МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Т. В. Афанасьева,
Ю. Е. Кувайскова,
В. А. Фасхутдинова

АЛГОРИТМЫ И ПРОГРАММЫ

Учебное пособие

Ульяновск
УлГТУ
2011

УДК 004.4 (075)
ББК 32. 973 – 018.2я7
А 94

Рецензенты: профессор кафедры «Информационные технологии» Ульяновского государственного университета, д-р техн. наук, профессор Семушин И. В.,

кафедра «Телекоммуникационные технологии и сети» Ульяновского государственного университета.

Утверждено редакционно-издательским советом университета в качестве учебного пособия

Афанасьева, Т. В.

А 94 Алгоритмы и программы : учебное пособие / Т. В. Афанасьева, Ю. Е. Кувайскова, В. А. Фасхутдинова. – Ульяновск : УлГТУ, 2011. – 227 с.

ISBN 978-5-9795-0857-3

Содержание учебного пособия включает информацию по составлению алгоритмов и практическому программированию на языке Turbo Pascal при решении сложных задач, которая может быть использована для подготовки к выполнению контрольных, лабораторных и практических заданий по курсам «Программные и аппаратные средства информатики», «Информационные технологии», «Алгоритмы и структуры данных».

Рассматриваются основные этапы решения задач на ЭВМ, основные понятия алгоритмов, средства их предствления, алгоритмы обработки данных, составление программ, реализующих разветвленные, циклические алгоритмы, работа с файлами, множествами и модулями, записями, процедурами, функциями и массивами, объектами.

Пособие предназначено для студентов вузов дневной, вечерней, заочной и дистанционной форм обучения.

УДК 004.4 (075)
ББК 32. 973 – 018.2я7

ISBN 978-5-9795-0857-3

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Компьютерные и информационные технологии стали неотъемлемой частью современного общества, экономики, транспорта, бизнеса, образования и науки.

В современных условиях необходимо владение комплексом компетенций по применению и управлению информационными и программными ресурсами, включающим компетенции, обеспечивающие в значительной степени решение новых задач на базе будущих информационных технологий. Представляется, что комплекс компетенций в области применения и управления информационными и программными ресурсами следует развивать на основе универсальных, базовых компетенций, имеющих междисциплинарный характер, позволяющих на основе научного подхода проектировать информационные продукты. К таким компетенциям относят способность к анализу, синтезу, пониманию, моделированию, знанию технологий – все то, что относится к понятию «системный подход». Владение таким комплексом компетенций позволит использовать его как средство решения нестандартных и новых задач.

Процесс решения задачи имеет последовательность этапов, включающую как анализ, так и синтез, при этом эти процедуры выполняются для каждого этапа на своем уровне детализации модели решения задачи: информационном, алгоритмическом, программном. Такой целостный, системный подход к решению задач с использованием программных средств и предлагается авторами данной работы, которые ориентируют студента на важность развития актуальных компетенций и предлагают экспериментальные средства для их оценки и самооценки.

Учебное пособие состоит из двух частей: основы алгоритмизации и программирование на языке Turbo Pascal.

Первая часть предназначена для тех, кто не умеет, но стремится научиться алгоритмически решать задачи. Умение составлять алгоритмы позволяет получить корректное решение, которое может быть использовано в качестве алгоритмической модели решения, независимой от используемых программных средств.

Вторая часть пособия посвящена изучению языка Turbo Pascal, основанного на принципах структурного программирования и используемого во многих вузах в качестве базового языка для обучения программированию в курсе «Информатика».

В разделах пособия приводятся краткие сведения о рассматриваемых алгоритмах и программах, задания для самостоятельного выполнения, контрольные вопросы. Задания для самостоятельного выполнения содержат цель, требования, а также порядок выполнения и перечень формируемых компетенций.

Отличительной чертой данного пособия от аналогичных изданий является ориентация на формирование внутренней мотивации к выполнению самостоятельных заданий за счет понятия «полезности для студента». Кроме навыков и умений выполнение самостоятельных работ позволит развить базовые (системные, инструментальные) и личностные компетенции. Таким образом, имеется возможность оценивать результат выполнения работ не только по критериям качества самостоятельной работы, но и в терминах проявленных личностных компетенций, таких как способность планировать свою работу, способность самостоятельно решать стоящие задачи, способность к самооценке. Такие компетенции развиваются в деятельности, и в данном пособии эта деятельность представлена комплексом самостоятельных работ, которые, как и в реальной жизни, ограничены по времени. Временной параметр позволит оценить эффективность деятельности по выполнению с позиции компетенций по планированию, организации и ответственности.

Исходя из указанных предпосылок, авторы предлагают следующую схему оценки качества выполнения работ студентом: развитие системных компетенций, развитие инструментальных компетенций, развитие личностных компетенций.

При оценке развития системных компетенций оцениваются ответы на контрольные вопросы, при оценке инструментальных компетенций оцениваются структура, объем и содержание выполненных работ. При оценке личностных компетенций оцениваются время выполнения работ (способность к планированию, организации, ответственность), качество выполнения лабораторных работ (стремление к качеству результата), качество выполнения индивидуальных работ (способность самостоятельно решать задачи, самооценка).

Пособие подготовлено на базе многолетнего опыта проведения занятий по информационным технологиям.

Часть I. Основы алгоритмизации

1. ЭТАПЫ РЕШЕНИЯ ЗАДАЧ НА ЭВМ

Решение любой проблемы является творческим процессом, состоящим из нескольких, зачастую повторяющихся этапов. Проблемы, требующие решения средствами прикладного программного обеспечения ЭВМ, относятся к классу задач, решение которых включает большой объем рутинных операций для обработки данных или не может быть получено без использования ЭВМ.

Для обоснованного решения таких проблем рекомендуется применять системный подход, основанный на решении задач декомпозиции, анализа и синтеза. Использование системного подхода, особенно при решении новых проблем, имеет ряд преимуществ. К ним можно отнести рассмотрение проблемы как системы элементов, данных, между которыми существуют некоторые связи, отношения. Часть отношений является отражением структурных свойств данных, а часть отношений определяет преобразования, позволяющие получить решение проблемы. В рамках системного подхода решение проблемы находится последовательно посредством выделения элементов и определения отношений между ними. Такой подход позволяет рассмотреть проблему в целом, с учетом различных связей, отношений, предусмотреть ситуации, оказывающие влияние на решение. Следовательно, полученное решение проблемы будет надежным и устойчивым. С другой стороны, приобретение умений и навыков применения системного подхода позволит сформировать и развить наиболее важные и востребованные компетенции в анализе проблем и синтезе решений.

Рассмотрим обобщенный процесс решения проблемы на основе системного подхода как последовательность, состоящую из трех этапов, приведенных ниже:

А. Анализ постановки задачи и ее предметной области:

1) понимание постановки и требований исходной задачи, определение предметной области, для которой поставлена задача;
2) анализ предметной области; выявление данных, которые фиксируют входную и выходную информацию (определение их структуры и свойств);
3) определение отношений между данными, задание условий и ограничений, накладываемых на эти отношения.

Б. Формальное моделирование решения задачи:

1) формирование основной идеи;

2) выбор и применение формальной системы для описания метода решения задачи;
3) построение алгоритмов, реализующих выбранный метод;
4) выбор оптимального алгоритма для заданных ранее условий и ограничений.

В. Практическое решение задачи:

1) определение технологий, средств и исполнителя решения задачи;
2) реализация оптимального алгоритма средствами и технологиями выбранного исполнителя решения задачи;
3) анализ решения и полученных результатов.

1.1. Анализ постановки задачи и ее предметной области

При решении любой задачи важным является первый этап, связанный с анализом и пониманием постановки задачи. На этапе анализа необходимо детализировать постановку задачи. Основные вопросы, на которые требуется ответить, можно сформулировать следующим образом:

- Какие данные являются исходными?
- Какие данные являются результирующими?
- Какие отношения между исходными данными и требуемым результатом?
- Являются ли исходные данные достаточными?
- К какому виду следует отнести исходные данные?
- К какому типу следует отнести исходные и результирующие данные?

На этом этапе уточняется постановка задачи, после чего выявляются отдельные явления, объекты, процессы, их связи и зависимости, то есть создается структурная модель предметной области.

Модель – это упрощенное представление о реальном объекте, процессе или явлении.

На этапе анализа постановки задачи определяются такие понятия, как исходные и результирующие данные, абстрактно представляющие информацию о предметной области решаемой задачи в виде значений.

Исходные данные должны быть полными, т. е. содержать данные, необходимые и достаточные для решения задачи. Если данные неполные, необходимо приложить дополнительные усилия для сбора дополнительных сведений; эта ситуация может также возникнуть на последующих этапах при выборе метода решения. Данные могут быть определены на основе свойств данных, которые задают их вид и тип.

Различают исходные данные трех видов: постоянные, условно-постоянные и переменные.

Постоянные исходные данные – это данные, которые сохраняют свои значения в процессе решения задачи (математические константы, координаты неподвижных объектов) и не зависят от внешних факторов.

Условно-постоянные данные – это данные, которые могут иногда изменять свои значения; причем эти изменения не зависят от процесса решения задачи, а определяются внешними факторами (величина подоходного налога, курс валют, количество дней в году).

Переменные данные – это данные, которые изменяют свои значения в процессе решения задачи.

В дальнейшем необходимо как для исходных, так и для результирующих данных выбрать определенный тип данных. **Тип данных** определяет множество значений и множество допустимых операций к ним. На этом этапе важно не только классифицировать данные по отношению к процессу решения, но определить их наименование, тип, структуру и ограничения, накладываемые на значения исходя из постановки задачи. Желательно также определить допустимые и недопустимые операции по отношению к различным типам исходных данных.

Данное относят к простому типу, если в любой момент времени оно определяет одно и только одно значение.

На рис. 1 представлена классификация типов данных по структурному признаку.

Классификация данных по структурному признаку

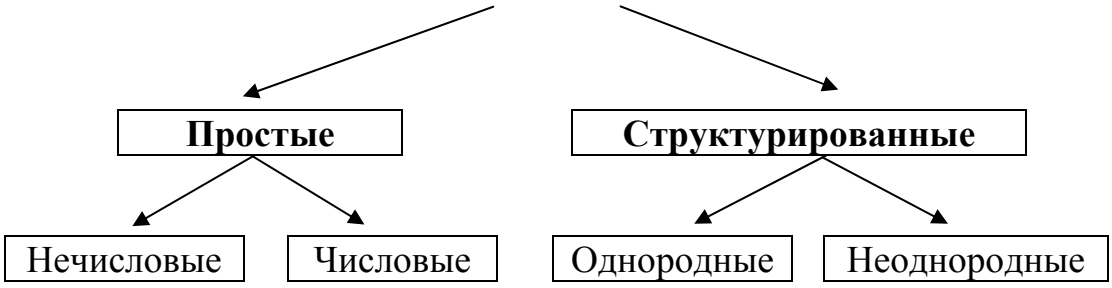


Рис. 1. Классификация типов данных

Например, требуется вычислить площадь поверхности некоторого тела. Очевидно, что для представления информации о вычисляемой площади поверхности некоторого тела достаточно использовать данное простого числового типа. Простые данные определяют такое отношение: одно имя – одно значение.

Структурированные данные отличаются от простых тем, что к ним применимо другое отношение: одно имя – много значений. Если все элементы, входящие в такую структуру, однотипны, то такая структура называется **однородной**, в противном случае – **неоднородной**.

Классическим примером однородной структуры является некоторая последовательность однотипных данных, заданная в виде массива значений, таких как, например (2, 51, 3, 7, 88).

Массив характеризуется фиксированным количеством входящих в него значений, эти значения упорядочены по номерам следования. **Номер элемента** в массиве является средством адресации, доступа к конкретному элементу, например в вышеприведенном массиве чисел значение 51 имеет номер два, а значение 7 – номер четыре. Неоднородная структура в отличие от однородной содержит значения различных типов, относящихся к одному понятию или объекту, и, значит, такое структурированное данное несет в себе больше информации. Для представления неоднородных структур используют запись. **Запись** – это структура, предназначенная для представления данных преимущественно различного типа. Значения, составляющие запись, называются **полями записи**. По-другому запись можно представить как совокупность полей, каждое из которых имеет свое наименование. Имя поля является средством адресации, доступа к значению этого поля в записи.

Рассмотрим простой пример. Задача заключается в определении в некоторой стране города с максимальным количеством жителей. Данные, которые необходимо проанализировать, это нечисловые данные, содержащие информацию о названии города, и числовые данные, содержащие информацию о численности в этом городе. В качестве структуры, содержащей данные о названии города и количестве в нем жителей, следует выбрать неоднородную структуру – запись, пример которой изображен в таблице 1. Запись содержит два поля: Название города и Количество жителей. В первой строке табл. 1 указаны названия полей, во второй строке – типы полей, в третьей строке значения полей, образующие запись.

Таблица 1

Пример записи

Название города	Количество жителей
Строковый тип	Числовой тип
Москва	8 578 676

В качестве структуры, содержащей информацию о множестве городов рассматриваемой страны, можно выбрать однородную структуру типа массив, состоящий из нескольких однотипных записей табл. 1.

Определение отношений между данными, условиями и ограничениями, накладываемыми на значения данных, зависит от конкретной постановки задачи и требований пользователя.

В результате анализа постановки задачи создается структурная модель, в которой определены данные, их свойства через виды, типы, ограничения и возможные соотношения между ними.

Задания для самостоятельного выполнения

Цель заданий. Приобрести умения в создании структурной модели задач. Сформировать компетенции анализа постановки простых задач.

Порядок выполнения. Опишите и обоснуйте структурную модель предлагаемых ниже задач посредством перечисления данных, их свойств и возможных отношений между ними.

1. В книжном магазине вы желаете купить три книги. У вас имеется небольшая сумма наличных денег и пластиковая карта, на счету которой большая сумма денег. Вам бы не хотелось сегодня пользоваться пластиковой картой, так как Вы в дальнейшем запланировали крупную покупку. Определите способ покупки.
2. Определите вариант пути от дома до места учебы, в зависимости от времени выхода из дома и времени начала учебных занятий.
3. В избирательной компании в органы власти участвуют две партии: зеленых и прозрачных. Какая информация будет опубликована в СМИ по итогам голосования?
4. Подсчитать количество цифр в целом числе.
5. В процессе обучения студент группы сдавал экзамены. Определить средний балл за сданные им экзамены.
6. Некоторая группа студентов сдавала экзамен по дисциплине «Редакторское дело». Определить средний балл группы по этой дисциплине.
7. В группе учатся как девушки, так и юноши. Определить, кто лучше сдал экзамен по дисциплине «Информатика» – юноши или девушки?
8. Определить среднюю и общую стоимость книг в книжном магазине для каждого автора.
9. Определить объем выпуска книжной продукции издательства «Эдельвейс» в печатных листах, учитывая коэффициент перевода авторских страниц в печатные листы.
10. Определить общую стоимость выпуска книжной продукции издательства «Прогресс» с учетом цены и тиража.

Контрольные вопросы

1. К какому классу задач относятся проблемы, требующие решения средствами прикладного программного обеспечения ЭВМ?
2. Какой подход рекомендуется применять для обоснованного решения проблем?
3. На решении каких задач основан системный подход?

4. Использование системного подхода особенно при решении новых проблем имеет ряд преимуществ. Перечислите их.
5. Опишите обобщенный процесс решения проблемы на основе системного подхода.
6. Перечислите основные вопросы, на которые требуется ответить при анализе постановки задачи.
7. Дайте определение модели.
8. Как можно определить структурную модель?
9. Что понимают под свойствами данных?
10. Какие существуют виды исходных данных?
11. Что определяет тип данных?
12. Приведите классификацию типов данных по структурному признаку.
13. Приведите примеры типов и видов данных.

1.2. Формальное решение задачи

После проведения анализа постановки задачи, выявления данных, их вида, типа и структуры можно приступить к построению, **синтезу** формальной модели решения задачи. Это наиболее важный этап в процессе решения задачи. На этом этапе детализируются отношения между исходными и результирующими данными, условиями и ограничениями, накладываемыми на значения данных, зависящими от конкретной постановки задачи и требований пользователя, а также генерируется идея, подход к решению задачи. Для выбора и описания метода решения необходимо выбрать некоторую формальную систему. Обычно, исходя из постановки задачи, можно сразу определить один или несколько видов моделей, подходящих для описания и моделирования решения вашей задачи: аналитические, геометрические, структурные, логические и др.

Наиболее распространенными и хорошо изученными являются **математические модели**. Например, в качестве математической модели звезды можно использовать систему уравнений, описывающую процессы, происходящие в недрах звезды. Математической моделью другого рода являются математические соотношения, позволяющие рассчитать оптимальный план работы предприятия. К основным достоинствам математических моделей относятся хорошо изученные и широко применяемые математические методы решения большого класса задач, что значительно облегчает формирование основной идеи и выбор методов решения задачи. В дальнейшем в пособии рассматриваются только математические модели.

Приступая к разработке формальной модели решения задачи, следует попытаться решить задачу для конкретных входных данных, затем

обобщить полученное решение на основе его анализа для любых значений входных данных.

К основным вопросам, на которые целесообразно обратить внимание на этапе формального решения задачи, следует отнести:

- Что дано? Как можно обозначить исходные данные исходя из их типа, определенного на предыдущем этапе?
- Что неизвестно? Как можно обозначить результирующие данные исходя из их типов?
- Какие условия, ограничения можно определить для исходных и результирующих данных? Как их записать?
- Какие соотношения между исходными данными и результатами можно записать еще? Как их записать?
- Какая формальная модель больше всего подходит для решения этой задачи?
- Известен ли метод решения для такой или родственной задачи? Если метод известен, как его записать в терминах введенных ранее обозначений исходных, результирующих данных и условий? Требуются ли промежуточные данные? К какому типу их следует отнести?

Если метод решения задачи неизвестен, решаемую задачу можно разделить на несколько задач, более простых, известных для Вас. Введение искусственных, дополнительных переменных является другим **способом поиска метода решения задачи**. Попытка переформулировать постановку задачи может оказать действенную помощь в поиске подходящего метода решения задачи, также как и консультации, и обращение к информационным ресурсам. Иногда метод решения задачи является настолько простым и очевидным, что его и не осознают в качестве метода, а иногда метод решения представляет собой сложные математические выводы и формулы, занимающие несколько страниц.

Пример 1. Постановка задачи. Требуется определить пригодность для проведения учебных занятий данной аудитории.

Решение.

Этап 1. Анализ постановки задачи и ее предметной области.

В результате анализа предметной области, выявляем, что эта предметная область связана с образовательным процессом. Постановка задачи может быть переформулирована следующим образом: определить, подходит ли некоторая аудитория для проведения занятий группы учеников при некоторой норме площади для каждого ученика. Введем обозначения для входных и выходных данных. Исходные данные: A – ширина аудитории, B – ее длина, K – количество учеников в группе, N – допустимое минимальное количество квадратных метров для одного ученика (норма), M – количество парт в аудитории. В качестве выходных данных будут выступать сообщения: «Аудитория может быть использована для проведения учебных занятий» и «Аудитория не может быть использована для проведения учебных занятий».

Этап 2. Формальное решение.

Определим отношения между входными и выходными данными. Введем дополнительные понятия: S – площадь аудитории, C – требуемая по нормам площадь для проведения занятий в группе из K учеников, D – требуемое количество парт для обучения группы из K учеников. Опишем соотношения между входными и выходными данными, используя математические зависимости. Математическая модель:

$$S = A \times B$$
$$C = N \times K, \quad S \geq C, \quad K \leq 2 \times D.$$

Задача построения алгоритмов, реализующих выбранные методы решения задачи, детализирует и визуализирует процесс ее решения. **Алгоритмизация** позволяет определить последовательность преобразования исходных данных в результирующие и уже на этом этапе оценить эффективность решения, уточнить методы решения для различных потоков входных данных и выявить некоторые ошибки. Поэтому особенно важно уделить серьезное внимание алгоритмизации решения заданной задачи. Вопросы корректного построения алгоритмов являются наиболее значимыми для большого класса вычислительных задач со сложными структурами данных, со сложной логикой и большим объемом вычислительных операций, решаемых программными средствами ЭВМ. В этом случае алгоритмическое представление, алгоритмическая модель решения задачи, созданная на основе метода решения рассматривается как промежуточное звено между методом решения и его программной реализацией. В принципе, желательно понимать, что между методом решения задачи и реализующим алгоритмом существует отношение «один метод – множество алгоритмов», поэтому необходимо принимать решение о выборе наиболее адекватного, оптимального алгоритма, удовлетворяющего заданным условиям и ограничениям.

Следует отметить, что алгоритмическое решение существует не для всех классов задач, к таким задачам относятся, например, задачи искусственного интеллекта.

Задания для самостоятельного выполнения

Цель заданий. Приобрести умения в синтезе формальной модели решения задач. Сформировать компетенции анализа и синтеза при решении простых задач.

Порядок выполнения. Опишите и обоснуйте формальную модель решения предлагаемых ниже задач посредством обозначения данных, их свойств и возможных математических соотношений, ограничений и условий между ними.

1. В книжном магазине вы желаете купить три книги. У вас имеется небольшая сумма наличных денег и пластиковая карта, на счету которой большая сумма денег. Вам бы не хотелось сегодня пользоваться

пластиковой картой, так как Вы в дальнейшем запланировали крупную покупку. Определите способ покупки.
2. Определите вариант пути от дома до места учебы, в зависимости от времени выхода из дома и времени начала учебных занятий.
3. В избирательной компании в органы власти участвуют две партии: зеленых и прозрачных. Какая информация будет опубликована в СМИ по итогам голосования?
4. Подсчитать количество цифр в целом числе.
5. В процессе обучения студент группы сдавал экзамены. Определить средний балл за сданные им экзамены.
6. Некоторая группа студентов сдавала экзамен по дисциплине «Редакторское дело». Определить средний балл группы по этой дисциплине.
7. В группе учатся как девушки, так и юноши. Определить, кто лучше сдал экзамен по дисциплине «Информатика» – юноши или девушки?
8. Определить среднюю и общую стоимость книг авторов в книжном магазине.
9. Определить объем выпуска книжной продукции издательства «Эдельвейс» в печатных листах, учитывая коэффициент перевода авторских страниц в печатные листы.
10. Определить общую стоимость выпуска книжной продукции издательства «Прогресс» с учетом цены и тиража.

Контрольные вопросы

1. Что понимается под синтезом формальной модели решения задачи?
2. Приведите примеры математических моделей.
3. Перечислите основные вопросы, на которые требуется дать ответ при синтезе формальной модели решения задачи.
4. Какие существуют способы поиска метода решения задач?
5. В чем заключается алгоритмизация?
6. Укажите место алгоритмической модели в процессе решения задачи.
7. Приведите пример формальной модели решения задачи.

1.3. Практическое решение задачи

Этап практического решения является наиболее интересным этапом решения задач на ЭВМ. Это объясняется возможностью программного экспериментирования, в процессе которого возможна проверка и уточнение решений, полученных на предыдущих этапах. С другой стороны, практическое решение позволяет расширить знания в области прикладного программного обеспечения, которое выступает эффективным

средством, инструментом решения задачи. На этом этапе решения задачи также следует понимать, что между алгоритмом решения задачи и его программной реализацией существует аналогичное отношение «один алгоритм – множество программных реализаций». Это отношение указывает на проблему выбора соответствующего программного средства (исполнителя решения задачи) и технологии из имеющихся в программном обеспечении.

Практическое решение позволяет получить долгожданное решение поставленной задачи. Это решение может отличаться от ожидаемого, может не соответствовать ему. Такая ситуация наиболее вероятна, если при решении задачи не используют системный подход, описанный выше, который позволяет предусмотреть и устранить возможные ошибки на более ранних стадиях. Известно, что в целях экономии времени возникает желание приступить сразу к практическому решению задачи, минуя этапы детального анализа постановки задачи, формального и алгоритмического моделирования ее решения, однако, так поступать целесообразно только после того, как будет накоплен значительный опыт в понимании и применении компетенций анализа и синтеза решений различных задач.

Тем не менее, всегда следует учитывать, что при анализе решения и полученных результатов, может выявиться и погрешность в примененном методе и алгоритме решения, которые могут быть улучшены или заменены на другие, рассматриваемые как альтернативные на этапе формального решения задачи.

2. АЛГОРИТМЫ

2.1. Основные понятия

Понятие «алгоритм» появилось в девятом веке и связано с именем математика Аль-Хорезми, который сформулировал правила выполнения четырех арифметических действий над многозначными числами.

В настоящее время понятие алгоритма – одно из фундаментальных понятий науки информатики. С одной стороны, алгоритм является предметом изучения такой отрасли математики как теория алгоритмов, с другой стороны, в информатике существует неформальное определение алгоритма, и алгоритмизация выступает в качестве общего метода информатики.

Алгоритм – это точно определенная последовательность действий для некоторого исполнителя, выполняемых по строго определенным правилам и приводящих через некоторое количество шагов к решению задачи.

Исполнитель алгоритмов определяет элементарные действия, из которых формируется алгоритм. Отдельные действия, составляющие алгоритм, называются **операциями**. При этом под операцией понимается как какое-то единичное действие, например, сложение, так и группа взаимосвязанных действий.

Основными особенностями любого алгоритма являются решение задачи в обобщенном виде и возможность выполнять действия по решению задачи для конкретных значений (не только человеку, но и различным техническим устройствам (исполнителям)). Основным исполнителем несложных алгоритмов является человек. Достаточно вспомнить последовательность действий для решения систем линейных уравнений, вычисления корней уравнений.

При решении сложных задач исполнителем является программное обеспечение ЭВМ, и составление алгоритма решения задачи является необходимым этапом, детализирующим метод решения для дальнейшей реализации.

Свойства алгоритма:

определенность *– выполнив очередное действие, исполнитель должен точно знать, что ему делать дальше;*

дискретность *– прежде, чем выполнить определенное действие, надо выполнить предыдущее;*

массовость *– по одному и тому же алгоритму решаются однотипные задачи и неоднократно;*

понятность *– алгоритм строится человеком для конкретного исполнителя и должен быть ему понятен. Это облегчает его проверку и модификацию при необходимости;*

результативность *– алгоритм всегда должен приводить к результату.*

Мы видим, что в процессе формального решения задачи ее решение сначала описывается на языке математики в виде системы формул, а затем на языке алгоритмов в виде некоторого процесса, в котором используются ранее определенные математические формулы и условия их выполнения.

Таким образом, алгоритм рассматривается как средство описания процесса решения задачи, как связующее, промежуточное звено в цепочке «метод решения – реализующая программа».

2.2. Средства представления алгоритмов

Построение алгоритмов подчиняется отдельным законам, использует специальный язык для обозначений основных понятий и терминов, активно используется для описания методов решения задач.

Алгоритм, реализующий решение задачи, можно задать различными способами на основе различных языковых средств: графических, текстовых, табличных.

Графические средства представления алгоритмов имеют ряд преимуществ благодаря визуальности и явному отображению процесса решения задачи. Алгоритмы, представленные на языке графических объектов, получили название **визуальные алгоритмы**.

Текстовое описание алгоритма является достаточно компактным и может быть реализовано на абстрактном или реальном языках программирования в виде программы для ЭВМ. Таблицы значений представляют алгоритм неявно, как некоторое преобразование конкретных исходных данных в выходные.

Табличный способ описания алгоритмов может быть с успехом применен для проверки правильности функционирования разработанного алгоритма на конкретных тестовых наборах входных данных, которые вместе с результатами выполнения алгоритма фиксируются в «таблицах трассировки».

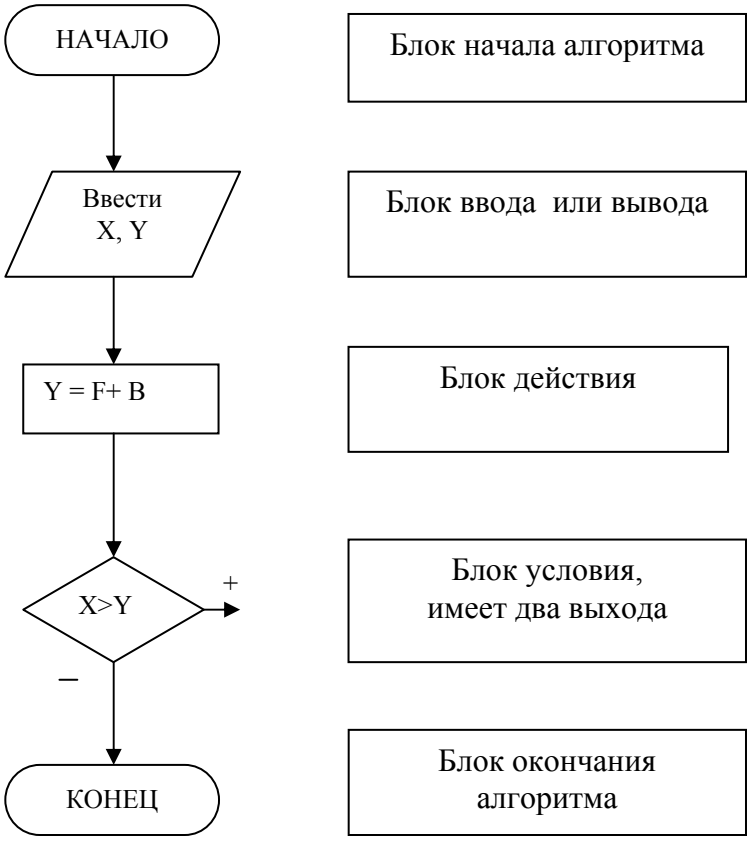


Рис. 2. Основные блоки визуальных алгоритмов

Таким образом, все три способа представления алгоритмов можно считать взаимодополняющими друг друга. На этапе проектирования алгоритмов наилучшим способом является графическое представление, на этапе проверки алгоритма – табличное описание, на этапе применения – текстовая запись в виде программы.

2.3. Визуальные алгоритмы

При проектировании визуальных алгоритмов используют следующие графические блоки, представленные на рис. 2.

Рассмотрим общие правила при проектировании визуальных алгоритмов:

- в начале алгоритма должны быть блоки ввода значений входных данных;
- после ввода значений входных данных могут следовать блоки обработки и блоки условия;
- в конце алгоритма должны располагаться блоки вывода значений выходных данных;

- в алгоритме должен быть только один блок начала и один блок окончания;
- связи между блоками указываются направленными или ненаправленными линиями.

Этап проектирования алгоритма следует за этапом выбора формальной модели и методов решения задачи, на котором определены входные и выходные данные, а также зависимости между ними.

При построении алгоритмов для сложной задачи используют системный подход с использованием последовательной декомпозиции решаемой задачи на ряд подзадач до тех пор, пока не будут определены задачи, для которых алгоритмическое решение известно или оно может быть оперативно найдено.

Одним из системных методов разработки алгоритмов является **метод структурной алгоритмизации**. Этот метод основан на визуальном представлении алгоритма в виде последовательности управляющих структурных фрагментов. Выделяют три базовые управляющие процессом обработки информации структуры: композицию, альтернативу и итерацию. С помощью этих структур можно описать решение большого класса задач обработки информации.

Композиция (следование) – это линейная управляющая конструкция, не содержащая альтернативу и итерацию. Она предназначена для описания единственного процесса обработки информации.

Альтернатива – это нелинейная управляющая конструкция, не содержащая итерацию. Она предназначена для описания процессов решения различных задач обработки информации, выбор которых зависит от значений входных данных.

Итерация – это циклическая управляющая конструкция, которая содержит композицию и ветвление. Она предназначена для организации повторяющихся процессов обработки последовательности значений данных.

Линейные алгоритмы не содержат блока условия. Они предназначены для представления линейных процессов. Такие алгоритмы применяют для описания обобщенного решения задачи в виде последовательности модулей.

Пример линейного алгоритма приведен на рис. 3.

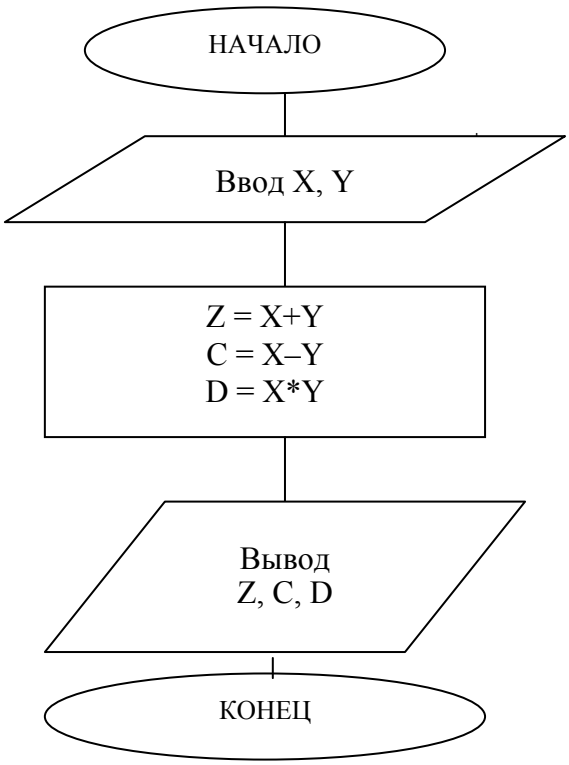


Рис. 3. Пример линейного визуального алгоритма

В соответствии с наличием в алгоритмах управляющих структур композиции, альтернативы и итерации алгоритмы классифицируют на линейные, разветвленные и циклические алгоритмы.

2.4. Разветвленные алгоритмы

Разветвленные алгоритмы в своем составе содержат блок условия и различные конструкции ветвления. **Ветвление** – это структура, обеспечивающая выбор между альтернативами.

Каждая управляющая структура ветвления имеет один вход и один выход. Ветвления содержат блок условия, в котором записывают логические условия, такие как A > C, X <= Y. В зависимости от значений переменных A, C в управляющей структуре ветвления на рис. 4, условие A > C принимает значение «истина» или «ложь» и процесс вычислений включает блок действия Z = A или Z = C. Аналогично происходит и в управляющей структуре неполного ветвления (рис. 4, б). Только в этом случае, если условие X <= Y истинно, то выполняется действие C = X, в противном случае никаких действий не выполняется.

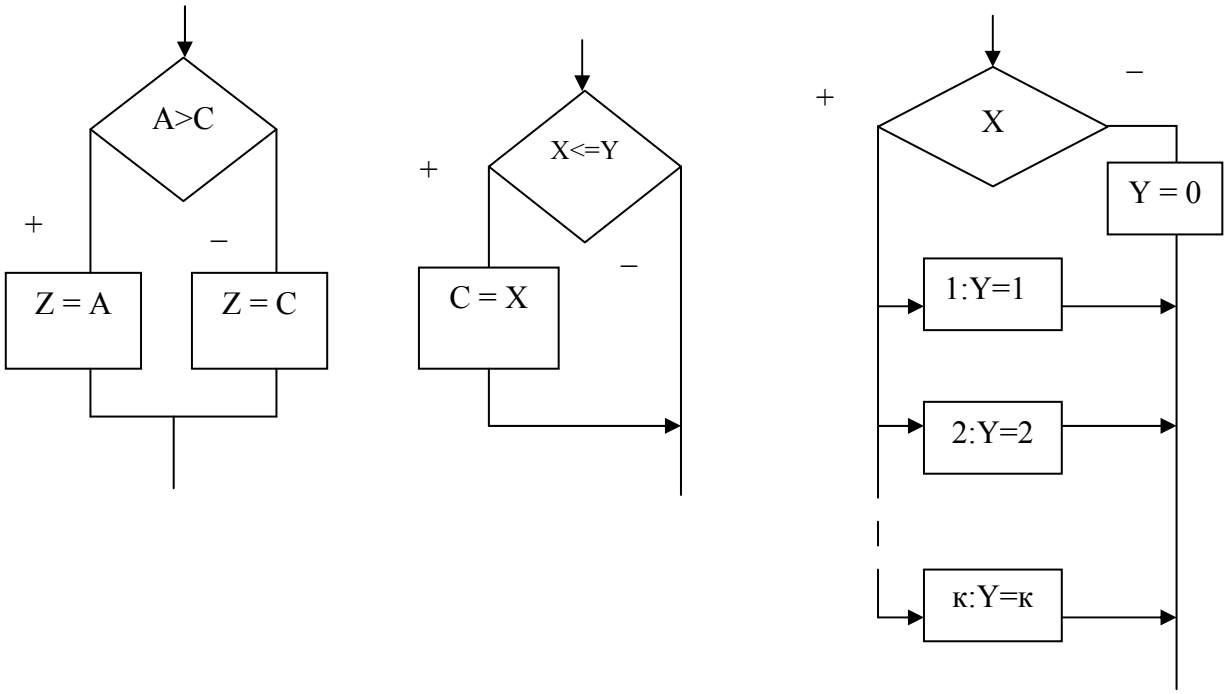


Рис. 4. Структуры разветвленных алгоритмов
а) ветвление; б) неполное ветвление; в) многоальтернативный выбор

В управляющей структуре многоальтернативный выбор (рис. 4, в) в блоке условия записывается переменная, в данном случае X, которая может принимать различные значения. Если значение переменной X совпадет с одним из значений в блоке действия, то выполняются действия, записанные в этом блоке. Например, если X = 1, то выполнится действие Y = 1. Если значение X не совпало ни с одним из значений, указанных в блоках справа, то выполняется действие в блоке слева, которого также, как и в неполном ветвлении, может и не быть.

Пример 2. Составить алгоритм нахождения минимального значения из 3 чисел.

Решение. Для определения минимального значения будем использовать проверку пары значений. Визуальные разветвленные алгоритмы приведены на рис. 5, 6, 7. Эти алгоритмы используют для обозначения чисел переменные A, B, C и различные подходы для анализа исходных данных.

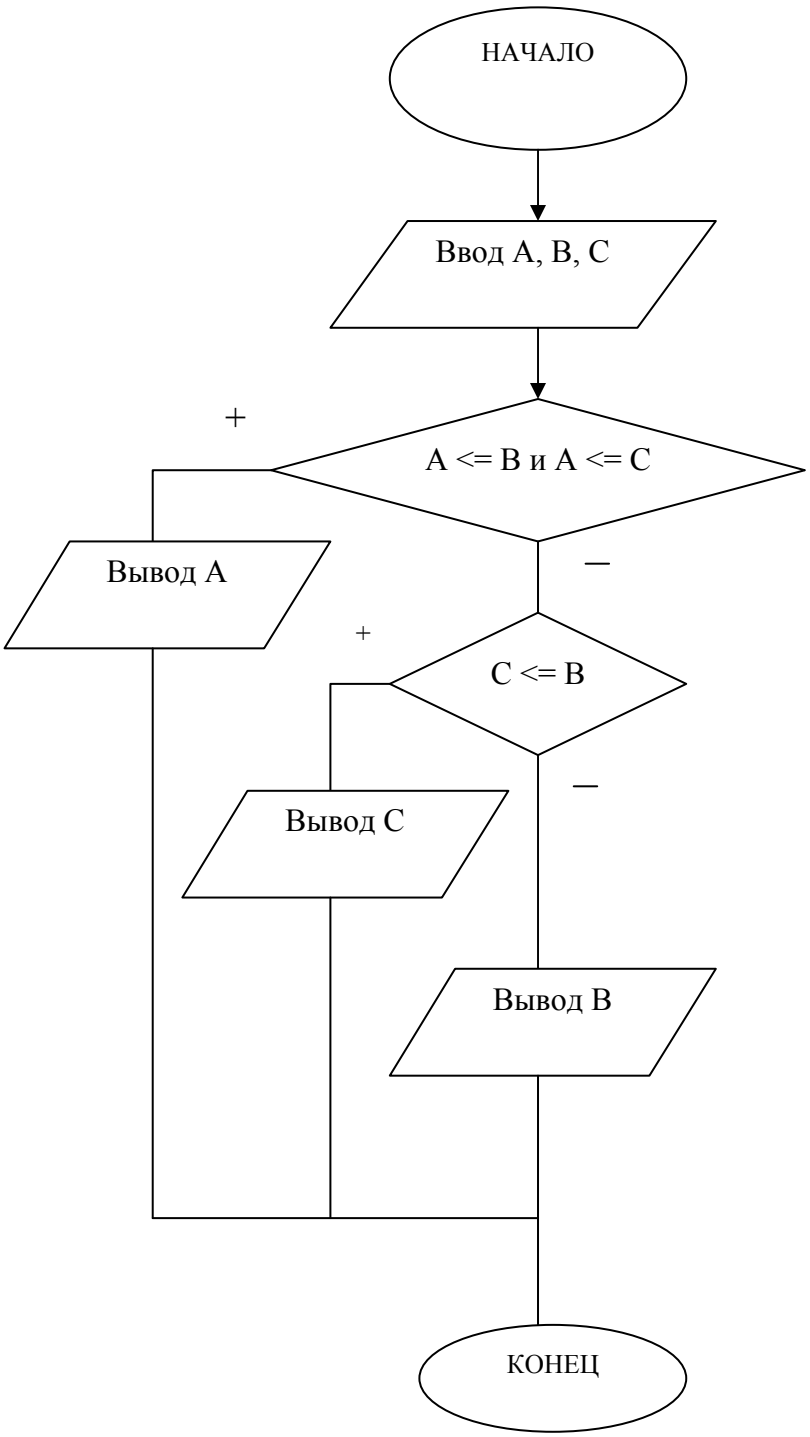


Рис. 5. Поиск минимального значения из трех чисел A, B, C
при помощи двойного сравнения

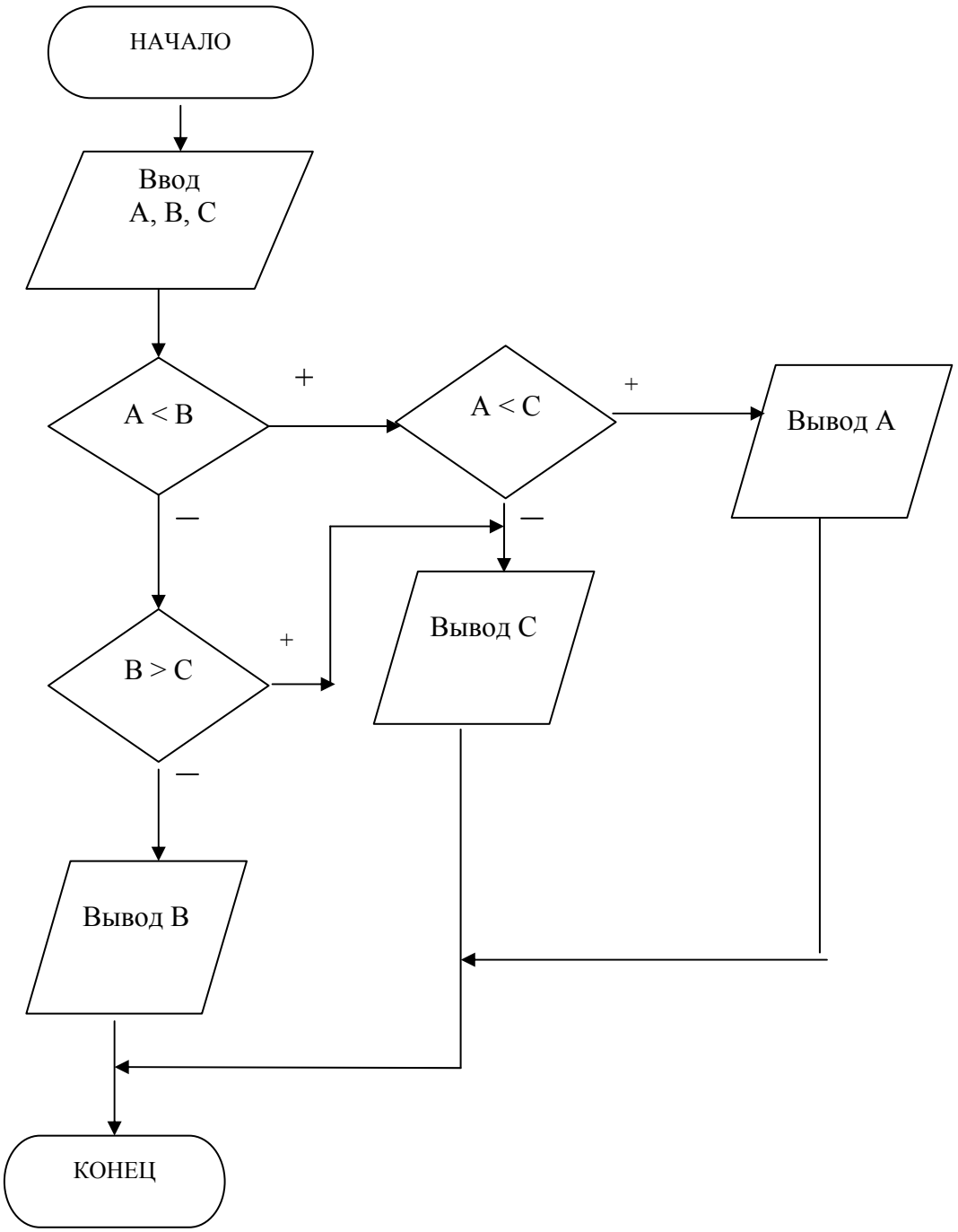


Рис. 6. Поиск минимального числа из трех A, B, C.
Метод последовательного сравнения

Пример 3. Составить алгоритм определения: находится ли точка M с координатами X, Y на окружности радиуса R.

Решение. Визуальный алгоритм приведен на рис. 8. Для решения в нем используется математическая модель в виде формулы окружности $R^2 = X^2+Y^2$.

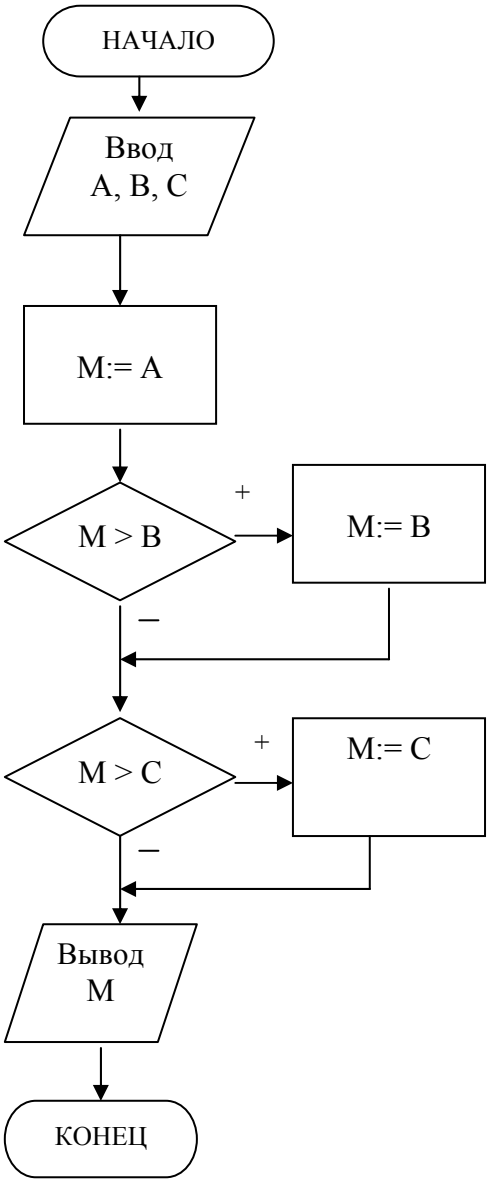


Рис. 7. Поиск минимального числа из трех A, B, C. Метод сравнения с промежуточной переменной M

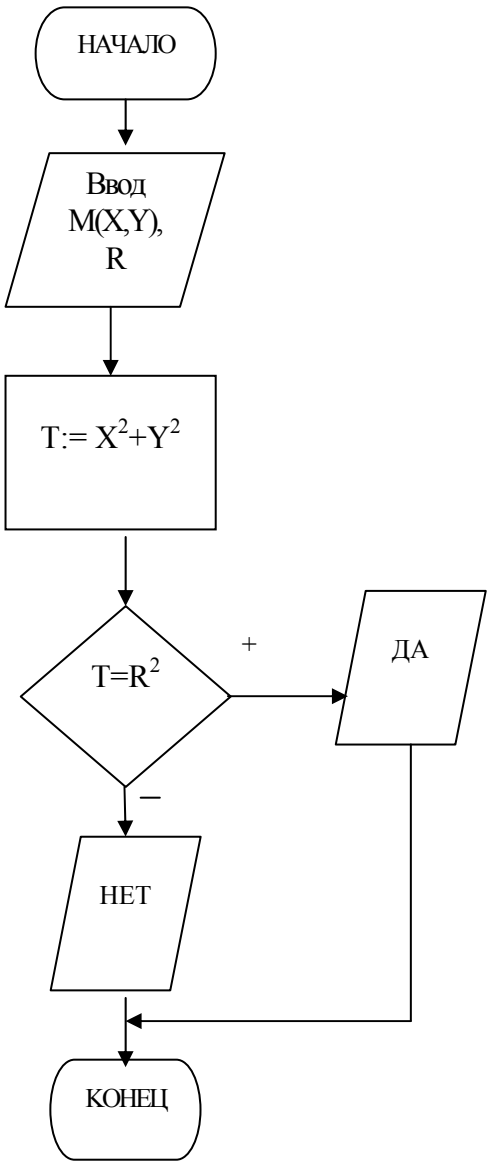


Рис. 8. Определение положения точка M с координатами X, У на окружности радиуса R

Пример 4. Составить алгоритм определения корней уравнения $X^2+B*X+C=0$.

Решение. При составлении этого алгоритма надо рассмотреть случаи, когда уравнение не имеет корней и когда имеется только один корень. Обозначим корни уравнения через переменные X1, X2. D – промежуточная переменная для вычисления дискриминанта.

Алгоритм вычисления корней уравнения заданного вида приведен на рис. 9.

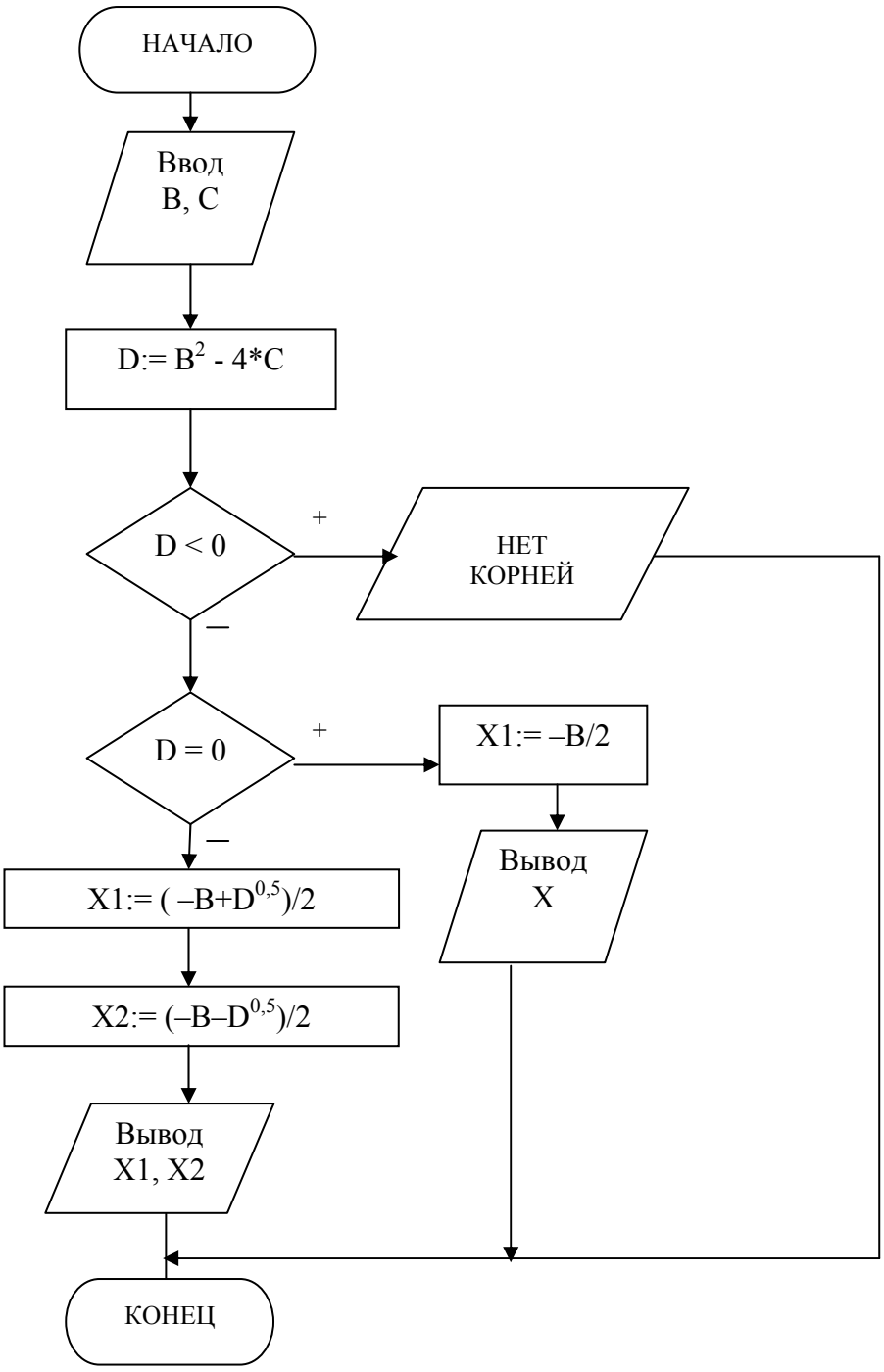


Рис. 9. Алгоритм вычисления корней уравнения $X^2+B*X+C=0$

Задания для самостоятельного выполнения

Цель заданий. Приобрести умения в синтезе формальной и алгоритмической моделей решения задач. Сформировать компетенции анализа и синтеза алгоритмических моделей при решении простых задач.

Порядок выполнения. Составить формальные и алгоритмические модели решения следующих задач:

1. В книжном магазине вы желаете купить три книги. У вас имеется небольшая сумма наличных денег и пластиковая карта, на счету которой

большая сумма денег. Вам бы не хотелось сегодня пользоваться пластиковой картой, так как Вы в дальнейшем запланировали крупную покупку. Определите способ покупки.

2. Определите вариант пути от дома до места учебы, в зависимости от времени выхода из дома и времени начала учебных занятий.

3. В избирательной компании в органы власти участвуют две партии: зеленых и прозрачных. Какая информация будет опубликована в СМИ по итогам голосования?

4. Для двух чисел X, Y определить, являются ли они корнями уравнения $A*P^4+D*P^2+C=0$.

5. Составить алгоритм для ответов на вопросы, возникающие на этапе формального решения задачи.

6. Составить алгоритм обобщенного решения задач на основе системного подхода.

7. Определить, является ли точка с координатами X, Y точкой пересечения диагоналей квадрата со стороной R, одна вершина которого расположена в начале координат.

8. Определить значения функции в зависимости от значения аргумента

$$У = \begin{cases} a*x^2, & \text{если } x > 10; \\ 1/x, & \text{если } -10 \le x \le 10; \\ Sin(x), & \text{если } x < 10. \end{cases}$$

Контрольные вопросы

1. Что называют алгоритмом?
2. Определите понятие «исполнитель алгоритмов».
3. Каковы особенности алгоритмов?
4. Перечислите свойства алгоритмов.
5. Каким образом используются алгоритмы при решении задач?
6. Для чего применяют алгоритмы?
7. Укажите способы задания алгоритмов и их соотношения.
8. Каковы средства и основные блоки визуального представления алгоритмов?
9. Сформулируйте общие правила при проектировании визуальных алгоритмов.
10. Как используется системный подход к построению алгоритмов?
11. В чем заключается метод структурной алгоритмизации?
12. Какие существуют базовые алгоритмические структуры? Приведите пример.
13. Чем отличаются алгоритмические структуры композиция и альтернатива?
14. Чем отличаются алгоритмические структуры альтернатива и итерация?
15. Как классифицируются алгоритмы по структурному признаку?
16. Приведите примеры линейных и разветвленных алгоритмов.

2.5. Циклические алгоритмы

Циклические алгоритмы являются наиболее распространенным видом алгоритмов, в них предусматривается повторное выполнение определенного набора действий в зависимости от некоторого условия, называемого **условием окончания повторений**. Такое повторное выполнение называют **циклом**. Повторяющиеся действия в цикле называются «**телом цикла**».

Существуют два основных вида циклических алгоритмов: циклические алгоритмы с предусловием, циклические алгоритмы с постусловием. Они отличаются друг от друга местоположением условия выхода их цикла.

Цикл с предусловием (цикл ПОКА) начинается с проверки условия выхода из цикла. Это логическое выражение, например I <= 6 (эквивалентно математическому выражению I ≤ 6). Пока оно истинно, то выполняются действия цикла, которые должны повторяться. Если при изменении переменной I логическое выражение I <= 6 станет ложным, то есть I станет больше 6, то цикл с предусловием прекратит свои действия.

Цикл с постусловием (цикл ДО ТЕХ ПОР) функционирует иначе. Сначала выполняются один раз те действия, которые подлежат повторению, затем проверяется логическое выражение, определяющее условие выхода из цикла, например, I > 6. Цикл повторяет действия, указанные в теле цикла, до тех пор, пока условие выхода не станет истинным, в противном случае происходит повторение действий, указанных в цикле. Разновидности циклов приведены на рис. 10, а, б.

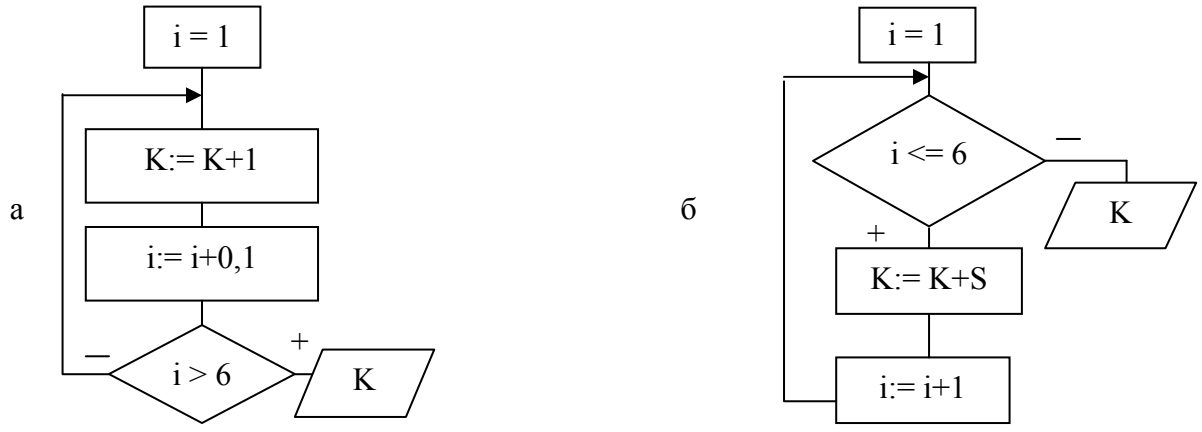


Рис. 10. Виды циклических алгоритмов
а) цикл с постусловием; б) цикл с предусловием

Классическим примером циклического алгоритма служит алгоритм для вычисления степени числа $Y=X^n$. Этот алгоритм может быть реализован на основе операции умножения. Табличное представление такого алгоритма, отражающего зависимость Y от X при изменении

показателя степени n от 1 до 3, представлено в табл. 2. В этой таблице показаны также рекуррентные соотношения между Y и X, определяющие, как на каждом шаге зависит значение Y от значения X и от значения Y, вычисленного на предыдущем шаге.

Таблица 2

Рекуррентные соотношения при вычислении $Y=X^n$

n	Y	Рекуррентные соотношения
1	Y[1]=X	Y=X
2	Y[2]=X*X или Y[2]=Y[1]*X	Y=X*X или Y=Y*X
3	Y[3]=X*X*X или Y[3]=Y[2]*X	Y=X*X*X или Y=Y*X

Пример 5. Требуется составить алгоритм получения на отрезке X Є [-15,15] последовательности значений функции Y = SIN(X) в виде таблицы значений (X,Y) при изменении аргумента X по формуле X[k] = X[k-1] + h, где h = 1,5.

Решение. Такие задачи относят к задачам табулирования функций. Из условия задачи определяем, что начальное значение отрезка табулирования X = –15, конечное значение – X = 15. Процесс получения множества пар (X, Y) является итерационным, значит проектируемый алгоритм будет циклическим. Условие выхода из цикла X > 15.

На рис. 12 представлен циклический алгоритм с предусловием вычисления табличного значения функции Y = SIN(X) на отрезке -15 < X < 15 при изменении X на каждом шаге итерации на величину 1,5. Результатом выполнения алгоритма является циклический вывод множеств пар (Y, X).

Текстовая форма алгоритма

НАЧАЛО

Присвоить X = –15

Пока X <= 15 повторять

1. Вычислить Y = SIN(X)

2. Вывести Y,X

3. Вычислить X = X+1,5

КОНЕЦ

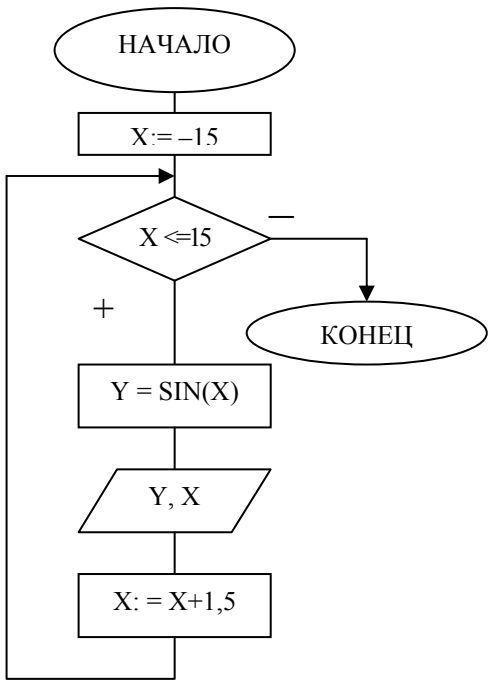


Рис. 12. Циклический алгоритм табулирования функции Y = SIN(X)

Пример 6. Пусть требуется составить алгоритм вычисления суммы ряда $S=x+x^2+x^3+\ldots+x^n$.

Решение. Исходные данные для алгоритма – это переменные x и n. На каждом шаге будем вычислять очередной член суммы Y и прибавлять его к предыдущему значению суммы S. Для этого используем рекуррентную формулу вычисления степени X (см. таблицу 3) Y=Y*X, тогда сумма ряда на каждом шаге итерации будет вычисляться по формуле S=S+Y. Количество итераций K изменяется от 1 до n и равно количеству членов ряда. Начальное значение суммы ряда S равно 0.

На рис. 13 представлен циклический алгоритм с предусловием для вычисления заданной суммы ряда.

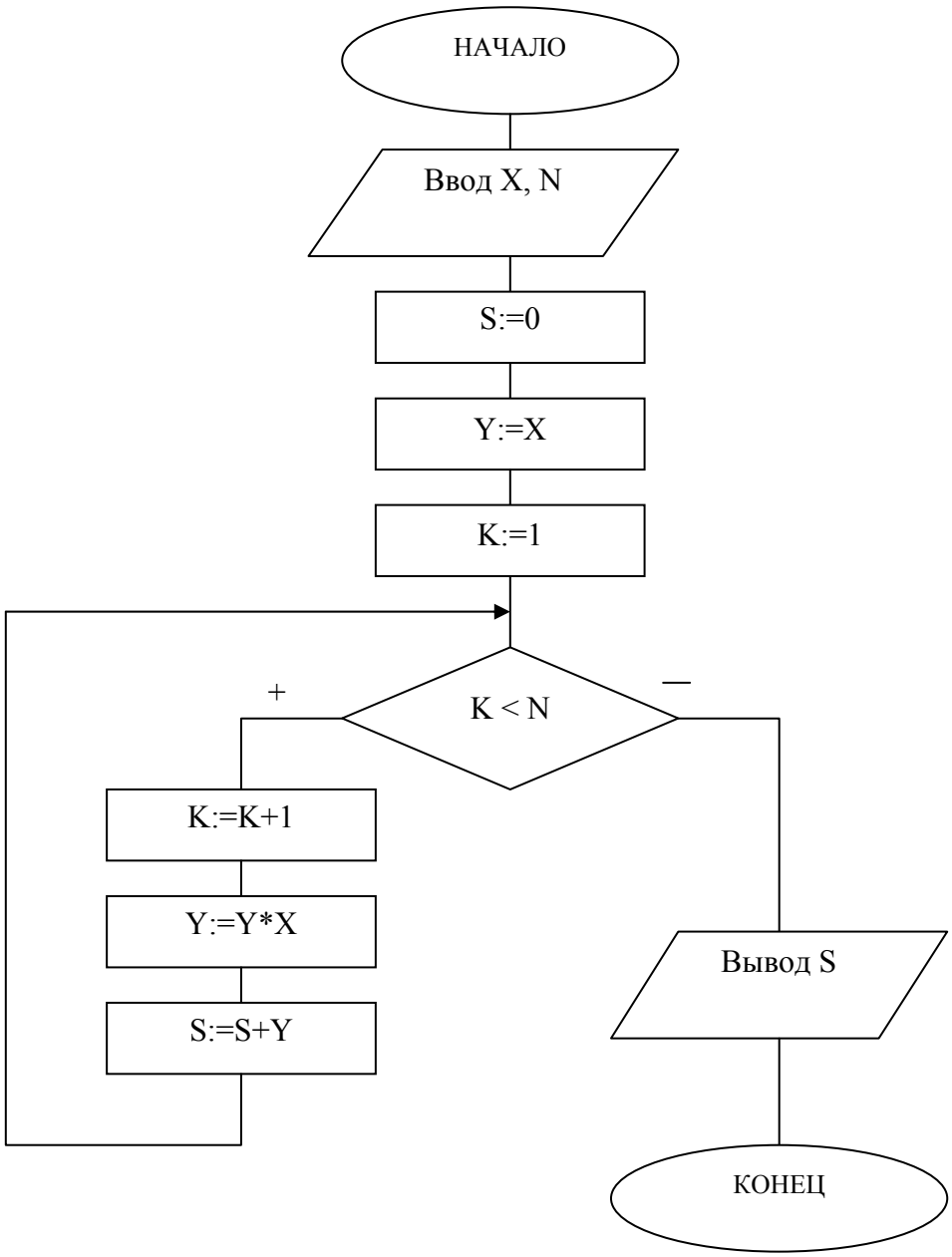


Рис. 13. Алгоритм вычисления суммы ряда $S=x+x^2+x^3+\ldots+x^n$

Задания для самостоятельного выполнения

Цель заданий. Приобрести умения в синтезе формальной и алгоритмической моделей решения задач. Сформировать компетенции анализа и синтеза при решении простых задач, требующих циклической обработки.

Порядок выполнения. Составить формальное и алгоритмическое решения следующих задач:

1. Вычислить число в факториале Y = X! (факториал X).
2. Вычислить сумму ряда, общий член которого задан формулой $A_n = (x*n)/n!$
3. При табулировании функции y = cos(x + a) на отрезке [1, 10] с шагом h=1 определить сумму значений y, больших p.
4. Подсчитать количество цифр в целом числе X.
5. Вычислить сумму значений функции $y = x^2$ на отрезке [1, 5] с шагом 1.

2.6. Алгоритмы обработки последовательности чисел

Последовательность значений – это набор однотипных величин, которые вводятся и обрабатываются циклически. Примером последовательности целых чисел может быть следующий набор значений: (2, 5, – 4, 10, 1, 0). Последовательности значений отличаются от массивов значений тем, что в памяти одновременно все значения последовательности не хранятся. Для обозначения значения последовательности используют одну переменную, в которую на каждом шаге итерации вводится очередное значение последовательности. Отличительной особенностью последовательности является также возможность содержания неопределенного или неизвестного заранее количества ее значений. В этом случае критерием окончания последовательности служит некоторое особое значение, например, ноль.

Пример 7. В числовой последовательности определить сумму положительных и произведение отрицательных чисел. Реализовать с помощью цикла с предусловием. Признак конца последовательности – значение 0.

Решение. Обозначим за X переменную, содержащую очередное значение последовательности, за S – сумму положительных значений, за P – произведение отрицательных значений. Полученный алгоритм приведен на рис. 14. Условие для выбора вычислений X>0. Для вычисления суммы значений воспользуемся рекуррентной формулой S=S+X с начальным значением S=0, для вычисления произведения – рекуррентной формулой P=P*X с начальным значением P=1. Условие выхода из цикла – неравенство X<>0.

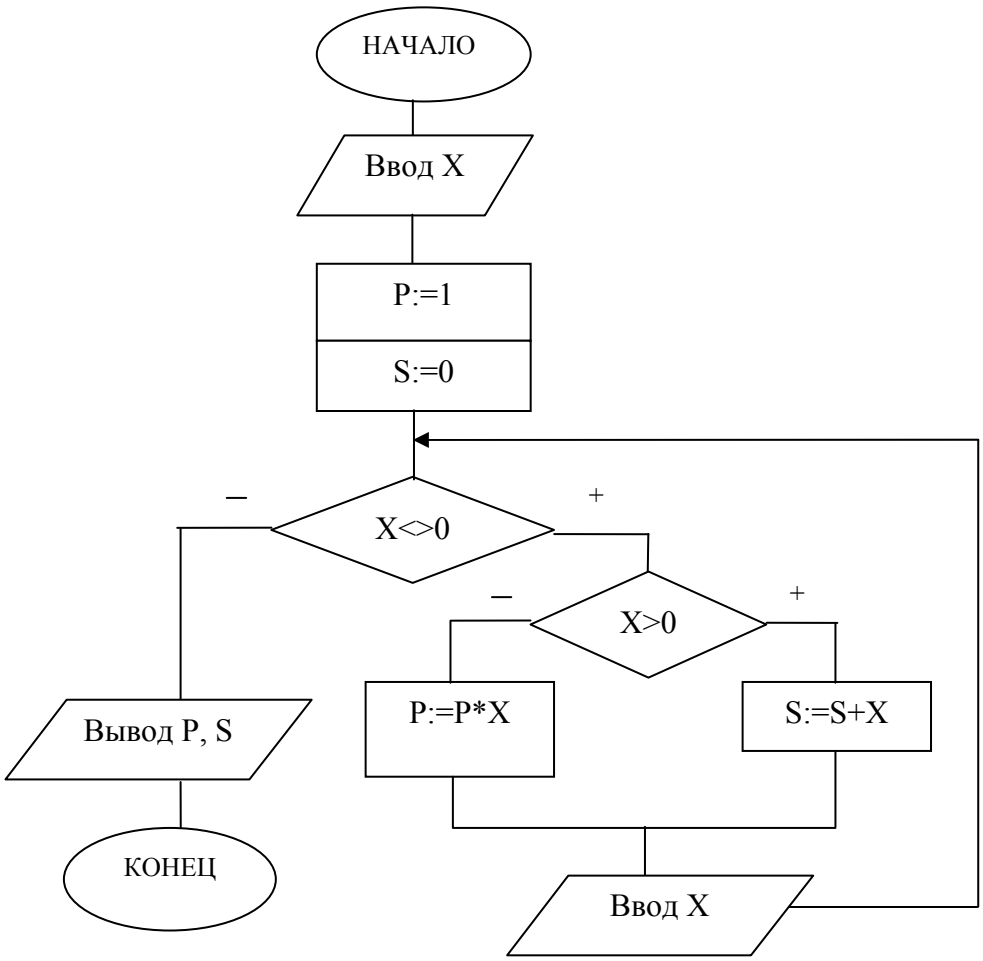


Рис. 14. Алгоритм вычисления суммы положительных и произведения отрицательных значений числовой последовательности

Пример 8. Составить циклический алгоритм для определения в последовательности целых чисел количества четных чисел.

Решение. Обозначим за X переменную, содержащую очередное значение последовательности, за K – количество четных значений (рис. 15). Условие для выбора четных значений X mod 2=0 (остаток при делении X на 2 равен 0). Для вычисления количества значений воспользуемся рекуррентной формулой K=K+1 с начальным значением K=0.

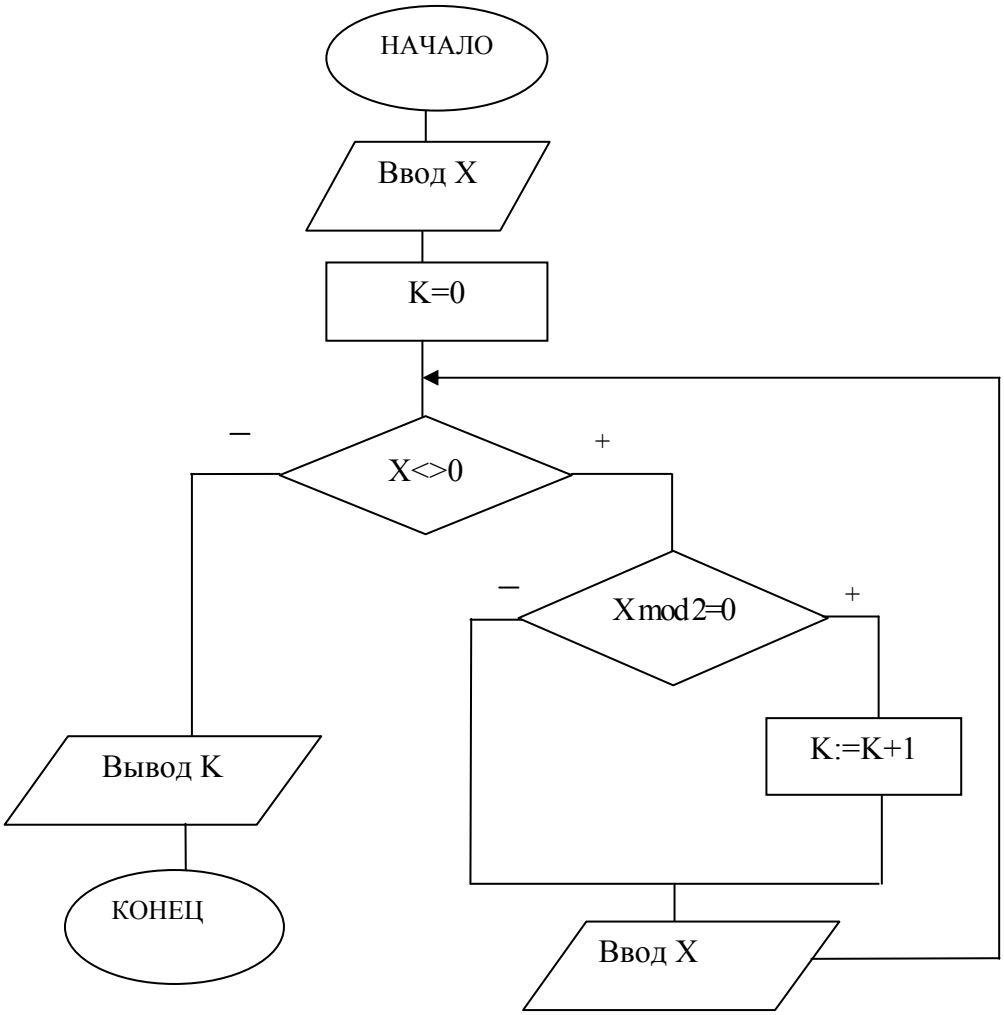


Рис. 15. Алгоритм определения количества четных чисел
в последовательности значений

Задания для самостоятельного выполнения

Цель заданий. Приобрести умения в синтезе формальной и алгоритмической моделей решения задач. Сформировать компетенции анализа и синтеза при решении простых задач циклической обработки последовательности значений.

Порядок выполнения. Составить формальное и алгоритмическое решения следующих задач обработки последовательности значений:

1. В процессе обучения студент группы сдавал экзамены. Определить средний балл за сданные им экзамены.
2. Некоторая группа студентов сдавала экзамен по дисциплине «Редакторское дело». Определить средний балл группы по этой дисциплине.
3. В группе учатся как девушки, так и юноши. Определить, кто лучше сдал экзамен по дисциплине «Информатика» – юноши или девушки?
4. Определить среднюю и общую стоимость книг в книжном магазине.

5. Определить объем выпуска книжной продукции издательства «Эдельвейс» в печатных листах, учитывая коэффициент перевода авторских страниц в печатные листы.
6. Определить общую стоимость выпуска книжной продукции издательства «Прогресс» с учетом цены и тиража.
7. В последовательности чисел определить предпоследнее отрицательное число. (При решении введите дополнительную переменную для хранения предпоследнего отрицательного числа).
8. В последовательности целых положительных чисел определить максимальное число. Рекомендуем реализовать такой алгоритм:

Вводим X
max=X
Цикл с постусловием
а. Если элемент X > max
то max:=X (значение этого элемента);
б. Вводим новый элемент последовательности X.
Условие выхода из цикла X=0

9. В последовательности целых чисел определить третье положительное число и подсчитать количество цифр в нем.

Контрольные вопросы

1. Какие алгоритмы относятся к циклическим?
2. Что называют телом цикла?
3. Какие существуют виды циклов?
4. Чем отличается цикл с предусловием от цикла с постусловием?

2.7. Алгоритмы обработки одномерных числовых массивов

Под структурой данных типа **массив** понимают однородную структуру однотипных данных, одновременно хранящихся в последовательных ячейках оперативной памяти. Эта структура должна иметь имя и определять заданное количество данных (элементов). Однотипность данных определяет возможность использования циклических алгоритмов для обработки всех элементов массива. Количество итераций цикла определяется количеством элементов массива. Одновременное хранение в памяти всех элементов массива позволяет многократно обращаться к элементу массива и решать большой набор задач, таких как поиск элементов, упорядочение и изменение порядка следования элементов.

Доступ к любому элементу массива осуществляется по его номеру (индексу). Поэтому для обращения к элементу массива используют имя массива (номер элемента), например, A(5).

Массив называется одномерным, если для получения доступа к его элементам достаточно одной индексной переменной.

Рассмотрим простой алгоритм ввода элементов одномерного числового массива А из 9 элементов. В этом циклическом алгоритме условие выхода из цикла определяется значением специальной переменной К, которая называется счетчиком элементов массива А (рис. 16).

Эта же переменная К определяет количество итераций циклического алгоритма ввода элементов массива. На каждом шаге итерации переменная К (значение номера элемента массива А) изменяется на 1, то есть происходит переход к новому элементу массива. В дальнейшем, при рассмотрении алгоритмов обработки одномерных массивов в целях устранения дублирования алгоритм ввода элементов массива будем заменять одним блоком, подразумевая, что он реализуется по схеме циклического алгоритма, представленного на рис. 16.

Пример 9. Составить алгоритм определения в одномерном числовом массиве А из N элементов суммы положительных элементов.

Решение. Алгоритм представлен на рисунке 17. В этом алгоритме переменная К является счетчиком элементов массива, S – сумма элементов массива, она вычисляется по рекуррентной формуле S=S+A(K). Ввод количества и значений элементов массива осуществляется сначала в отдельном блоке ввода, который реализуется по схеме алгоритма ввода элементов массива, изображенного на рис. 16.

Часто для проверки правильности работы алгоритмов на конкретных наборах данных используют таблицу трассировки. Эта таблица содержит столько столбцов, сколько переменных и условий в алгоритме, в ней выполняются действия шаг за шагом от начала до конца алгоритма для конкретных наборов входных данных.

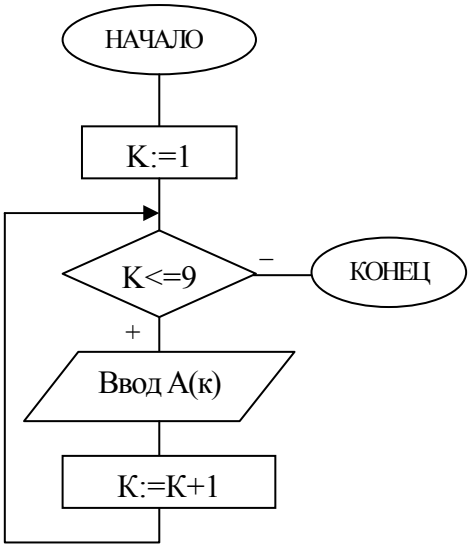


Рис. 16. Алгоритм ввода элементов

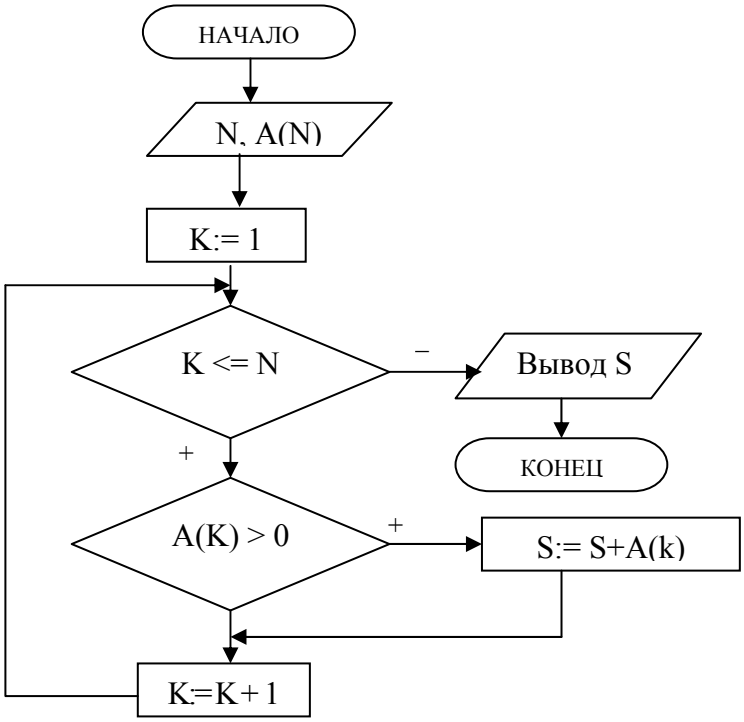


Рис. 17. Алгоритм вычисления суммы положительных элементов массива

Пример 10. Составить алгоритм поиска элемента с максимальным значением в одномерном массиве A(1..n) и его таблицу трассировки для значений (3, 7, 0, 9).

Решение. Введем обозначения: K – текущий номер элемента, A[K] – текущее значение элемента массива, N=4 – количество элементов одномерного массива, M – номер максимального элемента массива, A[M] – значение максимального элемента массива. Основной идеей алгоритма является выполнение сравнения текущего элемента массива A[K] и элемента с максимальным значением A[M], определенным на предыдущем шаге итерации. По алгоритму, изображенному на рис. 18, получено максимальное значение для массива (3, 7, 0, 9), процесс и правильный результат поиска которого показаны в табл. 3.

Таблица 3

Таблица трассировки алгоритма примера 10

Номер элемента массива K	Значение элемента A (K)	Номер максимального M		Значение максимального A(M)		Проверка A(K)>A(M)
1	3	1		3		нет
2	7	1	2	3	7	да
3	0	2		7		нет
4	9	2	4	7	9	да

2.8. Алгоритмы сортировки одномерных массивов

Под сортировкой понимают процесс перестановки объектов данного массива в определенном порядке. Целью сортировки является упорядочение массивов для облегчения последующего поиска элементов в данном массиве. Рассмотрим основные алгоритмы сортировки по возрастанию числовых значений элементов массивов. Существует много методов сортировки массивов. В этой работе будут рассмотрены алгоритмы двух методов: модифицированного метода простого выбора и метода парных перестановок.

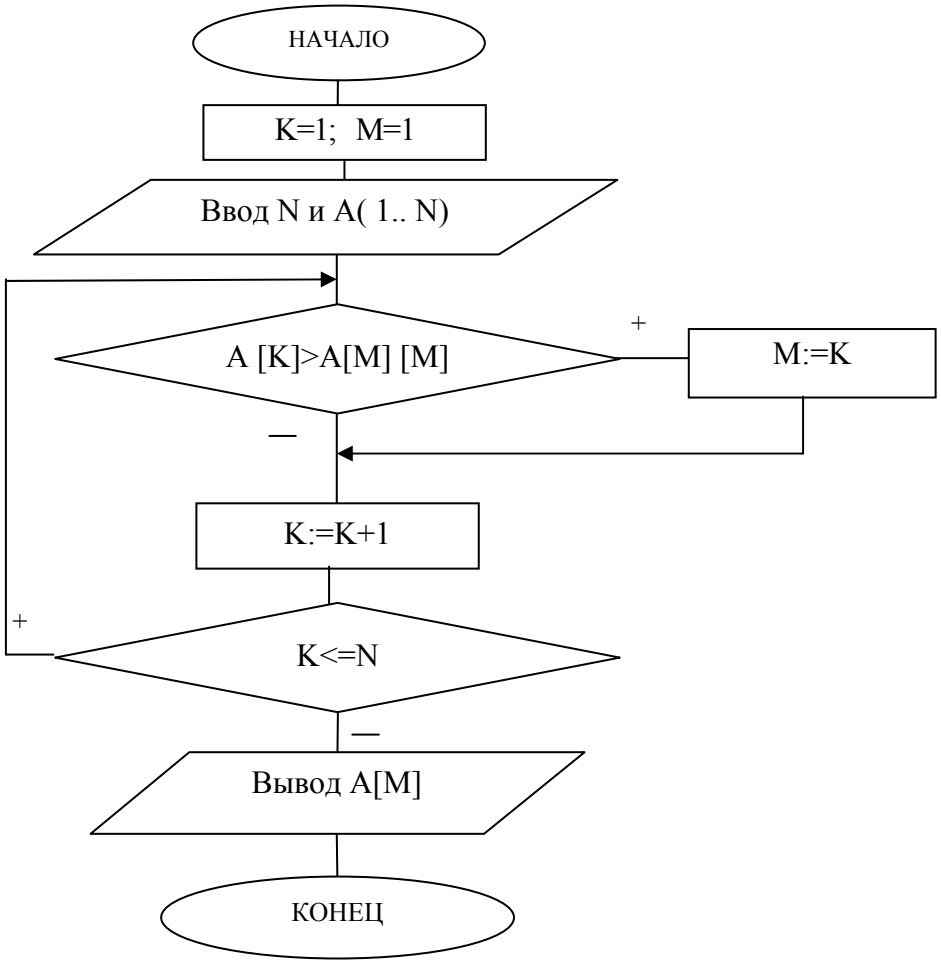


Рис. 18. Алгоритм поиска максимального значения в массиве

Сортировка модифицированным методом простого выбора

Этот метод основывается на алгоритме поиска минимального элемента. В массиве A(1..n) отыскивается минимальный элемент, который ставится на первое место. Для того, чтобы не потерять элемент, стоящий на первом месте, этот элемент устанавливается на место минимального. Затем в усеченной последовательности, исключая первый элемент, отыскивается минимальный элемент и ставится на второе место и так далее n–1 раз, пока не встанет на свое место предпоследний n–1 элемент массива A, сдвинув максимальный элемент в самый конец.

Рассмотрим алгоритмическое решение задачи на примере сортировки некоторого массива значений по возрастанию. В соответствии с вышеописанным методом необходимо несколько раз выполнить операции поиска минимального элемента и его перестановку с другим элементом, то есть потребуется несколько раз просматривать элементы массива с этой целью. Количество просмотров элементов массива, согласно описанию модифицированного метода простого выбора, равно

n–1, где n – количество элементов массива. Таким образом, можно сделать вывод, что проектируемый алгоритм сортировки будет содержать цикл, в котором будет выполняться поиск минимального элемента и его перестановка с другим элементом.

Обозначим через i счетчик (номер) просмотров элементов массива и изобразим обобщенный алгоритм сортировки на рис. 19.

Отметим, что для перестановки элементов местами необходимо знать их порядковые номера. Алгоритмы ввода исходного массива изображен на рис. 16. Алгоритм поиска в массиве минимального элемента и его номера будет аналогичен алгоритму примера 10, который представлен на рис. 18.

Однако, в этом алгоритме будут внесены изменения. Для того, чтобы определить какие изменения следует внести, рассмотрим выполнение сортировки данным методом с акцентом на поиск минимального элемента на конкретном примере. Пусть исходный массив содержит 5 элементов (2, 8, 1, 3, 7). Количество просмотров, согласно модифицированному методу простого выбора, будет равно 4. Покажем в таблице 4, как будет изменяться исходный массив на каждом просмотре.

Таблица 4

Пример сортировки

Номер просмотра массива i	Исходный массив	Минимальный элемент		Переставляемый элемент		Массив после перестановки
		Номер	Значение	Номер	Значение	
1	(2,8,1,3,7)	3	1	1	2	(1,8,2,3,7)
2	1,(8,2,3,7)	3	2	2	8	1,(2,8,3,7)
3	1,2,(8,3,7)	4	3	3	8	1,2,(3,8,7)
4	1,2,3,(8,7)	5	7	4	8	1,2,3,7,8

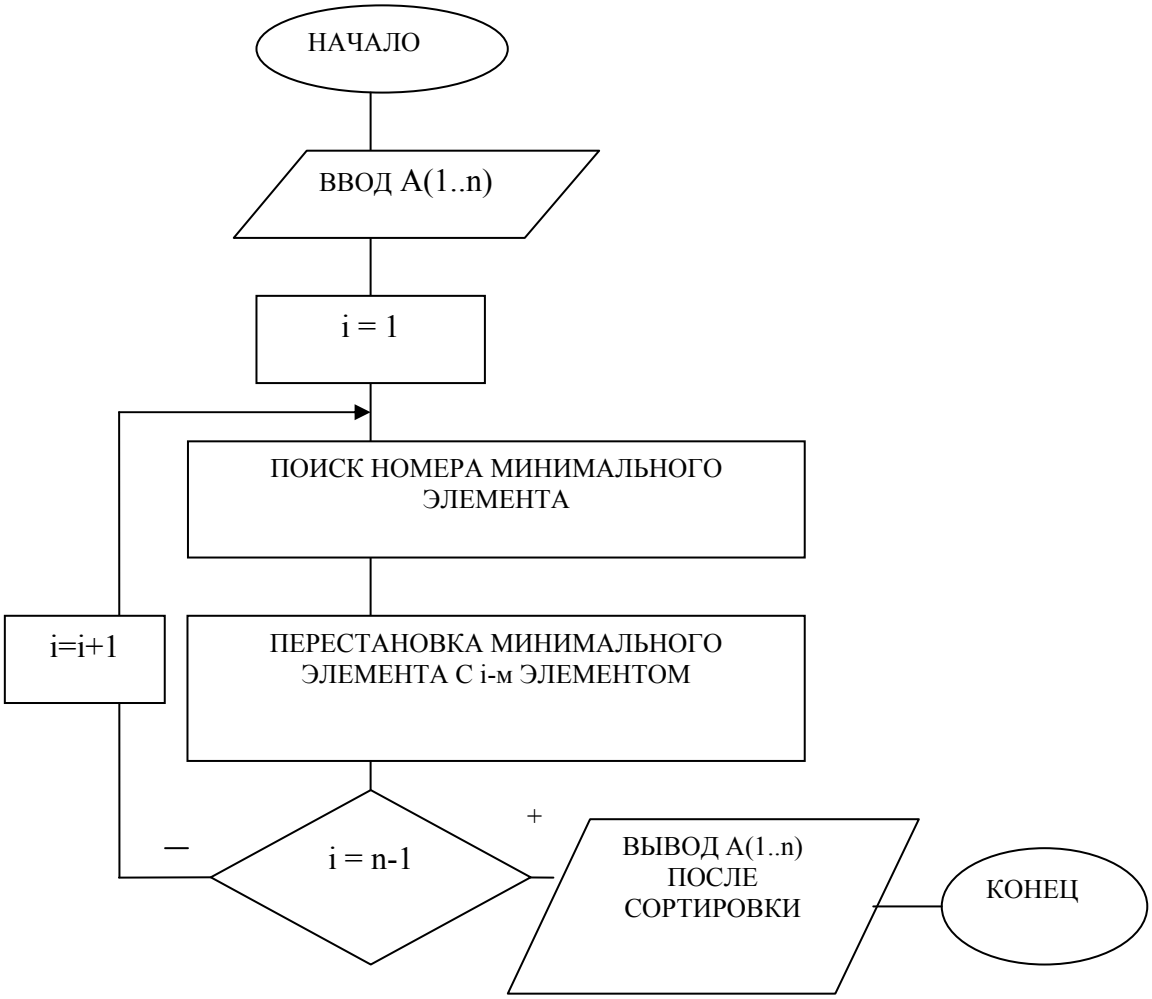


Рис. 19. Обобщенный алгоритм сортировки массива модифицированным методом простого выбора

Из данных, приведенных в таблице 4, следует, что поиск минимального значения в массиве на каждом просмотре осуществляется в сокращенном массиве, который сначала начинается с первого элемента, а на последнем просмотре массив, в котором ищется минимальный элемент, начинается уже с четвертого (или n–1) элемента. При этом можно заметить, что номер первого элемента массива для каждого поиска и перестановки совпадает с номером просмотра i.

Введем следующие обозначения:

K – номер минимального элемента,

J – номер элемента массива,

M и A(K) – одно и тоже значение минимального элемента массива,

i – номер переставляемого с минимальным элемента,

A(i) – значение переставляемого элемента.

Тогда циклический алгоритм сортировки модифицированным методом простого выбора будет выглядеть, как показано на рис. 20.

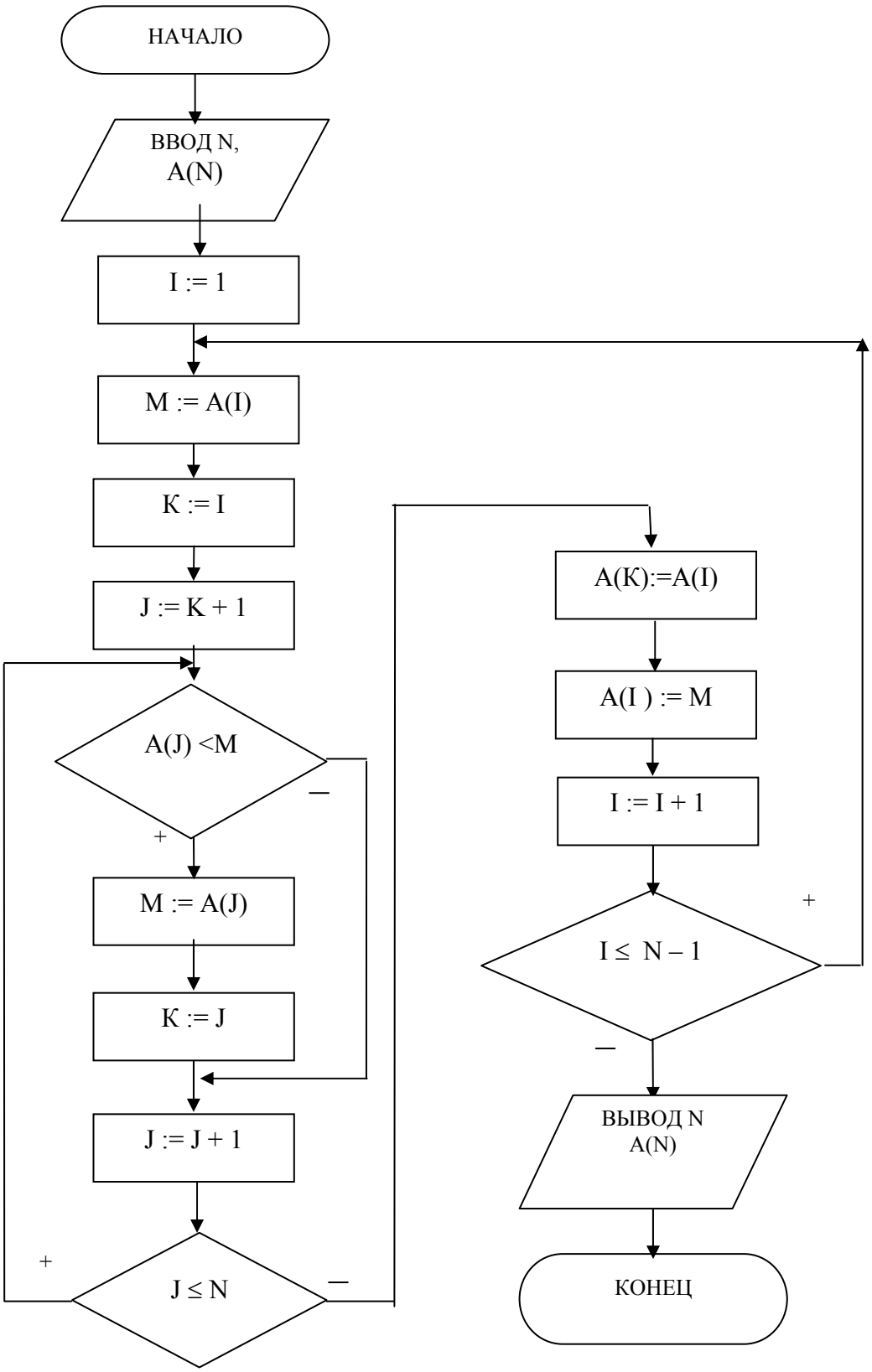


Рис. 20. Алгоритм сотрировки массива модифицированным методом простого выбора

Сортировка методом парных перестановок

Самый простой вариант этого метода сортировки массива основан на принципе сравнения и обмена пары соседних элементов.

Процесс перестановок пар повторяется просмотром массива с начала до тех пор, пока не будут отсортированы все элементы, т. е. во время очередного просмотра не произойдет ни одной перестановки. Для подсчета количества перестановок целесообразно использовать счетчик – специальную переменную B.

Если при просмотре элементов массива значение счетчика перестановок осталось равным нулю, то это означает, что все элементы отсортированы (см. рис. 21).

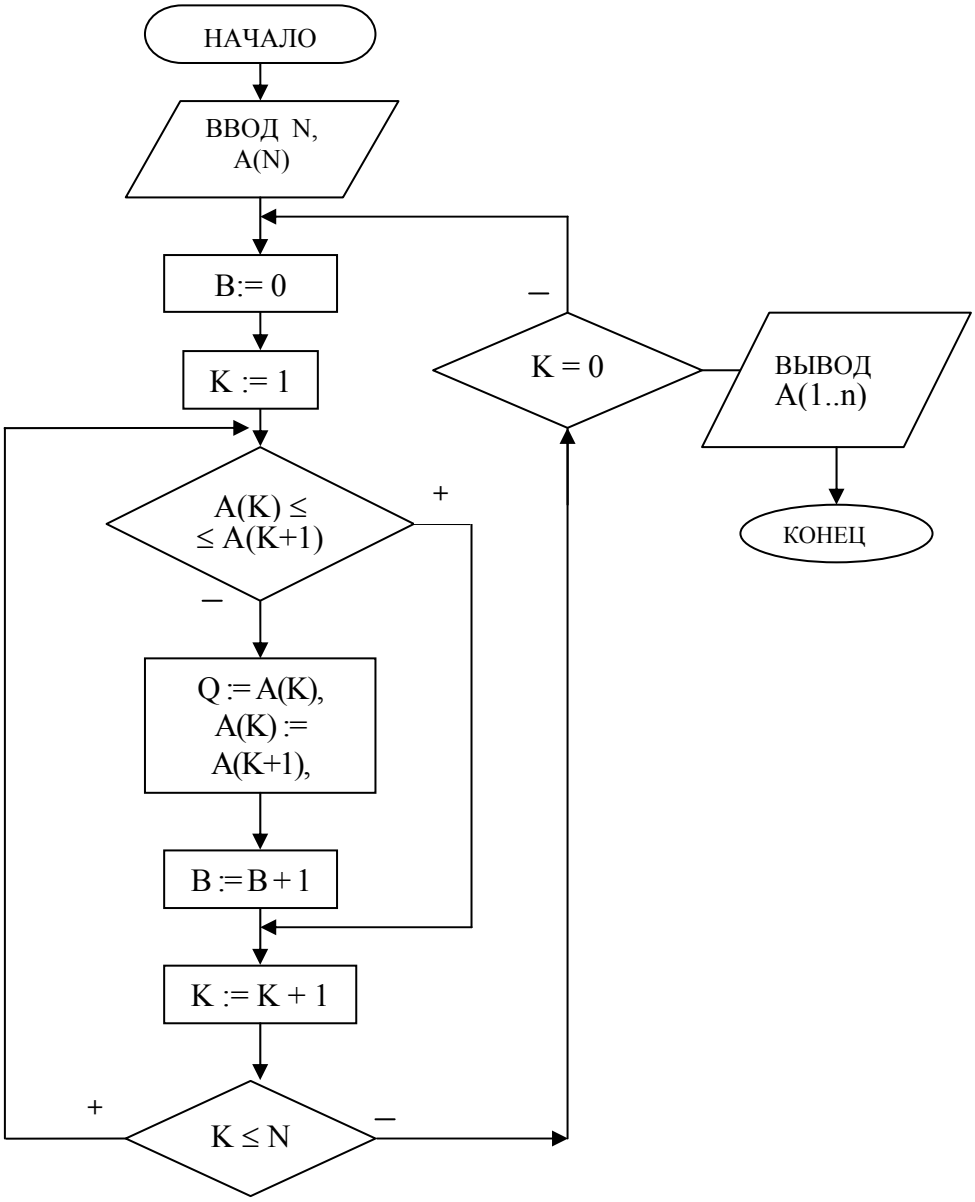


Рис. 21. Алгоритм сортировки методом парных перестановок

2.9. Алгоритмы обработки упорядоченных массивов

Рассмотренные выше алгоритмы сортировки считаются одними из важнейших процедур упорядочивания структурированных данных, хранимых в виде массивов. Одной из главных целей задач сортировки массивов является облегчение их дальнейшей обработки, так как для упорядоченных данных разработаны эффективные методы поиска и обновления. Так, например, поиск минимального или максимального значения в упорядоченном массиве сводится к выборке первого или последнего элемента массива. Рассмотрим алгоритм поиска произвольного элемента в упорядоченном массиве.

Задача поиска значения X в упорядоченном по возрастанию значений массиве A(1)<A(2)<,...,<A(n) формулируется следующим образом. Требуется выяснить, входит ли значение X в этот упорядоченный массив, и если входит, то определить значение k, для которого A(k)=X. Для такого типа задач применяется эффективный метод бинарного поиска, который также известен как метод деления пополам.

Сущность этого метода поиска заключается в последовательном определении номера S элемента, расположенного в точке деления упорядоченного массива пополам и сравнении искомого значения X с этим элементом массива A(s). Если A(s)=X, то поиск заканчивается. В противном случае возможны две ситуации.

Если A(s)<X, то все элементы, имеющие номера с 1 по s, также будут меньше X, если A(s)>X, то все элементы, имеющие номера с S по n, также будут больше X в силу упорядоченности массива по возрастанию значений.

Поэтому для дальнейшего поиска половину значений массива можно исключить из рассмотрения. В первом случае – левую, во втором случае – правую половину. Рассмотрим пример. Допустим, что требуется определить, совпадает ли значение X=12 с каким-либо элементом в упорядоченном массиве A и, если совпадает, то определить порядковый номер S этого элемента. Иллюстрация применения метода бинарного поиска для поиска элемента X=12 в массиве (2, 3, 4, 6, 10, 12, 20, 30, 35, 45) приведена на рис. 23.

На рис. 22 представлен алгоритм, реализующий метод бинарного поиска в упорядоченном массиве.

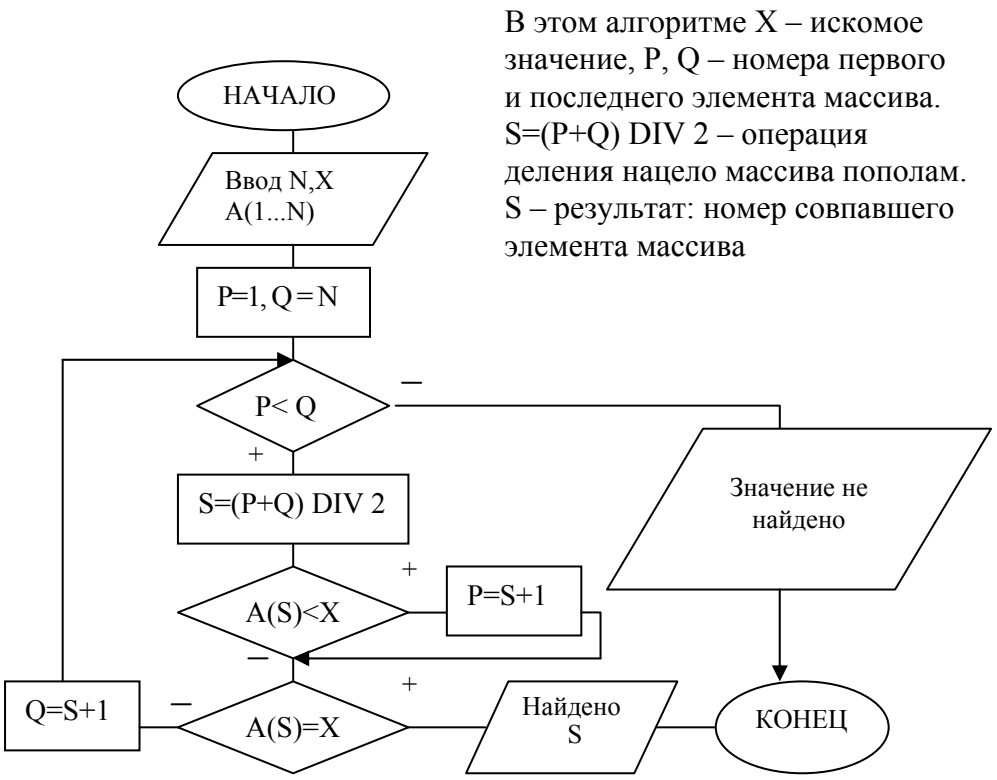


Рис. 22. Алгоритм метода бинарного поиска

Искомый элемент X=12.
Элементы массива для рассмотрения A (2, 3, 4, 6, 10, 12, 20, 30, 35, 45).
Номера элементов 1 2 3 4 5 6 7 8 9 10.

Первое бинарное деление массива : номер элемента S=5, A(5)=10 A(5)<X, исключаем левую половину массива из рассмотрения.

Элементы массива для рассмотрения A (12, 20, 30, 35, 45).
Номера элементов 6 7 8 9 10.

Второе бинарное деление массива: номер элемента S=8, A(8)=30 A(8)>X, исключаем правую половину массива из рассмотрения.

Элементы массива для рассмотрения A (12, 20).
Номера элементов 6 7
Третье деление S=6, A(6)=12 A(6)=X, элемент X=12 найден.

Рис. 23. Иллюстрация применения метода бинарного поиска

Задания для самостоятельного выполнения

Цель заданий. Приобрести умения в синтезе формальной и алгоритмической моделей решения задач. Сформировать компетенции анализа и синтеза при решении простых задач циклической обработки значений массива.

Порядок выполнения. Составить формальное и алгоритмическое решения следующих задач обработки значений массивов:

1. В одномерном массиве определить первый отрицательный элемент и его номер.
2. Исключить из массива a1,…, an первый отрицательный элемент.
3. Ввести массив a1, a2,..., a15. Расположить ненулевые элементы по убыванию.
4. Ввести массив x1,x2,...,x20. Элементы на нечетных местах, расположить в порядке возрастания, а на нечетных – в порядке убывания.
5. Ввести массив a1,a2,...,a15. Требуется упорядочить его по возрастанию абсолютных значений элементов.
6. Ввести массив x1,x2,...,x20. Требуется расположить отрицательные элементы в порядке убывания.
7. Ввести упорядоченный по неубыванию массив $X(1) \le X(2) \le ... \le X(n)$. Найти количество различных чисел среди элементов этого массива.
8. Ввести два упорядоченных по возрастанию числовых массива $X(1) < X(2) < ... < X(n)$ и $Y(1) < Y(2) < ... < Y(m)$. Найти количество общих элементов в этих массивах, то есть количество К таких чисел X(i)=Y(j).
9. Известно, что некоторое число содержится в каждом из трех целочисленных неубывающих массивов $X(1) \le X(2) \le ... \le X(n)$, $\mathrm{Y}(1) \le \mathrm{Y}(2) \le ... \le \mathrm{Y}(\mathrm{m})$ и $Z(1) \le Z(2) \le ... \le Z(k)$. Найти одно из этих чисел.

2.10. Алгоритмы обработки двумерных массивов

Двумерный массив – это структура однотипных элементов, расположенных в виде таблицы значений. Такое представление значений соответствует математическому понятию двумерного массива. Каждый элемент в двумерном массиве идентифицируется номером строки и номером столбца, на пересечении которых он расположен. Например, в двумерном массиве A, изображенном на рис. 24, элемент со значением 5 расположен на пересечении третьей строки и второго столбца. Этот элемент будет обозначаться как A(3, 2). А элемент

$$\begin{pmatrix} 3 & 7 & -1 & 0 \\ 4 & 2 & 6 & 1 \\ 9 & 5 & 12 & 4 \\ 6 & 22 & 31 & 1 \end{pmatrix}$$

Рис. 24. Пример двумерного массива

A(1, 4) имеет значение, равное нулю. Такое представление набора значений позволяет выполнять обработку как отдельных значений в двумерном массиве, так и последовательности значений, расположенных в строках или столбцах.

В дальнейшем будем считать, что для двумерного массива A(N, M) в обозначении элемента A(i, j) первое значение i соответствует номеру строки и изменяется от 1 до N, а j – номеру столбца и изменяется от 1 до M. В отличие от одномерного массива, в котором использовался только один номер для определения местоположения элемента и требовался только один цикл для ввода элементов, в двумерном массиве для обработки элементов необходимы два вложенных друг в друга цикла. Внешний цикл предназначен для изменения номера строки i, а второй, внутренний, – для изменения номера столбца j в текущей строке i. На рис. 25 представлен простой алгоритм ввода элементов, построенный в виде структуры из вложенных циклов. В дальнейшем при рассмотрении алгоритмов обработки элементов двумерного массива в целях сокращения их размера фрагмент ввода элементов будем заменять отдельным блоком ввода.

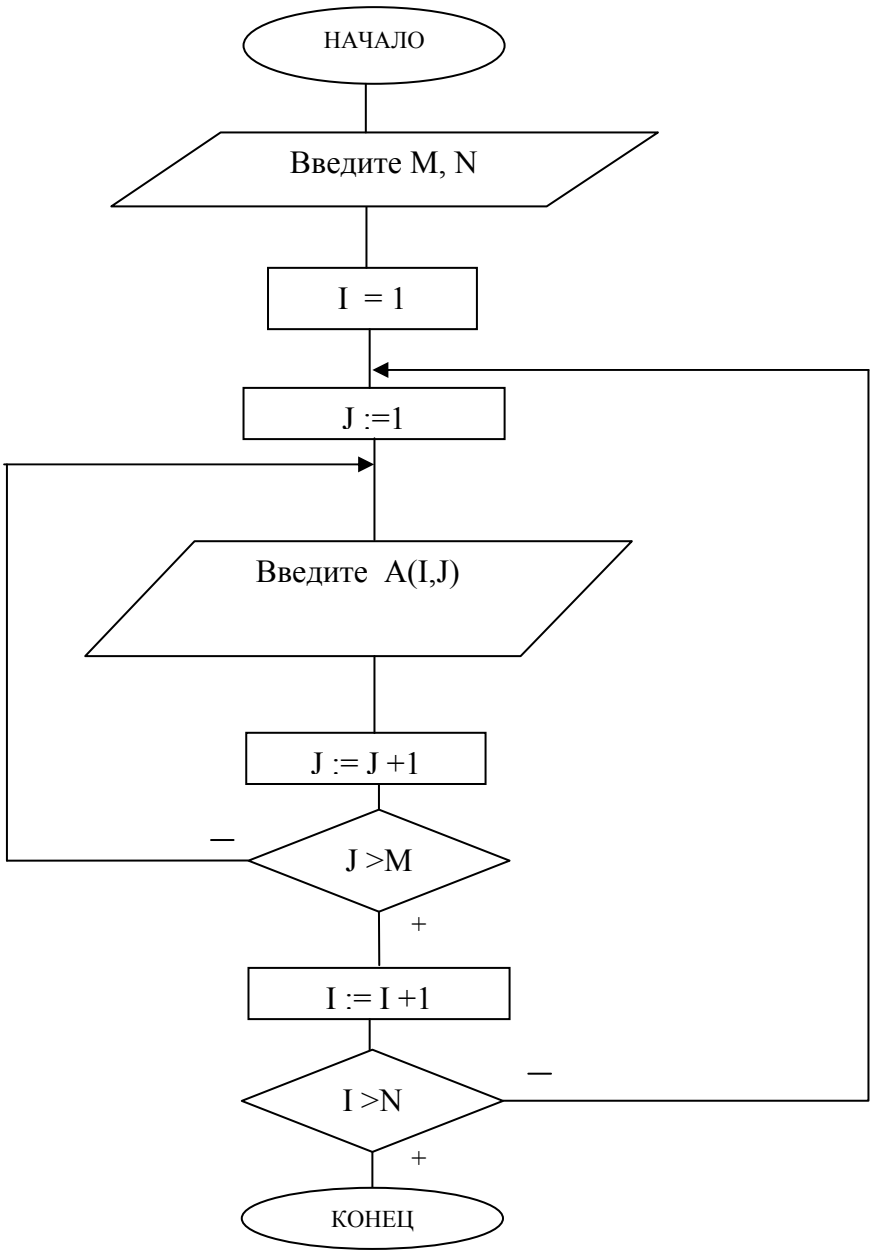


Рис. 25. Алгоритм ввода элементов двумерного массива

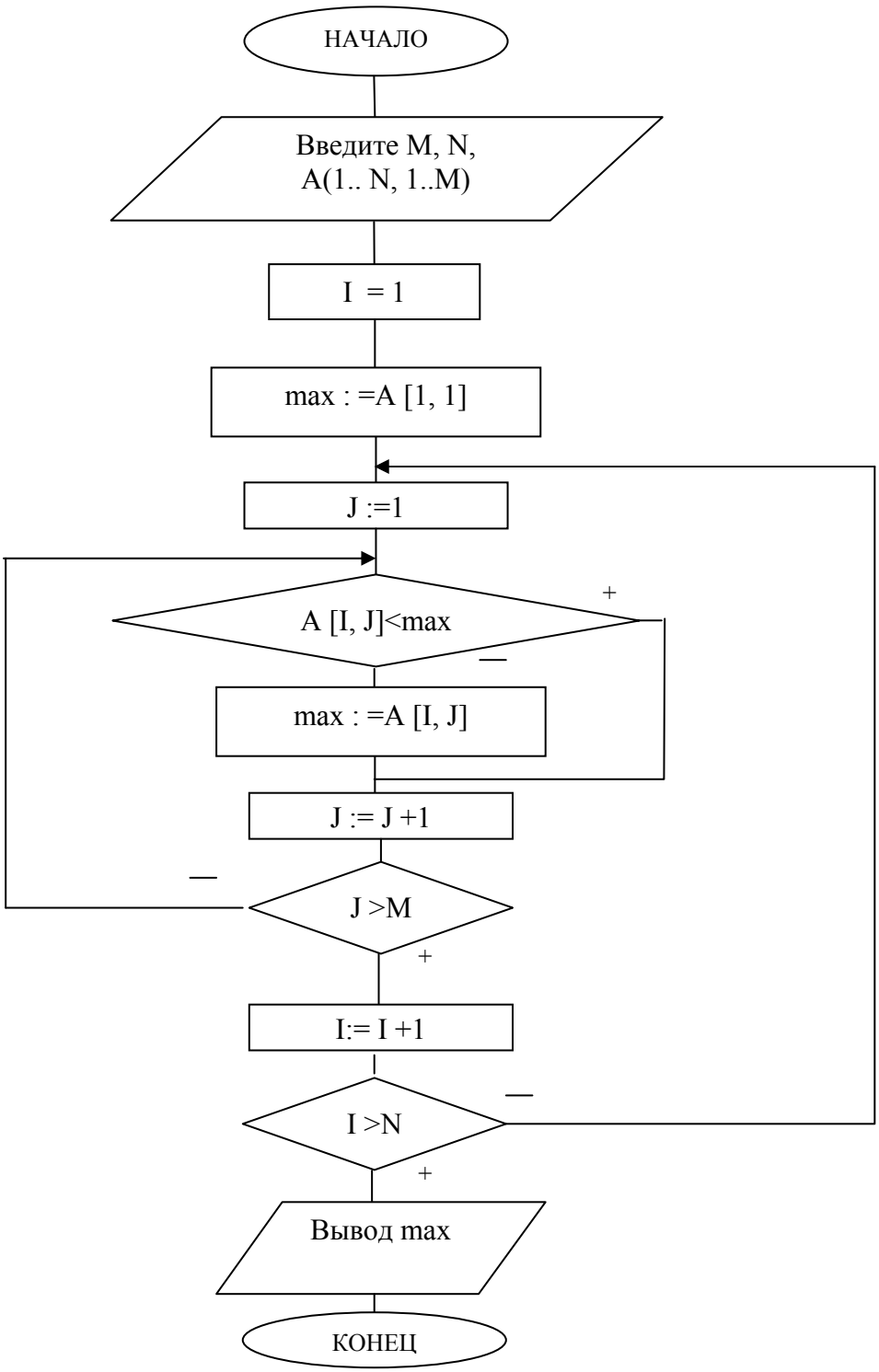


Рис. 26. Алгоритм поиска максимального значения в двумерном массиве

Пример 13. Составить алгоритм поиска максимального значения в двумерном массиве.

Решение. Поиск максимального элемента в двумерном массиве осуществляется аналогично поиску в одномерном массиве. Отличие состоит в том, что для обработки двумерного массива используем два вложенных цикла. Обозначим максимальный элемент переменной MAX. Значение этой переменной будет меняться на каждой итерации цикла, если очередное значение элемента массива окажется больше MAX (рис. 26).

Пример 14. Составить алгоритм вычисления количества нечетных элементов в каждой строке двумерного массива A(1.. N, 1..M).

Решение. Для определения нечетных элементов будем использовать проверку на нечетность A[I, J] mod 2≠0, для подсчета количества нечетных значений – формулу К=К+1 и вывод значения К столько раз, сколько строк в массиве. Алгоритм решения представлен на рис. 27.

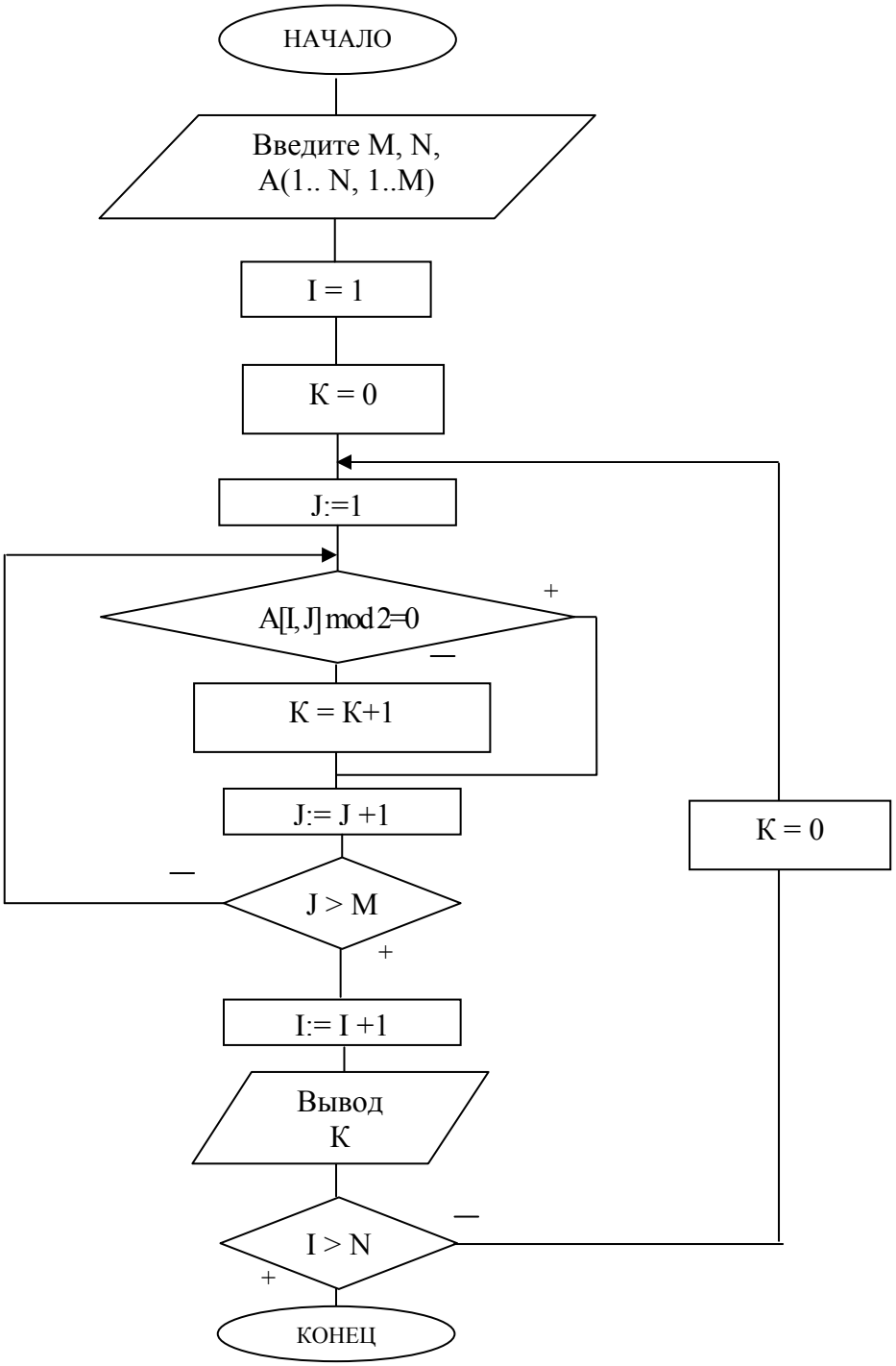


Рис. 27. Алгоритм вычисления в каждой строке двумерного массива количества нечетных элементов

Пример 15. Составить алгоритм вычисления суммы элементов двумерного массива A(1.. N, 1..M), расположенных выше главной диагонали.

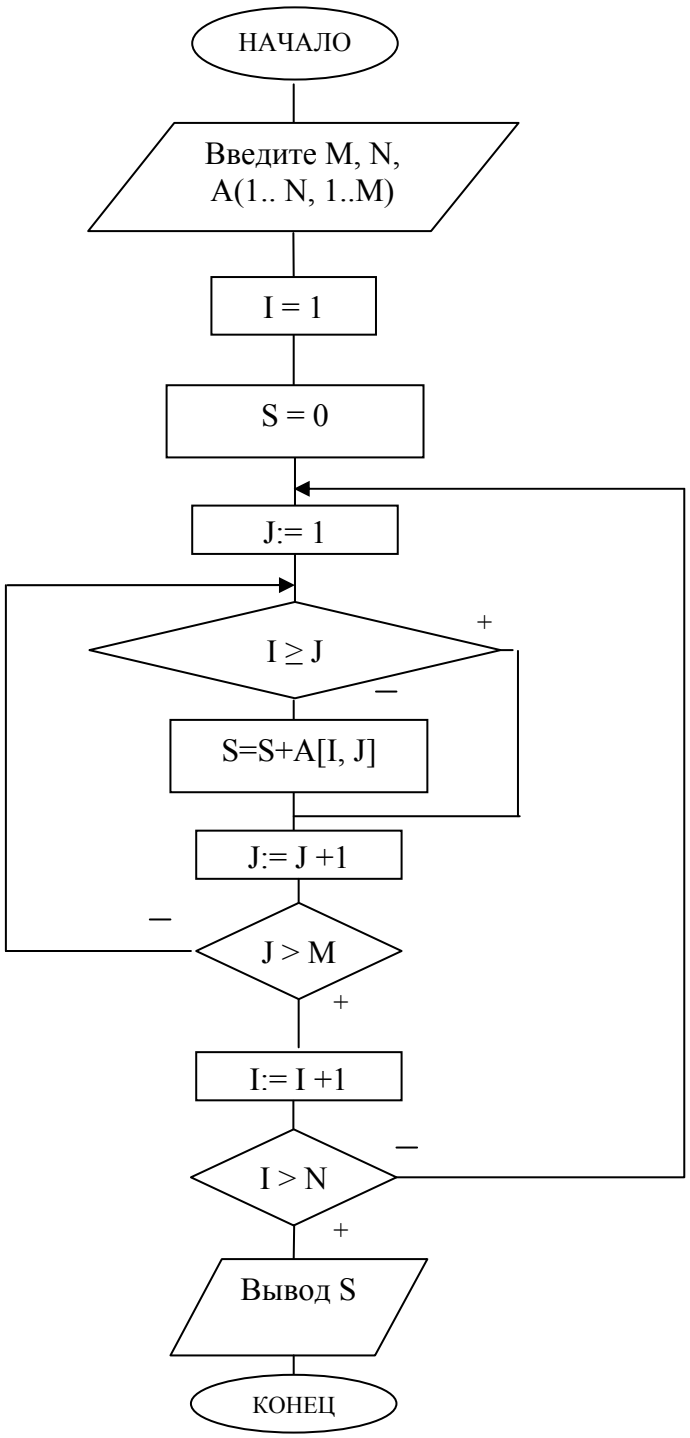


Рис. 28. Алгоритм вычисления суммы элементов двумерного массива, расположенных выше главной диагонали

Решение. Рассмотрим алгоритмическое решение, представленное на рис. 28. Для определения условия расположения элементов выше главной диагонали рассмотрим двумерный массив в обобщенном виде на рис. 29. Обратим внимание на диагональные элементы: номер строки и номер столбца совпадают. Значит для определения элементов на главной диагонали достаточно использовать условие I = J, где I – номер строки, J – номер столбца.

Для определения элементов выше главной диагонали достаточно использовать условие I<J, ниже главной диагонали – I>J. По условию задачи требуется найти сумму

элементов двумерного массива A(1.. N, 1..M), расположенных выше главной диагонали, значит, применим условие I<J, связывающее такие параметры элемента массива как номер строки I и номер столбца J.

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

Рис. 29. Пример двумерного массива

Теперь алгоритмическое решение задачи вычисления суммы элементов двумерного массива, расположенных выше главной диагонали, приведенное на рисунке 28, становится более понятным. Данный алгоритм содержит два вложенных цикла, каждый из которых относится к циклу с постусловием.

Задания для самостоятельного выполнения

Цель заданий. Приобрести умения в синтезе формальной и алгоритмической моделей решения задач. Сформировать компетенции анализа и синтеза при решении простых задач циклической обработки значений двумерного массива.

Порядок выполнения. Составить формальное и алгоритмическое решения следующих задач обработки значений двумерных массивов:

1. Ввести двумерный массив A(N, M). Составить визуальный алгоритм замены всех нулевых элементов на минимальный элемент.
2. Ввести двумерный массив A(N, N). Составить визуальный алгоритм подсчета среднего арифметического значений двумерного массива. Найти отклонение от среднего у элементов первой строки.
3. Ввести двумерный массив A(N, N). Составить визуальный алгоритм подсчета среднего арифметического значения двумерного массива. Вычислить отклонение от среднего для всех элементов двумерного массива.
4. Ввести двумерный массив A(N, N). Составить визуальный алгоритм замены всех отрицательных элементов на среднее арифметическое значение элементов двумерного массива.
5. Составить визуальный алгоритм нахождения числа строк двумерного массива A(N, N), количество отрицательных элементов в которых больше P.
6. Ввести двумерный массив размером 7*4. Найти наибольший элемент двумерного массива. Удалить строку с максимальным элементом.
7. Ввести двумерный массив размером 7*4. Поменять столбец с максимальным элементом с первым столбцом двумерного массива.
8. Ввести двумерный массив размером 7*7. Найти максимальный элемент двумерного массива, расположенный ниже побочной диагонали.
9. Ввести двумерный массив размером 7*4. Найти наименьший элемент двумерного массива. Перенести строку, содержащую этот элемент в конец.

10. Ввести двумерный массив размером 7*4. Найти максимальный элемент двумерного массива. Поменять столбец, содержащий этот элемент с последним столбцом двумерного массива.
11. Ввести двумерный массив размером 6*4. Найти минимальный элемент двумерного массива. Переставляя строки и столбцы, добиться того, чтобы элемент оказался в правом нижнем углу.

Контрольные вопросы

1. Дайте определение массива.
2. Чем массив отличается от последовательности значений?
3. Что определяет номер элемента массива?
4. Сформулируйте свойства массива.
5. Какие типы алгоритмических структур применяются для обработки массива?
6. Какие существуют способы обработки одномерного массива?
7. В чем заключается сортировка одномерного массива?
8. Опишите методы сортировки одномерного массива.
9. Приведите пример и алгоритм сортировки массива.
10. Опишите методы обработки упорядоченных массивов.
11. В чем заключается сущность метода двоичного поиска?
12. Чем двумерный массив отличается от одномерного?
13. Как осуществляется доступ к элементу двумерного массива?
14. Какие существуют способы обработки двумерного массива?

Часть II. Программирование на языке Turbo Pascal

1. ОСНОВНЫЕ ПОНЯТИЯ

Для того чтобы составить программу на языке Turbo Pascal надо знать структуру программы и правила использования в программах различных данных.

Самое общее правило гласит: все данные, используемые Вами, должны быть предварительно определены, т. е. Вам необходимо задуматься

- о том, как обозначить каждое данное (какое дать ему имя);
- о характере и диапазонах изменений их значений;
- о требуемой памяти для их размещения;
- о наборе допустимых к ним операций.

Для определения данных Turbo Pascal предлагает использовать способы объявления данных в программе посредством задания их значений или путем задания их типов – для данных, изменяющих свои значения при выполнении программы.

> Тип данных определяет форму машинного представления, допустимый набор значений, а также те действия, которые могут быть выполнены над этими данными

При определении данных посредством задания значений, тип данных может указываться, а может и не указываться, в этом случае он определяется автоматически.

Если стандартных средств для реализации необходимых операций недостаточно, то Вам следует определить дополнительные типы или операции. Это можно реализовать или в самой программе, или вне вашей программы, используя модули – библиотеки объявлений.

Рассмотрим обобщенную структуру программы на языке Turbo Pascal:

```
program <имя программы>;  {заголовок программы}
 uses <имена модулей>;       {раздел использования модулей}
                            {---------- РАЗДЕЛ объявлений ----------}
 const                       {раздел  объявлений значений-констант}
  <имя константы>= <значение>;
  <имя константы>:<тип>=<значение>;
    .....
```

```
type                          {раздел объявлений дополнительных типов}
 <имя типа>=<тип>;
  .................
var                           {раздел объявлений данных}
 <имя переменной>:<тип>;
  .................
label                         {раздел объявлений меток}
  <имя метки>;
  ..............
procedure <имя >(параметры); {раздел объявлений подпрограмм}
  ..............
end;                          {конец подпрограммы}
function <имя >(параметры);  {раздел объявлений подпрограмм}
  ..............
end;                          {конец подпрограммы}
                              {-------- РАЗДЕЛ ОПЕРАТОРОВ ----------}
begin
   ........
end.
```

Все имена, используемые в программе, могут иметь любую длину, должны начинаться с буквы и не должны содержать разделителей.

Текст, заключенный в фигурные скобки {}, называется **комментарием** и может располагаться в любом месте программы.

Последовательность объявлений не имеет значения, важно только, чтобы любое данное, тип, константа, подпрограмма были описаны до их использования.

Целые типы

В Turbo Pascal используются пять целочисленных типов данных, наиболее распространенным среди которых является тип INTEGER.

Таблица 1

Целые типы данных

Тип данных	Диапазон значений	Размер в байтах
BYTE (целое длиной в байт)	0..255	1

Окончание табл. 1

SHORTINT (короткое целое)	-128..127	1
INTEGER (целое)	-32768..32767	2
WORD (длиной в слово)	0..65535	2
LONGINT (длинное целое)	-2147483648..2147483647	4

Примеры объявления целочисленных данных:

```
var
 a:integer;
 c,d:byte;
 f:shortint;
const
 step=1;
 mm:word=65500;
```

Над целыми значениями допустимы следующие операции и функции:

sqr(x) – возведение в квадрат X,
abs(x) – модуль X,
+ сложение,
- вычитание,
* умножение,
/ деление,
div – деление нацело,
mod – получение целочисленного остатка и др.

Рассмотрим действия двух последних операций:

12 mod 2 = 0, так как 12 делится на 2 без остатка.

13 mod 2 = 1, 12 div 2 = 6, 10 div 3 = 3.

Вещественный тип

Эта группа типов обозначает множество вещественных значений в различных диапазонах. Turbo Pascal поддерживает четыре различных вещественных типа, которые приведены в таблице 2.

Таблица 2

Вещественный тип данных

Вещественный тип	Диапазон десятичного порядка	Число цифр мантиссы	Размер в байтах
REAL (вещественный)	-39...+39	11...12	6
SINGL (с одинарн. точн.)	-45...+38	7...8	4
DOUBLE (с двойн. точн.)	-324...+308	15...16	8
EXTENDED (повыш. точности)	-4951...+4932	19...20	10
COMP (сложный)	$-2^{63}+1...+2^{63}-1$	19...20	8

В памяти ПЭВМ вещественное число занимает участок памяти в несколько байт (от 4 до 10), который имеет следующую структуру:

ЗНАК	ПОРЯДОК	МАНТИССА

Вещественные числа типа SINGLE, DOUBLE, EXTENDED могут быть использованы в программах, если в конфигурации ПЭВМ имеется математический сопроцессор. Однако, если в программе установить директивы компилятора {$ N+}, {$ E+}, то программа с этими типами будет работать и без сопроцессора правильно.

Значения вещественных чисел могут быть представлены следующим образом:

```
-5.09
0.445
3.12E-01 {эта запись обозначает число 3.12*10^-1 или 0.312}
```

4.5E+10.

Рассмотрим примеры объявления данных вещественного типа.

```
const
  beg1=0.1234;
  en1:real=333E-11;
var
  x,y,z : real;
```

Над вещественными значениями допустимы следующие встроенные функции:

sin(x),
cos(x),
arctan(x),
exp(x) {экспонента},
frac(x) {дробная часть},
int(x) {целая часть},
ln(x),
round(x) {округление X до ближайшего целого},
sqrt(x) {корень квадратный}.

Логический тип

Данные логического типа могут принимать только два значения, которые обозначаются в программах как true и false, при этом считается, что true > false. Размер памяти для размещения значения логического типа определяется в один байт. Рассмотрим примеры объявления логических данных в разделе объявлений программы:

```
const
  a=true;
var
  b:boolean;
```

Для данных логического типа определены следующие логические операции

NOT – отрицание	NOT(True)=false;
OR – логическое сложение (ИЛИ)	True OR False = True;
AND – логическое умножение (И)	False AND True = False;
XOR – исключающее ИЛИ	True XOR True = False;
	True XOR False = True;

Символьный тип

Множеством значений символьного типа являются символы, упорядоченные в соответствии с расширенным набором кодов ASCII, включающим символы заглавных и строчных букв, цифр и специальных символов. Значение символьного типа занимает один байт и изображается в программах с помощью апострофов, например:

'f' 'Б' '='.

Так как каждый символ имеет свой уникальный код, то допускается использовать значение символьного типа посредством задания его целочисленного кода, например:

#10 #65 #13.

Для получения кода символа S можно применить функцию

ord(S).

С другой стороны любую величину символьного типа можно получить с помощью стандартной функции

chr(<Код символа>).

Над значениями символьного типа допустимы операции сравнения, которые выполняются над кодами этих символов.

Рассмотрим пример объявления переменной символьного типа:

```
 var
   cimv:char;
   const
   vsym='A';
```

Строковый тип

Данные строкового типа представляют собой последовательности символов переменной длины (от 1 до 255). Такие данные можно описывать в программах следующим образом:

```
    var
     vystr:string[18];{vysrt - 18 символов}
     str1:string[88];{str1- 88 символов}
     str2:string;{str2- 255 символов по умолчанию}
   const
     one_str='PASKAL';
     two_str='#13#70';
```

Фактическую длину строки SS можно определить с помощью стандартной функции length(SS).

К любому символу строки можно обратиться по его номеру, например: str1[2],str2[200].

К строкам можно применять операцию сцепления или конкатенации «+»:

vystr + one_str, 'Моя '+'программа',
а также операции отношения, которые выполняются над строками посимвольно слева направо с учетом кодов сравниваемых символов.

Рассмотрим некоторые функции, определенные над символьными данными:

copy(st,index,count) – функция копирует из строки st;

count символов, начиная с символа с номером index;

pos(subst,st) – функция отыскивает в строке st первое вхождение подстроки subst.

Контрольные вопросы

1. Что такое строки, какие стандартные функции применимы к ним?
2. Как изображаются и объявляются числа?
3. Что включают в себя имена данных?
4. Какие существуют способы определения данных?
5. Что такое тип данных и какие типы данных существуют в Turbo Pascal?
6. Диапазон значений типа BYTE, допустимые операции и функции для целых значений?
7. Из каких разделов состоит программа Turbo Pascal?
8. Какие допустимы операции над логическими данными, какие значения они могут иметь?
9. Для каких целей используют функции ORD и CHR?

2. ВВОД И ВЫВОД ЗНАЧЕНИЙ ДАННЫХ

Ввод исходных данных, над которыми должны выполняться преобразования в программе, можно организовать различными способами.

Рассмотрим эти способы.

2.1. Ввод с клавиатуры

Для этого используются операторы ввода

Read(<список имен переменных>) и

Readln(<список имен переменных>).

Эти операторы позволяют выполнять программу с различными значениями исходных данных.

Получив инструкцию read или readln, ПЭВМ приостанавливает работу и ждет, когда пользователь введет с терминала значения, которые будут по очереди присваиваться переменным, стоящим в списке ввода. Когда все переменные примут определенные пользователем значения, выполнение программы будет продолжено с оператора, следующего за read\readln.

В отличие от оператора read оператор readln после ввода значений всех указанных переменных осуществляет переход к следующей строке.

> При использовании read ввод последовательности значений с клавиатуры может осуществляться либо через пробел либо через клавишу ввод, а при использовании readln – только через клавишу ввода (ENTER).

Пример 2.1.

```
program with_klav;
var
  a,x:real;
  b:integer;
  c:char;
  stroka:string;
begin
   read(a,x);
   readln(b);
   readln(c);
   readln(stroka);
end.
```

> Данные логического типа вводить с клавиатуры нельзя.

2.2. Ввод с помощью константы

В разделе объявления констант происходит одновременное определение типа данных и их значений:

const
<имя>=<значение>; {именованная}
или
const
<имя>:<тип>=<значение>; {типизированная}

Типизированная константа в отличие от простой константы может изменять свое значение в программе.

Пример 2.2.

```
program post;
const
  h=7;              {константа целого типа}
  y=5.87;           {константа вещественного типа}
  x:real=6.98799; {типизированная константа}
  t='*';    {константа символьного типа}
  r='информатика'; {константа строкового типа}
  n:boolean=false;   {константа логического типа}
begin
   x:=x+h*y;
end.
```

2.3. Ввод с помощью оператора присваивания

Оператор присваивания := является основным оператором языка; он предназначен для изменения значения данных. Его обобщенный вид:

<переменная>**:=**<значение>;

при этом тип переменной и тип значения должны совпадать.

Пример 2.3.

```
program prisv;
var
  y: integer;
  k,x: real;
  v: char;
  s: string;
  log: boolean;
```

```
begin
    y:=4+188;
    k:=6.8;
    x:=6.767E-02-k;
    v:='*';
    log:=true;
    s:='информатика';
end.
```

Обратите внимание: каждое объявление и каждый оператор заканчиваются знаком «;».

2.4. Ввод с помощью датчика случайных чисел

Датчик случайных чисел используется в программах для генерации случайных значений.

Для того, чтобы в результате всегда выводились разные случайные числа в начале программы, вызывается стандартная процедура:

Randomize.

Затем в разделе операторов вызывается функция Random или Random(P), P – целое число, например:

```
X:=random(40);
Y:=random;
```

в этом случае переменная X примет в качестве своего значения целое случайное число в диапазоне от 0 до 39, а переменная Y – случайное число в диапазоне от 0 до 1.

Пример 2.4.

```
program datchik;
var
  a:integer;
  b,c:real;
begin
    randomize;
    a:=random(20);
    b:=10*random;
    c:=random;
end.
```

2.5. Вывод на экран

Общий вид процедур вывода:

Write(V1[:W1[:D1]],...,Vn[:Wn[:Dn]]);
Writeln(V1[:W1[:D1]],...,Vn[:Wn[:Dn]]);

В квадратных скобках – необязательные части процедур. Процедура вывода writeln обеспечивает завершение печати текущей строки и переход к следующей строке;

V1,...,Vn –выражения, значения которых выводятся на печать;

Wi и Di – выражения целого типа, определяющие соответственно общее число позиций и число позиций после запятой (Di – только для значений выражений вещественного типа).

Примеры операторов вывода значений переменных a, s, d целого типа.

а) ...write(a,s); write(d);...

Все числа печатаются на одной строке. Предположим, a=12, s=-25, d=7. Тогда напечатанная строка будет иметь вид:

12-257

Аналогичный результат можно получить, написав одну процедуру write(a, s, d).

б) ...writeln(a,s); write(d);...

Значения a и s печатаются на одной строке. Следующая процедура вывода осуществляет печать со следующей строки. Результат печати:

12-25
7

в) ...write(a:2, s:7, d:4);...

В этом случае под значение переменной a отводится 2 позиции, под s – 7 позиций, под d – 4 позиции, т. е.

12 –25 7

Если количество указанных позиций недостаточно, то происходит автоматическое увеличение поля до необходимых размеров.

г) ...write('A=',a:2,' S=',s:7,' D=',d:4);...

Здесь используется возможность вывода строк символов. При этом будет напечатано:

A=12 S=-25 D=7

Рассмотрим вывод значений вещественных чисел.

В случае задания общего количества позиций под выводимую величину можно задать и количество позиций после запятой.

Пусть A, S, D – переменные вещественного типа, A=123,456; S=23456,7; D=-3,4567.

д) ...writeln(a:6:2, s:9:2, d:8:4);...

Печать: 123,45 23456,70 –3,4567

е) Если указывается общее число позиций (W) и не указывается число позиций после запятой (D), то числа выводятся в экспоненциальной форме с шириной поля W.

ж) ...writeln(a, s, d);...

Ширина поля стандартная, числа выводятся в экспоненциальной форме:

1.2345000000E+03

2.3456700000E+04

-3,456700000E+00

При выводе значений символьного типа под каждый символ отводится 1 позиция, например, оператор writeln('S1=',s1:5,' S2=',s2); напечатает (Если символьной переменной s1 в программе присвоено значение '*', а s2 – 'W') следующую строку:

S1= * S2=W.

При выводе значений булевского типа на печать выводится TRUE или FALSE. Например, процедура writeln(a<s); напечатает слово TRUE, если значение переменной a<s, и слово FALSE в противном случае.

Пример 2.5.

```
const
  a1=12;
  a2=234;
  a3=–2;
  x1=1342.567;
  x2=0.00234;
  s1='*';
  s2='–';
  str1='информатика';
  str2='наука';
  log=false;
begin
  writeln('Вывод значений данных...');
  writeln('      ...целого типа:');
  write(a1,' ',a2,' ',a3);
  writeln(a2,a3);
  write(a1);
  writeln(a1:5,a2:7,a3:3); {под значение переменной a1 отводится 5 позиций,
                                        под a2 – 7 позиций, под a3 – 3 позиции}
  writeln('      ...вещественного типа:');
  write(x1);
  writeln(' ',x2);
  write(x1:6:2,x2:8:3);          {задается количество позиций под  выводимые
                                 величины  и количество позиций  после запятой}
  writeln(x2:4,x1:5);
  writeln('      ...символьного типа:');
  writeln(s1,s2);
```

```
  writeln(str1,' ',x1:4);
  write(log);
  write(a2:7,x2:3);
  writeln(str1,s2,str2);
  write(a1>x2);
end.
```

Контрольные вопросы

1. Напишите оператор ввода целой переменной К с клавиатуры.
2. Для каких целей используется оператор присваивания?
3. Чем отличается именованная константа от типизированной?
4. Какие существуют способы ввода случайных значений?
5. Какие типы данных в Turbo Pascal нельзя вводить с клавиатуры?
6. Напишите оператор вывода вещественного X в формате три знака до и два после запятой.
7. Какие существуют способы ввода значений данных?
8. Чем должны заканчиваться объявления и операторы?
9. Опишите форматный вывод.
10. Чем отличаются операторы ввода READ и READLN?
11. Чем отличаются операторы вывода write и writeln?

3. ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ

Практически в каждой задаче требуется выбрать необходимые действия из двух, либо из некоторого множества действий (с числом элементов более двух). Для решения этой задачи служат операторы условного перехода if и оператор безусловного перехода goto.

Рассмотрим последовательно эти операторы.

3.1. Условный оператор

Условный оператор позволяет проверить некоторое условие и в зависимости от результатов проверки выполнить то или иное действие.

Таким образом, условный оператор – это средство ветвления вычислительного процесса.

Условный оператор позволяет выбрать одно из двух действий. Существует 2 вида условного оператора:

1) **if** b **then** s1;

2) **if** b **then** s1 **else** s2 ;

где B – выражение логического типа;

s1,s2 – отдельные операторы или операторы, сгруппированные вместе при помощи операторных скобок begin – end. Такой оператор называется **составным**.

Для условного оператора 1-го вида, если выражение B принимает значение true, выполняется оператор s1, стоящий после then. Если же значение выражения B – false, то оператор s1 не выполняется.

Для условного оператора 2-го вида, если выражение B принимает значение true, то выполняется оператор s1, стоящий после then, иначе выполняется оператор s2, стоящий после false.

> Зарезервированному слову else в операторе if не должна предшествовать точка с запятой.

Рассмотрим пример

```
.....
 if x>max then
      y:=max
      else
      y:=x;
```

При выполнении этого фрагмента программы, переменная y получит значение x, если только это значение не превышает max, в противном случае y станет равным max. В качестве условия могут использоваться различные условные выражения

x>0,

```
(x>9) and (y<2) {круглые скобки обязательны!}
(f<>r) or (r>=4)
not (ttt<=1).
```

Следует помнить, что условный оператор управляет только одним оператором (т. е. после ключевых слов then и else может стоять только один оператор), поэтому если требуется произвести более 1-го действия, необходимо использовать составной оператор begin – end.

Рассмотрим другой пример:

```
(фрагмент программы)
 var
 x,y,max:real;
       .............
 if (x<0) and (x>max) then
begin
   max:=x;
   write(max);
end;
 y:=x;
```

В этом примере переменная y всегда будет иметь значение переменной x, а в max запоминать максимальное отрицательное значение x. Для обеспеченности однозначности в языке Turbo Pascal принято соглашение о том, что каждому else соответствует предыдущий свободный if:

```
if a>b then
      if c<0  then
            write(c)
           else
            c:=0;
```

3.2. Безусловный оператор

В программе можно осуществить переход, прервав последовательное выполнение программы. Для этого служит оператор безусловного перехода, имеющий вид:

GOTO < имя метки >;

В качестве меток используются целые числа от 0 до 999 или имена, начинающиеся с буквы. Метка должна быть объявлена в разделе объявления меток (label). Она используется перед оператором и отделяется от него двоеточием.

```
program es;
        label 1;
        var
          b,c:integer;
        begin
         1:writeln ('Введите B и C');
         read(b,c);
         if b>=c then goto 1
                else writeln ('B меньше C);
        end.
```

При выполнении данной программы сообщение «Введите B и C» будет выводиться на экран пока вы не введете b < c. В этом случае на экран будет выдано сообщение «B меньше C» и программа закончит свою работу.

> Старайтесь обходиться в программах без меток
> и Вы начнете себя уважать.

Рассмотрим некоторые программы, реализующие разветвляющиеся алгоритмы:

Пример 3.1. Поиск максимального из двух введенных чисел.

```
program n;
var a,b :integer;
 begin
  writeln('Введите два числа ');
  write('Введи a=  ');
  readln(a);
  write('Введи b= ');
  readln(b);
  if a>b then   write('максимальное число a= ',a)
        else
          begin
            if a=b then
                    write('Вы ввели два равных числа ');
            if a<b then
                    write('Максимальное число b= ',b);
           end;
   readln;
 end.
```

Пример 3.2. Поиск максимального четного числа из двух введенных.

```
program kurs4;
```

```
var
  a, b, max : integer;
begin
  writeln('Из двух введенных чисел определить максимальное четное');
  writeln('Введите a,b');
  read(a,b);
   if  a mod 2=0  then
                    if b mod 2=0   then
                                    if a>=b then   max:=a
                                             else    max:=b
                                    else   max:=a
                    else
   if b mod 2=0 then
                     max:=b
                    else
                      writeln('Нет четных чисел ');
writeln('Максимальное  четное число равно', max:5);
writeln('Нажмите enter.');
readln;
end.
```

Пример 3.3. Программа проверяющая, можно ли из отрезков с длинами x, y, z, построить треугольник.

```
program gar4;
var
  x, y, z: real;     {Длины отрезков, из которых строится треугольник}
  r: boolean;      {Переменная логического типа}
begin
  writeln('Составить программу проверяющую, можно ли из ');
  writeln('отрезков с длинами x, y, z, построить треугольник.');
  writeln('Введите x, y, z');
  read(x, y, z);
  r:=((x+y)>z) and ((x+z)>y) and ((z+y)>x) ;
  if r then
     writeln('Треугольник можно построить')
     else
     writeln('Треугольник нельзя построить');
     writeln('Нажмите enter.');
readln;
end.
```

Пример 3.4. Вычислить значение y в зависимости от значения x.

Если x<–0.5, y=1+sqrt(cos(x)),

x>1, y=1–x*x,

```
                                в противном случае       y=x+1.
program gar5;
var
  y,x:real;
begin
  writeln('Вычислить значение y в зависимости от значения x.');
  writeln('Если    x<–0.5                y=1+sqrt(cos(x)),');
  writeln('           x>1                 y=1–x*x.');
  writeln('в противном случае       y=x+1.');
  writeln('Введите x');
  read(x);
  if (x>=–0.5)and(x<=1)  then
     y:=x+1
  Else    if (x<–0.5) then if  (cos(x)>=0) then
                                y:=1+sqrt(cos(x));        {Оператор if }
                           else
                           begin                          {составной оператор}
                           writeln('Для x=',x,' нельзя вычислить функцию');
                           exit                            {завершение программы}
                           end;
            else   if x>1 then
                             y:=1–x*x;
 write('Значение y =',y:5:3);
 writeln('Нажмите enter ');
 readln;
 readln;
end.
```

Пример 3.5. Вычислить значение y в зависимости от x.

```
                    Если      x=1,                 то y не вычисляется,
                              x>0,                 y=2*x,
                              x>–1 и x<=10,     y=1–ln(abs(1–x*x)),
                              x<–1,                y=exp(–x).
program gar6;
var
x,y:real;
begin
writeln('Вычислить значение y в зависимости от x.');
writeln('Если      x=1                 то y не вычисляется,');
writeln('           x>0                 y=2*x,                  ');
writeln('           x>–1 и x<=10     y=1–ln(abs(1–x*x)),     ');
writeln('           x<–1                y=exp(–x).                ');
writeln('Введите y');
```

```
read(x);
if (x=1) then  write('y не вычисляется')
else
    begin
      if(x>0) then       y:=2*x
      else
           if(x>–1)and(x<=10)  then   y:=1–ln(abs(1–x*x))
            else
                 if x<–1  then        y:=exp(–x);
        writeln('Значение y=',y:5:3);
        writeln('Нажмите enter');
        readln;
        readln;
       end;
end.
```

Пример 3.6. Составить программу для решения квадратного уравнения a*x*x+b*x+c=0.

```
program gar7;
var
 a,b,c,d,x1,x2:real;
begin
 writeln('Составить программу для решения квадратного уравнения');
 writeln('a*x*x+b*x+c=0.');
 writeln('Введите a,b,c');
 read(a,b,c);
 d:=sqr(b)–4*a*c;
 if(a=0)and(b=0)and(c=0)
     then
      writeln('Уравнение имеет бесконечное множество решений')
     else
     if (a=0)and(b<>0)and(c<>0)
      then
       writeln('Уравнение имеет корень x=',c/b:5:3)
         else
          if d<0 then  writeln('Уравнение имеет комплексные корни')
          else
            if  d=0  then  writeln('Уравнение имеет 2 одинаковых корня
x=',–b/(2*a):6:3)
              else
                  begin
                     x1:=(–b+sqrt(d))/(2*a);
                     x2:=(–b–sqrt(d))/(2*a);
```

```
                writeln('x1=',x1:6:3,'     x2=',x2:6:3);
              end;
writeln('Нажмите enter');
readln;
end.
```

Корни уравнения находятся по формуле

$x_{1,2}=(–b+–sqrt(b*b–4*a*c))/2*a$.

При составлении программы необходимо предусмотреть следующие возможности:

1) если a=0, b=0, c=0, то уравнение имеет множество решений;
2) если a=0, b=0, c<>0, то уравнение не имеет решений;
3) если a=0,b<>0,c<>0, то уравнение имеет единственное решение x=c/b;
4) если a<>0, b<>0, c<>0, то решение уравнения зависит от значения дискриминанта d=b*b–4*a*c;
   a) если d<0, то уравнение имеет комплексные корни;
   b) если d=0, то уравнение имеет два одинаковых корня x = -b/(2a);
   c) если d>0, то уравнение имеет два корня x_1=(-b+sqrt(d)/(2a)), x_2=(–b–sqrt(d)/(2a)).

Пример 3.7. Составить программу для нахождения наибольшего значения функции y:=abs(a)*exp(a*x–x*x) при изменении аргумента x от 0 до b с шагом h.

```
program kurs2;
 label 1;
var
a, h, x, y, ymax, b:real;
begin
  writeln('Составить программу для нахождения наибольшего значения');
  writeln('функции y:=abs(a)*exp(a*x–x*x) при изменении аргумента ');
  writeln('x от 0 до b с шагом h.');
  writeln('Введите параметры a,h,b');
  readln(a, h, b);
  ymax:=–1e37;       { Задание начального значения ymax }
  x:=0;
 1:if x<b    then
        begin
        y:=abs(a)*exp(a*x–x*x);
        writeln('y=',y:4:2,'x=',x:4:2);  {Вывод на экран всех значений y,x на}
                                          {отрезке от 0 до b}
       if y>ymax then
        ymax:=y;                        {Выбор максимального значения y}
        x:=x+h;
```

```
    goto 1; {Безусловный переход на начало}
  end;
 writeln('Наибольшее значение функции','=',ymax:4:2);
 writeln('Нажмите ENTER.');
readln;
end.
```

Задания для самостоятельного выполнения

1. Для двух чисел X, У определить, являются ли они корнями уравнения $A*P^4+D*P^2+C=0$.
2. Если среди трех чисел A, B, C имеется хотя бы одно четное, вычислить максимальное, иначе – минимальное.
3. Ввести положительное A. Найти наибольшее число вида 1\P (P>=0), меньшее A.
4. Ввести два числа. Меньшее заменить полусуммой, а большее – удвоенным произведением.
5. Ввести три числа A, B, C . Удвоить каждое из них , если A>=B>=C, иначе поменять значения A и B.
6. Определить, является ли точка с координатами X, Y точкой пересечения диагоналей квадрата со стороной R, одна вершина которого расположена в начале координат.
7. Определить, лежит ли точка с координатами (X, Y) вне круга радиуса R с центром в точке (A, B) или внутри него.
8. Определить являются ли две точки (X1,Y1) корнями системы уравнений

$$\begin{cases} ax + by = c \\ nx + my = d \end{cases}$$

9. Вычислить

$$y = \begin{cases} \sin X\ , & X<0 \\ \\ tg\ X\ , & X>=0 \end{cases}$$

Контрольные вопросы

1. Напишите условие на языке Turbo Pascal X>0, Y>0 или Z=0.
2. Какие два вида условного оператора используются в программе?
3. Структура и алгоритм работы условного оператора.
4. Назначение составного оператора.
5. Что такое условие в операторе if?
6. Что обозначает ключевое слово if?
7. Что обозначают ключевые слова then, else?
8. Назначение оператора безусловного перехода

3.3. Программирование с оператором варианта CASE

Условный оператор позволяет при выполнении программы выбирать одно из двух возможных действий. Если же необходимо сделать много взаимоисключающих проверок, то удобней воспользоваться оператором выбора варианта.

Оператор варианта является обобщением условного оператора: он дает возможность выполнить один из нескольких операторов в зависимости от значения некоторого выражения, называемого **селектором**.

В общем случае оператор имеет вид:

case <селектор> **of**
<метка–1>:<оператор–1>;
<метка–2>:<оператор–2>;
.....
<метка–N>:<оператор–N>
else <оператор–(n+1)>;
end;

где case (выбор), of (из), end (конец) – служебные слова;
СЕЛЕКТОР – выражение любого типа, кроме вещественного и строкового;
ОПЕРАТОР – любой оператор языка, в том числе и составной;
МЕТКА – список значений выражения СЕЛЕКТОР или одно его значение.

Тип МЕТКИ совпадает с типом СЕЛЕКТОРА и состоит из любого числа отделенных друг от друга запятыми констант или диапазонов, за которыми следует двоеточие, например:

```
case berdsight of
   'C','c': curlews := curlews+1;
   'H','h': herons  :=  herons+1;
   'E','e': egrets  :=  egrets+1;
   'T','t': terns   :=   terns+1;
end;
```

Диапазоны записываются как две константы, отделенные друг от друга разделителем диапазона '..' . МЕТКА ВАРИАНТА – необычная метка: это не обязательно целое число, она не описывается в разделе label, на нее нельзя ссылаться в операторе goto.

ОПЕРАТОР, который следует за МЕТКОЙ, выполняется, если значение СЕЛЕКТОРА равно одной из констант или если он лежит внутри одного из диапазонов. В противном случае будет выполняться оператор, следующий за case.

Селектор иногда называют **ключом** выбора.

Пример 3.8. Определить день недели по его номеру.

```
program ex;
```

```
var
 dayno:integer;
 Begin
   write('введите число 1–7');
   read(dayno);
    case dayno of
          1:writeln('monday');
          2:writeln('tuesday');
          3:writeln('wednesday');
          4:writeln('thursday');
          5:writeln('friday');
          6:writeln('saturday');
          7:writeln('sunday');
        end;{of case}
     end.
```

Когда dayno равно 1, выбирается только оператор с меткой 1:. Когда dayno равно 2, выбирается оператор с меткой 2: и т. д.

Тип селектора и метки в операторе CASE не может быть строковым или вещественным.

Пример 3.9. Найти остаток от деления значения целого выражения C=K*(A+B) на 4 и вывести сообщение о величине остатка. Если остаток равен 0, то значение переменной C оставить без изменения, если 1 или 3 – уменьшить на величину остатка, если 2 – увеличить на величину остатка . Новое значение C вывести на печать.

```
program ostatok;
const
  del=4;
var
  a,b,c,k:integer;
  ost:integer;
begin
  writeln('Найти остаток от деления значения целого выражения');
  writeln('C=K*(A+B) на 4 и вывести сообщение о величине остатка.');
  writeln('Если остаток равен 0 , то значение переменной C');
  writeln('оставить без изменения , если 1 или 3 – уменьшить на');
  writeln('величину остатка , если 2 – увеличить на величину');
  writeln('остатка . Новое значение C вывести на печать.');
  write('Введите A=');
  read(a);
  write('Введите B=');
```

```
  read(b);
  write('Введите К=');
  read(k);
  c:=k*(a+b);
  ost:=c mod del;
  writeln('C=',c);
  case ost of
    0:writeln('Остаток 0');
    1:begin
        writeln('Остаток 1');
        c:=c–ost;
       writeln ('Новое значение C=',c);
        end;
    2:begin
        writeln('Остаток 2');
        c:=c+ost;
       writeln('Новое значение C=',c);
end;
    3:begin
        writeln('Остаток 3');
        c:=c–ost;
       writeln('Новое значение C=',c);
        end;
  end;{case}
end.
```

Пример 3.10. Вывести на экран число дней в любом месяце.

```
program mes;
var
 m,f,dn,d:integer;
begin
  writeln('Вывести на экран число дней в любом месяце ');
  write('   Введите месяц:');
 f:=0;
 read(m);
case m of
      1,3,5,7,8,10,12:d:=31;
      2:begin
          write('Сколько дней в году 365 или 366:');
          read(dn);
          if dn=365 then
                      d:=28
                   else
```

```
                    d:=29;
        end;
      4,6,9,11: d:=30;
    else f:=1;
  end;
  case m of
       1: write( 'в январе – ');
       2: write( 'в феврале – ');
       3: write( 'в марте – ');
       4: write( 'в апреле – ');
       5: write( 'в мае – ');
       6: write( 'в июне – ');
       7: write( 'в июле – ');
       8: write( 'в августе – ');
       9: write( 'в сентябре – ');
      10: write( 'в октябре – ');
      11: write( 'в ноябре – ');
      12: write( 'в декабре – ');
  end;
  case m of          1,3,5,7,8,10,12: write(d,' день');
                     2,4,6,9,11: write(d,' дней');
  end;
 end.
```

Пример 3.11. Напечатать заданное слово в заданном падеже, в единственном числе.

```
program pad;
var
  w,p:integer;
begin
   writeln(' Напечатать заданное слово в заданном падеже,');
   writeln(' в единственном числе.');
   writeln(' В каком падеже напечатать слово: ');
   writeln(' именительный (1), родительный (2), дательный (3),');
   write('    винительный (4),творительный (5), предложный (6)?: ');
   read(p);
   writeln(' Какое слово хотите изменить:');
   write('    степь (1), боль (2), тетрадь (3), дверь (4)? : ');
   read(w);
   case w of
       1:writeln(' Слово «степь» в');
       2:writeln(' Слово «боль» в');
       3:writeln(' Слово «тетрадь» в');
```

```
        4:writeln(' Слово «дверь» в');
  end;
  case p of
        1:write(' именительном падеже ');
        2:write(' родительном падеже ');
        3:write(' дательном падеже ');
        4:write(' винительном падеже ');
        5:write(' творительном падеже ');
        6:write(' предложном падеже ');
  end;
case p of
    1,2,4:case w of
                1:writeln('– «степь»');
                2:writeln('– «боль»');
                3:writeln('– «тетрадь»');
                4:writeln('– «дверь»');
           end;
    3,6:   case w of
                1:writeln('– «степи»');
                2:writeln('– «боли»');
                3:writeln('– «тетради»');
                4:writeln('– «двери»');
           end;
    5:   case w of
                1:writeln('– «степью»');
                2:writeln('– «болью»');
                3:writeln('– «тетрадью»');
                4:writeln('– «дверью»');
           end;
  end;
end.
```

Задания для самостоятельного выполнения

1. Составить программу, которая реализовала бы следующие действия: по введенному числу К (до 10) выдавала бы соответствующую ей римскую цифру.
2. Для целого числа К от 1 до 9 напечатать фразу «мне К лет», учитывая при этом, что при некоторых значениях К слово «лет» надо заменить на слово «год» или «года».
3. Для натурального числа К напечатать фразу «мы нашли К грибов в лесу», согласовав окончание слова «гриб» с числом К.
4. Составить программу, которая бы реализовала следующий алгоритм: переменной Т присвоить значение true если сочетание день.месяц.год

образует правильную дату, и значение false – иначе (учитывая количество дней в месяце и название месяца).
5. Составить программу, которая бы реализовала следующий алгоритм: по порядковому номеру дня года определить дату, т. е. число и месяц.

Контрольные вопросы

1. Отличие оператора варианта от условного оператора.
2. Для чего служит ключ выбора и какого он может быть типа?
3. Сколько меток может быть перед оператором в списке выбора?
4. Как выполняется оператор варианта:
   а) имеющий часть ELSE,
   б) не имеющий часть ELSE.
5. Использование составного оператора в операторе варианта.
6. Как работает оператор case?
7. Какой тип может иметь селектор?

4. ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

При разработке алгоритмов решения большинства задач возникает необходимость повторения ряда шагов. В языке Turbo Pascal имеются три различных оператора цикла, с помощью которых можно запрограммировать повторяющиеся фрагменты программ: с параметром, с предусловием и с постусловием.

4.1. Использование операторов цикла с условиями

В языке Turbo Pascal существуют два оператора цикла с условиями. С их помощью можно запрограммировать повторяющиеся фрагменты программы. Операторы цикла с условиями проверяют условия выполнения или повторения цикла.

Рассмотрим их.

Оператор цикла с предусловием

While <условие> **do** <оператор>;

здесь while, do – зарезервированные слова;
<условие> – выражение логического типа;
<оператор> – произвольный оператор, в том числе и составной.

Логическое выражение <условие> определяет, будет ли цикл выполняться или завершит свою работу. Если выражение <условие> имеет значение true (ист.), то выполняется <оператор>, после чего вычисление выражения <условие> и его проверка повторяются. Если <условие> имеет значение false (лож.), оператор while прекращает свою работу, т. е. цикл заканчивается. В качестве <оператора> может быть либо оператор, либо составной оператор, т. е. последовательность операторов, заключенная в операторные скобки begin ... end.

Пример 4.1.

```
program ctep; {возведение в степень}
const
   h:byte=1;
var
  a,b:integer;
  x:real;
  k:integer;
begin
writeln('Возведение в третью степень чисел от 1 до 10');
```

```
  a:=1;
  b:=10;
  k:=3;
while a<=b do
  begin
     x:=exp(k*ln(a));
    writeln(a,'^3=',x:4:0);
    a:=a+h
  end;
end.
```

Пример 4.2. Табулирование функции y=sin(x) на отрезке [1, 5] с шагом 0.5.

```
program tab1;
const
   h:real=0.5;
var
   x,a,b:real;
begin
writeln('Табулирование функции y=sin(x) на отрезке [1,5] с шагом 0.5');
  a:=1;
  b:=5;
  x:=a;
while x<=b do
  begin
     y:=sin(x);
     writeln('y=', y:2:1);
     x:=x+h;
   end;
end.
```

Пример 4.3. Сумма значений функции $y=x^2$ на отрезке [1, 5] с шагом 1.

```
program summa;
const        h:real=1;
var
   i,k:integer;
   x,a,b,s:real;
begin
  a:=1;      b:=5;
  x:=a;      s:=0;
writeln('Сумма значений функции y=x^2 на отрезке [1,5] с шагом 1');
while x<=b do begin
           y:=x*x;
```

```
            writeln('y=',y:2:1);
            x:=x+h;
            s:=s+y;
            end;
 writeln('Сумма='  s:2:1);
end.
```

Пример 4.4. Найти минимальное значение функции y=sin(x) на отрезке [1,10] с шагом 0.1

```
program min;
const
  h:real=0.1;
var
 y,b,x,min:real;
begin
 x:=1;
 b:=10;
 min:=sin(x);
writeln('Минимальное значение функции y=sin(x) ');
writeln(' на отрезке [1,10] шагом  0.1');
 while x<=b  do
        begin
            y:=sin(x);
            writeln(y:4:2);
            if y<min  then
                min:=y;
                x:=x+h;
            end;
 writeln('min=', min:2:1);
end.
```

Пример 4.5. Нахождение минимального нечетного и максимального четного элемента последовательности чисел.

Алгоритм:

1. присвоим min максимальное число, а max минимальное;
2. открываем цикл пока ch<>'n';
   2.1. вводим элемент последовательности a;
   2.2. если элемент четный, и если элемент > max,
        то max:=(значение этого элемента);
   2.3. если элемент нечетный, и если элемент < min,
        то min:=(значение этого элемента);
3. печать результата.

Четность и нечетность определяется при помощи оператора <mod>: определение дробной части от деления на 2.

```
var
i, max, min, a: integer;
ch: char;
begin
writeln('нахождение минимального нечетного');
writeln('и максимального четного ');
writeln('элемента последовательности');
write('вводить посл.(y/n)');
read(ch);
if ch='n' then halt(1);  { выход из программы }
max:=-32767;
min:=32767;
while ch<>'n' do begin
readln(a);
  if (a mod 2)=0 then
                if a>max then max:=a;
  if (a mod 2)<>0 then
                 if a<min then min:=a;
write('продолжить ввод(y/n)');
read(ch);
end;
writeln('миним.нечетн=',min);
writeln('макс.четн.=',max);
readln;
end.
```

Оператор цикла с постусловием

Repeat <тело цикла> **until** <условие>;

здесь repeat, until – зарезервированные слова (повторять до тех пор, [пока не будет выполнено условие]);
<тело цикла> – произвольная последовательность операторов;
<условие> – выражение логического типа.

Операторы <тело цикла> выполняются хотя бы один раз, после чего вычисляется выражение <условие>: если его значение есть FALSE (лож.), операторы <тело цикла> повторяются, в противном случае оператор repeat...until завершает свою работу.

Пример 4.6. Табулирование функции y=sin(x) на отрезке [1, 5] с шагом 0,5.

```
program tab2;
const
  h: real=0.5;
var
  a, b, x: real;
begin
writeln('Табулирование функции y=sin(x) на отрезке [1,5] с шагом 0,5');
a:=1;
b:=5;
x:=a;
repeat
  writeln('x=',x:2:1,'y=',sin(x));
  x:=x+h;
until x>=b;
end.
```

Пример 4.7. Корректный ввод данных.

```
program tab2;
const
  h:byte=0;
var
  i:integer;
begin
repeat
write('Введите номер месяца вашего рождения');
readln( i );
if (i<0) and (i>12) then
             begin
             writeln('Вы ввели некорректные данные');
            end
            else
            begin
              writeln('Вы ввели корректные данные');
               h:=1;
            end;
until h=1;
readln;
end.
```

Пример 4.8. Табулирование функции y=sin(x) на отрезке [1, 5] с шагом 0,5. Вывод предпоследнего положительного значения функции.

```
program tab2;
const
  h:real=0.5;
var
  a, b, x, p, y: real;
begin
writeln('Табулирование функции y=sin(x) на отрезке [1,5] с шагом 0,5');
writeln('Вывод предпоследнего положительного значения функции');
a:=1;
b:=5;
x:=a;
y:=0;
repeat
  if sin(x)>0 then
          begin
            p:=y;        { запоминание предыдущего значения }
            y:=sin(x);   { вычисление текущего значения  y }
          end;
  writeln('x= ',x:3:1,'y= ',sin(x):4:2);
  x:=x+h;
until x>=b;
writeln('Предпоследнее положительное значение функции = ',p:4:2);
write('Нажмите ENTER');
readln;
end.
```

Пример 4.9. Вычислить максимальное значение функции y=sin(x) на отрезке [1, 10]. Шаг=0.5.

```
program max;
const
  h: real=0.5;
var
  y, b, x, max: real;
begin
writeln('Максимальное значение функции y=sin(x) на отрезке [1,10].
Шаг=0.5');
 x:=1;
 b:=10;
 max:=–1; { начальное значение максимума }
repeat
 y:=sin(x);
 writeln(y:5:3);
 if y>max then max:=y;
```

```
 x:=x+h;
 until x>b;
writeln(max:2:9);
end.
```

Отличия двух операторов цикла с условиями

1. В операторе цикла с предусловием возможна такая ситуация, что операторы, содержащиеся внутри него, не будут выполнены ни одного раза, т. к. условие выхода из цикла там проверяется перед выполнением оператора. В операторе с постусловием оператор, содержащийся в цикле, выполняется хотя бы один раз.

2. В этих двух операторах по-разному проверяется логическое выражение на выход из цикла.

4.2. Оператор цикла FOR

Если число повторений тела цикла заранее известно, то чаще всего применяется оператор цикла с параметром. Общий вид оператора:

for <параметр_цикла>:=<нач.знач.> **to** <кон.знач.> **do** <оператор>;
или
for <параметр_цикла>:=< кон.знач> **downto** < нач.знач> **do** < оператор >;

где <параметр_цикла> – параметр цикла, переменная типа integer (точнее, любого порядкового типа);
<нач.знач.> – начальное значение – выражение, задающее начальное значение параметра цикла;
<кон.знач.> – конечное значение – выражение, задающее конечное значение параметра цикла.

Например:

```
а) For  i:=1 to 10   do
        a:= a+1 ;
б) For  i:=1 to 20   do
       begin
           a:= c+b ;
           WRITELN (a,b,c) ;
            c:= c+1 ;
        end ;
в) For  k:=20 downto 1 do
           WRITELN (k*k)
```

Задать шаг, отличный от 1 или –1 нельзя!

Выход из цикла

В циклах с условиями при большом объеме вычислений возможна ситуация, когда время выполнения цикла становится слишком большим.

В этом случае необходимо использовать стандартную процедуру прекращения выполнения цикла:

break.

Эта процедура используется также для досрочного выхода из цикла в случае получения результата до конца выполнения цикла. Процедура continue используется для досрочного перехода к следующей операции цикла.

Пример 4.10. Подсчет в строке букв <A> и <B>.

```
program k1;
var
a:char;
p,i,s:integer;
begin
  writeln('Подсчет в строке букв  <A> и <B> ');
  writeln;
  s:=0;
  p:=0;
  writeln('Введите строку из 10 букв без пробелов');
  for i:=1 to 10 do
   begin
   read(a);
   if a='a' then s:=s+1;
   if a='b' then p:=p+1;
   end;
  if s>0 then writeln ('Букв «a» в строке ',s)
        else writeln('Букв «a» в строке нет');
  if p>0 then writeln ('Букв «b» в строке ',p);
        else writeln('Букв «b» в строке нет');
  writeln;
  writeln('Для окончания работы нажмите ENTER');
  readln;
end.
```

Пример 4.11. Возведение числа в степень.

```
program k7;
var
 k,i:integer;
 n,y:longint;
 begin
   writeln;
   writeln('  Возведение числа в степень ');
   y:=1;
   writeln('введите число, возводимое в степень');
   writeln(' { число должно быть меньше 20 }');
   readln(k);
   writeln('введите целую степень  { меньше 5 }');
   readln(n);
   for i:=1 to n do y:=y*k;
   writeln;
   writeln('число ',к,' в степени ',n, '= ',y);
   writeln;
   writeln('Для окончания работы нажмите ENTER');
   readln;
   end.
```

Пример 4.12. Вычисление:
1: величины отчисления в подоходный налог ежемесячно,
2: среднемесячной зарплаты за текущий период.

```
program k5;
var
 i,k:integer;
 p,z,c,s:real;
begin
   writeln('вычисление:');
   writeln(' 1: величины отчисления в подоходный налог ежемесячно');
   writeln(' 2: среднемесячной зарплаты за текущий период ');
   writeln;
   writeln('введите количество месяцев      ');
   read(k);
   writeln('введите процент подоходного налога      ');
   read(p);
   c:=0;
   for i:=1 to k do
    begin
     writeln('введите зарплату за  ',I,' месяц    ');
     read(z);
     writeln('подоходный налог за ',I, ' месяц = ',p*z:4);
```

```
    writeln('————————————————————————————————————————');
    c:=c+z;
   end;
  writeln('средняя зарплата за ',K,' месяцев = ',c/k:4);
  end;
  writeln('————————————————————————————————————————');
  writeln;
  writeln('Для окончания работы нажмите ENTER');
  readln;
  end.
```

Пример 4.13. Вычисление числа в факториале.

```
program k4;
var
  y, i, p:integer;
begin
  writeln('Вычисление числа в факториале ');
  writeln;
    y:=1;
   writeln('введите целое число  { меньше 18 } ');
   readln(p);
   for i:=2 to p do
    y:=y*i;
   writeln('Число ',p,'!= ',y);
   writeln;
   writeln('для окончания работы нажмите ENTER');
   readln;
   end.
```

Пример 4.14. Вычисление числа в факториале.

```
program k4;
var
  y,i,p:integer;
begin
  writeln;
  writeln('       Вычисление числа в факториале ' );
  writeln;
    y:=1;
   writeln(' Введите целое число  { меньше 18 } ');
   readln(p);
   for i:=p downto 2 do
    y:=y*i;
   writeln('число ',p,'!= ',y);
```

```
   writeln;
   writeln;
   writeln('для окончания работы нажмите ENTER');
   readln;
   end.
```

Пример 4.15. Найти количество минимумов и максимумов последовательности.

Алгоритм:

1. присваиваем переменным k и n минимальное и максимальное число соответственно;
2. открываем цикл:
2.1. вводим элемент последовательности a;
2.2. если a> или =(предыдущего максимума k), то
если a=(пред. макс. k), то количество максимумов i увеличивается на 1;
если a>(пред. макс. k), то количество максимумов i=1, а переменной k присваиваем значение a (k:=a).

Аналогичные действия проводятся с минимальными элементами.

```
program orion_posl;
var
 a,n,k:real;
 r,i,l,j:integer;
begin
 i:=0;
 j:=0;
 k:=–1e–39;
 n:=1E+38;
 write('введите количество элементов последовательности : ');
 read(r);
 writeln('введите последовательность');
 for l:=1 to r do
begin
  readln(a);
  if a>=k then
 begin
   if a=k then i:=i+1;
   if a>k then i:=1;
   k:=a;
 end;
       if a<=n then
 begin
    if a=n then j:=j+1;
    if a<n then j:=1;
```

```
    n:=a;
end;
 end;
  writeln('min=',n:2:0,'  кол-во min=',j);
  writeln('max=',k:2:0,'  кол-во max=',i);
 readln;  end.
```

Задания для самостоятельного выполнения

1. При табулировании функции y=ln(x–a) на отрезке [1, 10] с шагом h определить второе значение y, больше p.
2. Вычислить сумму ряда, общий член которого задан формулой $A_n=(x^n)/n!$.
3. В последовательности символов вывести на печать TRUE, если количество гласных букв больше, чем согласных, и FALSE, – если иначе.
4. В последовательности целых положительных чисел определить максимальное четное число и его порядковый номер.
5. В последовательности вещественных чисел определить наименьшее отрицательное число и его порядковый номер.
6. В последовательности целых чисел определить третье положительное число и подсчитать количество цифр в нем.
7. В последовательности символов выдать на печать TRUE, если значение последнего символа равно Ф.
8. В последовательности чисел выдать на печать TRUE, если значение максимального числа больше числа 10.
9. В последовательности вещественных чисел подсчитать произведение чисел, кратных 3.
10. В последовательности чисел сравнить, что больше: сумма положительных или произведение отрицательных.
11. В последовательности символов подсчитать количество букв и количество цифр.
12. В последовательности чисел определить предпоследнее отрицательное число.

Контрольные вопросы

1. Что такое цикл, параметр цикла?
2. Оператор цикла с параметром.
3. Оператор цикла с предусловием.
4. Оператор цикла с постусловием. Отличия от других операторов цикла.

5. ФАЙЛОВЫЙ ВВОД-ВЫВОД

Под **файлом** понимают некоторую область памяти для хранения однотипной информации, которая располагается обычно на диске и имеет имя. Рассмотрим, как организовать работу с файлами для хранения входных и выходных данных в программе.

Сначала необходимо объявить переменную типа файл по правилу

```
 var
<имя 1>:file of <тип>; {для типизированного файла}
<имя 2>:text; {для текстового файла}
<имя 3>:file; {для не типизированного файла}
```

Затем в разделе операторов (обычно в самом его начале) необходимо связать переменную типа файл с реальным именем файла. Для этого используется оператор обращения к стандартной процедуре assign:

assign(<имя 1>,<имя файла>).

Например,

```
assign(f1,'F1.DAT');
```

При работе с файлами основными режимами работы являются запись информации в файл и считывание информации из файла в некоторую переменную программы для анализа или обработки.

При этом необходимо следовать правилам:

– в самом начале программы, после обращения к процедуре assign, все используемые файлы нужно открыть;

– в конце программы все открытые файлы требуется закрыть.

Для открытия файла применяют одну из нижеприведенных стандартных процедур в зависимости от режима работы с файлом:

reset (<имя1>) – открывает файл, которому в программе соответствует файловая переменная <имя1>. Если <имя1> определяет типизированный файл, открываемый файл предназначается как для чтения так и для записи. Если <имя1> определяет текстовый файл, то открываемый файл предназначается только для чтения;

rewrite(<имя1>) – открывает файл только для записи в первоначально пустой файл;

append(<имя1>) – открывает текстовый файл для добавления.

Для закрытия любого файла используется стандартная процедура **close**(<имя1>).

5.1. Основные процедуры и функции для работы с файлами

Для ввода данных из файла используются процедуры read, readln в форме

read (<имя1>,<список ввода>).

Для вывода в файл – процедуры WRITE,WRITELN в виде

write (<имя1>, <список вывода>)

Здесь <имя1> – имя файловой переменной, и

<список вывода> – последовательность одной или более переменных.

Тип переменных должен соответствовать типу файловой переменной, если файл типизированный.

Для определения конца файла используется функция **EOF**(<имя1>): boolean. Она возвращает значение TRUE, если достигнут конец файла. При записи это означает, что очередной компонент будет добавлен в конец файла, при чтении – что файл исчерпан.

В типизированном файле доступ к каждому компоненту можно производить по его порядковому номеру или указателю.

Перед первым обращением к процедуре read или write указатель файла стоит в начале и указывает на компонент файла с номером 0. После каждого чтения/записи указатель сдвигается к следующему компоненту.

При работе с типизированными файлами можно применять дополнительные стандартные процедуры и функции:

– процедура **SEEK**(F,N) (для смещения указателя файла F к компоненту N),

– функция **FILESIZE**(F) (для определения количества компонент в файле F),

– функция **FILEPOS**(F) (для определения следующего номера компоненты файла F),

– функция **SEEKEOF**(F) (для смещения в конец файла F).

Пример 5.1. Ввод из файла 10 значений x, вычисление для них значений функции y= x*x+2*x–3 и нахождение максимального значения y.

```
program kr3;
var x:integer;
   ma,y:integer;
   i:byte;
   c:integer;
   f:file of integer;          {объявление файловой переменной }
begin
   assign (f,'mkurs.dat'); {связывание файловой переменной с именем файла}
   reset (f);                  {открытие файла для чтения}
   ma:=–32000;
```

```
  while not (eof(f)) do
       begin
        read (f,x);       {чтение из файла}
        y:=x*x+2*x–3;
        write (y,' ',x);
        if ma<y then    { поиск максимального значения }
                ma:=y;
      end;
  writeln;
  writeln ('Максимальное значение функции равно  ',ma);
  close(f);                {закрытие файла}
  readln;
  end.
```

Пример 5.2. В исходном файле найти максимальное значение, поменять его с первым элементом и результат записать в выходной файл.

```
program k4;
var fz: file of integer;        {объявление файловых}
   fr: file of integer;         {переменных fr – исходный, fz –выходной}
   ma,c,i,n: integer;
begin
    assign (fr,'mkurs.dat');  {связывание файловых переменных}
    assign (fz,'izm.dat');     {с именами реальных файлов }
    reset(fr);                {открытие файла fr для чтения}
    rewrite (fz);             {открытие файла fz для записи}
    ma:=–32768;
    n:=0;
     writeln ('Поменять местами макс. и первый эл–ты файла. ');
    writeln (' Результат записать в файл.');
    writeln (' Исходный файл');
    while not Eof(fr) do     {проверка на конец файла}
      begin
       read (fr,c);           {чтение из исходного файла}
       if c>ma then           {поиск максимального элемента}
             begin
               ma:= c;
               n:=i;
             end;
      end;
    writeln ('Полученный файл');
    write (fz,ma);   {запись максимального элемента в начало выходного
файла}
    seek(fr,1);       {установка на второй элемент исходного файла}
```

```
    write (ma,' ');
    for i:=1 to n–1 do
   begin
    read (fr,c);
    write (fz,c);    {запись в выходной файл данных из исходного начиная}
     write(c,' ');     {со второй до позиции максимального элемента}
   end;
   seek (fr,0);       {установка на первый элемент исходного файла }
   read(fr,c);        {чтение этого элемента }
   write (fz,c);      {запись этого элемента на место максимального }
   write(' ',c,'  ' );  {в выходном файле }
   seek(fr,n+1);    {установка исходного файла в позицию}
                      {следующую за позицией максимального элемента}
    for i:=n+1 to 29 do
     begin
       read(fr,c);     {чтение}
       write (fz,c);  {и запись в выходной файл оставшихся элементов}
       write (c,' ') ;
     end;
   close (fr);        {закрытие файлов}
   close (fz);
   readln;
end.
```

Пример 5.3. Сформировать файл строк с проверкой на наличие файла с задаваемым именем.

```
program kr5;
var
fz:text;
name,aaa:string;
 begin
   writeln ('Введите имя файла (в каталоге имеется файл sub81.pas) ');
   readln (name);
   assign (fz,name);
   {$I–}          { отключение проверки ввода/вывода }
   reset(fz);
   {$I+}          { включение проверки ввода/вывода }
   if IOResult = 0 then   { не было ошибки при открытии файла }
         writeln ('Файл с таким именем уже существует.');
      else
begin
  rewrite(fz);
  writeln ('Введи строку записи ');
```

```
  readln(aaa);
  writeln(fz,aaa) ;
  close(fz);
  writeln ('Ваши записи в файле с именем  ', name);
 end;
    readln;
    end.
```

Пример 5.4. Переименовать файл с проверкой на наличие исходного файла.

```
program k6;
var
 fi: file of integer;
 name, newname: string;
begin
 writeln ('Переименовать файл с проверкой на наличие исходного файла.');
 writeln ('Введите имя исходного файла ');
 writeln('( в каталоге имеется файл sub81.pas).');
   readln(name);
   assign (fi,name );
    {$I–}
   reset (fi);
    {$I+}
   if IOresult = 0 then
    begin
      writeln ('В файл с каким именем Вы хотите переименовать этот файл.');
      readln(newname);
      close(fi);
      rename (fi,newname);  { переименовка файла }
      writeln ('Файл переименован.');
    end
        else
    begin
         writeln('Файл ',name,' не существует');
         writeln('Нажмите ENTER');
    end;
 readln;
end.
```

Пример 5.5. Создать свой каталог и записать туда типизированный файл с проверкой свободного пространства на диске.

```
program k7;
uses dos;  { подключение библиотеки дополнительных подпрограмм }
```

```
var f: file of integer;
    name,namef:string;
    q:word;
    w:string;
    r:integer;
begin
   q:=1; { определяет номер диска , так  A: имеет номер = 1}
     if (disksize(q)–diskfree(q))=disksize(q) then  { disksize – объем диска }
                                    { diskfree – объем свободной части диска }
      writeln ('Нет свободного места для каталога.');
   else
     begin
       writeln('Введите имя создаваемого каталога');
       readln(name);
       getdir(q,w);       { в w записывается текущий каталог }
       w:=w+name;
       mkdir (w );        { создание своего каталога }
       writeln ('Каталог создан.');
       writeln ('Введите имя файла.');
       readln (namef);
       assign(f,'\'+name+'\'+namef);
       rewrite(f);
       writeln ('Введите эл–ты файла');
       for i:=1 to 10 do
    begin
     read (r);
     write (f , r );
    end;
    close (f);
     writeln ('Файл записан в созданный каталог.');
   end;
 end.
```

5.2. Файлы без типа

Файлы без типа предназначены для организации высокоскоростного обмена между оперативной памятью и дисковой, а также когда тип компонент файла заранее неизвестен или несущественен.

Для объявления файловой переменной без типа используется следующее описание

```
var
   f,s,t:file;
```

Далее, как и для всех других видов файлов, необходимо в операторной части программы связать файловую переменную с реальным именем файла на диске с помощью процедуры assign, например:

assign(f,'my_file.dat');

Основные отличия в использовании файлов без типа заключаются в задании процедур открытия, чтения и записи. Рассмотрим их. При открытии файлов без типа с помощью процедур reset и rewrite допускается использование дополнительного параметра, определяющего длину в байтах каждого компонента файла. То есть у программиста появляется возможность самому определять структуру файла. Например, reset(f,1) открывает для чтения файл, компонента которого равна 1 байту, а rewrite(s,20) открывает файл для записи компонент длиной в 20 байт. Если длина компонент файла не указывается, то она принимается по умолчанию равной 128 байт.

При чтении/записи данных в файл без типа также добавляется дополнительный параметр, определяющий количество компонент, которое должно быть прочитано или записано за одно обращение к диску.

Для чтения информации из файла без типа используется процедура

blockread(<файл.перем.>,<перем.1>,<кол.компонент>);

а для записи информации в файл без типа

blockwrite(<файл.перем.>,<перем2>,<кол.компонент>);

Например, для чтения целого значения типа integer в переменную X из файла без типа, открытого процедурой reset(f,1) требуется записать blockread(f,x,2), так как переменная типа integer размещается в оперативной памяти в двух байтах.

Пример 5.6. Записать в файл без типа строку символов целиком, а затем считать посимвольно и вывести вертикально.

```
program f_out_type;
 var
  t:file;                  {файл без типа}
  a:string[17];            {исходная строка}
  sym:char;
begin
 assign(t,'rz.out');
 rewrite(t,17);    {открытие на запись с длиной компоненты в 17 байт}
 a:='Надейся на лучшее';
 blockwrite(t,a,1);       {запись в файл строки}
```

```
 close(t);                {закрытие файла}
 reset(t,1);              {открытие на чтение с длиной компоненты в 1 байт}
 while not Eof(t) do
  begin
  blockread(t,sym,1);   {чтение из файла 1 компоненты в 1 байт}
  writeln(sym);
  end;
 close(t);                {закрытие файла}
end.
```

В качестве примера, когда тип компонент несущественен, рассмотрим следующую программу.

Пример 5.7. Определить имя текущего каталога и удалить в нем все файлы с расширением .BAK.

```
program k8;
uses dos,crt;         { подключение модулей }
var  s: searchrec;  { этот тип объявлен в модуле DOS, он является
                      записью в которой имеются поля name и size файла }
w:string;
b:word;
f:file;               { файл без типа }
x:char;
begin
   b:=0;              {задает текущий диск для getdir }
   getdir(b,w);       {определение имени текущего каталога в переменной w }
  writeln('Определить имя текущего каталога и удалить в нем все файлы с');
  writeln('расширением .BAK .');
  writeln('Имя текущего каталога');
  writeln(w);
  writeln('Список удаляемых файлов в текущем каталоге .');
  findfirst('*.bak',anyfile,s);   {поиск файлов *.bak}
  while doserror=0 do          {нет ошибки при поиске файлов}
    begin
      with s do
     begin
       writeln (name:12,'  ');   {печать имен файлов с расширением BAK }
       assign (f,name);
       reset (f);  {$i–}     {отключение контроля операции открытия файла}
       if ioresult=0 then  {нет ошибок при открытии файла, он существует}
        begin
         close (f);
         write('Удалить?(Y/N)');
```

```
        readln(x);
        if (x='y') or (x='Y') then
        erase(f)  {удаление файла}
       end;
     findnext(s)            {поиск следующего файла *.bak}
    end;
  end;
end.
```

Контрольные вопросы

1. Что такое файл?
2. Какие типы файлов применяются в Turbo Pascal?
3. Основные функции для работы с файлами.
4. Основные правила использования файлов в программах.

6. РАБОТА СО МНОЖЕСТВАМИ

Множества – это наборы однотипных логически связанных друг с другом объектов. Характер связи между объектами лишь подразумевается программистом и никак не контролируется Turbo Pascal. Количество элементов, входящих в множество, может меняться в пределах от 0 до 255 (множество, не содержащее элементов, называется **пустым**). Именно непостоянством своих элементов множества отличаются от массивов.

Два множества называются эквивалентными тогда и только тогда, когда все элементы их одинаковы, причем порядок следования элементов в множестве безразличен. Если все элементы одного множества входят также и в другое, говорят о включении одного множества во второе. Пустое множество включается в любое другое.

Объявление типа множества имеет вид:

< имя типа > = set of < базовый тип >

Здесь < имя типа > – правильный идентификатор;

set , of – зарезервированные слова (множество, из);

<базовый тип> – базовый тип элементов множества, в качестве которого может использоваться любой порядковый тип, кроме word, integer, longint.

Для задания множества используется так называемый **конструктор множества**: список элементов множества, отделяемый друг от друга запятыми; список обрамляется квадратными скобками [ ].

Спецификациями элементов могут быть константы или выражения базового типа, а также – тип-диапазон того же базового типа.

Над множествами определены следующие операции:

* – пересечение множеств; результат содержит элементы общие для обоих множеств;

+ – объединение множеств; результат содержит элементы первого множества, дополненные недостающими элементами из второго множества;

– – разность множеств; результат содержит элементы из первого множества, которые не принадлежат второму множеству;

= – проверка эквивалентности; возвращает TRUE, если оба множества эквивалентны;

<> – проверка неэквивалентности; возвращает TRUE, если оба множества неэквивалентны;

<= – проверка вхождения; возвращает TRUE, если первое множество включено во второе;

>= – проверка вхождения; возвращает TRUE, если второе множество включено в первое;

IN – проверка принадлежности; в этой бинарной операции первый элемент – выражение, а второй – множество одного и того же типа;

возвращает TRUE, если выражение имеет значение, принадлежащее множеству.

Пример 6.1. Составить программу, реализующую нахождение пересечения множеств [5..25], [17..34], [1..20].

```
program mnog1;
var
 a:set of 5..25;     {объявление множеств}
 b:set of 17..34;
 c:set of 1..20;
 d:set of byte;
 n:byte;
begin
   a:=[5..25];
   b:=[17..34];
   c:=[1..20];    {Задание множеств}
   d:=a*b*c;     {Нахождение пересечений множеств}
 writeln(' Пересечением является множество:');
 for n:=1 to 34 do
  if n in d then write(n:3);   {Вывод множества}
end.
```

Пример 6.2. Составить программу-определитель, какое из множеств [C..N], [J..P], [M..Z] содержит наибольшее количество элементов.

```
program mnog2;
var
   a,b,c:set of 'A'..'Z';        {объявление множеств a,b,c}
   m:char;
   ka,kb,kc:integer;          {Количество элементов в множествах}
begin
 a:=['C'..'N'];
 b:=['J'..'P'];                  {Создание множеств}
 c:=['M'..'Z'];
for m:='C' to 'N' do ka:=ka+1;
for m:='J' to 'P' do kb:=kb+1;{Определение числа элементов в множествах}
for m:='M' to 'Z' do kc:=kc+1;
       {Нахождение максимального количества элементов множества.
        Вывод результата}
 if (ka>=kb) and (ka>=kc) then
  writeln('Множество [C..N] ')
                    else
 if (kb>=ka) and (kb>=kc) then
   writeln('Множество [J..P] ')
```

```
 else
   writeln('Множество [M..Z] ');
 writeln(' содержит наибольшее количество');
end.
```

Пример 6.3. Составить программу-определитель: принадлежит ли результат вычисления (17*3–25)*2+15 множеству [7..30].

```
program mnog3;
var
   a:set of 7..30;  {Описание множества}
begin
 writeln('Результат вычисления ');
 writeln('(17*3–25)*2+15 ');
                    {Проверка на принадлежность множеству}
 if (17*3–25)*2+15 in a
       then  writeln(' принадлежит множеству [7..30].')
       else  writeln(' не принадлежит множеству [7..30].');
end.
```

Пример 6.4. Составить программу-определитель, какое из множеств [D..J] , [H..K] , [I..O] входит в множество [F..N].

```
program mnog1;
var
a,b,c,d:set of 'A'..'Z';   {объявление множеств}
begin
a:=['D'..'J'];
b:=['H'..'K'];              {Задание множеств}
c:=['I'..'O'];
d:=['F'..'N'];
if a <= d then
   write('Множество [D..J] ');
if b <= d then
   write('Множество [H..K] ');  {Сравнение множеств}
if c <= d then
   write('Множество [I..O] ');
if (a>=d) and (b>=d) and (c>=d) then
       write('никакое множество не ');
writeln('входит в ');                {Вывод результата}
writeln('множество [F..N].');
end.
```

Пример 6.5. Составить программу, реализующую нахождение суммы элементов множества [3..27].

```
program mnog5;
var
 a:set of 3..27;         {объявление множества}
 s,n:integer;
begin
 s:=0;
 for n:=3 to 27 do
              s:=s+n;    {Суммирование элементов}
 writeln('Сумма элементов множества [3..27] равна ',s,'.');
end.
```

Пример 6.6. Составить программу, реализующую создание множества с помощью датчика случайных чисел. Удалить из него все элементы кратные 3.

```
program mnog6;
var
 a:set of 1..25;          {объявление множества}
 k,n,elem:integer;
begin
 writeln('  исходное множество:');
 randomize;
 for n:=1 to 10 do
  begin
      elem:=random(25);
      a:=a+ [elem];      {Создание множества}
      write(' ',elem);
      if (elem mod 3=0) then
         begin              {Удаление из множества}
           a:=a–[elem];  {элементов кратных 3}
           k:=k+1;
         end;
  end;
 writeln;
 writeln('   полученное множество:');
 for n:=1 to 24 do                          {Вывод множества}
    if n in a then
  write(' ',n);
end.
```

Пример 6.7. Составить программу, реализующую удаление гласных букв из множеств [D..K], [O..T], [V..Z]. Результат записать в виде одного множества.

```
program mnog7;
var
  a,b,c,glas,res:set of 'A'..'Z';    {объявление множеств}
  m:char;
begin
 a:=['D'..'K']; b:=['O'..'T'];              {Задание множеств}
 c:=['V'..'Z']; glas:=['A','E','I','O','U'];
 for m:='D' to 'K' do
   if m in glas then
    a:=a–[m];
 for m:='O' to 'T' do              {Проверка на наличие в }
   if m in glas then
    b:=b–[m];
 for m:='V' to 'Z' do              {множествах гласных букв}
    if m in glas then
     b:=b–[m];
 res:=a+b+c;                       {Суммирование множеств}
 for m:='A' to 'Z' do               {Вывод результата}
     if m in res then
      write(' ',m);
end.
```

Пример 6.8. Программа выводит на экран слова, содержащие буквы последней строки входного файла.

```
program lab7;
var
  s:char;
  b:set of 'A'..'я';
  sl,str:string;
  fin:text;
  i:integer;
  j: 'A'..'я';
  pr:boolean;
begin
 writeln('Вывести на экран слова, содержащие буквы слов последней');
 writeln('строки входного файла INPUT.DAT');
 assign(fin,'input.dat');
 reset(fin);
 while not eof (fin) do
   begin
```

```
      readln(fin,str);
      writeln(str);
     end;
  close(fin);
  b:=[ ];                          {формирование множества букв последней строки}
  for i:=1 to  length (str)  do
   begin
     s:=str[i];                              {выделение буквы}
     if not( s in b) then b:=b+[s];  {формирование множества}
   end;
  writeln ('Множество букв последней строки:');
  for j:='A' to 'я' do    if j in b then
   writeln (j:4);
   reset (fin);     {печать множества букв последней строки в выходной
файл}
   writeln ('Слова, содержащие буквы последней строки:');
   while not eof (fin) do
    begin
      readln(fin,str);
      i:=1;
     while i <= length (str) do
     begin
       begin
         sl:=' ';
         pr:=false;
         {формирование очередного слова и проверка его букв}
         while   not (Str[i] IN [' ',',','–',':','.'])  do
          begin
            if str[i] in b then
            pr:=true;
            sl:=sl+str[i];
            i:=i+1;
          end;
        if pr then writeln(sl);
           repeat            {пропуск пробелов}
             i:=i+1;
           until(str[i]<>' ');
     end;
    end;
  end;
 close (fin);
end.
```

Задания для самостоятельного выполнения

Входной информацией является произвольный текстовый файл, число строк в котором более 2-х. **Словом** считается любая последовательность подряд идущих символов. Считается, что слова разделяются пробелами. Реализовать поставленную задачу необходимо на основе использования множества.

1. Найти и вывести все гласные буквы (без повторений), которые встретились в словах и количество слов.
2. Найти и вывести все шипящие буквы (без повторений), которые встретились в самом длинном слове.
3. Найти и вывести все шипящие буквы, которые встретились во всех словах? и количество слов.
4. Найти и вывести все гласные буквы (без повторений), которые встречаются в самом коротком слове.
5. Элементами слов могут быть как буквы, так и цифры. Записать в файл слово, содержащее наибольшее количество цифр.
6. Элементами слов могут быть как буквы, так и цифры. Вывести на экран слово, содержащее наибольшее количество букв.
7. Элементами слов могут быть как буквы, так и цифры. Вывести на экран слово, содержащее наибольшее количество нечетных цифр.
8. Элементами слов могут быть как буквы, так и цифры. Вывести на экран слово, содержащее наибольшее количество четных цифр.
9. Элементами слов могут быть как буквы, так и цифры. Вывести все четные цифры (без повторений), которые содержатся во всех словах и количество слов.
10. Элементами слов могут быть как буквы, так и цифры. Вывести все нечетные цифры (без повторения), которые встречаются во всех словах и количество слов.
11. Элементами слов могут быть как буквы, так и цифры. Подсчитать количество нечетных цифр, содержащихся в самом длинном слове.
12. Элементами слов могут быть как буквы, так и цифры. Все цифры, входящие в самое длинное слово, заменить на символ «*».
13. Элементами слов могут быть любые символы. Найти и вывести слово, содержащее наибольшее количество согласных букв.
14. Найти и вывести слово, содержащее наибольшее количество гласных букв.
15. В слове, в котором обнаружено наибольшее количество шипящих букв, заменить их на символ «*».
16. Вывести все гласные буквы, содержащиеся в слове наибольшей длины, и вывести число повторений каждой этой буквы.

17. Слова могут содержать любые символы языка. Подсчитать количество слов и количество символов во всех словах, отличных от заглавных латинских букв.
18. Вывести все согласные буквы, содержащиеся в слове наибольшей длины, и вывести число повторений каждой буквы.
19. Слова могут содержать любые символы языка. Найти и вывести слово, содержащее наибольшее количество символов, отличных от заглавных букв.
20. Слова могут содержать любые символы языка. Найти и вывести в самом длинном слове все символы, отличные от заглавных латинских букв.
21. Слова могут состоять из букв и цифр. В самом коротком слове каждую входящую в него цифру заменить на символ «&».
22. Слова могут состоять из букв и цифр. Записать в файл слово, содержащее наибольшее количество четных цифр.

Контрольные вопросы

1. Что называется множеством?
2. Какое множество называется пустым?
3. Как описывается множественный тип?
4. Какие операции допустимы к данным типа множества?
5. Сколько элементов может содержать множество?
6. Значения каких типов может содержать множество?

7. ОБРАБОТКА МАССИВОВ

Под структурой данных типа **массив** понимают набор упорядоченных однотипных данных. Упорядоченность данных в массиве позволяет обращаться к любому элементу массива по его номеру (индексу), а однотипность данных определяет возможность использования циклической обработки для всех элементов массива.

При программировании задач обработки массивов данных на Turbo Pascal необходимо знать:

1) каким образом определять переменные типа массив;

2) правила работы с элементами массивов;

3) алгоритмы обработки массивов.

Для определения данных типа массив в Turbo Pascal можно использовать следующие структуры:

```
 type mas=array[1..5] of real;  {одномерный массив из 5 }
   var mymas:mas;                { вещественных элементов }
{————————————————————————————————}
   var
   my2mas:array[1..10] of mas;          { двумерный массив 10*5 вещ. }
   mymas2:array[1..10] of array[1..5] of real;     { элементов }
   mymas3:array[1..10,1..5] of real;
{————————————————————————————————}
```

В Turbo Pascal можно объявить (описать) массив любой размерности.

Ограничения, которые имеются при объявлении массивов:

1) границы изменения индексов (размерности) необходимо определить либо в описании массива, либо до его объявления;

2) максимальный объем памяти, выделенной под массив, не должен превышать 64кБ.

Например:

```
 const
       N=5;  M=4;
 var
       a:array[1..N,1..M] of  real;
const n=7
var a:array[1..n] of byte;
```

Таким образом, общая конструкция для определения структуры данных типа массив в программе следующая:

type <имя типа>=**array**[<тип индексов>] **of**<тип элементов>;

где

array, of – служебные слова, [,] – служебные символы.

В качестве <типа индексов> может использоваться любой простой тип Turbo Pascal, кроме real и string.

В качестве <типа символов> может использоваться любой простой или сложный тип Turbo Pascal.

Для обращения к элементу массива требуется указать имя переменной типа массив и номер элемента в квадратных скобках.

В исполняемой части программы один массив может быть присвоен другому, если типы их идентичны:

```
type
  mas=array[1..10] of byte;
var
 a,d:mas;
 ..............
 begin
   a:=d;
   ..........
 end.
```

7.1. Одномерные массивы

В информатике и математике массив называется **одномерным**, если для получения доступа к его элементам достаточно одной индексной переменной.

Вы можете объявить одномерный массив следующим образом:

```
var
a : array [1..5] of byte;
b : array [1..3] of char;
z : array ['d'..'g'] of integer;
```

Величины, соответствующие начальному и конечному индексам, т. е. значения, указанные в квадратных скобках, разделяются двумя точками.

7.1.1. Ввод элементов одномерного массива

1) С клавиатуры

```
for i:=1 to n do
read(a[i]);
```

2) С помощью типизированных констант

```
const
a1 : array [1..6] of integer = (–5,8,5,0,7,–8);
a2 : array [1..3] of char = ('a','b','c');
```

3) С помощью датчика случайных чисел

```
for i:=1 to 10 do
begin  a[i]:=random (20);
write(a[i]:5);
end;
```

В этом случае значениями элементов массива a[i] будут произвольные значения от 0 до 19. Для того чтобы получились дробные числа, нужно в функции random опустить параметр.

4) Из файла

```
var
  a:array [1..10] of integer ;
  i:byte ;
  c:integer ;
  f: file of integer ; {объявление типизированного файла целых чисел}
begin
  assign (f,'mkurs.dat'); {связывание файловой переменной с именем файла}
  reset (f);{открытие файла для чтения}
  writeln ('Полученный массив ');
  for i:=1 to 5 do
   begin
    read( f,a[i]); {чтение из файла}
    write (a[i],' ');
   end;
  close(f);
end.
```

7.1.2. Вывод элементов одномерного массива

1) Вывод элементов массива в таблицу на экране

```
write('Массив');
writeln('————————————————');
writeln('N |  значение |');
for i:=1 to 10 do
writeln(i,'|',a[i]:9,' |');
```

2) Вывод в файл

```
var
   a:array [1..5] of integer;
   i:byte;
   c:integer;
   f: file of integer;{объявление типизированного файла целых чисел}
begin
 assign (f,'mkurs.dat');{связывание файловой переменной с именем файла}
 reset (f);{открытие файла для чтения}
 writeln ('Введите массив ');
   for i:=1 to 5 do
      begin
         read(a[i]);
         write ( f,a[i]);{запись в файл}
      end;
  close (f);{закрытие файла}
end.
```

7.1.3. Основные алгоритмы работы с одномерными массивами

1) Присваивание массива

Одномерные массивы могут присваивать свое значение целиком, а не только поэлементно, если их типы эквивалентны:

```
var
  a,b:array[1..7]of real;
  i:byte;
begin
  write("Ввод массива");
  for i:=1 to 7 do
   read(a[i]);
  b:=a;
  for i:=1 to 7 do
   write(b[i]);
end.
```

Рассмотрим присваивание одномерных массивов, которые являются элементами двумерного массива:

```
type
  a:array[1..2]of real;
var
  b:array[1..3]of a;  { двумерный массив }
  i,j:byte;c:a;
```

```
begin
  writeln('перестановка строк');
  for i:=1 to 3 do
  for j:=1 to 2 do
    read(b[i,j]);
  c:=b[1];      { перестановка }
  b[1]:=b[3];
  b[3]:=c;
  for i:=1 to 3 do
  for j:=1 to 2 do
     write(b[i,j]);
end.
```

Замена или перестановка элементов массива с порядковыми номерами n и m:

```
var
  a:array[1..4]of real;
  p:real;
  i,n,m:integer;
begin
  read(n,m);
  for i:=1 to 4 do
       read(a[i]);
  p:=a[n];       { перестановка }
  a[n]:=a[m];
  a[m]:=p;
  for i:=1 to 4 do
            write(a[i]);
end.
```

2) Удаление из массива одного элемента с номером k.

Алгоритм:

a) ввод элементов массива;

b) удаление, которое осуществляется следующим образом:

```
for i:=k to  n–1 do     a[i]:=a[i+1];
```

a[i]:=0; {n – количество элементов массива, k – номер удаляемого элемента};

c) перенос в хвост массива из 10 элементов элемент с k-й позиции.

3) Перенос в конец одномерного массива элемента с k-й позиции.

Алгоритм:

a) ввод элементов массива;

b) сохранение элемента с k-й позиции в дополнительной переменной:

```
 2  (3)  4  5  6
      p———————¦
```
c) сдвиг элементов массива следующих за k-м на одну позицию влево;
d) восстановление последнего значения из переменной p.

```
  read(k);
{a} for i:=1 to n do read (a[i]);
{b}   p:=a[k];
{c} for i:=k  to n–1 do
       a[i]:=a[i+1];
{d} a[n]:=p;
{вывод a}.
```

4) Перенос в начало одномерного массива элемента с k-й позиции.

Алгоритм:

a) ввод элементов массива;
b) сохранение элемента с k-й позиции в дополнительной переменной p;
c) сдвиг всех элементов до элемента с номером k вправо;
d) записать на первое место значение из переменной p.

```
      read(k);
   for i:=1 to n do read (a[i]);        2 3 {4} 5 6
   p:=a[k];                                  |
   for i:=k downto 2 do                ———p
   a[i]:=a[i–1];                       2 2   3  5 6
     a[1]:=p;
   {вывод a}
```

Пример 7.1. Вычислить сумму элементов массива.

```
program a8;
var
 a:array[1..10]of real;
 i:integer;
 c:real;
begin
 c:=0;
 writeln('Вычислить сумму элементов массива');
 writeln('    Введите 10 элементов массива');
 for i:=1 to 10 do
  begin
  write(i,'   элемент= ');
  readln(a[i]);
  c:=c+a[i];
  end;
writeln('Сумма элементов массива = ',c);
```

```
writeln('Для окончания работы нажмите enter');
readln;
readln;
end.
```

Пример 7.2. В одномерном массиве найти сумму положительных, произведение отрицательных, количество четных элементов.

```
program a6;
var
  a:array[1..10] of integer;
  e,i,c,d:integer;
begin
   writeln('      В одномерном массиве найти сумму положительных,');
   writeln(' произведение отрицательных, кол-во четных элементов');
   writeln;
   writeln('Введите массив из 10 элементов через пробелы');
  d:=0;
  c:=1;
  e:=0;
  for i:=1 to 10 do
      begin
      read (a[i]);
     if a[i]>0 then  d:=d+a[i] ;
     if a[i]<0 then c:=c*a[i]  ;
     if a[i] mod 2 =0 then  e:=e+1 ;
     end;
  writeln;
  writeln('Результат:');
  if d<>0 then writeln('Сумма положительных элементов d= ',d)
          else  writeln('Положительных нет.');
  if c<>1 then writeln('Произведение отрицательных элементов c= ',c)
          else writeln('Отрицательных нет.');
  if e<>0 then writeln('Кол–во четных элементов e= ',e)
          else  writeln('Четных нет.');
  readln;
  readln;
end.
```

Пример 7.3. Найти минимальное значение элемента массива и его порядковый номер.

```
program a3;
const
 nmax=10; {количество элементов в массиве}
```

```
var
 imin,i:integer;
 x:array[1..nmax]of integer;
 begin
  writeln('Найти минимальное значение элемента массива');
  writeln('и его порядковый номер');
  writeln('Введите 10 элементов массива');
  imin:=1;      {номер минимального элемента}
  for i:=1 to nmax do
   begin
   read(x[i]);
   if x[i]<x[imin] then
   imin:=i;
   end;
writeln('Минимальное значение=',x[imin]);
writeln('Порядковый номер =',imin);
writeln('Для окончания работы нажмите enter');
readln;
end.
```

Пример 7.4. Найти минимальное значение элемента массива на нечетной позиции.

```
program a5;
var
 a:array[1..10]of integer;
 ind,i,min:integer;
begin
writeln('Найти минимальное значение элемента ');
writeln('массива на нечетной позиции');
writeln('Введите 10 чисел');
ind:=1;       {номер минимального элемента}
for i:=1 to 10 do
begin
read(a[i]);
if (i mod 2)=1 then
if a[i]<a[ind] then   ind:=i;
end;
writeln('Минимальное значение=',a[ind]);
writeln('Для окончания работы нажмите enter');
readln;
end.
```

Пример 7.5. Найти максимальное значение и его порядковый номер.

```
program a7;
const nmin=10;
var
 imax,i:integer;
 x:array[1..nmin]of integer;
begin
 writeln('Найти максимальное значение и его порядковый номер');
 writeln('Введите 10 элементов массива через пробел');
 imax:=1;
for i:=1 to nmin do
begin
read(x[i]);
if x[i]>x[imax] then  imax:=i;
end;
 writeln('Максимальное значение=',x[imax]);
 writeln('Порядковый номер =',imax);
 writeln('Для окончания работы нажмите enter');
readln;
end.
```

Пример 7.6. Программа, меняющая в массиве (5 –2 1 13 7) максимальное и минимальное числа.

```
program  iv44;
const n=5;
type
   mass=array[1..n]of integer;
const
   arr:mass=(5,–2,1,13,7);{ввод массива типизированной константой}
var
i,x,max,min:integer;
begin
writeln('Программа, меняющая в массиве   5  –2  1  13  7');
writeln('максимальное и минимальное числа.');
 max:=1;   {номер максимального}
 min:=1;   {номер минимального}
for i:=1 to n do
 begin
   if arr[i]>arr[max] then
      max:=i;
   if arr[i]<arr[min] then
      min:=i;
 end;
```

```
{перестановка}
 x:=arr[max];
 arr[max]:=arr[min];
 arr[min]:=x;
 writeln ('Измененный массив');
 for i:=1 to n do
      writeln('       ',arr[i]{:4:2});
 readln;
end.
```

Пример 7.7. В одномерном массиве найти третий нечетный элемент.

```
program a4;
var
  a:array[1..10] of integer;
  k,i:integer;
begin
  writeln('В одномерном массиве найти третий нечетный эл-т');
  writeln('Введите массив из 10 эл-в через пробелы');
  k:=0;  {количество нечетных элементов}
  for i:=1 to 10 do
  read(a[i]);
  for i:=1 to 10 do
  begin
   if (a[i] mod 2<>0) then
    begin
     k:=k+1;
     if  k=3 then
       begin
         writeln('Результат ',a[i]) ;
         exit ;  { выход из программы }
       end;
    end;
  end;
  if k=0 then writeln('нечетных нет');
readln;
end.
```

Пример 7.8. Поиск первого отрицательного, второго положительного и предпоследнего нечетного.

```
program a8;
var
 a:array[1..10]of integer;
 i:integer;
```

```
 c,otr,otr1,pol,pol1,n:integer;
begin
 c:=0;
 n:=0;
 otr:=0;
 otr1:=0;
 pol:=0;
 pol1:=0;
 writeln('Поиск первого отрицательного, второго ');
 writeln('положительного и предпоследнего нечетного.');
 writeln('Введите 10 элементов целочисленного массива');
 for i:=1 to 10 do
  begin
    write(i,'   элемент= ');
    readln(a[i]);
    if a[i]<0 then
     begin
       otr:=otr+1;
       if otr=1 then
                  otr1:=a[i];
     end;
    if a[i]>0 then
     begin
       pol:=pol+1;
       if pol=2 then
                   pol1:= a[i];
     end;
    if a[i] mod 2 =1 then
     begin
        c:=n;
        n:=a[i];
     end;
  end;
 if otr = 0 then writeln('Отрицательных элементов массива нет ')
          else writeln('Первый отрицательный элемент = ', otr1);
 if pol <2 then writeln('Положительных элементов массива нет')
          else writeln('Второй положительный элемент массива = ', pol1);
 if c = 0 then writeln('В массиве нет двух нечетных элементов')
         else writeln('Предпоследний нечетный элемент = ',c);
 writeln('Для окончания работы нажмите enter');
 readln;
end.
```

Пример 7.9. В одномерном массиве В найти минимальный элемент и заменить его предпоследним элементом кратным 3, который находится в массиве А.

```
program iv55;
var
  a,b:array[1..5] of integer;
  i:byte;
  p,n,ind,flag:integer;
begin
writeln('В одномерном массиве В найти минимальный элемент  и');
writeln(' заменить его предпоследним элементом кратным 3 ,’);
writeln(' который находится в массиве А.');
writeln('Измененный массив В  вывести на экран .');
write('Ввести эл–ты массива А',' ','Введите эл–ты массива В',' ', 'Результат');
repeat
 flag:=0;
  for i:=1 to 5 do
   begin
     read( a[i]);
     if (a[i] mod 3) =0 then     {поиск эл–та по условию кратности}
      begin
        n:=p;
        flag:=1;
        p:=i;
      end;
   end;
   if flag=0 then
          write('В введенном массиве нет эл–тов, кратных 3.');
until flag<>0;
write(' Предпоследний элемент кратный 3 – ',  a[n],' ,его номер  ',
write( n ,  {n– номер  предпоследнего эл–а} );
ind:=1;
for i:=1 to 5 do
  begin
   read(b[i]);
   if (b[i])<(b[ind]) then     {цикл поиска эл–та по условию min}
   ind:=i;
  end;
   write('MIN= ' , b[ind] );
   b[ind]:=a[n];
   for i:=1 to 5 do  writeln(b[i]);
   readln;
end.
```

Пример 7.10. В одномерном массиве перенести элемент, стоящий на первом месте в конец массива.

```
program a7;
var
 p,i:integer;
 a:array[1..10] of  integer;
begin
 writeln('    В одномерном массиве перенести элемент,');
 writeln('   стоящий на первом месте в конец массива,');
 writeln('Введите 10 элементов массива через пробелы');
 for i:=1 to 10 do
  read(a[i]);
 p:=a[1];  { перенос }
 for i:=1 to 9 do
  a[i]:=a[i+1];
 a[10]:=p;
 writeln('Результат');
 for i:=1 to 10 do
  write(a[i],' ');
 readln;
 readln;
end.
```

Пример 7.11. В одномерном массиве найти минимальное значение и удалить его.

```
program a7;
var
amin:integer;
imin,i:integer;
a:array[1..10]of integer;
begin
  writeln('Найти минимальное значение и удалить его ');
  writeln('Введите 10 элементов массива через пробел');
  for i:=1 to 10 do
  read(a[i]);
  amin:=a[1];
  imin:=1;
  for i:=2 to 10 do
   begin
     if a[i]<amin then
        begin  {поиск min}
          amin:=a[i];
          imin:=i;
```

```
      end;
   end;
for i:=imin to 9 do   { удаление }
a[i]:=a[i+1];
a[10]:=0;
writeln('Результат');
for i:=1 to 9 do
      write(a[i],' ') ;
readln;
readln;
end.
```

Пример 7.12. В одномерном массиве перенести второй нулевой элемент в начало массива.

```
program a1;
var
  a:array[1..10] of integer;
  k,i,n,p:integer;
begin
  writeln('    В одномерном массиве из 10 элементов перенести');
  writeln('     второй нулевой элемент в начало массива');
  k:=0;
  writeln;
  writeln('      Введите исходный массив хотя бы с ');
  writeln(' двумя нулевыми элементами через пробелы');
  for i:=1 to 10 do
   begin
     read(a[i]);
     if a[i]=0 then
      begin
        k:=k+1;
        if (k=2) then n:=i ;
      end;
   end;
 p:=a[n];
 for i:=n downto 2 do  { цикл переноса в начало }
 a[i]:=a[i–1];
 a[1]:=p;
  writeln('Результат');
  for i:=1 to 10 do
    begin
       write(a[i]:2);
    end;
```

```
writeln;
readln;
readln;
end.
```

Пример 7.13. В одномерном массиве найти группу из цифр <5>.

```
program a2;
var
  a:array[1..10] of integer;
  n1,n2,i,k:integer;
begin
  writeln;
  writeln('В одномерном массиве найти группу из цифр <5>  ');
  writeln;
  k:=0;
  n1:=0;
  writeln('  Введите массив из 10 элементов через пробелы');
   for i:=1 to 10 do
    begin
      read(a[i]);
      if a[i]=5 then
       begin
         if k=0 then
                  n1:=i;
         k:=k+1;
         n2:=n1+(k–1);
       end;
    end;
    if k<>0 then
     begin
       writeln('Начало группы:',n1);
       writeln('Конец группы:',n2);
       writeln('Кол–во элементов в группе: ',k);
     end;
     if k=0 then
              write('Такой группы нет');
  readln;
  readln;
end.
```

Пример 7.14. В одномерном массиве найти максимальную группу, содержащую нули.

```
program a5;
```

```
var
  a:array[1..10] of integer;
  n1,n2,i,k,p:integer;
 begin
writeln(' В одномерном массиве найти максимальную  ');
writeln(' группу , содержащую нули ');
  writeln('       Введите массив из 10 элементов через пробелы');
 k:=0;
 n1:=0;
 n2:=0;
 p:=1;
 for  i:=1 to 10 do
   begin
    read(a[i]);
    if a[i]=0 then
       begin
         if k=0 then
                  n1:=i;
         k:=k+1;
       end
    else
      begin
        if k>p then
         begin
           p:=k;
           n2:=n1;
          end;
         k:=0;
      end;
  end;
 if n2=0 then
           write('Такой группы нет')
 else
 writeln('Начало максим группы: ',n2);
 writeln('Кол–во элементов в максим группе: ',p);
 readln;
 readln;
end.
```

Пример 7.15. Записать в файл 10 элементов одномерного массива в следующем порядке:
– пять элементов массива, начиная с десятой позиции;
– пять элементов, начиная с первой позиции массива.

```
program kr1;
var a:array [1..10] of integer;
    i:byte;
    c:integer;
    f: file of integer;{объявление типизированного файла целых чисел}
begin
  assign (f,'mkurs.dat');{связывание файловой переменной с именем файла}
  reset (f);{открытие файла для чтения}
  for i:=1 to 30 do
   begin
     read(f,c);{чтение\ввод данных из файла}
     write(c,' ');
   end;
     writeln ('Полученный массив ');
     seek (f,9);{установка на 10 запись в файле}
   for i:=1 to 5 do
     begin
       read( f,a[i]);{чтение из файла}
       write (a[i],' ');
     end;
   seek (f,0);  {установка на начало файла}
   for i:=6 to 10 do
     begin
       read(f,a[i]);{чтение из файла}
       write (a[i],' ');
      end;
   close (f); {завершить работу с файлом – закрыть файл}
 readln;
end.
```

Пример 7.16. Ввод из файла 10 чисел в одномерный массив в следующем порядке:
– сначала все отрицательные числа файла с 12 по 20 позиции;
– затем все, начиная с первой позиции файла.

```
program kr2;
var a: array [1..10] of integer;
    i,j:byte;
    c:integer;
    f: file of integer;      {объявление файловой переменной}
begin
 assign (f,'mkurs.dat');{связывание файловой переменной с именем файла}
 reset (f);              {открытие файла}
 writeln(' Исходный файл ');
```

```
 while not Eof(f) do    {проверка на конец файла}
   begin
     read(f,c);          {считывание из файла всех чисел}
     write(c:4);
   end;
 seek (f,11);          {установка файла на 12 позицию}
 writeln ('полученный массив A ');
 for i:=1 to 10 do
  begin
    read(f,c);      {чтение из файла}
    if c<=0 then
      begin
        a[i]:=c;
        write (a[i],' ');
      end;
  if filepos(f)=20 then
   begin
     j:=i;
     i:=10;
   end;
 end;
  seek(f,0);       {установка в начало файла}
  for i:=j–3 to 10 do
    begin
      read (f,a[i]);  {чтение из файла}
      write (a[i],' ');
    end;
  readln;
 close (f);    {закрытие файла}
end.
```

Задания для самостоятельного выполнения

1. Дан массив целых чисел a1,...,an. Сформировать массив B из положительных элементов массива A.
2. Дан массив действительных чисел a1,...,an. Получить количество отрицательных элементов массива и произведение элементов, принадлежащих отрезку [c,в].
3. Дан массив целых чисел a1,...,an. Найти в нем все пары ai,ai+1, такие, что ai<=3 и ai+1<0. Распечатать их значения, номера, если таких пар нет, то выдать сообщение.
4. Дан массив целых чисел a1,...,an. Найти в данной последовательности все пары ai,ai+1, такие, что ai=0 и ai+2 кратно 2.

5. Даны действительные числа a1...a16. Получить max (a1+a16,a2+a15,..., a8+a9).
6. Даны целые числа a1,…,an. Все члены массива a1,...,an, предшествующие наименьшему числу, домножить на это число.
7. Дан массив символов s1,…,sn. Подсчитать, сколько раз встречается в этой последовательности символ К.
8. Дан массив символов S1,...,Sn. Удалить из него все буквы, непосредственно перед которыми находится буква С.
9. Дан массив символов S1,...,Sn. Напечатать true, если в заданном массиве буква a встречается чаще, чем буква b, и напечатать false в противоположном случае.
10. Даны действительные числа a1,…,a16. Получить min(a1*a9, a2*a10,..., a8*a16).
11. Дан массив действительных чисел a1,...,an. Выяснить, верно ли, что наибольший элемент массива a1,...,an по модулю больше единицы.
12. Дан массив целых чисел a1,...,an. Найти максимальный элемент массива и поменять его местами с первым элементом.
13. Дан массив целых чисел a1,...,an. Найти минимальный элемент массива и поменять его местами с последним элементом.
14. Дан массив действительных чисел a1,...,an. Выяснить, имеются ли в данном массиве 2 идущих подряд положительных элемента. Подсчитать количество таких пар.
15. Дан массив действительных чисел a1,...,an. Заменить все элементы массива, находящиеся в интервале от c1 до c2 на минимальный. Исходный и скорректированный массивы напечатать.
16. Дан массив действительных чисел a1,...,an. Если среди элементов массива есть хотя бы одно число больше 100, то все элементы массива поделить на 100. Исходный и скорректированный массивы напечатать.
17. Даны целые числа a1,...,an. Определить количество целых чисел, входящих в последовательность a1,...,an по одному разу.
18. Даны целые числа a1,…,an. Из модулей членов данной последовательности выбрать наибольший. Получить новую последовательность из n целых чисел, заменяя ai нулем, если /ai/ не совпадает с выбранным значением, и заменяя ai единицей, если совпадает.
19. Перенести в хвост одномерного массива максимальный элемент.
20. Даны действительные числа a1,...,an. Требуется найти B равное среднему арифметическому чисел a1,…,an, и наибольшее отклонение от среднего, т. е. max(/a1–b/,/a2–b/,../an–b/).
21. Даны 2 последовательности действительных чисел x1,…,xn и y1,…,yn. Выяснить, верно ли, что среди точек (xi,y1) есть хотя бы одна, принадлежащая квадрату, стороны которого параллельны координатным осям, центр совпадает с началом координат, а длина стороны равна 2. Вывести координаты точек, удовлетворяющих этому условию.

22. Даны 2 последовательности действительных чисел x1,..,x10, e1,..,e10. Выяснить, верно ли, что все точки (xi,yi) принадлежат кругу радиуса 2 с центром в точке (1,1).

УКАЗАНИЕ: воспользуйтесь формулой R*R=x*x+y*y для круга с центром в точке (0,0).

23. Дан массив целых чисел a1,..,an. Выяснить, какая из трех ситуаций имеет место: все числа a1,..,an равны нулю, в последовательности a1,...,an первое ненулевое число – положительное, первое ненулевое число – отрицательное.

24. Перенести в хвост одномерного массива первый отрицательный элемент.

25. Подсчитать в одномерном массиве максимальное количество подряд идущих единиц.

26. Перенести в начало одномерного массива второй нулевой элемент.

27. Подсчитать в одномерном массиве количество серий подряд идущих единиц.

28. Исключить из массива A1..AN первую серию отрицательных элементов.

29. Перенести в хвост одномерного массива все отрицательные элементы.

30. Перенести в начало одномерного массива все нечетные элементы.

31. Исключить из массива A1..AN первый четный элемент, следующий за максимальным.

32. Исключить из массива слов первое слово, заканчивающееся на букву А.

33. Дан массив действительных чисел a1,...,an. Найти максимальный элемент среди отрицательных элементов и поменять его местами с минимальным положительным.

34. Дан массив слов a1,...,an . Найти предпоследнее слово с максимальной длиной и напечатать его значение и индекс.

35. Даны 2 массива целых чисел x1,...,xn, y1,...,yn.

Получить новый массив по следующему правилу:

$$zi = \frac{xi+yi}{yi}$$

36. Дан массив целых чисел a1,...,an. Сформировать массив B по следующему правилу:

$$Bi = \begin{cases} 2(ai+i), & \text{если } a>p \\ 2(ai-i), & \text{если } a<p \\ 0, & \text{если } a=p \end{cases}$$

Значение константы p задается самостоятельно.

7.2. Двумерный массив

Описание двумерных массивов осуществляется одним из следующих способов:

а) с помощью типизированной переменной:

```
type
 mar=array[1..5] of byte;
var
  a:array[1..6] of mar;
```

б) в разделе объявления переменных:

```
var
  m:array[1..5] of array [1..8] of string;
  a:array[1..5,1..4] of integer;
```

7.2.1. Ввод элементов двумерного массива

1) С клавиатуры.

```
for i:=1 to n do
for j:=1 to m do
read(a[i,j]);
```

2) С помощью типизированной константы.

```
const
  c:array[1..3,1..5] of byte = ((0,1,2,3,4),
                                (5,6,7,8,9),
                                (2,1,0,3,4));
```

3) С помощью датчика случайных чисел.

```
randomize;                  { запуск генератора случайных чисел}
  for i:=1 to 5 do
  for j:=1 to 7 do
  a[i,j]:=random(20);
```

В этом случае элементами массива будут числа от 0–19.

4) Ввод элементов из файла.

```
var  f:file of integer;
     a:array[1..5,1..7] of integer;
begin
```

```
assign(f,'srsdf.txt');
reset(f);
for i:=1 to 5 do
for j:=1 to 7 do
read(f,a[i,j]);
close(f);
```

7.2.2. Вывод элементов двумерного массива

1) В строку.

```
for i:=1 to 5 do
for j:=1 to 7 do
write(a[i,j],’  ‘);
```

2) В матричной форме.

```
for i:=1 to 5 do
  begin
     for j:=1 to 7 do
     write(a[i,j]:3);
  writeln;
  end;
```

3) В файл.

```
var
     f:file of integer;
     a:array[1..5,1..7] of integer;
begin
     assign(f,'sdfg.txt');
     rewrite(f);
     for i:=1 to 5 do
     for j:=1 to 7 do
     write(f,a[i,j]);
     close(f);
```

7.2.3. Основные алгоритмы работы с двумерными массивами

В нижеприведенных задачах примем условные обозначения:

<read> – ввод элементов массива одним из вышеприведенных способов;

<write> – вывод элементов массива одним из вышеприведенных способов.

Пример 7.17. Подсчитать сумму положительных элементов.

```
 var
  a:array[1..5,1..7] of integer;
  i,j:byte;
  k:integer;
   begin
     <read>;
     k:=0;
     for i:=1 to 5 do
     for j:=1 to 7 do
     k:=k+a[i,j];
   writeln(k);
end.
```

Пример 7.18. Определить произведение элементов в каждой строке.

```
var
  a:array[1..4,1..4] of real;
  i,j:byte;
  p:real;
begin
 for i:=1 to 4 do
   begin
    p:=1; { начальное значение произведения }
    for j:=1 to 4 do
      begin
        read(a[i,j]);
        p:=p*a[i,j];      { вычисление  произведения}
      end;
    write(p);
   end;
  end.
```

Пример 7.19. В двумерном массиве подсчитать сумму элементов в каждом столбце.

```
program lupa7;
const
  n=3;
  m=3;
type arr=array[1..n,1..m] of integer;
var
```

```
  a:arr;
  i,j,p:integer;
begin
  writeln('Задание:     в двумерном массиве подсчитать');
  writeln('                сумму элементов в каждом');
  writeln('                        столбце.        ');
  writeln('исходный массив  3*3');
  for i:=1 to n do
  for j:=1 to m do
       read(a[i,j]);
  writeln ('  Результат:');
  for j:=1 to n do
    begin
      p:=0;
      for i:=1 to n do
          p:=p+a[i,j];
      writeln (j,' стобец: сумма=', p);
    end;
  readln;
end.
```

Рассмотрим задачи, в которых требуется выполнить поэлементную обработку некоторых компонент массива, удовлетворяющих определенному условию:

а) только положительных, отрицательных, кратных заданному числу;

б) в четных или нечетных столбцах или строках.

Для проверки условий можно использовать следующие условные операторы:

```
if (i mod 2=0) and (j mod 2=0) then ...{четность}
if (i=j)....      на главной диагонали
if (i<j)....      выше главной диагонали
if (i>j)....      ниже главной диагонали
if (i+j=N+1)...   на побочной  диагонали
if (i+j<N+1)...   выше побочной диагонали
if (i+j>N+1)...   ниже побочной диагонали
```

Пример 7.20. Дана квадратная матрица. Подсчитать количество положительных элементов, лежащих выше главной диагонали.

```
program mas2;
var
  a:array[1..5,1..5] of integer;
  i,j:byte;
  k:integer;
```

```
  f:text;
begin
  assign(f,'fmas2.txt');
  reset(f);
writeln;
writeln('Дана квадратная матрица.Подсчитать кол–во положительных');
writeln('элементов,лежащих выше главной диагонали.');
  writeln('Данная матрица');
   for i:=1 to 5 do
   for j:=1 to 5 do
          read(f,a[i,j]);  {считывание массива из файла}
   for i:=1 to 5 do      { вывод на экран матрицы }
     begin
        for j:=1 to 5 do
           write(a[i,j]:3);
        writeln;
     end;
   k:=0;
 for i:=1 to 5 do
   begin
      for j:=1 to 5 do
      if (i<j) and (a[i,j]>0) then {поиск количества элементов по условию}
      k:=k+1;
   end;
 writeln('Результат');
 writeln(k);    {вывод результата}
end.
```

Пример 7.21. Найти сумму четных элементов, расположенных ниже побочной диагонали

```
program mas3;
const     a:array[1..5,1..5] of integer=((0,1,2,3,4),
                                         (5,6,7,8,9),
                                         (0,1,2,3,4),
                                         (5,6,7,8,9),
                                         (0,1,2,3,4));
var
i,j:byte;
k:integer;
begin
 writeln('Дана матрица:');
 for i:=1 to 5 do
  begin
```

```
   for j:=1 to 5 do
    begin
     if  (i+j>6) and (a[i,j] mod 2=0) then
                                        k:=k+a[i,j];
     write(a[i,j]:3);
    end;
   writeln;
  end;
 write('Результат  '); writeln(k);
 readln;
end.
```

Рассмотрим задачи, в которых требуется в качестве обработки изменить порядок следования элементов в исходном массиве, например удалить или переставить местами один или несколько элементов в двумерном массиве.

При написании программы в этом случае можно записать элементы двумерного массива размерностью NxM в одномерный массив с количеством элементов N*M. Затем выполнить требуемые действия согласно алгоритму для одномерного массива. Завершающей операцией будет обратная запись одномерного массива на место двумерного.

Пример 7.22. Удаление в двумерном массиве элемента с номером X.

```
program ud_el;
  const  n=5;
           m=10;
  var a:array [1..N,1..M] of integer;
       b:array[1..N*M] of integer;
       x,i,j,k:byte;
begin
  k:=1;
  for i:=1 to n do
  for j:=1 to m do
    begin
      read(a[i,j]);
      b[k]:=a[i,j];       { заполнение двумерного массива и запись }
      k:=k+1;                    { его в одномерный }
    end;
          { Удаление некоторого элемента с номером x}
    for i:=x to n*m–1 do b[i]:=b[i+1];
    b[n*m]:=0;
    k:=1;
  for i:=1 to n do
```

```
    begin
      for j:=1 to m do
          begin
           a[i,j]:=b[k];
           k:=k+1;{обратная запись одномерного массива в двумерный}
           write(a[i,j]:5);
          end;
      writeln;
    end;
end.
```

Рассмотрим удаление или перенос одной или нескольких строк или столбцов. Удалить – это значит выбрать некоторую строчку (столбец), на ее (его) место записать следующую, сдвинув все строки вверх, а в последнюю 0.

Пример:

исходный массив	результат
a11 a12 a13	a11 a12 a13
a21 a22 a23	a31 a32 a33
a31 a32 a33	0 0 0

Здесь следует рассматривать двумерный массив как одномерный, элементами которого, в свою очередь, являются также одномерные массивы. В этом случае удаление строки или столбца реализуется с помощью аналогичных алгоритмов одномерного массива.

Пример 7.23. Удалить строку с номером k.

```
program mas6;
type
    t=array[1..5] of integer;
var
    a:array[1..5] of t;
    i,j,k:byte;
begin
 writeln('Дана матрица:');
 randomize;
 for  i:=1 to 5 do
 for j:=1 to 5 do
 a[i,j]:=random(10);   {заполнение и вывод массива}
 for i:=1 to 5 do
   begin
     for j:=1 to 5 do
           write(a[i,j]:3);
     writeln;
   end;
```

```
 writeln;
 writeln('Задание : удалить строку');
 writeln;
 repeat
   writeln('введите номер строки');
   readln(k);       { считывание номера строки }
                    { с последующей проверкой }
     if k>5 then
           writeln('Массив 5x5.Введите число < 5');
   until (0<k) and (k<5) ;

for i:=k to 4 do
       a[i]:=a[i+1];
   for j:=1 to 5 do
          a[5,j]:=0;
   writeln;
   writeln('Результат');
  for i:=1 to 4 do
  begin
    for j:=1 to 5 do
        write(a[i,j]:3);    {вывод измененной матрицы}
    writeln;
  end;
end.
```

Рассмотрим обработку двумерных массивов, представляющих собой числовые матрицы.

1) Суммирование матриц.

Фрагмент программы, реализующий суммирование:

```
for i:=1 to n do
for j:=1 to m do
c[i,j]:=a[i,j]+b[i,j];
{Можно   a[i,j]:=a[i,j]+b[i,j]}
```

2) Умножение вектора на матрицу. Результатом будет одномерный массив.

```
program  mm;
  { аналогично предыдущей  }
  begin
      for i:=1 to N do read(b[i]);
      for j:=1 to M do
```

```
      begin
       s:=0;
       for i:=1 to N do
         begin
           read(a[i,j]);
           s:=s+b[i]*a[i,j];        {выполнение умножения}
         end;
       c[j]:=s;
      end;
      <write>;
end.
```

3) Умножение матрицы на матрицу.

Умножаться могут только согласованные матрицы.

Фрагмент программы, реализующий умножение:

```
  for i:=1 to n do
  for j:=1 to m do    {n,m,k–размерность соответствующих параметров}
  begin
    s:=0;
    for e:=1 to k do
          s:=s+a[i,e]*b[e,j];
    c[i,j]:=s;
  end;
```

Примеры задач с двумерными массивами

Пример 7.24. В двумерном массиве вычислить произведение всех элементов массива.

```
program lupa3;
const
a:array[1..4,1..4] of integer =((4,2,–3,–6),
                                (1,8,3,2),   {Ввод с помощью}
                                (1,4,98,3), {типизированной константы}
                                (3,1,1,1));
var
  p,i,j:integer;
begin
 writeln('       В двумерном массиве вычислить произведение');
 writeln('              всех элементов массива.');
 writeln(' == Дана матрица ==');
 for i:=1 to 4 do
```

```
    begin
      for j:=1 to 4 do
           write(' ',a[i,j]:3);
      writeln;
    end;
  p:=1;
  for i:=1 to 4 do
  for j:=1 to 4  do
     begin
        p:=p*a[i,j];
     end;
    writeln('     *Результат*');
    write('      ***', p,'***');
 end.
```

Пример 7.25. Получить единичную матрицу путем замены элементов на главной диагонали на 1, а остальные – на 0.

```
program mas4;
var
  a:array[1..5,1..5] of integer;
  i,j:byte;
begin
  writeln('Дана матрица:5*5 ');
  for i:=1 to 5 do
  for j:=1 to 5 do          {заполнение и вывод массива}
    a[i,j]:=random(100); {датчик случайных чисел }
  for i:=1 to 5 do
   begin
    for j:=1 to 5 do
                  write(a[i,j]:3);
    writeln;
   end;
writeln;
writeln('Задание: получить единичную матрицу');
writeln('путем замены элементов на главной ');
writeln('диагонали на 1, а остальные – на 0');
for i:=1 to 5 do
for j:=1 to 5 do
  if i=j then           {замена элементов массива в соответствии }
                       { с определением единичной матрицы }
     a[i,j]:=1
     else
     a[i,j]:=0;
```

```
writeln;
writeln('Результат');
 for i:=1 to 5 do
     begin
       for j:=1  to 5 do         {вывод результата}
                    write(a[i,j]:3);
       writeln;
   end;
end.
```

Пример 7.26. Поменять местами два элемента в двумерном массиве.

```
program mas5;
type
    t=array[1..5] of integer;
var
    b:boolean;
    a:array[1..5]of t;
    i,j,f,e,g,s:byte;
    k:integer;
begin
   writeln('Дана матрица:');
   randomize;
   for i:=1 to 5 do
     begin
      for j:=1 to 5 do
        begin          {заполнение и вывод массива}
          a[i,j]:=random(10);
          write(a[i,j]:3);
        end;
       writeln;
     end;
    writeln('Задание : поменять местами любые два элемента');
    repeat
      writeln('введите номера строк');
       b:=false;
      readln(f);
      readln(e);    {считывание номеров строк и столбцов}
      writeln('введите номера столбцов');
      readln(g);
      readln(s);
      if (f<=5)and (e<=5) and (g<=5) and (s<=5 )  then
                                                  b:=true
                                               else
```

```
                    writeln('Массив 5x5.Введите числа <5.');
      writeln;
    until b=true;
    k:=a[e,s];
    a[e,s]:=a[f,g];                    {перестановка элементов}
    a[f,g]:=k;
    writeln('измененная матрица');
    for i:=1 to 5 do
      begin
       for j:=1 to 5 do
                  write(a[i,j]:3);    {вывод результата}
       writeln;
      end;
end.
```

Пример 7.27. В двумерном массиве найти сумму максимального и минимального элементов. Если она меньше 0, тогда все отрицательные элементы матрицы, лежащие выше главной диагонали, заменить на нули.

```
program konst1;
var
  a:array[1..6,1..6]  of integer;
  f:text;       { текстовый файл }
  max,min,i,j,p:integer;
begin
    assign(f,'fmas2.txt');
    reset(f);
writeln('В двумерном массиве найти сумму  макс. и мин. элемента,');
writeln('если она меньше 0, тогда все отрицательные элементы матрицы,');
writeln(' лежащие выше главной диагонали, заменить на нули.');
writeln(' Исходная матрица');
for i:=1 to 6 do
for j:=1 to 6 do
read (f,a[i,j]);    {считывание массива из файла}
for i:=1 to 6 do
 begin
  for j:=1 to 6 do
            write(a[i,j]:3);    {вывод массива в матричной форме}
  writeln;
 end;
 writeln;
 max:=a[1,1];
 for i:=1 to 6 do
```

```
 for j:=1 to 6 do       {поиск и вывод максимального элемента}
 if max<a[i,j] then
                 max:=a[i,j];
 writeln('максимальный элемент ', max);
 min:=a[1,1];
 for i:=1 to 6 do
 for j:=1 to 6 do
 if min>a[i,j] then
                min:=a[i,j];      {поиск и вывод минимального элемента}
 writeln ('минимальный элемент', min);
 p:=min+max;                              {поиск и вывод суммы}
 writeln ('сумма макс. и мин. элемента ',p);
 if p<0 then
   begin
     for i:=1 to 6 do
     for j:=1 to 6 do
     if (i<j) and (a[i,j]<0) then a[i,j]:=0;   {замена элементов по условию}
   end;
   writeln('измененная матрица ');
   for i:=1 to 6 do
      begin
        for j:=1 to 6 do    {вывод измененного массива в матричной форме}
        write(a[i,j]:3);
        writeln;
      end;
     close(f);
     readln;
end.
```

Пример 7.28. Найти произведение двух матриц a (2,3) и b (3,2).

```
program mas8;
uses crt;
var
  a,b,c:array[1..5,1..5] of integer;
  i,j,f,s:byte;
begin
 writeln('Даны две матрицы');
 randomize;
 for i:=1 to 2 do
 for j:=1 to 3  do
  a[i,j]:=random(10);
  for i:=1 to 2 do
   begin
```

```
    for j:=1 to 3 do
         write(a[i,j]:3);
    writeln;
  end;                    {заполнение и вывод матриц}
 writeln;
 for j:=1 to 3 do
 for f:=1 to 2  do
       b[j,f]:=random(10);
 for j:=1 to 3 do
 begin
   for f:=1 to 2 do  write(b[j,f]:3);
   writeln;
 end;
 writeln;
 writeln('Задание : найти произведение этих матриц');
 writeln('Результат:');
 writeln;
  for i:=1 to 2 do
  for f:=1 to 2 do
   begin
     s:=0;
     for j:=1 to 3 do          {поиск произведения}
          s:=s+a[i,j]*b[j,f];
     c[i,f]:=s;
   end;
   for i:=1 to 2 do
    begin
      for f:=1 to 2 do          {вывод результата}
      write(c[i,f]:5);
      writeln;
    end;
   readln;
   readln;
end.
```

Задания для самостоятельного выполнения

1. Дана вещественная матрица A(3,4). Составить программу подсчета количества элементов матрицы, удовлетворяющих условию p1<=a(i,j)<=p2. Значения p1 и p2 задаются самостоятельно.
2. Дана вещественная матрица A(4,4). Составить программу вычисления суммы элементов, расположенных выше главной диагонали.
3. Дана квадратная матрица A(N.N). Составить программу вычисления суммы элементов, расположенных ниже главной диагонали.

4. Дана целая матрица A(N,N). Составить программу подсчета количества четных элементов, расположенных ниже побочной диагонали.
5. Дана квадратная матрица A(N,N). Составить программу подсчета количества отрицательных элементов, расположенных ниже главной и ниже побочной диагонали.
6. Дана квадратная матрица A(N,N). Составить программу подсчета количества положительных элементов, расположенных выше главной и выше побочной диагонали.
7. Дана вещественная матрица A(N,M). Составить программу нахождения минимального положительного элемента матрицы и нахождения его местоположения.
8. Дана вещественная матрица A(N,M). Составить программу нахождения максимального отрицательного элемента матрицы и нахождения его местоположения.
9. Дана вещественная матрица A(N,M). Составить программу замены всех нулевых элементов матрицы на минимальный элемент.
10. Дана целая матрица A(N,N). Составить программу подсчета среднего арифметического значения матрицы. Найти отклонение от среднего у элементов первой строки.
11. Дана целая матрица A(N,N). Составить программу подсчета среднего арифметического значения матрицы. Вычислить отклонение от среднего для всех элементов матрицы.
12. Дана целая матрица A(N,N). Составить программу замены всех отрицательных элементов матрицы на среднее арифметическое значение элементов матрицы.
13. Дана вещественная матрица A(N,M). Составить программу замены всех положительных элементов матрицы на элемент, имеющий мини-мальное значение.
14. Дана вещественная матрица A(N.M). Составить программу замены всех отрицательных элементов матрицы на элемент, имеющий макси-мальное значение.
15. Составить программу замены всех отрицательных элементов матрицы A(6,6) на 0, если сумма минимального и максимального элементов этой матрицы окажется меньше P.
16. Составить программу замены всех отрицательных элементов матрицы A(N,N) на элемент этой матрицы, имеющий минимальное значение. Скорректированную матрицу напечатать.
17. Составить программу нахождения числа строк матрицы A(N,N), сумма элементов у которых отрицательна.
18. Составить программу нахождения числа строк матрицы A(N,N), количество отрицательных элементов в которых больше P.

19. Составить программу формирования и выдачи на печать вектора (b1...b6), если bi–сумма минимального и максимального элементов i–строки матрицы A(6*4).
20. Составить программу нахождения числа строк матрицы A(6,6), максимальный элемент которых больше P.
21. Составить программу нахождения числа строк матрицы A(6,6), минимальный элемент которых меньше P.
22. Составить программу замены всех четных элементов матрицы A(N,N) на элемент этой матрицы, имеющий максимальное значение, расположенный выше главной диагонали.
23. Составить программу нахождения минимального положительного элемента в каждом столбце матрицы A(N,N).
24. Дана вещественная матрица размером 7*4. Найти наибольший элемент матрицы. Удалить строку с максимальным элементом.
25. Дана вещественная матрица размером 7*4. Поменять столбец с максимальным элементом с первым столбцом матрицы.
26. Дана вещественная матрица размером 7*7. Найти максимальный элемент матрицы, расположенный ниже побочной диагонали.
27. Дана вещественная матрица размером 7*4. Найти наименьший элемент матрицы. Перенести строку, содержащую этот элемент, в конец.
28. Дана вещественная матрица размером 7*4. Найти максимальный элемент матрицы. Поменять столбец, содержащий этот элемент с последним столбцом матрицы.
29. Дана вещественная матрица размером 6*4. Найти минимальный элемент матрицы. Переставляя ее строки и столбцы, добиться того, чтобы он оказался в правом нижнем углу.
30. Составить программу удаления в матрице A(6,6) максимального элемента.
31. Составить программу удаления строк матрицы A(6,6), минимальный элемент которых отрицательный.
32. В матрице A(6,6) перенести в строке все отрицательные элементы в начало.
33. Дан массив C(6,6). Определить количество «особых» элементов массива, считая элемент «особым», если он больше суммы остальных элементов своего столбца. Напечатать индексы «особых» элементов.
34. Дан массив C(6,6). Определить количество «особых» элементов массива, считая элемент «особым», если в его строке слева от него находятся элементы меньшие его, а справа – большие.
35. По массиву A(5,6) получить массив B(6), присвоив его j–элементу значение true, если все элементы j–столбца массива A нулевые, и значение false, если иначе.

36. По массиву A(5,6) получить массив B(5), присвоив его i–элементу значение true, если все элементы i–строки положительны, и значение false, если иначе.
37. Дана вещественная матрица A(5,4). Строку, содержащую максимальный элемент, поменять местами со строкой, содержащей минимальный элемент.
38. Дана вещественная матрица A(5,4). Столбец, содержащий максимальный элемент, поменять местами со столбцом, содержащим минимальный элемент.
39. Составить программу замены всех отрицательных элементов матрицы A(4,5) на минимальный элемент.
40. Дан массив A(6,6). Сформировать массив B(6), где Bi – количество элементов в i–строке матрицы A, удовлетворяющих условию a[i–1,j]<a[i,j]<a[i+1,j].
41. Дан массив A(6,6). Сформировать массив B(6), где Bj – количество элементов в j–столбце матрицы A, удовлетворяющих условию a[i,j]<=a[i,j+1]/2.
42. Дана вещественная матрица C(4,6). Найти минимальный элемент в каждой строке матрицы. Если все полученные минимальные элементы положительны, то выдать сообщение об этом.
43. Дана вещественная матрица C(5,4). Найти максимальный элемент в каждом столбце матрицы. Выдать на печать сообщение, если все эти максимальные элементы положительны.

Контрольные вопросы

1. Что такое массив?
2. Какие способы объявления массивов применяются в TP?
3. Основные функции для работы с массивами.
4. Основные правила использования массивов в программах.

8. АЛГОРИТМЫ СОРТИРОВКИ МАССИВОВ

Под сортировкой понимают процесс перестановки объектов данного массива в определенном порядке. Целью сортировки является упорядочение массивов для облегчения последующего поиска элементов в данном массиве. Рассмотрим основные алгоритмы сортировки по возрастанию значений эленентов массивов.

8.1. Модифицированный метод простого выбора

В последовательности a1,a2,...an отыскивается минимальный элемент, который ставится на первое место. Для того чтобы не потерять элемент, стоящий на первом месте, этот элемент устанавливается на место минимального. Затем в усеченной последовательности (исключая первый элемент) отыскивается минимальный элемент и ставится на второе место и так далее [n–1] раз, пока не встанет на свое место предпоследний [n–1] элемент массива A, сдвинув максимальный элемент в самый конец.

Пример 8.1. Сортировка одномерного массива по возрастанию значений модифицированным методом простого выбора.

```
 var
  a:array[1..10] of real;
  i,j,k,n:integer;
  m:real;
 begin
writeln('Сортировка массива модифицированным ');
writeln('          методом простого  выбора');
  writeln('Введите количество элементов');   readln(n);
  for i:=1 to n do
begin
    write('Введите A[',i,']  ');
    readln(a[i]);
   end;
 for i:=1 to n–1 do
  begin
     m:=a[i];
     k:=i;
     for j:=i+1 to n do
     begin
      if a[j]<m then
       begin
         m:=a[j];
```

```
        k:=j;
       end;
     end;
   a[k]:=a[i];
   a[i]:=m;
 end;
 for i:=1 to n do
 writeln(a[i]:8:4);
 readln;
end.
```

Пример 8.2. Сортировка двумерных массивов. Переставить элементы по убыванию в двумерном массиве методом модифицированного простого выбора.

```
program sortm2;
var
 a:array[1..4,1..4] of integer;
 b:array[1..16] of integer;
 n,q,i,j,k,m:integer;
begin                                {ввод исходного массива с помощью }
 writeln('Исходный массив'); { датчика случайных чисел }
 for i:=1 to 4 do
   begin
     for j:=1 to 4 do
       begin
           a[i,j]:=random(30);
           write(a[i,j]:3);
         end;
        writeln;
   end;
 q:=1;
 for i:=1 to 4 do
 for j:=1 to 4 do
 begin          {переписываем двумерный массив в одномерный}
  b[q]:=a[i,j];
  q:=q+1;
 end;
 for q:=1 to 15 do   {начало сортировки}
  begin
   m:=b[q];k:=q;
   for n:=q+1 to 16 do
   begin
```

```
    if b[n]>m then   {поиск максимального элемента}
      begin
        m:=b[n];
        k:=n;
      end;
   end;
    b[k]:=b[q];
    b[q]:=m;
  end;          {конец сортировки}
 q:=1;
for i:=1 to 4 do
for j:=1 to 4 do
  begin        {переписываем одномерный массив в двумерный}
    a[i,j]:=b[q];
    q:=q+1;
  end;
  writeln('Результат');
  for i:=1 to 4 do   {вывод результата}
 begin
  for j:=1 to 4 do
  write(a[i,j]:3);
  writeln;
 end;
 readln;
end .
```

Пример 8.3. Сортировка двумерных массивов. В двумерном массиве расположить элементы на четных местах по убыванию методом модифицированного простого выбора.

```
program sortm3;
 var
   a:array[1..4,1..4] of integer;
   b:array[1..16] of integer;
   q,n,max,i,j,k:integer;
 begin                    {ввод исходного массива с помощью}
   writeln('Исходный массив');   {датчика случайных чисел}
   randomize;
   for i:=1 to 4 do
   for j:=1 to 4 do
        a[i,j]:=random(100);
   for i:=1 to 4 do
     begin
```

```
          for j:=1 to 4 do
                write(a[i,j],' ');
          writeln;
      end;
   q:=1;
   for i:=1 to 4 do
   for j:=1 to 4 do
       begin          {переписываем двумерный массив в одномерный}
         b[q]:=a[i,j];
         q:=q+1;
       end;
  for q:=1 to 7 do  {начало сортировки}
     begin
       max:=b[q*2];
       k:=q;
  for n:=q+1 to 8 do
     begin
       if b[n*2]>max then    {поиск максимального элемента}
        begin
          max:=b[n*2];
          k:=n;
        end;
     end;
   b[k*2]:=b[q*2];
   b[q*2]:=max
 end;         {конец сортировки}
    q:=1;
    for i:=1 to 4 do
    for j:=1 to 4 do
  begin         {переписываем одномерный массив в двумерный}
    a[i,j]:=b[q];
    q:=q+1;
  end;
   writeln('Полученный массив');
   for i:=1 to 4 do   {вывод результата}
     begin
       for j:=1 to 4 do
                write(a[i,j]:4);
       writeln;
     end;
 readln;
end.
```

Пример 8.4. Сортировка двумерных массивов. Расположить элементы в строках двумерного массива по возрастанию значений модифицированным методом простого выбора.

```
program  sortm9;
type
 arr=array[1..4,1..4] of integer;
var
 a:arr;
 i,j,k,f,m:integer;
begin
 writeln(' Исходный массив');
 randomize;       {ввод исходного массива с помощью}
 for i:=1 to 4 do         {датчика случайных чисел}
 for j:=1 to 4 do
  a[i,j]:=random(100);
 for i:=1 to 4 do
 begin
     for j:=1 to 4 do write (a[i,j]:3);
     writeln;
 end;
 for i:=1 to 4 do
 for j:=1 to 3 do
 begin        {начало сортировки по строкам}
  m:=a[i,j];
  k:=j;
  for f:=j+1 to 4 do
  begin
   if a[i,f]<m then  {поиск минимального элемента}
   begin
    m:=a[i,f];
    k:=f;
   end;
  end;
  a[i,k]:=a[i,j];
  a[i,j]:=m;
end; {конец сортировки}
writeln('Полученный массив');
for i:=1 to 4 do
begin              {распечатка результата}
 for j:=1 to 4 do  write(a[i,j]:3);
 writeln;
end;
```

```
 readln;
end.
```

Пример 8.5. Сортировка двумерных массивов. Расположить по убыванию элементы каждого столбца двумерного массива методом модифицированного выбора.

```
program sortm4;
type
   myarr= array[1..4] of integer;
   myre = array[1..4] of myarr;
var
   a:myarr;
   c:myre;
   n,k,i,j,q:byte;
begin
 writeln('Исходный массив');  {ввод исходного массива с помощью}
 randomize;                   {датчика случайных чисел}
 for i:=1 to 4 do
 for j:=1 to 4 do
 c[i,j]:=random(100);
 for j:=1 to 4 do
 begin
  for i:=1 to 4 do
            write(c[i,j],' ');
  writeln;
 end;
 for i:=1 to 4 do
  begin          {переписываем каждый столбец двумерного}
  q:=1;          {массива в одномерный}
   for j:=1 to 4 do
     begin
       a[q]:=c[i,j];
       q:=q+1;
     end;
   begin         {начало сортировки}
  for q:=1 to 3 do
    begin
      m:=a[q];
      k:=q;
      for n:=q+1 to 4 do
        begin
          if a[n]>m then  {поиск максимального элемента}
```

```
          begin
            m:=a[n];
            k:=n;
          end;
        end;
      a[k]:=a[q];
      a[q]:=m;
    end;
   end;    {конец сортировки}
   q:=1;
   for j:=1 to 4 do
   begin
     c[i,j]:=a[q]; {переписываем каждый одномерный массив}
     q:=q+1;              {в столбец двумерного}
   end;
  end;
  writeln('Полученный массив');
  for j:=1 to 4 do         {вывод результата}
  begin
     for i:=1 to 4 do
                write(c[i,j],' ');
     writeln;
  end;
 readln;
end.
```

8.2. Метод парных перестановок

Самый простой вариант этого метода основан на принципе сравнения и обмена пары соседних элементов.

Процесс перестановок пар повторяется просмотром массива с начала до тех пор, пока не будут отсортированы все элементы, т. е. во время очередного просмотра не произойдет ни одной перестановки.

Пример 8.6. Сортировка одномерного массива методом парных перестановок.

```
var
  a:array[1..10] of real;
  i,k,n:integer;
  q:real;
begin
  writeln('Сортировка массива методом парных перестановок');
```

```
 writeln('Введите количество элементов');
 readln(n);
 for i:=1 to n do
  begin
    write('Введите A[',i,']  ');
    readln(a[i]);
  end;
repeat
 k:=0;
 for i:=1 to n–1 do
 begin
  if a[i]>a[i+1] then
    begin
      q:=a[i];
      a[i]:=a[i+1];
      a[i+1]:=q;
      k:=k+1;
    end;
 end;
until k=0 ;
for i:=1 to n do
              writeln(a[i]:8:4);
readln;
end.
```

Пример 8.7. Сортировка двумерных массивов. Отсортировать массив по возрастанию элементов методом парных перестановок.

```
program sortm1;
 const n=4;
var
 b:array[1..n*n] of integer;
 a:array[1..n,1..n] of integer;
 q,i,j,k,g,m,gluk:integer;
begin
 writeln('Исходный массив');
 randomize;
 for i := 1 to n do
 begin
  for j := 1 to n do
  begin             {ввод исходного массива}
   gluk:=random(100);  {с помощью датчика случайных чисел}
   a[i,j]:=gluk–50;      write(a[i,j]:4);    {вывод массива}
  end;
```

```
 writeln;
end;
q:=1;
for i:=1 to n do
for j:=1 to n do
begin          {переписываем двумерный массив в одномерный}
 b[q]:=a[i,j];
 q:=q+1;
end;
repeat       {начало сортировки}
 k := 0;
 for q :=1 to (n*n–1) do
 begin
  if b[q] > b[q+1] then
  begin      {меняем элементы местами}
   g :=b[q];
   b[q]:=b[q+1];
   b[q+1] := g;
   k :=k +1;
  end;
 end;
until k =0;  {конец сортировки}
q:=1;
for i:=1 to n do
for j:=1 to n do
begin         {переписываем одномерный массив в двумерный}
 a[i,j]:=b[q];
 q:=q+1;
end;
writeln('результат');
for i := 1 to n do
begin        {вывод результата}
 for j := 1 to n do
 write(a[i,j]:4);
 writeln;
end;
readln;
end.
```

Пример 8.8. Сортировка двумерных массивов. В двумерном массиве отсортировать элементы на четных местах по убыванию методом парных перестановок.

```
program sortm10;
```

```
var
    g,n,i,k,q,j:integer;
    b:array [1..16] of integer;
    a:array [1..4,1..4] of integer;
begin                  {ввод исходного массива с помощью }
 writeln('Исходный массив');   { датчика случайных чисел}
 randomize;
  for i:=1 to 4 do
  for j:=1 to 4 do
      a[i,j]:=random(100);
 for i:=1 to 4 do
  begin
    for j:=1 to 4 do
    write(a[i,j],' ');
    writeln;
  end;
   q:=1;
   for i:=1 to 4 do
   for j:=1 to 4 do
    begin        {переписываем двумерный массив в одномерный}
      b[q]:=a[i,j];
      q:=q+1;
    end;
    repeat       {начало сортировки}
      k:=0;
      for q:=1 to 14 do
      if (q mod 2)=0 then
      begin
        n:=q;    {поиск элемента для сравнения}
          repeat
            n:=n+2
          until n<=16;
        if b[q]<b[n] then
          begin    {перестановка элементов}
             g:=b[q];
             b[q]:=b[n];
             b[n]:=g;
             k:=k+1;
     end;
    end;
   until k=0;   {конец сортировки}
q:=1;
for i:=1 to 4 do
```

```
  for j:=1 to 4 do
  begin          {переписываем одномерный массив в двумерный}
    a[i,j]:=b[q];
    q:=q+1;
  end;
     writeln('результат');
     for i:=1 to 4 do
     begin         {вывод результата}
       for j:=1 to 4 do
          write(a[i,j]:3);
          writeln;
     end;
    readln;
end.
```

Пример 8.9. Сортировка двумерных массивов. Расположить по возрастанию элементы каждой строки двумерного массива методом парных перестановок.

```
program sortm7;
type
 an = array [1..4] of integer;.
var
  a:array [1..4] of an;
  n,l,j:byte;
  k, q:integer;
begin
  writeln ('Исходный массив');
  randomize;       {ввод исходного массива с помощью}
  for j:=1 to 4 do    {датчика случайных чисел}
  for l:=1 to 4 do
  a[j,l]:=random(100);
  for j:=1 to 4 do
     begin
       for l:=1 to 4 do
   write(a[j,l],' ');
   writeln;
  end;
  n:=4;
  for j:=1 to 4 do
  begin    {начало сортировки по строкам}
   repeat
    k:=0;
    for l:=1 to n–1 do
```

```
   if a[j,l]>a[j,l+1] then   {перестановка элементов}
   begin
    q:=a[j,l];
    a[j,l]:=a[j,l+1];
    a[j,l+1]:=q;
    k:=k+1;
   end;
  until k=0; {конец сортировки}
 end;
 writeln('Конечный массив');
 for j:=1 to 4 do    {распечатка результата}
  begin
     for l:=1 to 4 do  write(a[j,l],' ');
     writeln;
  end;
 readln;
end.
```

Пример 8.10. Сортировка двумерных массивов. Расположить элементы в столбцах в порядке убывания методом парных перестановок.

```
program sortm6;
type
 arr=array[1..4,1..4] of integer;
var
 a:arr;
 i,k,q,j:integer;
begin
 randomize;     {ввод исходного массива с помощью}
 for i:=1 to 4 do        {датчика случайных чисел}
 for j:=1 to 4 do
  a[i,j]:=random(10);
 writeln;
 writeln('Исходный массив');
 for i:=1 to 4 do
 begin
  for j:=1 to 4 do
  write(a[i,j]:4);
  writeln;
 end;
 for j:=1 to 4 do
 begin       {начало сортировки по столбцам}
  repeat
   k:=0;
```

```
   for i:=1 to 3 do
    if a[i,j]<a[i+1,j] then  {перестановка элементов}
    begin
     q:=a[i,j];
     a[i,j]:=a[i+1,j];
     a[i+1,j]:=q;
     k:=k+1;
    end;
  until k=0;   {конец сортировки}
 end;
 writeln('Результат');
 for i:=1 to 4 do  {распечатка результата}
 begin
  for j:=1 to 4 do
  write(a[i,j]:4);
  writeln;
 end;
 readln;
end.
```

Задания для самостоятельного выполнения

Нечетные варианты задания выполнять модифицированным методом простого выбора, а четные – методом парных перестановок.

1–2. Дана последовательность a1,a2,...,a20. Расположить положительные элементы последовательности, стоящие на нечетных местах, по возрастанию.

3–4. Дана последовательность a1,a2,...,a15. Расположить ненулевые элементы последовательности по убыванию.

5–6. Дана последовательность x1,x2,...,x20. Элементы, стоящие на нечетных местах, расположить в порядке возрастания, а на нечетных - в порядке убывания.

7–8. Дана последовательность a1,a2,...,a15. Требуется упорядочить ее по возрастанию абсолютных значений элементов.

9–10. Дана последовательность x1,x2,...,x20. Требуется расположить отрицательные элементы последовательности в порядке убывания.

11–12. Дана последовательность a1,a2,...,a20. Расположить положительные элементы последовательности по убыванию.

13–14. Дана последовательность a1,a2,...,a15. Расположить отрицательные элементы по возрастанию.

15–16. Дана последовательность a1,a2,...,a15. Расположить элементы на четных местах по убыванию.

17–18. Дана последовательность a1,a2,...,a15. Расположить четные элементы последовательности по возрастанию.

19–20. Дана последовательность a1,a2,...,a20. Расположить нечетные элементы последовательности по убыванию.
21–22. Дана последовательность x1,x2,...,x15. Расположить четные положительные элементы по возрастанию.
23–24. Дана последовательность x1,x2,...,x20. Расположить нечетные отрицательные элементы по убыванию.
25–26. Дана последовательность x1,x2,...,x20. Расположить элементы большие 10 по возрастанию.
27–28. Дана последовательность x1,x2,...,x20. Расположить элементы меньшие 10 по убыванию.
29–30. Дана последовательность x1,x2,...,x20. Расположить по возрастанию четные элементы последовательности, стоящие на четных местах.
31–32. Расположить элементы в нечетных строках двумерного массива в порядке возрастания значений.
33–34. Расположить элементы в четных столбцах в порядке убывания их значений.
35–36. Расположить отрицательные элементы двумерного массива в порядке возрастания значений.
37–38. С помощью процедуры сортировки в каждой строке двумерного массива перенести отрицательные элементы в конец строки.

Контрольные вопросы

1. Какова цель сортировки массивов?
2. В чем заключается сортировка модифицированным методом простого выбора?
3. В чем заключается сортировка методом парных перестановок?
4. На каком принципе основан метод парных перестановок?

9. ПОДПРОГРАММЫ

В системе программирования Turbo Pascal концепция подпрограмм реализуется в виде подпрограмм-процедур и подпрограмм-функций. Подпрограммы, реализованные в системе Turbo Pascal, называются **стандартными**. С некоторыми вы уже познакомились.

В качестве примера приведем стандартные функции sin(x), cos(x), length(s), eof(f) и стандартные процедуры READ(x,y), WRITE(x,y). Основное отличие функций от процедур состоит в том, что они предназначены для вычислений единственного значения, а в процедурах можно вычислять сколь угодно много значений.

Кроме стандартных подпрограмм Вы можете объявить и использовать свои подпрограммы, реализующие отдельные алгоритмы и вычисления.

Применение подпрограмм дает возможность уменьшать число повторений одной и той же последовательности операторов, а также конструировать программу как набор отдельных подпрограмм. Это позволяет получить более логичный процесс программирования. Каждая процедура или функция определяется только один раз, но может использоваться многократно.

Структура процедур и функций аналогична структуре полной программы на языке Turbo Pascal. В процедурах и функциях могут быть описаны собственные метки, константы, типы, собственные переменные и даже собственные процедуры и функции. Внутренние описания должны следовать в том же порядке, что и разделы основной программы.

Структура программы, использующей подпрограммы, является многоуровневой, то есть внутри каждой подпрограммы может содержаться любое количество других подпрограмм. Все объекты, описанные на уровне главной программы, называются **глобальными**. Они доступны для использования внутри любой подпрограммы, входящей в состав основной программы. Объекты, описанные в виде локальных переменных, располагаются внутри соответствующей подпрограммы. Если в некоторой процедуре описан объект, имя которого совпадает с именем глобального объекта, то в этом случае этот глобальный объект становится недоступным в данной подпрограмме.

9.1. Процедуры

Процедурами в Turbo Pascal называют вид подпрограмм. Они предназначены для оформления любой самостоятельной части программы в виде отдельной синтаксической конструкции, которая решает

определенную задачу. Процедуры используются для изменения окружающей среды программы и, как правило, определяют некоторые действия.

Описание каждой процедуры начинается с заголовка, в котором задаются имя процедуры и список формальных параметров с указанием их типов. Процедура может быть и без параметров, тогда в заголовке указывается только ее имя. С помощью параметров осуществляеся передача исходных данных в процедуру, а также передача результатов работы обратно в вызвавшую ее программу.

Структура процедуры:

```
Procedure <имя>(список параметров);
<раздел описания>
begin
<раздел операторов>
end;
```

Список параметров, указанный в заголовке любой подпрограммы, в том числе и процедуры, содержит имена параметров и их типы. Это формальные параметры. Заголовок процедуры может не содержать списка параметров.

Пример 9.1.

```
program nat4;
{ процедура без параметров }
procedure text;
begin
    writeln(' Я обучаюсь на естественнонаучном факультете ');
    writeln('  Ульяновского Государственного Технического  ');
    writeln('   Университета по специальности прикладная   ');
    writeln('          математика и информатика.          ');
end;{text}

{***MAIN***}
begin
    writeln(' Пример 1.1: вывести текст на экран ');
    { вызов процедуры }
        text;
end.
```

Формальные параметры указываются в заголовке подпрограммы.

Само описание процедуры не вызывает конкретных действий.

Для того, чтобы выполнить действия, указанные в процедуре, необходимо в вашей программе использовать оператор вызова соответствующей процедуры. Например, для вызова процедуры в нужном месте программы указывается имя этой процедуры, за которым в круглых скобках могут следовать параметры, передаваемые из программы в процедуру. Такие параметры называются **фактическими**. При их указании в этом списке типы не используются:

< имя >(x,y);

Пример: write(x,y);

При вызове процедуры из основной программы формальные параметры заменяются на фактические. Поэтому количество формальных и фактических параметров должно совпадать.

Фактические параметры указываются при вызове подпрограммы.

9.2. Функции

Подпрограмма-функция предназначена для вычисления какого-либо одного значения. Описание каждой функции начинается с заголовка, в котором задаются имя функции, список формальных параметров с указанием их типов и тип значения функции.

С помощью параметров осуществляется передача исходных данных в подпрограмму, а также передача результатов работы обратно в вызвавшую ее программу. Описание функции в общем случае выглядит следующим образом:

```
FUNCTION <имя>(список параметров):<тип>;
<раздел описания>
begin
<раздел операторов>
end;
```

Функция может возвратить параметры следующих типов: целого, символьного, вещественного, строкового и логического.

В разделе операторов функции хотя бы раз имени функции должно быть присвоено значение.

Пример 9.2. Функции вычисления факториала числа N.

```
function Factorial(n :Integer): Longint;{заголовок функции}
VAR Fact: Longint;                    {раздел объявлений}
     i: Byte;
begin                                 {раздел операторов}
  Fact := n;
  for i := n–1 downto 2 do
    Fact := Fact*i;
  Factorial := Fact;       {вычисленное значение факториала}
end;
```

Для вызова функции из основной программы или другой подпрограммы следует в выражении, где необходимо использовать значение функции, указать имя функции со списком фактических параметров, которые должны совпадать по количеству и типам с формальными параметрами функции, например:

```
  Part := Sqr(T)/Factorial(i);
```

В этом операторе:

Sqr(T) – вызов стандартной функции возведения в квадрат с фактическим параметром T;

Factorial(i) – вызов функции, вычисляющей факториал с фактическим параметром i.

9.3. Формальные и фактические параметры

Все формальные параметры можно разбить на четыре категории:

1) параметры-значения (они в основной программе подпрограммой не меняются);

2) параметры-переменные (их подпрограмма может изменить в основной программе);

3) параметры-константы (используются только в версии 7.0);

4) параметры без типа.

Для каждого формального параметра следует указать имя и, как правило, тип. Имена параметров могут быть любыми, в том числе и совпадать с именами объектов программы. Необходимо лишь помнить, что в этом случае параметр основной программы с таким именем становится недоступным для непосредственного использования подпрограммой. Тип формального параметра может быть практически любым, однако в

заголовке подпрограммы нельзя использовать структурный тип, например, описание массива или записи.

Например, нельзя писать:

```
function Max(A: array[1..100] of Real): Real;
```

Чтобы правильно записать этот заголовок, следует в основной программе ввести тип-массив, а затем использовать его в заголовке:

```
type tArr = array[1..100] of Real;
function Max(A: tArr): Real;
```

Пример 9.3. Функция вычисления максимального элемента массива, размер массива и его элементы определяются датчиком случайных чисел.

```
Program Maxelement;
Type
  tArr=array[1..100] of Integer;
Var
  Massiv  : tArr;
  Maxim   : Integer;
  K ,X    : Byte;
Function Max(Mas : tArr; N: Byte): Integer;
Var Ma : Integer;
    i : Byte;
Begin
  Ma := Mas[1];
  for i := 2 to N do
    if Ma < Mas[i] then
      Ma := Mas[i];
  Max := Ma;
End;{Max}
{************************************}
Begin
  randomize;
  k := random(10) + 1;          {размер массива}
  write(' элементы массива :');
  for x := 1 to K do
    begin
      Massiv[x] := random(100);
      write(' ',massiv[x]);
    end;
```

```
  Maxim := Max(Massiv,K);
  writeln('   максимальный элемент массива : ',Maxim:3);
End.
```

9.3.1. Параметры-значения

Параметры-значения передаются основной программой в подпрограмму через стек в виде копий и собственный параметр программы подпрограммой изменится не может.

Если параметров-значений одного типа несколько, их можно объединить в одну группу, перечислив их имена через запятую, а затем уже указать общий тип. Отдельные группы параметров отделяются друг от друга точкой с запятой. Например:

```
function Mult(X, Y: Integer; A: REAL):REAL;
```

В качестве фактического параметра на месте параметра-значения при вызове подпрограммы может выступать любое выражение совместимого для присваивания типа, не содержащее файловую компоненту. В примере 9.3 формальные параметры являются параметрами-значениями.

Параметры-значения используются для передачи исходных данных в подпрограмму.

Пример 9.4. Программа вычисления корней уравнения.

```
program nat3;
uses  crt;
var   a,b,c:integer;
      x1,x2:real;
procedure srt( a1,b1,c1:integer );{вычисление корней уравнения}
{a1,b1,c1–формальные параметры–значения}
var   d:real;
{d локальная переменная}
begin
    d:=b1*b1–4*a1*c1;
    if d<0.0 then begin
                writeln('нет корней.');
              end
            else
    if d=0.0 then begin
                writeln('корни равны.');
                x1:=–b1/(2*a1);
```

```
                writeln(' x= ',x1:7:2);
              end
            else begin
                x1:=(–b1+sqrt(d))/(2*a1);
                x2:=(–b1–sqrt(d))/(2*a1);
                writeln('       x1= ',x1:7:2,',');
                writeln('       x2= ',x2:7:2,'.');
              end;
 end;{srt}
****** MAIN ********}
 begin
  writeln(' Пример 9.4:вычисление корней квадратного уравнения ');
  writeln('         (коэффициенты уравнения вводятся с      ');
  writeln('           помощью датчика случайных чисел).     ');
  randomize;
  a:=random(10);
  a:=a–random(10);
  b:=random(10);
  b:=b–random(10);
  c:=random(10);
  c:=c–random(10);
  writeln(' Решаемое уравнение: ');
  writeln('                  ',a,'*x*x+',b,'*x+c=0.');
  writeln(' Ответ: ');
  {вызов процедуры}
                srt(a,b,c);
 end.
```

9.3.2. Параметры-переменные

При передаче параметров-переменных в подпрограмму фактически через стек передаются их адреса в порядке, объявленном в заголовке подпрограммы. Следовательно, подпрограмма имеет доступ к этим параметрам и может их изменять.

Параметр–переменная указывается в заголовке подпрограммы аналогично параметру–значению, но только перед именем параметра записывается зарезервированое слово var. Действие слова var распространяется до ближайшей точки с запятой, т. е. в пределах одной группы.

Пример.

```
procedure MaxMin(A: tArr; var Max, Min: Real; N: Word);
```

Здесь Max, Min – параметры-переменные, A и N – параметры-значения. Тип параметров-переменных может быть любым, включая и файловый. При вызове подпрограммы на месте параметра-переменной в качестве фактического параметра должна использоваться переменная идентичного типа.

Так, если формальный параметр имеет тип, определенный следующим образом:

```
type tArr= array[1..10] of Integer;
```

то и фактический параметр должен быть переменной или типизированной константой типа tArr.

Пример 9.5. Процедура, меняющая местами первый и последний элемент массива.

```
Procedure MM(var Mas : tArr; N: Byte);
begin
  Ma := Mas[1];
  Mas[1]:=Mas[n];
  Mas[n]:=Ma;
end;
```

В данном случае в стеке не создается копия исходного массива, что улучшает быстродействие и экономит память.

Параметры-переменные используются для передачи преобразованных в подпрограмме данных в основную программу.

9.3.3. Параметры-константы

Часто в качестве параметра в подпрограмму следует передать ту или иную переменную, но изменять ее подпрограмма не должна. В этой ситуации параметр лучше передать как параметр-константу. Такой параметр, если он структурированного типа, передается своим адресом, но предусматривается защита от его изменения.

function NewString(const S: integer):integer;

Параметр-константу нельзя передавать в другую подпрограмму в качестве фактического параметра.

Пример 9.6. Функция вычисления первого нулевого элемента массива с использованием в качестве параметра параметр-константу размер массива и его элементы определяются датчиком случайных чисел.

```
Program NoolevoiElement;
Type
  tArr=array[1..15] of Integer;
Var
  Massiv          : tArr;
  Ma,Number,K ,X   : Byte;
Function Nool(const Mas : tArr; N: Byte): Byte;
Var
  i: Byte;
begin
  Nool:=0;
  for i := 1 to N do
    if  Mas[i]=0 then
      begin
        Nool := i;
        Exit; {выход из подпрограммы}
      end   else Ma := Ma +1; {Ma-глобальный параметр}
end;{Nool}
Begin
  randomize;
  K := random(10) + 1;
  writeln(' размер массива : ',K);
  write(' элементы массива :');
  for x := 1 to K do
    begin
      Massiv[x] := random(100);
      write(' ',massiv[x]);
    end;
  Ma := 0;
  Number := Nool(Massiv,K);
  if Ma = K then writeln(' нулевого элемента в массиве нет') else
  writeln(' первый нулевой элемент в массиве с номером ',Number);
End.
```

9.3.4. Параметры без типа

В Turbo Pascal можно использовать параметры-переменные и параметры-константы без указания типа. В этом случае фактический параметр может быть переменной любого типа, а ответственность за правильность использования того или иного параметра возлагается на программиста.

Пример.

```
function Equal(var Param1, Param2; Len: Word):Boolean;
```

Здесь Param1, Param2 – параметры–переменные без типа (вместо них можно использовать, например, любые переменные простого типа, типа-массив, типа-запись и т. д.); Len – параметр-значение.

Следует иметь в виду, что параметр без типа внутри подпрограммы типа не имеет и его перед использованием следует преобразовать к конкретному типу, применяя идентификатор соответствующего типа, при этом полученный результат может быть любого размера.

Пример 9.7. Функция вычисления максимального элемента в массиве.

```
function Max(Var Mas; N: Byte;Var Ma:integer);
type
  tArray = array[1..Maxint] of Integer;
                        {тип массива максимального размера}
var
     i: Byte;
begin
  Ma := tArray(Mas)[1]; {преобразование}
  for i := 2 to N do
    if tArray(Mas)[i] > Ma then
      Ma := tArray(Mas)[i];
end;
```

В этом случае в качестве первого передаваемого параметра можно использовать любой массив (и не только массив), так что подпрограмма становится более универсальной.

9.3.5. Массивы открытого типа

В версии 7.0 можно в качестве параметров-переменных использовать массивы открытого типа, у которых не задаются размеры.

В качестве фактического параметра в этом случае можно использовать массив любого размера, однако массив должен состоять из тех же

компонент что и компоненты открытого массива. Такие параметры введены для того, чтобы подпрограмма могла обрабатывать массив любого размера. Фактический размер массива в этом случае может быть определен с помощью функции High. Открытый массив задается как и обычный массив, но только без указания типа индекса. Индексация элементов открытого массива начинается с нуля, а максимальный индекс элемента равен значению функции High.

Пример 9.8. Функция вычисления максимального элемента в массиве.

```
function Max(var Mas : Array of Integer): Integer;
var Ma: Integer;
    i: Byte;
begin
  Ma := Mas[0];
  for i := 1 to High(Mas) do       {цикл до наибольшего индекса}
  if Ma < Mas[i] then
       Ma :=Mas[i];
  Max := Ma
end;
```

Пример 9.9. Функция нахождения суммы элементов массива меньше заданного числа с использованием в качестве передаваемого параметра массива открытого типа; размер массива и его элементы определяются датчиком случайных чисел.

```
Program SummaElementov;
Type
  tArr=array[1..15] of Integer;
Var
  Massiv      : tArr;
  Ma,K,X      : Byte;
  C           : Integer;
  Sum         : Integer;
Function Summa(var Mas : Array of integer): Integer;
Var S: Integer;
    i: Byte;
begin
  S := 0;
  for i := 0 to High(Mas) do
    if  Mas[i] < C then S := S + Mas[i] else Ma := Ma + 1;
  Summa := S;
end;{Summa}
```

```
Begin
  randomize;
  K := random(10) + 1;
  writeln(' размер массива : ',k);
  write(' элементы массива :');
  for x := 1 to K do
    begin
      Massiv[x] := random(100) – 50;
      writeln(' ',Massiv[x]);
    end;
  writeln(' введите число для сравнения ');
  READLN(C);
  Ma := 0;
  Sum := Summa(Massiv);
  if Ma=K then writeln(' элементов меньше ',C:3,' в массиве нет')
          else
          begin
          writeln(' сумма элементов массива меньше числа ',C:3);
          writeln(' равно ',Sum);
         end;
End.
```

Пример 9.10. Программа сортировки столбцов двумерного массива модифицированным методом простого выбора.

```
program nat6;
uses crt;
type
    d=array[1..4] of integer;
    b=array[1..4] of d;        {двумерный массив}
var
    c:b;
    s:d;
    i,j:byte;
  procedure sort(  var a:array of integer);  {сортировка одномерного
массива, формальный параметр а типа открытый массив  передается по
ссылке}
var
    i,j,k:byte;
    m:integer;
{i,j,k,m–локальные переменные}
begin
```

```
            {high–возвращает верхний предел диапозона значений
аргумента}
            for i:=0 to high(a) do
            begin
                m:=a[i];
                k:=i;
                for j:=i+1 to high(a) do
                 if a[j]>m then begin
                                  m:=a[j];
                                  k:=j;
                              end;
                a[k]:=a[i];
                a[i]:=m;
            end;
         end;{sort}
        {****** MAIN **************}
         begin
            writeln(' Пример : расположить по убыванию элементы  ');
            writeln('          каждого столбца двумерного массива ');
            writeln('          (элементы массива вводятся с       ');
            writeln('           помощью датчика случайных чисел). ');
            writeln(' Исходный массив: ');
            randomize;
            for i:=1 to 4 do
            begin
                for j:=1 to 4 do
                begin
                    c[j,i]:=random(100);
                    write(c[j,i],' ');
                end;
            writeln;
            end;
            for j:=1 to 4 do
            begin
            {move–копирует заданное количество последоватепльных байтов
из исходной области памяти в назначенную область}
                move(c[j],s,8);
            {вызов процедуры,где s типа массив фактический параметр}
                sort(s);
                move(s,c[j],8);
            end;
            writeln(' Измененный массив: ');
            for j:=1 to 4 do
```

```
begin
    for i:=1 to 4 do
                write(c[i,j],' ');
    writeln;
end;
end.
```

Задания для самостоятельного выполнения

А. С помощью процедур сортировки одномерного массива (параметр – одномерный массив) реализовать следующие задачи.

А.1. Расположить по возрастанию элементы каждой строки двумерного массива.

А.2. Расположить по убыванию элементы каждого столбца двумерного массива.

А.3. Расположить четные элементы двумерного массива в порядке убывания их значений.

А.4. С помощью процедуры сортировки в каждом столбце двумерного массива определить три максимальных элемента.

А.5. Расположить элементы в первой и последней строке по возрастанию, а в остальных – по убыванию значений.

В. В следующих заданиях реализация повторяющихся операций должна быть осуществлена с помощью процедур. Выходной файл должен содержать входные данные, результат выполнения каждой операции, окончательный результат решения уравнения.

В.1. Решить уравнение линейной алгебры вида:

$(A*X - beta^2 * Y) * B^2 + Z$, где

A, B – матрицы, X, Y, Z – вектора, beta - константа.

В.2. Решить уравнение линейной алгебры вида:

$X^2 * (A^T - B)^T * Y - Z$, где

A, B – матрицы, X, Y, Z – вектора, T – операция транспонирования.

В.3. Решить уравнение линейной алгебры вида:

$Y * (alfa * A^2 - B^2)^2 - X$, где

A, B – матрицы, X, Y – вектора, alfa – скаляр.

В.4. Решить уравнение линейной алгебры вида:

$(X * A^T + 1/(alfa+beta)*Y)*B^T - Z$, где

A, B – матрицы, X, Y, Z – вектора, alfa, beta – скаляры.

В.5. Решить уравнение линейной алгебры вида:

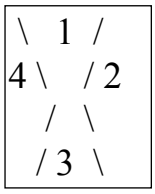
$X * (alfa / beta * A^2)^T + B^T * Y$, где

A, B – матрицы, X, Y – вектора, alfa, beta – скаляры.

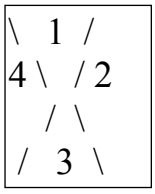
С. Все необходимые действия по определению минимальных, максимальных значений, суммы, произведения и количества, а также

формирования других значений в программах реализовать с использованием подпрограмм–функций с параметрами.

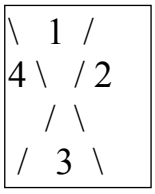
С.1. В квадратной матрице размером 5х5, заполненной случайными целыми числами из диапазона (–30,+30) в секторах 1 и 2 найти соответственно максимальный и минимальный элементы. Сектор включает диагональные элементы.



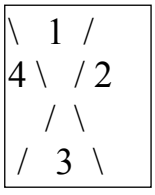
С.2. В квадратной матрице размером 5х5, заполненной случайными целыми числами из диапазона (–35,+35) в секторах 2 и 3 найти соответственно максимальный отрицательный и минимальный положительный элементы. Сектор включает диагональные элементы.



С.3. В квадратной матрице размером 5х5, заполненной случайными целыми числами из диапазона (–40,+40) в секторах 3 и 4 найти соответственно максимальный отрицательный и минимальный положительный элементы. Сектор включает диагональные элементы.



С.4. В квадратной матрице размером 5х5, заполненной случайными целыми числами из диапазона (–45,+45) в секторах 1 и 3 найти количество положительных элементов. Сектор включает диагональные элементы.



Контрольные вопросы

1. Для чего используют подпрограммы?
2. Чем отличаются описания процедур от описания функций?
3. Где задаются формальные и фактические параметры?
4. Когда применяются параметры-значения и параметры-переменные?
5. Глобальные и локальные объекты.
6. Что передается при вызове процедуры?
7. Переменные, описанные в основной программе, являются:
глобальными, внешними, автоматическими, локальными?
8. Переменные, описанные внутри процедур, являются:
локальными, автоматическими, глобальными, внешними?
9. Функции выхода из главной программы: exit, reset, assign, read.

10. МОДУЛИ

Модуль – это автономно компилируемая программная единица, содержащая любые описания. Иногда под модулем понимают библиотеки подпрограмм, которые могут быть использованы программами. Если у вас возникают проблемы с памятью при выполнении или компиляции программы, разбейте вашу программу на модули. В Turbo Pascal различают стандартные (CRT, GRAPH) и пользовательские модули.

10.1. Разработка пользовательских модулей

Рассмотрим структуру модуля.

```
  unit <идентификатор>;  { Имя модуля }

{ интерфейсная часть - открытая часть}

interface      { Начало раздела объявлений }:
uses          { Используемые при объявлении модули }
const         { Подраздел объявления доступных глобальных}
               {  констант }
type          { Подраздел объявления доступных глобальных}
               { типов }
var           { Подраздел объявления доступных глобальных}
               {переменных }
label         { Подраздел объявления доступных глобальных}
               { меток }
procedure      { Заголовки доступных процедур }
function       { Заголовки доступных функций }

{ реализационная часть – закрытая часть }

implementation { Начало раздела инициализации }:
uses          { Используемые при реализации модули }
label         { Подраздел объявления скрытых глобальных }
               {меток}
const         { Подраздел объявления скрытых глобальных }
               {констант }
type          {Подраздел объявления скрытых глобальных типов}
var           { Подраздел объявления скрытых глобальных}
               {переменных }
procedure      { описание процедур }
```

```
function       {  описание функций }

{ инициализационная часть.}

begin          { Основной блок модуля }
end.
```

Заголовок модуля специфицирует имя модуля, которое вы будете использовать для обращения к нему в разделе USES вашей программы.

В интерфейсной части описываются константы, типы, переменные, процедуры и функции доступные пользователю данного модуля.

В интерфейсной части записаны только заголовки процедур и функций. Их тела расположены в разделе реализации.

Реализационная часть содержит тела всех доступных процедур и функций.

Также в ней описаны скрытые константы, типы, переменные, процедуры и функции не доступные пользователю данного модуля.

Инициализационная часть является последним разделом модуля. Она состоит либо из зарезервированного слова END, не имеющего реализационного кода, или из последовательности операторов, выполняющихся при инициализации модуля.

Пример 10.1. Модуль, содержащий одну процедуру сортировки элементов массива, стоящих на четных местах.

```
unit lov_1m;
interface
uses crt;
type  arr=array[1..10] of integer;
procedure mas(var a:arr);

implementation
Procedure mas; {процедура сортировки mas методом парных
перестановок}
var
  s,q,i,j,k,f,m:integer;
Begin
repeat
  s:=0;
  For i:=2 to 8 do
     If i mod 2=0 then     {выбор элементов, стоящих на четных
местах}
     If a[i]>a[i+2] then begin   {сортировка элементов по
возрастанию}
```

```
                    j:=a[i];
                    a[i]:=a[i+2];
                    a[i+2]:=j;
                    s:=s+1;
                    end;
      until s=0;
       end;{mas}
      End.
```

Программа, использующая вышеописанный модуль LOV_1.PAS.

```
      Program REX;        {программа с использованием процедуры
модуля}
      uses lov_1m,crt;      {подключение модуля lov_1m}
       var
        myarray:arr;{тип arr описан в модуле lov_1m}
        i,k,q,j,s:integer;
      Begin
          clrscr;
          randomize;
              Write(' Исходная последовательность: ');
          For i:=1 to 10 do
             begin
               myarray[i]:=random(55); {заполнение массива случайными
элементами}
                Write(myarray[i]:2,' ');
             end;
          Writeln;
              Write(' Новая  последовательность:  ');
          mas(myarray);  {обращение к процедуре сортировки mas модуля
lov_1m}
          For i:=1 to 10 do Write(myarray[i]:2,' ');
       End.
```

Пример 10.2. Модуль, содержащий коонстанты и 3 процедуры.

```
      unit lov_3m;                {модуль lov_3m}
      interface
      uses crt;
       type  arr=array[1..5,1..5] of integer;
        const  fam=' Задача. Расположить элементы двумерного массива в
нечетных   ';
        const fam1='            строках по возрастанию их значений.     ';
```

```
      const fam2='     После просмотра результатов программы нажмите
<Esc>.   ';

      procedure  win1(x1,y1,x2,y2,c,b:byte);  {вывод окна }
      procedure  win2(x1,y1,x2,y2,c,b:byte);  {вывод окна с мерцающими
символами}
      procedure  mas(var g:arr);      {сортировка массива}

    implementation
      Procedure win2;               {процедура win2}
        begin
         window(x1,y1,x2,y2);          {окно}
         textbackground(c);           {цвет экрана}
         textcolor(b+blink);          {цвет мерцающего символа}
        end;
      Procedure win1;               {процедура win1}
       begin
        window(x1,y1,x2,y2);          {окно}
        textbackground(c);           {цвет экрана}
        textcolor(b);                {цвет  символа}
       end;
      Procedure mas;
        var
          i,j,k,f,m:integer;
        begin
         for i:=1 to 5 do
          if i mod 2<>0 then            {поиск нечетных строк}
                    for j:=1 to 5 do
                      begin       {сортировка элементов по }
                                  {возрастанию}
                       m:=g[i,j];
                       k:=j;
                       for f:=j+1  to 5 do
                              if g[i,f]<m then
                                begin
                                 m:=g[i,f];
                                 k:=f;
                                 end;
                        g[i,k]:=g[i,j];
                        g[i,j]:=m;
                      end;
        end;{mas}
     end.
```

Программа, использующая модуль LOV_3.PAS.

```
Program  mod_sort;
uses lov_3m;
var
  myarray:arr;
  i,j,x11,x12,y11,y12,a1,b1:byte;
Begin
      randomize;
      writeln(fam);                {печать констант fam,fam1,fam2}
   writeln(fam1);
   writeln;
   writeln(fam2);
   win1(13,15,68,22,14,15);        {установка окна win1}
   writeln(' Данная матрица');
   writeln;
   win1(13,17,68,22,7,0);          {установка окна win1}
   for i:=1 to 5 do
    for j:=1 to 5 do
    myarray[i,j]:=random(88);    {заполнение массива случайными
элементами}
   for i:=1 to 5 do
    begin
    for j:=1 to 5 do
      write(myarray[i,j]:3);         {печать элементов массива}
    writeln;
    end;
   win2(50,15,68,22,4,15);         {установка окна win2}
   writeln(' Полученная матрица');
   mas(myarray);                  {обращение к процедуре mas}
   win1(52,17,68,22,7,0);
   for i:=1 to 5 do
    begin
     for j:=1 to 5 do
      write(myarray[i,j]:3);         {печать полученной матрицы}
    writeln;
   end;
 End.
```

Пример 10.3. Модуль с разделом инициализации.

```
unit lov_4m; {модуль lov_4m использует раздел инициализации}
interface
uses crt;
```

```
  type  arr=array[1..5,1..5] of integer;
  f:text;
procedure  mas(var a:arr);

implementation

 Procedure mas; {процедура поиска минимального и максимального
элемента}
var
  i,j,Min,Max:integer;
begin                        {поиск max. и min. элементов}
   Max:=A[1,1];
   Min:=A[1,1];
   For i:=1 to 5 do
        For j:=1 to 5 do
     begin
      If A[I,j]>=Max
              then Max:=A[I,j];
      If A[i,j]<=Min
             then Min:=A[i,j];
    end;
end;
begin {раздел инициализации}
assign(f,'love.dat');
rewrite(f);
    End.
```

Программа, использующая модуль LOV_4M.

```
Program rifle; {программа  использует модуль с разделом
инициализации}
uses lov_4m;              {подключение модуля lov_4m}
  var
      A : arr;    {массив описан в модуле lov_4m}
      B :array[1..5,1..5] of integer;
      I,j:byte;
      Z,P : integer;
      Max,Min:real;
Begin
 Randomize;
 Writeln;
 Writeln(' Задача.  Заменить все отрицательные элементы '  );
 Writeln('              матрицы A(5,5)    ');
```

```
 Writeln('              на 0, если сумма Min. и Max. эл–ов этой ');
 Writeln ('             матрицы');
 Writeln('              меньше произвольного числа P,которое ');
 Writeln ('              задается с   клавиатуры.                    ');
 Writeln;
 Writeln;
 Write('          Введите любое целое число:P=');
 READLN(P);
 writeln('   Данная матрица   ');
 writeln;
 For I:=1 to 5 do
   begin
   For J:=1 to 5 do
    begin
    A[I,j]:=Random(55);            {заполнение матрицы}
    A[i,j]:=A[i,j]–50;
    Write(A[i,j]:3,' ');
    Writeln(F,A[I,j]); {запись в файл, описанный и открытый в }
                       {     модуле lov_4m}
    end;
   Close(F);
   end;
   mas(a);                    {обращение к процедуре mas}
   Reset(F);                  {открытие файла для считывания}
   writeln(' Полученная матрица ');
   writeln;
   For I:=1 to 5 do
     begin
     For j:=1 to 5 do
       begin
       READLN(F,A[I,j]);              {считывание из файла}
       If (Min+Max<P) and  (A[I,j]<0)
                  then
                  A[I,j]:=0
Write(A[I,j]:3,' ');
       end;
    writeln;
    end;
  Close(F);                   {закрытие файла}
 End.
```

Задания для самостоятельного выполнения

ТРЕБОВАНИЯ! Создайте и откомпилируйте свой модуль, в который включите:

1) описания ранее разработанных вами подпрограмм,

2) описание типов, переменных и констант, например, тип – массив, константа – назначение программы.

Составьте программу, которая использовала бы описания из вашего модуля.

10.2. Модуль CRT

Модуль CRT представляет собой библиотеку процедур и описаний, которая расширяет возможности пользователя при работе с текстом, текстовым экраном и клавиатурой. Рассмотрим некоторые из них.

1) TextMode(режим: **integer)** – выбирает указанный текстовый режим

```
 Режимы CRT:
 BW40     = 0;       { 40x25 Ч/Б на цветном адаптере }
 CO40     = 1;       { 40x25 цветной на цветном адаптере}
 BW80     = 2;       { 80x25 Ч/Б на цветном адаптере }
 CO80     = 3;       { 80x25 цветной на цветном адаптере }
 Mono     = 7;       { 80x25 на монохромном адаптере }
 Font8x8  = 256;     { Add–in for ROM font }
{ константы для установки режимов монитора }
 C40          = CO40;
 C80          = CO80;
```

2) InsLine – начиная с позиции курсора, вставляет пустую строку.

Все строки, расположенные ниже добавленной строки, перемещаются на одну строку вниз, а нижняя строка исчезает с экрана. Всем позициям новой строки присваивается значение пробела с текущими атрибутами цвета.

3) DelLine – удаляет строку, на которой находится курсор.

Все строки, расположенные ниже удаляемой, перемещаются на одну строку вверх. А внизу экрана добавляется новая строка. Данная процедура зависит от текущего окна. Значение: очищает экран и ставит курсор в позицию с координатами (1,1).

4) TextBackground (цвет: byte) – выбирает фоновый цвет.

Параметр «цвет» представляет собой выражение целого типа.

Пример 10.4.

```
uses Crt;
 begin
   { зеленые символы на черном фоне}
   TextColor(Green);
   TextBackground(Black);
   WriteLn('Hey there!');
   { мигающие ярко красные символы на сером фоне}
   TextColor(LightRed+Blink);
   TextBackground(LightGray);
   WriteLn('Hi there!');
   { желтые символы на голубом }
   TextColor(14); { Yellow = 14 }
   TextBackground(Blue);
   WriteLn('Ho there!');
   NormVideo; { оригинальный цвет }
   WriteLn('Back to normal...');
   repeat until keypressed;
 end.
```

5) ClrScr – очистка экрана.

Все позиции символов заполняются пробелами. При этом используются текущий фоновый цвет, заданный в процедуре TextBackGround.

6) TextColor (цвет: byte) – устанавливает цвет символов.

```
  { константы для установки  цвета}
  Black{черный} = 0;  Blue{синий}   = 1;  Green        = 2;
  Cyan{голубой} = 3;  Red{красный}  = 4;  Magenta      = 5;
  Brown{корич.} = 6;  LightGray     = 7;
{ константы общего цвета }
  DarkGray      = 8;  LightBlue     = 9;  LightGreen    = 10;
  LightCyan     = 11;  LightRed     = 12;  LightMagenta  = 13;
  Yellow        = 14;  White        = 15;  Blink        = 128;
```

7) Window(x1, y1, x2, y2) – определяет на экране текстовое окно.

Параметры x1, y1 представляют собой координаты верхнего левого угла окна, параметры x2, y2 – координаты правого нижнего угла. Минимальный размер – один столбец на одну строку.

Если координаты являются недопустимыми, то обращение к процедуре Window игнорируется. По умолчанию задается окно на весь экран.

Рассмотрим пример 10.5. Вывод в окна в программе поиска суммы четных элементов ниже побочной диагонали.

Пример 10.5.

```
program mas3;
uses crt;

const    a:array[1..5,1..5] of integer=((0,1,2,3,4),
                                        (5,6,7,8,9),
                                        (0,1,2,3,4),
                                        (5,6,7,8,9),
                                        (0,1,2,3,4));
var
        i,j:byte;
        k:integer;
begin
        textbackground(0);
        clrscr;
        textcolor(5);
        window(5,5,21,12);
        textbackground(3);
        clrscr;
        writeln('Дана матрица:');
        for i:=1 to 5 do
           begin
                for j:=1 to 5 do
                   begin
                        if  (i+j>6) and (a[i,j] mod 2=0) AND (A[i,j]<>0)
                          THEN   textcolor(4)
                           else      textcolor(0);
                        write(a[i,j]:3);
                   end;
                writeln;
           end;
        textcolor(0);
        writeln;
        window(26,10,78,13);

        textbackground(13);
        clrscr;
        writeln(' Задание:найти сумму четных
          элементов,расположенных');
        writeln(' ниже побочной диагонали.');
        k:=0;
```

```
        for i:=1 to 5 do
        for j:=1 to 5 do
        if (i+j>6) and (a[i,j] mod 2=0) then  k:=k+a[i,j];
        writeln;
        window(50,19,65,20);
        clrscr;
        write('Результат  ');
        writeln(k);
        READLN;
end.
```

8) GoToXY(x, y: byte) – позиционирует курсор.

Курсор перемещается в ту позицию внутри текущего окна, которая задана координатами x и y (x задает столбец, y задает строку). Верхний левый угол задается координатами (1,1)

пример:
```
Window(1,10,60,20);
GoToXY(1,1);
```

Это приведет к тому, что курсор уйдет в точку с абсолютными координатами (1,10).

9) WhereX и **WhereY** возвращают для текущей позиции курсора относительно текущего окна координаты X и Y соответственно.

Тип результата Byte.

10) Delay(мсек : word) – выполняет задержку на заданное число миллисекунд.

Параметр «мсек» задает число миллисекунд интервала ожидания.

Но данная процедура является приблизительной, поэтому период задержки не будет точно равняться заданному числу миллисекунд.

11) READKey – считывает символ с клавиатуры.

Считываемый символ не отображается на экране. Если перед обращением к функции READKey функция KeyPressed имела значение TRUE, то символ считывается немедленно, в противном случае функция ожидает нажатия клавиши. При нажатии спецальных клавиш функция READKey возвращает сначала нулевой символ (0), а затем расширенный скен–код клавиши.

Программа вывода ASCII кода алфавитно цифровых клавиш клавиатуры

```
 uses Crt;
var
 C: Char;
```

```
begin
  Writeln('Please press a key');
  C := READkey;
  Writeln(' You pressed ', C, ', whose ASCII value is ', Ord(C), '.');
end.
```

12) KeyPressed – возвращает значение TRUE, если на клавиатуре нажата клавиша и FALSE – в противном случае.

Примечание: символ (или символы) остаются в буфере клавиатуры. Данная процедура не распознает клавиш перевода регистра, таких как Shift, Alt, NumLock и т.д.

Пример использования KeyPressed

```
uses Crt;
begin
  repeat
    Write('Xx');       { выводит надпись Xx пока не будет }
  until KeyPressed;    { нажата какая–либо клавиша }
end.
```

13) Sound – является процедурой, включающей внутренний динамик.

Описание: Sound(герц: word); где параметр «герц» задает частоту генерируемого сигнала в герцах. Звук будет звучать до тех пор, пока не будет выключен посредством обращения к процедуре NoSound;

14) NoSound – выключает внутренний динамик.

Пример использования Sound, Delay, NoSound.

```
uses Crt;
begin
   Sound(220);  { включить звук  }
   Delay(300);  { ждать 300 ms   }
   NoSound;     { выключить звук }
end.
```

Задания для самостоятельного выполнения

1. Разработать функцию, реализующую запрос одной из двух альтернатив «ДА – НЕТ» и возвращающую номер выбранной альтернативы. («ДА» – 1, «НЕТ» – 2»).Запрос организовать в форме вертикального меню в центре экрана с выбором с помощью навигационной клавиатуры.
2. Разработать функцию, входными данными для которой является массив из N строк. Функция должна вывести на экран строки в виде вертикального меню (координаты верхнего левого угла меню фиксированы). Выбор

нужной строки осуществить с помощью навигационной клавиатуры. Функция должна возвращать номер выбранной строки.
3. Разработать функцию, входными данными для которой является массив из N строк. Функция должна вывести на экран строки в виде горизонтального меню (координаты верхнего левого угла меню фиксированы). Выбор нужной строки осуществить с помощью первой буквы этой строки. Функция должна возвращать номер выбранной строки.
4. Разработать процедуру, организующую вывод в центр экрана «всплывающего» окна до заданных пределов. Процесс «всплытия» должен происходить с замедлением по времени (обеспечивающим наблюдение за ним). После завершения формирования окна вывести в него произвольную строку. Входными данными для процедуры являются предельные размеры окна и строка, выводимая в него.
5. Разработать процедуру, входными данными для которой является массив из 5 строк вида: «F1 – XXXX», «F2 – XXX»,..., «F5 – XXX», где XXX – произвольная комбинация символов.

Процедура должна вывести на экран строки в виде горизонтального меню (координаты верхнего левого угла меню фиксированы). Выбор нужной строки осуществить с помощью выбора нужной функциональной клавиши. Процедура должна возвращать номер выбранной строки.

10.3. Модуль GRAPH

В персональных компьютерах используется растровый способ генерации изображений. Это означает, что изображение формируется из ряда светящихся точек – пикселей. Каждый пиксель задается парой координат. Для различных графических режимов работы координатные сетки могут быть разными. Начало координат всегда находится в левом верхнем углу экрана; координата левого верхнего угла (0,0). Состояние каждого пикселя (точнее его цвет) кодируется несколькими байтами, хранящимися в видеопамяти. Специальные программы–драйверы управляют работой адаптера, т. е. определяют, каким образом в процессе формирования изображения будут интерпретироваться данные, расположенные в видеопамяти. Тип драйвера должен обязательно соответствовать типу адаптера. Приведем список наиболее распространенных адаптеров.

Адаптер	Режим	Драйвер	Разрешение	Число цветов
CGA	0–3	CGA	320*200	4
EGA	0	EGALo	640*200	16
EGA	1	EGAHi	640*350	16

VGA	0	VGALo	640*200	16
VGA	1	VGAMod	640*350	16
VGA	2	VGAHi	640*480	16

Для перехода из текстового режима в графический в программе необходимо использовать процедуру открытия графического режима, INITGRAPH, которая

1) определяет графический адаптер, установленный на компьютере;

2) загружает и инициализирует соответствующий графический драйвер;

3) переводит адаптер в соответствующий графический режим и

4) возвращает управление вызывающей программе.

Например:

```
Uses  GRAPH;
Var
 Driver,Regim:integer;
Begin
 Driver:=Vga;
 Regim:= 1;

InitGraph(Driver,Regim,'a:/bgi');
.....
closeGraph;
{или}
Driver:=detect;
Initgraph(Driver,Regim,'');
.....
CloseGraph;
End.
```

Процедура CloseGraph

1) завершает работу в графическом режиме;

2) выгружает драйвер из памяти;

3) восстанавливает текстовый режим.

При работе с графикой необходимо наличие в конфигурации системы Turbo Pascal одного или нескольких программ–драйверов (файлов с расширением .BGI), графических шрифтов (файлов с расширением .CHR) и модуля Graph, в котором содержится множество графических функций и процедур. Рассмотрим применение некоторых из них.

В модуле Graph для отображения точки на экране используется процедура

PutPixel(X,Y:integer;Color:word),

где X и Y – экранные координаты точки, Color – ее цвет.

Рассмотрим пример вывода множества точек на экран.

В первых строках после оператора Begin идут операторы инициализации графики и проверка правильности их выполнения. X, Color имеют тип Longint. X берется достаточно большим, а затем постоянно уменьшается, этим и достигается тот эффект, который виден на экране после выполнения программы.

```
uses Crt, Graph;
    var
      Gd, Gm: Integer;
       x, Color:longint;
 begin
   Gd := Detect;                {автоопределение}
   InitGraph(Gd, Gm, '');           {инициализация графики}
   if GraphResult <> grOk then     {проверка инициализации графики}
         Halt(1);              {прекращение выполнения программы}
   x:=100000;
  repeat                        {повторять}
   color:=color+3;
   PutPixel(Random(x div color), Random(x div color), Color div 13);
  until KeyPressed;               {до нажатия клавиши}
  settextstyle(3,0,4);            {установка параметров текста}
  outtextxy(250,400,'press enter');  {вывод текста}
  setcolor(14);                  {установка желтого цвета }
   outtextxy(200,290,'Procedure PutPixel');      {вывод текста}
  READLN;
  CloseGraph;              {прекращение работы в графическом режиме}
 end.
```

Для изображения окружностей используется процедура

```
Circle(x,y:INTEGER;radius:WORD);
```

Здесь (X,Y) – координаты центра окружности, Radius – ее радиус.

Пример построения окружности со случайными координатами и радиусом.

```
uses Graph,crt;
  var
    Gd, Gm: Integer;
    x,y:integer;
```

```
  Radius: Integer;
 begin
  Gd := Detect;          {автоматическая выборка режима}
  InitGraph(Gd, Gm, ''); {инициализация графики}
  if GraphResult <> grOk then   {проверка инициализации,если}
                                { ошибка то}
  Halt(1);               {прекращение выполнения программы}
  repeat
   x:=x+1;
   y:=y+1;
   Radius:=Radius+1;
   setcolor(random(15));      {задание случайного цвета}
   Circle(random(x*7),random(y*5), Radius) ;    {рисует окруж}
                                                {ности со слу–}
  until keypressed;                             {чайными координатами}
  settextstyle(3,0,4);        {установка параметров текста}
  outtextxy(250,400,'press enter');      {вывод текста}
  setcolor(14);               {установка желтого цвета }
  outtextxy(200,290,'Procedure Circle');     {   вывод текста}
  readln;
  CloseGraph;
 end.
```

Для построения эллиптических дуг предназначена процедура

Ellipse (X, Y: integer; StAngle, EndAngle: word; XR, YR: word),

где (X, Y) – центр эллипса в дисплейных координатах, XR и YR – горизонтальная и вертикальная оси. Дуга эллипса вычерчивается текущим цветом от начального угла StAngle до конечного угла EndAngle. Значения StAngle=0 и EndAngle=359 приведут к вычерчиванию полного эллипса. В следующем примере процедура Ellipse(100, 100, 0, 360, 30, 50) строит полный эллипс, а процедура Ellipse(100, 100, 0, 460, 30, 100) примерно половину.

Пример построения эллипсов

```
 Program Mih;
 uses Graph,Crt;
   var Gd, Gm,x,y: Integer;
 begin
  Gd := Detect;               {автоматическая выборка режима}
  InitGraph(Gd, Gm, '');      {инициализация графики}
  if GraphResult <> grOk then   {проверка на ошибку инициализации}
       Halt(1);               {прекращение выполнения программы}
  x:=1;
  y:=1;
  repeat
```

```
    setcolor(random(16));        {выбор случайного цвета}
    Ellipse(x, y, 0, 360, 30, 50);
    x:=x+1;
   y:=y+1;
  until y>=480;              {до тех пор пока y не
                          выйдет за границу экрана}
  settextstyle(3,0,4);        {установка параметров текста}
  outtextxy(400,200,'press enter');      {вывод текста}
  setcolor(14);               {установка желтого цвета }
  outtextxy(350,100,'Procedure ELLIPSE');      {вывод текста}
  readln;
  CloseGraph;                 {выход из графического режима}
end.
```

Процедура вывода отрезка прямой на экран (в текущем цвете и стиле) определена следующим образом:

Line(x1,y1,x2,y2:integer).

В ней задаются координаты начальной (x1, y1) и конечной точек (x2, y2) линии. Применяется только в графическом режиме. Рассмотрим программу построения линий.

В первых строках, после оператора Begin идут операторы инициализации графики и проверка правильности их выполнения. Процедура l(d,c:byte) собственно и производит построение получаемой картины на экране. Процедура начинается тем, что устанавливаются координаты x1, y1, x2, y2 линии. Координаты соответствуют правому и левому нижним углам. Затем координата левого угла (по y) начинает уменьшаться, каждый раз на d пикселов (текущий цвет – c). Примерно то же самое для остальных углов. В результате вызова процедуры в главной программе экран заполняется множеством линий, причем различного цвета и на разные промежутки.

Пример построения линий

```
 uses graph,crt;
  procedure l(d,c:byte);
    var   x1,y1,x2,y2:integer;
  begin
    x1:=639;
    y1:=479;
    x2:=1;
    y2:=479;
    repeat
      setcolor(c);
      y2:=y2–d;
```

```
          line(x1,y1,x2,y2);
        until y2<=1;
        x1:=1;
        y1:=1;
        x2:=639;
        y2:=479;
        repeat
          setcolor(c);
          line(x1,y1,x2,y2);
          y2:=y2–d;
        until y2<=1;
        x1:=639;
        y1:=1;
       x2:=1;
       y2:=479;
       repeat
        setcolor(c);{ задание цвета линий }
        line(x1,y1,x2,y2);
        y2:=y2–d;
       until y2<=1;
       x1:=1;
       y1:=479;
       x2:=639;
       y2:=1;
       repeat
        setcolor(c);{ задание цвета линий }
        line(x1,y1,x2,y2);
        x2:=x2–d;
       until x2<=1;
     end;{l}

     {**********************MAIN*********}
     var
    driver,mode:integer;
     begin
          driver:=vga;                {драйвер}
          mode:=vgahi;                 {режим графики}
          initgraph(driver,mode,");      {инициализация графики}
          if graphresult<>0 then       {проверка на ошибку инициализации
графики }
                halt(1);             {прекращение выполнения программы}
          l(8,2);
          l(15,14);
```

```
    setcolor(11);              {установка желтого цвета }
    settextstyle(3,0,4);       {установка параметров текста}
    outtextxy(250,400,'press enter');   {вывод текста}
    outtextxy(200,290,'Procedure Line');  {вывод текста}
    readln;                    {ждет нажатия ввода}
    closegraph;                {выход из графического режима}
end.
```

Для построения отрезков применяются еще две процедуры (кроме line):

LineTo и LineRel.

Процедура LineTo(x,y) строит отрезок из точки текущего положения указателя до точки с координатами (x,y). Процедура LineRel(dx,dy) проводит отрезок от точки текущего положения указателя до точки (cpx + dx,cpy + dy), где cpx и cpy – текущие координаты ср.

В данном примере экран заполняется множеством отрезков различных цветов, при этом получается интересный визуальный эффект (GetmaxX, GetmaxY – определение значений максимальных координат экрана).

Пример построения линий (LineTo,LineRel)

```
program wind;
uses Crt, Graph;
 var
   Gd, Gm: Integer;
 begin
  Gd := Detect;              {автоопределение}
  InitGraph(Gd, Gm, '');     {инициализация графики}
  if GraphResult <> grOk then  {проверка ошибки инициализации графики}
Halt(1);          {прекращение выполнения программы}
    Randomize;        {при каждом запуске получается новая картина}
  repeat
   setcolor(random(15));         {случайный выбор цвета}
   delay(100);                   {задержка на 100 мс.}
   lineto(random(getmaxx),random(getmaxy));
   linerel(50,–50);
   until KeyPressed;             {пока не нажата клавиша}
   settextstyle(3,0,4);          {установка параметров текста}
   outtextxy(250,400,'press enter');   {вывод текста}
   setcolor(14);                 {установка желтого цвета }
   outtextxy(150,290,'Procedures LineTo and LineRel');  {вывод текста}
   readln;
  CloseGraph;                    {выход из графики}
```

```
end.
```

Для построения прямоугольников используется процедура построения прямоугольника на плоскости:

```
Rectangle(x1,y1,x2,y2:integer),
```

где x1, y1 – координаты левого верхнего угла, x2, y2 – координаты правого нижнего угла прямоугольника. Это очень полезная процедура, с ее помощью, в частности, можно легко построить диаграмму для визуального анализа данных.

Пример построения прямоугольников

```
Program Mih6;
 uses graph,crt;
 var x1,y1,x2,y2,d,j,k:integer;
     Gd, Gm: Integer;
 begin
   Gd := Detect;                  {автоопределение}
InitGraph(Gd, Gm, '');           {установка графического режима}
  if GraphResult <> grOk then       {проверка инициализации графики}
            Halt(1);             {прекращение выполнения прграммы}
  cleardevice;                    {очистка окна}
  x1:=640;
  y1:=480;
 x2:=1;
 y2:=1;
 repeat
   setcolor(yellow);                {установка цвета}
   rectangle(x1,y1,x2,y2);
  x1:=x1–2;
  y1:=y1–2;
  x2:=x2+2;
  y2:=y2+2;
 until x1<=1;
 setcolor(1);
 settextstyle(3,0,4);               {установка параметров текста}
 outtextxy(250,400,'press enter');      {вывод текста}
 outtextxy(200,290,'Procedure Rectangle' );  {вывод текста}
 READLN;
 closegraph;{выход из графики}
end.
```

В модуле Graph имеется множество средств построения различных геометрических фигур, которые позволяют сделать выводимое изображение более красочным и эффективным. Среди таких средств прежде всего следует отметить «заливку» замкнутых областей экрана с помощью различных узоров. В модуль Graph включен ряд стандартных шаблонов различных узоров для заполнения внутренних и внешних областей различных геометрических фигур. Узор может быть окрашен в допустимые для установленной палитры цвета. Комбинацию узор–цвет принято называть **стилем заполнения**. Для работы со стандартными стилями используются рассматриваемые ниже функции GetFillSettings и SetFillStyle.

SetFillStyle (Pattern: WORD; Color: WORD)

устанавливает маску Pattern и ее цвет Color, т. е. определяет стиль заполнения.

Процедура

GetFillSettings(VAR FillType: FillSettingsType)

возвращает в переменной FillType, которая имеет тип, объявленный в модуле Graph следующим образом:

```
TYPE
FillSettingsType = RECORD
        Pattern: WORD; {шаблон}
        Color: WORD; {цвет}
        END;
```

номер шаблона заполнения (Pattern) и его цвет (Color).

Осталось рассмотреть последнюю процедуру этого семейства – FloodFill. Она служит для заполнения заданным с помощью SetFillStyle и SetFillPattern стилем области, расположенной либо внутри замкнутого контура, либо вне его. Для ее задания используется следующий синтаксис:

FloodFill (X, Y: INTEGER; Border: WORD),

где X, Y – координаты точки внутри или вне замкнутого контура. Параметр Border задает цвет контура. В зависимости от расположения указанной точки по отношению к контуру будет производиться заполнение текущим узором либо области, ограниченной контуром, либо части экрана, расположенной вне границ контура. Если указанный контур не является замкнутым (отметим, что контуры, включающие в себя часть границы экрана, относятся к замкнутым), то будет заполнен весь экран. Использование шаблонов с достаточно редким заполнением для

небольших по размеру областей может привести к некорректной работе процедуры.

Процедра

SetTextStyle(Font:word;direction:Word;Charsize:Word) устанавливает текущие тип шрифта, направление текста и размер символов.

Font – тип шрифта;

Direction – направление текста;

CharSize – размер символов.

В случае аварийной ситуации процедура устанавливает код ошибки :

-8 – не найден файл со шрифтом;

-9 – нет памяти для загрузки шрифта;

-11 – ошибка графической системы;

-12 – ошибка ввода–вывода графической системы;

-13 – ошибка в файле со шрифтом;

-14 – неверный номер шрифта.

Процедура OutText(TextString:String) выводит на экран строку текста, начиная с текущего положения курсора.

Пример построения закрашенных областей на экране

```
 uses graph,crt;
 var
      p:fillpatterntype;{маска заполнения}
      driver,mode,i:integer;
 begin
   driver:=vga;                    {драйвер}
   mode:=vgahi;                     {режим графики}
   Initgraph(driver,mode,");          {инициализация графики}
   if graphresult<>0 then       {проверка инициализации графики}
        halt(1);            {прекращение выполнения программы}
   for i:=1 to 8 do begin
     p[1]:=7;  {            в результате            }
     p[2]:=221;{          выполнения              }
     p[3]:=7;  {                                  }
     p[4]:=10; {       этих операторов на         }
     p[5]:=114;{             экране               }
     p[6]:=80; {            появится              }
     p[7]:=114;{           множество              }
     p[8]:=130;{           кружочков              }
     end;
   line(10,10,630,10) ;             {вычерчивет границы рамки }
```

```
  line(10,10,10,470);              {вычерчивет границы рамки }
  line(10,470,630,470);            {вычерчивет границы рамки }
  line(30,30,610,30);              {вычерчивет границы рамки }
  line(30,460,610,460);            {вычерчивет границы рамки }
  line(30,460,610,460);            {вычерчивет границы рамки }
  line(30,30,30,460);              {вычерчивет границы рамки }
  line(610,30,610,460);            {вычерчивет границы рамки }
  setfillstyle(9,7);
  floodfill(20,25,15);
  setlinestyle(0,0,1);
  rectangle(45,45,595,445);        {построение прямоугольника}
  setfillstyle(1,8);    {задает стандартный орнамент и цвет
                    заполнения фигур}
  floodfill(85,34,15);     {Закрашивает область,ограниченную
                     непрерывной линией,}
  setfillpattern(p,15);    {текущим орнаментом и цветом заполнения}
  bar(45,45,595,445);      {построение прямоугольника}
  setcolor(12);            {установка цвета}
  settextstyle(1,0,12);    {установка шрифта}
  outtextxy(55,170,'Pmd–11');      {вывод текста}
  settextstyle(3,0,4);             {установка параметров текста}
  outtextxy(250,400,'press enter');  {вывод текста}
  setcolor(14);                    {установка желтого цвета }
 outtextxy(150,435,'Procedure SetFillPattern');     {вывод текста}
 READLN;
end.
```

Пример вывода текста на экран

```
{Процедура  SetTextStyle}
uses graph,crt,font;
var
     driver,mode:integer;
     c,g,r:byte;
    k:char;
    s:string;
begin
   driver:=vga;                    {драйвер}
   mode:=vgahi;                    {режим графики}
   initgraph(driver,mode,");       {инициализация графики}
   if graphresult<>0 then      {проверка инициализации графики}
      halt(1);             {прекращение выполнения программы}
   settextstyle(4,0,2);
   setcolor(14);                   {задание цвета}
```

```
  outtextxy(100,200,'Введите номер шрифта (от 1 до 7)');
  READLN(g);
  cleardevice;                    {очистка окна}
  outtextxy(100,200,'Введите номер цвета (от 0 до 15)');
  READLN(c);
  cleardevice;
  outtextxy(30,200,'Введите текст (желательно латинскими
                                   буквами)');
  READLN(s);
  cleardevice;
  outtextxy(90,200,'Введите размер текста (от 1 до 10 ) ');
  READLN(r);
  cleardevice;
  outtextxy(10,200,'Вывод вертикальный(Y) или
                    горизонтальный (N) ?' );
  setcolor(9);
  outtextxy(100,300,'       Нажмите на Y или на N');
  k:= READkey;           {запоминает введенный с клавиатуры
                         символ}
  if k = 'y'then  settextstyle(g,1,r)     {вертикальный вывод}
          else settextstyle(g,0,r);     {горизонтальный вывод}
 cleardevice;                          {очистка окна}
 moveto(100,100);                      {перемещение курсора в точку с
                                       координатами 100,100}
 setcolor(c);          {задание цвета}
 outtext(s);
 settextstyle(3,0,4);    {установка параметров текста}
 outtextxy(250,400,'press enter');     {вывод текста}
 setcolor(14);          {установка желтого цвета }
 outtextxy(200,290,'Procedure  SetTextStyle');     {вывод текста}
 READLN;
 closegraph;           {прекращение работы в графическом режиме}
end.
```

Контрольные вопросы

1. Что понимают под модулем?
2. Из каких частей состоит модуль?
3. Когда возникает необходимость использовать модули?
4. Типы модулей.
5. Назовите стандартные модули Turbo Pascal и опишите их назначение.
6. Правила использования модулей в программах.
7. Какой объем памяти отводится под программу, а какой – под модуль?
8. Какое расширение имеет имя файла модуля?

11. ИСПОЛЬЗОВАНИЕ ЗАПИСЕЙ

Запись – это структура данных, состоящая из фиксированного числа компонентов, называемых **полями записи**. С помощью записи (record) представляется некоторая структура статических данных. Тип данных record предоставляет программисту возможность объединить в одну связную структуру различные по типу и смыслу элементы (поля). Причем элементами записи могут быть и структурированные типы данных, например, массивы и другие (подчиненные) записи.

Для обработки доступна как вся запись целиком, так и отдельные ее поля.

Под структурой данных обычно понимают данные, объединенные в упорядоченное множество. Особо удобны записи при обработке взаимо-связанных разнородных данных.

В качестве примера приведем запись, представляющую информацию из адресного справочника:

```
var reference_book = record
                       surname,
                       name,
                     address,
                     city       : string[20];
                     post_index  : string[4];
                     telerhone   : string[12];
                   end;
```

При обращении к отдельным полям указывается имя всей записи и имя отдельного поля, причем они разделяются точкой. Так, с помощью операции

```
reference_book. surname :='Иванов';
      ↑              ↑
 имя записи        поле
```

вы присваиваете полю surname записи reference_book значение 'Иванов'. Таким способом можно обратиться к любому полю, любой записи, как для занесения туда некоторого значения, так и для извлечения необхо-димой информации. Следует только применять эти операции с учетом типа данных обрабатываемой записи.

При оформлении записей удобно использовать псевдографику.

Пример 11.1. Ввод/вывод записей.

```
program ky1;
    uses  crt;
    type
      books=record
```

```
            name:string;
            after:string;
            izd:string;
            tom:integer;
          end;
        var
          lib:array [1..3] of books;
          i,j:integer;
        begin
         {Ввод записей}
       writeln('Ввести 3 раза название книги, фамилию автора, ');
       writeln('издательство, число томов');
        writeln('|———————————|———————————|———————————|———————————|');
        writeln('| Название книги  | ФИО автора  | Издательство  | Кол. томов  |');
        writeln('|———————————|———————————|———————————|———————————|');
          j:=0;
         for i:=1 to 3 do
          begin
                  gotoxy(1,6+j);
                  write('|');
          READLN(lib[i].name);
                 gotoxy(16,6+j);
                 write('|');
          READLN(lib[i].after);
                 gotoxy(31,6+j);
                 write('|');
          READLN(lib[i].izd);
                gotoxy(45,6+j);
                write('|');
          READLN(lib[i].tom);
                 gotoxy(58,6+j);
                 write('|');
          j:=j+2;
          if i<=3 then
                  writeln;
writeln('|———————————|———————————|———————————|———————————|');
if i=3 then
writeln('|———————————|———————————|———————————|———————————|');
         end;
          write;
{Вывод записей на экран}
         writeln('              информация о книгах ');
```

```
        writeln('|———————————|———————————|———————————|———————————|');
        writeln('| Название книги  | ФИО автора  | Издательство  | Кол. томов  |');
        writeln('|———————————|———————————|———————————|———————————|');
        j:=11;
          for i:=1 to 3 do
          begin
              gotoxy(1,6+j);
              write('|');
          writeln(lib[i].name);
                gotoxy(16,6+j);
                write('|');
           writeln(lib[i].after);
                gotoxy(31,6+j);
                write('|');
          writeln(lib[i].izd);
                gotoxy(45,6+j);
                write('|');
          writeln(lib[i].tom);
                gotoxy(58,6+j);
                write('|');
         j:=j+2;
         if i<=3 then
                writeln;
       writeln('|———————————|———————————|———————————|———————————|');
    if i=3 then
       writeln('|———————————|———————————|———————————|———————————|');
    end;
       READLN;
     end.
```

Пример 11.2. Удаление из массива записей записи о студентке Николаевой.

```
program nat8;
uses crt;
type
    typzap=record
    fam:string;
    gr:integer;
    sb:byte;
    end;
var
    gruppa:array [1..4] of typzap;
    i,j:byte;
```

```
  procedure vud(zap:typzap);
  {параметр zap типа запись передается по значению}
  begin
         window(43,j,52,j);
          write(zap.fam);
          window(56,j,63,j);
          write(zap.gr);
          window(69,j,76,j);
          write(zap.sb);
          j:=j+1;
   end;{vud}
  begin
      window(1,1,80,25);
      textbackground(0);
      textcolor(15);
      clrscr;
      writeln(' Пример :удаление записи о выбывшей     ');
      writeln('                  студентке Николаевой ');
      writeln('          (переменным  присваиваются     ');
      writeln('                  значения в программе).');
      gruppa[1].fam:=' Атурова ';
      gruppa[1].gr:=1977;
      gruppa[1].sb:=4;
      gruppa[2].fam:='Вострецов';
      gruppa[2].gr:=1978;
      gruppa[2].sb:=4;
      gruppa[3].fam:='Николаева';
      gruppa[3].gr:=1978;
      gruppa[3].sb:=4;
      gruppa[4].fam:='Тутурина';
      gruppa[4].gr:=1978;
      gruppa[4].sb:=4;
      window(4,5,37,6);
      writeln('———————— ———————— ————————————————');
      window(4,7,37,8);
      writeln(' фамилия  год рожд.  средний балл');
      window(4,9,37,10);
      writeln('———————— ———————— ————————————————');
j:=10;
      for i:=1 to 4 do with gruppa[i] do
      begin
          window(4,j,13,j);
          write(fam);
```

```
            window(17,j,24,j);
            write(gr);
            window(30,j,37,j);
            write(sb);
            j:=j+1;
        end;
        window(43,5,76,6);
        writeln('————————— ————————— ————————————————————');
 window(43,7,76,8);
        writeln(' фамилия  год рожд.  средний балл');
        window(43,9,76,10);
        writeln('————————— ————————— ————————————————————');
 j:=10;
         for i:=1 to 3 do
           begin
           if gruppa[i].fam = 'Николаева' then
                    gruppa[i]:=gruppa[i+1];{удаление}
           vud(gruppa[i]);{вывод на экран результата}
           end;
        READkey;
   end.
```

Задания для самостоятельного выполнения

1. Сформировать массив записей, описывающих следующую таблицу:

ФИО студента	Группа	Число пропусков занятий за год		
		по болезни	по другим причинам	итого

Вывести фамилии студентов, имеющих более ста часов сумарных пропусков, по форме:

ФИО студента	Число часов пропусков

2. Сформировать массив записей, описывающих следующую таблицу:

Фамилия	Адрес	Номер телефона

Вывести информацию о телефонных абонентах, проживающих по улице Гончарова, в виде:

Фамилия	Номер телефона

3. Сформировать массив записей, описывающих следующую таблицу:

Фамлия	Адрес		
	Улица	Дом	Квартира

Удалить фамилии жильцов дома №1 по улице Минаева по форме:

Фамилия	Номер квартиры

4. Ввести из файла массив записей, описывающих следующую таблицу:

Фамилия	Адрес	Число членов семьи	Занимаемая жилая площадь

Вывести фамилии жильцов, в семьях которых жилая площадь на одного человека не превышает 9 кв.м. Результат вывести в виде:

Фамилия	Жилая площадь	Площадь на одного человека

5. Сформировать массив записей, описывающих следующую таблицу:

ФИО	Место работы	Домашний адрес	Стаж работы

Вывести фамилии работников, имеющих стаж более 10 лет и работающих на УАЗе, по форме:

ФИО	Стаж работы

6. Сформировать массив записей, описывающих следующую таблицу:

Наименование кафедры	Фамилия преподавателя	Ученое звание	Стаж работы

Вывести список преподавателей-доцентов с кафедры «Прикладная математика» по форме:

Фамилия преподавателя	Стаж работы

7. Сформировать массив записей, описывающих следующую таблицу:

Номер поезда	Маршрут	Время отправления	Время прибытия

Записать в текстовый файл информацию о поездах, отправляющихся из Москвы, по форме:

Номер поезда	Время отправления

Контрольные вопросы

1. Какова структура объявления типа записи?
2. Могут ли компоненты записи быть различных типов?
3. Могут ли компоненты одной записи иметь одинаковые имена?
4. Как обращаются к компонентам записи?
5. Какие операции можно выполнять над компонентами записи?
6. Отличие записей от массивов.

12. УКАЗАТЕЛИ И ДИНАМИЧЕСКАЯ ПАМЯТЬ

Все переменные, объявленные в программе, размещаются в оперативной памяти в непрерывном сегменте, размер которого не может превышать 64 кБайт. Для размещения данных, требующих больший размер памяти, или для данных, количество которых неизвестно заранее и может модифицироваться в программе (массивы с динамическими размерами, переменными) используется механизм управления динамической памятью (ДП).

ДП – это оперативная память компьютера, предоставляемая программе за вычетом сегмента в 64 кБайта (для статических переменных, стека 16 кБайт) и соответственно кода тела программы.

Для управления ДП, с целью изменения ее размеров, используется следующая директива компилятора:

{ $M,<стек>,<min ДП>,<max ДП> }

По умолчанию предполагается существование следующей директивы:

{ $M,16384,0,655360 }

Для управления данными ДП в TURBO PASKAL существует гибкое средство – указатели.

Указатель – это переменная, которая в качестве своего значения содержит адрес некоторого байта памяти.

Адрес байта оперативной памяти задается совокупностью двух 16-разрядных слов, которые называются **сегментом** и **смещением**.

Сегмент – это участок оперативной памяти длиной в 64 кбайта, который начинается с адреса, кратного 16.

Смещение – указывает, сколько байт от начала сегмента надо пропустить, чтобы обратиться к нужному адресу. Вообще можно узнать каждый адрес переменной с помощью операции взятия адреса *@*.

Таким образом, по своей структуре любой указатель представляет совокупность двух слов (данных типа WORD). С помощью указателя ДП можно разместить данные любого типа. Для описания указателей используются ссылочные типы.

Как правило, в TP указатель связывается с некоторым типом данных (ссылочным типом), такой указатель называется **типизированным**.

Для объявления типизированного указателя используется значок ^ и имя типа.

Пример:

```
var
  p1,p2:^byte;
  k:byte;
```

Для того, чтобы присвоить переменной ссылочного типа некоторое значение, необходимо следовать правилам:

1) если присваивается переменной ссылочного типа значение того же ссылочного типа, то они должны принадлежать одному и тому же типу.

Например:

```
  begin
   .......
   p2:=p1;
   p1:=k;невозможно
```

2) для реализации присваивания ссылочной переменной некоторого значения, не являющегося указателем, необходимо воспользоваться операцией взятия адреса @.

Пример:

```
   var i:byte;
   .............
   begin p1:=@i;
        p2:=nil;
```

p1 присваивается адрес,

где хранится переменная, определяемая идентификатором i;

nil – это совместимое по типу с любым указателем значение, которое означает указатель в никуда.

Вся ДП может рассматриваться как сплошной массив, этот массив называется **кучей**. Адрес начала кучи хранится в стандартной переменной

HEAPORG,

конец кучи хранится в стандартной переменной

HEAPEND.

Границу кучи указывает стандартная переменная HEAPPTR.

Под динамически размещаемую переменную во время выполнения программы, память выделяется с помощью процедуры

NEW(P); P – указатель.

В результате обращения к этой процедуре указатель P приобретает значение, соответствующее адресу, начиная с которого могут размещаться

данные соответствующего указателю типу, кроме того, в ДП отводится место или область для хранения этих данных.

В связи с процедурой New возникает важная проблема исчерпания ДП.

Стандартная функция Maxavail возвращает максимальный размер непрерывного участка ДП.

Функция Sizeof (<переменная или ее тип>) возвращает число байт необходимых для хранения переменной.

Функция Memavail – суммарный размер всех свободных областей ДП.

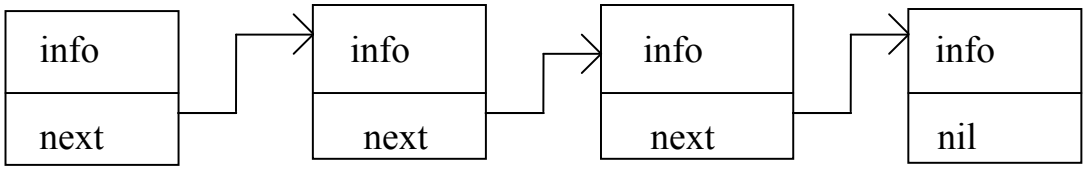
Для освобождения ДП во время работы программы используют функцию Dispose с параметром указателем на динамическую переменную, причем эта переменная должна быть ранее размещена в куче.

```
Var
P: ^Person;
.....
begin
      new(p);
      Действия
      Dispose(p)
end.
```

12.1. Работа со списком

Односвязный список – это динамическая структура, каждый элемент которой содержит информационную (info) и адресную (next) компоненты. Адресная компонента указывает на адрес следующего элемента списка.

Рассмотрим односвязный список



который объявлен в программе так:

```
type   ptr = ^zap;
       zap = record;
             info:string[35];
```

```
                next:ptr;
                 end;
var
  headptr:ptr;
s,p:ptr;
```

Пример 12.1. Процедура добавления элемента в начало списка.

```
procedure Add_begin;
begin
        p:=headptr;
        new(s);{выделение памяти под новый элемент списка}
        s^.next:=p;{связывание нового элемента с головой исходного
списка}
        s^.info:='добавляемая запись';
        headptr:=s;{объявление нового элемента головой списка}
end;
```

Пример 12.2. Пример процедуры добавления элемента в конец списка.

```
procedure Add_End;
begin
          p:=headptr;{установка на начало}
          while p^.next<>nil do
                p:=p^next;{переход к последней записи}
          new(p^.next);
          p^.info:='добавляемая запись';
          p^.next:=nil;
end;
```

Пример 12.3. Процедура удаления из ДП элементов списка.

```
 Procedure DelAllZap;
 Var
  PP,P : Ptr;
 begin
   P := HeadPtr; {связывание текущего указ.P с головой списка}
   PP := P^.Next; {связывание указ.PP со след.элементом списка}
   Dispose(P);  {освобождение памяти для указ.P}
   While PP^.Next <> Nil Do   {пока PP не последний}
   begin
     P := PP; {текущий указ.P стал равен PP  }
     PP := PP^.Next; {переход с след.элементу}
     Dispose(P); {освобождение памяти для текущ.указателя}
```

```
end;
 end;
```

Пример 12.4. Программа вывода на экран студентов группы ПМ–11.

```
uses crt;
    type
       ptr=^student;
       student=record
       fio,group:string[10];
       next:ptr;
    end;
    var
       headptr:ptr;
 PROCEDURE FORMSPISOK;
     var
        p:ptr;
        let:char;
    procedure vvod(var stud:student);
      begin
         with stud do begin
             write('  fio ');
             READLN(fio);
             writeln('  Не забудьте ввести группу ПМ–11');
             write('  group ');
             READLN(group);
            end
         end;
  Begin{простроение списка}
    headptr:=nil;
      repeat
        write('  s–закончить  ');
        READLN(let);
        let:=Upcase(let);
      if let='S' then exit;
      if headptr=nil then
       begin
        new(headptr);
        p:=headptr;
                  end
      else
        begin
           new(p^.next);
           p:=p^.next;
```

```
          end;
    vvod(p^);
    p^.next:=nil;
        until false;
   end;{formspisok}
Procedure pechspisok;
    var
      p:ptr;
       begin
         p:=headptr;
         while p <> nil do
          begin
           with p^ do
             if group = 'ПМ–11' then
            writeln('  ',fio,' ',group);
          p:=p^.next;
          end;
       end; {pechspisok}
Procedure Writefile;
 VAR
 P:PTR;
 i :byte;
 begin
writeln(' |———————————————|———————————————|');
writeln(' |      ФИО        |  Группа      |');
writeln(' |———————————————|———————————————|');
  P := Headptr ;
  i := 1;
  while p <> Nil do
    begin with p^ do
    writeln(' |',fio:17,'|',group:14,'|');
     p := p^.next;
if p <> nil then writeln(' |———————————————|———————————————|');
               else writeln(' |———————————————|———————————————|');
i := i + 1;
    end;
 End;{writefile}
{*******************************}
 Begin
 clrscr;
 window(14,2,70,23);
 textbackground(3);
 clrscr;
```

```
 textcolor(0);
 writeln('  Программа вывода на экран студентов группы ПМ–11');
    formspisok;
   writefile;
   textcolor(1);
   writeln('   Студенты группы ПМ–11');
   pechspisok;
   textcolor(4);
   writeln(' Нажмите на клавишу..');
   repeat until keypressed
End.
```

Рассмотрим организацию односвязного списка в виде стека, т. е. списка, запись и чтение в котором осуществляются только в вершине стека.

Пример 12.5. Организация стека.

```
uses crt;
type
 cc=^zap;
 zap=record
   inf:string;
   next:cc;
   end;
var
 p,top,newelement:cc;
 value:string;
procedure sozdsteka;
begin   top:=nil;  {top – вершина}
      while true do  begin
                  write('Введи информацию (конец ввода – 999) ');
                   READLN(value);
                   if value='999' then exit;
                   new(p);
                   p^.next:=top;
                   p^.inf:=value;
                   top:=p;
                   end
end;
```

Пример 12.6. Добавление элементов в стек.

```
procedure dobavs;
begin
```

```
    while true do
    begin
   write('Введи добавляемую информацию (конец ввода – 999)');
    READLN(value);
    if value='999' then exit;
    new(newelement);
    newelement^.next:=top;
    newelement^.inf:=value;
    top:=newelement
    end
end;
```

Пример 12.7. Удаление элементов из стека.

```
procedure udals;
begin
    top:=top^.next
end;
```

Пример 12.8. Просмотр элементов стека.

```
procedure rasps;
begin
    kon:=top;
    while kon<>nil do
    begin
    writeln('    ',kon^.inf);
    kon:=kon^.next
    end
end;
begin
textbackground(1);
textcolor(15);
window(1,1,80,25);
clrscr;
writeln('Программа работы со стеком');
writeln('выполняет процедуры создания,добавления и удаления');
writeln('Создание стека');
sozdsteka;
writeln(' Содержимое стека ()');
rasps;
READLN;
textcolor(6);
writeln(' Добавление элемента в стек');
```

```
        dobavs;
        textcolor(14);
        writeln(' Содержимое стека после добавления (всегда в вершину)');
        rasps;
        READLN;
        textcolor(7);
        writeln(' Удаление элемента из стека (последним пришел – первым
ушел)');
        udals;
        rasps  ;
        READLN;
        end.
```

Задания для самостоятельного выполнения

1. Создать список, содержащий сведения о месячной заработной плате рабочих завода. Каждая запись содержит поля – фамилии рабочего, наименование цеха, размер заработной платы за месяц. Количество записей произвольное.
2. Создать список, содержащий сведения о количестве изделий, собранных сборщиками цеха за неделю. Каждая запись содержит поля: фамилия сборщика, количество изделий, собранных им ежедневно в течении шестидневной недели, т. е. раздельно – в понедельник, вторник и т. д. Количество записей произвольное.
3. Создать список, содержащий сведения о количестве изделий категорий A, B, C, собранных рабочим за месяц. Структура записи имеет поля: фамилия сборщика, наименование цеха, количество изделий по категориям, собранных рабочим за месяц. Количество записей произвольное.
4. Создать список со следующими полями

Название книги	ФИО автора	Издательство	Число томов

Вывести информацию о книгах, выпущенных издательством «Прогресс», в виде:

Название книги	Фамилия автора

Контрольные вопросы

1. Для чего используется механизм управления динамической памятью?
2. Что используется для описания указателей?
3. Что называется кучей?
4. Что такое односвязный список?
5. Что такое стек?

13. ОБЪЕКТНО–ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

По мере прогресса вычислительной техники в решении прикладных задач стал развиваться новый подход, согласно которому программа должна быть моделью предметной области. Этот подход получил название **объектно–ориентированного программирования** (ООП). Он улучшил и ускорил процесс создания больших прикладных и программных систем.

В процессе ООП программист руководствуется привычными понятиями той предметной области, для которой создается программа. Эти понятия он описывает как объект. Объект имеет определенные свойства. Состояние объекта задается значениями его признаков. Объект «знает», как решить определенные задачи, то есть располагает методами решения. Программа, написанная с использованием ООП, состоит из объектов, которые могут взаимодействовать между собой.

Объект представляет собой совокупность данных и подпрограмм, предназначенных для работы с этими данными. Подпрограммы объекта называются **методами**. Таким образом, характерной чертой новой структуры «объект» является объединение данных и методов и их обработка, называемая **инкапсуляцией**.

Любая объектно–ориентированная программа состоит из двух частей: описания объектов и последовательности действий, связанных с передачей сообщений между этими объектами.

13.1. Описание объекта в Turbo Pascal

Программная реализация объекта представляет собой объединение данных и процедур их обработки. В Turbo Pascal имеется тип object, который можно считать обобщением структурного типа record. Переменные объектного типа называются **экземплярами** объекта. Здесь требуется уточнение – экземпляр лишь формально можно назвать переменной. Его описание дается в предложении описания переменных, но в действительности экземпляр – нечто большее, чем обычная переменная.

Для описания объекта используется синтаксис, аналогичный описанию записи:

```
type <имя объекта>=object
  <список имён полей>: <тип полей>;
  ....
  <список имён полей>: <тип полей>;
  <объявление метода>
  ...
  <объявление метода>
        end;
```

В отличии от типа «запись», объектный тип содержит не только поля, описывающие данные, но также процедуры и функции, описания которых содержатся в описании объекта. В описании объекта фактически содержатся лишь **шаблоны** обращений к методам, которые необходимы компилятору для проверки соответствия количества параметров и их типов при обращении к методам. После описания объекта описываются методы:

procedure <имя объекта>.<имя метода> <параметры>

<описание процедуры>

Аналогично можно описать и метод-функцию. Методы (также как и обычные подпрограммы) могут не иметь параметров.

Вот пример описания объекта:

```
type
    Location = object
        X, Y: Integer;
        procedure Init (InitX, Inity: Integer);
        function GetX: Integer;
        function GetY: Integer;
end;
```

Здесь описывается объект, который может использоваться в дальнейшем, скажем, в графическом режиме и который предназначен для определения положения на экране произвольного графического элемента. Объект описывается с помощью зарезервированных слов object…end, между которыми находятся описания полей и методов. В нашем примере объект содержит два поля для хранения значений графических координат, а также описания процедуры и двух функций – это **методы** данного объекта. Процедура предназначена для задания первоначального положения объекта, а функции – для считывания его координат.

Зарезервированное слово private позволяет ограничить доступ к полям объекта. В следующем примере доступ к переменным X и Y возможен только через методы объектного типа Location:

```
type
    Location = object
        procedure Init (InitX, Inity: Integer);
```

```
        function GetX: Integer;
        function GetY: Integer;
    private
        X, Y: integer;
end;
```

В секции private могут находится и методы объекта.

Полное описание методов, то есть описание их реализации, должно находится после описания объекта. Имена методов составные и складываются из имени объекта и имени метода, разделенных точкой:

```
procedure Location. Init (InitX, Inity: Integer);
begin
    X : = InitX;
    Y : = InitY;
end;
fuction Location. GetX: Integer;
begin
     GetX : = X;
 end;
fuction Location. GetY: Integer;
begin
     GetY : = Y;
 end;
```

После того как объект описан, в программе можно использовать его экземпляры, то есть переменные указанного объектного типа:

```
 var
     GrMarket : Location;
```

Приведем реализацию объекта «окно» (для реализации интерфейса с помощью модуля crt). Мы можем считать, что окно задается координатами левого верхнего угла и размерами по горизонтали и вертикали (с учетом рамки окна), а также иметь флаг видимости (true означает, что окно на экран выведено).

Пример 13.1.

```
type CWindow=object
  x,y:       integer; {координаты окна}
  lenx,leny: integer; {размеры окна}
  visible:   boolean; {флаг видимости}
  procedure Init (x,y,lenx,leny: integer);
  procedure Show;
```

```
  function isVisible: boolean;
end;
procedure CWindow.Init (x,y,lenx,leny: integer);
{Задает начальные параметры окна}
begin
  Self.x:=x;
  Self.y:=y;
  Self.lenx:=lenx;
  Self.leny:=leny;
  visible:=true;
end;

procedure CWindow.show;
{Рисует окно}
  var i,j: integer;
begin
  textbackground (BLUE);
  gotoxy (x,y);
  write ('╔'); {ALT+201}
  for i:=1 to lenx–2 do write ('═'); {ALT+205}
  write ('╗') {ALT+187}

  for j:=y+1 to y+leny–2 do
  begin
    gotoxy (x,j);
    write ('║'); {ALT+186}
    for i:=1 to lenx–2 do write (' '); {ALT+205}
    write ('║'); {ALT+186}
  end;

  gotoxy (x,y+leny–1);
  write ('╚'); {ALT+200}
  for i:=1 to lenx–2 do write ('═'); {ALT+205}
  write ('╝') {ALT+188}

  visible:=true;
end;

function CWindow.isVisible: boolean;
begin
  isVisible:=visible;
            end;
```

Для создания экземпляра данного объекта можно использовать стандартные способы объявления переменных и создания динамических переменных.

```
var Window: CWindow;
begin
  Window.Init (20,5,40,10);
  Window.Show;
end.
```

13.2. Инкапсуляция

Инкапсуляция – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя. Инкапсуляция или сокрытие реализации является базовым свойством ООП. Она позволяет создавать пользовательские объекты, обладающие требуемыми методами и далее оперировать ими, не вдаваясь в устройство этих объектов.

В Turbo Pascal средства объектно–ориентированного программирования связаны с тремя зарезервированными словами: OBJECT CONSTRUCTOR и DESTRUCTOR и двумя стандартными директивами: PRIVATE и VIRTUAL.

Зарезервированное слово OBJECT используется для описания объекта. Описание объекта должно помещаться в разделе описания типов, например:

Пример 13.2.

```
type
    Tpoint = object
X,Y: Integer;  {Координаты точки}
Color:word; {Цвет точки)
Visible: Boolean; {Признак светимости}
Procedure Setlocation (NewX, NewY: integer);
        {Задает новое положение, точки на экране}
Procedure SetCoforfNewColor: word); {Устанавливает цвет точки}
Procedure SetVislble(VIS: Boolean);
        {Выводит или гасит точку)
Procedure GetLocatIon(var Xloc, YLoc:integer);
        {Возвращает координаты точки}
Function GetColor: word;
```

```
        {Возвращает цвет точки)
    Function GetVislble: Boolean;
        {Возвращает признак светимости точки}
    end; {Конец описания объекта TPOINT)
```

В этом примере описывается объект TPOINT, представляющий собой данные и методы (процедуры и функции), необходимые для работы с графическими точками на экране ПК. Как видим, каждая точка характеризуется некоторым набором данных (своими координатами X и У, цветом COLOR и признаком светимости VISIBLE). Над этими данными определены все необходимые алгоритмические действия. С помощью этих переменных можно осуществлять все предусмотренные в объекте действия, например, для переменных типа TPOINT можно высветить или погасить любую точку, переместить ее по экрану, изменить цвет.

13.3. Наследование

Достоинства объектно–ориентированного программирования проявляются только в случае, если все объекты расположить в виде иерархической структуры, в которой отражены наследование свойств родительских (выше расположенных) объектов дочерними (ниже расположенными) объектами, или потомками.

Наследование – это такое отношение между объектами, когда дочерний объект повторяет структуру и поведение другого объекта (родителя).

Наследование – это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс – потомком, наследником или производным классом.

При объектно-ориентированном программировании необходимо как можно полно описать методы каждого объекта, определить отношение между ними, т. е. наследование, и, если необходимо, переопределить методы объекта родителя.

Любой объект может быть объявлен потомком ранее описанного объекта. В этом случае он наследует все данные и методы объекта-родителя и может дополнять их своими данными и методами.

При объявлении объекта-потомка за словом OBJECT в круглых скобках указывается имя объекта–родителя. У объекта может быть сколько угодно потомков, но только один родитель. При объявлении объекта-потомка TLINE перечислены лишь те данные и методы, которых недостает

в объекте–родителе TPOINT, остальные TLINE автоматически наследует от своего родителя.

Пример 13.3.

```
type
      TLine = object (TPoint)
      X, Y Integer; {Координаты начала линии}
Color word; {Цвет линии)
Visible Boolean; (Признак светимости)
XE.YE: Integer; {Координаты второго конца)
Procedure SetLocation(NewX, NewY: integer);
{Задает новое положение начала линии)
Procedure SetColor(NewColor: word);
{Устанавливает цвет линии)
Procedure SetVIsible(Vis: Boolean);
{Выводит или гасит линию}
Procedure GetLocation(var XLoc, YLoc: integer);
{Возвращает координаты начала линии}
Function GetColor: word;
{возвращает цвет линии}
Function GetVisible: Boolean;
{Возвращает признак светимости линии}
Procedure SetLIneLocationfxl.X1,Y1,x2 ,y2: integer);
{Задает новое положение линии на экране}
Procedure GetLineLocatlon(var x11,y1l,x2l,y2l):integer);
{Возвращает координаты линии}
Procedure SetLineVisible(vis: Boolean);
{Выводит или гасит линию)}
end; {Конец описания объекта TLIne }
```

Из этого примера видно главное преимущество наследования: при описании объекта-потомка вам нет необходимости заново описывать уж существующие в объекте-родителе поля и методы. Потомок просто использует их нужным образом для реализации требуемых от него действий; все, в чем нуждается потомок, – это описать специфичные для него поля методы, недостающие в объекте-родителе.

Наследование распространяется на любые объекты, в том числе и объекты–потомки: если в качестве родителя указано имя объекта, который сам по себе является потомком, новый объект наследует все свойства своего родителя и все свойства своих прародителей. Таким образом, наследование обеспечивает создание дерева родственных объектов.

Как и любое другое дерево, дерево объектов имеет «корень» – объект, являющийся прародителем всех других объектов иерархии, и «ветви» – порожденные от него потомки. По мере передвижения от корня к ветвям и перехода с ветви на ветвь объекты разрастаются в своих размерах, присоединяя к себе все новые и новые поля и методы. Если иерархия объектов хорошо продумана, на каждом ее уровне к объекту-родителю добавляются только необходимые поля и методы.

Механизм наследования – это, пожалуй, самое мощное свойство ООП. Без наследования объекты превращаются в простую комбинацию данных и подпрограмм, не дающую качественных преимуществ по сравнению с традиционными для Паскаля процедурами и модулями.

13.4. Полиморфизм

Объект-потомок может не только дополнять поля и методы родителя, но и заменять методы родителя на новые (заменять поля родителя нельзя!). Например, вместо правила SETLINEVISIBLE мы могли бы в объекте TLINE объявить правило SETVISIBLE, которое в этом случае перекроет (заменит собой) одноименное правило объекта–родителя TPOINT. В результате, к разным родственным объектам TPOINT и TLINE можно было бы применять одноименные правила SETVISIBLE, обеспечивающие сходные в смысловом отношении действия – показать или сделать невидимым графический объект. Свойство, позволяющее называть разные алгоритмические действия одним именем, называется **полиморфизмом**.

Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Задания для самостоятельного выполнения

В соответствии с вариантом задания самостоятельно разработать объект и наследника. Для каждого варианта приведен рекомендуемый перечень операций над объектами.

1. Объект «Студент». Действия над объектами: начисление стипендии, получение информации об успеваемости, вывод личных данных, перевод на следующий курс, операции сравнения объектов, присваивание.

2. Объект «Дата и время суток». Действия над объектами: сравнение объектов, арифметические операции над объектами, вывод на экран данных, присваивание.

3. Объект «Цена в рублях и копейках». Действия над объектами класса: сравнение объектов, арифметические операции над объектами, вывод на экран данных, присваивание.

4. Объект «Цветной заполненный круг». Действия над объектами: вывод изображения на экран и удаление изображения (в графическом режиме), перемещение объекта по экрану, изменение цвета и вида заполнения, масштабирование, сравнение объектов, присваивание.

5. Объект «Цветной заполненный прямоугольник». Действия над объектами: вывод изображения на экран и удаление изображения (в графическом режиме), перемещение объекта по экрану, изменение цвета и вида заполнения, масштабирование, сравнение объектов, присваивание.

6. Объект «Цветной заполненный треугольник». Действия над объектами: вывод изображения на экран и удаление изображения (в графическом режиме), перемещение объекта по экрану, изменение цвета и вида заполнения, масштабирование, сравнение объектов, присваивание.

7. Объект «Цветной отрезок прямой». Действия над объектами: вывод изображения на экран и удаление изображения (в графическом режиме), перемещение объекта по экрану, изменение цвета, масштабирование, поворот на заданную величину в градусах относительно одного из концов отрезка, сравнение объектов, присваивание.

8. Объект «Цветная строка». Действия над объектами: вывод на экран (в графическом режиме), конкатенация, присваивание, сравнение. Объекты должны иметь свойства, регулирующие способ вывода информации на экран: вертикально или горизонтально.

9. Объект «Цветная заполненная трапеция». Действия над объектами: вывод изображения на экран и удаление изображения (в графическом режиме), перемещение объекта по экрану, изменение цвета и вида заполнения, масштабирование, сравнение объектов, присваивание.

10. Объект «Сотрудник». Действия над объектами: получение информации о месте работы, занимаемой должности и стаже работы, изменение должности, начисление заработной платы, вывод личных данных, операции сравнения объектов, присваивание.

11. Объект «Цветной заполненный сектор». Действия над объектами: вывод изображения на экран и удаление изображения (в графическом режиме), перемещение объекта по экрану, изменение цвета и вида заполнения, масштабирование, сравнение объектов, присваивание.

12. Объект «Абитуриент». Действия над объектами: получение и изменение информации о факультете и шифре специальности, вывод личных данных, получение информации о наличии медали, операции сравнения объектов, присваивание

Контрольные вопросы

1. Для чего используется механизм ООП?
2. Что такое объект?

3. Что такое инкапсуляция?
4. Что такое наследование?
5. Что такое полиморфизм?
6. Что такое методы объекта?
7. Как описываются объекты в Turbo Pascal?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Аляев, Ю. А. Практикум по алгоритмизации и программированию на языке Паскаль: учебное пособие / Ю. А. Аляев, В. П. Гладков, О. А. Козлов. – М. : Финансы и статистика, 2004. – 528 с.
2. Антонов, А. В. Системный анализ : учебник для вузов / А. В. Антонов. – 2-е изд., стер. – М. : Высш. шк., 2006. – 453 с.
3. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт. – М. : ДМК Пресс, 2011. – 272 с.
4. Гагарина, Л. Г. Алгоритмы и структуры данных : учебное пособие / Л. Г. Гагарина. – М. : Финансы и статистика: ИНФРА–М, 2009. – 303 с.
5. Голицына, О. Л. Основы алгоритмизации и программирования / О. Л. Голицына, И. И. Попов. – М. : Форум, 2010. – 432 с.
6. Епанешников, А. М. Программирование в среде Turbo Pascal 7.0 / А. М. Епанешников, В. А. Епанешников. – 4–е изд., испр. и доп. – М. : Диалог–Мифи, 2004. – 368 с.
7. Каймин, В. А. Информатика : учебник / В. А. Каймин. – М. : Проспект, 2011. – 270 с.
8. Перегудов, Ф. И. Введение в системный анализ / Ф. И. Перегудов, Ф. П. Тарасенко. – М. : Высшая школа, 1997. – 389 с.
9. Могилев, А. В. Информатика / А. В. Могилев, Н. И. Пак, Е. К. Хеннер ; под ред. Е. К. Хеннера. – М. : Академия, 2009. – 848 с.
10. Моргун, А. Н. Программирование на языке Паскаль. Основы обработки структур данных / А. Н. Моргун, И. А. Кривель. – М. : Вильямс, 2006. – 576 с.
11. Немнюгин, С. А. Turbo Pascal. Программирование на языке высокого уровня: учебник для вузов / С. А. Немнюгин. – 2–е изд. – СПб. : Питер, 2007. – 543 с.
12. Программирование на языке Паскаль / под ред. О. Ф. Усковой. – СПб. : Питер, 2003. – 333 с.
13. Семакин, И. Г. Основы алгоритмизации и программирования / И. Г. Семакин, А. П. Шестаков. – М. : Академия, 2011. – 400 с.
14. Симонович, С.В. Информатика : универсальный курс / С. В. Симонович. – СПб. : Питер, 2007. – 432 с.
15. Фаронов, В. В. Turbo Pascal 7.0 : учебный курс / В. В. Фаронов. – М. : КноРус, 2009. – 368 с.

Учебное издание

АФАНАСЬЕВА Татьяна Васильевна,
КУВАЙСКОВА Юлия Евгеньевна,
ФАСХУТДИНОВА Венера Арифзяновна

АЛГОРИТМЫ И ПРОГРАММЫ

Учебное пособие

Редактор М.В. Штаева

ЛР №020640 от 22.10.97.
Подписано в печать 15.11.2011. Формат 60×84/16.
Усл. печ. л. 13,25. Тираж 150 экз. Заказ 1181.

Ульяновский государственный технический университет
432027, г. Ульяновск, ул. Сев. Венец, 32.

Типография УлГТУ, 432027, г. Ульяновск, ул. Сев. Венец, 32.