Министерство сельского хозяйства Российской Федерации
Департамент образования, научно-технологической политики и рыбохозяйственного комплекса
Новочеркасский инженерно-мелиоративный институт им. А.К. Кортунова
ФГБОУ ВО Донской ГАУ

Г.А. Полубедова

Введение в информационные технологии

Практикум

для студентов очной формы обучения по направлениям «Экономика», «Менеджмент», «Педагогическое образование»

Новочеркасск
2022

УДК 004

П 53

Рассмотрено на заседании кафедры менеджмента и информатики (№ 7 от 24.05.22 г.) и рекомендовано к изданию.

Рецензент: **Иванов П.В.**, д-р. техн. наук, профессор кафедры менеджмента и информатики НИМИ ФГБОУ ВО Донской ГАУ.

Полубедова, Г.А.

П 53 Введение в информационные технологии: практикум для студ. оч. формы обуч. по направл. «Экономика», «Менеджмент», «Педагогическое образование» / Г.А. Полубедова ; Новочерк. инж.-мелиор. ин-т Донской ГАУ. - Новочеркасск, 2022. – 65 с.

Практикум содержит основные вопросы по алгоритмизации и программированию на алгоритмическом языке Паскаль.

Предназначен для изучения дисциплины «Введение в информационные технологии» студентами, обучающимися по направлениям подготовки: «Экономика», «Менеджмент», «Педагогическое образование», а также может быть полезен для студентов других специальностей.

Ключевые слова: Алгоритм, способы описания алгоритмов, основные объекты языка паскаль, операторы языка паскаль, программирование алгоритмов линейной, разветвляющейся и циклической структуры, одномерные массивы, комбинированные данные.

ОГЛАВЛЕНИЕ

с.

ВВЕДЕНИЕ

Как показывает многолетний опыт работы научить студентов использовать готовые программы, проще, чем научить их разрабатывать свои алгоритмы решения задач, составлять и отлаживать программы. Однако, общие тенденции развития программирования указывают на то, что идти по простому пути в данном вопросе не самое лучшее решение. В процессе общения с компьютером умение программировать позволяет выйти на высокий уровень развития интеллектуальных качеств. Изучая программирование, студенты лучше понимают сущность работы компьютеров, их возможности и границы их применения. Программирование помогает на собственном опыте пройти все основные этапы формализованного решения некоей творческой, точно сформулированной задачи. Написав однажды свои собственные пусть даже простые, но работающие программы, студенты понимают, что компьютеры совсем не всемогущие машины, а является всего лишь инструментом, которым управляют люди.

В данном практикуме описывается ядро языка *Паскаль* — минимальный набор средств, достаточный для написания сравнительно простых программ. В частности, рассматриваются все операторы языка, наиболее популярные типы данных и операции над ними.

Материал практикума разбит на темы, в соответствии с рабочими программами по данной дисциплине для указанных направлений подготовки, где рассмотрены основные понятия, приведены примеры и задачи для самостоятельного выполнения студентами. Решение данных задач поможет приобрести и закрепить навыки практической работы на языке *Паскаль*.

Данный практикум позволяет студентам самостоятельно изучать представленный материал, а так же углубить и расширить полученные знания на практических занятиях.

Практикум предназначен для студентов высших учебных заведений изучающих дисциплину «Введение в информационные технологии».

ТЕМА 1. ОСНОВЫ АЛГОРИТМИЗАЦИИ

1.1 Алгоритм и его свойства

Каждый из нас постоянно встречается с множеством задач — от самых простых и хорошо известных до очень сложных. Для многих задач существуют определённые правила (инструкции, предписания), объясняющие исполнителю, как решать данную задачу. Эти правила человек может изучить заранее или сформулировать сам в процессе решения задачи. Такие правила принято называть алгоритмами.

Алгоритм— это процедура, которая позволяет путём выполнения последовательности элементарных шагов получить однозначный результат или за конечное число шагов прийти к выводу о том, что решения не существует. Это точное предписание, определяющее вычислительный процесс, ведущий от варьируемых начальных данных к исходному результату, — понятное и точное предписание (указание) исполнителю совершить определённую последовательность действий, направленных на достижение указанной цели или решение поставленной задачи.

В качестве примера можно привести алгоритм прыжка в длину с разбега: разбег — отталкивание — полёт — приземление. Из примера видно, что последовательность этапов прыжка в длину поменять местами невозможно, так как она разбита на элементарные фазы. В обычной жизни с алгоритмом решения какой-либо задачи мы можем столкнуться, например, при изготовлении различных кулинарных изделий и т.д.

Слово «алгоритм» происходит от «algorithmi» − латинской формы написания имени великого математика IX в. аль-Хорезми, который сформулировал правила выполнения арифметических действий. Первоначально под алгоритмами и понимали только правила выполнения четырёх арифметических действий над многозначными числами. В дальнейшем это понятие стали использовать вообще для обозначения последовательности действий, приводящих к решению поставленной задачи.

Любой алгоритм обладает рядом свойств:

1. Дискретность алгоритма. Процесс решения задачи, определяемый алгоритмом, расчленён на отдельные элементарные действия, и, соответственно, алгоритм представляет последовательность указаний, команд, определяющих порядок выполнения шагов процесса.

2. Определённость алгоритма. Каждая команда алгоритма должна быть понятна исполнителю, не оставлять места для её неоднозначного толкования и неопределённого исполнения. Описание алгоритма должно быть таким, чтобы его мог выполнить любой грамотный пользователь.

3. Результативность алгоритма. Выполнение алгоритма должно приводить к получению определённого результата после конечного числа шагов.

4. Массовость алгоритма. Каждый алгоритм, разработанный для решения некоторой задачи, должен быть применим для решения задач этого типа при всех допустимых значениях исходных данных.

Выполняя алгоритм, исполнитель может не вникать в смысл того, что он делает, и вместе с тем получать нужный результат. В таком случае говорят, что исполнитель действует формально, т.е. отвлекается от содержания поставленной задачи и только строго выполняет некоторые правила, инструкции. Это очень важная особенность алгоритмов. Наличие алгоритма формализовало процесс, исключило рассуждения. Если обратиться к примерам других алгоритмов, то можно увидеть, что и они позволяют исполнителю действовать формально. Таким образом, создание алгоритма даёт возможность решать задачу формально, механически исполняя команды алгоритма в указанной последовательности. Построение алгоритма для решения задачи из какой-либо области требует от человека глубоких знаний в этой области, бывает связано с тщательным анализом поставленной задачи, сложными, иногда очень громоздкими рассуждениями. На поиски алгоритма решения некоторых задач учёные затрачивают многие годы. Но когда алгоритм создан, решение задачи по готовому алгоритму уже не требует каких-либо рассуждений и сводится только к строгому выполнению команд алгоритма. В этом случае исполнение алгоритма можно поручить не человеку, а машине. Действительно, простейшие операции, на которые при создании алгоритма расчленяется процесс решения задачи, может реализовать и машина, специально созданная для выполнения отдельных команд алгоритма и выполняющая их в последовательности, указанной в алгоритме. Это положение и лежит в основе работы автоматических устройств, автоматизации деятельности человека.

На примере квадратного уравнения рассмотрим процесс создания алгоритма.

Пусть есть квадратное уравнения:

$$\mathbf{ax^2 + bx + c = 0}$$

1. Вычислим значение дискриминанта:

$$\mathbf{D = b^2 - 4ac}$$

2. Если значение дискриминанта больше или равно нулю, то вычисляем корни уравнения:

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}.$$

3. Если дискриминант меньше нуля, то уравнение действительных корней не имеет.

Каждое указание алгоритма предписывает исполнителю выполнить одно конкретное законченное действие. Исполнитель не может перейти к выполнению следующей операции, не закончив полностью выполнения предыдущей. Предписания алгоритма надо выполнять последовательно, одно за другим, в соответствии с указанным порядком их записи. Выполнение всех предписаний гарантирует правильное решение задачи. Данный алгоритм

будет понятен исполнителю, умеющему работать с циркулем и знающему, что такое поставить ножку циркуля, провести окружность и т.д.

Анализ примеров различных алгоритмов показывает, что запись алгоритма распадается на отдельные указания исполнителю выполнить некоторое законченное действие. Каждое такое указание называется командой. Команды алгоритма выполняются одна за другой. После каждого шага исполнения алгоритма точно известно, какая команда должна выполняться следующей. Поочередное выполнение команд алгоритма законченное число шагов приводит к решению задачи, к достижению цели.

1.2Способы описания алгоритмов

Для описания алгоритмов используются следующие способы:

- словесно-формульное описание алгоритма, т.е. описание алгоритма с помощью слов и формул;
- графическое описание алгоритма с использованием ***блок-схем***. В этом случае схема алгоритма представляет собой систему связанных геометрических фигур. Каждая фигура обозначает один этап процесса решения задачи и называется ***блоком***. Порядок выполнения этапов указывается стрелками, соединяющими блоки;
- описание алгоритма на алгоритмическом языке. ***Алгоритмический язык*** **—** это средство для записи алгоритмов в аналитическом виде, причём используется ограниченный набор терминов, более строгие правила записи операций и т.д. с целью обеспечения однозначности понимания алгоритма.

Если при описании алгоритма необходимы пояснения (преобразования и вывод формул), то рационально использовать описательный алгоритм.

Графический алгоритм целесообразно приводить при укрупнённом описании сложных программ, содержащих условные переходы, циклы (многократно выполняемые действия) и подпрограммы. Современные языки программирования позволяют получить запись алгоритма в виде простого логичного и наглядного текста, снабжённого соответствующими комментариями.

Рассмотрим более подробно второй тип описания алгоритмов **—** ***блок-схемы***. В схеме – блоки стараются размещать сверху вниз, в порядке их выполнения. Операции разного вида изображаются в схеме различными геометрическими фигурами, как показано в таблице 1.

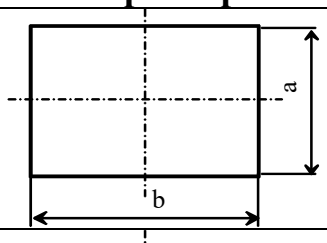
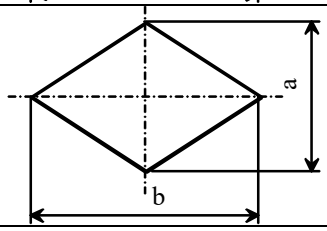
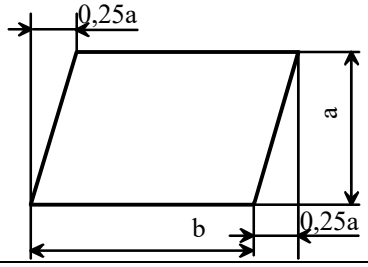
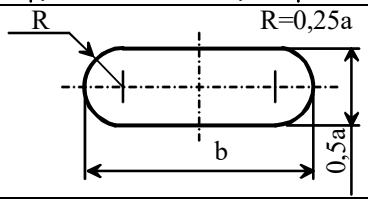
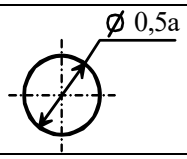
При построении блок-схем должны выполняться следующие правила:

- линии, соединяющие различные блоки (линии потока), показывающие направление движения информации, могут быть либо вертикальными, либо горизонтальными;
- направления линии потока сверху вниз и слева направо принимаются за основные и, если линии потоков не имеют изломов, *стрелками их можно не обозначать*. В остальных случаях направление линии

потока обозначать стрелкой обязательно;

- расстояние между блоками должно составлять не менее 5 мм, расстояние между линиями потока — не менее 3 мм.

Таблица 1– Перечень основных блоков, используемых при составлении блок-схем и отображаемые ими функции

Название блока	**Обозначение и размеры**	**Функция**
Процесс		Выполнение операции или группы операций, в результате которых изменяется значение, форма представления или расположения данных
Решение		Выбор направления выполнения алгоритма или программы в зависимости от некоторых переменных условий
Ввод-вывод		Преобразование данных в форму, пригодную для обработки (ввод) или отображение результатов обработки (вывод)
Начало-конец		Начало, конец, прерывание процесса обработки данных или выполнения программы
Соединитель		Указание связи между прерванными линиями потока

• Описание алгоритма на языке программирования.

Выделяют следующие виды алгоритмов:

− линейный;

− разветвляющийся;

− циклический.

Линейным называется алгоритм, в котором все этапы решения задачи выполняются строго последовательно.

Блок-схема нахождения периметра прямоугольного треугольника при известных длинах его катетов имеет следующий вид (рисунок1).

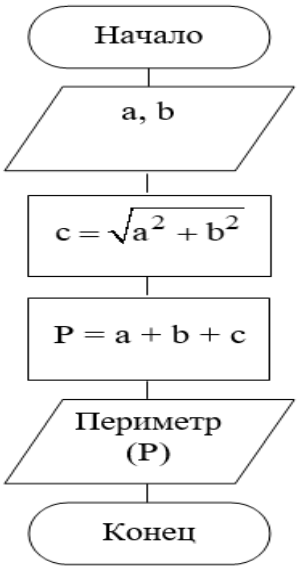


Рисунок 1 − Блок-схема линейного алгоритма

Разветвляющийся алгоритм — это такой алгоритм, в котором выбирается один из нескольких возможных путей вычислительного процесса. Каждый подобный путь называется ветвью алгоритма. Признаком разветвляющегося алгоритма является наличие условия.

Блок-схема алгоритма решения квадратного уравнения выглядит следующим образом (рисунок 2).

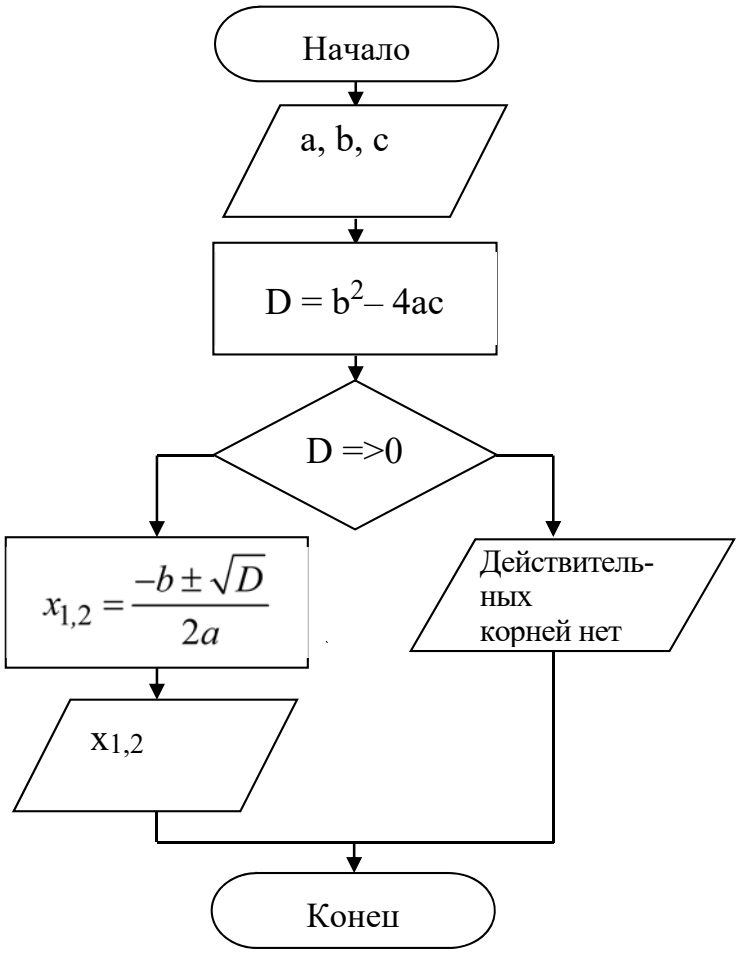


Рисунок 2 − Блок-схема разветвляющегося алгоритма

Циклическим называют такой алгоритм, в котором получение результата обеспечивается многократным выполнением одних и тех же операций.

Задача. Построить блок-схему возведения числа *а*в степень *n* (рисунок 3).

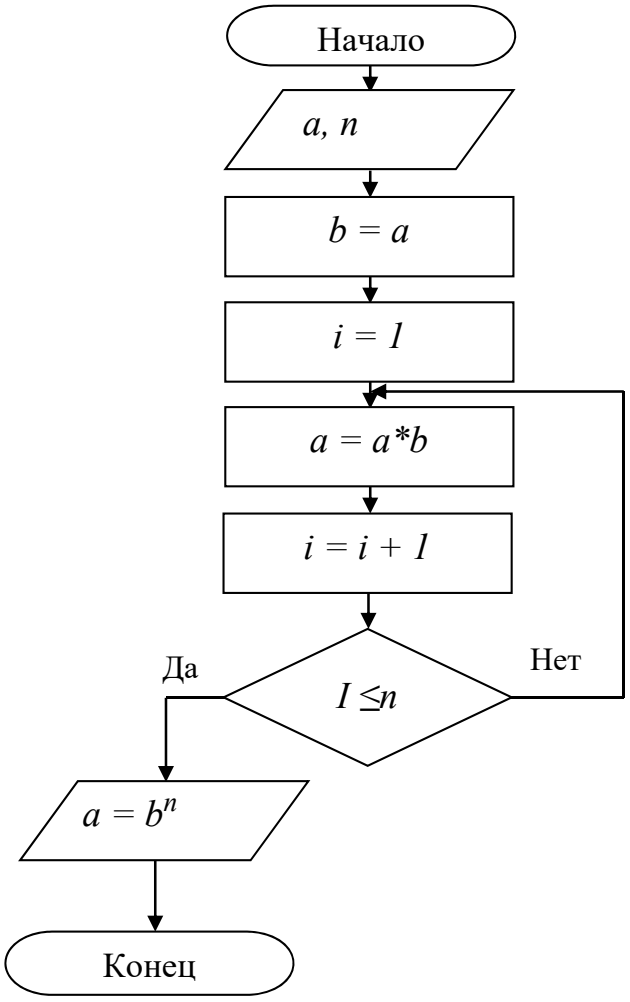


Рисунок 3 − Блок-схема циклического алгоритма

1.3Языки программирования. Виды программирований

Прогресс компьютерных технологий определил процесс появления новых разнообразных знаковых систем для записи алгоритмов — языков программирования. Смысл появления такого языка — оснащённый набор вычислительных формул дополнительной информации, что превращает данный набор в алгоритм.

Языки программирования — это искусственно созданные языки. От естественных они отличаются ограниченным числом «слов» и очень строгими правилами записи команд (операторов). Совокупность подобных требований образует синтаксис языка программирования, а смысл каждой команды и других конструкций языка − его семантику.

Языки программирования — это формальные языки общения человека с ЭВМ, предназначенные для описания совокупности инструкций, выполнение которых обеспечивает правильное решение требуемой задачи. Их основная роль заключается в планировании действий по обработке информации. Любой язык программирования основан на системе понятий, и уже с её помощью человек может выражать свои соображения.

Связь между языком, на котором мы думаем/программируем, и задачами и решениями, которые мы можем представлять в своём воображении, очень близка. По этой причине ограничивать свойства языка только целями исключения ошибок программиста в лучшем случае опасно. Как и в случае с естественными языками, есть огромная польза, быть по крайней мере двуязычным. Язык предоставляет программисту набор концептуальных инструментов, если они не отвечают задаче, то их просто игнорируют. Например, серьёзные ограничения концепции указателя заставляют программиста применять вектора и целую арифметику, чтобы реализовать структуры, указатели и т.п. Хорошее проектирование и отсутствие ошибок не может гарантироваться за счёт чисто языковых средств.

Может показаться удивительным, но конкретный компьютер способен работать с программами, написанными на его родном машинном языке. Существует почти столько же разных машинных языков, сколько и компьютеров, но все они суть разновидности одной идеи — простые операции производятся со скоростью молнии на двоичных числах.

Машинно-зависимые языки программирования

Машинно-зависимые языки — это языки, наборы операторов и изобразительные средства которых существенно зависят от особенностей ЭВМ (внутреннего языка, структуры памяти и т.д.). Эти языки называются языками программирования низкого уровня. Они ориентированы на конкретный тип процессора и учитывают его особенности. Операторы такого языка близки к машинному коду и ориентированы на конкретные команды процессора, то есть данный язык является машинно-зависимым. Языком низкого уровня является язык Ассемблер. С его помощью создаются очень эффективные и компактные программы, так как разработчик получает доступ ко всем возможностям процессора. Подобные языки применяются для написания небольших системных приложений, драйверов устройств, библиотек. В тех случаях, когда объём ОЗУ и ПЗУ мал (в районе нескольких килобайт), альтернативы ассемблеру нет. Именно эти языки программирования позволяют получать самый короткий и самый быстродействующий код программы.

Машинно-независимые языки программирования

Машинно-независимые языки — это средство описания алгоритмов решения задач и информации, подлежащей обработке. Они удобны в использовании для широкого круга пользователей и не требуют от них знания особенностей организации функционирования ЭВМ и вычислительной системы.

Подобные языки получили название высокоуровневых языков программирования. Программы, составляемые на таких языках, представляют собой последовательности операторов, структурированные согласно правилам рассматривания языка (задачи, сегменты, блоки и т.д.). Операторы языка описывают действия, которые должна выполнять система после трансляции программы на машинный язык.

Командные последовательности (процедуры, подпрограммы), часто используемые в машинных программах, представлены в высокоуровневых языках отдельными операторами. Программист получил возможность не расписывать в деталях вычислительный процесс на уровне машинных команд, а сосредоточиться на основных особенностях алгоритма.

Языки программирования высокого уровня значительно ближе и понятнее человеку. В них не учитываются особенности конкретных компьютерных архитектур, то есть данные языки являются машинно-независимыми. Это позволяет использовать однажды записанную на таком языке программу на различных ЭВМ.

Можно писать программы непосредственно на машинном языке, хотя это и сложно. На заре компьютеризации (в начале 1950-х гг.) машинный язык был единственным языком, бóльшего человек к тому времени не придумал. Для спасения программистов от сурового машинного языка программирования были созданы языки высокого уровня (т.е. немашинные языки), которые стали своеобразным связующим мостом между человеком и машинным языком компьютера. Языки высокого уровня работают через трансляционные программы, которые вводят «исходный код» (гибрид английских слов и математических выражений, который считывает машина) и в конечном итоге заставляют компьютер выполнять соответствующие команды, которые даются на машинном языке.

К языкам программирования высокого уровня можно отнести следующие: Fortran, Cobol, Algol, Pascal, Basic, C, C++, Java, HTML, Perl и другие.

С помощью языка программирования создаётся не готовая программа, а только её текст, описывающий ранее разработанный алгоритм. Чтобы получить работающую программу, надо либо автоматически перевести этот текст в машинный код и затем использовать отдельно от исходного текста, либо сразу выполнять команды языка, указанные в тексте программы. Для этого используются программы-трансляторы.

Существует два основных вида трансляторов (рисунок 4): *интерпретаторы*, которые сканируют и проверяют исходный код в один шаг, и *компиляторы*, сканирующие исходный код для производства текста программы на машинном языке, которая затем выполняется отдельно.

При использовании компиляторов весь исходный текст программы преобразуется в машинные коды, и именно эти коды записываются в память микропроцессора. При использовании интерпретатора в память микропроцессора записывается исходный текст программы, а трансляция производится при считывании очередного оператора. Естественно, что быстродействие интерпретаторов намного ниже по сравнению с компиляторами, т.к. при использовании оператора в цикле он транслируется многократно. Однако при программировании на языке высокого уровня объём кода, который нужно хранить во внутренней памяти, может быть значительно меньше по сравнению с исполняемым кодом. Ещё одним преимуществом применения интерпретаторов является лёгкая переносимость программ с одного процессора на другой.

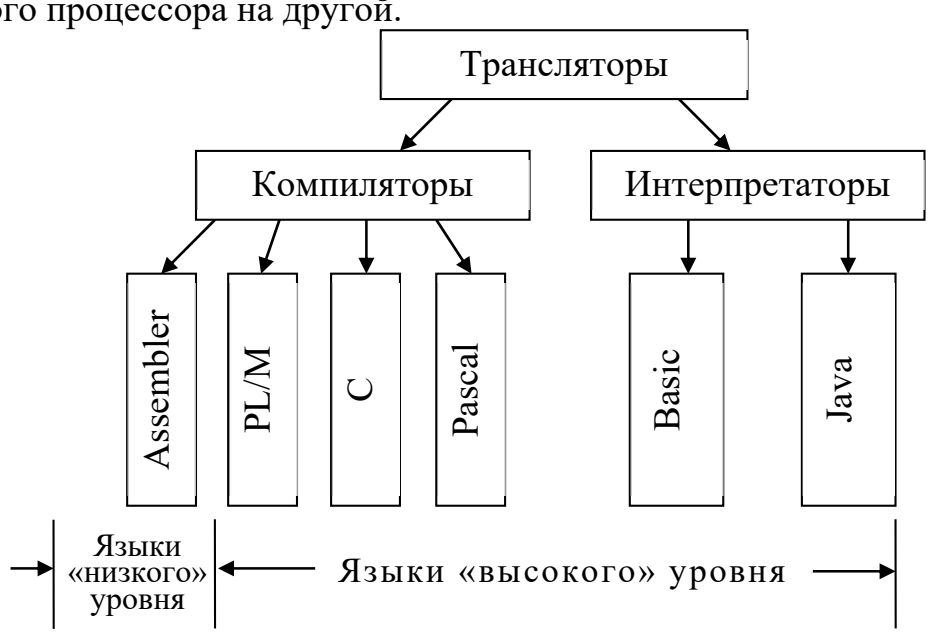


Рисунок4– Виды трансляторов

Одно, часто упоминаемое преимущество интерпретаторной реализации состоит в том, что она допускает «непосредственный режим». Непосредственный режим позволяет вам задавать компьютеру задачу и возвращает вам ответ, как только вы нажмёте клавишу ENTER. Кроме того, интерпретаторы имеют специальные атрибуты, которые упрощают отладку. Можно, например, прервать обработку интерпретаторной программы, отобразить содержимое определённых переменных, бегло просмотреть программу, а затем продолжить исполнение. Однако интерпретаторные языки имеют недостатки. Необходимо, например, иметь копию интерпретатора в памяти всё время, тогда как многие возможности интерпретатора, а следовательно, и его возможности могут не быть необходимыми для исполнения конкретной программы. При исполнении программных операторов интерпретатор должен сначала сканировать каждый оператор с целью прочтения его содержимого (что этот человек просит меня сделать?), а затем выполнить запрошенную операцию. Операторы в циклах сканируются излишне много.

Компилятор — это транслятор текста на машинный язык, который считывает исходный текст. Он оценивает его в соответствии с синтаксической конструкцией языка и переводит на машинный язык. Другими словами, компилятор не исполняет программы, он их строит. Интерпретаторы невозможно отделить от программ, которые ими прогоняются, компиляторы делают своё дело и уходят со сцены. При работе с компилирующим языком, таким, как Турбо-Бейсик, вы придёте к необходимости мыслить о ваших программах в признаках двух главных фаз их жизни: периода компилирования и периода прогона. Большинство программ будут прогоняться в четыре−десять раз быстрее их интерпретаторных эквивалентов. Если вы поработаете над улучшением, то сможете достичь 100-кратного повышения быстродействия. Оборотная сторона монеты состоит в том, что программы, расходующие бóльшую часть времени на возню с файлами на дисках или ожидание ввода, не смогут продемонстрировать какое-то впечатляющее увеличение скорости.

Процесс создания программы называется *программированием*.

Выделяют несколько разновидностей программирования:

Алгоритмическое или модульное

Основная идея алгоритмического программирования — разбиение программы на последовательность модулей, каждый из которых выполняет одно или несколько действий. Единственное требование к модулю − чтобы его выполнение всегда начиналось с первой команды и всегда заканчивалось на самой последней (то есть чтобы нельзя было попасть на команды модуля извне и передать управление из модуля на другие команды в обход заключительной).

Алгоритм на выбранном языке программирования записывается с помощью команд описания данных, вычисления значений и управления последовательностью выполнения программы.

Текст программы представляет собой линейную последовательность операторов присваивания, цикла и условных операторов. Таким способом можно решать не очень сложные задачи и составлять программы, содержащие несколько сот строк кода. После этого понятность исходного текста резко падает из-за того, что общая структура алгоритма теряется за конкретными операторами языка, выполняющими слишком детальные, элементарные действия. Возникают многочисленные вложенные условные операторы и операторы циклов, логика становится совсем запутанной, при попытке исправить один ошибочный оператор вносится несколько новых ошибок, связанных с особенностями работы этого оператора, результаты выполнения которого нередко учитываются в самых разных местах программы.

Структурное программирование

При создании средних по размеру приложений (несколько тысяч строк исходного кода) используется структурное программирование, идея которого заключается в том, что структура программы должна отражать структуру

решаемой задачи, чтобы алгоритм решения был ясно виден из исходного текста. Для этого надо иметь средства для создания программы не только с помощью трёх простых операторов, но и с помощью средств, более точно отражающих конкретную структуру алгоритма. С этой целью в программирование введено понятие подпрограммы − набора операторов, выполняющих нужное действие и не зависящих от других частей исходного кода. Программа разбивается на множество мелких подпрограмм (занимающих до 50 операторов — критический порог для быстрого понимания цели подпрограммы), каждая из которых выполняет одно из действий, предусмотренных исходным заданием. Комбинируя эти подпрограммы, удаётся формировать итоговый алгоритм уже не из простых операторов, а из законченных блоков кода, имеющих определённую смысловую нагрузку, причём обращаться к таким блокам можно по названиям. Получается, что подпрограммы — это новые операторы или операции языка, определяемые программистом.

Возможность применения подпрограмм относит язык программирования к классу процедурных языков.

Наличие подпрограмм позволяет вести проектирование и разработку приложения сверху вниз — такой подход называется *нисходящим проектированием*. Сначала выделяется несколько подпрограмм, решающих самые глобальные задачи (например, инициализация данных, главная часть и завершение), потом каждый из этих модулей детализируется на более низком уровне, разбиваясь, в свою очередь, на небольшое число других подпрограмм, и так происходит до тех пор, пока вся задача не окажется реализованной.

Такой подход удобен тем, что позволяет человеку постоянно мыслить на предметном уровне, не опускаясь до конкретных операторов и переменных. Кроме того, появляется возможность не реализовывать сразу некоторые подпрограммы, а временно откладывать, пока не будут закончены другие части. Например, если имеется необходимость вычисления сложной математической функции, то выделяется отдельная подпрограмма такого вычисления, но реализуется она временно одним оператором, который просто присваивает заранее выбранное значение. Когда всё приложение будет написано и отлажено, тогда можно приступить к реализации этой функции.

Немаловажно, что небольшие подпрограммы значительно проще отлаживать, что существенно повышает общую надёжность всей программы.

Очень важная характеристика подпрограмм — это возможность их *повторного использования*. С интегрированными системами программирования поставляются большие библиотеки стандартных, подпрограмм, которые позволяют значительно повысить производительность труда за счёт использования чужой работы по созданию часто применяемых подпрограмм.

Подпрограммы бывают двух видов − *процедуры* и *функции*. Отличаются они тем, что процедура просто выполняет группу операторов, а функция вдобавок вычисляет некоторое значение и передаёт его обратно в главную

программу (*возвращает* значение). Это значение имеет определённый тип (говорят, что функция имеет такой-то тип).

Подпрограммы решают три важные задачи:

- избавляют от необходимости многократно повторять в тексте программы аналогичные фрагменты;
- улучшают структуру программы, облегчая, её понимание;
- повышают устойчивость к ошибкам программирования и непредвидимым последствиям при модификациях программы.

Объектно-ориентированное программирование

В середине 80-х годов в программировании возникло новое направление, основанное на понятии объекта. До того времени основные ограничения на возможность создания больших систем накладывала разобщённость в программе данных и методов их обработки.

Реальные объекты окружающего мира обладают тремя базовыми характеристиками: они имеют набор свойств, способны разными методами изменять эти свойства и реагировать на события, возникающие как в окружающем мире, так и внутри самого объекта. Именно в таком виде в языках программирования и реализовано понятие объекта как совокупности свойств (структур данных, характерных для этого объекта), методов их обработки (подпрограмм изменения свойств) и событий, на которые данный объект может реагировать, и которые приводят, как правило, к изменению свойств объекта.

Появление возможности создания объектов в программах качественно повлияло на производительность труда программистов. Максимальный объём приложений, которые стали доступны для создания группой программистов из 10 человек, за несколько лет увеличился до миллионов строк кода, при этом одновременно удалось добиться высокой надёжности программ и, что немаловажно, повторно использовать ранее созданные объекты в других задачах.

Объекты могут иметь идентичную структуру и отличаться только значениями свойств. В таких случаях в программе создаётся новый тип, основанный на единой структуре объекта. Он называется *классом*, а каждый конкретный объект, имеющий структуру этого класса, называется *экземпляром класса*.

Объектно-ориентированный язык программирования характеризуется тремя основными свойствами:

1. *Инкапсуляция* — объединение данных с методами в одном классе;

2. *Наследование* — возможность создания, на основе имеющегося класса, нового класса с наследованием всех его свойств и методов и добавлением собственных;

3. *Полиморфизм* — присвоение действию одного имени, которое затем совместно используется вниз и вверх по иерархии объектов, причём каждый объект иерархии, выполняет это действие способом, подходящим именно ему.

С помощью компьютера может быть решена практически любая задача, если она поддаётся формализации. При этом процесс её решения представляет собой регулярный процесс обработки информации по определённым правилам, совокупность которых называют ***алгоритмом*** решения задачи.

Вопросы для самоконтроля

1. Алгоритм – понятие и формы представления, свойства.
2. Блок-схема – понятие, основные требования ГОСТ к оформлению (привести примеры).
3. Описание алгоритма на языке программирования.
4. Понятие языки программирования.
5. Машинно-зависимые и машинно-независимыеязыки программирования.
6. Виды программирования.

ТЕМА 2. ОСНОВНЫЕ ОБЪЕКТЫ ЯЗЫКА ПАСКАЛЬ

2.1 Общая характеристика языка

Язык программирования Паскаль назван в честь французского учёного Блейза Паскаля (1623-1662) и разработан профессором Цюрихского Института информатики (Швейцария) Никлаусом Виртом в 70-х годах.

В этом языке отразились лучшие черты языков программирования шестидесятых годов, в первую очередь языка Алгол.

Язык Паскаль является языком высоко уровня. Язык прост и удобен для обучения программированию и для создания систем программирования.

2.2 Алфавит языка

Алфавит языка Паскаль использует следующие символы:

1. Буквы – 26 латинских букв от A до Z. Разницы между прописными и заглавными буквами нет.
2. Цифры: от 0 до 9.
3. Специальные символы: **; : = + - * / _ ( ) [ ] { } , . ‘ “ <> # $ ^ &.**
4. Ключевые (зарезервированные) слова, таких слов 55 и смысл каждого из них зафиксирован в языке: AND, BEGIN, END, ARRAY и т.д.
5. Знаки операций:

- арифметические: +, -, *, /, DIV (целочисленное деление), MOD (нахождение остатка от деления);
- отношения: <, >, =, <=, >=, <>;
- логические: NOT (отрицание), AND (логическое умножение), OR (логическое сложение).

2.3 Идентификаторы языка, константы и переменные

Для сокращения объёма программы, возможности многократного использования её при различных значениях данных и упрощения работы программиста, в записи программы используются не сами значения или их фактические адреса (адреса ячеек памяти, где хранятся значения), а их имена – идентификаторы. Каждому идентификатору Pascal автоматически ставит в соответствие, какое то место в оперативной памяти ПЭВМ (номер ячейки, где храниться нужное значение).

Имя образуется из букв и цифр и начинается всегда с буквы. Имена используются для записи в программах переменных, констант, типов данных, функций, процедур (подпрограмм), файлов. Желательно чтобы имя переменной было логически связанно с хранимым в ней значением.

В качестве идентификаторов нельзя использовать служебные слова и имена стандартных функций.

Примеры записи имён: X, X1, SUMMA, SUMMA2, MAX_ALFA.

MIN и **min**– одно и тоже.

Константа представляет собой значение, которое известно заранее и не изменяется в ходе выполнения программы.

Переменная используется для записи значений изменяющихся в программе.

2.4 Стандартные типы данных

Паскаль является типизированным, или статическим языком. Это значит, что тип переменной определяется при её описании и не может быть изменён. Переменная может участвовать только в тех операциях, которые допускаются её типом. Такой подход способствует большей аккуратности и ответственности при составлении программы, облегчает компиляцию (перевод) и повышает надёжность создаваемых программ.

К стандартным типам данных относятся данные целочисленного типа, вещественного, булевского, литерные (строковые и символьные).

Данное является **ЦЕЛОЧИСЛЕННЫМ**, если оно представляет собой целое число в диапазоне от -32768 до +32767. Для описания целочисленного данного в Паскале используется слово **INTEGER**.

К данным целого типа можно применять битовые операции, т.е. работать напрямую с байтами. Можно прибавлять, отнимать, умножать. Для этого используют специальные функции.

Данные **ВЕЩЕСТВЕННОГО** типа, по своей сути представляют десятичные дроби.

Для описания используют служебное слово **REAL**. Данные работают в диапазоне от 2.9 E -39 до 1.7 E +38 (положительные и отрицательные). Воспринимается 11-12 цифра после десятичной точки (без сопроцессора).

Переменные и константы **БУЛЕВСКОГО** типа могут иметь 2 значения: **TRUE** (истина) и **FALSE** (ложь). Для описания используют слово **BOOLEAN**.

ЛИТЕРНЫЙ тип данных. Переменные и константы этого типа могут содержать любые символы латинского и русского алфавита. Значения переменных литерного типа заключаются в апострофы.

Константы литерного типа часто используют для вывода на экран дисплея или при печати необходимых комментариев.

2.5 Стандартные функции

Стандартные (встроенные) функции служат для облегчения записи и обращения к наиболее часто встречающимся функциям при обработке данных. В языке Pascal используются следующие стандартные функции:

- арифметические;
- функции преобразования типа переменной;

– функции для обработки величин порядкового типа.

При обращении к стандартной функции необходимо записать имя функции и в круглых скобках указать аргумент (аргументы). Наиболее часто встречаемые функции приведены в таблице 2.

Таблица 2 – Часто встречаемые стандартные (встроенные) функции

Математическая запись формулы	Обозначение функции на Pascal
x^2	SQR (x)
$\sqrt{x}$	SQRT (x)
$\lvert x \rvert$	ABS (x)
e^x	Exp (x)
Sin x	Sin (x)
Cos x	Cos (x)
Arctg (x)	ARCTAN (x)
Ln x	Ln (x)
Lg x	Ln(x)/Ln(10)
$\mathrm{Log}_a x$	Ln(x)/Ln(a)
a^x	Exp (x*Ln(a))

2.6 Арифметические выражения

Арифметическим выражением называется совокупность арифметических переменных, констант и функций, соединённых знаками арифметических действий.

Арифметические действия выполняются слева направо с соблюдением следующего старшинства (приоритета):

1) **NOT**

2) *****, /, **DIV**, **MOD**, **AND**

3) +, **-**, **OR**

Для изменения последовательности арифметических действий могут использоваться круглые скобки.

2.7 Оператор

Оператор **—** это предписание в данном языке программирования, предназначенное для задания некоторого завершающего действия в процессе обработке информации на ЭВМ.

С помощью операторов описываются действия над данными, выполняемые для реализации алгоритма решения задачи.

В языке Паскаль по своему составу определены операторы двух типов: простые и структурные.

Простым считается оператор, который не содержит других операторов.

К простым относятся: операторы присвоения, перехода.

Структурный — это оператор, содержащий в качестве компонентов один или несколько операторов.

К структурным относят: условные, выбора, цикла, составные.

2.8 Структура Паскаль-программы

Программа на языке Паскаль состоит из заголовка программы, описательной части (раздел описания) и исполнительной части (раздел операторов).

Заголовок программы

Заголовок программы имеет вид:

PROGRAM<Имя>;

Например, **PROGRAMPROBA;**

В заголовке определяется имя программы.

Раздел описаний

Раздел **ОПИСАНИЯПЕРЕМЕННЫХ** начинается со служебного слова **VAR**, за которым перечисляются все имена переменных используемых в программе и их тип.

Общий вид: **VAR <Имя переменной> :< Тип данных >;**

Например, **VAR**

A, B, I : INTEGER; MAX, MIN: REAL;

Раздел операторов

Раздел операторов начинается со служебного слова **BEGIN** и заканчивается словом **END** с точкой (**.**). В разделе записываются необходимые операторы в соответствии с алгоритмом решения поставленной задачи. Каждый оператор заканчивается символом точка с запятой (**;**).

Вопросы для самоконтроля

1. Алфавит алгоритмического языка Паскаль.
2. Идентификаторы языка, константы и переменные.
3. Стандартные типы данных.
4. Стандартные (встроенные) функции языка Паскаль. Формирование с их помощью математических выражений (привести пример).
5. Арифметические выражения (привести примеры).
6. Оператор. Типы операторов (привести примеры).
7. Структура Паскаль программы, схема её формирования на экране (привести пример).
8. Структура раздела описания (привести примеры).

ТЕМА 3. ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ЛИНЕЙНОЙ СТРУКТУРЫ

3.1 Понятие алгоритма линейной структуры

Линейным называется вычислительный процесс, в котором все действия выполняются последовательно без изменения их естественного порядка следования. Алгоритм (блок-схема) линейного процесса представляет собой последовательное расположение блоков. Значения данных и промежуточные результаты на порядок выполнения этих блоков не оказывают влияния (рисунок 5).

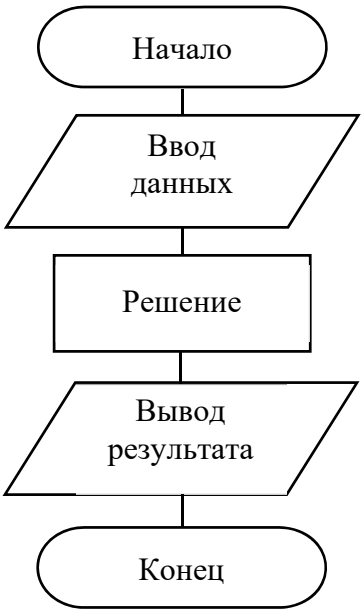


Рисунок 5– Блок-схема линейного процесса

Для программирования линейных вычислительных процессов используют оператор присвоения, оператор ввода и вывода.

3.2 Оператор присвоения

Оператор присвоения служит для вычисления значения выражения и присвоения его переменной, расположенной слева от символа**:=**

Общий вид оператора:

P := A;

где P–имя переменной;

A – арифметическое выражение (в частном случае может быть константа либо имя переменной).

Данный оператор так же называют арифметическим оператором, что подчёркивает арифметический характер вводимых в него объектов.

Например:

X := 0.37; **X := X + 0.49;**

T := -26.7; **A := T;**

F := 3 * C + 2 * SIN(X);

Действия в операторе выполняются слева направо с соблюдением следующего приоритета:

1) **NOT**
2) *, /, **DIV**, **MOD**, **AND**
3) +, -, **OR**

Любое выражение в скобках вычисляется раньше, чем операция, предшествующая скобкам.

В операторе присвоения переменная **P** и выражение **A** должны иметь один и тот же тип, за исключением:

REAL := INTEGER – (“маленький тип” присваивается в “большой”).

3.3 Операторы ввода и вывода

Эти операторы обеспечивают программу исходными данными, необходимыми для решения задачи и осуществляют вывод результатов решения.

ОПЕРАТОР ВВОДА

Общий вид оператора ввода:

READ(перемен1, перемен2, ..., переменN);

READLN(перемен1, перемен2, ..., переменN);

Имена вводимых переменных в обоих операторах ввода, заключаются в круглые скобки и отделяются друг от друга запятыми.

Данные операторы производят считывание исходных данных, вводимых с клавиатуры (данные высвечиваются на экране монитора). При выполнении данных операторов, выполнение программы приостанавливается и ЭВМ ожидает ввода значения исходных данных. Вводимые значения отделяются друг от друга пробелами. После окончания ввода блока исходных данных, пользователь подтверждает их значения нажатием клавиши**Enter** (ввод).

Оператор READLN отличается от оператора READ тем, что оператор READ выполняет ввод нескольких значений в строчку, (значения отделяются друг от друга пробелами), а оператор READLN позволяет ввести значения в столбец (т.е. после каждого значения нажимается Enter).

ОПЕРАТОР ВЫВОДА

Эти операторы позволяют вывести значения переменных используемых в программе.

Общий вид оператора вывода:

WRITE(перемен1, перемен2, ..., переменN);

WRITELN(перемен1, перемен2, ..., переменN);

Также как и в операторе ввода, переменные заключаются в круглые скобки и разделяются запятыми. Но теперь это имена выводимых переменных (т.е. в переменной хранится, какое либо значение, которое нам необходимо вывести для просмотра).

Операторы вывода кроме имени выводимых переменных в скобках могут содержать комментарии к вводимым и выводимым значениям. Комментарий представляет собой строку литеров, которая заключаются в апострофы.

Оператор **WRITELN** отличается от оператора **WRITE** тем, что после окончания вывода, перемещает курсор на начало новой строки.

Для организации диалога при вводе и выводе исходных данных операторы ввода и вывода часто используют совместно (диалоговый режим ввода):

WRITE('ВВОДХ = '); READLN(X);

WRITE('ВВОДС = '); READLN(C);

Оператор **WRITELN;** без параметров просто переводит курсор на новую строку, что можно использовать для более наглядного изображения диалога на экране.

Для более понятного восприятия выводимых значений используют формат. Формат записывается после переменной:

WRITE(ПЕРЕМЕННАЯ :ФОРМАТ);

Если выводимое значение является переменной типа REAL, то формат состоит из двух значений:

ПЕРЕМЕННАЯ : Всего символов : Кол-во символов после дес. точки

Для переменной типа INTEGER или STRING используется только одно значение:

ПЕРЕМЕННАЯ : Всего символов

Например,

WRITE(ALFA:10:3); – оставить под выводимое значение 10 позиций, три из которых отводятся под дробную часть числа.

_ _ _ _ - 0 . 270 – впереди 4 пробела, 1 минус, 0, точка и три знака под дробную часть.

3.4 Разработка алгоритма линейной структуры

Общий алгоритм решения задач линейной структуры

1. Определение входных и выходных данных.
2. Составление блок-схемы.
3. Составление Паскаль-программы.

Задача

№ 1. Вычислить $y = F(a+b)$, где $a = x^2$;

№ 2. Вычислить $A = \cos\alpha + x^2$, где $x = \frac{y-c}{2}$;

№ 3. Вычислить $D = \frac{b^2 - c^2}{2}$, где $b = \sqrt{c-a}$, $c = x + y$;

№ 4. Вычислить $y = \frac{\sin\alpha - \cos\beta}{2a}$, где $a = x^2 - 5$;

№ 5. Вычислить $h = \left(x + \pi^2\right) - \frac{\pi}{2}$;

№ 6. Вычислить $d = \sqrt{\cos 2x} - \lg\frac{z}{2}$;

№ 7. Вычислить $S = 1 + \frac{(1+x)/x}{x + (1/y)}$;

№ 8. Вычислить $Z = \frac{y-1}{\sqrt{x^3+1}} + e^{-ax}$;

№ 9. Вычислить $r = \frac{a^2 \cdot x^3 + \ln\frac{a}{x}}{e^{-x^2} + \cos(bx-1)}$;

№ 10. Вычислить $Z = \left((x + 3a + y)/2x\right)^4 - \frac{x}{x + 3a - y}$.

Вопросы для самоконтроля

1. Какой вычислительный процесс называют линейным?
2. Общий вид блок-схемы линейного процесса.
3. Оператор присваивания, приоритеты выполнения математических и логических операций в языке Паскаль.
4. Операторы ввода данных, организация ввода (привести пример).
5. Операторы вывода данных, организация вывода (привести пример).
6. Алгоритм решения задач линейной структуры.
7. Особенности ввода и вывода данных в программе на языке Паскаль.

ТЕМА 4. ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ РАЗВЕТВЛЯЮЩЕЙСЯ СТРУКТУРЫ

4.1 Понятие разветвляющегося процесса

Вычислительный процесс называется разветвляющимся, если в зависимости от выполнения некоторого условия он протекает по одному из нескольких направлений.

Каждое направление называется ветвью. Процесс, имеющий две ветви называется простым, а более двух ветвей – сложным разветвляющимся процессом.

Простые разветвляющие процессы могут быть двух типов (рисунок 6):

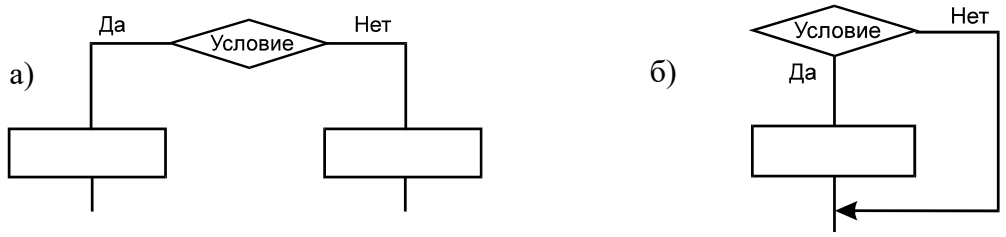


Рисунок 6 – Разветвляющиеся процессы

– вычисления предусматриваются по обеим ветвям (рисунок 3, а);

– вычисления предусматриваются только по одной ветви, а по другой выполняется переход к следующему этапу алгоритма (рисунок 3, б).

Для организации разветвления в программах используются операторы БЕЗУСЛОВНОГО ПЕРЕХОДА и УСЛОВНОГО ПЕРЕХОДА.

4.2 Оператор безусловного перехода

Общий вид оператора: **GOTO m;**

где **GOTO** – зарезервированное слово (перейти на метку);

m– метка оператора, куда производится переход.

Таким образом, следующим после оператора **GOTO m;** будет выполняться оператор с меткой **m**.

Метка **—** это последовательность букв и цифр, входящих в алфавит языка. Все используемые в программах метки должны быть описаны в обязательной части программы, в разделе **LABEL**.

Например: **GOTO a1;**

В программе перед оператором, на который необходимо перейти ставится метка и двоеточие.

Например: **a1: Writeln(‘Решения нет’);**

4.3 Оператор условного перехода

Общий вид:

IF <условие> THEN <оператор 1> ELSE < оператор 2>;

где **IF –** зарезервированное слово *если;*

THEN – зарезервированные слова *то;*

ELSE – зарезервированные слова *иначе*;

<условие> – произвольное выражение логического типа;

<оператор1>, <оператор2> – любые отдельные операторы языка *Паскаль*.

Условный оператор работает по следующему алгоритму.

Если условие выполняется, то выполняется оператор, записанный после слова **THEN**, т.е. **оператор 1**. Если условие не соблюдается, то выполняется **оператор 2**, записанный после слова **ELSE**. Таким образом, в каждом конкретном случае будет выполнен либо **оператор 1**либо **оператор 2**. Блок-схема данного оператора представлена на рисунке 3,а.

Данный оператор может быть использован в сокращённом варианте:

IF <условие> THEN <оператор>;

В этом случае, при не соблюдении условия будет выполняться оператор, записанный следующим за оператором условного перехода. Этому оператору соответствует второй тип разветвляющего процесса, блок-схема которого представлена на рисунке 3,б.

Например,**IF P > 0.5 THEN T:=X*X;**

Перед служебным словом **ELSE** точка с запятой не ставиться.

По требованию Паскаля после служебных слов **THEN** и **ELSE** может располагаться только по одному оператору. Если по алгоритму решения задачи после **THEN** или **ELSE** необходимо выполнить несколько операторов, то используется составной оператор.

4.4 Составной оператор

Составной оператор представляет собой совокупность нескольких операторов разделённых символом точка с запятой (**;**)и заключённых в операторные скобки **BEGIN** и **END:**

Общий вид:

BEGIN <оператор 1>; <оператор 2>; ... END;

Составной оператор воспринимается компилятором как один оператор и, следовательно, составной оператор может располагаться в том месте программы где, по требованию PASCAL, разрешается расположение только одного оператора.

Оператор условного перехода допускает в качестве своих операторов использовать новые условия. Новые условия так же записываются после слов **THEN** или **ELSE**. По соглашению слово **ELSE** всегда относится к ближайшему оператору условного перехода (к слову **IF**):

IF<условие1>THEN

IF <условие2> THEN <оператор1 усл.2> ELSE <оператор2 усл.2>;

Например,

```
IF X <= A THEN Z:=SIN(X) ELSE
    IF X > B THEN Z:=SIN(X)/COS(X) ELSE
        IF X <= B THEN Z:=COS(X) ELSE GOTO AA1;
```

В операторе условного перехода можно записывать и более сложные условия. Для этого используют служебные слова **AND** (и) и **OR** (или).

IF (<условие1>) <сл. слово> (<условие2>) THEN <опер1> ELSE <опер2>;

Так при использовании служебного слова **AND** оператор будет работать по следующей схеме – если условие 1 и условие 2 верно – то выполнится оператор 1, в противном случае, если хотя бы одно из условий неверно выполняется оператор 2.

При использовании служебного слова **OR** оператор работает так – если хотя бы одно из двух условий верно (условие 1 или условие 2) то выполнится оператор 1, в противном случае (когда два условия неверны) выполняется оператор 2.

4.5 Разработка алгоритма разветвляющейся структуры

Общий алгоритм решения задач разветвляющейся структуры

1. Определение входных и выходных данных.
2. Составление блок-схемы.
3. Составление Паскаль-программы.

Задача

№ 1. Вычислить $y = \begin{cases} x^2, & \text{если } x \ge 0 \\ 10 - x, & \text{если } x < 0 \end{cases}$.

№ 2. Вычислить $z = \begin{cases} 2ax + |a - 1|, & \text{при } x \ge 1 \\ \sin^2 x + \ln x, & \text{иначе} \end{cases}$.

№ 3. Вычислить $k = \begin{cases} \sqrt{|a^3 - x^2|}, & \text{при } x > a \\ e^x - \sin a, & \text{иначе} \end{cases}$.

№ 4. Вычислить $k = \begin{cases} \cos x^2, & \text{при } x < 3{,}5 \\ \ln x, & \text{при } x = 3{,}5 \\ 1 - \sin x, & \text{при } x > 3{,}5 \end{cases}$.

№ 5. Вычислить $k=\begin{cases} a x^2+b, & \text{при } x=1 \\ x^3, & \text{при } x>1 \\ x^5+c, & \text{при } x<1 \end{cases}$;

№ 6. Вычислить $p=\begin{cases} \ln k+e^x, & \text{при } x<2 \\ \mathrm{x}^2+\cos a^2, & \text{при } x=2 \\ \dfrac{a^3-\sin^2 b}{\cos x}, & \text{при } x>2 \end{cases}$.

№ 7. Вычислить $i=\begin{cases} \dfrac{a^2}{c^2\cdot R\cdot W^2}, & \text{при } W<W1 \\ \dfrac{a}{W\cdot H}, & \text{при } W\geq W2 \end{cases}$.

№ 8. Вычислить $t=\begin{cases} \dfrac{m\cdot a\cdot n\cdot l}{60\cdot p}, & \text{при } n=0{,}33 \\ n\cdot k\cdot\dfrac{j_0}{j-j_0}, & \text{при } n=0{,}36 \end{cases}$.

№ 9. Вычислить $y=\begin{cases} x^2+a, & \text{при } 5<a<10 \\ \sqrt{a}, & \text{при } a\geq 10 \\ 15, & \text{в остальных случаях} \end{cases}$.

№ 10. Вычислить $k=\begin{cases} x+1, & \text{при } x<10 \\ x-2{,}1, & \text{при } x>\pi/2 \\ \sin x, & \text{при } x\leq \pi/2 \end{cases}$.

№ 11. Даны два числа. Если первое число меньше второго – то заменить первое – на нуль, а если нет – то оставить без изменения.

№ 12. Вычислить значение функции $y=\dfrac{1}{x}$.

№ 13. Вычислить $d=\begin{cases} e^{a+b}, & \text{при } a>b \\ (a+b)^2+\sin a, & \text{при } a=b \\ \sqrt{a+b}-\mathrm{b}^3, & \text{при } a<b \end{cases}$.

Вопросы для самоконтроля

1. Какой вычислительный процесс называется разветвляющимся?
2. Графическое представление разветвляющегося процесса.
3. Оператор безусловного перехода.
4. Оператор условного перехода.
5. Понятие составного оператора
6. Что представляют в Паскаль-программе операторные скобки, в каких случаях они используются?
7. Алгоритм решения задачи разветвляющегося вычислительного процесса.

ТЕМА 5.ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ЦИКЛИЧЕСКОЙСТРУКТУРЫ

5.1 Понятие об алгоритмах циклической структуры

Вычислительный процесс называется циклическим, если он содержит многократно повторяемые действия алгоритма или операторов программы.

Циклический процесс иначе называется повторяющимся. Повторяющие участки называются телом цикла. Кроме тела цикла выделяют заголовок цикла и конец цикла.

Для того чтобы сообщить компьютеру, в какой момент требуется прекратить выполнение тела цикла и перейти к очередному действию программы необходимо предусмотреть в программе условие выхода из цикла. Этот выход из цикла обычно реализуется путём проверки условия над переменной цикла.

Таким образом, алгоритм циклического процесса состоит из следующих шагов:

1. Выбор переменной цикла, установления диапазона и шага её изменения.

2. Присвоить переменной цикла начальное значение.

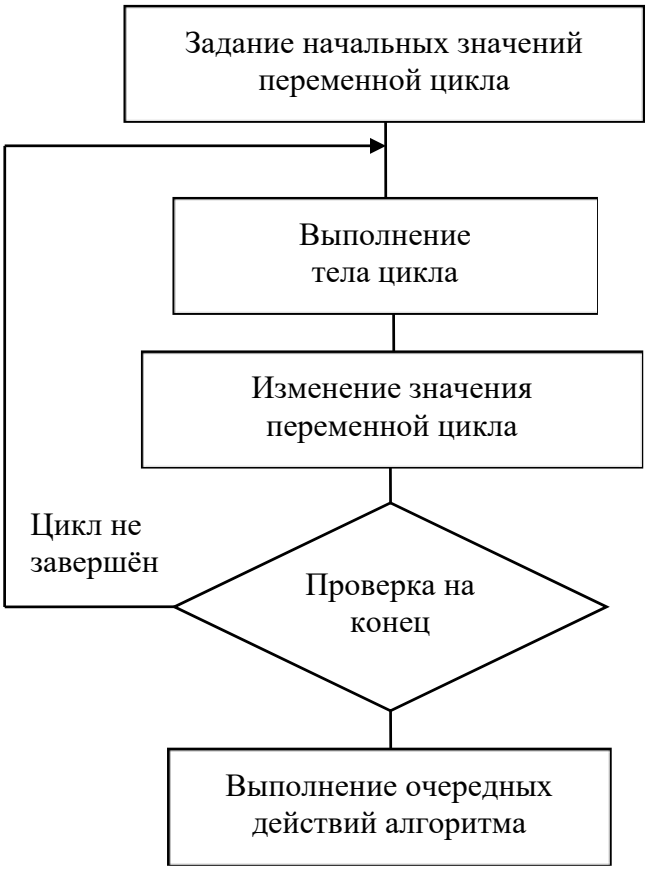


Рисунок 7– Общая схема циклического процесса

3. Выполнить требуемое условием задачи действие над переменной цикла.

4. Вывести полученный результат (если это требуется по условию задачи).

5. Произвести модификацию переменной цикла.

6. Выполнить проверку: достигла ли переменная цикла конечного значения? Если нет, то вернуться на пункт 3, в противном случае выйти из цикла.

Общая схема циклического процесса представлена на рисунке 7:

5.2 Операторы цикла с предусловием

Общий вид оператора:

WHILE <условие> DO <оператор>;

где **WHILE** – зарезервированное слово *пока;*

DO – зарезервированное слово *делать*;

<условие> – выражение логического типа, определяющее условие повторения действий, заданных в теле цикла;

<оператор> – произвольный оператор *Паскаля* простой либо составной.

Оператор работает по следующей схеме:

Пока условие истина (**TRUE**) то выполняется оператор (т.е. тело цикла), как только условие принимает значение ложь (**FALSE**) то выполнение тела цикла прекращается.

Если условие с самого начала имеет значение **FALSE**, то цикл не выполнится ни разу.

После служебного слова **DO** можно записать только один оператор. Если по условию алгоритма необходимо записать несколько операторов, то необходимо использовать операторные скобки **BEGIN** и **END**.

Графически оператор с предусловием можно представить в следующем виде (рисунок 8):

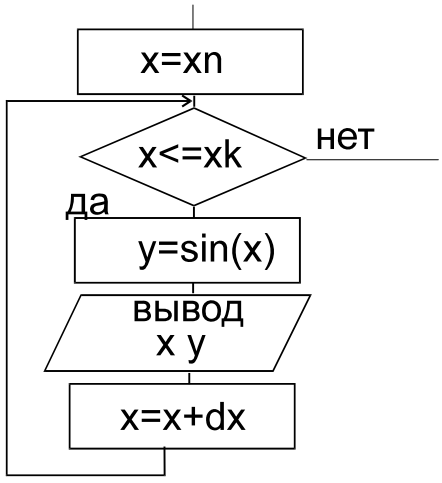


Рисунок 8– Алгоритм оператора цикла с предусловием

где **x** – переменная цикла;

xn, **xk** – диапазон изменения **X**;

dx – шаг изменения **X**.

В цикле вычисляется и выводится значение функции **y = sinx**.

Текст программы этого алгоритма выглядит следующим образом:

```
x:=xn;
While x <= xk Do Begin
    y:=Sin(x);
    Writeln(x:8:3,y:8:3);
    x:=x+dx;
End;
```

5.3 Оператор цикла с постусловием

Оператор имеет вид

REPEAT <тело_цикла> UNTIL <условие>;

где **REPEAT** – зарезервированное слово *повторять;*

UNTIL – зарезервированное слово *пока*;

<тело_цикла> – произвольная последовательность операторов *Паскаля*;

<условие> – выражение логического типа;

Этот оператор, в отличие от оператора цикла с предусловием, позволяет организовать цикл с неизвестным числом повторений. Условие проверятся после выполнения тела цикла. Таким образом, гарантируется хотя бы однократное выполнение тела цикла.

Оператор работает по следующей схеме: Оператор (тело цикла) повторяется пока условие **FALSE** (ложь), как только условие станет **TRUE** (истина) цикл прекращается.

Графически оператор с постусловием можно представить в следующем виде (рисунок 9).

Текст программы этого алгоритма выглядит следующим образом:

```
x:=xn;
Repeat
    y:=Sin(x);
    Writeln(x:8:3,y:8:3);
    x:=x+dx;
Untilx>xk;
```

Обратите внимание: составной оператор в этом цикле не используется.

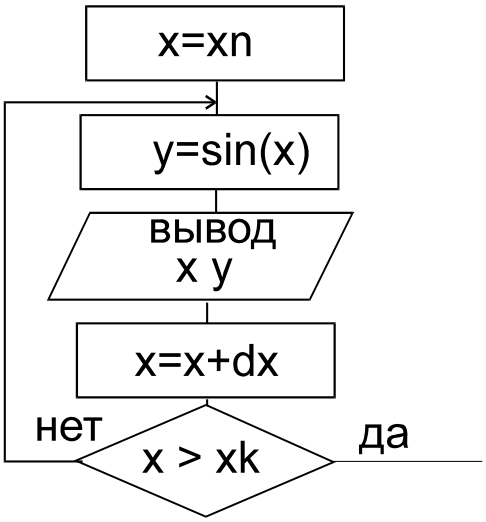


Рисунок 9 – Алгоритм оператора цикла с постусловием

5.4 Оператор цикла с параметром (целочисленный цикл)

Данный оператор используется для организации цикла с известным числом повторений. Так как число повторений тела цикла может быть числом целого типа (т.е. пять, десять повторений) то этот цикл также называется целочисленным циклом.

Оператор имеет вид:

FOR <пар_цик> := <нач. знач.> TO <конеч. знач.> DO <оператор>;

где **FOR** – зарезервированное слово *для;*

TO – зарезервированное слово *до;*

DO – зарезервированное слово *делать*;

<пар_цик> – параметр цикла (счётчик циклов, управляющая переменная цикла);

<нач._знач> – начальное значение параметра цикла;

<конеч._знач> – конечное значение параметра цикла;

<оператор> – оператор *Паскаля* (простой либо составной).

FOR <пар_цик>:= <нач.знач.>DOWNTO<конеч.знач.> DO <оператор>;

Оператор работает по следующей схеме: ДЛЯ переменной цикла равной начальному значению пока переменная цикла не станет больше конечного значения выполнять оператор (тело цикла). Шаг приращения для этого оператора равен 1 (для **DOWNTO** -1). То естьв первом случае цикл работает по «восходящей», а во втором случае по «нисходящей».

Все переменные этого цикла имеют только тип **INTEGER**.

Графически цикл представляется, так же как и цикл с предусловием, шаг равен 1 или -1.

Например:

Write (‘Введите число повторений N = ‘); readln (n);
for i:=1 to n do write(‘-’);

гдеIиN – переменные целого типа.

В результате выполнения программы на экране появится строка минусов (черта) длиной в N символов.

5.5 Программирование циклической структуры в случае задания аргумента в виде арифметической последовательности (задача табулирования функции)

Постановка задачи:

Для некоторых значений аргумента x необходимо вычислить значение функции $y = F(x)$. Причём значения аргумента отличаются друг от друга на постоянную величину шага Δx, т.е. $x_{i+1} = x_i + \Delta x$.

Исходными данными при табулировании заданной функции являются:

- ✓ начальное значение аргумента x_n;
- ✓ конечное значение аргумента x_k ;
- ✓ шаг изменения аргумента Δx.

В результате решения задачи табулирования функции, должна быть получена таблица, содержащая все значения вычисленных аргументов функции (таблица 3).

Таблица 3– Табулирование функции

x	y
x_n	$y = f(x_n)$
$x_n + \Delta x$	$y = f(x_n + \Delta x)$
⋮	⋮
x_k	$y = f(x_k)$

Общее количество значений аргумента и соответствующих ему значений функции можно определить по зависимости:

$$n = \frac{x_k - x_n}{\Delta x} + 1.$$

Таким образом, при программировании задачи табулирования функции мы имеем дело с классическим циклическим алгоритмом.

Например, вычислить функцию $y = \sin(x)$, при изменяющемся значении x от $x_n = 1$ до $x_k = 5$ с шагом $\Delta x = 1$.

Алгоритм табулирования функции может иметь следующий вид:

- при использовании цикла с предусловием (рисунок 10);
- при использовании цикла с постусловием (рисунок 11).

Использовать целочисленный цикл при программировании задачи табулирования функции нежелательно, так как диапазон изменения переменной цикла не обязательно может быть целочисленным.

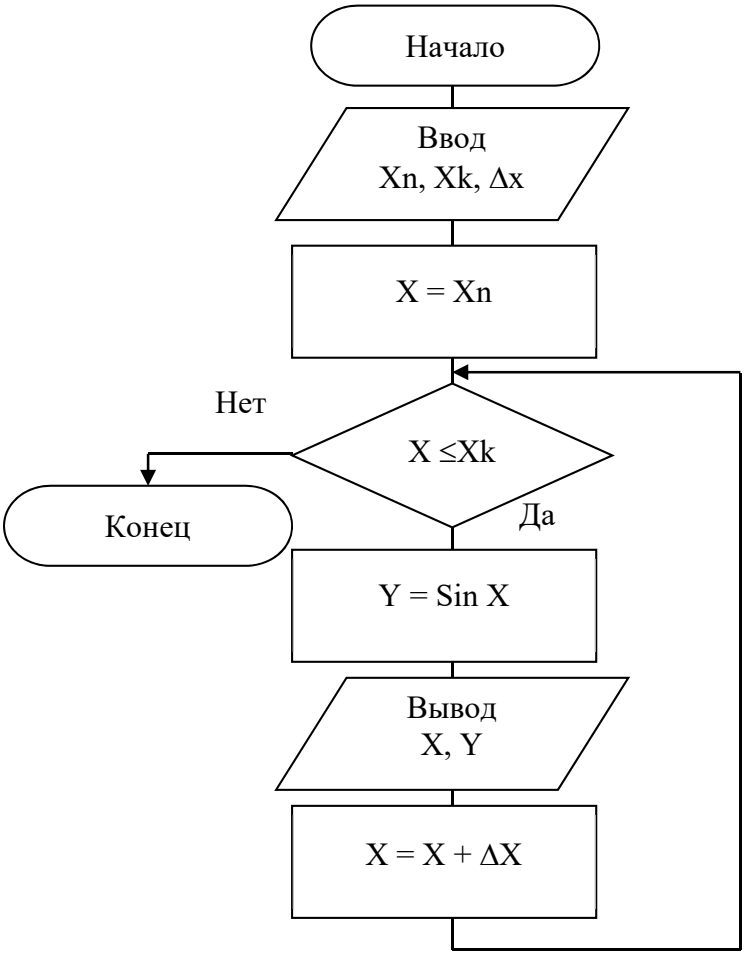


Рисунок 10 – Алгоритм табулирования функции (цикл с предусловием)

Программа табулирования функции с использованием цикла с предусловием выглядит следующим образом:

```
Program tabulir1;
Var
x,y,xn,xk,dx:Real;
Begin
    writeln('Введите диапазон изменения переменной');
    readln(xn,xk,dx);
    writeln(' ЗНАЧЕНИЯ ФУНКЦИИ')
    x:=xn;
```

```
    While x <= xk Do Begin
        y:=Sin(x);
        Writeln(x:8:3,y:8:3);
        x:=x+dx;
    End;
End.
```

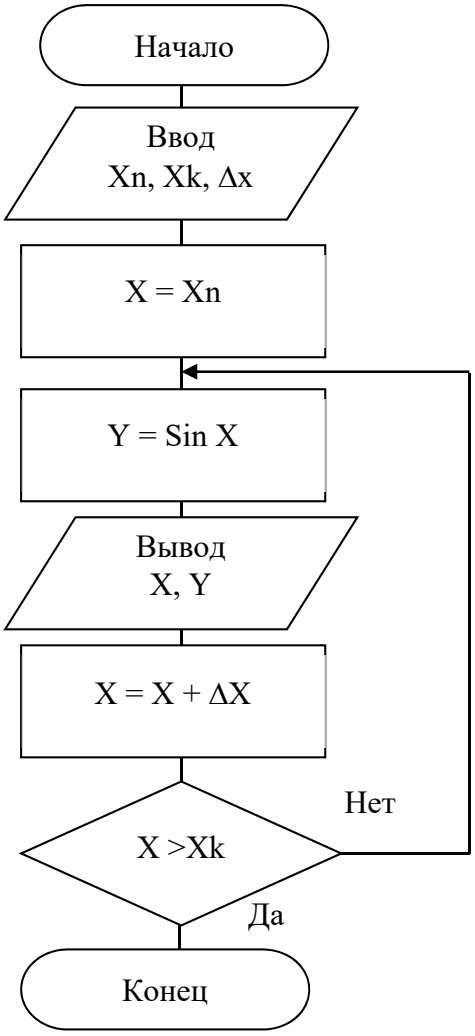


Рисунок 11 – Алгоритм табулирования функции (цикл с постусловием)

Программа табулирования функции с использованием цикла с постусловием выглядит следующим образом:

```
Program tabulir2;
Var
x,y,xn,xk,dx:Real;
Begin
    writeln('Введите диапазон изменения переменной');
    readln(xn,xk,dx);
    writeln('ЗНАЧЕНИЯ ФУНКЦИИ')
    x:=xn;
```

```
    Repeat
        y:=Sin(x);
        Writeln(x:8:3,y:8:3);
        x:=x+dx;
    Untilx>xk;
End.
```

При прогоне данного примера будет получен следующий результат (независимо от используемого оператора цикла):

Исходные данные **Xn = 1**, **Xk = 5**, **dX = 1**.

Таблица 4 – Контрольный пример задачи табулирования функции

Шаг	X	Y
1	1	0,841
2	2	0,909
3	3	0,141
4	4	-0,757
5	5	-0,959

5.6 Разработка алгоритмов циклической структуры

Общий алгоритм решения задач циклической структуры

1. Определение входных и выходных данных.
2. Составление блок-схемы.
3. Составление Паскаль-программы.

Задача

№ 1. Вычислить значение функции:

$y = \frac{\sqrt{\ln x}}{x^2} + c \cdot \sin(x + c)$, где x меняется от x_n до x_k с шагом Δx.

№ 2. Вычислить значение функции:

$x = (abz)^3 - \sqrt{\frac{a}{z} + \frac{b}{z^2}}$, где z меняется от z_n до z_k с шагом Δz.

№ 3. Вычислить значение функции: $Q = 1 + |w - r| + \sqrt[3]{w^2 + m^2}$,

$R = 2s + 3m$, где w меняется от w_n до w_k с шагом Δw.

№ 4. Вычислить значение функции: $T = 2x(1 + x^2 y^2)$, $x = a + b$,

где a меняется от a_n до a_k с шагом Δa.

№ 5. Вычислить значение функции: $y = \begin{cases} \sqrt{\dfrac{a\,x}{\sqrt{b+1}}}, & \text{при } a < 0 \\ 1{,}2(x+d), & \text{при } a \geq 0 \end{cases}$,

где x меняется от x_n до x_k с шагом Δx.

№ 6. Вычислить значение функции:,
где a меняется от a_n до a_k с шагом Δa.

№ 7. Вычислить значение функции: $p = \begin{cases} \dfrac{x^2+\sqrt{a}}{b}, & \text{при } a \geq 0 \\ a^3+\sqrt{x}, & \text{при } a < 0 \end{cases}$,

где x меняется от x_n до x_k с шагом Δx.

№ 8. Вычислить значение функции: $y = \begin{cases} a^3+5\sqrt{x}+c\,a^4, & \text{при } a \geq c \\ \ln(x+1), & \text{при } a < c \end{cases}$,

где x меняется от x_n до x_k с шагом Δx.

№ 9. Вычислить значение функции: $y = \begin{cases} a\,x+bx^2-c, & \text{при } x < 2 \\ \dfrac{a}{x}+\sqrt{x+1}, & \text{при } x = 2 \\ (a+6x)/\sqrt{x+1}, & \text{при } x > 2 \end{cases}$

где $x \in [0{,}1;2]$, с шагом $\Delta x = 0{,}25$.

Вопросы для самоконтроля

1. Какой вычислительный процесс называется циклическим?
2. Оператор цикла с предусловием.
3. Оператор цикла с постусловием.
4. Оператор цикла с параметром.
5. Задача табулирования функции.
6. Алгоритм решения задачи циклического вычислительного процесса.

ТЕМА 6. ПРОГРАММИРОВАНИЕ ЗАДАЧ С ОДНОМЕРНЫМИМАССИВАМИ

6.1 Понятия массива

Массив это фиксированная совокупность элементов, имеющих общее имя. Все компоненты массива имеют один и тот же тип данных.

Элементом массива является переменная с индексом, где индекс указывает местоположение элемента в массиве.

Математическим аналогом одномерного массива является вектор, состоящий из *n*-компонент $\overline{a} = (a_1, a_2, \ldots, a_n)$. Примером одномерного массива может служить среднесуточная температура воздуха за десять дней:

$$t = (t_1, t_2, \ldots, t_{10}).$$

Индексом в данном случае является номер суток в данной декаде.

Графически одномерный массив можно представить в виде (рисунок 12).

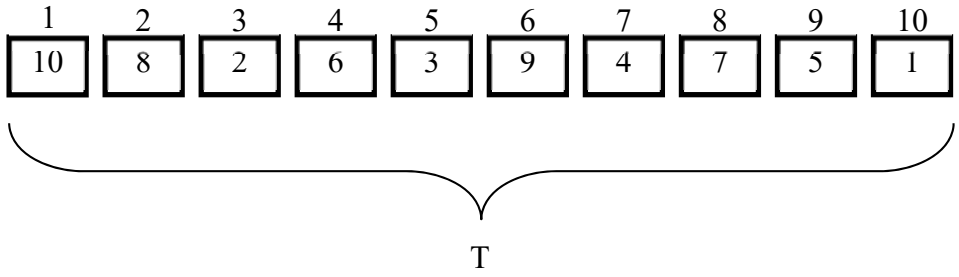


Рисунок 12 – Графическое представление одномерного массива

6.2 Описание размерности массива

Перед использованием массива его, как и другую переменную необходимо описать в разделе описания. В отличие от простых переменных, при описании массива необходимо описать ИМЯ МАССИВА, описать ТИП ПЕРЕМЕННЫХ в массиве и описать КОЛИЧЕСТВО ПЕРЕМЕННЫХ в массиве.

Описать массив в общем, виде можно по следующей схеме:

<ИМЯ> : ARRAY [L . . N] OF <ТИП>;
ИМЯ– имя массива (имя переменной);
ARRAY– служебное слово означающее МАССИВ;
L– начальное значение индекса массива;
N– конечное значение индекса массива;
OF– служебное слово, означающее ИЗ;
ТИП– стандартный или описанный тип данных.

Например,

T : ARRAY [1 .. 10] OF REAL;– массив (T) состоящий из 10 компонент имеющих вещественный тип данных (например среднесуточная температура воздуха за 10 дней).

Чаще всего начальное значение индекса L = 1. Однако это необязательно. Для удобства PASCAL позволяет присваивать любые целые значения индексов. Например, в программе используется массив, элементы которого представляют собой численность населения Москвы, в отдельные годы, начиная с года основания города (1147 г.). Описать такой массив можно следующим образом: **M :ARRAY [1147 . . 1999] of integer;**

6.3 Порядок индексации

Для обращения к элементу массива необходимо записать имя массива и указать индекс нужного элемента.

Например, **T[3]**– обратиться к третьему элементу массива. В нашем примере T[3] = 2.

Индекс элемента можно задавать несколькими способами:

– непосредственно с помощью числа (T[3]);

– косвенно с помощью переменной (T[i]);

– косвенно с помощью арифметического выражения (T[i+2]).

При задании индекса с помощью переменной, эту переменную необходимо описать в разделе описания. Данная переменная должна носить целочисленный тип данных (INTEGER).

6.4 Обработка массивов

Ввод, обработка и вывод массива осуществляется поэлементно. Например, при вводе массива температур сначала вводится t_1 , затем t_2 и так далее до t_{10} . Таким образом, у нас повторяется одно и то же действие (ввод, вывод или обработка), но для разных элементов массива, то есть, мы имеем циклический процесс.

Для программирования алгоритмов с массивами удобно использовать оператор цикла с параметрами или целочисленный цикл.

Пример алгоритма обработки одномерного массива представлен на рисунке 13.

Данный алгоритм на языке PASCAL будет выглядеть следующим образом:

```
For I := 1 to N do
Обработка элемента [I];
```

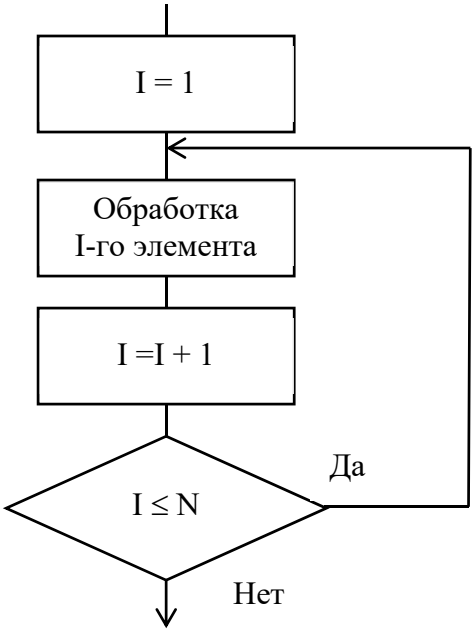


Рисунок 13– Пример алгоритма обработки одномерного массива

6.5 Характерные приёмы программирования задач с одномерными массивами (стандартные алгоритмы)

К характерным приёмам программирования относятся приёмы позволяющие вычислить (выполнить):

1) Ввод и вывод элементов массива.
2) Сумму элементов массива (всех или по какому либо критерию).
3) Произведение элементов массива (всех или по какому либо критерию).
4) Определение количества элементов массива.
5) Нахождение максимального элемента массива.
6) Нахождение минимального элемента массива.
7) Нахождение порядкового номера элемента в массиве, по какому либо признаку (Например, максимальное значение и его порядковый номер).
8) Нахождение среднего значения массива (всех или по какому либо критерию).

Для всех типов примеров будем рассматривать только часть программы и алгоритма.

Примем следующие обозначения:

N – количество элементов в массиве;

A – массив.

Для всех рассматриваемых алгоритмов раздел описания переменных будет иметь следующий вид:

Var

A : ARRAY [1 .. 50] OF Real;
I, N:Integer;

Ввод элементов массива

Для работы с массивом его необходимо заполнить конкретными значениями. Ввод данных в массив начинается с ввода количества элементов массива (N). Это число не может превышать размерность массива описанного в разделе описания. Затем вводятся N элементов массива.

Алгоритм этого процесса будет иметь вид (рисунок 14):

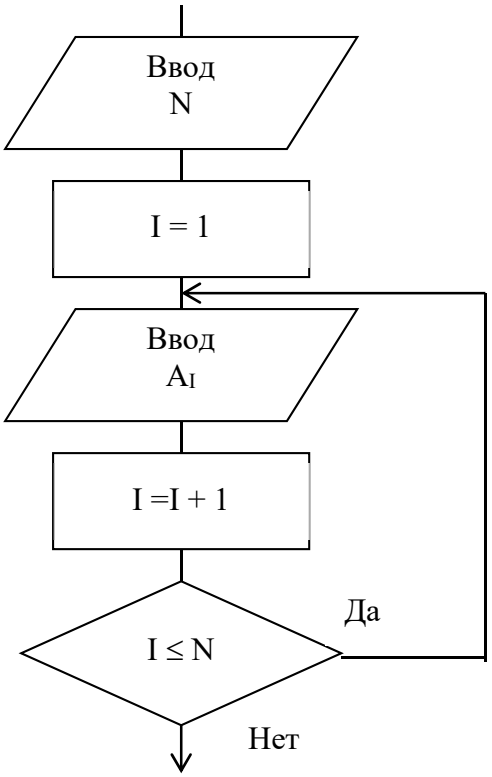


Рисунок 14 – Алгоритм ввода исходных данных в массив

Текст программы:

Read(n);
For i:=1 to n do
Readln (A[i]);

Вывод элементов из массива

Данный алгоритм аналогичен предыдущему, за исключением того, что вместо ввода данных в массив необходимо данные вывести из него последовательно друг за другом.

Алгоритм имеет вид:

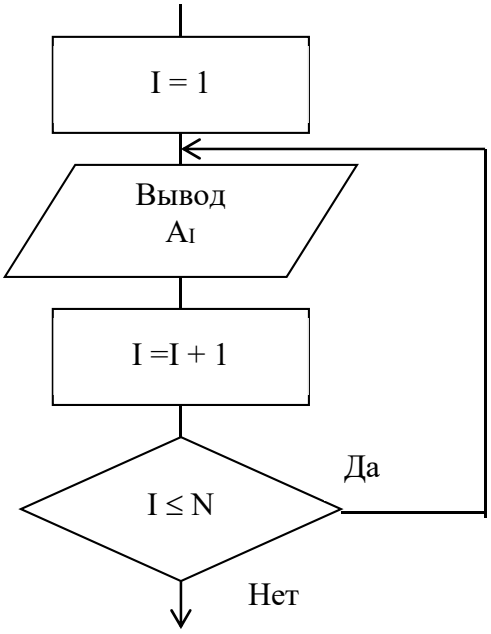


Рисунок 15 – Алгоритм вывода данных из массива

Текст программы:

```
For i:=1 to n do
Writeln (A[i]);
```

В диалоговом режиме данные алгоритмы будут выглядеть:

– ввод данных	– вывод данных
Write(‘ N = ‘); Readln(n); **For i:=1 to n do Begin** **Write(‘A[‘,i,’]= ‘);** **Readln(a[i]);** **End;**	**For i:=1 to n do** **Writeln(‘A[‘,i,’]=’,a[i]:8:3);**

В рассмотренных ниже алгоритмах блоки ввода массива будут опущены. Ввод элементов массива, как уже отмечалось выше, включает в себя ввод количества элементов массива N и поэлементный ввод массива.

Сумма элементов массива

Математическая запись:

$$S = A_1 + A_2 + A_3 + \ldots + A_n = \sum_{i=1}^{n} A_i \ .$$

Вычисление суммы происходит путём последовательного накапливания в переменной (S). Вначале $S = 0$, а затем для всех $i = 1,2,...,n$ выполняется суммирование $S = S + A_i$.

Например, массив A = {5 3 2}.

	S=0
i=1	S=S+A1=A1 0+5=5
i=2	S=S+A2= A1+A2 5+3=8
i=3	S=S+A3=A1+A2+A3 8+2=10

Алгоритм выглядит следующим образом (рисунок 16).

Переменная S должна иметь тот же тип что и массив. В нашем примере в раздел описания переменных необходимо добавить следующую строку:

S :Real;

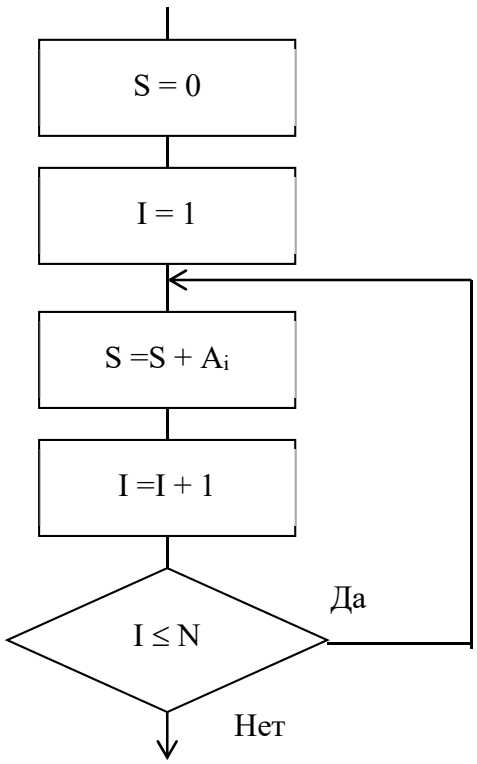


Рисунок 16 – Алгоритм определения суммы всех элементов массива

Элемент программы имеет вид:

```
s:=0;
For i:=1 to n do
s:=s+a[i];
```

Сумма элементов массива, удовлетворяющих какому-либо критерию

Отличие этого алгоритма от предыдущего в том, что происходит суммирование не всех элементов массива, а только тех, что удовлетворяют определённому условию. Например, элемент положительный (больше нуля), отрицательный (меньше нуля) и т.п. При программировании этого алгоритма необходимо в алгоритм суммы добавить оператор условного перехода. Алгоритм выглядит следующим образом (рисунок17).

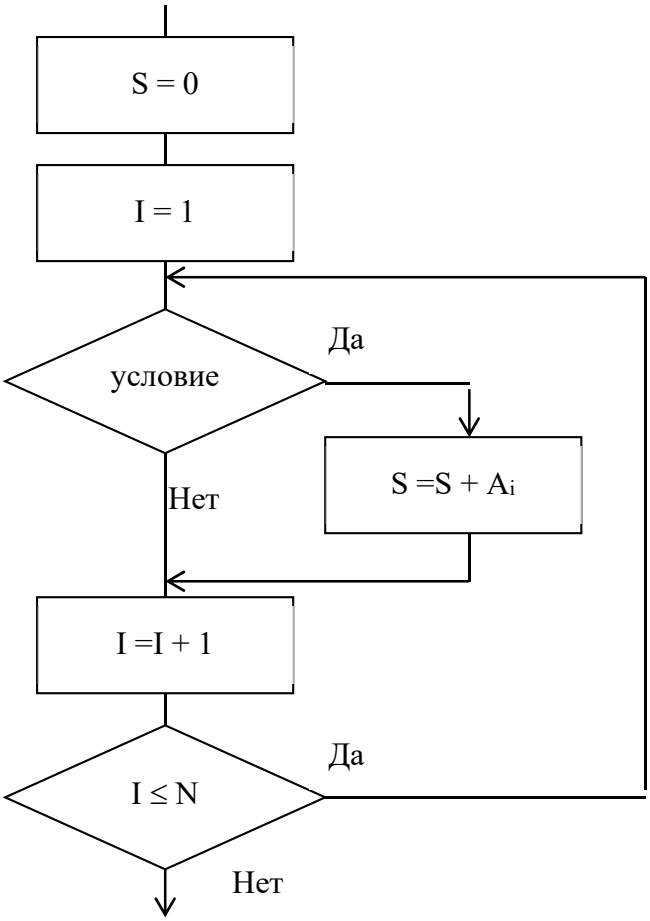


Рисунок 17 – Алгоритм определения суммы элементов массива, удовлетворяющих заданному критерию

Элемент программы имеет вид:

```
s:=0;
For i:=1 to n do
    If  <условие>  Then s:=s+a[i];
```

Произведение элементов массива

Математическая запись:

$$P = A_1 \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_n = \prod_{i=1}^{n} A_i \ .$$

Алгоритм произведения аналогичен алгоритму суммы. Так же добавляется переменная (P) в которой происходит накопление произведения элементов массива. Переменная P должна иметь тот же тип что и массив. Основное отличие алгоритма вычисления произведения от алгоритма нахождения суммы, в том, что переменную P перед использованием не обнуляют, а присваивается значение 1.

В нашем примере в раздел описания переменных необходимо добавить следующую строку:

P :Real;

Алгоритм произведения представлен на рисунке 18.

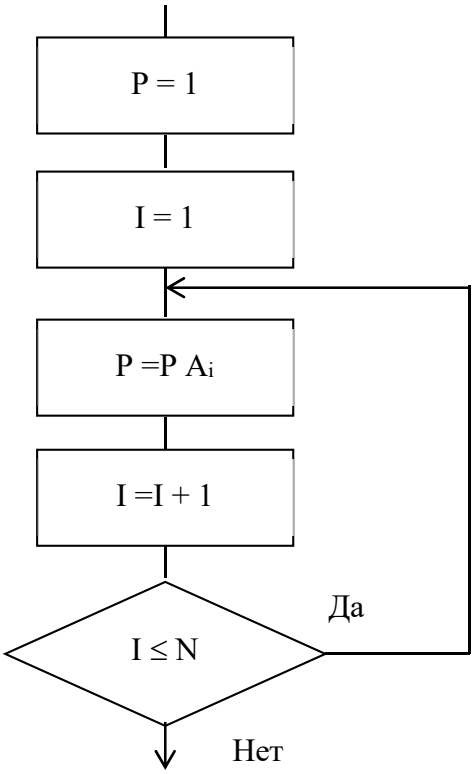


Рисунок 18 – Алгоритм определения произведения всех элементов массива

Элемент программы имеет вид:

```
p:=1;
For i:=1 to n do
p:=p*A[i];
```

Произведение элементов массива, удовлетворяющих какому-либо критерию

Так же как и в случае определения суммы элементов по условию, в этом алгоритме происходит произведение не всех элементов массива, а только тех, что удовлетворяют определённому условию. При программировании этого

алгоритма необходимо в алгоритм произведения добавить оператор условного перехода. Алгоритм выглядит следующим образом (рисунок 19).

Элемент программы выглядит следующим образом:

```
p:=1;
For i:=1 to n do
      If <условие> Then p:=p*a[i];
```

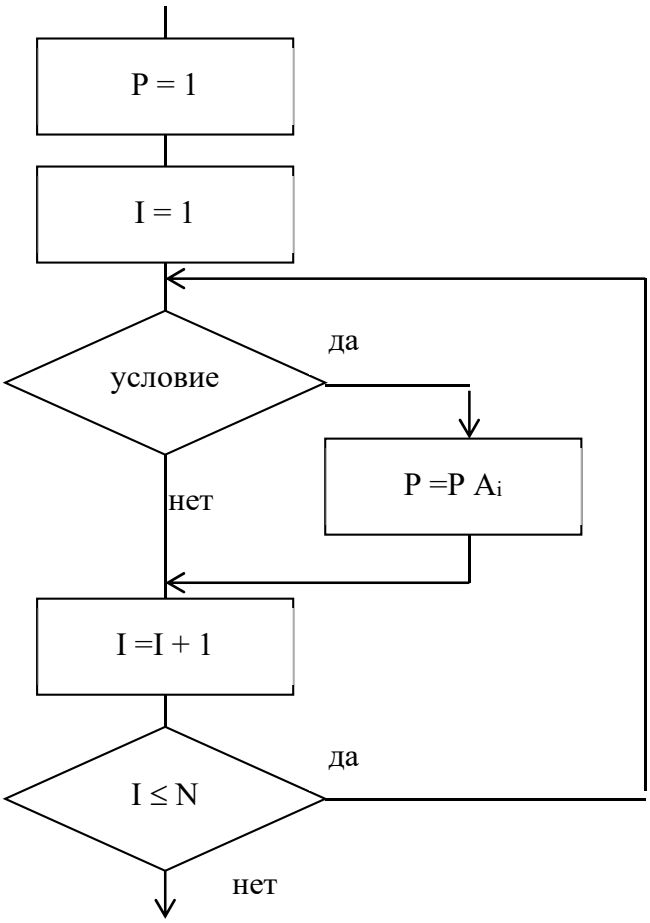


Рисунок 19 – Алгоритм определения произведения элементов массива удовлетворяющих заданному критерию

Нахождение количества элементов массива, отвечающих заданному критерию

Данный алгоритм отличается от ранее рассмотренных (сумма и произведение) тем, что при выполнении условия происходит не накапливание текущего элемента в переменной (K), а увеличение этой переменной на единицу (т.е. счётчик).

Переменная (K), в которой происходит увеличение счетчика на единицу, имеет целочисленный тип. В нашем примере в раздел описания переменных необходимо добавить следующую строку:

K :Integer;

Алгоритм представлен на рисунке 20.

Элемент программы выглядит следующим образом:

```
k:=0;
For i:=1 to n do
      If <условие>  Then k:=k+1;
```

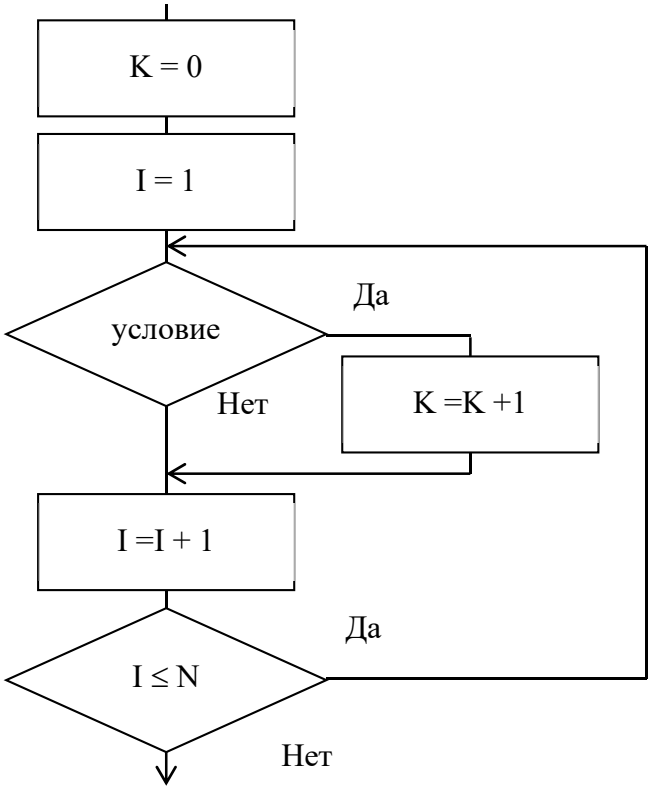


Рисунок 20 – Алгоритм определения количества элементов массива, удовлетворяющих заданному критерию

Необходимо отметить, что данный алгоритм всегда используется для определения количества элементов по заданному условию. Использование алгоритма без условия приведет к тому, что элемент K будет равен общему количеству элементов в массиве N.

Нахождение минимального (максимального) элемента массива

Нахождение минимального (максимального) значения осуществляется последовательным перебором и сравнением всех элементов массива.

В общем случае для нахождения минимума (максимума) задаются начальным значением, равному какому либо элементу массива (или заведомо малой (большой) величине). Как правило, за начальное значение берется первый элемент массива, т.е. принимается, что **MIN=A[1]** или **MAX=A[1]**. Затем последовательно проверяется, так ли это, т.е. проверяется условие:

На максимум **A[i] > MAX**

На минимум **A[i] < MIN.**

Если условие не выполняется, то в переменной MAX или MIN заносится текущий элемент массива.

Переменная MAX (MIN) должна иметь тот же тип что и массив.

В нашем примере в раздел описания переменных необходимо добавить следующую строку:

MAX : Real;или

MIN : Real;

Алгоритм нахождения минимума представлен на рисунке 21.

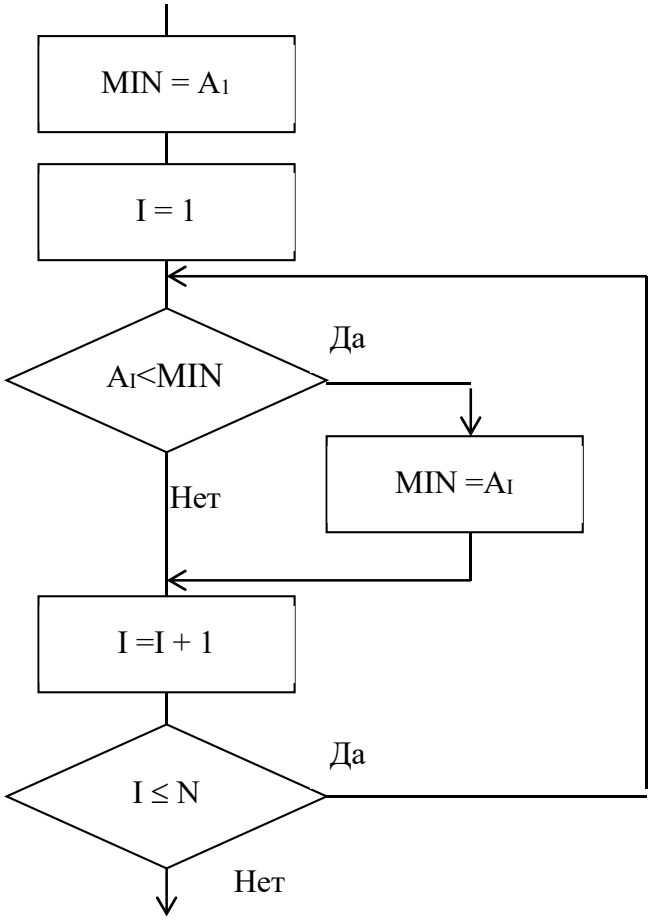


Рисунок 21 – Алгоритм нахождения минимального значения массива

Элемент программы:

```
min:=a[1];
For i:=1 to n do
      If a[i]<min Then min:=a[i];
```

Как уже отмечалось ранее, алгоритм нахождения максимума аналогичен алгоритму нахождения минимального значения. В алгоритм и текст программы необходимо вместо переменной MIN вписать переменную MAX, и заменить условие. Алгоритм и элемент программы (рисунок 22).

```
max:=a[1];
Fori:=1 tondo
If a[i]>max Then max:=a[i];
```

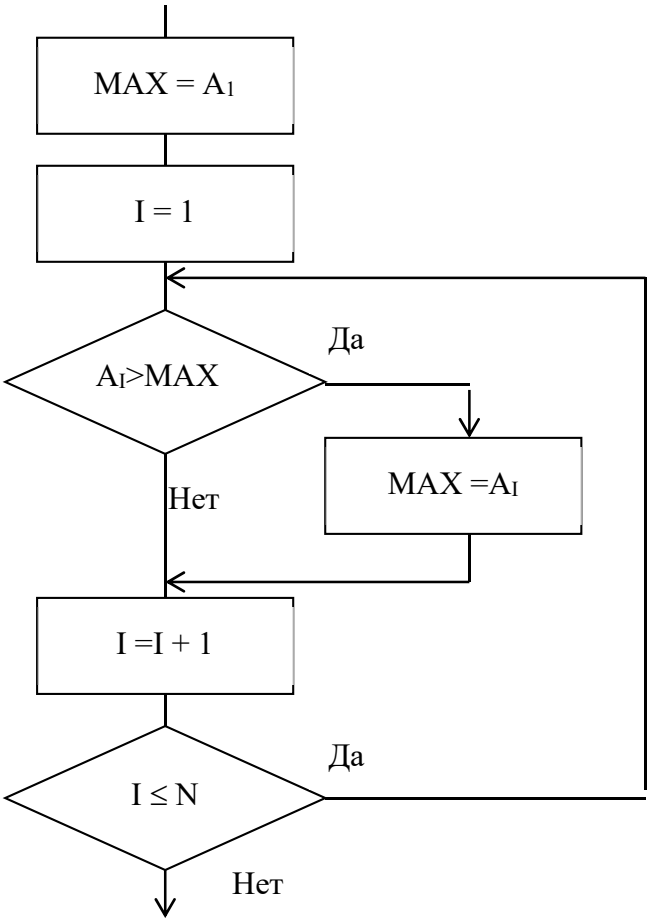


Рисунок 22 – Алгоритм нахождения максимального значения массива

Нахождение порядкового номера элемента в массиве, отвечающего какому-либо признаку

Алгоритм аналогичен алгоритму нахождения количества элементов массива, удовлетворяющих заданному условию. В случае если условие выполняется, то в переменной фиксируется индекс элемента. Переменная

носит целочисленный тип. В нашем примере в раздел описания переменных необходимо добавить следующую строку:

II :Integer;

Элемент программы выглядит следующим образом:

```
ii:=1;
For i:=1 to n do
   If <условие>  Then ii:=i;
```

Алгоритм выглядит следующим образом (рисунок 23).

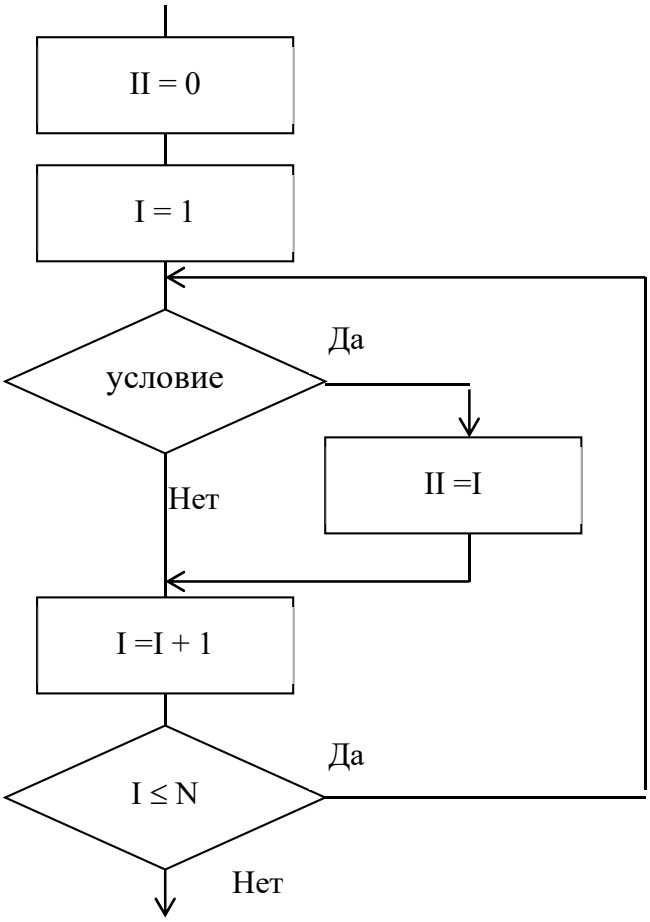


Рисунок 23 – Алгоритм нахождение порядкового номера элемента в массиве, отвечающего какому-либо признаку

Среднее значение всех элементов массива

Для нахождения среднего значения одномерного массива необходимо просуммировать все значения массива и разделить найденную сумму на количество элементов:

$$\overline{X} = \frac{\sum_{i=1}^{n} A_i}{n}.$$

Таким образом, для нахождения среднего значения массива нам необходимо воспользоваться алгоритмом нахождения суммы и дополнить его делением найденной суммы на количество элементов.

В раздел описания необходимо добавить вещественную переменную **SR :REAL;**

Алгоритм нахождения среднего значения одномерного массива представлен на рисунке 24.

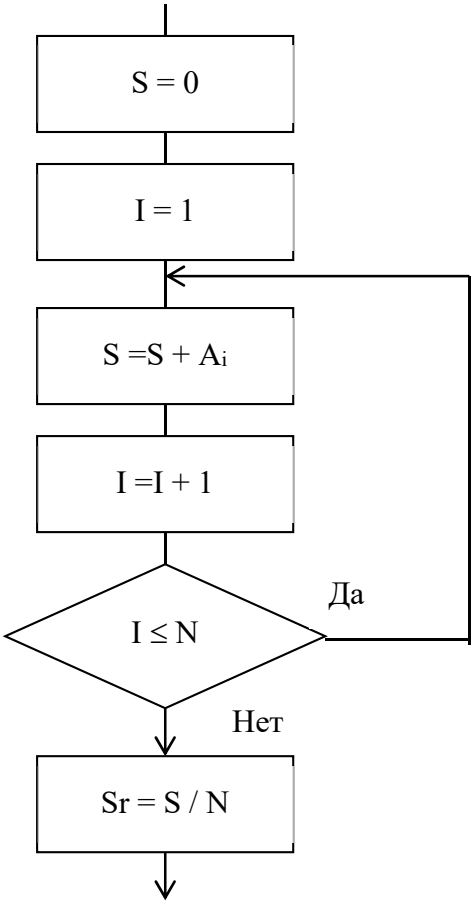


Рисунок 24 – Алгоритм нахождения среднего значения одномерного массива

Элемент программы выглядит следующим образом:

```
S:=0;
for i:=1 to n do
        S:=S + A[I];
Sr :=S / N;
```

Среднее значение элементов массива, удовлетворяющих заданному критерию

Суть данного алгоритма практически полностью аналогична предыдущему. Отличия заключаются в том, что необходимо найти сумму элементов которые удовлетворяют заданному критерию и определить количество этих элементов. Т.е. используются алгоритмы нахождения суммы по критерию и алгоритм нахождения количества элементов. Алгоритм представлен на рисунке 25.

Элемент программы выглядит следующим образом:

```
S:=0;
K:=0;
For i:=1 to n do
    If<условие>Then Begin
        S:= S + A[I];
        K:=K+1;
    End;
Sr :=S / K;
```

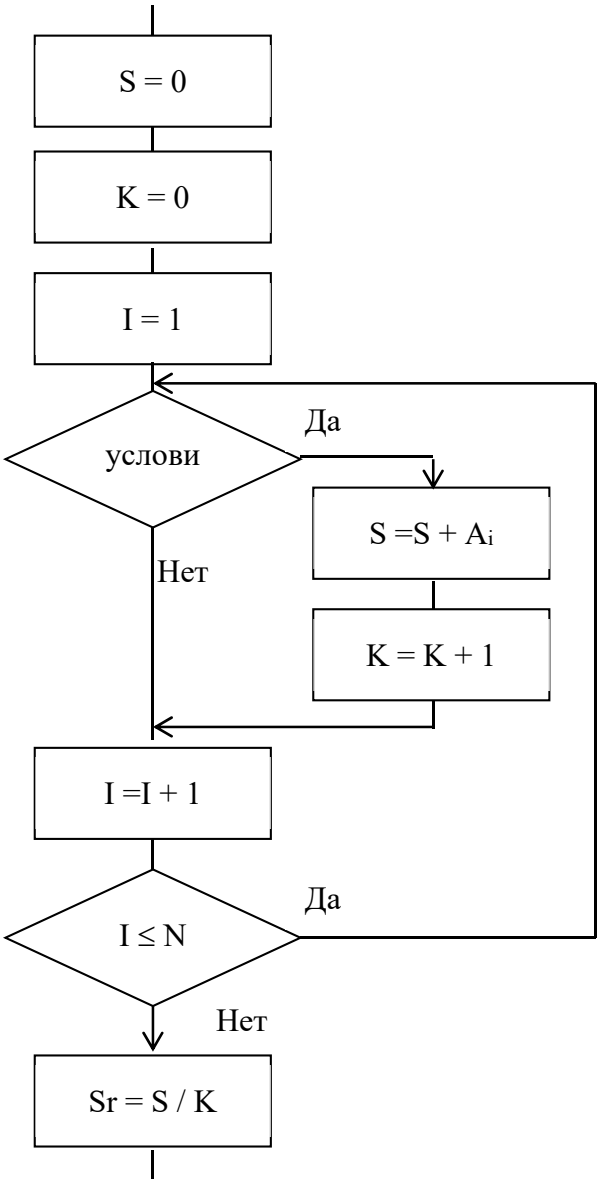


Рисунок 25– Алгоритм нахождения среднего значения одномерного массива по заданному критерию

6.6 Разработка алгоритма обработки одномерного массива

Общий алгоритм решения задач с одномерными массивами

1. Определение условия задачи.
2. Определение переменных задачи.
3. Ручной расчёт задачи.
4. Составление блок-схемы.
5. Составление паскаль - программы.

Задача

№ 1.Дан массив *A(N)*. Найти среднее значение всех элементов данного массива.

№ 2. Дан массив *B(N).* Определить количество нулевых элементов.

№ 3. Дан массив *C(N).* Найти произведение элементов, значение которых не меньше 10, но не более 30.

№ 4. Дан массив *B(N)*. Найти произведение положительных элементов, сумму отрицательных элементов и количество нулевых.

№ 5. Дан массив *A(N)*. Найти среднее значение положительных элементов.

№ 6. Дан массив *P(N)*. Посчитать число элементов, значения которых меньше среднего значения.

№ 7. Дан массив *V(N)*. Вычислить отношение минимального элемента заданного массива к его максимальному элементу.

№ 8. Дан массив *R(N)*. Сформировать новый массив *Q(N)*путём замены положительных элементов массива R(N) на 5.

№ 9. Дан массив *A(N)*. Найти среднее значение всех элементов и образовать новый массив *B(N)* путём замены всех элементов значения которых, больше среднего на ноль. В новом массиве *B(N)* найти количество нулевых элементов.

№ 10. Дан массив *L(N)*. Заменить отрицательные элементы на значение максимального, среди его положительных элементов и определить количество выполняемых замен.

№ 11. Дан массив *B(N)*. Определить сумму первого и последнего элементов. Разделить положительные элементы массива *B(N)* на эту сумму.

№ 12. Дан массив *C(N).* Сформировать новый массив*D(N)* путём умножения положительных элементов массива *C(N)* на 15, а отрицательных на 9. В массиве*D(N)* определить количество элементов, значения которых больше 150.

№ 13. Даны массивы *P(N)* и*R(N).* Образовать новый массив *Q(N)* путём попарного умножения элементов массивов *P(N)* и *R(N)*. В массиве *Q(N)* определить число положительных элементов.

№ 14. Дан массив *V(N*). Разделить все его положительные элементы на среднее значение элементов массива. В измененном массиве определить произведение положительных элементов.

№ 15.В заданном массиве *P(N)*найти сумму и количество элементов, значения которых равны первому элементу. Заменить положительные элементы

массива *P(N)*на найденную сумму.

№ 16. Даны массивы *C(N)*и *D(N)*. Сформировать массив*A(N)*путём прибавления к элементам массива *D(N)* суммы положительных элементов массива *C(N).*

№ 17. Дан массив *B(N)*. Сформировать новый массив *S(N)* путём замены отрицательных элементов массива *B(N)* на их квадраты. В массиве *S(N)* найти максимальный элемент.

№ 18. Разделить все элементы заданного массива *D(M),*значения которых больше 20, на минимальный элемент заданного массива *C(M)*.

№ 19. Даны массивы *Z(N)* и *X(N)*. Вычесть из элементов массива *Z(N)* среднее значение элементов массива *X(N)*, а к элементам массива *X(N)*прибавить максимальный элемент массива *Z(N).*

№ 20. Даны массивы *A(N)* и *B(N)*. Если количество положительных элементов в массиве *A(N)* больше количества нулевых элементов в массиве *B(N)*, то положительные элементы в обоих массивах заменить на их квадраты.

№ 21. Дан массив*C(N).* Заменить отрицательные элементы заданного массива на ноль, а положительные элементы – возвести в квадрат.

№ 22. Сформировать массив *S(N)*из заданного массива *D(N)*путём замены отрицательных элементов на их модули, а положительных – на их натуральные логарифмы.

№ 23. В заданном массиве *S(L)*определить количество элементов, значения которых не меньше 10, но не больше 20. Если это число больше 3, то все элементы, попадающие в заданный интервал, заменить на 0.

№ 24. Даны массивы *A(N)* и *B(N)*. Образовать новый массив *C(N)*путём прибавления к элементам массива *A(N)* среднего значения массива *B(N)*. В новом массиве найти количество элементов, равных 10.

№ 25. Даны массивы *A(N*) и *B(N).* Вычесть из элементов массива *A(N)* произведение максимального и минимального элементов массива *B(N).*

№ 26. Дан массив *D(M).* Определить максимальный элемент, значение которого не меньше -5, но не больше 10 . Изменить массив, увеличивая элементы на сумму отрицательных элементов.

№ 27. Дан массив *K(N).* Определить отношение наибольшего элемента к наименьшему элементу. Изменить массив, уменьшая элементы в диапазоне от 10 до 30 на найденное отношение.

Вопросы для самоконтроля

1. Понятие одномерного массива.
2. Описание одномерного массива.
3. Порядок индексации.
4. Обработка одномерного массива.
5. Характерные приёмы программирования задач с одномерными массивами (привести примеры).
6. Алгоритм решения задачи с одномерными массивами.

ТЕМА 7. ПРОГРАММИРОВАНИЕ ЗАДАЧ ИЗ ДАННЫХ КОМБИНИРОВАННОГО ТИПА

7.1 Понятие данных комбинированного типа

Как мы уже выяснили в курсе информатики, массивы объединяют однородные единицы информации – элементы одного и того же типа. Но многообразие информации нельзя свести только к какому-то одному типу данных. Например, описывая человека, мы должны указать его имя, рост, цвет глаз и волос, то есть в одном описании объединим **разнородную** информацию. Точно так же, описывая автомобиль, мы укажем не только его марку, но и год выпуска, модификацию, да и цвет кузова может нас заинтересовать. Составляя автоматизированный каталог книгохранилища, мы для каждой книги должны указать её название, имя автора, область знания, количество страниц, год издания, а также, возможно, признак нахождения на руках или в хранилище.

Так возникла необходимость в инструментарии описания таких структурных конструкций данных. **Под структурой данных** обычно понимают данные, объединённые в упорядоченное множество. В языке Паскаль для использования разнородной, но логически связанной информации предусмотрен механизм создания **комбинированных типов данных,** основу которых составляют стандартные данные (целые, вещественные и строковые).

Данное комбинированного типа состоит из нескольких данных стандартного типа, но в отличие от массива эти данные могут иметь разные типы и доступ к ним осуществляется не по индексам (как у массива), а по именам.

Данное комбинированного типа обычно называют **записью**, а компоненты в него входящими **полями**.

7.2 Описание данных комбинированного типа

Для создания комбинированного данного используется специальная конструкция, которая начинается служебным словом type (тип) и в программе всегда предшествует разделу описания.

```
type имя типа =RECORD
имя поле 1: стандартный тип,
имя поле 2: стандартный тип,
……
END;
Var имя записи: имя типа;
```

Например, необходимо создать запись, поля которой содержат данные о сотруднике предприятия: фамилия, должность, стаж.

```
Type st = record
fam: string [15];
dolg: string [10];
stag: integer;
end;
Var sved: st;
```

st– имя типа.

sved – имя записи, которое содержит поле: fam,dolg, stag.

fam, dolg, stag – поля, входящие в комбинированное данное.

fam и dolg – имеют строковый тип (string),а в квадратных скобках указывается размер, который определяет длину данного поля.

Чаще всего записи используются при формировании массивов из данных комбинированного типа, тогда раздел описание имеет вид:

```
Var имя записи: array [1..50] of имя типа;
```

Например:

```
Var sved: array [1..50] of st;
```

При работе с данными комбинированного типа выполняется работа с полями в него входящими. Чтобы обратиться к конкретному полю, надо сначала указать имя записи, в которое оно входит, поставить точку, а затем указать имя поля.

```
имя записи.имя поля
```

Например

```
sved. fam, sved. dolg, sved. stag.
```

Если запись представляет собой массив, то составное имя имеет вид:

```
Имязаписи[i].имя поля
```

Например:

```
sved[i]. fam, sved[i]. dolg, sved[i]. stag.
```

7.3 Оператор присоединения

Для удобства обращения к полям записи в Паскале используется оператор **присоединения**.

```
With имя записи do
begin
операторы содержащие имена элементов (полей) записи.
end;
```

где with – зарезервированное слово «с»;

do – зарезервированное слово «делать».

Оператор присоединения помогает существенно упростить работу с элементарными данными, входящими в состав комбинированного.

Например:

1) ввод данных комбинированного типа без оператора присоединения read (sved. fam, sved. dolg, sved. stag);

2) ввод данных комбинированного типа с оператором присоединения with sved do read (fam, dolg, stag);

Если запись представляет собой массив, то ввод данных выглядит следующим образом:

1) read (sved[i]. fam, sved[i]. dolg, sved[i]. stag);
2) withsved[i] do read (fam, dolg, stag);

7.4 Ввод данных комбинированного типа

Ввод данных комбинированного типа осуществляется двумя способами:

1) по записям;
2) по полям.

1. Ввод данных по записям:

При вводе данных по записям все поля одной записи набираются в одну строку, с учётом размеров символьных полей и правил ввода числовых данных.

1) Размер данных строкового типа должен соответствовать описанию в разделе типов. Выравнивание до требуемого размера выполняется с помощью пробелов;

2) Строковое данное от строкового данного при вводе ничем не отделяется;

3) Числовое данное от строкового данного при вводе ничем не отделяется.

4) Числовое данное от числового данного отделяется одним пробелом.

2. Ввод данных комбинированного типа по полям:

При вводе данных по полям каждое поле отдельной записи набирается в отдельной строке. Такой ввод данных не требует выравнивания до размера указанного в описании.

Задача 1. Имеются сведения о студентах: шифр группы, Ф.И.О., пол, возраст, размер стипендии. Определить среднюю стипендию в группе М-1-2.

Контрольный пример

1.Исходные данные лучше представить в виде таблицы. Число столбцов определяется из условия задачи, а число строк задаётся произвольно.

Шифр группы	Ф.И.О.	Пол	Возраст	Размер стипендии
М-1-2	Иванов С.В.	м	17	1400,00
М-1-3	Петров И.И.	м	18	0,0
М-1-2	Сидорова И.С.	ж	17	2000,00
ИМ-1-10	Миронова А.А.	ж	16	2500,00
ИМ-1-2	Кузнецов О.А.	м	17	1400,00

Для того чтобы определить среднюю стипендию в группе М-1-2надо общую стипендию этой группы разделить на количество студентов этой группы.

Введём обозначения :S– общая стипендия группы М-1-2;

K– количество студентов группы М-1-2;

SR– средняя стипендия группы М-1-2.

S=1400+2000=3400

K=2

SR=S/K=3400/2=1700

2. Для облегчения составления программы можно составить таблицу описания полей.

	Имя поля	Тип поля	Длина поля
Шифр группы	G	STRING	7
Ф.И.О.	FIO	STRING	15
Пол	P	STRING	1
Возраст	V	INTEGER	-
Размер стипендии	RS	REAL	-

Имя поля – задаётся согласно правилам, которые предъявляются к именам в языке Pascal.

Тип поля – если над полем производится математические операции, то эти поля должны иметь *числовой* тип (REAL, INTEGER), в остальных случаях – символьный тип (STRING).

Длина поля — это максимальное число символов в данном поле из всех записей Длина поля определяется только для символьных полей.

3. Pascal программа.

```
PROGRAM Z1;
TYPE SVED=RECORD
          G:STRING[7];
        FIO:STRING[15];
          P:STRING[1]
          V:INTEGER;
        RS:REAL;
     END;
 VAR A:ARRAY[1..100] OF SVED;
     S,SR:REAL;N,K,I:INTEGER;
```

```
BEGIN
        WRITELN('ВВЕДИТЕ ЧИСЛО ЗАПИСЕЙ');
        READLN(N);
        S:=0;K:=0;
        WRITELN('ВВЕДИТЕ G:7,FIO:15,P:1,V,RS');
        FOR I:=1 TO N DO WITH A[I] DO
        BEGIN READLN(G,FIO,P,V,RS);
        IF G='M-1-2 ' THEN BEGIN
        S:=S+RS;
        K:=K+1;
        END;
        END;
        SR:=S/K;
        WRITELN('СРЕДНЯЯ СТИПЕНДИЯ В ГРУППЕ М-1-2 РАВНА
',SR:6:2);
END.
```

Пояснения к программе.

1. SVED – имя типа(задаётся), который представляет собой запись;
   A – одномерный массив элементами которого являются записи;
   N – размер массива или число записей.
2. При вводе данных комбинированного типа лучше использовать оператор ввода READLN.
3. Работая с данными комбинированного типа, надо помнить о правильном обращении к отдельным полям записи. В данной программе используется оператор присоединения WITH.
4. При работе с символьными полями надо помнить о длине поля.
5. Работая с записями вывод, сопровождать смысловыми пояснениями.
6. В данной программе используется ввод данных по записям, т.е. все поля одной записи набираются в одну строку с учётом размеров полей.

Например. Исходные данные этой программы при наборе будут иметь вид:

```
ВВЕДИТЕ ЧИСЛО ЗАПИСЕЙ
5
М-1-2  ИВАНОВ С.В.    М17 1400
М-1-3  ПЕТРОВ И.И.    М18 0
М-1-2  СИДОРОВА И.С.  Ж17 2000
ИМ-1-10МИРОНОВА А.А.  Ж16 2500
ИМ-1-2 КУЗНЕЦОВ О.А.  М17 1400
```

Как видно из этого примера, данные имеющие символьный тип выравниваются до указанного в описании размера с помощью пробелов.

Задача 2.Имеются сведения о студентах шифр группы, Ф.И.О., возраст, пол, размер стипендии. Определить студента, получающего максимальную стипендию.

Контрольный пример

1. Используя данные задачи 1 определяем, что максимальная стипендия составляет 2500 рублей и получает её Миронова.

2. Pascalпрограмма.

```
PROGRAMZ2;
TYPESVED=RECORD
              G:STRING[7];
            FIO:STRING[15];
              P:STRING[1]
              V:INTEGER;
            RS:REAL;
         END;
   VAR A:ARRAY[1..100] OF SVED;
       MAX:REAL;N,I:INTEGER;
   BEGIN
            WRITELN('ВВЕДИТЕ ЧИСЛО ЗАПИСЕЙ');
            READLN(N);
            WRITELN('ВВЕДИТЕ G:7,FIO:15,P:1,V,RS');
            FOR I:=1 TO N DO WITH A[I] DO
                 READLN(G,FIO,P,V,RS);
                   MAX:=A[1].RS; PN:=1;
            FOR I:=1 TO N DO WITH A[I] DO
                 IF RS>MAX THEN BEGIN MAX:=RS;
                                      PN:=I;
                                 END;
            WITH A[PN] DO
WRITELN('МАКСИМАЛЬНАЯ СТИПЕНДИЯ СОСТАВЛЯЕТ  ',
MAX:6:2,'И ЕЁ ПОЛУЧАЕТ СТУДЕНТ ГРУППЫ ',G:7,' ',FIO);
   END.
```

Пояснения к программе.

1. Описание записи совпадает с описанием задачи 1.
2. MAX – максимальный размер стипендии;
   PN – номер максимальной стипендии.

7.5 Разработка программ обработки массивов из данных комбинированного типа

Общий алгоритм решения задач обработки массивов из данных комбинированного типа

1.Разработка таблицы исходных данных.

2. Составление паскаль - программы.

3.Результат выполнения программы на компьютере.

Задача

№1. Имеются сведения о студентах: шифр, Ф.И.О., область, город, улица и номер дома, возраст, размер стипендии. Определить количество студентов, проживающих в Ростовской области и выдать информацию об этих студентах. Вычислить среднюю стипендию группы М-1-3.

№2. Имеются сведения о сотрудниках: ФИО, шифр подразделения, должность, пол, стаж, оклад. Определить количество работающих женщин со стажем не менее 15 лет и вывести их на печать.

№3. Имеется каталог подписных изданий: название газеты или журнала, индекс, ФИО подписчика, домашний адрес, дата подписки, подписная цена. Определить количество изданий с ценой больше средней. Вычислить число подписок на журнал «Здоровье».

№4. Имеются сведения о призывниках на службу в армию: Ф.И.О., адрес, образование, год рождения, рост. Выдать информацию о призывниках, имеющих рост выше 180 см. Определить количество призывников с высшим образованием.

№5. Имеются сведения о товарах: наименование, страна-изготовитель, срок годности, единицы измерения, количество, цена за единицу, стоимость партии. Определить суммарную стоимость всех товаров и вывести на экран сведения о всех товарах отечественного производителя.

№6. Имеются сведения о библиотечном фонде вуза: автор, шифр книги (01 –художественная,02–техническая,03 – научно-популярная), название книги, год издания, стоимость одного экземпляра, количество экземпляров. Определить количество художественной, технической и научно-популярной литературы. Вычислить стоимость всего библиотечного фонда.

№7. Автотранспортное предприятие имеет парк машин. Каждая характеризуется следующими показателями: марка машины, балансовая стоимость, число лет эксплуатации, пробег машины, количество ремонтов. Определить марку машин с максимальным пробегом. Определить общее количество ремонтов для машин «Жигули».

№8. Имеются сведения о врачах поликлиники: ФИО врача, учёная степень (д.м.н., к.м.н., или «–» (без степени)), специализация (терапевт, окулист, кардиолог, хирург, ревматолог), загруженность. Выдать информацию о врачах с максимальной загруженностью. Определить количество кардиологов, имеющих степень к.м.н.

№9. Имеются сведения о туристических поездках: номер тура, страна, количество дней пребывания, вид транспорта, стоимость путевки.
Определить тур с минимальной стоимостью.
Вычислить количество путевок в Турцию.

Вопросы для самоконтроля

1. Понятие данных комбинированного типа.
2. Описание данных комбинированного типа.
3. Оператор присоединения.
4. Ввод данных комбинированного типа.
5. Алгоритм решения задач массивов из данных комбинированного типа.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Советов, Б.Я. Информационные технологии : учебник для бакалавров по направлению подготовки "Информатика и выч.техника" и "Информ. системы" / Б. Я. Советов, В. В. Цехановский. - 6-е изд. - Москва : Юрайт, 2013. - 263 с. - (Бакалавр. Базовый курс). - Гриф Мин. обр. - ISBN 978-5-9916-2824-2 : 228-00. - Текст : непосредственный.

2. Информационные технологии : учебник / Ю. Ю. Громов, И. В. Дидрих, О. Г. Иванова, М.А. Ивановский. - Тамбов : Изд-во ФГБОУ ВПО «ТГТУ», 2015. - 260 с. - Гриф УМО. - URL : https://biblioclub.ru/index.php?page=book&id=444641 (дата обращения: 25.04.2022). - ISBN 978-5-8265-1428-3. - Текст : электронный.

3. Федотова Е.Л. Информационные технологии и системы : учебное пособие для вузов по специальности 080801 "Прикладная информатика" и другим экономическим специальностям / Е. Л. Федотова. - Москва : ФОРУМ : ИНФРА-М, 2013. - 351 с. - (Высшее образование). - Гриф УМО. - ISBN 978-5-8199-0376-6 : 425-00. - Текст : непосредственный.

4. Хныкина, А. Г. Информационные технологии : учебное пособие / А. Г. Хныкина, Т. В. Минкина. - Ставрополь : СКФУ, 2017. - 126 с. : схем., ил. - URL : https://biblioclub.ru/index.php?page=book&id=494703 (дата обращения: 25.04.2022). - Текст : электронный.

5. Волкова, Т. И. Введение в программирование : учебное пособие / Т. И. Волкова. - Москва ; Берлин : Директ-Медиа, 2018. - 139 с. : ил., схем., табл. - URL : https://biblioclub.ru/index.php?page=book&id=493677 (дата обращения: 25.04.2022). - ISBN 978-5-4475-9723-8. - Текст : электронный.

6. Колокольникова, А. И. Спецразделы информатики: основы алгоритмизации и программирования : практикум / А. И. Колокольникова. - Москва ; Берлин : Директ-Медиа, 2019. - 424 с. : ил., табл. - URL : https://biblioclub.ru/index.php?page=book&id=560695 (дата обращения: 25.04.2022). - ISBN 978-5-4499-0097-5. - Текст : электронный.

7. Нагаева, И. А. Основы алгоритмизации и программирования: практикум : учебное пособие / И. А. Нагаева, И. А. Кузнецов. - Москва ; Берлин : Директ-Медиа, 2021. - 169 с. : схем. - URL : https://biblioclub.ru/index.php?page=book&id=598404 (дата обращения: 25.04.2022). - ISBN 978-5-4499-1612-9. - Текст : электронный.

8. Юрина, Т. А. Программирование и алгоритмизация : учебно-методическое пособие / Т. А. Юрина. - Омск : СибАДИ, 2021. - 88 с. - URL : https://e.lanbook.com/book/179228 (дата обращения: 25.04.2022). - Текст : электронный.

Учебное издание

Полубедова Галина Абрамовна

Введение в информационные технологии

Практикум

для студентов очной формы обучения по направлениям «Экономика», «Менеджмент», «Педагогическое образование»

Издается в авторской редакции

Подписано в печать Формат 60 × 84 1/16

Объём Тираж Заказ №

Отдел оперативной полиграфии НИМИ ФГБОУ ВО Донской ГАУ,
346428, г. Новочеркасск, ул. Пушкинская, 111