Лекции по дисциплине «Информатика»

1 семестр

Раздел 1. Понятие информации. Общая характеристика процессов сбора, передачи, обработки и накопления информации.

Раздел 2. Алгоритмы и способы их описания. Основные алгоритмические конструкции (линейные структуры, ветвление, циклы).

Раздел 3. Язык программирования высокого уровня Pascal. Линейные конструкции языка Pascal. Условные конструкции языка Pascal. Циклические конструкции языка Pascal. Одномерные массивы.

Список литературы

1. Валова О.В. Основы программирования на языке Паскаль: учеб. пособие / О.В. Валова, С.Н. Розова; Забайкал. гос. ун-т. – Чита: ЗабГУ, 2017.

2. Могилев А.В. Информатика: учеб. пособие / под ред. Е.К. Хеннера. – 7-е изд., стер. – Москва: Академия, 2009. – 848 с.

3. Могилев А.В. Практикум по информатике: учеб. пособие / А.В. Могилев, Н.И. Пак, Е.К. Хеннер; под ред. Е.К. Хеннера; под ред. Е.К. Хеннера. – 4-е изд., стер. – Москва: Академия, 2008. – 608 с. – (Высшее профессиональное образование).

4. Культин Н.Б. Turbo Pascal в задачах и примерах / Н.Б. Культин. – Санкт-Петербург: БХВ-Петербург, 2008. – 256 с.: ил.

Лекция состоится в режиме видеоконференции zoom согласно расписанию занятий, 30 января 2021 года.

Подключиться к конференции zoom можно по ссылке

https://us05web.zoom.us/j/88160981953?pwd=QTZuOHUrS1FtaUtuWUxQVlNtVnV2UT09

Раздел 1.
Понятие информации. Общая характеристика процессов сбора, передачи, обработки и накопления информации.

1.1.Информатика как наука. Понятие об информации

Термин "информация" происходит от латинского слова "informatio", что означает сведения, разъяснения, изложение. Несмотря на широкое распространение этого термина, понятие информации является одним из самых дискуссионных в науке.

Несмотря на то, что с понятием информации мы сталкиваемся ежедневно, строгого и общепризнанного ее определения до сих пор не существует, поэтому можно дать следующие определения:

Определение 1. Информация – набор символов, графических образов или звуковых сигналов, несущих определенную смысловую нагрузку. Т.о. это сведения, которые один реальный объект содержит о другом реальном объекте.

Определение 2. Информация – это продукт взаимодействия данных и адекватных им методов.

Определение 3. Информация есть форма движения материи, это одна из трех составляющих основ мировоззрения наряду с материей и энергией.

Определение 4. Информация – это обозначение содержания, полученного из внешнего мира в процессе нашего приспособления к нему и приспособления к нему наших чувств.

Информация может существовать в виде:

- текстов, рисунков, чертежей, фотографий;
- световых или звуковых сигналов;
- радиоволн;
- электрических и нервных импульсов;
- магнитных записей;
- жестов и мимики;
- запахов и вкусовых ощущений;
- хромосом, посредством которых передаются по наследству признаки и свойства

организмов и т.д.

Люди обмениваются информацией в форме сообщений. Сообщение – это форма представления информации в виде речи, текстов, жестов, взглядов, изображений, цифровых данных, графиков, таблиц и т.п.

Одно и то же информационное сообщение (статья в газете, объявление, письмо, телеграмма, справка, рассказ, чертёж, радиопередача и т.п.) может содержать разное количество информации для разных людей – в зависимости от их предшествующих знаний, от уровня понимания этого сообщения и интереса к нему.

Информация о любом материальном объекте может быть получена путем наблюдения за этим объектом, вычислительного эксперимента над ним или путем логического вывода. В связи с этим информацию делят на доопытную, или *априорную*, и послеопытную, или *апостериорную*, полученную в результате проведенного эксперимента.

Термин *"информатика"* (франц. informatique) возник в 60-х г.г. во Франции и происходит от французских слов information (информация) и automatique (автоматика) и дословно означает "информационная автоматика". Возник для названия области, занимающейся автоматизированной обработкой информации с помощью электронных вычислительных машин.

Широко распространён также англоязычный вариант этого термина – "Computer science", что означает буквально "компьютерная наука".

Информатика – это основанная на использовании компьютерной техники наука, изучающая структуру и общие свойства информации, а также закономерности и методы её создания, хранения, поиска, преобразования, передачи и применения в различных сферах человеческой деятельности.

Для *информатики* как технической науки понятие информации не может основываться на таких понятиях, как *знание,* и не может опираться только на объективность фактов и свидетельств. Средства вычислительной техники обладают способностью обрабатывать информацию автоматически, без участия человека, и ни о каком знании или незнании здесь речь идти не может. Эти средства могут работать с искусственной, абстрактной и даже с ложной информацией, не имеющей объективного

отражения ни в природе, ни в обществе.

1.2. Свойства информации

С точки зрения информатики наиболее важными представляются следующие свойства:

- объективность;
- достоверность;
- полнота;
- точность;
- ценность;
- своевременность;
- понятность;
- доступность;
- краткость;
- адекватность;
- актуальность;
- эргономичность.

Объективность и субъективность информации. Понятие объективности информации является относительным. Это понятно, если учесть, что методы являются субъективными. Более объективной принято считать ту информацию, в которую методы вносят меньший субъективный элемент. Так, например, принято считать, что в результате наблюдения фотоснимка природного объекта или явления образуется более объективная информация, чем в результате наблюдения рисунка того же объекта, выполненного человеком.

Достоверность информации. Информация достоверна, если она отражает истинное положение дел и не имеет скрытых ошибок. Недостоверная информация может привести к неправильному пониманию или принятию неправильных решений. Достоверная информация со временем может стать недостоверной, так как она обладает свойством устаревать, то есть перестаёт отражать истинное положение дел.

Полнота информации. Полнота информации во многом характеризует *качество*

информации и определяет *достаточность* данных для принятия решений или для создания новых данных на основе имеющихся. Чем полнее данные, тем шире диапазон методов, которые можно использовать, тем проще подобрать метод, вносящий минимум погрешностей в ход информационного процесса. Как неполная, так и избыточная информация сдерживает принятие решений или может повлечь ошибки.

Точность информации. Точность информации определяется степенью ее близости к реальному состоянию объекта, процесса, явления и т.п.

Ценность информации. Ценность информации зависит от того, насколько она важна для решения задачи, а также от того, насколько в дальнейшем она найдёт применение в каких-либо видах деятельности человека.

Своевременность информации. Только своевременно полученная информация может принести ожидаемую пользу. Одинаково нежелательны как преждевременная подача информации (когда она ещё не может быть усвоена), так и её задержка.

Понятность информации. Если ценная и своевременная информация выражена непонятным образом, она может стать бесполезной. Информация становится понятной, если она выражена языком, на котором говорят те, кому предназначена эта информация.

Доступность информации. Информация должна преподноситься в доступной (по уровню восприятия) форме. Поэтому одни и те же вопросы по-разному излагаются в школьных учебниках и научных изданиях.

Краткость информации. Информацию по одному и тому же вопросу можно изложить кратко (сжато, без несущественных деталей) или пространно (подробно, многословно). Краткость информации необходима в справочниках, энциклопедиях, учебниках, всевозможных инструкциях.

Адекватность информации – это степень соответствия реальному объективному состоянию дела. Неадекватная информация может образовываться при создании новой информации на основе неполных или недостоверных данных. Однако и полные, и достоверные данные могут приводить к созданию неадекватной информации в случае применения к ним неадекватных методов.

Актуальность информации – это степень соответствия информации текущему моменту времени. Нередко с актуальностью, как и с полнотой, связывают коммерческую ценность информации. Поскольку информационные процессы

растянуты во времени, то достоверная и адекватная, но устаревшая информация может приводить к ошибочным решениям. Необходимость поиска (или разработки) адекватного метода для работы с данными может приводить к такой задержке в получении информации, что она становится неактуальной и ненужной.

Эргономичность – свойство, характеризующее удобство формы или объема информации с точки зрения данного потребителя.

Так же можно рассматривать такие свойства информации:

- запоминаемость;
- передаваемость;
- воспроизводимость;
- преобразуемость;
- стираемость.

Запоминаемость – одно из самых важных свойств. Запоминаемую информацию будем называть макроскопической (имея в виду пространственные масштабы запоминающей ячейки и время запоминания). Именно с макроскопической информацией мы имеем дело в реальной практике.

Продаваемость информации с помощью каналов связи (в том числе с помехами) хорошо исследована в рамках теории информации К. Шеннона. В данном случае имеется в виду несколько иной аспект – способность информации к копированию, т.е. к тому, что она может быть “запомнена” другой макроскопической системой и при этом останется тождественной самой себе. Очевидно, что количество информации не должно возрастать при копировании.

Воспроизводимость информации тесно связана с ее передаваемостью и не является ее независимым базовым свойством. Если передаваемость означает, что не следует считать существенными пространственные отношения между частями системы, между которыми передается информация, то воспроизводимость характеризует неиссякаемость и неистощимость информации, т.е. что при копировании информация остается тождественной самой себе.

Фундаментальное свойство информации – *преобразуемость*. Оно означает, что информация может менять способ и форму своего существования. Копируемость есть

разновидность преобразования информации, при котором ее количество не меняется. В общем случае количество информации в процессах преобразования меняется, но возрастать не может.

Свойство *стираемости* информации также не является независимым. Оно связано с таким преобразованием информации (передачей), при котором ее количество уменьшается и становится равным нулю.

1.3. Методы оценки и виды информации

При оценке информации различают три аспекта: синтаксический, семантический и прагматический.

Синтаксический аспект связан со способом представления информации вне зависимости от ее смысловых и потребительских качеств и рассматривает формы представления информации для ее передачи и хранения (в виде знаков и символов). Данный аспект необходим для измерения информации.

Информацию, рассмотренную только в синтаксическом аспекте, называют *данными*.

Семантический аспект передает смысловое содержание информации и соотносит ее с ранее имевшейся информацией (рисунок 1.1).

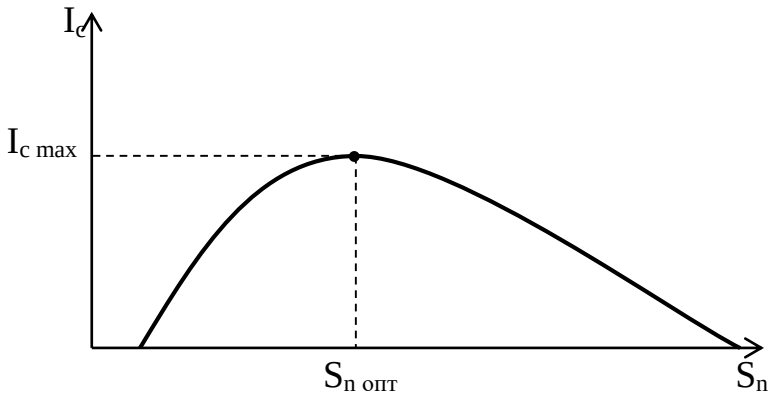


Рисунок 1.1. Семантический аспект информации

S_n – тезаурусная мера получателя; I_c – семантическое количество информации.

Прагматический аспект передает возможность достижения цели с учетом

полученной информации.

$$I_n = \log_a \frac{P_1}{P_2},$$

где P_0 – вероятность достижения цели до получения информации;

P_1 – вероятность достижения цели после получения информации;

I_n – прагматическое количество информации; $a > 1$.

Рассмотрим какие бываю виды информации.

Виды информации

1. Научная информация (наиболее полно отражает объективные закономерности природы, общества и мышления);

2. Информация управления:

а) производственная, связанная с управлением людьми;

б) техническая, связанная с управлением техническими объектами.

Также классификация информации может производиться по следующим основаниям:

1. По областям применения:

а) политическая;

б) техническая;

в) педагогическая;

г) физическая;

д) экономическая и др.;

2. По назначению:

а) массовая;

б) специальная.

1.4. Носители данных

Данные – диалектическая составная часть информации. Они представляют собой зарегистрированные сигналы. В соответствии с методом регистрации данные могут храниться и транспортироваться на носителях различных видов.

Самым распространенным носителем данных, хотя и не самым экономичным, по-видимому, является бумага. В качестве носителей, использующих изменение

магнитных свойств, можно назвать магнитные ленты и диски. Регистрация данных путем изменения химического состава поверхностных веществ носителя широко используется в фотографии. На биохимическом уровне происходит накопление и передача данных в живой природе.

Любой носитель можно характеризовать параметром *разрешающей способности* (количеством данных, записанных в принятой для носителя единице измерения) и *динамическим диапазоном* (логарифмическим отношением интенсивности амплитуд максимального и минимального регистрируемого сигналов). От этих свойств носителя нередко зависят такие свойства информации, как полнота, доступность и достоверность. Так, например, мы можем рассчитывать на то, что в базе данных, размещаемой на компакт-диске, проще обеспечить полноту информации, чем в аналогичной по назначению базе данных, размещенной на гибком магнитном диске, поскольку в первом случае плотность записи данных на единице длины дорожки намного выше. Для обычного потребителя доступность информации в книге заметно выше, чем той же информации на компакт-диске, поскольку не все потребители обладают необходимым оборудованием. И, наконец, известно, что визуальный эффект от просмотра слайда в проекторе намного больше, чем от просмотра аналогичной иллюстрации, напечатанной на бумаге, поскольку диапазон яркостных сигналов в проходящем свете на два-три порядка больше, чем в отраженном.

1.5. Операции с данными

В ходе информационного процесса данные преобразуются из одного вида в другой с помощью методов. Обработка данных включает в себя множество различных операций. В структуре возможных операций с данными можно выделить следующие основные:

- *сбор данных* – накопление информации с целью обеспечения достаточной полноты для принятия решений;
- *формализация данных* – приведение данных, поступающих из разных источников, к одинаковой форме, чтобы сделать их сопоставимыми между собой, то есть повысить их уровень доступности;
- *фильтрация данных* – отсеивание "лишних" данных, в которых нет

необходимости для принятия решений; при этом должен уменьшаться уровень "шума", а достоверность и адекватность данных должны возрастать;

- *сортировка данных* – упорядочение данных по заданному признаку с целью удобства использования; повышает доступность информации;
- *архивация данных* – организация хранения данных в удобной и легкодоступной форме; служит для снижения экономических затрат по хранению данных и повышает общую надежность информационного процесса в целом;
- *защита данных* – комплекс мер, направленных на предотвращение утраты, воспроизведения и модификации данных;
- *транспортировка данных* – прием и передача (доставка и поставка) данных между удаленными участниками информационного процесса; при этом источник данных в информатике принято называть *сервером,* а потребителя – *клиентом;*
- *преобразование данных* – перевод данных из одной формы в другую или из одной структуры в другую. Преобразование данных часто связано с изменением типа носителя, например, книги можно хранить в обычной бумажной форме, но можно использовать для этого и электронную форму, и микрофотопленку.

Приведенный здесь список типовых операций с данными далеко не полон. Можно сделать вывод: *работа с информацией может иметь огромную трудоемкость, и ее надо автоматизировать.*

1.6. Обработка информации. Кодирование данных

Источниками и носителями информации могут быть сигналы любой природы: речь, музыка, текст, показания приборов и т. д. Однако хранение, передача и переработка информации в ее естественном физическом виде большей частью неудобна, а иногда и просто невозможна.

Для автоматизации работы с данными, относящимися к различным типам, очень важно унифицировать их форму представления – для этого обычно используется прием *кодирования.*

Кодирование – это процесс установления взаимно однозначного соответствия элементам и словам одного алфавита элементов и слов другого алфавита, то есть

выражение данных одного типа через данные другого типа.

Естественные человеческие *языки* – это не что иное, как системы кодирования понятий для выражения мыслей посредством речи. К языкам близко примыкают *азбуки* (системы кодирования компонентов языка с помощью графических символов).

Кодом называется правило, по которому сопоставляются различные алфавиты и слова.

Всю информацию, участвующую в электронном вычислительном процессе, можно разделить на обрабатываемую (данные) и управляющую (программы).

В схеме преобразования информации в данные (рисунок 1.2) представлены проводимые над информацией и данными процессы, которые образуются после введения информации в компьютер. Также представлены процедуры и связи между ними, с помощью которых осуществляются эти процессы.

Процедура *отображения* – преобразование информации в вид, удобный для восприятия человеком.

Рисунок 1.2. Схема преобразования информации в данные и действий над ними

Проблема универсального средства кодирования достаточно успешно реализуется в отдельных отраслях техники, науки и культуры. В качестве примеров можно привести

систему записи математических выражений, телеграфную азбуку, морскую флажковую азбуку, систему Брайля для слепых и многое другое.

Компьютер может обрабатывать только информацию, представленную в числовой форме. Вся другая информация для обработки на компьютере должна быть преобразована в числовую форму. Например, чтобы перевести в числовую форму музыкальный звук, можно через небольшие промежутки времени измерять интенсивность звука на определенных частотах, представляя результаты каждого измерения в числовой форме.

Аналогичным образом на компьютере можно обработать и текстовую информацию. При вводе в компьютер каждая буква кодируется определенным числом, а при выводе на внешние устройства для восприятия человеком по этим числам строится соответствующее изображение буквы.

Своя система существует и в вычислительной технике – она называется двоичным кодированием и основана на представлении данных последовательностью всего двух знаков: 0 и 1. Эти знаки называются двоичными цифрами, по-английски – binary digit или сокращенно bit (бит).

Бит – единица информации, представляющая собой двоичный разряд, который может принимать значение 0 или 1, (да или лет, черное или белое, истина или ложь и т. п.).

Байт – восемь последовательных битов. В одном байте можно кодировать значение одного символа из 256 возможных ($256 = 2^8$). Более крупными единицами информации являются следующие:

1 байт=8 бит

1 Кбайт=1024 байт=2^{10} байт

1 Мбайт = 1024 Кбайт = 2^{20} байт

1 Гбайт = 1024 Мбайт = 2^{30} байт

1 Тбайт =1024 Гбайт = 2^{40} байт

В них обычно измеряется емкость запоминающих устройств.

Обычно приставка “кило” означает тысячу, а приставка “мега” – миллион, но в вычислительной технике все “привязывается” к принятой двоичной системе кодирования.

В силу этого один килобайт равен не тысяче байтов, а 2^{10} = 1024 байтов.

Аналогично, 1 Мб = 2^{10} Кб = 1024 Кб = 2^{20} байт = 1 048 576 байт.

1 Гб = 2^{10} Мб = 2^{20} Кб = 2^{30} байт = 1 073 741 824 байт.

Поскольку одним байтом, как правило, кодируется один символ текстовой информации, то для текстовых документов размер в байтах соответствует лексическому объему в символах.

То есть 1 символ = 1 байт = 8 бит

Если количество битов увеличить до двух, то уже можно выразить четыре различных понятия:

00 01 10 11

Тремя битами можно закодировать восемь различных значений:

000 001 010 011 100 101 110 111

Увеличивая на единицу количество разрядов в системе двоичного кодирования, мы увеличиваем в два раза количество значений, которое может быть выражено в данной системе, то есть общая формула имеет вид: $N=2m$,

где N – количество независимых кодируемых значений;

m – разрядность двоичного кодирования, принятая в данной системе.

Существует множество систем представления данных. С одной из них, принятой в информатике и вычислительной технике, двоичным кодом, мы познакомились выше. Наименьшей единицей такого представления является бит (двоичный разряд).

Совокупность двоичных разрядов, выражающих числовые или иные данные, образует некий битовый рисунок. Практика показывает, что с битовым представлением удобнее работать, если этот рисунок имеет регулярную форму. В настоящее время в качестве таких форм используются группы из восьми битов, которые называются байтами.

Выше мы видели, что во многих случаях целесообразно использовать не восьмиразрядное кодирование, а 16-разрядное, 24-разрядное, 32-разрядное и более. Группа из 16 взаимосвязанных бит (двух взаимосвязанных байтов) в информатике называется словом. Соответственно, группы из четырех взаимосвязанных байтов (32 разряда) называются удвоенным словом, а группы из восьми байтов (64 разряда) -

учетверенным словом. Пока, на сегодняшний день, такой системы обозначения достаточно.

Из курса физики вы знаете, что прежде, чем измерять значение какой-либо физической величины, надо ввести единицу измерения. У информации тоже есть такая единица – *бит,* но смысл ее различен при разных подходах к определению понятия “информация”.

I ПОДХОД. Неизмеряемость информации в быту (информация как новизна)

ПРИМЕР

Вы получили какое – то сообщение, например, прочитали статью в любимом журнале. В этом сообщении содержится какое-то количество информации. Как оценить, сколько информации Вы получили? Другими словами, *как измерить информацию?* Можно ли сказать, что чем больше статья, тем больше информации она содержит?

Разные люди, получившие одно и то же сообщение, по-разному оценивают его информационную ёмкость, то есть количество информации, содержащееся в нем. Это происходит оттого, что знания людей о событиях, явлениях, о которых идет речь в сообщении, *до получения* сообщения были различными. Поэтому те, кто знал об этом мало, сочтут, что получили много информации, те же, кто знал больше, могут сказать, что информации не получили вовсе. *Количество информации в сообщении, таким образом, зависит от того, насколько ново это сообщение для получателя***.**

В таком случае, количество информации в одном и том же сообщении должно определяться отдельно для каждого получателя, то есть иметь субъективный характер. Но субъективные вещи не поддаются сравнению и анализу, для их измерения трудно выбрать одну общую для всех единицу измерения.

Таким образом, с точки зрения информации как новизны, мы не можем однозначно и объективно оценить количество информации, содержащейся даже в простом сообщении. Что же тогда говорить об измерении количества информации, содержащейся в научном открытии, новом музыкальном стиле, новой теории общественного развития.

Поэтому, когда информация рассматривается как новизна сообщения для получателя, **не** ставится вопрос об измерении количества информации.

II ПОДХОД – объемный. Измерение информации в технике (информация как сообщения в форме знаков или сигналов, хранимые, передаваемые и обрабатываемые с помощью технических устройств).

В технике, где информацией считается любая хранящаяся, обрабатываемая или передаваемая последовательность знаков, сигналов, часто используют простой способ определения количества информации, который может быть назван *объемным*. Он основан на *подсчете числа символов в сообщении,* то есть связан только с длиной сообщения и не учитывает его содержания.

Длина сообщения зависит от числа знаков, употребляемых для записи сообщения. Например, слово “мир” в русском алфавите записывается тремя знаками, в английском - пятью (peace), а в КОИ -8 - двадцатью четырьмя битами (111011011110100111110010).

ПРИМЕР

Исходное сообщение		**Количество информации**		
на языке	в машинном представлении (КОИ - 8)	в символах	в битах	в байтах
рим	11110010 11101001 11101101	3	24	3
мир	11101101 11101001 11110010	3	24	3
миру мир!	11101101 11101001 11110010 11110101 00100000 11101101 1110101 11110010 00100001	9	72	9
(** */	00101000 00101010 00101010 00100000 00101010 00101111	6	48	6

Конечно, будет правильно, если вы скажете: “В слове “Рим” содержится 24 бита информации, а в сообщении “Миру мир!” – 72 бита”. Однако, прежде, чем измерить информацию в битах, Вы определяете количество символов в этом сообщении. Нам привычней работать с символами, машине – с кодами. Каждый символ в настоящее время в вычислительной технике кодируется 8-битным или 16-битным кодом. Поэтому, для удобства была введена более “крупная” единица информации в технике (преимущественно в вычислительной) – *байт*. Теперь вам легче подсчитать количество информации в техническом сообщении - оно совпадает с количеством символов в нем.

ПРИМЕР

В 100 Мб можно “уместить”:

страниц текста	50 000 или 150 романов
цветных слайдов высочайшего качества	150
аудиозапись речи видного политического деятеля	1.5 часа
музыкальный фрагмент качества CD -стерео	10 минут
фильм высокого качества записи	15 секунд
протоколы операций с банковским счетом	за 1000 лет

III ПОДХОД – вероятностный. Измерение информации в теории информации (информация как снятая неопределенность)

Рисунок 1.3. Информация как снятая неопределенность

Получение информации (ее *увеличение*) одновременно означает увеличение знания, что, в свою очередь, означает *уменьшение* незнания или информационной *неопределенности.*

За *единицу* количества информации *принимают выбор одного из двух равновероятных* сообщений (“да” или “нет”, “1” или “0”). Она также названа *бит*. Вопрос ценности этой информации для получателя - это уже из иной области.

ПРИМЕР

Книга лежит на одной из двух полок – верхней или нижней. Сообщение о том, что книга лежит на верхней полке, уменьшает неопределенность ровно вдвое и несет 1 бит информации.

Сообщение о том, как упала монета после броска – “орлом” или “решкой”, несет один бит информации.

В соревновании участвуют 4 команды. Сообщение о том, что третья команда набрала большее количество очков, уменьшает первоначальную неопределенность ровно в четыре раза (дважды по два) и несет два бита информации.

Очень приближенно можно считать, что количество информации в сообщении о каком-то событии совпадает с количеством вопросов, которые необходимо задать и ответом на которые могут быть лишь “да” или “нет”, чтобы получить ту же информацию. Причем событие, о котором идет речь, должно иметь ***равновероятные*** исходы.

ПРИМЕР

Сколько вопросов надо задать, чтобы отгадать одну из 32 карт (колода без шестерок), если ответами могут быть лишь “да” или “нет”?

Оказывается, достаточно всего лишь 5 вопросов, но задавать их надо так, чтобы после каждого ответа можно было “отбрасывать” из рассмотрения ровно половину карт, среди которых задуманной не может быть. Такими, например, являются вопросы о цвете масти карты (“Задуманная карта красной масти?”), о типе карты (“Задуманная карта – “картинка”?”) и т.п.

То есть сообщение о том, какая карта из 32 задумана несет 5 бит информации.

Во всех приведенных примерах число равновероятных исходов события, о котором идет речь в сообщении, было кратным степени числа 2 ($4 = 2^2$, $32 = 2^5$). Поэтому сообщение “несло” количество бит информации всегда было целым числом. Но в реальной практике могут встречаться самые разные ситуации.

ПРИМЕР

Сообщение о том, что на светофоре красный сигнал, несет в себе информации больше, чем бит. Попробуйте объяснить почему.

ПРИМЕР

Известно, что Иванов живет на улице Весенней. Сообщение о том, что номер его дома есть число четное, уменьшило неопределенность. Получив такую информацию, мы стали знать больше, но информационная неопределенность осталась, хотя и уменьшилась.

Почему в этом случае мы не можем сказать, что первоначальная неопределенность уменьшилась вдвое (иными словами, что мы получили 1 бит информации)? Если Вы не

знаете ответа на этот вопрос, представьте себе улицу, на четной стороне которой, например, четыре дома, а на нечетной – двадцать. Такие улицы не такая уж большая редкость.

Последние примеры показывают, что данное выше определение количества информации слишком упрощено. Уточним его. Но прежде разберем еще один пример.

ПРИМЕР

Пылкий влюбленный, находясь в разлуке с объектом своей любви, посылает телеграмму: “Любишь?”. В ответ приходит не менее лаконичная телеграмма: “Да!”. Сколько информации несет ответная телеграмма? Альтернатив здесь две- либо Да, либо Нет. Их можно обозначить символами двоичного кода 1 и 0. Таким образом, ответную телеграмму можно было бы закодировать всего одним двоичным символом.

Можно ли сказать, что ответная телеграмма несет одну единицу информации?

Если влюбленный уверен в положительном ответе, то ответ “да” почти не даст ему никакой новой информации. То же самое относится и к безнадежно влюбленному, уже привыкшему получать отказы. Ответ “нет” также принесет ему очень мало информации. Но внезапный отказ уверенному влюбленному (неожиданное огорчение) или ответ “да” безнадежному влюбленному (нечаянная радость) несет сравнительно много информации, настолько много, что радикально изменяется все дальнейшее поведение влюбленного, а, может быть, его судьба!

Таким образом, с точки зрения на информацию как на снятую неопределенность *количество информации зависит от вероятности получения* данного сообщения. Причем, чем больше вероятность события, тем меньше количество информации в сообщении о таком событии.

Иными словами, количество информации в сообщении о каком-то событии зависит от вероятности свершения данного события.

Научный подход к оценке сообщений был предложен еще в 1928 году Р. Хартли. Расчетная формула имеет вид:

$$I = \log_2 N \quad \text{или} \quad 2^I = N,$$

где N – количество *равновероятных* событий (число возможных выборов),

I – количество информации.

Если $N = 2$ (выбор из двух возможностей), то $I = 1$ бит.

Бит выбран в качестве единицы количества информации потому, что принято считать, что двумя двоичными словами исходной длины k или словом длины 2k можно передать в 2 раза больше информации, чем одним исходным словом. Число возможных равновероятных выборов при этом увеличивается в 2^k раз, тогда как I удваивается.

Иногда формула Хартли записывается иначе. Так как наступление каждого из N возможных событий имеет одинаковую вероятность $\boldsymbol{p = 1 / N}$, то $\boldsymbol{N = 1 / p}$ и формула имеет вид

$$\boldsymbol{I = \log_2 (1/p) = - \log_2 p}$$

Познакомимся с более общим случаем вычисления количества информации в сообщении об одном из *N*, но уже *неравновероятных* событий. Этот подход был предложен К. Шенноном в 1948 году.

Пусть имеется строка текста, содержащая тысячу букв. Буква “о” в тексте встречается примерно 90 раз, буква ”р” ~ 40 раз, буква “ф” ~ 2 раза, буква “а” ~ 200 раз. Поделив 200 на 1000, мы получим величину 0.2, которая представляет собой среднюю частоту, с которой в рассматриваемом тексте встречается буква “а”. Вероятность появления буквы “а” в тексте ($p_а$) можем считать приблизительно равной 0.2. Аналогично, $p_р = 0.04$, $p_ф = 0.002$, $p_о = 0.09$.

Далее поступаем согласно К. Шеннону. Берем двоичный логарифм от величины 0.2 и называем то, что получилось количеством информации, которую переносит одна-единственная буква “а” в рассматриваемом тексте. Точно такую же операцию проделаем для каждой буквы. Тогда количество собственной информации, переносимой одной буквой равно

$$\boldsymbol{h_i = \log_2 1/p_i = - \log_2 p_i,}$$

где p_i – вероятность появления в сообщении *i*-го символа алфавита.

Удобнее в качестве меры количества информации пользоваться не значением h_i, а средним значением количества информации, приходящейся на один символ алфавита

$$\boldsymbol{H = S\, p_i\, h_i = - S\, p_i \log_2 p_i}$$

Значение *H* достигает максимума при равновероятных событиях, то есть при равенстве всех p_i

$$\boldsymbol{p_i = 1 / N.}$$

В этом случае формула Шеннона превращается в формулу Хартли.

Раздел 2.
Алгоритмы и способы их описания. Основные алгоритмические конструкции (линейные структуры, ветвление, циклы).

2.1. Понятие и свойства алгоритмов

Определение 1. Алгоритм – строго установленный порядок выполнения каких-либо действий, необходимых для получения конечного результата.

Определение 2. Алгоритмизация – процесс разработки алгоритма (плана действий) для решения задачи.

Появление алгоритмов связывают с зарождением математики. Более 1000 лет назад (в 825 году) ученый из города Хорезма Абдулла (или Абу Джафар) Мухаммед бен Муса аль-Хорезми создал книгу по математике, в которой описал *способы выполнения арифметических действий над многозначными числами*. Само слово «алгоритм» возникло в Европе после перевода на латынь книги этого среднеазиатского математика, в которой его имя писалось как «Алгоритми».

Свойства алгоритмов

1. *Понятность* для исполнителя – исполнитель алгоритма должен понимать, как его выполнять. Иными словами, имея алгоритм и произвольный вариант исходных данных, исполнитель должен знать, как надо действовать для выполнения этого алгоритма.

2. *Дискретност***ь (**прерывность, раздельность)**.** Свойство состоит в том, что алгоритм должен представлять собой процесс решения задачи как последовательное выполнение простых шагов, причем для выполнения каждого шага требуется определенный отрезок времени т.е. процесс преобразования исходных данных в результат происходит дискретно по времени.

3. *Определенность* (*детерминированность*). Указывает на то, что каждое правило (шаг) алгоритма должно быть четким, одноязычным и не оставлять места для произвола (произвольного трактования). Это свойство придает алгоритму местометрический характер.

4. *Результативность* (*конечность*). Означает, что алгоритм должен приводить к решению задачи за конечное число шагов.

5. *Массовость.* Указывает на то, что алгоритм должен решать некоторый класс задач, отличающихся друг от друга только исходными данными.

2.2. Способы описания алгоритмов

На практике наиболее распространены следующие формы представления алгоритмов:

- *словесная* (запись на естественном языке);
- *графическая* (изображения из графических символов);
- *псевдокоды* (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
- *программная* (тексты на языках программирования).

1. *Словесный алгоритм*.

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке.

Например, записать алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел (алгоритм Эвклида).

Алгоритм может быть следующим:

1. задать два числа;

2. если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма;

3. определить большее из чисел;

4. заменить большее из чисел разностью большего и меньшего из чисел;

5. повторить алгоритм с шага 2.

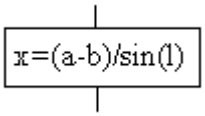
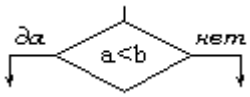
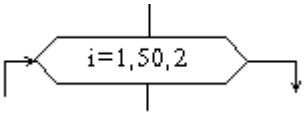
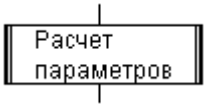
2. *Графическое представление данных*

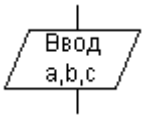
Графический способ представления алгоритмов является более компактным и наглядным по сравнению со словесным.

При *графическом представлении* алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.

Такое графическое представление называется схемой алгоритма или блок-схемой. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде *блочного символа*. Блочные символы соединяются *линиями переходов*, определяющими очередность выполнения действий. В таблице приведены наиболее часто употребляемые символы.

Блок-схемы в алгоритмах и их назначение

Название символа	**Обозначение и пример заполнения**	**Пояснение**
Процесс		**Вычислительное действие или последовательность действий** Блок "*процесс*" применяется для обозначения действия или последовательности действий, изменяющих значение, форму представления или размещения данных. Для улучшения наглядности схемы несколько отдельных блоков обработки можно объединять в один блок. Представление отдельных операций достаточно свободно.
Решение		**Проверка условий** Блок *"решение"* используется для обозначения переходов управления по условию. В каждом блоке "решение" должны быть указаны вопрос, условие или сравнение, которые он определяет.
Модификация		**Начало цикла** Блок *"модификация"* используется для организации циклических конструкций. (Слово модификация означает видоизменение, преобразование). Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра для каждого повторения.
Предопределенный процесс		**Вычисления по подпрограмме, стандартной подпрограмме** Блок *"предопределенный процесс"* используется для указания обращений к вспомогательным алгоритмам, существующим автономно в виде некоторых самостоятельных модулей, и для обращений к библиотечным подпрограммам.

Ввод-вывод		**Ввод-вывод в общем виде** Внутри блока указываются переменные (через запятую) значения, которых необходимо получить извне. При выполнении такого блока исполнитель алгоритма приостанавливает свою работу и ожидает поступление исходных данных. После того как исходные данные будут получены и присвоены переменным, выполнение алгоритма будет продолжено.
Пуск-останов		**Начало, конец алгоритма, вход и выход в подпрограмму**
Документ		**Вывод результатов на печать** Внутри блока указываются переменные которые надо вывести на экран.

3. *Псевдокод*

Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов.

Псевдокод занимает промежуточное место между естественным и формальным языками. С одной стороны, он близок к обычному естественному языку, поэтому алгоритмы могут на нем записываться и читаться как обычный текст. С другой стороны, в псевдокоде используются некоторые формальные конструкции и математическая символика, что приближает запись алгоритма к общепринятой математической записи.

В псевдокоде не приняты строгие синтаксические правила для записи команд, присущие формальным языкам, что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд, рассчитанный на абстрактного исполнителя.

Однако в псевдокоде обычно *имеются некоторые конструкции, присущие формальным языкам*, что облегчает переход от записи на псевдокоде к записи алгоритма на формальном языке. В частности, в псевдокоде, так же, как и в формальных языках, есть *служебные слова*, смысл которых определен раз и навсегда. Они выделяются в печатном тексте жирным шрифтом, а в рукописном тексте подчеркиваются.

Единого или формального определения псевдокода не существует, поэтому возможны различные псевдокоды, отличающиеся набором служебных слов и основных (базовых) конструкций.

Примером псевдокода является школьный алгоритмический язык в русской нотации (школьный АЯ), описанный в учебнике А.Г. Кушниренко и др. "Основы информатики и вычислительной техники", 1991. Этот язык в дальнейшем мы будем называть просто "алгоритмический язык".

ПРИМЕР

```
алг Сумма квадратов (арг цел n, рез цел S)
    дано | n > 0
    надо | S = 1*1 + 2*2 + 3*3 + ... + n*n
нач цел i
    ввод n; S:=0
    нц для i от 1 до n
      S:=S+i*i
    кц
    вывод "S = ", S
кон
```

4. Программная реализация алгоритмов

Алгоритм решения задачи, заданный в виде последовательности команд на языке вычислительной машины (в кодах машины), называется *машинной программой.*

Язык программирования – формальная знаковая система, предназначенная для описания алгоритмов в форме, которая удобна для исполнителя (например, компьютера). Язык программирования определяет набор лексических, синтаксических и семантических правил, используемых при составлении компьютерной программы. Он позволяет программисту точно определить то, на какие события будет реагировать компьютер, как будут храниться и передаваться данные, а также какие именно действия следует выполнять над этими данными при различных обстоятельствах. Со времени создания первых программируемых машин человечество придумало уже более восьми с половиной тысяч языков. Каждый год их число пополняется новыми. Некоторыми языками умеет пользоваться только небольшое число их собственных разработчиков,

другие становятся известны миллионам людей. Профессиональные программисты иногда применяют в своей работе более десятка разнообразных языков программирования.

Функция: язык программирования предназначен для написания компьютерных программ, которые применяются для передачи компьютеру инструкций по выполнению того или иного вычислительного процесса и организации управления отдельными устройствами.

Компьютер при всей своей вычислительной мощи является быстрым, аккуратным, точным и вместе с тем совершенно «тупым» исполнителем. При использовании его при решении различных задач нельзя рассчитывать на то, что компьютер о чем-то догадается сам, ему для работы нужны очень точные и подробные инструкции.

В настоящее время в мире существует несколько сотен реально используемых языков программирования. Для каждого есть своя область применения.

Любой алгоритм, как мы знаем, есть последовательность предписаний, выполнив которые можно за конечное число шагов перейти от исходных данных к результату. В зависимости от степени детализации предписаний обычно определяется уровень языка программирования – чем меньше детализация, тем выше уровень языка.

По этому критерию можно выделить следующие уровни языков программирования:

- машинные;
- машинно-ориентированные (ассемблеры);
- машинно-независимые (языки высокого уровня).

Машинные языки и машинно-ориентированные языки – это языки *низкого уровня*, требующие указания мелких деталей процесса обработки данных. Языки же *высокого уровня* имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы. Эти языки более удобны для человека.

Языки высокого уровня делятся на:

- *процедурные (алгоритмические)* (Basic, Pascal, *C* и др.), которые предназначены для однозначного описания алгоритмов; для решения задачи процедурные языки требуют в той или иной форме явно записать процедуру ее решения;

- *логические* (Prolog, Lisp и др.), которые ориентированы не на разработку алгоритма решения задачи, а на систематическое и формализованное описание задачи с тем, чтобы решение следовало из составленного описания;
- *объектно-ориентированные* (Object Pascal, C++, Java и др.), в основе которых лежит *понятие объекта, сочетающего в себе данные и действия над нами.* Программа на объектно-ориентированном языке, решая некоторую задачу, по сути описывает часть мира, относящуюся к этой задаче. Описание действительности в форме системы взаимодействующих объектов естественнее, чем в форме взаимодействующих процедур.

Алгоритмический язык (как и любой другой язык) образуют три его составляющие: *алфавит, синтаксис* и *семантика.*

Алфавит – это фиксированный для данного языка набор основных символов, то есть "букв алфавита", из которых должен состоять любой текст на этом языке – никакие другие символы в тексте не допускаются.

Синтаксис – это *правила построения фраз*, позволяющие определить, правильно или неправильно написана та или иная фраза. Точнее говоря, синтаксис языка представляет собой набор правил, устанавливающих, какие комбинации символов являются осмысленными предложениями на этом языке.

Семантика определяет смысловое значение предложений языка. Являясь системой правил истолкования отдельных языковых конструкций, семантика устанавливает, какие последовательности действий описываются теми или иными фразами языка и, в конечном итоге, какой алгоритм определен данным текстом на алгоритмическом языке.

2.3. Основные алгоритмические конструкции (линейные структуры, ветвление, циклы).

Человеку в жизни и практической деятельности приходится решать множество различных задач. Решение каждой из них описывается своим алгоритмом, разнообразие этих алгоритмов очень велико. Можно выделить три основных вида алгоритмов:

1. линейной структуры,

2. разветвляющейся структуры,

3. циклической структуры.

Для краткости их называют просто: линейные, разветвляющиеся и циклические алгоритмы. Разнообразие же алгоритмов определяется тем, что любой алгоритм распадается на части, фрагменты и каждый фрагмент представляет собой алгоритм одного из трех указанных видов. Поэтому важно знать структуру каждого из алгоритмов.

В алгоритмах *линейной* структуры действия выполняются последовательно одно за другим:

В алгоритмах *разветвленной* структуры в зависимости от выполнения или невыполнения какого-либо условия производятся различные последовательности действий. Каждая такая последовательность действий называется *ветвью алгоритма*.

В алгоритмах *циклической* структуры в зависимости от выполнения или невыполнения какого-либо условия выполняется повторяющаяся последовательность действий, называющаяся телом цикла. *Вложенным* называется цикл, находящийся внутри тела другого цикла.

Различают циклы с *предусловием* (ПОКА) и *постусловием (ДО)*:

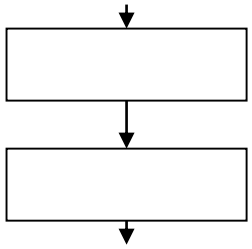
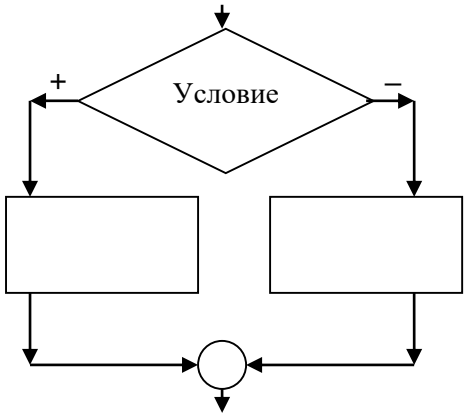
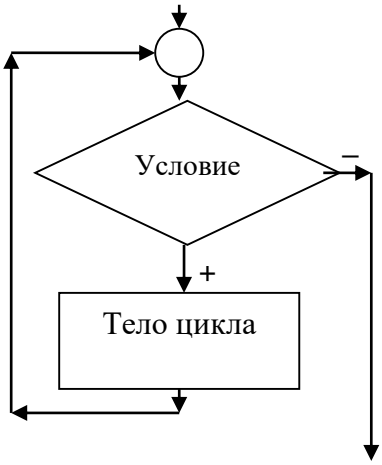
Итерационным называется цикл, *число повторений которого не задается*, а определяется в ходе выполнения цикла. В этом случае одно повторение цикла называется *итерацией*.

4. *Вспомогательный алгоритм* – алгоритм, который можно использовать в других алгоритмах, указав только его имя.

Например, вы в детстве учились суммировать единицы, затем десятки, чтобы суммировать двузначные числа содержащие единицы вы не учились новому методу суммирования, а воспользовались старыми методами.

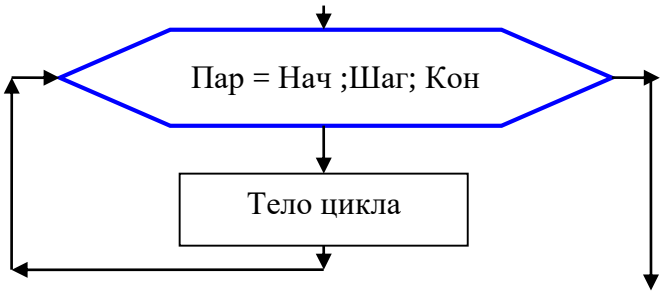
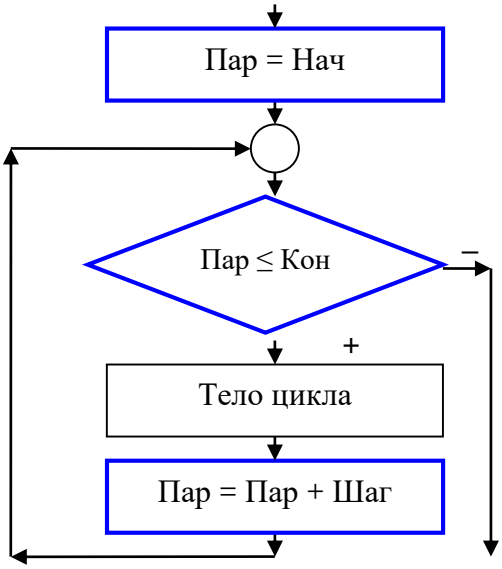
Рассмотрим основные управляющие конструкции алгоритма.

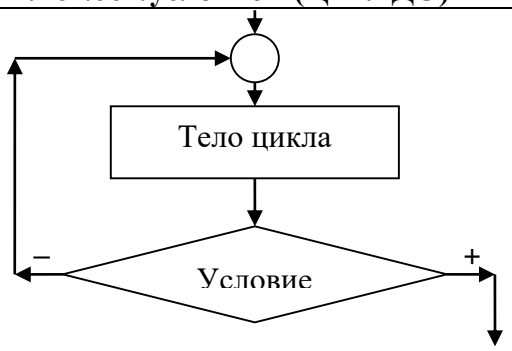
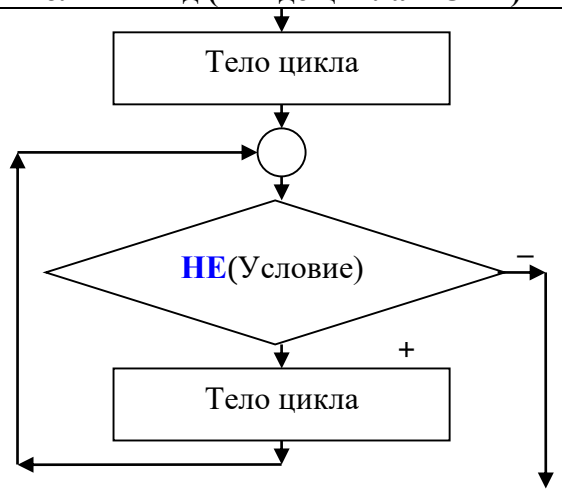
Основные управляющие конструкции

Линейная	**Ветвление**	**Повторение (Цикл ПОКА)**
 Два и более подряд идущих блока одного или разного вида выполняются один за другим.	 Разветвление алгоритмы в зависимости от текущего значения указанного *Условия* с последующим объединением. На одной из ветвей операторы могут отсутствовать. Операторы по ветви, помеченной знаком +, выполняются только в том случае, если выполняется *Условие*. Операторы по ветви, помеченной знаком –, выполняются только в том случае, если *Условие* НЕ выполняется. После выполнения операторов одной из ветвей выполняются операторы, следующие за данной конструкцией.	 Повторение выполнения операторов *Тела цикла*, **ПОКА** выполняется указанное *Условие*. Если *Условие* не выполняется, начинают выполняться операторы, следующие за эти циклом *Тело цикла* может не выполнится ни разу Это цикл с *пред*условием

Принцип структурного программирования. Этих трех конструкций достаточно для описания любого алгоритма!

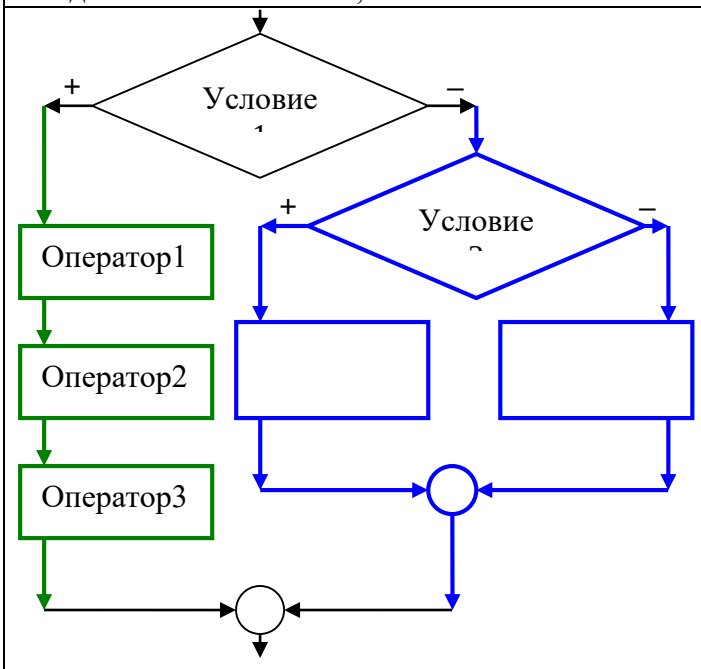
Дополнительные виды циклов

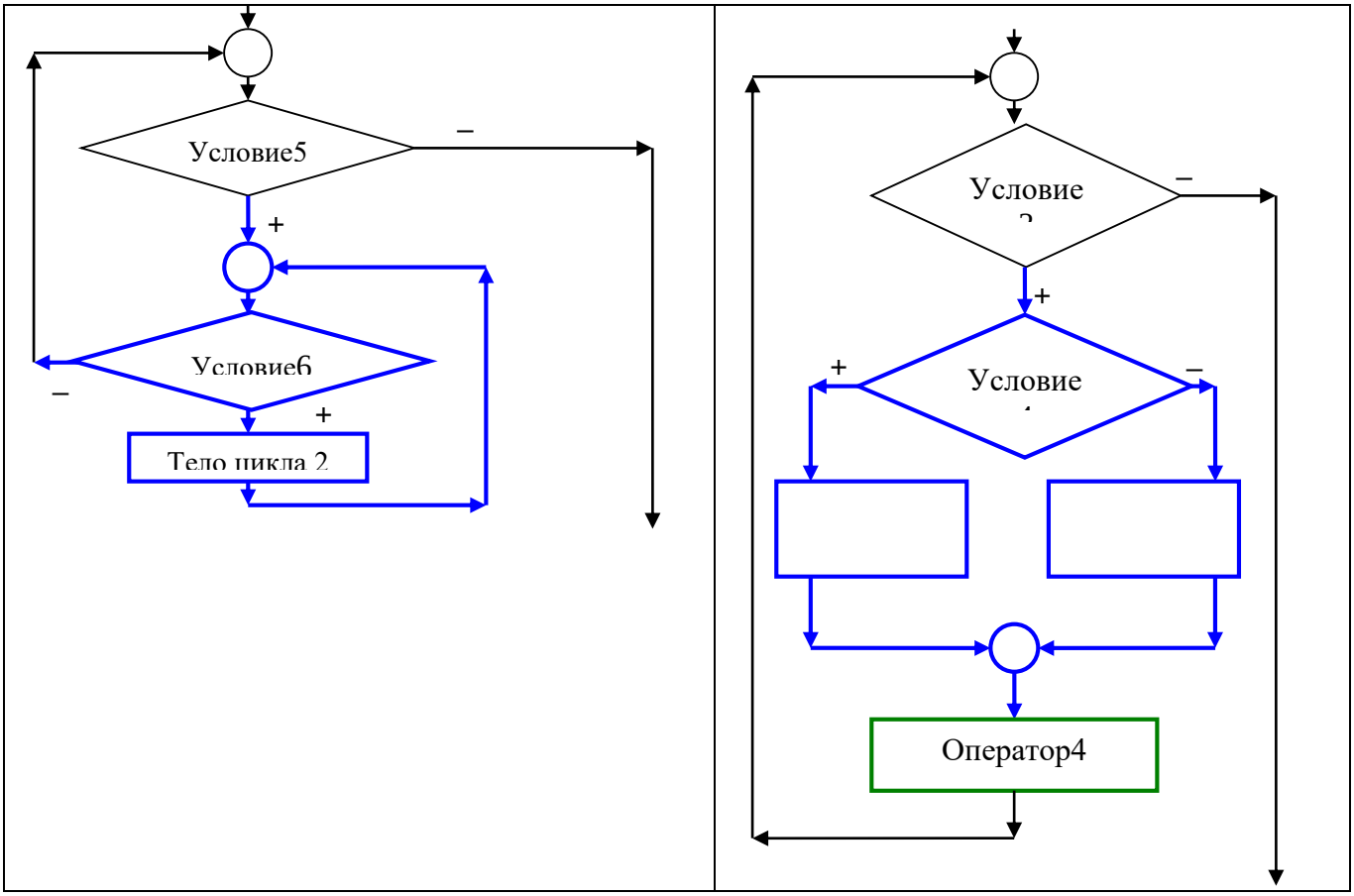
Параметрический (Цикл ДЛЯ)	**Полный вид (в виде цикла ПОКА)**
 Является краткой (1 новый блок вместо трех) записью для цикла с *пред*условием, управляемым целочисленным параметром, меняющим значение с *Нач* по *Кон* с шагом *Шаг* Например, *тело цикла* с параметром I=1;+1;10 выполнится 10 раз. При первом выполнении I=1, при втором I=2, на 10 шаге I=10.	Для положительного шага: 

То есть цикл выполнится **ДЛЯ** I = 1,2,3,…10	*Тело цикла* может не выполнится ни разу, если при положительном шаге Нач>Кон
Цикл с *пост*условием (Цикл ДО)	**Полный вид (в виде цикла ПОКА)**
 Перед проверкой *Условия* выполняется *Тело цикла*. Выполнение *Тела цикла* осуществляется, пока У*словие* НЕ истинно. Другими словами, *Тело цикла* выполняется **ДО** тех пор, пока Условие не станет истинным	 *Тело цикла* выполняется хотя бы 1 раз.

Комбинации конструкций

Конструкции могут следовать друг за другом или вкладываться друг в друга. Конструкции вкладываются друг в друга ЦЕЛИКОМ!

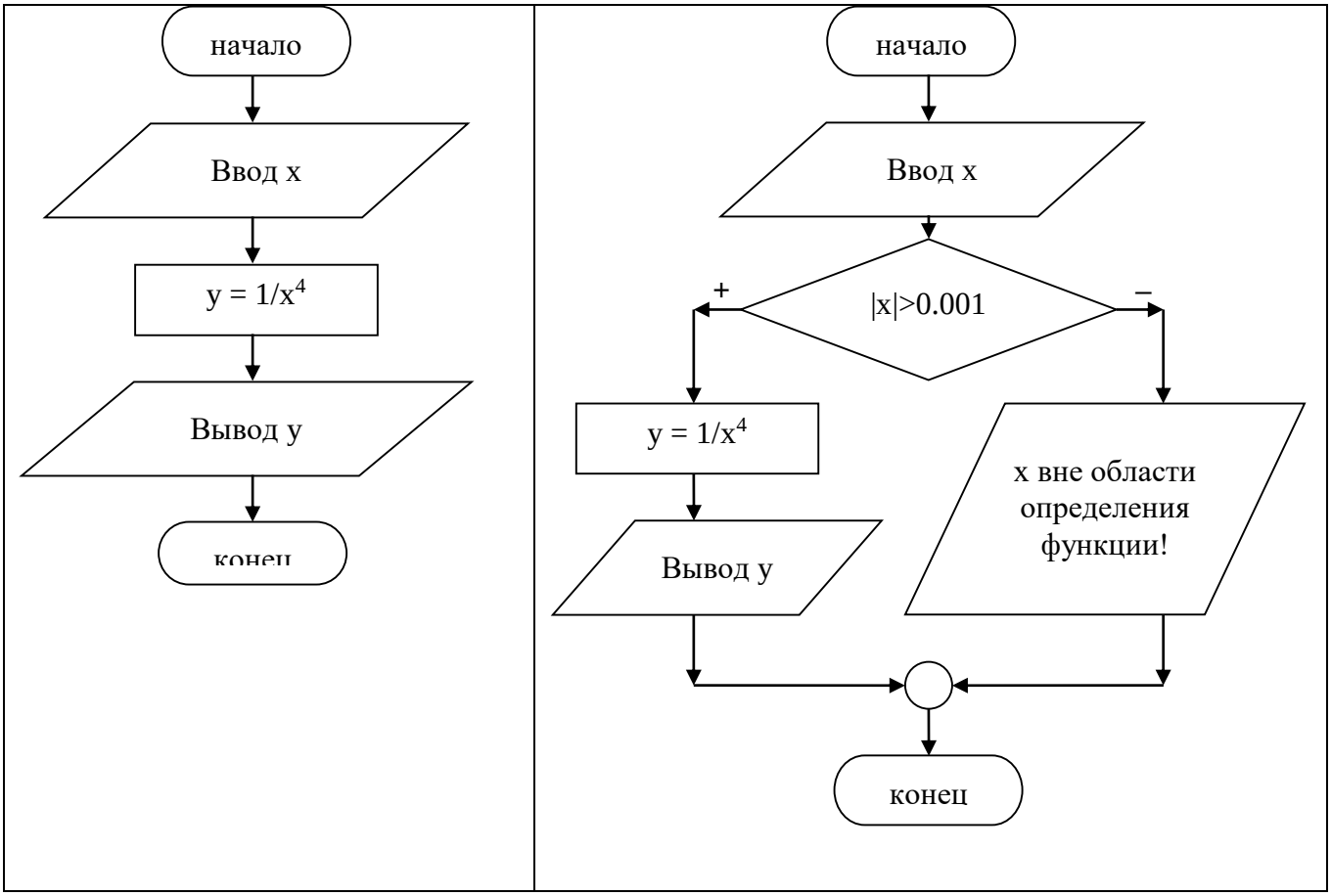
Следование и Ветвление, вложенное в Ветвление	Ветвление и Следование в Цикле ПОКА
	
Цикл ПОКА в Цикле ПОКА	



ПРИМЕР

Линейный алгоритм Ветвящийся

Вычисление значения функции $y=1/x^4$	Вычисление значения функции $y=1/x^4$, только для x, лежащих в *области определения* функции, и выводом сообщения в противном случае.



Раздел 3.
Язык программирования высокого уровня Pascal. Линейные конструкции языка Pascal. Условные конструкции языка Pascal. Циклические конструкции языка Pascal. Одномерные массивы.

3.1. Текст программы. Алфавит языка. Типы данных

Текст программы представляет собой последовательность строк, состоящих из символов, образующих алфавит языка. Максимальная длина строки составляет 126 символов.

Алфавит языка состоит из следующих символов:

1) заглавные и строчные латинские (26) и русские буквы, а также символ "подчеркивание". Этот символ считается буквой;

2) десять арабских цифр: 0, 1, 2, ..., 9;

3) двадцать два специальных символа: + - * / = > < . , ; : @ ' ( ) [ ] { } # $ ^ . К специальным символам также относятся <>, <=, >=, :=, (. .), (* *) .

Пробелы в Pascal являются разделителями.

Замечание 1. Русские буквы используются в программах только для записи комментариев и выдачи на экран поясняющих сообщений. Во всех других случаях использование русских букв **запрещено**.

Замечание 2. Язык программирования Pascal является *регистронезависимым*, то есть малые (строчные) и большие (прописные) буквы не различаются по значению: имена *abc* и *ABC* считаются равными и компилятором не различаются.

Идентификаторы или имена

Каждое данное должен иметь определенное имя (идентификатор). То же самое относится и к каждому типу данных, и к процедурам, и к функциям.

Идентификаторы – это имена констант, переменных, меток, типов, объектов, процедур, модулей, функций и других конструкций языка.

- длина имени может быть от 1 до 127 символов;
- идентификатор состоит из любых букв латинского алфавита, цифр, знака;
- подчёркивания; никакие другие символы в идентификаторе недопустимы;
- идентификатор не может начинаться с цифры;
- идентификатор не должен совпадать ни с одним из зарезервированных слов.

Идентификаторы, у которых одинаковы первые 63 символа, считаются идентичными.

Замечание 3. Нельзя использовать тип (переменную...) до тех пор, пока он (она) не определена**.**

Таблица 1. Зарезервированные слова

and	destructor	for	interface	or	shr	var
array	div	function	label	packed	string	while
asm	do	goto	library mod	procedure	then	xor
begin	downto	if	nil	program	to	
case	else	implementation	not	record	type	
case	end	in	object	repeat	unit	
const	exports	inherited	of	set	until	
constructor	file	inline		shl	uses	

3.2. Типы данных

Данные – величины, обрабатываемые программой. Имеется три основных вида данных: *константы, переменные и массивы* (рисунок 3.1).

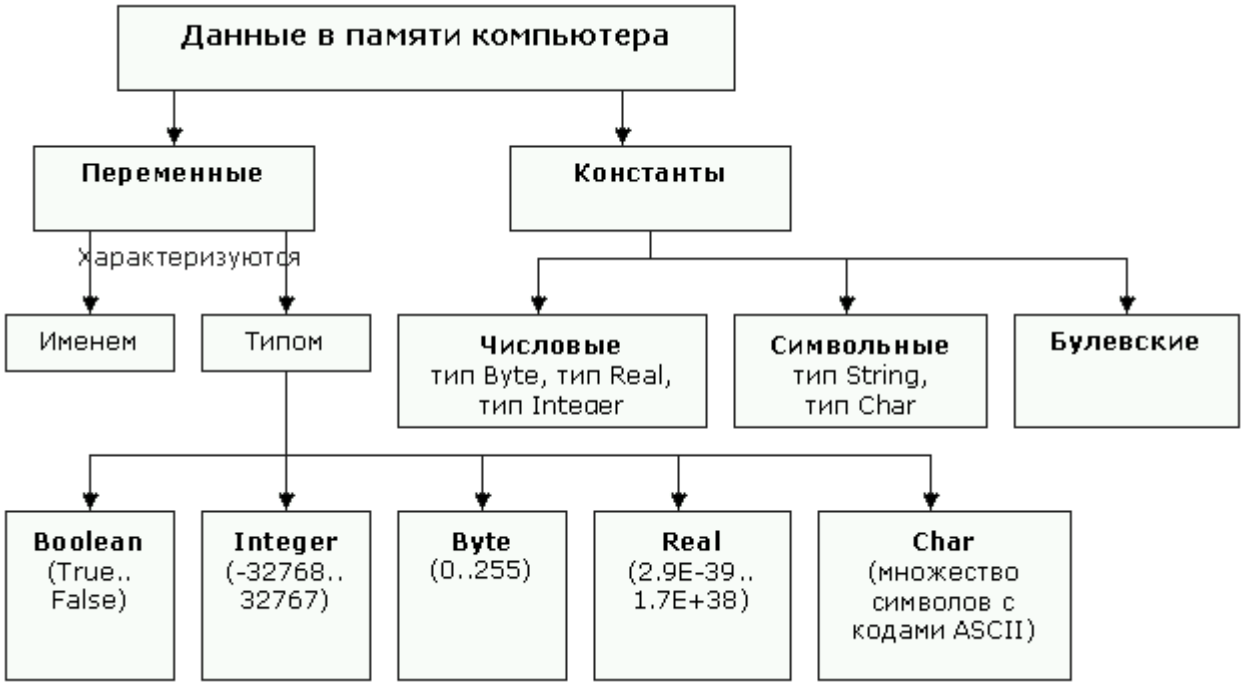


Рисунок 3.1. Типы данных

К простейшим конструкциям языка Pascal относятся константы, переменные, стандартные функции и выражения.

Константы – это данные, которые зафиксированы в тексте программы и не изменяются в процессе ее выполнения.

В качестве констант в Pascal могут использоваться целые, вещественные логические, символы, строки символов.

Переменная – это область памяти компьютера. Любая переменная – величина, способная изменяться в процессе выполнения программы.

С каждой переменной программы связывается одна и только одна её характеристика, называемая типом.

После описания переменной в компьютере выделяется для нее выделяется память, где храниться ее значение.

Переменная имеет имя и значение. Желательно, чтобы имя переменной было логически связано с назначением переменной. (Например, для решения квадратного уравнения $ax^2+bx+c=0$, a, b, c – свободные члены, D – дискриминант, $x1$, $x2$ – корни уравнения).

Переменные в программе могут получать свои значения двумя способами:

1) ввода значения с клавиатуры (операторы Write и Read);

2) путем присваивания в программе

Значение переменной можно изменять, записывая в нее новое значение. Одним из основных операторов является оператор присваивания, с помощью которого можно присвоить переменной, стоящей слева от знаков ":=", значение выражения, стоящего справа от ":=".

Команда присваивания "стирает" предыдущее значение переменной и *"придает"* ей новое значение. Например, допустим переменная *a* – целого типа, тогда после оператора

a:=2;

в область памяти, которая выделена для переменной *a* запишется значение 2. Если потом выполнить оператор

a:=*a*+1;

то компьютер сначала вычислит выражение справа от знака присваивания. Вместо *a* он берет значение, записанное в область памяти, выделенное для этой переменной, то есть число 2. К нему прибавляет 1, получает 3. Эту 3 записывает обратно в область памяти, при этом старое значение 2 стирается.

Общий вид:

Имя переменной: = Выражение

Выражения – предназначаются для выполнения необходимых вычислений, состоят из констант, переменных, указателей функций (например, exp(x)), объединенных знаками операций.

Выражения записываются в виде *линейных последовательностей символов* (без подстрочных и надстрочных символов, "многоэтажных" дробей и т.д.), что позволяет вводить их в компьютер, последовательно нажимая на соответствующие клавиши клавиатуры.

Например, такое выражение как $y = \frac{a+b}{a-b}$ будет записано как: $y = (a+b)/(a-b)$.

Выражение состоит из операторов и операндов. Операторы находятся между операндами. Операндами выражений могут быть числа (константы) и переменные. Операторы обозначают действия, выполняемые над операндами.

Различают выражения арифметические, логические и строковые.

Операторы (команды). Оператор – это наиболее крупное и содержательное понятие языка***: каждый оператор представляет собой законченную фразу языка и определяет некоторый вполне законченный этап обработки данных.*** В состав операторов входят:

- ключевые слова;
- данные;
- выражения и т.д.

Операторы подразделяются на исполняемые и неисполняемые. **Неисполняемые** операторы предназначены для описания данных и структуры программы, а *исполняемые* – для выполнения различных действий (например, оператор присваивания, операторы ввода и вывода, условный оператор, операторы цикла, оператор процедуры и д. р.).

Массивы – последовательности однотипных элементов, число которых фиксировано и которым присвоено одно имя. Положение элемента в массиве однозначно определяется его индексами (одним, в случае одномерного массива, или несколькими, если массив многомерный). Иногда массивы называют таблицами.

Тип – это множество значений переменной вместе с множеством операций, которые можно выполнять над элементами этого множества.

Приписывая переменной некоторый тип, мы тем самым явно определяем множество значений, которые можно присвоить этой переменной, а также операции, с помощью которых можно манипулировать ее значениями.

Любые данные, т.е. константы, переменные, значения функций или выражения в Pascal характеризуются своими типами. Тип определяет множество допустимых значений, которые может иметь тот или иной объект, а также множество допустимых операций, которые применимы к нему.

В языке Pascal существуют *скалярные (простые)* и *структурные (составные)* типы, состоящие из элементов простых типов данных.

К *скалярным типам относятся* стандартные типы и типы, определяемые пользователем.

3.2.1. Скалярные типы данных

1. Стандартные типы данных

Они включают: *целые, действительные, символьный, логический, адресный* типы.

1. Целые – не имеют дробной части.

2. Числа действительного (вещественного) типа – числа записываются с фиксированной точкой в виде целой и дробной частей (например, -4.5, 6.78) или с плавающей точкой для значений с десятичным порядком (например число 170000 = $1.7*10^5$ и запишется в виде 1.7E+5, а число 0.00056 = $5.6*10^{-4}$ – в виде 5.6E-4, т.е. *e*(E) означает десятичный порядок и имеет смысл «умножить на 10 в степени» (символ E - экспоненциальная часть).

3. Логические.

4. Символьный тип.

5. Адресный.

Таблица 2. Простые типы данных

Тип	Название
целый тип	
Integer	Принимают значения из промежутка от -32768 до 32767. В памяти для переменной этого типа выделяется 2 байта.
ShortInt (малое или короткое целое)	от – 128 до 127. В памяти для переменной этого типа выделяется 8 бит со знаком или 1 байт со знаком)
LongInt (большое или длинное целое)	от – 2147483648 до 2147483647 (от -2^{31} до $2^{31}-1$). В памяти для переменной этого типа выделяется 4 байта со знаком
Byte (натуральное малое) длиной в байт	Байтовые. От 0 до 255. В памяти для переменной этого типа выделяется 1 байт.
Word (длиной в слово)	от 0 до 65535. В памяти для переменной этого типа выделяется 2 байта.

Вещественные типы данных хранятся в памяти компьютера иначе, чем целые. Внутреннее представление вещественного числа состоит из двух частей - мантиссы и порядка, и каждая часть имеет знак. Например, число -0,087 представляется в виде $-0{,}87x10^{-1}$, и в памяти хранится мантисса -87 и порядок -1

Существует несколько вещественных типов, различающихся точностью и диапазоном представления данных. Точность числа определяется длиной мантиссы, а диапазон –длиной порядка.

вещественный тип	
Real	(по модулю) от 2.9E-39 до 1.7E+38. В памяти для переменной этого типа выделяется 6 байт, количество цифр после запятой может достигать 11-12 цифр.
Singel	от 1.5E-45 до 3.4E+45. Точность составляет 7-8 знаков после запятой. В памяти отводится 4 байта
Comp (сложное)	от -9.2E+18 до 9.2E+18 Точность составляет 19-20 знаков после запятой. В памяти компьютера отводится 8 байт
логический тип	
Boolean, ByteBool, WordBool, LongBool	Логические. Принимают только два значения: **True** (истина или 1) и **False** (ложь или 0). **Boolean, ByteBool** занимают 1 байт, переменная **WordBool** – 2 байта**, LongBool** – 4 байта **Boolean** – это предпочтительный тип, использующий меньше памяти. Остальные типы обеспечивают совместимость с другими языками и средой Windows.
символьный тип	
Char	Литерные. В качестве своего значения могут иметь один символ, заключенный в апострофы. Например, 'C'. В памяти для переменной этого типа выделяется 1 байт.
адресный тип	
Pointer	Адресный тип определяет переменные, которые могут содержать значения адресов данных или фрагментов программы. Для хранения адреса требуется 4 байта.

2. Типы данных определяемые пользователем

Они включают: *перечисляемый* и *интервальный* типы.

Перечисляемый тип данных

Перечисляемый тип представляет собой ограниченную упорядоченную последовательность скалярных констант, составляющих данный тип. Значение каждой константы задается ее именем. Имена отдельных констант отделяются друг от друга запятыми, а вся совокупность констант, составляющих данный перечисляемый тип, заключается в круглые скобки.

Пример описания перечисляемого типа:

type

a=(*RED, ORANGE, YELLOW, GREEN, LIGHT_BLUE, BLUE, VIOLET*);

Теперь переменная с этого типа может принимать следующие значения: *RED*, *ORANGE*, *YELLOW*, *GREEN*, *LIGHT_BLUE*, *BLUE* и *VIOLET*.

Замечание 4. Каждое значение является константой своего типа и может принадлежать только одному из перечисляемых типов. Т.е. если мы хотим записать новый тип, например a1, то он не должен содержать ни одной такой же константы: *RED*, *ORANGE*, *YELLOW*, *GREEN*, *LIGHT_BLUE*, *BLUE*, *VIOLET*.

Интервальный тип данных

Отрезок любого порядкового типа может быть определен как интервальный или ограниченный тип. Отрезок задается диапазоном от минимального до максимального значения констант, разделенных двумя точками. В качестве констант могут быть использованы константы, принадлежащие к целому, символьному, логическому или перечисляемому типам. Скалярный тип, на котором строится отрезок, называется базовым типом.

Минимальное и максимальное значения констант называются нижней и верхней границами отрезка, определяющего интервальный тип. Нижняя граница должна быть меньше верхней.

Над переменными, относящимися к интервальному типу, могут выполняться все операции и применяться все стандартные функции, которые допустимы для соответствующего базового типа.

Пример описания интервального типа:

type

interval=0..50;

t=-100..100;

3.2.2.Структурный тип данных

Таблица 3**.** Составные типы данных

Тип	**Название**
String	Строки. Состоят из значений литерного типа (набор символов). Например, '*Summa S*='
Array	Массивы. Представляют собой совокупность элементов одного (любого) типа.
Record	Записи. В отличие от предыдущего типа могут состоять из элементов различных типов.
Set	Множества. Представляют собой совокупность элементов одного типа (целого положительного или литерного).
File	Файловые. Позволяют организовать доступ к файлам из программы.

3.3. Структура Pascal-программы

Программа на языке программирования Pascal состоит из трех основных частей (рисунок 3.2):

1. *Заголовка* – содержит имя (название) программы;

2. *Декларация* (*раздел описаний*) – раздела, содержащий описание всех элементов программы;

3. *Блок описания операторов* – набора команд (операторов), выполняемых компьютером.

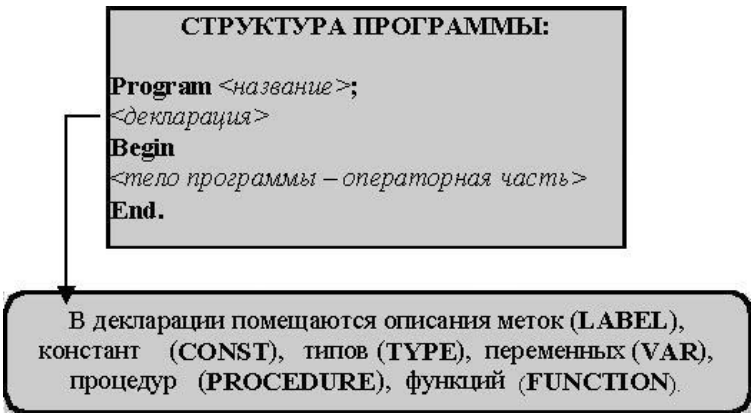


Рисунок 3.2. Общая структура Pascal-программы

Заголовок чаще всего состоит из служебного слова *Program* и имени программы.

Program ␣ имя;

Имена – это слова, используемые автором программы (программистом) для обозначения каких-либо элементов. Именем может быть любая последовательность букв и цифр, начинающаяся с буквы, например: *Elena, Zadacha_1, Zadacha2* (вспомните символ подчеркивания, ведь это буква). Понятно, что в качестве имен нельзя использовать служебные слова и названия участвующих в программе переменных.

Раздел описаний.

1) *Uses* {Описание используемых модулей}

После служебного слова *uses* через запятую перечисляются модули, процедуры и функции которые, используются в программе. В конце ставится " ; ". Это могут быть как стандартные модули Pascal – Crt, Graph, так и модули, разработанные пользователем.

Например,

```
Uses Crt, Graph;
```

Теперь, когда описаны нужные вам модули, вы можете в любой части вашей программы, после служебного слова Begin использовать процедуры и функции из этих модулей.

2) Раздел описание констант *Const*

Список констант и их значений прописываются через **;** .

Например,

```
Const min=1000; max=4000; center=1500; name='Pavel';
```

3) Список типов пользователя *Type*

Позволяет определять новый набор свойств, который затем будет использован для создания переменных, обладающих этими свойствами, прописываются через **;** .

```
Type
   t1=<вид типа>;
   t2=<вид типа>;
.......
```

где *t1, t2* – идентификаторы вводимых типов, а <*вид типа*> – вид типа

Например,

```
type
  int=Integer;
  b=boolean;
```

4) Раздел описания переменных *Var*

Определяет элементы, используемые для хранения и изменения необходимых программе значений. Данный раздел описания обозначается ключевм словом **Var** за которым через точку с запятой перечисляются

<имя переменной>:<тип переменной>.

Пример описания переменных:

```
Var a: word; b: byte=160; c, d, e: Integer;
```

Замечание. В разделе описания переменных, переменным можно задавать начальное значение (*b*: *byte*=160;). Переменные одинакового типа записываются через запятую, затем после двоеточия указывается их тип (*c, d, e*: *Integer*;).

5) Раздел описания процедур *Procedure*

Определяет именованный набор действий, к которому можно обращаться по заданному имени.

6) Раздел описания функций *Function*

Определяет именованный набор действий, в результате работы возвращающий значение. К этому набору действий можно обращаться по заданному имени.

Все элементы, используемые в программе (переменные, константы и т.д.), обязательно должны быть описаны в **декларации**. Если это требование будет нарушено, то компьютер выведет на экран сообщение об ошибке.

Блок описания операторов (*Операторная часть*) содержит перечень операторов, которые компьютер должен выполнить. Эта часть заключается в операторные скобки *Begin* и *End*. После служебного слова *End* **ставится точка!**

Каждый оператор или команда заканчивается символом ' **;** ' - этот символ показывает окончание одной команды и начало следующей. Команды могут содержать произвольное количество пробелов, а также могут быть разделены на несколько экранных строк и наоборот - несколько команд может быть в одной строке. В принципе вся программа на языке Pascal может быть записана в одну экранную строку. Но следует отметить, что лучше писать программу так, чтобы ее легко было читать и понимать, легко было использовать. Для этого в программе используются комментарии, пробелы, пустые строки. Желательно смысловые части выделять одинаковыми отступами от начала строки.

Общий вид операторной части:

```
Begin   { ключевое слово, указывающее, что за ним следует текст программы}
    Оператор 1;
    Оператор 2;
    Оператор n
End. { указывает на окончание программы }
```

Комментарии в программе

Комментарий – это последовательность символов, заключенная в фигурные скобки { и }. Служит для пояснения программы или отдельных ее частей. Комментарии

игнорируются компилятором, и поэтому не влияют на выполнение программы. Наличие комментариев делает программу понятной и удобной для чтения.

```
{Это пример комментария}
```

Выполнение программы

Процесс выполнения программы состоит из двух шагов:

- *компиляции* – здесь осуществляется перевод программы на язык машинных команд;
- *линковки (компоновки, сборки)* – здесь происходит сборка выполняемой программы из откомпилированных модулей, подключение необходимых библиотек (модулей) и еще ряд операций.

3.4. Основные операции и функции языка Pascal

Операции подразделяются на арифметические операции, операции отношения, логические операции, операции со строками, операции над множествами, операцию @ (операция получения адреса).

Таблица 4. Арифметические операции

Операция	**Название**
+	Сложение
–	Вычитание
*	Умножение
/	Деление

Таблица 5. Операции отношения

Операция	**Название**
=	Равно
<>	Не равно
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно

Таблица 6. Логические операции

Логическая операция	**Ее значение**	**Примеры записи**	**Значение примера**
And	Логическое "**И**"	(x<7) **and** (x>3)	*x* меньше 7 и *x* больше 3 (3<x<7)
OR	Логическое "**ИЛИ**"	(y>100) **or** (y<10)	*y* больше 100 или *y* меньше 10 (*y*<10, *y*>100)
Not	Логическое "**НЕ**"	**not** (x=2)	не *x* равно 2
Xor	Исключающее "**ИЛИ**"		

Логические операции чаще всего используются для организации сложных условий. Операция *Not* является одноместной, а остальные операции – двуместные. Приведем таблицу истинности для этих операций.

Таблица 7. Результаты выполнения логических операций

Значения операндов		**Результат операции**			
X	***Y***	**Not *X***	***X* And *Y***	***X* Or *Y***	***X* Xor *Y***
False	**False**	**True**	**False**	**False**	**False**
False	**True**	**True**	**False**	**True**	**True**
True	**False**	**False**	**False**	**True**	**True**
True	**True**	**False**	**True**	**True**	**False**

Таблица 8. Основные функции

Название функции, обозначение	**Обозначение в языке Pascal**
sin *x*	Sin(*x*)
cos *x*	Cos(*x*)
tg *x*	Sin(*x*)/Cos(*x*)
arctg *x*	Arctan(*x*)
e^x	Exp(*x*)
ln *x* ($\log_e x$)	Ln(*x*)
x^2	Sqr(*x*)
$\sqrt{x}$	Sqrt(*x*)
x^a	$\exp(a*\ln(x))$
|*x*| (модуль *x*)	Abs(*x*)

[*x*] (целая часть от *x*)	Int(*x*)
{*x*} (дробная часть от *x*)	Frac(*x*)
Округление до целого	Round(*x*)
Генерация "случайного" целого числа на промежутке от 0 до *k*-1	Random(*k*)
Уменьшение значения *x* на *y (если нет y, то на 1)*	Dec (*x*,*y*)
Увеличение значения *x* на *y (если нет y, то на 1)*	Inc(*x*,*y*)
X div Y	Целочисленное деление X на Y
X mod Y	Нахождения остатка от деления X на Y

Например,

10:3 = 3 (1) (**mod - 1, div - 3**)

Для вычисления значений других функций следует пользоваться тождествами:

$$\arcsin x = arctg \frac{x}{\sqrt{1-x\cdot x}};$$

$$\arccos x = \frac{\pi}{2 - arctg \frac{x}{\sqrt{1-x\cdot x}}};$$

$$arcctg\ x = \frac{\pi}{2 - \arctan x};$$

$$\log_b a = \frac{\ln a}{\ln b};$$

$$a^x = e^{x\ln a}\ (a>0).$$

Замечание.

1. Аргументы всех функций всегда заключаются в круглые скобки!

2. Аргументы функций **Sin(*X*)** и **Cos(*X*)** должны быть выражены в радианах. Для этого можно воспользоваться следующими формулами:

$$X^{(рад)} = X^{o} * \pi/180 \qquad X^{o} = X^{(рад)} * 180/\pi$$

3. Число $\pi = 3{,}14...$ записывается через раздел описания констант. Const pi=3.1415;

4. Порядок выполнения операций определяется скобками, а при их отсутствии – согласно приоритету операций:

1. Операция отрицания not.

2. Операции: *, /, div, mod, and.

3. Операции: +, -, or.

4. Операции отношения: <=, <, =, <>, >, >=.

3.5. Линейная программа

Линейная программа – это программа, в которой операторы выполняются в порядке следования, без проверки каких-либо условий. Каждый шаг алгоритма выполняется только один раз, после шага под номером *i,* обязательно выполняется шаг под номером *i*+1. Линейная структура предполагает последовательное выполнение действий, без их повторения или пропуска некоторых действий. Линейные программы используются достаточно редко.

Для реализации алгоритмов линейной структуры используются операторы:

- Оператор (процедура) вывода – **Write ( );** или **Writeln ( );**

- Оператор (процедура) ввода – **Read ( );** или **Readln ( );**

- Оператор присваивания – :=

Оператор присваивания мы рассмотрели уже выше, теперь рассмотрим операторы ввода и вывода.

3.5.1. Операторы ввода и вывода

Для общения компьютера с человеком или, по-другому, *диалога*, необходимо, чтобы программа выводила на экран разную информацию и запрашивала данные. Для этого существуют операторы ввода (read) и вывода (write) данных.

Оператор вывода Write (WriteLn)

Оператор **WRITE** предназначен для вывода на экран монитора сообщений и значения выражений или переменных.

Общий вид которого следующий (рисунок 3.3):

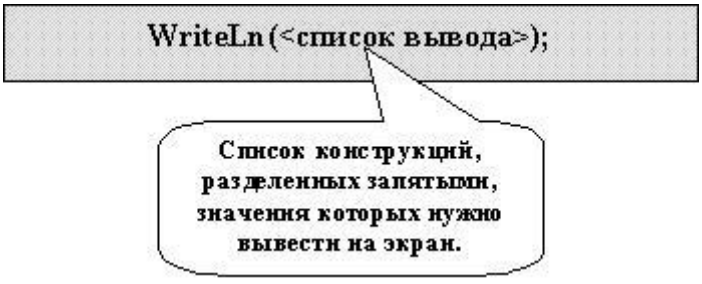


Рисунок 3.3. Общий вид оператора *WriteLn*

Список вывода (список переменных) заключается в апострофы (одинарные кавычки), то что заключено в апострофы при работе вашей программы появится на экране монитора без изменений.

Список вывода может состоять:

- *из поясняющих фраз.* В этом случае такая фраза заключается; Например, ***Write*('текст').**
- *из имен переменных.* В этом случае на экран выводятся значения этих переменных; Например, ***Write*('*x*=').**
- *из выражений.* Компьютер вычисляет значение этого выражения и выводит его на экран. Например, ***Write*('*x*+3=').**

Буквы ***Ln*** (сокращение английского слова *Line – строка, линия*), добавляемые к операторам ввода/вывода, осуществляют перемещение курсора в начало следующей строки. Оператор *WriteLn* часто используется *для пропуска пустых строк.*

Для того, чтобы вывести результате вычислений на экран, необходимо записать:

```
write ('D',D:5:2);
```

Здесь использован формат вывода, где первая цифра означает общее количество знаков в выводимом числовом значении, а вторая цифра - количество знаков после запятой

Оператор ввода Read (ReadLn)

Для ввода данных с клавиатуры в языке программирования Pascal используется оператор ввода **Read, ReadLn** – ввод значения переменных в память компьютера.

Общий вид которого следующий:

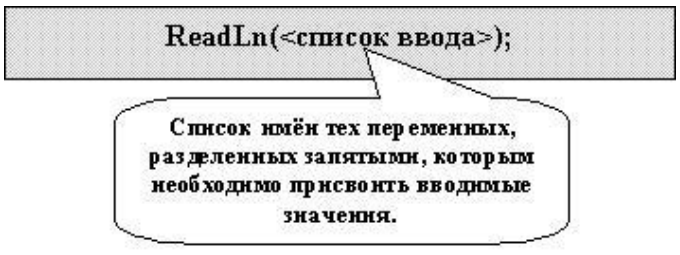


Рисунок 3.4. Общий вид оператора *ReadLn*

Встретив в программе оператор ***ReadLn*(*x,y*);** компьютер будет "ждать" ввода исходных данных.

Следует заметить, что данные вводятся после набора на экране всей программы и запуска её на выполнение. Если вводятся числа, то они отделяются друг от друга ***одним или несколькими пробелами***. Ввод завершается нажатием клавиши **Enter**.

После того, как ввод произведен, продолжается дальнейшее выполнение программы.

В операторе ввода можно указывать список имён тех переменных, которым надо присвоить вводимые значения. Так один оператор ввода ***Read(x,y)*** полностью равносилен двум операторам ввода: ***Read(x), Read(y)***.

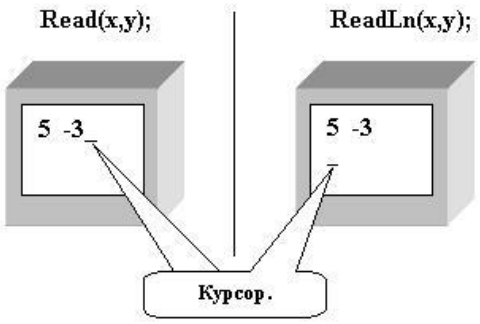


Рисунок 3.5. Различие в использовании операторов

Из данного рисунка видно, что после выполнения оператора ***ReadLn*** осуществляется перенос курсора в начало следующей строки, в то время, как при выполнении оператора ***Read*** курсор остается за последним элементом. Мы советуем использовать в большинстве случаев оператор ***ReadLn***.

ПРИМЕР. Программа-приветствие: выводит на экран сообщение.

```
Program hello;
Begin
 Write(' * * * Hello * * *');
End.
```

В этой программе используется только одна команда, в которой нет ни одной переменной, а просто текст, поэтому раздел описания опущен (описывать нечего).

ПРИМЕР. Вычислить площадь треугольника по формуле Герона, где *a*, *b*, *c* – стороны треугольника. Формула Герона: $S=\sqrt{p(p-a)(p-b)(p-c)}$, где p – полупериод треугольника

```
program  zadacha1;
uses crt;
```

```
var  S,p:real;
a,b,c:integer;
begin
clrscr;
     writeln ('vvedite a');
     read (a);
     writeln ('vvedite b');
     read (b);
     writeln ('vvedite c');
     read (c);
     p:=(a+b+c)/2;
     S:=sqrt(p*(p-a)*(p-b)*(p-c));
     writeln ('S=',S:2:3);
end.
```

ПРИМЕР. Дано $a = -2$, $b = 0{,}00005$, $c = 100000$. Записать a, b, c в экспоненциальной форме. Составить программу для вычисления значения выражения $y = \frac{1+\cos^2(\pi - b)}{a + c^{1/2} + \ln|a|}$.

Вывести значение y на экран.

```
program prog; //Заголовок программы
const a=-2; b=5e-5; c=1e+5; //Описание констант в экспоненциальной форме
var y:real; //Описание переменных
     begin //Начало программы
     //Основные вычисления
          y:=(1+sqr(cos(pi-b)))/(a+exp(1/2*ln(c))+ln(abs(a)));
          writeln('y=',y) //Вывод результата на экран
     end. //Конец программы
```

ПРИМЕР. Дано: $d = 200$, a и b – любые вещественные числа. Записать x в экспоненциальной форме. Составить программу для вычисления значения выражения $y = a + d \cdot x^2$, где $x = b \cdot \sin a$. Вывести значения x и y на экран.

```
program prog; //Заголовок программы
const d=2e+2; //Описание констант
var y,x,a,b:real; //Описание переменных
begin //Начало программы
          writeln('Введите a, b'); //Вывод на экран подсказки
          read(a,b); //Ввод данных с клавиатуры
          x:=b*sin(a);
          y:=a+d*sqr(x); //Основные вычисления
          writeln('y=',y) //Вывод результата на экран
end. //Конец программы
```

ПРИМЕР. Дано n=0,1241456, h – любое число. Составить программу для вычисления функций: $c = a \cdot h^3 \cdot \sin^2 b^3$

$$b = 1 - \sqrt{\frac{3}{3 + \left|\operatorname{tg} h^2 - \sin h\right|}},\ a = \frac{\sin(b-n)}{e^{\sin h} \cdot \cos^2(b-h) \cdot \frac{1}{\ln|b|}}.$$

```
program prog;
const n=1.241456e-1;
var h:integer; a,b,c:real;
begin
     writeln('Введите h=');
     read(h);
     b:=1-sqrt(3/(3+abs(sin(sqr(h))/cos(sqr(h))-sin(h))));
     a:=sin(b-n)/exp(sin(h))*sqr(cos(b-h))*1/ln(abs(b));
     c:=a*exp(3*ln(h))*sqr(sin(exp(3*ln(b))));
     writeln('b=',b,' a=',a,' c=',c);
end.
```

ПРИМЕР. Дан радиус круга. Вычислить длину окружности и площадь круга. Результаты вывести на экран.

```
program prog;
var r,s,l: real;
begin
```

```
        writeln ('Введите радиус');
        read(r);
        s:=pi*sqr(r);
        l:=2*pi*r;
        writeln('Площадь круга=',s:8:2);
        writeln('Длина окружности=',l:8:2);
end.
```

3.6. Программирование алгоритма ветвящейся структуры

Алгоритм ветвящейся структуры – это такой алгоритм, в котором выбирается один из нескольких возможных путей (вариантов) вычислительного процесса.

Каждый подобный путь называется *ветвью алгоритма.*

Признаком разветвляющегося алгоритма является наличие операций условного перехода, когда происходит проверка истинности некоторого логического выражения (проверяемое условие) и в зависимости от истинности или ложности проверяемого условия для выполнения выбирается та или иная ветвь алгоритма.

Для программной реализации этого типа алгоритмов в языке программирования Pascal можно использовать условный оператор.

Условный оператор

Условный оператор предназначен для выполнения какого-нибудь действия при выполнении определенного условия. То есть условный оператор позволяет проверить некоторое условие и в зависимости от результатов проверки выполнить то или иное действие.

Различают *простой оператор* и *составной*. В *простом* операторе после условия выполняется один оператор. Если после проверки условия следует выполнить не один, а группу операторов, то эта группа заключается в операторные скобки (*begin – end*) и оператор называется *составным.*

В качестве условия могут использоваться операции сравнения, логические операции или переменные логического типа *Boolean.*

Также условный оператор подразделяется с простым и сложным условием. Простое условие – это условие, которое включает в себя два числа или две переменные или два арифметических выражения, которые сравниваются между собой посредством операции сравнения

Например, $a<b$.

И *сложное* – это последовательность простых условий, объединенных между собой знаками логической операции

Например, ($a<b$) and ($b<c$).

Рассматривают полную и неполную форму условного оператора.

1) полная форма простого оператора

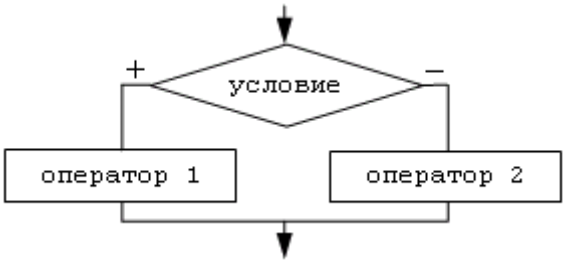


Рисунок 3.6. Полная форма простого оператора

Структура программы:

```
If <условие> then <оператор1> else <оператор2>;
```

Если условие выполняется, то действует оператор 1, иначе действует оператор 2.

Замечание. Перед служебным словом **else** точка с запятой не ставится!

2) неполная форма простого оператора

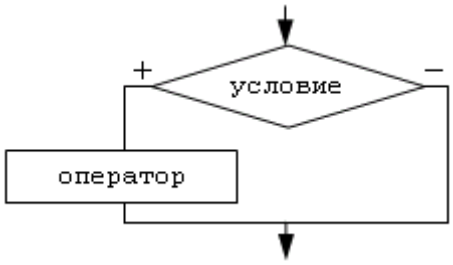


Рисунок 3.7. Неполная форма простого оператора

Структура программы:

```
If <условие> then <оператор1>;
```

Если условие выполняется, то действует оператор 1, другие варианты не рассматриваются.

3) неполная форма с составного оператора

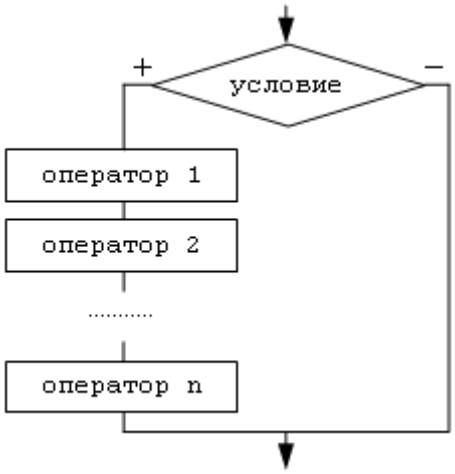


Рисунок 3.8. Неполная форма составного оператора

Структура программы:

```
If <условие> then
                begin
                    <оператор 1>;
                    <оператор 2>;
                    ...
                    <оператор n>;
                end;
```

4) полная форма составного оператора

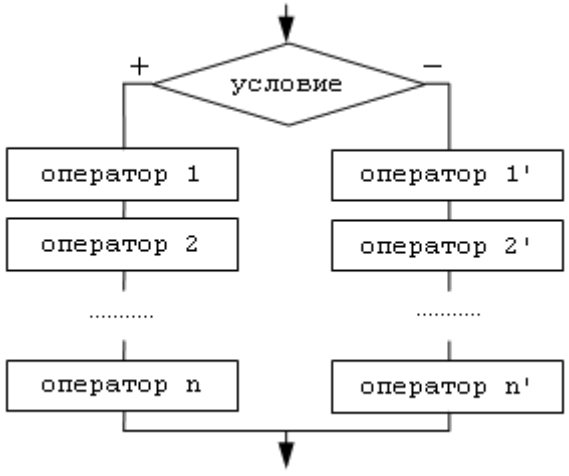


Рисунок 3.9. Полная форма составного оператора

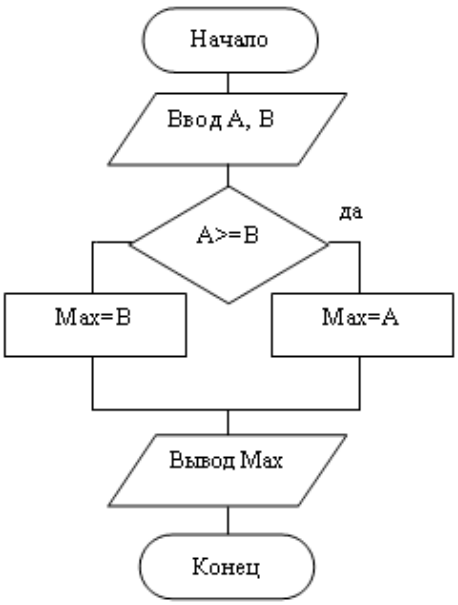
Структура программы:

```
If <условие> then
                begin
                    <оператор 1>;
                    <оператор 2>;
```

```
                ...
                <оператор n>;
            end
        else
            begin
                <оператор 1’>;
                <оператор 2’>;
                ...
                <оператор n’>;
            end;
```

ПРИМЕР. Найти максимальное значение из двух целых чисел *A* и *B*.

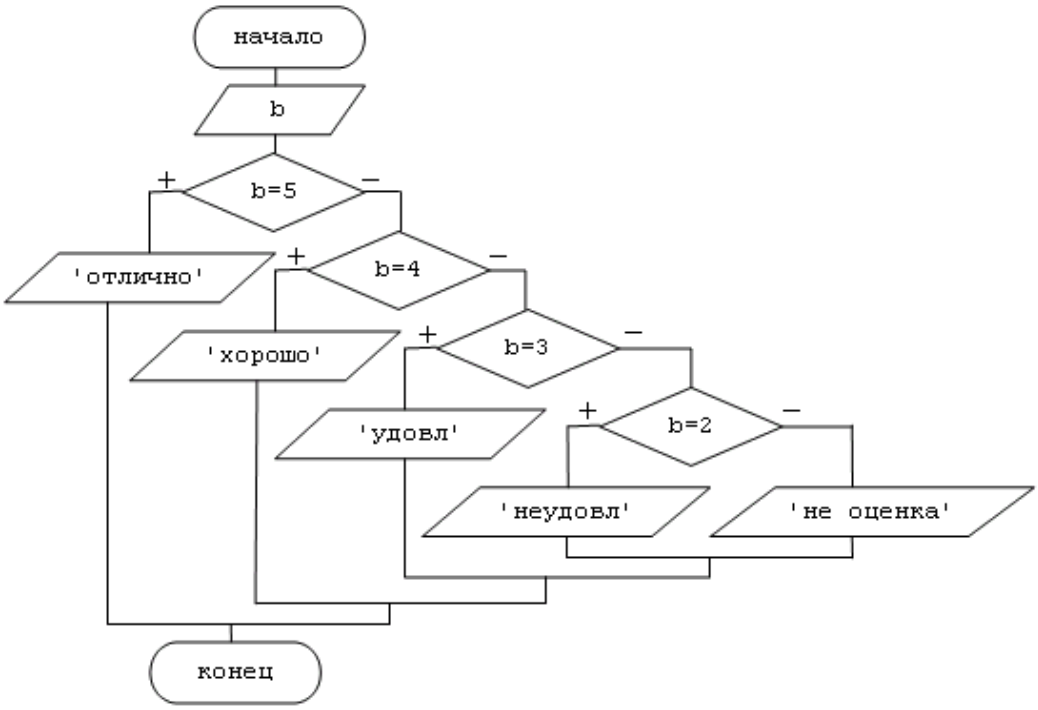
Блок-схема алгоритма решения задачи:




```
Program zadacha1;
    Var A, B: Integer;
     Max: Integer;
    Begin
     Write ('Введите значение A = '); ReadLn (A);
     Write ('Введите значение B = '); ReadLn (B);
     If A>=B then Max:=A else Max:=B;
```

```
   WriteLn ('большее из двух целых чисел A и B: ',Max);
End.
```

ПРИМЕР. Ввести оценку студента в баллах и сообщить ее название.

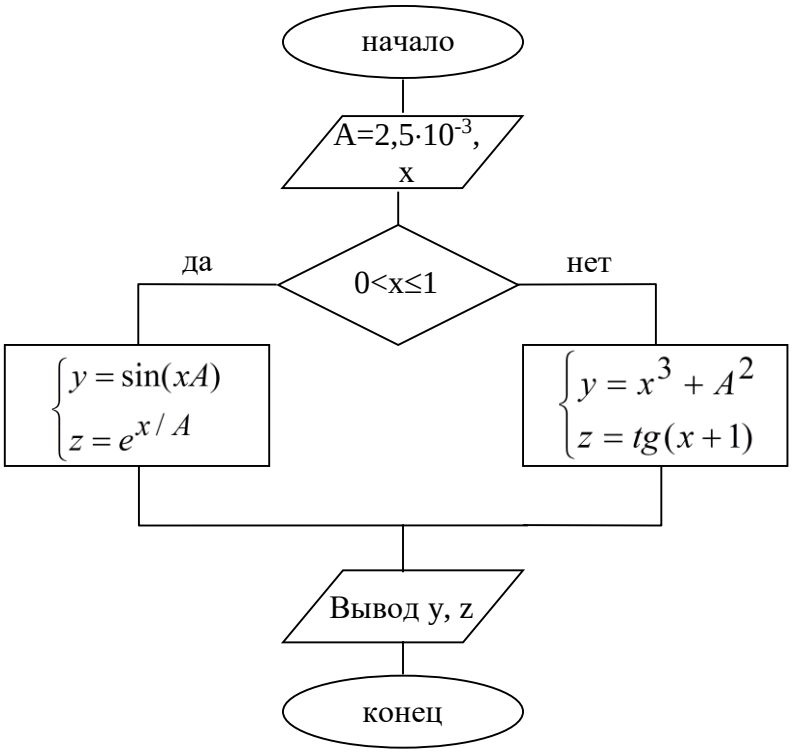



```
Program zadacha2;
Var b:integer;
Begin
   Write ('Введите оценку студента');
   Read(b);
        If b=5 then Write('отлично') else
          If b=4 then Write('хорошо') else
            If b=3 then Write('удовл.') else
              If b=2 then Write('неудовл.') else
              Write('это не оценка');
End.
```

ПРИМЕР.

Вычислить $\begin{cases} y = \sin(xA) \\ z = e^{x/A} \end{cases}$, *если* $0 < x \le 1$ и $\begin{cases} y = x^3 + A^2 \\ z = tg(x+1) \end{cases}$, *если* $x > 1$, $A = 2.5 \cdot 10^{-3}$.

Алгоритм:




```
Program zadacha3;
      Const A=2.5E-3;
      Var x, y, z: real;
   Begin
    Write ('Введите значение x = '); ReadLn (x);
       If (x>0) and (x<=1) then begin
       y:=sin(x*A); z:=exp(x/A); end
       else begin y:=exp(3*ln(x))+sqr(A);
                  z:=sin(x+1)/cos(x+1); end;
       WriteLn ('y=',y,' ','z=',z);
      End.
```

ПРИМЕР. *Использование полной формы условного оператора* `if`.

Дано x – любое вещественное число. Необходимо вычислить значение выражения

$$y = \begin{cases} x-1, & если \quad x < 0, \\ x+1, & если \quad x \geq 0. \end{cases}$$

```
program prog;
var x,y:real;
begin
    writeln('Введите x');
    read(x);
    if x<0 then y:=x-1
           else y:=x+1;
    writeln('y=',y:5:2);
end.
```

ПРИМЕР. *Использование неполной формы условного оператора* if. Дано x – любое вещественное число. Необходимо вычислить значение выражения

$$y = \begin{cases} x-1, & если \quad x < 0, \\ x, & если \quad x = 0, \\ 2 \cdot x, & если \quad x > 0. \end{cases}$$

```
program prog;
var x,y:real;
begin
     writeln('Введите x');
     read(x);
     if x<0 then y:=x-1;
     if x=0 then y:=x;
     if x>0 then y:=2*x;
     writeln('y=',y:5:2)
end.
```

Данную программу можно записать по-другому, с помощью вложенных условных операторов:

```
program prog;
```

```
var x,y:real;
begin
    writeln('Введите x');
    read(x);
    if x<0 then y:=x-1
    else if x=0 then y:=x
    else y:=2*x;
    writeln('y=',y:5:2)
end.
```

ПРИМЕР. Использование составного оператора в условном операторе if. Даны *a* и *b* – любые вещественные числа. Необходимо вычислить значения переменных *s*, *p* , *r*.. Известно, что при *a* > *b* переменные вычисляются по формулам:

1) $s = a + b$;

2) $p = a * b$;

3) $r = a – b$;

иначе (при $a <= b$) по формулам:

1) $s = a^2 + b^2$;

2) $p = a^2 * b^2$;

3) $r = a^2 – b^2$.

```
program prog;
var a,b,s,p,r:real;
        begin
            writeln('Введите a и b');
            read(a,b);
            if a>b then
                    begin
                        s:=a+b;
                        p:=a*b;
                        r:=a-b;
                    end
```

```
                    else
                        begin
                            s:=a*a+b*b;
                            p:=a*a+b*b;
                            r:=a*a-b*b;
                        end;
        writeln('s=',s:5:2, ' p=',p:5:2, ' r=',r:5:2);
end.
```

ПРИМЕР. *Использование логических операций в условном операторе* `if`. Дано x – любое вещественное число. Вычислить значение выражения

$$y = \begin{cases} x^2 \cdot \cos x, & \text{если} \quad x < 0, \\ \sin x^2, & \text{если} \quad 0 \le x \le 2, \\ e^x, & \text{если} \quad x > 2. \end{cases}$$

```
program prog;
var x,y:real;
begin
    writeln('Введите x');
    read(x);
        if x<0 then y:=sqr(x)*cos(x);
        if(x>=0)and(x<=2) then y:=sin(sqr(x));
        if x>2 then y:=exp(x);
    writeln('y=',y:5:2);
end.
```

3.7. Алгоритмы и программы циклической структуры

Очень часто при разработке программ необходимо, чтобы один или более операторов выполнялись два или более раз. Такие алгоритмы называют циклическими.

Определение. Цикл – это последовательность действий, выполняемых многократно, каждый раз при новых значениях параметров. Повторяющиеся операторы - *телом цикла*.

Количество повторений тела цикла может быть известно или нет. Если неизвестно количество повторений тела цикла, завершение его работы происходит по достижению определенного условия. Таким образом, циклы делятся на два типа:

1. Цикл с параметром.

2. Цикл с условием.

В свою очередь, циклы с условием делятся на: а) цикл с предусловием; б) цикл с постусловием.

Оператор цикла с параметром

Цикл с параметром (или с счетчиком) – это такой цикл, в котором тело цикла выполняется определенное количество раз (данный цикл еще называется циклом «Для»).

В цикле с параметром задается переменная, выполняющая роль параметра цикла, ее начальное и конечное значения, приращение (шаг изменения значения параметра цикла).

Блок-схема алгоритма цикла с параметром представлена на рисунке 3.10.

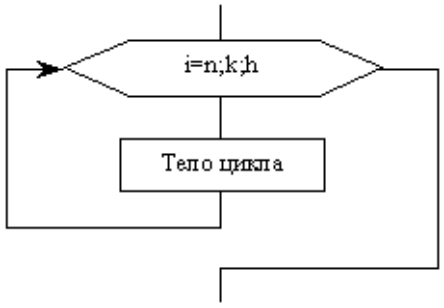


Рисунок 3.10. Блок-схема алгоритма цикла с параметром

Структура программы:

```
for <параметр>:= <нач. знач.> to <конеч. знач.>
                    do <оператор>;
```

или

```
for <параметр>:= <нач. знач.> downto <конеч. знач.>
                    do <оператор>;
```

Первоначальное значение параметра (счетчика) устанавливается равным начальному значению. При каждом «проходе» цикла переменная параметра увеличивается на величину шага. Если счетчик достигнет конечного значения, то цикл завершается, и выполняются следующие за ним операторы.

Если при изменении переменной параметра необходимо использовать переход к следующему значению, то используется оператор *to*; если переход необходимо осуществить к предыдущему значению, то используется оператор *downto*.

Основные особенности цикла *for*:

1) в качестве параметра используется переменная целого (*Integer*, *Byte* и т.д.) или символьного типа (*Char*);

2) шаг изменения счетчика цикла является величиной постоянной и равен 1 или -1. В последнем случае вместо служебного слова **to** используется *downto***.**

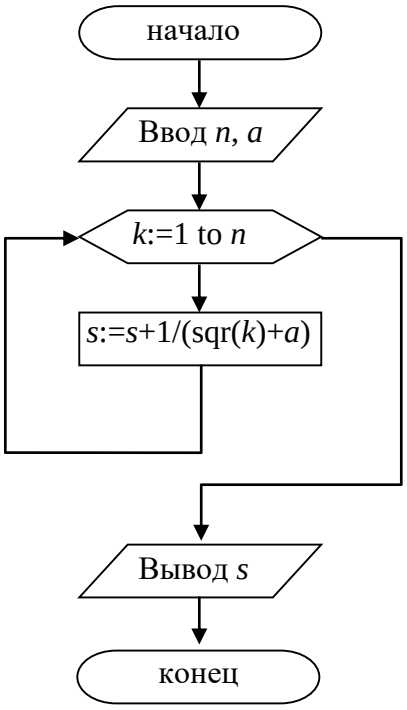
3) если тело цикла состоит из нескольких операторов, то они заключаются в операторные скобки *begin..end*.

Тогда в общем виде цикл записывается так:

```
for I:=1 to n do
            begin
               <оператор 1>;
               <оператор 2>;
               ...
               <оператор n>;
            end;
```

ПРИМЕР. Вычислить сумму $s = \sum_{k=1}^{n} \frac{1}{k^2 + a}$, где a – любое число

Блок-схема



```
program zadacha1;
var k,n:integer; a,s:real;
begin
     write ('a='); readln (a);
     write ('n='); readln (n);
     s:=0;{начальное значение суммы, всегда =0}
     for k:=1 to n do
                    s:=s+1/(sqr(k)+a);
     write ('s=',s);
     readln;
end.
```

ПРИМЕР. Вычислить произведение $y = \prod_{i=1}^{6}\left(i^2 + 0{,}3\right)$.

```
program zadacha2;
var i:integer; y:real;
begin
     y:=1;{начальное значение произведения, всегда =1}
     for i:=1 to 6 do
     y:=y*(sqr(i)+0.3);
     write ('y=',y);
     readln;
end.
```

ПРИМЕР. Дан ряд с общим членом ряда $\frac{1}{1+k}$, вычислить сумму первых *n* членов ряда $s = \sum_{k=1}^{n}\frac{1}{k}$. Результат вывести на экран

```
program prog;
var s:real; k,n:integer;
begin
     writeln ('Введите n');
     read(n);
```

```
    s:=0;
    for k:=1 to n do
                  s:=s+1/k;
    writeln('s=',s:5:2);
end.
```

ПРИМЕР. Дан ряд с общим членом ряда $\frac{1}{1+2\cdot k}$ вычислить произведение первых n членов ряда $p=\prod_{k=1}^{n}\frac{1}{1+2\cdot k}$. Результат вывести на экран

```
    program prog;
    var p:real;k,n:integer;
    begin
        writeln('Введите n');
        read(n);
        p:=1;
        for k:=1 to n do
                     p:=p*1/(1+2*k);
        writeln('p=',p)
    end.
```

ПРИМЕР. Вычислить значение выражения $x=\sum_{k=1}^{5}\frac{1}{k^2}+\prod_{k=1}^{4}\frac{1}{k+1}$. Результат вывести на экран

```
    program prog;
    var s,p,x:real;k:integer;
    begin
        s:=0;
        for k:=1 to 5 do
        s:=s+1/sqr(k);
        p:=1;
        for k:=1 to 4 do
```

```
        p:=p*1/(k+1);
        x:=s+p;
        writeln('x=',x:8:2);
end.
```

Оператор цикла с условием

Циклы с условием применяются для программирования процессов, в которых число повторений оператора неизвестно, а задается некоторое условие выполнения цикла.

Цикл с предусловием

В цикле с предусловием перед выполнением тела цикла осуществляется проверка значения логического выражения или переменной логического типа, если значение этих величин удовлетворяют условию работы цикла, то выполняется тело цикла, а затем вновь проверяется условие и так далее. В противном случае (если условие не соблюдается), выполняется следующий за циклом оператор.

Таким образом, операторы тела цикла с предусловием могут быть не выполнены ни одного раза.

Блок-схема алгоритма цикла с предусловием показана на рисунке 3.11.

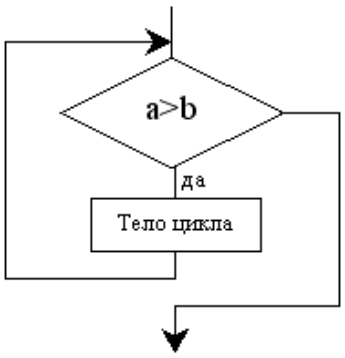


Рисунок 3.11. Блок-схема алгоритма цикла с предусловием

Структура программы

```
while <условие> do begin <тело цикла>; end;
```

Читается так: «Пока условие истинно, выполняется тело цикла».

Операторные скобки *begin – end* в данном цикле ставятся обязательно!

ПРИМЕР. Дано натуральное число *m*. Подсчитать количество цифр в этом числе.

Решение. Подсчет количество цифр начнем с последней цифры числа. Увеличим счетчик цифр на единицу. Число уменьшим в 10 раз (тем самым мы избавимся от

последней цифры числа). Далее с получившимся числом проделаем ту же последовательность действий, и будем повторять ее до тех пор, пока число не станет равным нулю.

```
program zadacha3;
uses crt;
var n,m:longint; k:integer; {m – число, полученное в результате деления на 10}
begin;
     write ('Введите целое число n='); readln(n);
     m:=n; k:=0;
     while m<>0 do
            begin
               k:=k+1; {или Inc(k)}
               m:=m div 10; {уменьшение числа на последнюю цифру}
             end;
     writeln ('в числе',n,'-',k,'цифр');
     readln;
end.
```

Рассмотрим данную программу пошагово.

n	k	m
75487398	0	75487398
75487398	1	7548739
75487398	2	754873
75487398	3	75487
75487398	4	7548
75487398	5	754
75487398	6	75
75487398	7	7
75487398	8	0

ПРИМЕР. Таблица значений функции. Составить таблицу значений функции $y = \cos x$, для аргумента, изменяющегося на отрезке $x \in [a,b]$ с постоянным шагом h, где a, b, h – вещественные числа, $a < b$, $h = \dfrac{|b-a|}{2}$

```
program prog;
var x,y,a,b,h:real;
begin
  writeln('Введите a,b,h');
  read(a,b,h);
  x:=a;//переменной цикла присваивается начальное значение
  while x<=b do//проверяется условие выхода из цикла
         begin
            y:=cos(x);
            writeln('x=',x:5:2,' y=',y:5:2);
            x:=x+h;{значение переменной цикла увеличивается на
  величину шага}
         end;
end.
```

Оператор цикла с постусловием

Для программной реализации с неизвестным числом повторений существует еще один оператор – оператор цикла с постусловием. Цикл с постусловием предназначен для организации циклических алгоритмов, в которых проверка условия работы цикла проводится после выполнения последовательности операторов тела цикла, т.е. тело цикло выполняется до тех пор, пока не будет выполнено условие выхода. По этой причине, *операторы тела цикла всегда будут выполнены хотя бы один раз***.**

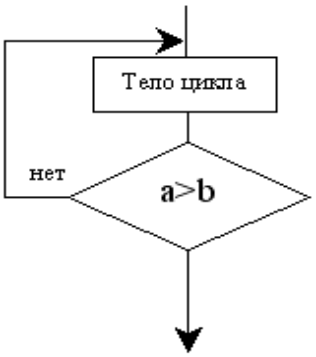


Рисунок 3.12. Блок-схема алгоритма цикла с постусловием

Структура программы:

```
Repeat <оператор 1>; <оператор 2>; …;<оператор n>;
Until <условие>;
```

Читается так: "Выполнять оператор 1, оператор 2, …, оператор n до тех пор, пока не выполнится условие выхода". Данный оператор предполагает наличие в теле цикла нескольких операторов, поэтому операторные скобки *begin – end* не ставятся.

ПРИМЕР. Написать программу, которая выводит таблицу значений функции $y = 2{,}4x^2 + 5x - 3,\ x \in -[-2;2],\ h = 0{,}5$

```
program z5;
uses crt;
var x,y:real;
begin;
     x:=-2;
     repeat
         y:=-2.4*sqr(x)+5*x-3;
         writeln ('x=',x,' ','y=',y);
         x:=x+0.5;
     until x>2;
     readln;
end.
```

ПРИМЕР. Найти сумму квадратов всех натуральных чисел от 1 до 100.

Решить эту задачу с использованием всех трех видов циклов.

Цикл "while"	**Цикл "repeat"**	**Цикл "for"**

Program Ex1;
Var A:integer;
S:longint;
Begin
A:=1; S:=0;
While A<=100 do
Begin
S:=S+A*A;
A:=A+1;
End;
Writeln('S=',S);
End.
	Program Ex2;
	Var A:integer;
	S:longint;
	Begin
	A:=1; S:=0;
	Repeat
	S:=S+A*A;
	A:=A+1;
	Until A>100;
	Writeln('S=',S);
	End.
		Program Ex3;
		Var A:integer;
		S:longint;
		Begin
		S:=0;
		For A:=1 to 100
		do S:=S+A*A;
		Writeln('S=',S);
		End.

3.8. Одномерный массив

Определение. Массив – это пронумерованная последовательность величин одинакового типа, обозначаемая одним именем. Элементы массива располагаются в последовательных ячейках памяти, обозначаются именем массива и индексом. Каждое из значений, составляющих массив, называется его *компонентой* (или *элементом* массива).

Массив данных в программе рассматривается как переменная структурированного типа. Массиву присваивается имя, посредством которого можно ссылаться как на массив данных в целом, так и на любую из его компонент.

Переменные, представляющие компоненты массивов, называются переменными с индексами в отличие от простых переменных, представляющих в программе элементарные данные. Индекс в обозначении компонент массивов может быть константой, переменной или выражением порядкового типа.

Количество индексов элементов массива определяет *размерность* массива. По этом признаку массивы делятся на одномерные (линейные), двумерные, трёхмерные и т.д.

Числовой массив называется одномерным, если его можно записать в строку, то есть он имеет одно измерение – *длину***,** то есть за каждым элементом массива закреплен только один его порядковый номер, и такой массив называется *линейным*.

Очень часто одномерный массив называют *вектором*.

ПРИМЕР. Числовая последовательность четных натуральных чисел 2, 4, 6, ..., N представляет собой линейный массив, элементы которого можно обозначить:

$A[1]=2,\ A[2]=4,\ A[3]=6,\ ...,\ A[k]=2*(k+1)$,

где k – номер элемента, а 2, 4, 6, ..., N – значения.

Индекс (порядковый номер элемента) записывается в квадратных скобках после имени массива.

Например, $A[7]$ – седьмой элемент массива A; $D[6]$ – шестой элемент массива D.

Для размещения массива в памяти ЭВМ отводится поле памяти, размер которого определяется типом, длиной и количеством компонент массива.

Длиной массива называется число его элементов.

Тип компонент называется *базовым типом*.

В языке Pascal эта информация задается в разделе описаний. Массив можно задать или описать следующими способами:

1. Описание массива в разделе описания переменной

```
имя массива: Array [нач. знач. индекса..кон. знач. индекса] Of базовый тип;
```

Например,

```
Var B: Array [1..5] Of Real, R: Array [1..34] Of Char;
```

– описывается массив *B*, состоящий из 5 элементов и символьный массив *R*, состоящий из 34 элементов. Для массива *B* будет выделено 5*6=30 байт памяти, для массива *R* – 1*34=34 байта памяти.

2. Описание массива через создание нового типа

```
Type Mas = Array[1..100] Of Integer;
```

А затем в разделе переменной описываем заданные массивы через этот тип. В этом случае компьютер задействует меньше памяти, чем в первом.

```
Var a, b: Mas;
```

Этот случай часто применяется для дальнейшего ввода процедуры).

В качестве типа элементов массива можно использовать все типы, известные нам на данный момент (к ним относятся все числовые, символьный, строковый и логический типы).

Нумеровать элементы массивов можно не только от единицы, но и от любого целого числа. Вообще для индексов массивов подходит любой порядковый тип, то есть такой, который в памяти машины представляется целым числом. Единственное ограничение состоит в том, что размер массива не должен превышать 64 Кб.

Рассмотрим некоторые примеры объявления массивов.

```
var a: array [0..1000] of integer;
    b: array [1..10] of string;
    c: array[char] of integer;
    f: array['a'..'z'] of integer; (массив состоит из букв)
```

3. Описание массива константой

Для того чтобы не вводить массивы вручную во время отладки программы (особенно если они имеют большую размерность), можно пользоваться не только файлами**.** Существует и более простой способ, когда входные данные задаются прямо в тексте программы при помощи типизированных констант.

Если массив линейный (вектор), то начальные значения для компонент этого вектора задаются через запятую, а сам вектор заключается в круглые скобки.

Исключение составляют только массивы, компонентами которых являются величины типа *char*. Такие массивы можно задавать проще: строкой символов.

Примеры задания массивов типизированными константами:

```
const a: array[1..5] of byte = (3,1,6,2,0); {линейный}
s: array[0..9] of char = '0123456789'; (символьный тип данных)
```

После того как массив описан, необходимо его заполнить. Заполнить массив можно следующим образом:

1) *ввод элементов массива с клавиатуры* – используется обычно тогда, когда между элементами не наблюдается никакой зависимости. Например, последовательность чисел 1, 2, -5, 6, -111, 0 может быть введена в память следующим образом:

2) *заполнение массива с помощью генератора случайных чисел***.** Этот способ удобен, когда в массиве много элементов. Функция ***Random*(*k*),** где *k* – натуральное число, "выбирает" случайным образом целое число из промежутка от 0 до *k*-1. Каждое обращение к этой функции дает в результате произвольное число из указанного промежутка. Однако можно заметить, что от выполнения к выполнению программы генерируемая последовательность "случайных" чисел **будет одной и той же!** Чтобы этого избежать, нужно, например, в начале программы разместить процедуру ***Randomize;***, которая "заставляет" компьютер "менять" последовательность генерируемых чисел от запуска к запуску программы.

Конструкция ***Random*(20)** будет генерировать целые числа из промежутка от 0 до 19.

Приведем конструкцию, которая будет генерировать:

а) целые числа от 10 до 50

Так как ***Random*(50)** генерирует числа от 0, то левый конец отрезка нужно "переместить" вправо: **10+*Random*(50)**. Однако в этом случае правый конец отрезка будет находится в точке **59: 10 + 49** (наибольшее число, которое генерируется функцией ***Random*(50)**). Следовательно, нужно перенести правый край отрезка влево на **9** единиц. Таким образом, ответ будет следующим: **10+*Random*(41)**.

б) целые числа от -5 до 5

Ответ будет таким: **-5+*Random*(11)**. Пояснения аналогичны предыдущим.

в) вещественные числа от 0 до 1

Для этого примера один из вариантов ответа такой: ***Random*(101)/100.**

ЗАДАЧА 1. Заполнить массив с клавиатуры.

```
program vvod;
var a: array[1..n] of real;
    i: integer;
begin
    write ('n'); read (n);
    for i:=1 to n do
    begin
    write('a[',i,']=');
    readln(a[i]);
    end;
end.
```

ЗАДАЧА 2. Заполнить одномерный массив с помощью датчика случайных чисел и вывести его на экран.

```
Program Pr_2;
Type Mas = Array[1..n] of integer;
Var a: Mas; i, n: integer ;
Begin
    Write('n'); ReadLn(n);
    Randomize; {включение генератора случайных чисел}
```

```
    For i:= 1 to n do
    A[1]:= -25 + random(54);
    Writeln('massiv');
    For i:= 1 to n do
    Writeln('a[',i,']=',a[i]); {вывод массива}
End.
```

В этом случае `random` выбирает случайное число из отрезка от -25 (54 числа).

Еще один пример:

```
Program Massiv;
Var
      i: Byte;
      A: Array[1..5] Of Integer;
Begin
    Randomize;
    For i:=1 To 5 Do
        A[i]:=Random(11);
    For i:=1 To 5 do
        Write(A[i], ' ');
    WriteLn;
End.
```

Инициализация генератора случайных чисел.

Генератор случайных целых чисел от 0 до 10.

Так же задать массив можно с помощью оператора присваивания. Это самый простой способ, каждому элементу массива определенного значения:

```
begin
    A[1]:=1;
    A[2]:=2;
    A[3]:=3;
```

Этот способ заполнения элементов массива особенно удобен, когда между элементами существует какая-либо зависимость, например, арифметическая или геометрическая прогрессии.

ЗАДАЧА 3. Заполнить одномерный массив элементами, отвечающими следующему соотношению:

$a_1=1$; $a_2=1$; $a_i=a_{i-2}+a_{i-1}$ (i = 3, 4, ..., n).

```
Read(n); {Ввод количества элементов}
a[1]:= 1;
a[2]:= 1;
```

```
for i:= 3 to n do
     a[i]:= a[i-1] + a[i-2];
```

Получим в результате массив: 1, 1, 2, 3, 5, 8, 13 …

Над элементами массивами чаще всего выполняются такие действия, как

а) поиск значений;

б) сортировка элементов в порядке возрастания или убывания;

в) подсчет элементов в массиве, удовлетворяющих заданному условию;

г) нахождение суммы элементов массива;

д) создание нового массива на основании старого с некоторыми условиями.

ЗАДАЧА 4. Нахождение суммы элементов массива

```
program summa;
uses crt;
var a: array[1..10] of integer;
    i, S: integer;
begin
  for i:=1 to 10 do
  begin
    write('a[',i,']=');
    readln(a[i]);
  end;
  S:=0;
  for i:=1 to 10 do
  S:=S+a[i];
  writeln('Summa=',S);
end.
```

ЗАДАЧА 5. Нахождение элементов с заданными свойством.

Найти номера четных элементов. (Для нахождения необходимо просмотреть весь массив, и если просматриваемый элемент является четным, то выводится его номер).

```
program pr_5;
uses crt;
var a: array[1..10] of integer;
```

```
    i: integer;
begin
  for i:=1 to 10 do
  begin
    write('a[',i,']=');
    readln(a[i]);
  end;
  for i:=1 to 10 do
  if a[i] mod 2=0 then
  write (i);
end.
```

ЗАДАЧА 6. Найти количество положительных и отрицательных элементов в массиве.

```
program pr_6;
uses crt;
var a: array[1..10] of integer;
    i, k1, k2: integer;
begin
clrscr;
  for i:=1 to 10 do
  begin
    write('a[',i,']=');
    readln(a[i]);
  end;
  k1:=0; k2:=0;
  for i:=1 to 10 do
  if a[i]>0 then inc(k1)
  else if a[i]<0 then inc(k2);
  writeln('Количество положительных элементов =',k1);
  writeln('Количество отрицательных элементов =',k2);
end.
```

ЗАДАЧА 7. Нахождение максимального элемента массива и его номера.

```
program Maximum;
uses crt;
Type mass= array[1..15] of integer;
var b: mass;
    i, max,imax : integer;
begin
  Randomize;
  for i:=1 to 15 do
  b[i]:=-15+random(25);
  for i:=1 to 15 do
  write (b[i],' ');
  writeln;
  max:=b[1];
  for i:=2 to 15 do
    if b[i]>max then
    begin
    max:=b[i]; imax:=i;
    end;
  writeln('Максимум =',max, 'его номер -',imax);
end.
```

ЗАДАЧА 8. Дан одномерный массив $a(n)$, состоящий из вещественных чисел (n – размерность массива, n =10). Ввести значения элементов массива $a(n)$ с клавиатуры. Вывести массив $a(n)$ на экран. Составить программу для нахождения:

1) суммы элементов массива;

2) произведения элементов массива;

3) количества отрицательных элементов массива;

4) максимального элемента массива.

```
program prog;
uses crt;
var a:array[1..10] of real;
```

```
s,p,max:real; k,n:integer;
begin
//Ввод элементов одномерного массива с клавиатуры
for k:=1 to 10 do
begin
     write('Введите a[',k:2,']=');
     read(a[k]);
end;
//Вывод элементов одномерного массива на экран
writeln('Массив a:');
for k:=1 to 10 do
              write(a[k]:5:2,' ');
writeln;
//Сумма элементов одномерного массива
s:=0;
for k:=1 to 10 do
          s:=s+a[k];
writeln('s=',s);
//Произведение элементов одномерного массива
p:=1;
for k:=1 to 10 do
              p:=p*a[k];
writeln('p=',p);
//Количество отрицательных элементов одномерного массива
n:=0;
for k:=1 to 10 do
          if a[k]<0 then n:=n+1;
writeln('n=',n);
//Максимальный элемент одномерного массива
max:=a[1];
```

```
for k:=1 to 10 do
              if a[k]>max then
                              max:=a[k];
writeln('max=',max);
end.
```

Сортировка и поиск

В прикладных программах широко распространены два типа операций, связанных с массивами:

1. Упорядочивание элементов массива по возрастанию или убыванию (сортировка)
2. Поиск элемента в массиве.

Рассмотрим простейший вариант сортировки массива (сортировка выбором). Пусть есть массив из *n* элементов; сначала найдём в нём самый маленький среди элементов с номерами 2,3,...*n* и поменяем местами с первым элементом, затем среди элементов с номерами 3,4,...*n* найдём наименьший и обменяем со вторым, и т. д. В результате наш массив окажется отсортированным по возрастанию.

```
program SelectSort;
const n = 10;
var a: array [1..n] of integer;
    i,j,jmin,buf: integer;
    {jmin – номер наименьшего элемента,
     buf используется при обмене значений двух элементов}
begin
  for i:=1 to 10 do begin
    write('Введите элемент номер ',i,' -> ');
    readln(a[i]);
  end;

  for i:=1 to n-1 do begin
    jmin:=i;
    for j:=i+1 to n do
      if a[j]<jmin then jmin:=j;
```

```
        buf:=a[i];
        a[i]:=a[jmin];
        a[jmin]:=buf;
    end;

    write('Результат: ');
    for i:=1 to 10 do write(a[i],' ');
    readln;
end.
```

Другой способ – пузырьковая сортировка, он работает чуть быстрее, чем предыдущий. На первом этапе двигаемся от *n*-го элемента до 2-го и для каждого из них проверяем, не меньше ли он предыдущего; если меньше, то меняем местами текущий и предыдущий. В итоге первый элемент будет наименьшим в массиве. На втором этапе также проходим элементы от *n*-го до 3-го, на третьем – от *n*-го до 4-го, и т. д. В итоге массив будет отсортирован по возрастанию.

```
program zadacha;
...
var i,j: integer;
    buf: integer;
begin
  ...
  for i:=2 to n do
    for j:=n downto i do
      if a[j]<a[j-1] then begin
        buf:=a[j];
        a[j]:=a[j-1];
        a[j-1]:=buf; end;  end.
```

ЗАДАЧА 8. Дан линейный массив. Упорядочить его элементы в порядке возрастания.

Сортировка массива выбором (в порядке возрастания). Идея решения: пусть часть массива (по *k*-й элемент включительно) отсортирована. Нужно найти в неотсортированной части массива минимальный элемент и поменять местами с (*k*+1)-м

```
Program Sortirovka;
Var n, i, j, k, Pr : integer; A : Array [1..30] of integer;
Begin
  Write('Введите количество элементов: ');
  Readln(n);
For i := 1 to n do
                    Begin
                          Write('Введите A[', i, '] ');
                          Readln(A[i]);
                    End;
WriteLn;
For i := 1 to n - 1 do
                          Begin
                          k := i;
                          For j := i + 1 to n do
                          if A[j] <= A[k] then k := j;
                          Pr := A[i]; A[i] := A[k]; A[k]:= Pr;
                          End;
  For i := 1 to n do Write(A[i], ' ');
End.
```

Тест: *N* = 10; элементы массива - 1, 2, 2, 2, -1, 1, 0, 34, 3, 3.

Ответ: -1, -1, 0, 1, 2, 2, 2, 3, 3, 34.