *Копирование* и *Перемещение*?
11. Какими двумя различными способами можно выполнять действие *Копирование* (*Перемещение*)?

РАЗДЕЛ 2. ОСНОВЫ ПРОГРАММИРОВАНИЯ

Человек мыслит, а ЭВМ выполняет команды человека автоматически ("машинально"). Но не наоборот! Если машина что-то не смогла сделать, это не говорит о ее неправильной работе, а это означает, что человек не сумел объяснить ЭВМ то, что ему нужно получить. Главное отличие вычислительных машин от всех остальных искусственно созданных предметов – это программное управление их работой.

На ЭВМ работают две категории исполнителей:

- ✓ *пользователи* выполняют действия, могут сразу проверить свои результаты и их проанализировать;
- ✓ *программисты* мыслят с опережением своих действий: сначала планируют цепочку нужных команд, после этого ее выполняют и проверяют правильность своих предположений.

📖 *Компьютерная программа* – это набор элементарных команд процессора, представленная в файле в виде последовательности байтов (машинного кода). Программы в таком виде можно составлять вручную, но подобная работа очень трудоемкая и в большинстве случаев человеку просто не под силу. Поэтому программа пишется на одном из языков программирования, в виде *алгоритма* – набора команд, который называется исходным тестом (или исходным кодом) программы. С помощью специальной программы, называемой компилятором, исходный текст переводится в набор инструкций процессора.

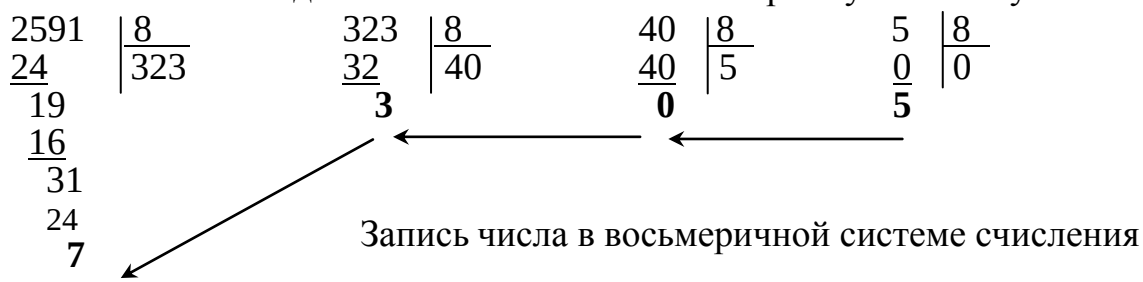
2.1. СИСТЕМЫ СЧИСЛЕНИЯ И КОДИРОВАНИЕ ИНФОРМАЦИИ

Позиционная система счисления характеризуется тем, что каждый символ приобретает различное значение в зависимости от того, какое он занимает место (позицию) при записи числа. Сравните 123, 231 и 312.

📖 Каждая *позиция* в числовом ряду соответствует определенной степени *основания* системы счисления. Общепринятой является десятичная система счисления (основание системы равно 10). В качестве основания системы счисления можно использовать любое число как меньшее, так и большее 10. Правила записи многоразрядных чисел, выполнения арифметических операций во всех позиционных системах счисления одинаковые.

Примеры перевода из одной системы счисления в другую (показаны разные способы).

1. Число 2591 из десятичной системы в восьмеричную систему счисления.



Таким образом, $2591_{10}=5037_8$ (нижний индекс обозначает основание системы счисления).

Сделаем проверку $5037_8=5\cdot8^3+0\cdot8^2+3\cdot8^1+7\cdot8^0=2560+24+7=2591_{10}$.

2. Число 53 из десятичной системы в двоичную систему счисления.

Частное	Результат	Остаток
53 : 2	= 26	1
26 : 2	= 13	0
13 : 2	= 6	1
6 : 2	= 3	0
3 : 2	= 1	1
1 : 2	= 0	1

Запись числа в двоичной системе счисления

Таким образом, $53_{10}=110101_2$.

Проверка: $110101_2=1\cdot2^5+1\cdot2^4+0\cdot2^3+1\cdot2^2+0\cdot2+1=32+16+4+1=53_{10}$.

✍ *2.1.1. Выполнить перевод чисел:*

в десятичную систему счисления:

a) 12210_3, b) 24602_7, c)$5d8c4_{16}$ (где a=10_{10}, b=11_{10}, c=12_{10} и т.д.);

из десятичной системы в систему с другим основанием

число	основание системы
255	2
421	5

число	основание системы
5129	11
2458	16

? Как можно складывать числа, записанные в одной системе счисления? В разных системах счисления?

🕮 Из всех систем записи числовой информации для современной вычислительной техники наиболее удачной оказалась двоичная, в которой используются всего *два* символа: *нуль* и *единица*. Но в общем случае это какие-то два (и только два) явно различающихся между собой состояния. На магнитных носителях наличие намагниченности означает 1, а ее отсутствие – 0. В самой машине информация представляется электрическими сигналами двух различных уровней: высокий уровень напряжения соответствует 1, а низкий уровень означает 0.

ДВОИЧНАЯ СИСТЕМА СЧИСЛЕНИЯ

Позиция	10	9	8	7	6	5	4	3	2	1	0
Степень	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Значение	1024	512	256	128	64	32	16	8	4	2	1

Один разряд двоичного числа называют *бит*. Термины "бинарный" и "двоичный" являются равнозначными. При записи числовой информации последовательность бит называют *словом* независимо от того, имеет оно какой-либо смысл или нет. Фиксированное количество двоичных разрядов, с которыми оперирует данная ЭВМ, называют *машинным словом*. Алфавитная и другая информация выражается через цифровые данные.

Восьмиразрядное бинарное слово называется *байтом*. Более крупная единица измерения – килобайт (Кбайт). Условно можно считать, что одна страница неформатированного текста составляет около 2 Кбайт.

1Кбайт=1024 байт=2^{10} байт *килобайт*
1Мбайт=1024 Кбайт=2^{20} байт *мегабайт*
1Гбайт=1024 Мбайт=2^{30} байт *гигабайт*
1Тбайт=1024 Гбайт=2^{40} байт *терабайт*

✍ *2.1.2. Выберите правильный вариант ответа.*

1. Наибольшее натуральное число, кодируемое 8 битами:
   a) 127; b) 128; c) 255; d) 256; e) 512.
2. Число байт, необходимое для записи числа 2^{44}:
   a) 5; b) 6; c) 11; d) 44; e) 88.
3. Число байт, необходимое для записи числа 8^{14}:
   a) 6; b) 10; c) 42; d) 112; e) 192.
4. В пяти килобайтах:
   a) 5000 байт; b) 5120 байт; c) 500 байт; d) 5000 бит; e) 5120 бит.
5. Количество чисел, которое можно закодировать нулями и единицами в 10 позициях, равно:
   a) 1000; b) 1024; c) 10; d) 256; e) 512.
6. Наибольшее натуральное число, кодируемое 16 битами:
   a) 255; b) 256; c) 512; d) 32768; e) 65535; f) 99999999.
7. Число байт, необходимое для записи выражения $8^4 \cdot 4^8$:
   a) 3; b) 4; c) 8; d) 12; e) 16.
8. Наибольшее целое решение x неравенства 4^{x+4} бит > 8^{x-3} Кбайт равно:
   a) 2; b) 4; c) 3; d) 5; e) 9.

2.2. АЛГОРИТМЫ И АЛГОРИТМИЗАЦИЯ

Каждый из нас ежедневно использует различные алгоритмы: инструкции, правила, рецепты и т.п. Например, открывая дверь ключом, никто не задумывается над тем, в какой последовательности выполнять действия, и что эта последовательность есть определенный алгоритм. Обычно это делают *автоматически*. Однако чтобы обучить человека некоторым новым для него действиям, нужно четко указать *инструкции* по их выполнению.

🕮 *Алгоритм* – это система предписаний, определяющая процесс перехода от исходных данных к результату, предназначенная некоторому исполнителю для решения некоторого класса задач.

Одно действие из этой системы носит название *команда*.

Человек не единственно возможный исполнитель. Все живые существа и даже отдельные клетки исполняют различные алгоритмы. Способны на это и созданные человеком устройства – роботы-манипуляторы, станки с программным управлением, автоматы, ЭВМ. Но прежде чем составлять алгоритм решения задачи, нужно узнать, какие действия предполагаемый *исполнитель* способен выполнить. Иначе каждый алгоритм должен быть составлен в *системе команд* некоторого исполнителя.

📖 Множество задач, для которых можно определить один способ решения, называется *классом задач*. Алгоритм работает не над конкретными значениями, а над *переменными* (значения которых изменяются).

Решения задач, содержащих переменные, производят в общем виде. При этом возникает необходимость исследовать все различные наборы значений переменных и определить их ограничения.

В качестве примера рассмотрим класс задач для нахождения значения переменной *y* по формуле $y = \sqrt{x}$. Однозначно определить значение *y* нельзя, так как оно зависит от переменной *x*, называемой *аргументом*.

Подставим вместо *x* некоторое значение и получим конкретную задачу. Например, при *x*=1 получим *y*=1; при *x*=0 результат *y*=0; при *x*= -1 значение квадратного корня среди вещественных чисел не существует. Таким образом, при подстановке значения переменной *x* получается конкретная задача, имеющая однозначное решение. Из курса математики известно, что квадратный корень из *x* существует при $x \geq 0$. То есть при решении данного класса задач будут получены принципиально различающиеся ответы: числовое значение *y* (при $x \geq 0$), текстовое сообщение (при $x < 0$). Следовательно, при разработке алгоритма необходимо рассмотреть все возможные варианты, которые зависят от класса задач, в частности, от области определения функции.

Пример вложенности классов задач приведен на рис. 8.

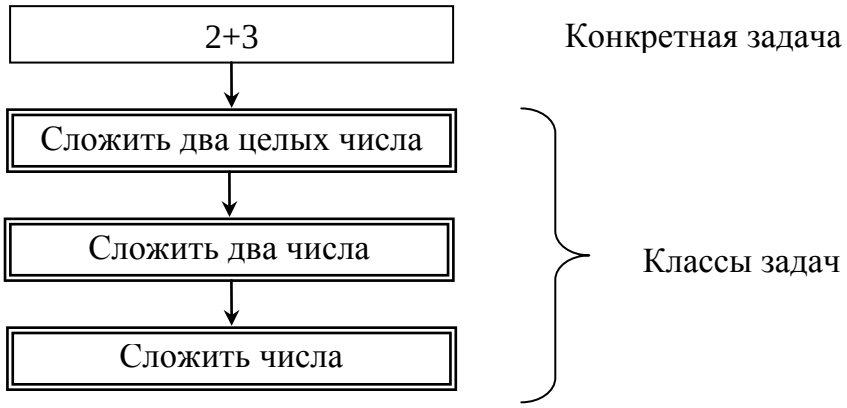


Рис. 8. Примеры классов задач

? Какая задача в приведенной схеме (рис.8) будет иметь самое универсальное решение. Почему?

Каждый алгоритм должен обладать следующими *свойствами*:

1. *Точность* – алгоритм должен обстоятельно и подробно рассказывать, что делать в любой момент времени.
2. *Дискретность* – решение задачи должно быть расчленено на отдельные элементарные действия.
3. *Упорядоченность* – все действия в алгоритме должны быть выстроены в четком, определенном порядке.
4. *Результативность* – результат алгоритма должен быть получен за минимальное число шагов, т.е. он должен быть компактным.

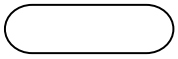
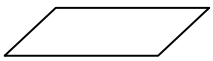
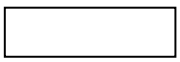
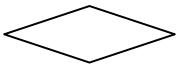
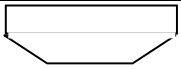
5. *Массовость* – алгоритм должен быть как можно более универсальным, подходящим для решения разных типов задач.
6. *Понятность* – если алгоритм не может понять разработчик, то как же в нем разобраться компьютеру или другому исполнителю.

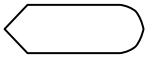
СПОСОБЫ ОПИСАНИЯ АЛГОРИТМА

- ✓ *На разговорном языке.*
- ✓ *На языке блок-схем.*
- ✓ *На алгоритмическом языке* (языке программирования).

📖 *Блок-схема* – это графическое изображение алгоритма в виде определенным образом связанных между собой нескольких типов блоков.

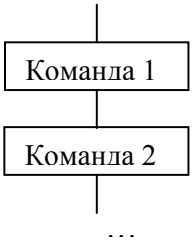
ГРАФИЧЕСКИЕ ЭЛЕМЕНТЫ БЛОК-СХЕМЫ

Основные блоки	Назначение
	Начало; конец
	Ввод; вывод
	Вычисления и присваивания
	Проверка условия
	Начало цикла
	Конец цикла

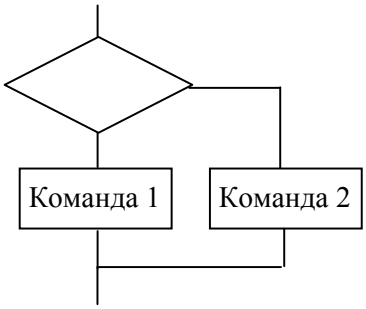
Вспомогательные блоки	Назначение
	Терминал
	Печатный документ
	Магнитный диск
	Соединение

ТИПЫ АЛГОРИТМИЧЕСКИХ СТРУКТУР

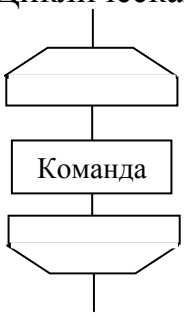
Линейная



Разветвляющаяся



Циклическая



Язык схем настолько четок, что исполнитель, получивший блок-схему алгоритма, ни в каких дополнительных разъяснениях не нуждается.

При составлении блок-схем следует руководствоваться правилами:

- ✓ блок-схема должна развиваться сверху вниз; при необходимости – слева направо;
- ✓ блок *Начало* имеет 1 выход, *Конец* – 1 вход; блоки ввода/вывода, вычисления и присваивания, начала и конца цикла имеют 1 вход и 1 выход; блок условия – 1 вход и 2 выхода;
- ✓ выход рисуется точно под входом.

📖 *Алгоритмизация задачи* – процесс разработки алгоритма для решения задачи человеком или машиной. Это сложный творческий процесс, требующий профессиональных знаний и большой изобретательности. Весь процесс разделен на этапы (рис. 9).

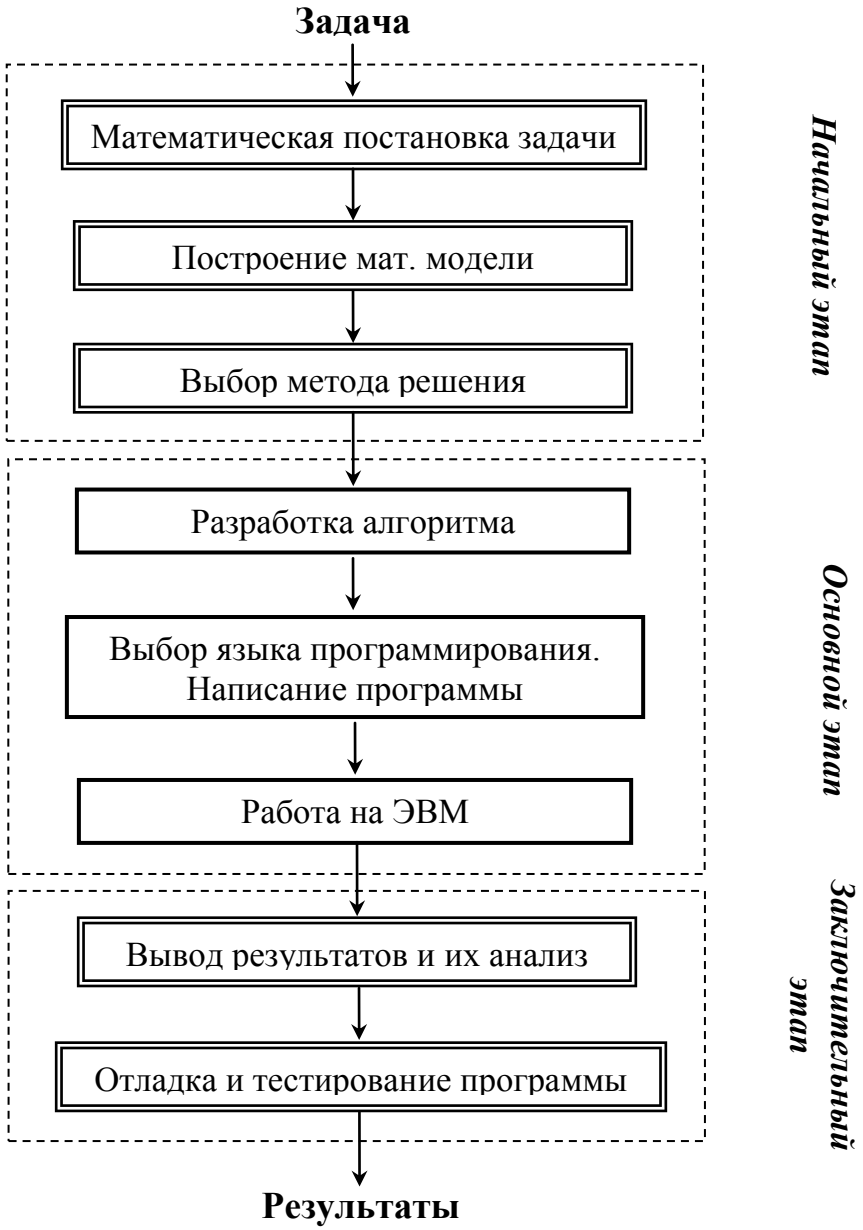


Рис. 9. Этапы алгоритмизации задачи

Математическая постановка задачи предполагает определение того, что дано (исходных данных), что требуется найти (результатов) и условий допустимости исходных данных.

Составление математической модели – создание образа того объекта, процесса или явления, с которым приходится иметь дело. Модель обычно отражает наиболее существенные свойства объекта в контексте решаемой задачи, исключая из рассмотрения несущественные.

Выбор метода решения – оптимальное использование имеющейся математической модели.

Отладка – процесс поиска *логических* ошибок в программе путем прогона ее на ЭВМ.

Тестирование – выполнение программы для различных наборов данных, называемых *контрольными примерами*. Этот процесс проводится после *отладки* программы, когда программист убеждается в отсутствии явных ошибок.

В данном пособии для большинства задач (в силу простоты и однозначности их решения) вместо построения математической модели и выбора метода решения используется *описание решения*, на заключительном этапе выполняются контрольные примеры.

✍ *2.2.1. Что из нижеперечисленного можно считать алгоритмами? Ответ пояснить.*

a) Порядок набора международного телефонного номера
b) Каталог книг в библиотеке
c) Рецепт приготовления клея
d) Настенный календарь на текущий год
e) Расписание движения самолетов
f) Порядок оказания первой медицинской помощи
g) Инструктаж по технике безопасности
h) План лекции

✍ *2.2.2. Описать следующие алгоритмы: заварить чай, найти корни квадратного уравнения.*

? Вопросы для повторения.

1. Что такое алгоритм? Для чего он предназначен?
2. Перечислить свойства алгоритмов.
3. Какие существуют способы описания алгоритмов?
4. Какие существуют типы алгоритмических структур? Чем они отличаются? Привести примеры из жизни.
5. Что такое класс задач? Привести примеры.
6. Что такое алгоритмизация задачи? Из каких этапов она состоит?

2.3. ЯЗЫКИ ПРОГРАММИРОВАНИЯ И СРЕДЫ

📖 *Языки программирования* – это искусственно созданные языки, которые в отличие от естественных имеют ограниченное число "слов" и очень строгие правила записи действий (*команд*).

ИСПОЛЬЗОВАНИЕ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ ПРЕДПОЛАГАЕТ

- ✓ проведение *анализа* – умения разбить задачу на более мелкие части;
- ✓ проведение синтеза – умения объединить части в единое решение задачи;
- ✓ построение плана – определение того, как решить каждую из частей задачи;
- ✓ понимание семантики – смысла команд и синтаксиса – правильного способа выразить то, что нужно выполнить, посредством последовательности команд.

ПРЕИМУЩЕСТВА ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

- ✓ Выполнение последовательностей однотипных команд для разных наборов данных.
- ✓ Организация удобного интерфейса пользователя с четким разграничением действий, удобство ввода данных и просмотра результатов.

Сложность программирования заключается в том, что при записи алгоритма решения задачи необходимо четко знать правила написания и использования языковых единиц.

Конструкции разных языков программирования различаются главным образом "внешним видом", набором ключевых слов или порядком следования компонентов, составляющих его основу, которая прошла проверку временем и практикой.

Язык Паскаль был разработан Никлаусом Виртом в 1968-1971гг. в Цюрихинском институте информатики (Швейцария) как специальный язык для обучения студентов, но вскоре получил распространение и среди программистов. В настоящее время он претерпел модификации (версии), связанные с расширением его возможностей и сохранением его основ.

Система программирования – комплекс языковых и программных средств, предназначенных для автоматизации процесса составления, отладки программ и подготовки их к выполнению. Включает в себя язык программирования, один или несколько трансляторов разных типов (интерпретаторы и компиляторы), библиотеку программ, языково-ориентировочный текстовый редактор, отладчик. Часть из этих компонентов может отсутствовать.

2.3.1. Среда программирования BORLAND PASCAL

📖 С помощью языков программирования создается не готовая программа, а только ее текст. Для того чтобы выполнить программу на компьютере, надо набрать текст в специальной среде программирования и перевести его на язык *машинных кодов* (создать файл с расширением *.exe*), используя *компилятор*. Запуск программы на выполнение называется ее *прогоном*.

Назначение компилятора:

1) проверяет текст программы на языке программирования на наличие или отсутствие синтаксических ошибок;
2) при наличии синтаксических ошибок выдает сообщение о них;
3) при отсутствии синтаксических ошибок создает объектный файл или программу на машинном-ориентированном языке.

Среда программирования Borland Pascal имеет два *режима*:

1. *окно редактора*, в котором можно работать с текстом программы;
2. *окно программы* пользователя, появляющееся при прогоне программы, в котором ее текст не отображается.

Окно редактора среды программирования (рис. 12) содержит:

1. меню (для перехода в которое используется клавиша F10) с пунктами:
   File – работа с файлами;
   Edit – многооконное редактирование;

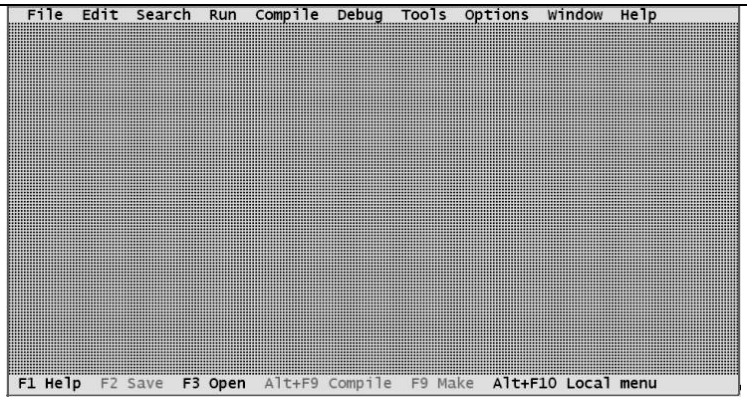
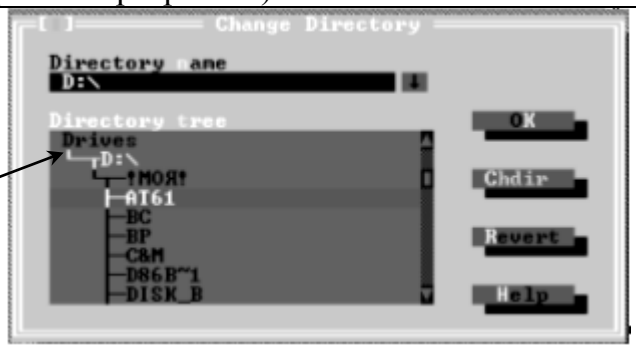
Search – поиск и замена в тексте;
Run – выполнение программы;
Compile – компиляция программы;
Debug – отладка программы;
Options – установка режимов;
Window – работа с окнами;
Help – помощь.

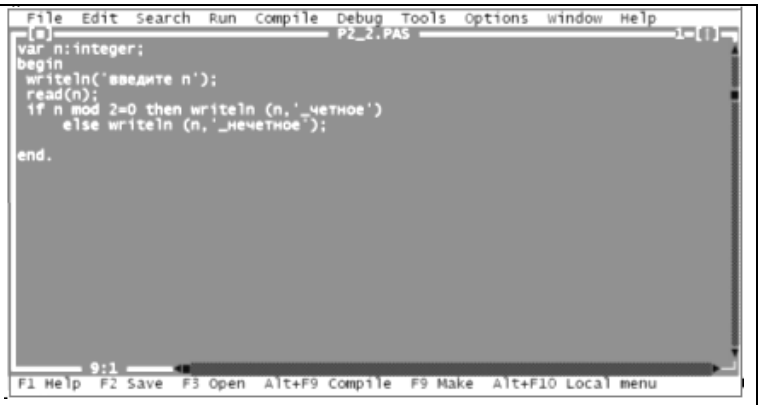
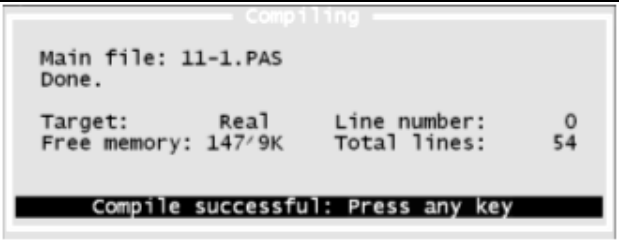
2. рабочую область, предназначенную для отображения окон открытых файлов (с текстами программ), в каждом из которых расположены:
   - ✓ на верхней линии: номер окна (=1=) и кнопки [■] – закрыть окно, [↑] – развернуть окно ([↕] – восстановить размеры окна);
   - ✓ на нижней линии: координаты текущей позиции курсора = 1:1 =;
   - ✓ горизонтальная и вертикальная полосы прокрутки;
3. строку-подсказку о назначении функциональных клавиш.

ОСНОВНЫЕ КОМАНДЫ, ВЫПОЛНЯЕМЫЕ В ОКНЕ РЕДАКТОРА

Название действия	**Команда**	**Клавиши**
Создание файла	File – New	
Открытие файла	File – Open	F3
Сохранение файла	File – Save	F2
Сохранение под другим именем	File – Save as…	
Компиляция программы	Compile – Compile	Alt + F9
Запуск программы на выполнение	Run – Run	Ctrl + F9
Просмотр результатов	Debug – User screen	Alt + F5
Переход к другому окну документа	Window – List…	Alt + “номер окна”
Закрытие текущего окна	Window – Close	Alt + F3
Закрытие всех окон	Window – Close all	
Помощь	Help	F1
Выход из программы	File – Exit	Alt + X
При работе с блоком текста его необходимо предварительно выделить: SHIFT + клавиши управления курсором либо мышь с нажатой левой кнопкой. Затем к блоку можно применять команды:		
Копирование блока	Edit→Copy, Edit→Paste	(Ctrl+Ins, Shift+Ins) Ctrl + K, C
Перемещение блока	Edit→Cut, Edit→Paste	(Shift+Del, Shift+Ins) Ctrl + K, V
Удаление блока	Edit→Clear	Ctrl+Del

2.3.2. Алгоритм работы с программой в среде Borland Pascal

Действие	Способ выполнения	Вид окна или диалогового окна
1.Запустить BORLAND PASCAL	D:\BP\BIN\BP.exe или с помощью ярлыка	 Рис. 10. Окно редактора, не содержащее окно файла (текста программы)
2.Сделать текущей папку (директорию) своей группы	File→Change dir.. Двойным щелчком последовательно открыть следующие объекты: Drives→D:\→ Номер группы	 Рис. 11. Диалоговое окно изменения директории
3.Создать новое окно для программы Или открыть существующий файл	File→New File→Open…	 Рис. 12. Окно редактора, содержащее окно нового файла (текста программы)
4.Сохранить программу под новым именем	File→Save as.., ввести имя программы (без расширения). Проконтролируйте, что работаете в своей папке!!	 Рис. 13. Диалоговое окно сохранения файла

5.Набрать или изменить текст программы, периодически сохраняя изменения	File→Save или клавиша F2	 Рис. 14. Окно редактора, содержащее окно редактируемого файла (программы)
6.Откомпилировать программу, исправляя синтаксические ошибки	Compile→Compile или клавиши Alt+F9	 Рис. 15. Сообщение об успешном завершении компиляции
7.Запустить программу и выполнить контрольный пример	Run→Run или клавиши Ctrl+F9 Ввести необходимые данные и нажать Enter	 Рис. 16. Окно прогона программы пользователя
8.Просмотреть результат и проанализировать его	Debug→User screen или клавиши Alt+F5 Для возврата в программу нажать любую клавишу	Смена режимов Borland Pascal См рис. 14 и рис.16

2.4. ЛЕКСЕМЫ ЯЗЫКА ПАСКАЛЬ

2.4.1. Алфавит языка ПАСКАЛЬ. Идентификаторы

📖 Программа на языке Паскаль формируется из латинских букв (прописные и строчные воспринимаются одинаково), цифр и специальных символов, совокупность которых образует алфавит языка.

Неделимые последовательности символов алфавита образуют слова (*идентификаторы*), к которым предъявлены следующие ограничения:

- ✓ могут состоять из букв латинского алфавита, цифр, знака подчеркивания; никакие другие символы недопустимы;

- ✓ не могут начинаться с цифры;
- ✓ длина идентификатора может быть произвольной, но значащими считаются первые 63 символа.

Идентификаторы подразделяются на зарезервированные слова, стандартные и пользовательские идентификаторы (*имена* констант, переменных, типов и т.д.).

Зарезервированные (*служебные* или *ключевые*) слова составляют основу языка, имеют определенное смысловое значение и не могут быть переопределены в программе пользователя. В окне редактора они выделяются цветом.

✍ *2.4.1. Определить, допустимы ли следующие имена переменных. Почему?*

112233, MyProgram1, Мой1, My1, hh1h, 7h, {n}, N*N, Begin, _END, x^2

✍ *2.4.2. Определить, сколько различных идентификаторов приведено:*
hhh, h, Hhh, hHh, abc, bac, acb, aBc.

✍ *2.4.3. Какие составные символы (комбинации специальных символов) используются в Паскале? Перечислить и написать их назначение.*

2.4.2. Данные и их обработка

📖 *Информация* является первичным и неопределяемым в рамках науки понятием. *Данные* – это зарегистрированные сигналы, они несут в себе информацию о событиях, происходящих в материальном мире.

КЛАССИФИКАЦИЯ ДАННЫХ

I. по организации хранения:
   1) *константы* или *постоянные* (их значения задаются посредством описания в программе и не могут быть изменены при выполнении программы);
   2) *переменные* (их значения задаются при выполнении программы и могут изменяться неограниченное число раз);

II. по их назначению в программе:
   1) *входные* параметры (*исходные* данные);
   2) *выходные* параметры (*результаты*);
   3) *внутренние* данные (промежуточные).

При работе на ЭВМ всем константам и переменным ставятся в соответствие определенные типы данных, от которых зависят размер выделяемой для них памяти и множество допустимых значений. Типы констант определяются по умолчанию (зависят от их значений), а типы переменных указываются явным образом.

Паскаль является статическим языком. Это означает, что тип переменной определяется при ее описании и не может быть переопределен при выполнении программы. Переменная может участвовать только в операциях, которые предусмотрены для ее типа. Такой подход способствует большей аккуратности и ответственности при составлении программ. А в конечном итоге приводит к более высокой надежности создаваемых программ.

2.4.3. Иерархия типов

Все используемые в языке Паскаль типы данных можно систематизировать (рис.17).

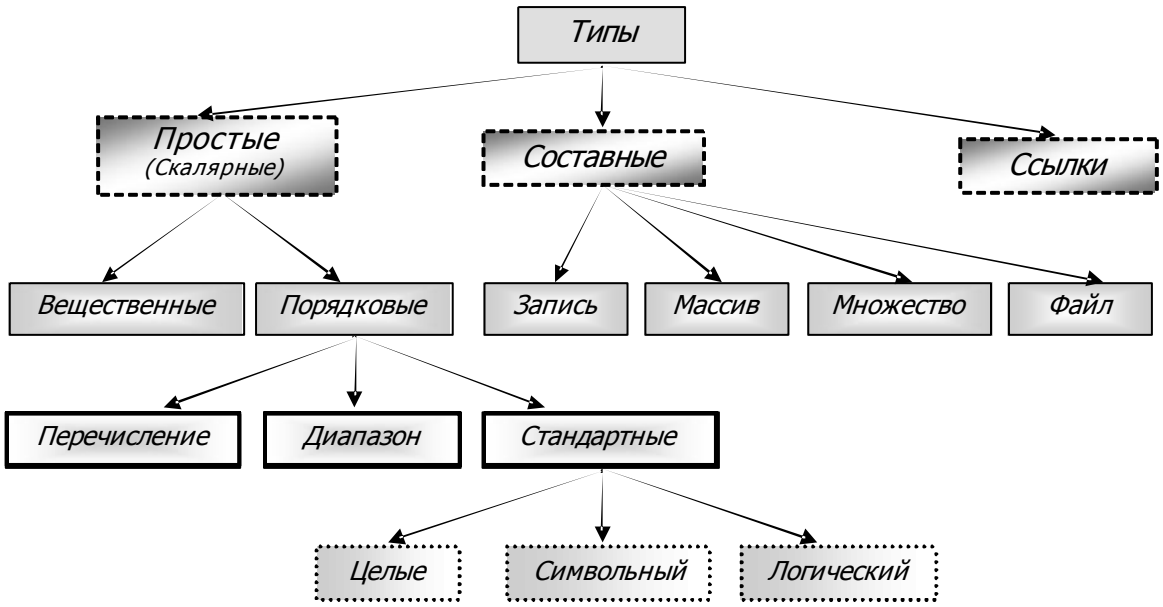


Рис. 17. Иерархия типов языка Паскаль

Переменная простого типа данных при выполнении программы в каждый момент времени имеет единственное значение (число, символ или логическое значение). При вводе новых данных предыдущее значение уничтожается.

Порядковые типы данных отличаются от других тем, что каждый из них имеет конечную последовательность возможных значений, для которых установлен порядок (поставлены в соответствие целые числа).

Целые (целочисленные) типы различаются в зависимости от диапазона и количества байт памяти: *Integer*-2б, *Word*-2б, *Byte*-1б, *Shortint*-1б, *Longint*-4б.

Пример: a:=35; b:=-32 000; c:=0;

Существует несколько *вещественных* типов, однако только тип *Real* можно использовать без ограничений и без дополнительных требований к аппаратуре. В памяти компьютера занимает 6 байт. Вещественные числа могут изображаться в форме с фиксированной точкой и в форме с плавающей точкой. Дроби допускаются лишь десятичные, в которых точка разделяет целую и дробную части.

Пример: d:=3.5; e:=-48.896214;

0.33333 и 3.3333E-01 – вещественные числа
(записи 3.3333E-01 соответствует математическое выражение $3.3333 \cdot 10^{-1}$),
а 1/3 – деление двух целых чисел (не дробное число!).

При работе с вещественными числами важна машинная точность. Например, можно столкнуться с такой ситуацией, когда $0 \neq 0$.

Символьный тип *Char* представляет собой множество, состоящее из 256 различных символов, упорядоченных определенным образом, содержит символы заглавных и строчных букв латинского и русского алфавита, псевдографики, цифры, управляющие символы и другие. Переменная символьного типа в памя-

ти компьютера занимает 1 байт. Символы, используемые в программе, нужно заключать в апострофы. Иначе символ можно записывать с помощью его числового кода (из диапазона от 0 до 255), перед которым ставится #.

Пример: f:='A'; g='#'; h:='1'; k:= #123;

Логический тип *Boolean (1 байт)* может принимать только два значения *false* (ложь), *true* (истина). При сравнении значений логических переменных действует правило false < true.

Пример: i:=false; j:=true;

✍ *2.4.4. Определить, какие выражения представляют собой запись числа 10000?*

1.00E4; E4; 100E2; 1E+4; 10^4.

✍ *2.4.5. Записать диапазоны значений количество памяти, отводимой под вещественный и стандартные типы данных.*

✍ *2.4.6. Привести примеры Перечислений и Диапазонов.*

✍ *2.4.7. Привести примеры констант различных типов.*

✍ *2.4.8. Сколько переменных и какого типа следует определить для:*

a) вычисления стоимости переговоров;
b) вычисления площади круга,
c) вычисления площади треугольника;
d) ввода данных о студенте.

✍ *2.4.9. Определить тип результата для*

a) деления двух чисел;
b) вычисления факториала числа (n!=1*2*…*n).

2.4.4. Операции языка ПАСКАЛЬ

📖 Действия, выполняемые над данными, называются операциями. Они отображаются с помощью специальных символов, например, + * > = или слов, например, or, and, not. Данные, над которыми выполняется какая-либо операция, называются *операндами*.

Операции делятся:

- ✓ по количеству операндов на *унарные* (над одним операндом), *бинарные* (над двумя операндами) и *тернарные* (над тремя операндами);
- ✓ по назначению на *арифметические* (+ – * / **div mod**), *логические* (**and or not**) и *сравнения* (= <> < > <= >=).

От типа данных зависит набор операций над ними. Например, числа можно складывать, вычитать, перемножать, делить, сравнивать; символы можно складывать, сравнивать и т.д. И, наоборот, для каждой операции определены типы операндов и результата. Таким образом, существует *взаимообратная связь* между типами данных и операциями над ними.

ОПЕРАЦИИ ЯЗЫКА ПАСКАЛЬ

Операция	Действие	Тип операнда	Тип результата
Арифметические операции			
+	Сложение	Целый Вещественный	Целый Вещественный
-	Вычитание	Целый Вещественный	Целый Вещественный
*	Умножение	Целый Вещественный	Целый Вещественный
/	Деление	Целый Вещественный	Вещественный Вещественный
div	Целочисленное деление	Целый	Целый
mod	Остаток от деления	Целый	Целый
Операции сравнения (отношения)			
= < > > < < = > =	Равно Не равно Больше Меньше Меньше или равно Больше или равно	Любой простой тип	Логический
Логические операции			
Not And Or	отрицание и или	Логический	Логический

Приоритет операций – это порядок выполнения операций.

Приоритет	Операции
1	Not,
2	*, /, div, mod, and
3	+, -, or
4	=, < >, <, >, < =, > =, in

✍ *2.4.10. Записать назначение каждой операции, указать типы операндов и результата. Оформить в виде таблицы.*

✍ *2.4.11. Записать приоритет операций в языке Паскаль. Какие знаки используются для изменения приоритета?*

При работе с логическими переменными пользуются таблицами истинности.

A	B	A **and** B	A **or** B
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

A	**not** A
true	false
false	true

? Какие операции определены для переменных логического типа?

Какие операции используются для деления целых чисел? Чем они отличаются?

К какому типу ошибок (логическим, семантическим, синтаксическим) относится деление на 0? Почему?

✍ *2.4.12. Найти синтаксические ошибки в следующих выражениях:*

a) 'A'<'c'
b) *false>true*
c) '0' **or** '9'
d) '1'>'9'
e) 'A'<'B' **and** 1>2
f) ('d>z') **and** (1<2)
g) 1<=10
h) (10<10) **or** (10=10)
i) 12 **or** 15 **or** 18

✍ *2.4.13. Вычислить значения выражений, если A=2.5; B=7.8; C=-17.3; M=5; X=8.7; E= True.*

a) (A+B)/C*M
b) 2+X*X/(X+(A+B)/5)
c) (A<B) **and** (X+A<B) **or** (C<M) **or** E

✍ *2.4.14. Записать на языке Паскаль выражения, соответствующие указанным условиям:*

a) x∈[0;1];
b) x∈[2;5] или x∈[-1;1];
c) x∉[0;1);
d) x∉[2;5] и x∉[-1;1];
e) каждое из чисел x, y, z положительное;
f) хотя бы одно из чисел x, y, z положительное.

✍ *2.4.15. Чему будет равно значение выражения*

a) **not**(A **or** B **or** (X>2) **and** (Y<0)) при A = False, B = False, X = 3, Y = 2;
b) 15 **div** (16 **mod** 7);
c) 34 **mod** 3 * 5 – 29 **mod** 5 * 2;

2.4.5. Встроенные функции и процедуры

📖 При работе на ЭВМ используются встроенные *функции*, которые входят в состав *модулей* (библиотек) вычислительных комплексов (разработаны поставщиками программного обеспечения). Если их недостаточно, то можно создавать собственные (в соответствии с требованиями языка программирования).

Для каждой функции определены ее имя, синтаксис, количество и тип аргументов, а также тип результата.

Синтаксис – правила записи.

Аргументы – переменные или выражения, от которых зависит результат вычисления функции. Аргументы заключаются в круглые скобки, перечисляются через запятые. Существуют функции, не содержащие аргументов или имеющие нефиксированное их число.

Каждая функция обязательно должна иметь *единственный* результат.

Основной модуль *System* подключается к любой программе автоматически. В его состав входят *математические функции*, такие как Abs(x), Sqr(x), Sqrt(x), Sin(x), Cos(x) и другие.

ТАБЛИЦА ОСНОВНЫХ СТАНДАРТНЫХ ФУНКЦИЙ

Функция	**Назначение**	**Тип аргумента**	**Тип функции**
abs(*x*)	Вычисление абсолютного значения *x* (|*x*|)	Вещественный Целый	Вещественный Целый
sqr(*x*)	Вычисление квадрата *x* (x^2)	Вещественный Целый	Вещественный Целый
sin(*x*)	Вычисление синуса *x* (*sin x*) (*x* в радианах)	Вещественный Целый	Вещественный Вещественный
cos(*x*)	Вычисление косинуса *x* (*cos x*)	Вещественный Целый	Вещественный Вещественный
arctan(*x*)	Вычисление арктангенса *x* (*arctg x*)	Вещественный Целый	Вещественный Вещественный
exp(*x*)	Вычисление экспоненты *x* (e^x)	Вещественный Целый	Вещественный Вещественный
ln(*x*)	Вычисление натурального логарифма *x*	Вещественный Целый	Вещественный Вещественный
sqrt(*x*)	Вычисление квадратного корня из *x* ($\sqrt{x}$)	Вещественный Целый	Вещественный Вещественный
trunc(*x*)	Нахождение целой части *x*	Вещественный Целый	Целый Целый
round(*x*)	Округление *x* в сторону ближайшего целого	Вещественный Целый	Целый Целый
chr(*x*)	Определение символа по его порядковому номеру	Целый	Символьный

2.4.16. Привести примеры других функций, входящих в состав модуля System.

Например, e^x записывается с помощью функции exp(*x*) – экспонента от переменной *x*.

- ✓ *exp* – имя функции;
- ✓ *x* является аргументом функции целого / вещественного типа, вместо которого можно использовать число или выражение;
- ✓ результат вычислений будет вещественного типа.

Замечание: в большинстве языков программирования и в табличных редакторах π – это не число, а встроенная функция (*Pi*), которая не имеет аргументов и возвращает значение числа π, равного 3,14159265359.

Функции могут использоваться в выражениях и быть вложенными.

$f_1(f_2(x))$ – пример вложенной функции. Значение x является аргументом для f_2, а результат функции $f_2(x)$ является аргументом для функции f_1.

Пример: $\sin(\sqrt{1+e^{2x}})$ на Паскале запишется *sin(sqrt(1+exp(2*x)))*

🗐 Если `s` – символ; `i` – число от 0 до 255, то верны равенства:

1. Chr(Ord(s))=s; Ord(Chr(i))=i;
2. Pred(s)=Chr(Ord(s)-1); Succ(s)=Chr(Ord(s)+1);
3. Ord(false)=0; Ord(true)=1;
4. Succ(false)=true; Pred(true)=false.

? Объяснить приведенные равенства.

✍ *2.4.17. Определить имена функций, их аргументы, допустимые типы аргументов и результатов.*

a) ln(x)
b) cos(2*z)
c) 5*round(7.96)
d) sqrt(a)
e) sqr(n)
f) abs(k/2)

✍ *2.4.18. Определить функции и их аргументы. Какие из этих функций являются вложенными?*

a) Sin(1/sqr(x-1))
b) Cos(x)*Sin(y)
c) Sqrt(sqr(x)). Как иначе можно записать это выражение?

✍ *2.4.19. Определить, правильно ли записаны функции. Если неправильно, исправить ошибки.*

a) 5 Sqr
b) 10*Random
c) sQr(–5)
d) ABS x
e) Pred(3.2)
f) eXp(N)
g) sin(2X)
h) TG(Y)
i) abs(A)
j) Succ('b')
k) LN(LN(S))
l) 2cos(x)
m) siN(3,5)
n) ArcTan(100)
o) SqrT(-3)

✍ *2.4.20. Определить тип результата выражений, если известно, что I, J, K: INTEGER; X, Y, Z: REAL.*

a) Sqr(I)+ Sqr(J*J)+2*K/2
b) Sin(X)+2*Cos(Y)+Z
c) Round(X)<Y
d) I+Sqrt(J)

🗐 *Примеры записи некоторых математических функций, не являющихся встроенными.*

1. Для записи степени x^y используются следующие преобразования: т.к. $b = e^{\ln b}$, тогда $x^y = e^{\ln x^y} = e^{y \ln x}$. На языке Паскаль: `Exp(y*Ln(x))`.
2. Корень, отличный от квадратного, сначала следует записать в виде степени: $\sqrt[n]{x} = x^{1/n}$. На языке Паскаль: `Exp(1/n*Ln(x))`.
3. Так как $tgx = \frac{\sin x}{\cos x}$, поэтому tg(x) на языке Паскаль записывается: `sin(x)/cos(x)`.
4. Логарифм с произвольным основанием можно выразить: $\log_n x = \frac{\ln x}{\ln n}$ и записать на языке Паскаль в виде выражения: `ln(x)/ln(n)`.

📖 Для повторения однотипных действий в разных частях одной программы или в разных программах используются *процедуры*. По записи (при использовании и описании) они похожи на функции, но в отличие от них, не используются в выражениях и не возвращают результаты.

Модуль *CRT* содержит процедуры и функции, которые задают дополнительные возможности экрана: чистка экрана, цвет фона, цвет символов и т.д.

✍ *2.4.21. Перечислить основные функции и процедуры модуля CRT и их назначение.*

✍ *2.4.22. Определить константы и переменные, входные и выходные параметры в формулах:*

a) $y = \sin^2\frac{\pi}{16}x + \sin^2\frac{3\pi}{16}x + \sin^2\frac{5\pi}{16}x;$

b) $c = \sqrt{a^2 + b^2}$, где *a, b, c* – стороны прямоугольного треугольника;

c) плотность Земли на расстоянии r от центра $\rho(r) = \rho_0 - \frac{4}{3R_з}(\rho_0 - \rho_c)r$, где $R_З$ – средний радиус Земли, ρ_0 и ρ_c – плотности в центре Земли и среднее значение (соответственно);

d) $h = \frac{gt^2}{2}$, где t – время падения, h – высота, g – ускорение свободного падения.

? Вопросы для повторения.

1. Что такое данные? На какие группы они делятся?
2. Чем отличаются понятия: "переменная x" и "значение переменной x"?
3. Что такое операции над данными?
4. Для чего определяют типы данных?
5. Что такое функция? Какие параметры должны быть определены для функции?

2.4.6. Ввод и вывод данных

Для составителя программы важны типы переменных, а не их конкретные значения, которые определяются пользователем во время выполнения программы. Записать или вычислить значение выражения еще недостаточно для решения задачи, так как все вычисления производятся в оперативной памяти и надо их оттуда извлечь. Поэтому при программировании очень важная роль отводится использованию в программах процедур ввода-вывода данных.

📖 *Ввод данных* – это передача информации от пользователя или внешнего носителя в оперативную память.

Вывод данных – это обратный процесс, когда данные после обработки из оперативной памяти передаются на экран монитора, принтер или на внешний носитель.

? Какая процедура (ввод или вывод) обязательно присутствует в любой программе?

Процедуры ввода и вывода допускают использование произвольного числа аргументов (*параметров*), которые передаются в виде списка, записанного в круглых скобках.

По умолчанию на языке Паскаль ввод данных выполняется с клавиатуры, вывод – на экран монитора. Иначе следует указывать устройства ввода или вывода.

`Read(x1,x2,...,xn)` – ввод значений переменных `x1,x2,...,xn` пользователем с клавиатуры;

`ReadLn(x1,x2,...,xn)` – ввод, отличается от `Read` тем, что после ввода значений всех переменных считывается нажатие клавиши ENTER.

`Write(y1,y2,...,yn)` – вывод значений `y1,y2,...,yn` компьютером на экран монитора.

Примечание 1. В качестве аргументов процедуры `Read` можно использовать только имена переменных, значения которых набираются через пробел (с клавиатуры) после запуска программы на выполнение. Признаком конца ввода является нажатие клавиши ENTER.

Примечание 2. В качестве аргументов процедуры `Write` могут использоваться выражения и строковые константы. Числа и переменные является частными случаями выражений. Данная процедура может содержать как один из указанных аргументов, так и несколько.

🗐 *Пример.*
```
Write('нет');
Write(2*x+5);
Write('кол-во = ',k,'  S=',S:4:1);
```
`WriteLn(y1,y2,...,yn)` – вывод, отличается от `Write` тем, что после вывода значений аргументов курсор перейдет на следующую строку.

Примечание 3. Для приостановки работы программы используется `ReadLn` без аргументов (ожидается нажатие ENTER). `Read` без аргументов не несет никакой смысловой нагрузки.

Примечание 4. Можно использовать форматный вывод. Для этого после выводимого параметра ставится ":" и указывается число позиций. Для данных вещественного типа дополнительно можно указать еще ":" и количество десятичных знаков числа.

✍ *2.4.23. Какие процедуры ввода записаны без ошибок?*
```
Read('a'); Read(a,c); Read('Введите a',a); ReadLn(a);
```
✍ *2.4.24. В результате выполнения каких процедур будет выведено на экран значение переменной* x*?*
```
Write(x); WriteLn('x'); WriteLn(x); Write('x');
```

🗐 *Примеры.*

1. Если переменные `a,b,c: Byte` и в программе используются ввод-вывод
```
Write('Введите a, b, c через пробелы');Read(a,b,c);
Write('a=',a);Write('  a+b=',a+b);WriteLn('  b:=',b);
```
то результат выполнения этой части программы будет иметь вид:
```
4 5 10
a=4  a+b=9  b:=5
```
2. Пусть `a=3` и `b=12`.
Операторы `Write(a,'   ',b);` и `Write(a,b:5);`

выполняют одинаковые действия. Результатом выполнения будет вывод
`3  12` (между числами три пробела).

3. Пусть `x=3.1245774011` и в программе используется вывод `WriteLn('x=',x:6:3);` результатом выполнения будет `x= 3.125`.
4. а при `WriteLn('x=',x:1:2);` результатом выполнения будет `x=3.12` (1 – количество позиций, отводимых под число, 2 – количество десятичных знаков. Так как 1<4 (кол-ва знаков в числе `x`), то автоматически выводится вся целая часть и 2 десятичных знака),

✍ *2.4.25. Какие из приведенных фрагментов программ выполняются одинаково:*

a) `Read(a, b);`
b) `Read(a); Read(b);`
c) `ReadLn(a, b);`
d) `ReadLn(a); ReadLn(b)`?

✍ *2.4.26. Известно, что x:=5.1268, M=3. Что будет выведено на экран при выполнении процедур:*

a) `Write(x);`
b) `WriteLn('x');`
c) `Write('x=',x:6:2);`
d) `WriteLn('x='); WriteLn(x:6:2);`
e) `Write('A':3,2:1);`
f) `WriteLn(5*3.2);`
g) `WriteLn(5<6,5>6:8);`
h) `WriteLn('4*M+1')`?

? Вопросы для повторения.

1. Что называется идентификатором? На какие группы они делятся?
2. Какие типы данных используются в языке Паскаль?
3. Какие существуют операции для работы с данными?
4. Что называется функциями и процедурами?
5. Как записывается процедура ввода? Какие данные можно вводить?
6. Как записывается процедура вывода? Какие данные можно выводить?

2.5. АЛГОРИТМИЧЕСКИЕ СТРУКТУРЫ ЯЗЫКА ПАСКАЛЬ

Далее вместо слов "текст программы на языке Паскаль" будет использоваться термин "программа".

📖 *Программа* (с точки зрения программирования) – это формальная запись некоторого алгоритма. Каждая конструкция программы имеет точно определенную семантику. Все действия алгоритма записываются с помощью специально предназначенных ключевых слов и называются *операторами*. Классификация операторов приведена на рис.18.

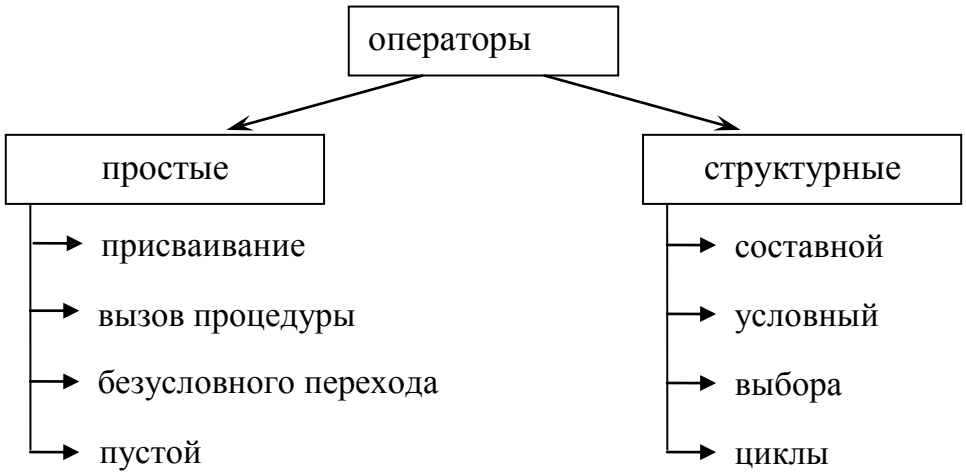


Рис. 18. Классификация операторов

2.5.1. Структура программы на языке Паскаль

📖 Программа на языке Паскаль *состоит из* заголовка, описательной и исполнительной частей.

program имя_программы;	*Заголовок* является необязательной частью, содержит имя программы, может содержать имена стандартных устройств ввода (`Input`) вывода (`Output`).
Описательная часть не связана с какими-либо конкретными действиями при работе программы, сообщает компилятору некоторые сведения о самой программе и использующихся в ней объектах	
uses модули;	подключение *модулей*;
label метки;	описание *меток*;
const константа1=значение1; константа2=значение2;	описание *констант* с указанием их значений;
type тип_пользователя= стандартный_тип;	описание *типов*, созданных программистом на основе стандартных типов;
var список_переменных1**:**тип1**;** список_переменных2**:**тип2**;**	описание переменных с указанием их типов;
Исполнительная часть является обязательной для работы программы. Имеет другие названия: раздел операторов, основная часть.	
begin *Операторы*; **end.**	действия из алгоритма

Примечание 1. Выделенные слова являются ключевыми (служебными, зарезервированными), их переопределить нельзя.

После каждого раздела или оператора ставится знак "точка с запятой".

Программа может содержать *Комментарии*, которые служат для пояснения программы или отдельных ее частей. Комментарии заключаются в скобки (* *) или { } и воспринимаются программой как пробел.

В описательной части используются только нужные для конкретной программы разделы, ключевые слова которых используются только один раз.

Последовательно записанные операторы и другие конструкции программы по синтаксису могут быть расположены на одной строке, однако для удобства чтения и определения их семантики используется ступенчатая форма записи (см. примеры программ).

🗐 *Пример описания типа:*

```
type Animals=(cat, got, mouse, horse);
   figure='0'..'9';
```

? Какие типы данных описаны?

🗐 *Пример описания констант:*

```
const S='s'; min=-100; f=false;
```

? Какой тип имеют константы `S, min, f` и выражения `S<='s', 2.5*min`?

🗐 *Пример* описания *переменных:*

```
var x: Real; y: Real; z: Real; b: Boolean;
```

другой вариант описания тех же переменных:

```
var x,y,z: Real; b: Boolean;
```

? Назвать имена переменных и их типы.

Примечание 2. Использовать константу без указания ее значения нельзя. Если описывается несколько переменных, то они разбиваются на группы, в которых однотипные переменные перечисляются через запятую (списком).

✍ *2.5.1. Описать две константы: a вещественного типа и k целого типа.*

✍ *2.5.2. Описать две переменные целого типа и одну диапазонного типа.*

Основные действия, которые должны содержаться в *исполнительной* части программы:

1. *ввод* (или *присваивание*) значений переменных, относящихся к исходным данным. Иначе их значения будут взяты из памяти компьютера и будут неправильными, что приведет к неверному результату (к сожалению, компьютер не может сканировать наши мысли);
2. *вычисление* значений промежуточных переменных (если они есть) и результатов,
3. *вывод* результатов работы (иначе они так и останутся в памяти компьютера, заглянуть в которую мы тоже не можем).

🗐 *Пример программы.*

Известен радиус круга, вычислить его площадь.

Обозначим R – радиус круга (исходные данные). Площадь круга (результат) вычисляется по формуле πR^2.

Программа на языке Паскаль	Назначение строк (семантика)
program sq;	– программа имеет имя sq, которое отличает ее от других программ;
(*Вычисление площади круга*)	– комментарии к программе;
var R,S: real;	– описаны две переменные R и S вещественного типа;
begin	– начало действий;
Write('Введите радиус круга ');	– вывод сообщения-подсказки;
Read(R);	– ввод значения переменной R;
S:=Pi*Sqr(R)	– вычисление площади;
WriteLn('Площадь круга равна ', S:2:2);	– вывод результата;
end.	– конец программы.

Контрольные примеры:

Если R=0, то площадь круга равна 0. Если R=1, то площадь круга ≈3.14.

💻 *Запустить среду программирования Паскаль, набрать текст программы и выполнить ее прогон при значениях R, равных 0; 1 и 5.*

? Совпадают ли результаты, полученные при выполнении программы, с контрольными примерами?

В программах могут быть использованы *вспомогательные* действия, которых в алгоритмах, как правило, нет. Их назначение – создание *интерфейса* программы:

1. организация *диалога* пользователя и программы (например, вывод подсказок, запрос значений переменных);
2. изменение внешнего вида (например, цвет фона или символов, окно для ввода-вывода).

В рассмотренном примере `Write('Введите радиус круга ');` относится к организации интерфейса (если его не использовать, то алгоритм не нарушится, но он предназначен для устранения затруднений при работе с программой).

Вместо текста `'Введите радиус круга '` можно использовать другой, например, `'Введите R='`, это зависит от смысловой нагрузки, которую хочет передать составитель программы.

Возможен и такой вариант, когда часть оператора отвечает за обеспечение интерфейса, а другая часть – за выполнение основных действий.

Например, `WriteLn('Площадь круга равна ', S:2:2);` – выводит сообщение `'Площадь круга равна '`, относящееся к интерфейсной части, и значение переменной `S`, относящееся к основной части.

🗐 *Вывести графический текст, состоящий из символов.*

```
program pscl;
uses Crt;
begin
```

```
  ClrScr;
  TextColor(Green);
  GotoXY(15,5);
  Write('Для продолжения работы нажмите клавишу <ENTER>');
  GotoXY(12,12); Write('***    *    ***   **    *    *  ');
  GotoXY(12,13); Write('*  *  * *   *    *     * *   *  ');
  GotoXY(12,14); Write('***   ***     *  *     ***   *  ');
  GotoXY(12,15); Write('*    *   *  ***   **  *   *  ***');
  ReadLn;
end.
```

💻 *Ввести текст программы и дополнить его операторами для формирования интерфейса (например, цвет фона и символов).*

? Вопросы для повторения.

1. Что представляет собой программа на языке Pascal? Какие разделы она имеет?
2. Для чего предназначены основные команды и вспомогательные действия?
3. Что такое оператор? На какие группы можно разделить все операторы?

2.5.2. Оператор присваивания

📖 Оператор присваивания предназначен для изменения значения переменной без участия пользователя. Он предписывает вычислить значение выражения, заданного в его правой части, присвоить результат переменной, расположенной в левой части (записать в эту переменную значение выражения). Общий вид оператора присваивания:

переменная:=*выражение*;

? Как связаны между собой типы переменной и выражения? Какие допускаются исключения?

✍ *2.5.3. Чему будет равна переменная a после выполнения операторов присваивания:*

a) `b:=6; c:=4; a:=b/3+2*c-1;`
b) `a:=1; a:=2+a; a:=a*a;`
c) `b:=5; a:=b; a:=a+b;`
d) `a:=5; b:=15; c:=-b; a:=c?`

Какой тип могут иметь переменные?

✍ *2.5.4. Какую серию операторов можно использовать для вычисления значения выражения* $\left(\frac{x+y}{x-y}-\frac{x-y}{x+y}\right)\cdot\left(\frac{x-y}{x+y}+\frac{x+y}{x-y}\right)$*?*

a) `A:=(x+y)/(x-y); B:=(A-1/A)*(1/A+A);`
b) `A:=(x+y)/(x-y); B:=Sqr(A)-Sqr(1/A);`
c) `A:=(x-y)/(x+y); B:=Sqr(1/A)-Sqr(A).`

✍ *2.5.5. Переписать операторы присваивания, исправив возможные ошибки:*

a) `M:=1-2*{B+N};`
b) `T:=((11-y));`
c) `K+100;`
d) `8*X:=P;`
e) `Y:=Y*X;`
f) `Y:=Y*2X;`
g) `Д:=1;`
h) `k:=232R;`
i) `g:=g;`
j) `s:=l:=u;`
k) `M:=2*M;`

✍ *2.5.6. Вычислить значения переменных после выполнения приведенных фрагментов программ:*

a)
```
x:=3;
y:=x*x+x*2;
x:=y-x;
x:=x+y;
```

b)
```
x:=1;   y:=x;
x:=x+1; y:=y+x;
x:=x+1; y:=y+x;
x:=x+1; y:=y+x;
```

✍ *2.5.7. Определите, какие задачи выполняются с помощью следующих серий операторов:*

a)
```
Read(x);
p:=sqr(x);
p:=sqrt(p);
WriteLn(p);
```

b)
```
Read(x);
k:=x*x;
k:=k*k;
k:=x*k;
WriteLn(k);
```

✍ *2.5.8. Что будет выведено на экран в результате выполнения каждой программы?*

a)
```
program S1;
var x,y: Integer;
begin
 x:=2;
 y:=3;
 x:=x*x;
 y:=y*y;
 x:=x+y;
 WriteLn('x=, x');
end.
```

b)
```
program S2;
var x,y: Integer;
begin
 x:=2;
 y:=0.5*x;
 x:=x*x;
 y:=y*y*x*x;
 x:=x+y;
 WriteLn('x=', x);
end.
```

? Найти и исправить ошибки (синтаксические и логические) в программах.

✍ *2.5.9. Записать ввод исходных данных, которые используются в операторе присваивания*:

```
y:=sin(pi*x)-e*x.
```

🗐 *Дана площадь равностороннего треугольника. Найти его периметр.*

Входные параметры: S – площадь равностороннего треугольника;
Выходные параметры: P – периметр этого треугольника.
Используемые формулы:

$$S = \frac{a^2\sqrt{3}}{4} \text{ и } P=3a, \text{ отсюда } P = 3\sqrt{\frac{4S}{\sqrt{3}}}.$$

? Определить метод решения задачи.

Алгоритм в виде блок-схемы:

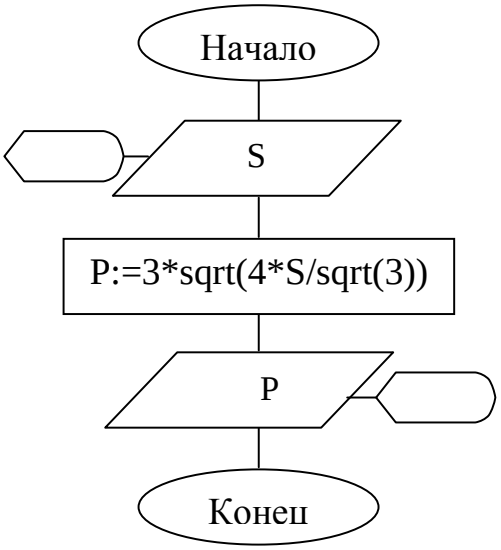


Рис. 19. Пример блок-схемы линейного алгоритма

Программа на языке Паскаль

```
program S_tr;
  {вычисление периметра равно-
 стороннего треугольника}
var  S,P: Real;
begin
 WriteLn('Введите площадь ',
   'треугольника:');
 Write('S=');
 ReadLn(S);
 P:=3*Sqrt(4*S/Sqrt(3));
 WriteLn('Периметр треуг-ка');
 WriteLn('P=', P:1:2);
 ReadLn
end.
```

💻 *Ввести программу, проверить ее работу на контрольных примерах (сравнить результаты вычислений на калькуляторе и после выполнения программы).*

✍ *2.5.10. В прямоугольном треугольнике известны длины его катетов. Вычислить углы треугольника (составить блок-схему и программу).*

Входные параметры: a, b – стороны прямоугольного треугольника;

Выходные параметры: α, β – углы треугольника.

Используемые формулы: $\alpha = arctg\frac{b}{a}$ и $\beta = \frac{\pi}{2} - \alpha$.

В алфавите языка Паскаль нет символов α, β, поэтому заменим их другими идентификаторами, например, a1, b1.

💻 *Ввести программу, проверить ее работу на контрольных примерах. Вычислить длину гипотенузы.*

? Определить условия допустимости исходных данных.

✍ *2.5.11. Составить программу для нахождения значения функции:*

1. $z = \sin^2(xy + e^x) + \sqrt[3]{1-x}$;
2. $y = a^{\cos x} - \log_2(2x^2 + 3)$, где a=5.

? При всех ли наборах исходных данных можно определить результаты? Универсальны ли данные программы?
Какого типа алгоритмы использованы?

2.5.3. Условные операторы

📖 Условные операторы предназначены для записи разветвляющихся алгоритмов. Имеют два варианта представления:
сокращенную форму: **if** *условие* **then** *оператор*;

полную форму: **if** *условие* **then** *оператор1* **else** *оператор2*;

где *условие* – выражение (или переменная) *логического* типа;

операторы могут быть простыми или структурными.

Ключевые слова: **if** (если), **then** (тогда), **else** (иначе).

Примечание 1. Перед словом **else** точка с запятой не ставится, так как оператор в этом месте не заканчивается.

Примечание 2. После слов **then** и **else** по синтаксису можно записать только по одному оператору. Если по логике решения задачи операторов должно быть больше, чем один, то они заключаются в операторные скобки

begin … **end** (объединяются в *составной* оператор).

Примечание 3. Условные операторы могут быть вложенными. Допускается 7 уровней вложенности. Каждое слово **else** относится к ближайшему **if**. Второе и последующие условия удобнее проверять после слов **else**.

? Объяснить принципы работы условных операторов.

✍ *2.5.12. Какое значение примет переменная x после выполнения каждого фрагмента программ:*

a)
```
if a<=b then x:=a+b
        else x:=2*a-b*4;
```
при a=53, b=14;

b)
```
if a>b then x:=a+b
       else
        begin
         a:=b+c;
         x:=a-3*b
        end;
```
при a=5, b=4, c=3;

c)
```
if (a>b)and(b>c) then
   x:=2*a+b+c
     else x:=a-b+c;
```
при a=9, b=8, c=2;

d)
```
if (a<=b)or(a>c) then
   x:=5*a-b
     else x:=4*a*b;
```
при a=1, b=4, c=8.

✍ *2.5.13. Переписать условные операторы, исправив ошибки:*

a) `if x<y then x:=0 else y:=0;`

b) `if x>=y then x:=0; y:=0; else Write (z);`

c) `if a<>b else s:=s+1;`

d) `if Write(x) then Inc(x);`

e) `if x then y:=6-x else z:=2;`

f) `if x>0 then y;`

✍ *2.5.14. Известно, что a,b: Boolean; x,y: Real. Найти синтаксические ошибки в условных операторах:*

a) `if a then WriteLn('Результат ', x<y) else x:=y+b;`

b)
```
if 5=3 then Write('введите два числа ') Read(x,y)
       else WriteLn(5<>3);
```

✍ *2.5.15. Записать вывод данных, полученных в условном операторе:*
```
if x<0 then z:=z-x else z:=0;
```

Примеры использования условных операторов.

🗐 *Пример 1.* Если вещественное число *x* является отрицательным, то заменить его противоположным.

```
if x<0 then x:=-x;
```

Пример 2. Даны два вещественных числа *x* и *y*. Вывести первое число, если оно меньше второго, иначе вывести оба числа.

```
if x<y then WriteLn(x) else WriteLn(x,y);
```

Пример 3. Определить, есть ли среди чисел *x*, *y* и *z* положительное число.

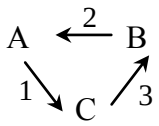
```
if (x>0)or(y>0)or(z>0)
  then WriteLn('Есть положительное число')
    else WriteLn('Нет положительных чисел');
```

Пример 4. Даны вещественные числа *a* и *b*. Упорядочить эти числа таким образом, чтобы выполнялось неравенство *a*<=*b*.

Значения двух переменных меняются через третью (вспомогательную) переменную.

```
if a>b then
  begin
   c:=a; a:=b; b:=c
  end;
```

Механизм замены (цифрами обозначен порядок действий):



Пример 5. Дано целое число *a*. Определить и вывести, четное или нечетное это число.

Число четное, если остаток от деления этого числа на 2 равен 0.

```
if n mod 2=0 then WriteLn (n,' - четное')
   else WriteLn (n,' - нечетное');
```

Пример 6. Даны вещественные числа *a* и *b* (*a*≠*b*). Меньшее из этих чисел заменить их полусуммой, а большее – их удвоенным произведением.

```
c:=(a+b)/2;
d:= 2*a*b;


if a<b then
  begin
   a:=c; b:=d
  end

else
   begin
    b:=c; a:=d
   end;
Writeln('a=',a,
' b=',b);
```

– переменной *c* присвоили значение полусуммы
– переменной *c* присвоили значение полусуммы
– переменной *d* присвоили значение удвоенного произведения
– если условие a<b принимает значение «истина», тогда
– произвели замену
(слова BEGIN и END объединяют два оператора присваивания в один составной)
– иначе
– произвели замену
– противоположное условие (b<a) не проверяли
– вывели результаты

Пример 7. Вычислить значение функции $y = \begin{cases} \sqrt{|x^2-1|}, & если\ -1 < x < 1, \\ 1/x, & если\ x \le -1\ или\ x \ge 1. \end{cases}$

Переменная *x* может принимать любое значение, условия взаимно дополняют друг друга и не пересекаются. Значение функции *y* зависит от *x*: если первое условие $-1 < x < 1$ принимает значение *true*, то $y = \sqrt{|x^2-1|}$, иначе $y = \frac{1}{x}$ (первое

условие принимает значение *false*, второе – принимает значение *true*, это условие проверять не нужно).

```
if (-1<x)and(x<1) then y:=Sqrt(Abs(x*x-1)) else y:=1/x;
Writeln('зн.функ=', y);
```

Пример 8. Найти наибольшее из двух чисел *x* и *y*.

Результат занесем в переменную *max*. Если данные числа окажутся равными, то значение любого из них можно считать наибольшим.

```
if x>y then max:=x else max:=y;
Writeln('наибольшее значение =',max);
```

2.5.16. Записать условные операторы, используемые для решения задач.

a) Дано целое число N. Определить, делится ли это число на 5 без остатка.
b) Даны два числа *a* и *b*. Проверить, являются ли они взаимно противоположными.
c) Даны три числа *a, b, c*. Найти наименьшее из них.
d) Даны три числа *a, b, c*. Проверить, могут ли они быть сторонами прямоугольного треугольника.
e) Даны два числа *m* и *n*. Проверить, имеют ли они одинаковый знак (+ или −).
f) Является ли данное целое число *z* однозначным?

Пример 9. Определить количество корней квадратного уравнения $x^2 + bx + c = 0$.

Дискриминант этого уравнения вычисляется по формуле $D = b^2 - 4c$. Если $D < 0$, то корней нет, если $D = 0$, то корень один, иначе два корня.

```
D:=b*b-4*c;
IF D<0 THEN WriteLn('Корней уравнения нет') ELSE
   IF D=0 THEN WriteLn('Уравнение имеет один корень')
     ELSE WriteLn('Уравнение имеет два корня');
```

? Найти логическую ошибку в следующей записи (при каких значениях D получается неправильный результат):

```
D:=b*b-4*c;
IF D<0 THEN WriteLn('Корней уравнения нет');
IF D=0 THEN WriteLn('Уравнение имеет один корень')
  ELSE WriteLn('Уравнение имеет два корня');
```

2.6.17. Определить значение переменной z после выполнения операторов

```
z:=0;
IF x>0 THEN z:=1 ELSE
   IF y>0 THEN z:=2;
```

если значения переменных x и y:

a) x=1, y=1;
b) x=1, y=-1;
c) x=-1, y=1;
d) x=-1, y=-?

✍ *2.5.18. Какая задача решается в следующей программе?*

```
var a,b,c,d,m: Integer;
begin
 ReadLn(a,b,c,d);
 if a>b then m:=a else m:=b;
 if m<c then m:=c;
 if m<d then m:=d;
 WriteLn(m);
end.
```

✍ *2.5.19. Составить программу для вычисления значения функции:*

$$f(x)=\begin{cases}\sin^2 x & \text{при } x\le 0,\\ \left(x-\frac{2x}{3}\right)^2 & \text{при } 0<x\le 1,\\ e^x+\cos\pi x^2 & \text{при } x>1.\end{cases}$$

Входные параметры: ***x*** – вещественное число;

Выходные параметры: ***f*** – значение функции (вещественное число).

В зависимости от введенного значения переменной ***x*** значение функции вычисляется по одной из формул:

если $x\le 0$, то ***f*** вычисляется по первой формуле, иначе

если $x\le 1$, то ***f*** вычисляется по второй формуле,

иначе – по третьей формуле.

? Почему не нужно проверять условия: ***x*>0** и ***x*>1**?

Алгоритма в виде блок-схемы

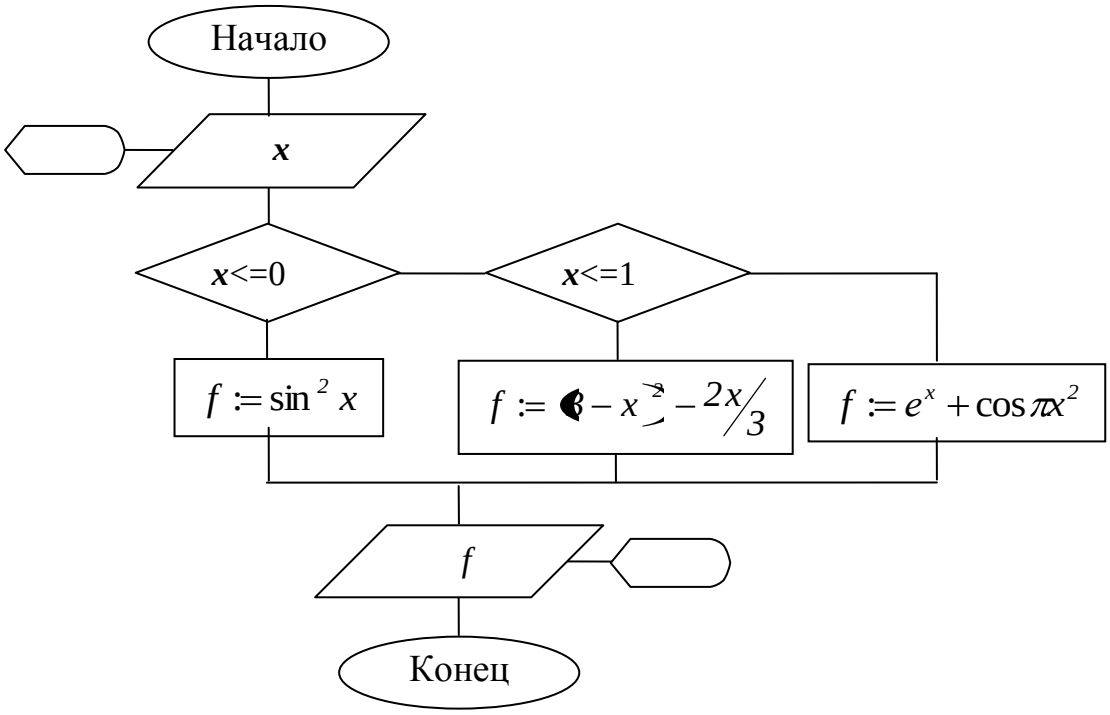


Рис. 20. Пример блок-схемы разветвляющегося алгоритма

Вставить в текст программы значения функции. Дополнить процедурами из модуля CRT.

```
PROGRAM FUNC;
USES Crt;
VAR x,f: ;
BEGIN
 ClrScr;
 Write(' Введите число x=');
 ReadLn(x);
 IF x<=0 THEN f:=
    ELSE
     IF x<=1 THEN f:=   ELSE f:=    ;
 WriteLn ('  f(x)=',f);
 ReadLn
END.
```

💻 *Проверить работу программы, выполнив контрольные примеры.*

✍ *2.5.20. Вычислить значение функции, учитывая область определения, иначе вывести сообщение вида: 'Значение аргумента не входит в область определения'.*

a) $y = \dfrac{a+3}{x-2}$; b) $y = \dfrac{\ln x}{\sqrt{x-1}}$.

? Вопросы для повторения.

1. Что такое условие?
2. Какие операции используются для написания логических выражений?
3. Какую структуру имеет условный оператор в языке Паскаль?

2.5.4. Оператор выбора

📖 *Оператор выбора* используется для записи разветвляющихся алгоритмов, но область его применения (в отличие от условного оператора) ограничена. С его помощью можно повысить наглядность программы в том случае, когда используется много вариантов проверки. Оператор выбора имеет следующую структуру:

case *Выражение_выбора* **of**
  Список_констант1: оператор1;
  Список_констант2: оператор2;
  {и т.д.}
  СписокконстантN: операторN
  else *операторN+1*
end;

где *выражение_выбора* может быть любого порядкового типа;
часть **else** *оператор N+1* является необязательной;
служебные слова **case** (выбор), **of** (из), **else** (иначе),
end (конец выбора).

Замечание. Все возможные значения *Выражения_выбора* должны быть ограничены и распределены по *Спискам_констант* или должны относиться к части **else**.

? Объяснить принцип работы оператора выбора.

По введенному номеру месяца определить и вывести на терминал название времени года.

Входные параметры: a – целое число;

Выходные параметры: название времени года.

Алгоритм в виде блок-схемы приведен на рис. 20.

? При каких исходных данных можно получить результат?

Определить метод решения.

Какие данные следует дописать в свободные блоках?

Каких вспомогательных блоков не хватает?

Объяснить программу.

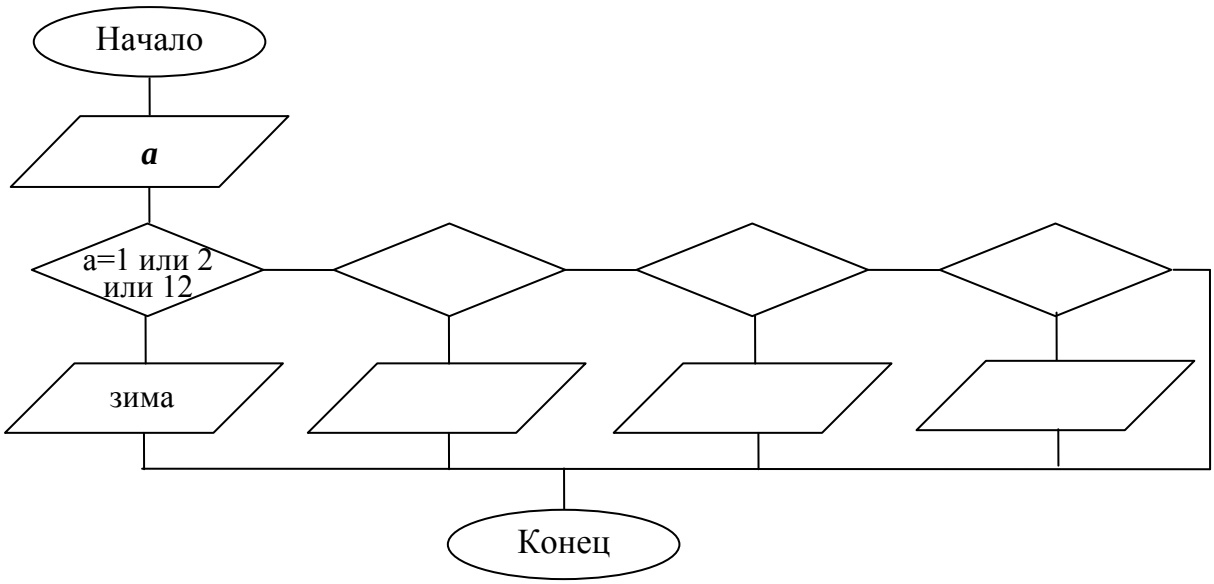


Рис. 20. Пример блок-схемы разветвляющегося алгоритма

Программа на языке Паскаль

```
program Vr_Goda;
var a: Integer;
begin
 Write('  Введите номер месяца:  '); ReadLn(a);
 case a of
  1,2,12: WriteLn(' Время года - зима');
  3..5:  WriteLn(' Время года - весна');
  6..8:  WriteLn(' Время года - лето');
  9..11: WriteLn(' Время года - осень');
  else WriteLn(' Номер месяца введен неправильно!');
 end;
end.
```

2.5.21. Изменить программу таким образом, чтобы по введенному номеру месяца определять номер квартала.

✍ *2.5.22. Составить программы для решения задач, используя оператор выбора.*

a) По введенному номеру дня недели определить рабочий день или выходной.
b) По введенной температуре воды определить ее агрегатное состояние (лед, жидкость или пар).
c) По введенной температуре воздуха определить погоду (холодно, тепло, жарко и т.п.)

2.5.5. Операторы циклов

Операторы циклов используются для записи в программе повторяющихся действий. Бывают трех типов: цикл-счетчик, цикл с предусловием и цикл с постусловием.

🕮 Цикл-счетчик (цикл с параметром или с известным числом повторений) на промежутке [*нач_значение; кон_значение*] имеет два варианта записи:

for *переменная* := *нач_значение* **to** *кон_значение* **do** *оператор;*
for *переменная* := *кон_значение* **downto** *нач_значение* **do** *оператор;*
Пример: **for** i := 1 **to** n **do** S:=S+i ;
for i := n **downto** 1 **do** S:=S+i ;

Служебные слова **for** (для), **to** (до), **do** (делать), **downto** (после);
переменная, называемая также *параметром*, может быть любого порядкового типа;
типы *переменной*, *нач_значения* и *кон_значения* должны совпадать;
оператор, используемый после слова **do**, является "телом цикла";
служебное слово **to** показывает, что шаг изменения *переменной* равен 1;
служебное слово **downto** показывает, что шаг равен −1.
Условие работы цикла **for** скрыто в служебном слове **to**, в котором проверяется «*переменная* **до** *кон_значения ?»* (i<=n), в слове **downto** проверяется «*переменная* **после** нач*_значения ?»* (i>=1).

? Каким образом происходит выполнение цикла-счетчика?

Изменения значения переменной-параметра внутри цикла считается грубой логической ошибкой, так как она может привести к неправильной работе компилятора. Для досрочного выхода из цикла можно использовать оператор `break` (хотя и нежелательно).

✍ *2.5.23. Сколько раз выполнится тело оператора цикла:*

a)
```
z:=0; n:=1;
for j:=1 to n do
  z:=z+Exp(j);
```

b)
```
for k:=1 to -5 do
   Write('*');
```

c)
```
for t:=-10 to -3 do
   WriteLn(Sqr(t));
```

d)
```
z:=0;
for i:=-5 to 5 do
  z:=z+Abs(i);
```

e)
```
k:=0;
for i:=1 to k+3 do
  k:=k+1;
```

f)
```
for k:=10 downto 1 do
   Write (k,' ');
```

✍ *2.5.24. Переписать фрагменты программ, исправив синтаксические и логические ошибки, если известно, что K,I :Integer; X,Y :Real:*

a)
```
y:=0;
for x:=0.1 to 0.9 do
    y:=y+Sin(x);
```

b)
```
k:=81; y:=1;
for i:=1 to sqrt(k) do
    y:=y*2;
```

c)
```
k:=0; x:=5;
for i:=1 to x do
 begin k:=Sqr(i)+k;
       k:=k+1;
 end;
```

d)
```
for k:=-12 downto 0 do
  ReadLn(k);
```

✍ *2.5.25. Что будет выведено на экран после выполнения фрагмента программы:*

a)
```
for a:=1 to 9 do
   Write('a':2);
```

b)
```
for a:=9 downto 0 do
   Write(a:2);
```

c)
```
for x:='a' to 'z' do
   WriteLn(x);
```

d)
```
for i:=10 downto 8 do
   Write(i+1,'  ');
```

e)
```
p:=1; n:=5;
for i:=1 to n do
   p:=p*i;
 WriteLn(p);
```

Операторы циклов могут быть вложенными, при этом параметры циклов обязательно должны иметь разные имена. Например:

```
for i:=1 to 5 do
 for j:=1 to 10 do  WriteLn(i,j:3);
```

✍ *2.5.26. Дополнить фрагмент необходимыми разделами, чтобы получилась программа. Проверить ее работу на компьютере.*

? Какой параметр `i`, или `j`, будет изменяться быстрее?

Определить принцип работы вложенных циклов.

Примеры использования цикла-счетчика.

🗐 *Пример 1*. Вычислить N!

Входные параметры: N – натуральное число;

Выходные параметры: P – факториал числа N;

По определению N!=1*2*3*…*N, где N – натуральное число. Отсюда

1!=1

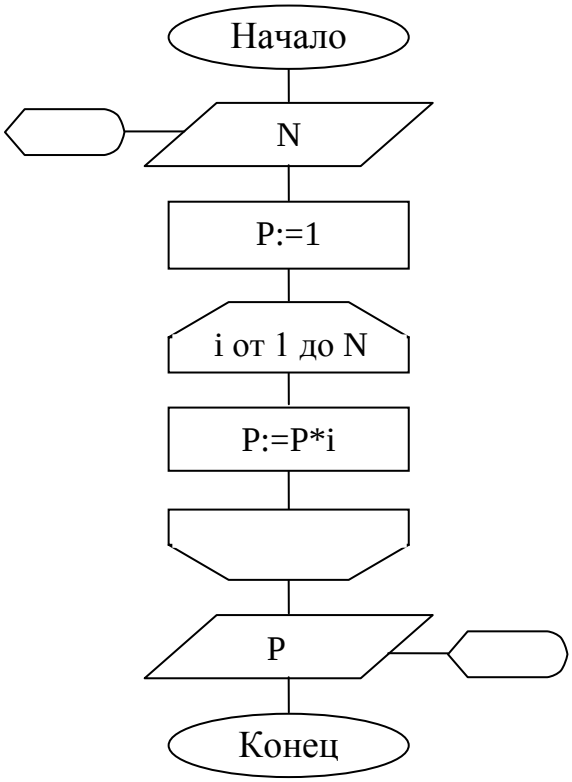
2!=1*2=2

3!=1*2*3=6 и т.д.

Иначе $S = \prod_{i=1}^{N} i$.

С точки зрения математики произведение натуральных чисел – это число натуральное. Значение переменной P будет очень быстро увеличиваться (для примера 8!=40320, 9!=362880, 10!=3628800 и т.д.). Но при составлении программы все используемые типы данных имеют ограничения, поэтому для описания переменной P лучше использовать вещественный тип, но при выводе отформатировать как целое число (без десятичных знаков). Однако и в этом случае не при любых значениях N можно вычислить результат (так как происходит выход за диапазон вещественных чисел).

Алгоритм в виде блок-схемы



Программа на языке Паскаль

```
PROGRAM Fact;
    {Вычисление N!}
USES Crt;
VAR N, I: Integer;
    P: Real;
BEGIN
 ClrScr;
 Write('Введите нат. число N=');
 ReadLn(N);
 P:=1;
 FOR I:=1 TO N DO P:=P*I;
 Write(' N!=',P:1:0);
 ReadKey
END.
```

💻 *Ввести текст программы, выполнить прогонку программы и определить условия допустимости исходных данных.*

✍ *2.5.27. Изменить программу таким образом, чтобы в ней использовался цикл-счетчик с шагом изменения аргумента -1.*

✍ *2.5.28. Изменить текст программы для вычисления суммы факториалов S=1!+2!+3!+…+n!*

🗐 *Пример 2*. Дано натуральное число N. Вычислить N первых сомножителей в выражении $P = \frac{1}{1} \cdot \frac{3}{2} \cdot \frac{5}{3} \cdot ...$

Входные параметры: N – число сомножителей (натуральное число);
Выходные параметры: P –произведение (вещественное число);

В данном выражении числители и знаменатели дробей изменяются по некоторой закону. Например, знаменатель равен порядковому номеру дроби.

Порядок выполнения вычислений можно оформить в виде таблицы.

Параметр **i**	1	2	3	…
Множитель	$^1/_1$	$^3/_2$	$^5/_3$	…
Произведение S	$=^1/_1$	$S=^1/_1*^3/_2$	$S=^1/_1*^3/_2*^5/_3$	…

? Определить закономерность изменения числителя.

Переписать исходную формулу в виде $P = \prod_{i=1}^{N}$ –

```
ReadLn(N); S:=1;
for I:=1 to N do P:=P*(2*I-1)/I;
```

```
Write('P=',P:1:2);
```

Пример 3. Определить и вывести все делители данного натурального числа *N*.

```
ReadLn(n);
for i:=1 to n do
  if n mod i=0 then Write(i,' ')
```

Пример 4. Найти наибольший общий делитель натуральных чисел N, M.

```
ReadLn(n,m);
if n<m then k:=n else k:=m;
for i:=1 to k do
  if (n mod i=0) and (m mod i=0) then d:=i;
WriteLn('Наибольший общий делитель равен ',d);
```

Пример 5. Последовательность чисел вводится с терминала (количество чисел известно). Вычислить сумму этих чисел.

Входные параметры: *N* – количество чисел,
X – число последовательности;
Выходные параметры:S – сумма чисел

```
ReadLn(N);
S:=0;
 for i:=1 to n do
  begin
   Write('Введите ',i,'-е число: ');
  ReadLn(X);
  S:=S+X
 end;
WriteLn('Сумма S=', S:1:2);
```

2.5.29. Изменить фрагмент так, чтобы в нем вычислялись:

a) сумма положительных чисел;
b) среднее арифметическое положительных чисел;
c) среднее арифметическое положительных и отрицательных чисел (отдельно).

Пример 6. Дана последовательность чисел. Определить, входит ли в нее число 5.

Входные параметры: *N* – количество чисел,
X – число из последовательности;
Выходные параметры: сообщение вида "*да–нет*"

Используется FL – промежуточная переменная логического типа (флаг) принимает значение *True*, если найдено искомое число, и *False*, если таких чисел нет.

Программа на языке Pascal	*Комментарии программы*
`program Posl;`	– Заголовок программы
`var X: Real; N,I: Byte;` `FL: Boolean;`	– Описание переменных
`begin`	
`Write('Кол-во чисел N=');`	– Ввод *N*

`ReadLn(N);`	
`FL:=False;`	– предположение, что число 5 не входит в данную последовательность
`WriteLn('Введите числа');`	
`for I:=1 to N do`	– в цикле выполняется
`begin`	
`Read(X);`	– ввод элементов последовательности в переменную X
`if X=5 then FL:=True;`	– сравнение X с числом 5 (если нашли, то изменили FL)
`end;`	
`Write('Число 5 в последовательность ');`	
`if FL then WriteLn('входит')`	– вывод результата
`else WriteLn('не входит')`	
`end.`	

Пример 7. Определить, является данное число простым или составным.

```
f:=True; ReadLn(n);
for i:=2 to Round(Sqrt(n)) do
  if n mod i=0 then f:=False;
Write('Число ',n);
if f then WriteLn(' - простое')
   else WriteLn(' - составное');
```

2.6.30. Оформить решение задач в виде программ, используя фрагменты из примеров 2-5, 7. Проверить их работу.

Ввести тексты программ, запустить их на выполнение и определить, какие задачи в них решаются.

Программа 1

```
program Line;
uses Crt;
const N=70;
var I: Integer;
begin
 for I:=1 to N do
  begin
   ClrScr;
   GotoXY(I,10);
   Write('Pascal');
   Delay(10000)
  end;
end.
```

Программа 2

```
program Abc;
uses Crt;
var I,J: Byte;
begin
  TextBackGround(7);
  TextColor(1);  ClrScr;
  for I:=1 to 25 do
   begin
    WriteLn;
    for J:=1 to I do
     Write(Chr(Ord('A')+I-1))
   end;
  ReadKey;
end.
```

? Объяснить назначение каждой строчки программ.

2.5.31. Изменить программу Line таким образом, чтобы организовать в нем движение текста в другом направлении: справа налево; сверху вниз; снизу вверх или по диагонали.

Заменить в программе Abc в процедуре вывода I на J. Что изменилось? Почему?
Изменить программу Abc таким образом, чтобы буквы на строках выводились в порядке убывания.

✍ *2.5.32. Составить программы для вычисления*

a) суммы ряда $S = \cos(1) + 2 \cdot \cos(2) + 3 \cdot \cos(3) + \ldots + n \cdot \cos(n) = \sum_{i=1}^{n} i \cdot \cos(i)$

b) $S = \sin x + \sin^2 x + \ldots + \sin^n x$

🕮 Цикл с предусловием (цикл ПОКА) имеет структуру записи:

while *условие* **do** *оператор;*

где *условие* – выражение (или переменная) логического типа;
служебные слова **while** (пока), **do** (делать).

🕮 Цикл с постусловием (цикл ДО) имеет структуру записи:

repeat
Оператор1;
Оператор2; {и т.д.}
until *условие_выхода*;

где *условие_выхода* – выражение (или переменная) логического типа.
служебные слова **repeat** (повторять), **until** (до тех пор, пока).

Работа с этими циклами будет рассмотрена параллельно.

✍ *2.5.33. Сколько раз выполнится тело цикла? Какие действия выполняются в каждом фрагменте?*

a)
```
a:=1;
while a<10 do
 begin
  a:=a+2; Write(a:3)
 end;
```

b)
```
d:=5;
repeat
 Write('Введите число: ');
 d:=d-1
until d<-5;
```

c)
```
a:=1; b:=1;
while a+b<=8 do
 begin
  a:=a+1; b:=b+2
 end;
```

d)
```
repeat
 Write('Введите число: ');
 ReadLn(x)
until x=0;
```

✍ *2.5.34. Разделить все фрагменты программ на группы, отвечающие за решение одинаковых задач:*

a)
```
a:=12; y:=0;
while a>1 do
 begin
  y:=y+a; a:=a+1
 end;
```

b)
```
y:=0;
for i:=1 to 12 do
 if i mod 2=0 then y:=i+y;
```

c)
```
d:=14; y:=0;
repeat
 d:=d-2; y:=y+d
until d>0;
```

d)
```
a:=2; y:=2;
while a<12 do
 begin
  a:=a+2; y:=a+y
 end;
```

e)
```
y:=2;
for i:=12 downto 3 do
 y:=y+i;
```
f)
```
d:=1; y:=0;
repeat
 d:=d+1; y:=y+d
until d>=12;
```

✍ *2.5.35. Какое значение будет принимать переменная Y после выполнения фрагментов программы?*

a)
```
y:=0; x:=10;
while x>0 do
 begin
  x:=x-2; y:=y+x
 end;
```
b)
```
y:=1; x:=15;
while x>5 do
 begin
  x:=x-3; y:=y*x
 end;
```
c)
```
y:=1; x:=15;
repeat
 y:=y*x;
 x:=x-3;
until x<5;
```
d)
```
y:=1; x:=10;
repeat
 y:=x*x;
 x:=x-2;
until x<=0;
```

✍ *2.5.36. Найти и исправить ошибки (синтаксические и логические):*

a)
```
x:=1; n:=15;
 While x<=n do;
 S:=S+x*x;
 x:=x+2;
```
b)
```
a:=1;
 repeat
 S:=S+a*a  a:=a+2;
 until x>=15;
```
c)
```
x:=1;
 while x>=15 do
  begin
   S:=S+2x;
   X:=x+2;
  end;
```
d)
```
x:=18;
 repeat
  y:=4*x*x-5;
  Write('при x=', x:1:2,
     'y=', y:1:2);
  x:=x+2;
 until x<=12;
```
e)
```
x:=18;
Write('X            Y');
repeat
 y:=4*x*x-5;
 Write( x:1:2 , y:10:2);
 x:=x-2
Until x<=12;
```
f)
```
b:=8; n:=28;
while b<=n do
 begin
  S:=S+b*b;
  B:=b-2
 end;
```

🗐 *Пример 8*. Вычислить "машинное эпсилон".

В дискретной машинной арифметике всегда существуют такие числа `0<X<Eps`, что `1.0+X=1.0`. Дело в том, что внутреннее представление типа REAL может дать лишь приблизительно 1014 возможных комбинаций значащих разрядов в отведенных для него 6 байтах. Это очень большое число, но оно несопоставимо с бесконечным множеством вещественных чисел, что приводит к появлению "машинного эпсилон".

```
program EpsilonDetect;
var Eps: Real;
begin
 Eps:=1;
 while 1+Eps>1 do
```

```
   Eps:=Eps/2;
  WriteLn('Машинное эпсилон = ',Eps)
end.
```

Ввести текст программы, выполнить прогонку программы и определить "машинное эпсилон".

2.5.37. Изменить программу таким образом, чтобы в ней использовался цикл с постусловием.

Пример 9. Вычислить среднее арифметическое чисел, вводимых с терминала (клавиатуры).

В примере 5 было рассмотрено решение аналогичной задачи. Однако на практике чаще бывают случаи, когда количество чисел заранее неизвестно. Тогда для завершения работы цикла (окончания выполнения действий) используется некоторый символ, вводимый с клавиатуры. Например, ответ на вопрос "Продолжить ввод чисел? Д/Н", либо нажатие клавиши ENTER.

Ниже приведены программы для решения этой задачи двумя способами с помощью циклов с предусловием и постусловием.

Вариант 1

```
program Middle1;
var n: Byte;
  x,S: Real;
begin
 n:=0;  {Кол-во чисел}
 S:=0;  {Сумма чисел}
 WriteLn('Введите числа через пробелы. ');
 WriteLn('Для окончания ввода - Enter ');
 while not EoLn do   {EoLn – конец строки}
  begin
   Read(x);
   n:=n+1; S:=S+x
  end;
 WriteLn('Среднее арифметическое чисел равно ', S/n:1:3);
 ReadLn; ReadLn
end.
```

Вариант 2

```
program Middle2;
var n: Byte; ch: Char;
  x,S: Real;
begin
 n:=0; S:=0;
 repeat
  Write('Введите число: '); ReadLn(x);
  n:=n+1; S:=S+x;
  Write('Продолжить ввод чисел? Д/Н');
  ReadLn(ch);
  Until (ch='Н')or(ch='н');
 WriteLn('Среднее арифметическое чисел равно ', S/n:1:3);
```

```
  ReadLn
end.
```

🗐 *Пример 10*. Определить, входит ли некоторый символ в текстовый фрагмент.

Обозначим X – символ для проверки, ch – очередной символ из текстового фрагмента.

При использовании цикла с предусловием:

```
Write('Введите символ для
     проверки: ');
ReadLn(X);
FL:=False;
WriteLn('Введите текст');
while not EoLn do
 BEGIN
  Read(ch);
  if ch=X then FL:=True
 end;
```

При использовании цикла с постусловием следующие изменения

```
Write('Введите символ для
      проверки: ');
 ReadLn(X);
 FL:=False;
 WriteLn('Введите текст');
repeat
 Read(ch);
 if ch=X then FL:=True
until EoLn;
```

✍ *2.5.38. Оформить решение задачи в виде двух программ.*

🗐 *Пример 11*. Угадать целое число (меньшее 100), задуманное компьютером.

Пусть A – целое число, задуманное компьютером;

B – целое число, вводимое с клавиатуры.

Компьютер "задумывает" число с помощью генератора случайных чисел – функции `random(100)`. Для возможности получения различных вариантов решения используется процедура `randomize` – инициализация датчика псевдослучайных чисел.

```
Program TellFortunes;   {Угадай число}
uses Crt;
var A,B: Integer;
begin
 ClrScr;  Randomize;
 A:=Random(100);
 repeat
  Write('Введите число: ');
  ReadLn(B);
  if B<A then WriteLn('Задуманное компьютером число больше')
   else
   if B>A then WriteLn('Задуманное компьютером число меньше');
 until A=B;
 WriteLn('УГАДАЛ! ');
 ReadLn;
end.
```

✍ *2.5.39. Изменить программу таким образом, чтобы в ней определялось, за сколько раз вам удалось угадать число.*

? Придумать стратегию для более быстрого угадывания чисел.

Пример 12. Дано целое число n. Определить сумму и количество цифр этого числа.

Входные параметры: n – целое число;

Выходные параметры: S – сумма цифр числа n;

k – количество цифр числа n.

Используя операцию `mod` (остаток от деления), можно определить последнюю цифру числа.

Используя операцию `div` (целочисленное деление), можно получить число, состоящее из тех же цифр, но без последней цифры (т.е. отбросить ее).

Повторять действия до тех пор, пока данное число не станет равным 0.

```
program Number;
var n: LongInt;
    S,k: Byte;
begin
 Write('Введите число:');
 ReadLn(n);
 S:=0; k:=0;
 while n<>0 do
  begin
   S:=S+n mod 10;
   n:=n div 10;
   k:=k+1
  end;
 WriteLn('Сумма цифр: ',S);
 WriteLn('Кол-во цифр: ',k);
 ReadLn
end.
```

Примечание. Если данное число равно 0, то оно содержит одну цифру.

2.5.40. Изменить программу так, чтобы при вводе отрицательного числа или нуля результат выводился правильный.

2.5.41. Вывести цифры данного числа n в обратном порядке.

Пример 13. Составить таблицу значений функции $y = ax^2 \square b$ на отрезке [-1;1] с шагом изменения аргумента 0.2.

```
program TableFunction;
var a,b,x,y: Real;
begin
 x:=-1;
 while x<=1 do
  begin
   y:=a*x*x+b;
   x:=x+0.2;
   WriteLn('При x=', x:1:1,
           'y=', y:1:4);
  end; ReadLn
end.
```

2.5.42. Исправить логическую ошибку, допущенную в программе.

Пример 14. Вычислить сумму $S = \sum_{i=1}^{\infty} \frac{1}{i^2}$ с заданной точностью ε.

Считается, что требуемая точность достигнута, если модуль очередного слагаемого меньше ε (Eps), все последующие слагаемые не суммируются.

```
program SumNumber;
var s,Eps: Real;
    i: Integer;
begin
 Write('Введите точность: '); Read(Eps);
 s:=0; i:=1;
 while 1/Sqr(i)>=Eps do
  begin
```

```
    s:=s+1/Sqr(i);  i:=i+1;
  end;
 WriteLn('S = ', s:7:4);
end.
```

✍ *2.5.43. Записать программы для решения задач из примеров 12-14, используя цикл с постусловием.*

✍ *2.5.44. Составить таблицу значений функции* $y = \begin{cases} a\lg x + \sqrt[3]{x} & \text{при } x > 1 \\ 2a\cos \pi x & \text{при } x \le 1 \end{cases}$,

где $a = 3.5$ *- константа, на отрезке [0.5;2] с шагом изменения аргумента 0.25.*

💻 *Проверить работу программ на контрольных примерах.*

? Вопросы для повторения.

1. Что такое цикл? Привести примеры.
2. На какие типы делятся все циклы? Чем они отличаются?
3. Формат записи каждого цикла.

2.6. ИСПОЛЬЗОВАНИЕ СОСТАВНЫХ ТИПОВ ДАННЫХ

Простые типы данных определяют атомарные (неделимые) значения.

Часто имеет смысл объединить некоторые данные одним именем. Исполнение многих алгоритмов было бы просто невозможно, если бы соответствующие объекты не были каким-либо образом организованы.

Составные (структурные) типы данных базируются на простых типах. Паскаль допускает образование структур данных произвольной сложности. *Переменные составных* типов состоят из *отдельных компонент*.

2.6.1. Строковые переменные

📖 *Строка* – это последовательность символов. Строковый тип имеет идентификатор `String[n]`, где `n` – максимально возможное количество символов в строке, используется для экономии машинной памяти или форматного вывода. Количество символов не должно превышать 255.

Примечание 1. При описании `[n]` указывать не обязательно.

Длина строки – это фактическое количество символов, содержащееся в данный момент в строковой переменной. Для ее определения используется встроенная функция `Length(s)`, где `s` – строковая переменная.

Каждый символ строки имеет порядковый номер, который используется для обращения к нему. Например, `s[1]`, `s[5]`, `s[n]`.

Все задачи со строками можно разделить на три группы:

- ✓ нахождение символов или их подсчет;
- ✓ замена одних символов другими или формирование новых строк;
- ✓ преобразование строк с использованием встроенных функций и процедур.

Примечание 2. При работе со строковыми переменными следует использовать процедуры ввода `ReadLn` и вывода `WriteLn`.

? Что такое "пустая строка"?
Какие операции можно использовать при работе со строками?
Как выполняется сравнение двух строк?

ЗАДАЧИ НА НАХОЖДЕНИЕ НЕКОТОРЫХ СИМВОЛОВ ИЛИ ИХ ПОДСЧЕТ

🗐 *Пример 1*. Дана строка st. Найти количество букв "д" и "т" (отдельно), входящих в эту строку.

```
program Stroka1;
var st: String;
   i,k,n: Byte;
begin
 WriteLn('Введите строку'); ReadLn(st);
 k:=0;  n:=0;
 for i:=1 to Length(st) do
  begin
   if st[i]='д' then  k:=k+1;
   if st[i]='т' then  n:=n+1;
  end;
 WriteLn('Количество букв "д" равно ', k);
 WriteLn('Количество букв "т" равно ', n);
 ReadLn
end.
```

✍ *2.6.1. Изменить программу таким образом, чтобы в ней определялось и выводилось, каких букв, "д" или "т", больше.*

ЗАДАЧИ НА ЗАМЕНУ ОДНИХ СИМВОЛОВ ДРУГИМИ ИЛИ ФОРМИРОВАНИЕ НОВЫХ СТРОК

🗐 *Пример 2*. Дана строка s. Заменить в ней все символы "и" на "ы" и наоборот, "ы" на "и".

```
program Replace;
var s: String; i: Byte;
begin
 WriteLn('Введите строку'); ReadLn(s);
 for i:=1 to Length(s) do
  begin
   if s[i]='и' then s[i]:='ы';
   if s[i]='ы' then s[i]:='и'
  end;
 WriteLn('Полученная строка:'); WriteLn(s);
 ReadLn
end.
```

? Найти и исправить логическую ошибку.

🗐 *Пример 3*. Программа имеет вид:

```
program OverTurn;
var s1,s2: String[20];
```

```
    i,n: Byte;
begin
 WriteLn('Введите строку');
 ReadLn(s1);
 n:=Length(s1); s2:='';
 for i:=n downto 1 do  s2:=s2+s1[i];
 if  s1=s2 then WriteLn('Строка является перевертышем')
      else WriteLn('Строка не является перевертышем');
end.
```

? Какая задача решается в программе?

Является ли данная программа универсальной? Почему?

ЗАДАЧИ НА ПРЕОБРАЗОВАНИЕ СТРОК
С ИСПОЛЬЗОВАНИЕМ ВСТРОЕННЫХ ФУНКЦИЙ И ПРОЦЕДУР

Примечание. При удалении или добавлении символов изменяется длина строки. При использовании циклов с предусловием и постусловием каждый раз при проверке условия пересчитывается длина строки.

🗐 *Пример 4*. Дана строка `Str`. Вставить перед каждой буквой 'м' сочетание 'до'.

```
S:='до'; i:=1;
while i<=Length(Str) do
 if s[i]='м' then Insert(s,Str,i)
      else i:=i+1;
```

🗐 *Пример 5*. Дана строка `s`. Удалить из нее все сочетания "на".

Фрагмент 1

```
i:=1;
 while i<=Length(s)-1 do
  if (s[i]='н')and(s[i+1]='a')
then Delete(s,i,2)
      else i:=i+1;
```

Фрагмент 2

```
d:=Pos('на',s);
repeat
 Delete(s,d,2);
 d:=Pos('на',s)
until d=0;
```

? Какой фрагмент дает более универсальное решение? Почему?

✍ *2.6.2. Составить программы для задач из примеров 4, 5.*

💻 *Ввести программы и выполнить контрольные примеры.*
Для задачи из примера 5 использовать исходную строку "Бнааннаан"?

? Вопросы для повторения.

1. Что называется строкой? Каким образом она задается в программе?
2. Что такое длина строки? Как ее можно определить?
3. Какие действия можно выполнять над строками?

2.6.2. Массивы

Один из самых распространенных способов организации данных – *табличный*. С таблицами мы сталкиваемся всюду: телефонный справочник, каталог, список группы, расписание занятий и т.д. Основное требование к табличным данным – это однородность их структуры. В информатике для работы с

ними используются переменные, которые имеют специальный тип данных, называемый *массивом*.

📖 *Массив* – это тип данных, характеризующий фиксированное число однотипных элементов.

Примечание 1. При решении задач часто понятие "массив" используется для типа данных и переменных этого типа.

Сформулируем *свойства массива* с помощью различных *примеров из жизни*.

1. Элементы массива должны быть одного типа.

Представим такую ситуацию: в телефонном справочнике наряду с номерами телефонов можно найти рецепты блюд, прогнозы погоды. Это уже будет явно не телефонный справочник, а записная книжка студента. А как вы отнесетесь к тому, если в кассе вместо стипендии станут выдавать билеты в театр?

2. Каждый элемент массива должен быть однозначно определен своим порядковым номером (обозначением), который называется *индексом*.

В качестве примера массива рассмотрим список студентов группы. По порядковому номеру в журнале можно узнать ФИО студента. И наоборот, зная ФИО студента, можно найти его порядковый номер в журнале. Под одним номером можно записать только одного студента. Состав группы может измениться, но всегда останется неизменным требование взаимно однозначного соответствия между ФИО студента и его номером в списке. Когда используется список, то индексы элементов (объектов) следуют по порядку: 1, 2, 3, … – так удобнее.

3. Значения элементов могут повторяться, а все индексы должны быть различны.

Например, несколько преподавателей могут иметь одинаковые фамилии, но всегда найдется различающий их признак. В массиве в качестве такого "признака" используются индексы элементов.

Любой поезд не может иметь два первых вагона, иначе, купив билет в первый вагон, вы окажетесь перед вопросом: в какой из них следует садиться.

При покупке билетов в рейсовый автобус пассажиру указывают номер его места, по которому он точно определяет, куда должен сесть. Однако один человек не обязательно должен иметь только одно место, может купить себе два места и даже забронировать целый автобус.

4. Количество элементов массива определяется (фиксируется) при его описании и не может быть изменено при выполнении программы.

Например, в зачетной книжке студента уже имеются ячейки для заполнения данных о зачетах и экзаменах, т.е. учтено их максимальное количество.

При работе с массивом не должна возникнуть аварийная ситуация, когда некоторым его элементам не хватило ячеек памяти (пригласили на юбилей

20 человек, а пришли 30). Всегда определяется максимально возможное количество элементов массива (пусть лучше останутся свободные места).

5. Не обязательно все элементы массива должны быть заполнены значениями. Например, в расписании на железнодорожном вокзале номер поезда является его индексом, по которому определяются станции назначения. Однако в расписании указаны не все номера поездов, и не потому, что других маршрутов нет, а потому, что другие поезда не заходят на данную станцию.

6. Могут использоваться одномерные, двумерные и многомерные массивы. Ограничения – удобство работы и общее количество памяти под элементы (максимальный размер памяти статического массива – 65520 байт).

Чем отличается распределение мест в театре и в автобусе? Если на театральных билетах указать только номера мест, то этих данных будет недостаточно. Каждый билет (элемент массива) для однозначного определения местоположения в театре должен иметь две числовые характеристики (два индекса).

При покупке билета на поезд следует указать номер поезда, номер вагона и номер места, т.е. нужны три индекса (будет использован трехмерный массив).

? Достаточно ли этих данных на железнодорожном билете для того, чтобы отправиться в путь?

При решении жизненных задач всегда приходится строить математическую модель, в которой отражаются только существенные признаки объекта или явления.

Одномерные массивы

🕮 Если в таблице всего одна строка содержит данные, то такая таблица называется линейной. Для ее обработки в Паскале используются одномерные массивы.

Формат *описания* одномерного массива: **`array`** `[T1]` **`of`** `T2;`

где `T1` – тип индексов, `T2` – тип элементов массива.

? Дать определение одномерного массива.

Какие типы индексов и элементов можно использовать в Паскале?

Примечание 2. Количество элементов массива может быть задано явно при описании переменной, в разделе констант или введено при выполнении программы.

Примечание 3. Тип Массива можно задать в разделе **`type`**.

🗐 *Пример 1*. Дана таблица. Описать переменную **A** для работы с ней.

Индексы, i	1	2	3	4	5	6
Элементы, A[i]	2.5	-1	0.4	-4.8	0	14

Приведены различные варианты описания данных этой таблицы.

```
1. var A: array[1..6] of Real;
2. const n=6;
```

```
var A: array[1..n] of Real;
```

3. `var A: array[1..50] of Real;` – описано максимально возможное кол-во элементов массива 50, но работать следует только с частью массива (т.е. с 6 элементами).

4.
```
const n1=1; n2=50;
type t1=n1..n2;
     mas=array[t1] of Real;
var A: mas;
```

? Все ли возможные варианты описания приведены?

✍ *2.6.3. Имеется описание:*

```
var A: array[0..100] of Real;
    M: array[1..5] of Byte;
```

Какой объём памяти (байт) необходим для хранения каждого массива?

✍ *2.6.4. Какие допущены ошибки при описании одномерных массивов? Переписать, исправив ошибки.*

```
const n=50; k=-1;
var A: array[Integer] of Real;
   B: array[n] of Char;
   C: array[1..k] of False..True;
   D: array['a'..'z'] of 1..10;
```

🕮 Две переменных типа массив называются эквивалентными, если они описаны одним именем типа.

Например, переменные A и B эквивалентны при следующих описаниях:

1.
```
var A,B: array[1..10] of Real;
```

2.
```
type mas=array[1..50] of Real;
var A: mas; B: mas;
```

Переменные C и D не эквивалентны при следующем описании:

```
var  C: array[1..10] of Real;
     D: array[1..10] of Real;
```

Переменные типа массив могут участвовать только в операциях сравнения (= или <>) и в операторе присваивания. При этом обе переменные должны быть эквивалентны. Все остальные действия выполняются только над элементами массива.

Для того чтобы обратиться к элементу массива, следует указать имя массива и в квадратных скобках – *индекс* элемента. Например, A[1], A[2], A[i], …

В качестве индекса может быть использовано выражение, тип которого соответствует описанному типу индексов.

Если над элементами массива нужно выполнять однотипные действия, используют оператор цикла, в котором параметр цикла определяет индексы элементов массива.

? К какому типу ошибок относится выход индекса за границы допустимых значений?

✍ *2.6.5. Описать в тетради следующие массивы:*

Индексы, i	1	2	3	4	5	6	7	7	8	9	10	11	12	13	14
Элементы, A[i]	2	-5	6	0	12	7	-8	6	10	22	-9	0	3	3	7

Индексы, i	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
Элементы, B[i]	-3.7	0.5	2.4	-8.3	1.0	0.6	-5.0	1.9	11.2	-15.1	0	31.5

Индексы, i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Элементы, C[i]	g	f	r	y	d	3	u	/	5	h	f	4	v	j	u	g	7

Чему равны следующие элементы массива:

```
A[1]=                  b[5]=                  c[1]=
a[4]=                  B[-2]=                 C[2]=
a[9]=                  b[0]=                  c[7]=
```

✍ *2.6.6. Объяснить следующее описание массивов*

```
const n=12;
type mass1=array [2..15] of Real;
     mass2=array [Char] of 0..2;
var A: mass1; A1: mass2;
    A2: array [1..10] of mass2;
    B: array [-10..19] of Char;
    K: array [Byte] of Integer;
    L: array [1..n] of Boolean;
```

✍ *2.6.7. Элементы массива P, имеющие индексы 1..5, равны соответственно 1, –1, 5, 2, 4. Вычислить значение выражения*

```
p[1]*p[3]-p[2*p[2]+p[p[5]-p[2]]]
```

ЗНАЧЕНИЯ ЭЛЕМЕНТОВ МАССИВА (количество которых n)
можно задавать несколькими способами:

1. вводом с клавиатуры:

```
for i:=1 to n do
 begin
  Write('Введите A[',i,']=');
  ReadLn(A[i]);
 end;
```

или

```
WriteLn('Введите ', n,
     ' элементов массива');
for i:=1 to n do
 Read(A[i]);
```

2. программным заполнением элементов значениями, которые вычисляются по некоторому закону (например, $d_i=i^2$):

```
for i:=1 to n do
 d[i]:=i*i;
```

3. программным заполнением элементов числами, выбранными случайным образом (random):

```
for i:=1 to n do
 A[i]:=Random(255);
```

```
k:=100;
for i:=1 to n do
 B[i]:=Random*k;
```

Массив A содержит целые числа, меньшие 255, B содержит неотрицательные вещественные числа, меньшие 100.

? Каким образом можно задать отрицательные элементы массива; значения символьного типа?

ВЫВОД ЭЛЕМЕНТОВ МАССИВА:

```
WriteLn('Измененный массив: ');
for i:=1 to n do
  Write(A[i],'  '); {вывод очередного элемента массива}
WriteLn; {переход на новую строку}
```

Все задачи на использование одномерных массивов можно разделить на несколько групп.

I. ПОИСК ИЛИ ВЫЧИСЛЕНИЕ. ОТВЕТ – ЗНАЧЕНИЕ ИСКОМОЙ ПЕРЕМЕННОЙ

🗐 *Пример 1*. В одномерном массиве, состоящем из ***n*** целых чисел, найти произведение положительных однозначных элементов.

Входные параметры: *A* – массив целых чисел,
n – количество элементов массива;
Выходные параметры:P – произведение чисел.

```
program m1;
var A: array[1..20] of Integer;
    n,i,P: Integer;
begin
    (*Ввод массива*)
 Write('введите кол-во элементов: ');
 Read(n);
 WriteLn('Введите элементы массива');
 for i:=1 to n do Read(A[i]);
  (*Вычисление произведения положит. однозн. элементов*)
 P:=1;
 for i:=1 to n do
  if (A[i]>0)and(A[i]<10) then P:=P* A[i];
    (*Ввод результата P*)
 WriteLn('Произведение P=',P);
 ReadLn; Readln
end.
```

✍ *2.6.8. Изменить программу таким образом, чтобы в ней*

1. *определялись сумма положительных двузначных чисел и их количество;*
2. *в случае отсутствия искомых элементов выдавалось сообщение об этом.*

🗐 *Пример 2*. В одномерном числовом массиве найти максимальный элемент. Определить индекс этого элемента.

Входные параметры: *A* – массив целых чисел,
n – количество элементов массива;
Выходные параметры: m – индекс максимального элемента,
A[*m*] – максимальный элемент.

Максимальный элемент находится следующим образом:

1) до проверки элементов массива любой из них можно считать максимальным (предположим, что им окажется первый элемент массива), заносим его индекс в переменную m;
2) сравнивается значение A[m] со значениями всех остальных элементов массива; если A[m] оказался меньше некоторого A[i] элемента, то заносим значение i в переменную m.

В этой программе будет использовано заполнение массива произвольными целыми неотрицательными числами, меньшими 50.

```
program m2;
const n=20;
var A: array[1..n] of Integer;
    i,m: Integer;
begin
    (*Заполнение массива*)
 Randomize;
 for i:=1 to n do
  A[i]:=Random(50);
    (*Вывод исходного массива*)
 WriteLn(Исходный массив: ');
 for i:=1 to n do
  Write(A[i],'  ');
 WriteLn;
  (*Нахождение индекса максимального элемента*)
 m:=1;
 for i:=2 to n do
  if A[i]>A[m] then m:=i;
    (*Ввод результата*)
 WriteLn('Максимальный элемент A[',m,']=',A[m]);
 ReadLn
end.
```

✍ *2.6.9. Изменить программу таким образом, чтобы вычислялась разность между максимальным и минимальным элементами массива.*

II. ОПРЕДЕЛИТЬ НАЛИЧИЕ. ОТВЕТ – ВИДА "ДА" – "НЕТ"

🗐 *Пример 3*. Определить, есть ли в одномерном числовом массиве равные рядом стоящие элементы.

В этом классе задач используется переменная логического типа Fl (флажок) или целого типа (счетчик).

```
  (*Проверка наличия одинаковых соседних элементов*)
 fl:=False; {таких элементов может не быть}
 for i:=1 to n-1 do
  if A[i]=A[i+1] then fl:=True;
    (*Ввод результата*)
 if fl then WriteLn('Одинаковые соседние элементы есть')
    else WriteLn('Одинаковых соседних элементов нет');
```

III. ИЗМЕНЕНИЕ НЕКОТОРЫХ ЭЛЕМЕНТОВ МАССИВА. ОТВЕТ – ИЗМЕНЕННЫЙ МАССИВ.

Пример 4. В одномерном числовом массиве все элементы, значения которых больше 20 или меньше –10, заменить 1. Измененный массив вывести на терминал.

```
   (*Проверка и замена элементов массива*)
 for i:=1 to n do
  if (A[i]>20)or(A[i]<-10) then A[i]:=1;
   (*Вывод измененного массива*)
 WriteLn('Полученный массив');
 for i:=1 to n do
  Write(A[i]:4);
 WriteLn;
```

IV. ВЫВОД ЧАСТИ МАССИВА

Пример 5. Вывести на экран все элементы одномерного массива целых чисел, индексы которых кратны 3.

Вывод элементов в данном примере можно выполнить двумя способами:

```
WriteLn('Полученный массив');
  i:=3;
  while i<=n do
   begin
    Write(A[i]:4); i:=i+3;
   end;
  WriteLn;
```

иначе

```
WriteLn('Полученный массив');
 for i:=1 to n do
  if i mod 3=0 then
    Write(A[i]:4);
 WriteLn;
```

V. ПОСТРОЕНИЕ МАССИВА ПО НЕКОТОРЫМ ПРАВИЛАМ

Пример 6. Дан массив целых чисел **A**. Получить массив **B** по следующему правилу:

$b_1=a_1$;

$b_2=a_1+a_2$;

$b_3=a_1+a_2+a_3$ и т.д.

Полученный массив **B** вывести на терминал.

? Продолжить предложенную закономерность и определить, как иначе можно переписать формирование элементов массива B?

```
   (*Построение массива B*)
 B[1]:=A[1];
 for i:=2 to n do
   B[i]:=B[i-1]+A[i];
  (*Вывод измененного массива*)
WriteLn('Полученный массив');
for i:=1 to n do  Write(B[i]:4);
WriteLn;
```

VI. КОМБИНИРОВАННЫЕ ЗАДАЧИ

Пример 7. Определить, есть ли в одномерном числовом массиве отрицательные элементы. Если они есть, то каждый второй элемент увеличить в 2 раза.

```
  (*Проверка наличия отрицательных элементов*)
 fl:=False;
 for i:=1 to n do
  if A[i]<0 then fl:=True;
    (*Проверка значения fl и замена элементов массива*)
 if fl then
  for i:=1 to n do
   if i mod 2=0 then A[i]:=A[i]*2;
     (*Вывод измененного массива*)
 WriteLn('Полученный массив');
 for i:=1 to n do  Write(A[i]:4);
 WriteLn;
```

✍ *2.6.10. Оформить решение задач из примеров 3-7 в виде программ.*
💻 *Проверить работу программ на контрольных примерах.*

🗐 *Пример 8*. Отсортировать (упорядочить) элементы массива в порядке возрастания.

В данной программе используется следующий метод.

Массив делится на две части: отсортированную и неотсортированную (первоначально весь массив считается неупорядоченным). В неотсортированной части находится минимальный элемент (цикл по переменной j) и меняется с первым элементом из этой части. Далее действия повторяются n-1 раз (цикл по переменной i).

```
program m8;
var MAS: array[1..10] of
          Integer;
    min,i,n,j,X,K: Integer;
begin
 Write('n=');  Read(n);
 for i:=1 to n do
  begin
   Write(' MAS[',i,']=');
   ReadLn(MAS[i]);
  end;
 for i:=1 to n do
  begin
   min:=i; {номер min эл-та}
    for j:=i+1 to n do
     if MAS[j]<MAS[min] then
        min:=j;
    X:=MAS[i];
    MAS[i]:=MAS[min];
    MAS[min]:=X;
  end;
 for i:=1 to n do
  Write(MAS[i],' ');
 WriteLn;
 ReadLn
end.
```

✍ *2.6.11. Составить программы для решения задач:*

a) *В одномерном массиве действительных чисел найти кол-во элементов, меньших –2.*
b) *Даны два вектора (массива с количеством элементов n в каждом). Вычислить скалярное произведение этих векторов.*
c) *В одномерном числовом массиве вывести все элементы, значения которых меньше, чем значение первого элемента. Если таких элементов нет, то вывести сообщение об этом.*

? Вопросы для повторения.

1. Что такое массив, элемент массива?

2. Какими свойствами обладает массив?
3. Что такое индекс элемента, для чего он предназначен?
4. Как можно задавать значения элементов одномерного массива? Выводить массив?

Двумерные массивы (матрицы)

Формат *описания* одномерного массива: **array** [T1,T2] **of** T3;
где T1, T2 – типы индексов, T3 – тип элементов массива.

? Что общего и чем отличаются двумерный и одномерный массивы?

? Каким образом выполняется обращение к элементам двумерного массива?

? Сколько циклов следует использовать при обращении ко всем элементам двумерного массива?

✍ 2.6.12. Описать двумерные массивы. Записать обращение к элементам этого массива.

массив A

Индексы	1	2	3	4	5	6	7
1	2	-5	6	0	12	7	-8
2	4	7	10	-5	7	3	-1
3	1	7	0	0	1	4	6
4	2	8	-2	-3	5	8	0
5	7	2	-3	-7	-3	9	11

массив B

Индексы	1	2	3	4	5	6
1	0.5	9	0	3	1.5	0.7
2	-1	-7.1	5.4	4.1	2.8	-1.5
3	4	7.3	-4.4	6	0	-8.9
4	0.2	-8	2.1	0	0.4	1

Например, A[1,1]=2; B[4,6]=1.

✍ 2.6.13. Объяснить описание массивов и использование их элементов. Привести собственные примеры.

```
const k=15;
type
  mass1=array[char, char] of Integer;
  mass2=array[1..10] of real;
var C: mass1;
  D: array[1..10] of mass2;
  E: array[1..k,1..k] of Boolean;
  X: array[1..20,1..20] of real;
  Y: mass2;
```

Привести примеры возможных значений элементов массивы.

? Как выполняется обращение к элементам массива?

Двумерные массивы (как и одномерные) можно создавать разными способами. Сначала необходимо задать число строк и столбцов двумерного массива (матрицы), например, ввести их с клавиатуры:

```
Write('Введите число строк'); Read(n);
Write('Введите число столбцов'); Read(m);
```

Далее следует заполнять элементы массива данными:

1. вводом с клавиатуры:

```
for i:=1 to n do
  for j:=1 to m do
   begin
    Write('Введите A[',i,',',
      j,']=');
    ReadLn(A[i,j]);
   end;
```

эти строки можно заменить следующими

эти строки можно заменить следующими

```
WriteLn('Введите матрицу ', n,
        ' * ',m);
for i:=1 to n do
 for j:=1 to m do
  Read(A[i,j]);
```

2. программным способом, например, значения элементов вычислять по некоторому закону ($d_{i,j}$=i*j)

```
 for i:=1 to n do
 for j:=1 to m do
  d[i,j]:=i*j;
```

3. программным способом, при котором значения элементов задаются случайным образом:

```
for i:=1 to n do
 for j:=1 to m do
  A[i,j]:=Random(255);
```

```
k:=100;
k:=100;
for i:=1 to n do
for j:=1 to m do
  B[i,j]:=Random*k;
```

? Перечислить способы заполнения элементов матрицы данными.

ВЫВОД ЭЛЕМЕНТОВ ДВУМЕРНОГО МАССИВА:

```
WriteLn('Измененный массив: ');
for i:=1 to n do
 begin
  for j:=1 to m do
   Write(A[i,j],'  '); {вывод очередного элемента массива}
  WriteLn; {переход на новую строку}
 end;
```

Пример 9. В двумерном массиве (матрице), состоящем из ***n*m*** целых чисел, все отрицательные элементы возвести в квадрат.

```
program m1;
var A: array[1..20,1..20] of Integer;
    i,j,n,m: Integer;
begin
 Write('Введите количество строк и столбцов: ');
 Read(n,m);
 WriteLn('Введите матрицу: ');
 for i:=1 to n do
  for j:=1 to m do
   Read(a[i,j]);
 ReadLn;
 for i:=1 to n do
  for j:=1 to m do
   if a[i,j]<0 then a[i,j]:=Sqr(a[i,j]);
```

```
  WriteLn('Полученная матрица: ');
  for i:=1 to n do
   begin
    for j:=1 to m do
     Write(a[i,j]:4);
    WriteLn
   end;
  ReadLn
end.
```

При работе с двумерными массивами можно выполнять действия над всеми элементами (или к ним обращаться) массива, а также строки, столбца или диагоналей (для квадратной матрицы).

Поэтому кроме вышеперечисленных типов задач используются задачи на работу с отдельными строками, столбцами или диагоналями (для квадратной матрицы).

Пример 10. В квадратной матрице (n×n) заменить элементы главной диагонали нулями. Вывести на экран измененный массив.

```
program matr1;
var B: array [1..10,1..10] Of Real;
    n,i,j: Integer;
begin
 WriteLn ('Введите количество строк (столбцов) матрицы');
 Write('n='); ReadLn(n);
   {заполнение матрицы}
 for i:=1 to n do
  for j:=1 to n do
   begin
    Write('Введите b[',i,',',j,'] элемент ');
    Read(b[i,j]);
   end;
   {возведение в квадрат диагональных элементов}
 for i:=1 to n do
  b[i,i]:=0;
   {вывод измененной матрицы}
 WriteLn('Измененный массив: ');
 for i:=1 to n do
  begin
   for j:=1 to n do
    Write(b[i,j]:6:1);
   WriteLn;
  end;
end.
```

Пример 11. В двумерном массиве (матрице) вычислить среднее арифметическое значение элементов последнего столбца матрицы.

```
program  matr;
var A: array [1..10,1..10] of Real;
    n,m,i,j: Integer;
```

```
    sum: Real;
begin
 WriteLn ('Введите количество строк ’);
 Write('N='); Read(N);
 WriteLn('Введите количество столбцов ’);
 Write('M='); Read(M);
  {заполнение матрицы}
 for i:=1 to n do
  for j:=1 to m do
   Read(a[i,j]);
  {вычисление и вывод результатов}
 sum:=0;
 for i:=1 to n do sum:=sum+a[i,m];
 if n>0 then
  WriteLn ('Среднее арифметическое =', sum/n:6:2)
   else WriteLn ('Ошибка ввода данных!');
end.
```

✍ 2.6.14. В двумерном массиве (матрице) вычислить произведение элементов второй строки матрицы.

✍ 2.6.15. В двумерном массиве (матрице) вычислить сумму положительных четных элементов и их количество.

✍ 2.6.16. В двумерном массиве (матрице) найти максимальный элемент. Поменять местами строки (столбцы): первый и с максимальным элементом. Полученный массив вывести на терминал.

2.7. ЗАДАЧИ ПОВЫШЕННОЙ СЛОЖНОСТИ ПО ВСЕМ ТЕМАМ

1. Вывести все простые числа, используя решето Эратосфена.
2. Определить первую и последнюю цифры данного вещественного числа.
3. Написать программу, которая преобразует число из римской системы в десятичную систему счисления и наоборот.
4. Написать программу, которая преобразует число из одной позиционной системы счисления в другую.
5. Найти n первых чисел Фибоначчи, заданные рекуррентной формулой: $a_1=1, \quad a_2=2, \quad a_n=a_{n-1}+a_{n-2}$.
6. Составить программу для выполнения "длинной" арифметики, то есть выполняющую операции сложения, вычитания, умножения или деления над числами, имеющими произвольное количество десятичных разрядов.
7. Составить программу для вывода на экран n первых строк треугольника Паскаля, имеющего вид:

```
                1
             1     1
          1     2     1
       1     3     3     1
    1     4     6     4     1
                . . .
```

8. Составить список слов, которые входят в данное предложение.
9. Из данного предложения выбрать слова, имеющие заданное количество букв.
10. Каждый столбец (строку) матрицы упорядочить по возрастанию (убыванию).
11. Проверить, является ли многоугольник, заданный координатами своих вершин, выпуклым.
12. Определить, можно ли кирпич со сторонами A, B,C "протащить" сквозь прямоугольное отверстие со сторонами P, S (стороны отверстия должны быть параллельны сторонам кирпича).
13. Определить, есть ли в одномерном числовом массиве повторяющиеся элементы.
14. Найти натуральное число из интервала от 1 до n с максимальной суммой делителей.
15. Натуральное число называется совершенным, если оно равно сумме своих делителей, за исключением самого числа (например, 6=1+2+3). Получить все совершенные числа, меньшие n.
16. Дана квадратная матрица. Заменить нулями все ее элементы, расположенные на главной диагонали и выше нее.
17. Дано натуральное число n (n<1000). Записать его словами (например, семнадцать, сто семьдесят один).
18. Билет считается счастливым, если в его шестизначном номере сумма трех первых цифр равна сумме трех последних цифр. Определить, явля-

ется ли некоторый билет счастливым. Чему равна вероятность получения счастливого билета (определить, какие данные должны быть известны)?

19. Дана матрица размерности n×m.
    a) Удалить из нее столбец, сумма элементов которого наибольшая.
    b) Отсортировать строки матрицы в порядке возрастания их первых элементов.

СПИСОК ЛИТЕРАТУРЫ

1. Информатика. Базовый курс. 2-е издание/ С.В. Симонович и др. СПб.: Питер, 2007.–640 с.: ил.
2. Информатика: Учебник.– 3-е изд., перераб. / Под. ред. Н.В. Макаровой. – М.: Финансы и статистика, 2002.–768 с.: ил.
3. Информатика: Учеб. пособие для сред. спец. учеб. заведений/ Под ред.П.П. Беленького. – Ростов н/ Дону: Феникс, 2002. – 448 с.
4. Каймин, В.А. Информатика: Учебник. – 2-е изд., перераб. и доп. – М.: ИНФРА-М, 2002. – 272 с.
5. Касаев, Б.С. Информатика: практикум на ЭВМ: Учеб. пособие/ Б.С. Касаев, В.А. Каймин. – М.: ИНФРА, 2003. – 271 с.
6. Колдаев, В.Д Сборник задач и упражнений по информатике/ В.Д. Колдаев, Е.Ю. Павлова.- М.: ИНФРА-М, 2007. – 256 с.
7. Панкратова, Л.П. Контроль знаний по информатике: тесты, контрольные задания, экзаменационные вопросы, компьютерные проекты/ Л.П. Панкратова, Е.Н. Челак. – СПб.: БХВ-Петербург, 2004. – 448 с.
8. Симонович, С.В., Евсеев, Г.А., Алексеев, А.Г. Специальная информатика: Учебное пособие. – М.: АСТ-ПРЕСС: Инфорком-Пресс, 2002. – 480 с.
9. Попова, Л.А., Шульман, И.Б. Основы программирования на языке Pascal: Учебное пособие / Рубцовский индустриальный институт. – Рубцовск, 2002. – 60 с.

Попова Людмила Анатольевна
Шульман Ирина Борисовна

ИНФОРМАТИКА–ПРАКТИКУМ
ЧАСТЬ 1

Учебное пособие для студентов технических специальностей 1-го курса

Редактор Е.Ф. Изотова
Подготовка оригинала-макета Н.В. Коленко

Подписано к печати 18.09.09. Формат 84×108 /16.
Усл. печ. л. 4,68. Тираж 180 экз. Зак. 09-745. Рег. № 57.

Отпечатано в РИО Рубцовского индустриального института
658207, Рубцовск, ул. Тракторная, 2/6.