КОНСПЕКТ ЛЕКЦИЙ

по учебной дициплине

ОП.08 ОСНОВЫ ПРОГРАММИРОВАНИЯ

Разработчик:

Блохина Т.В.– преподаватель ГБПОУ РО «РКРИПТ».

Содержание

Раздел 1. Программирование на языке Паскаль 2

Тема 1.1 Понятие алгоритма, виды структур алгоритмов. 2

Тема 1.2 История и классификация языков программирования 5

Тема 1.3. Структура программы на Паскале 6

Тема 1.4.Типы данных 8

Тема 1.5. Программирование линейных задач 17

Тема 1.6. Управление символьным выводом 21

Тема 1.7. Использование средств графики в Паскале 24

Тема 1.8. Программирование циклических структур 29

Тема 1.9. Программирование разветвленных структур 37

Тема 1.10. Работа со структурированными данными 45

Тема 1.11. Типовые приемы решения задач 57

1. **Протокол выполнения программы** 64
2. **Протокол выполнения программы** 67

Тема 1.12. Работа со строковым типом данных 69

Тема 1.13. Использование множеств в программе 79

Тема 1.14. Использование подпрограмм в программе 85

Тема 1.15. Вычисление рекуррентных последовательностей 91

Тема 1.16. Работа с файлами в Паскале 97

Тема 1.17. Указатели и динамические структуры 125

Тема 1.18. Технология модульного программирования 130

https://studopedia.ru/5_40705_uslovniy-operator.html

Раздел 1. Программирование на языке Паскаль

Тема 1.1 Понятие алгоритма, виды структур алгоритмов.

Введение. Этапы разработки задач.

Определение алгоритма, виды структур алгоритмов.

Подготовка задачи к решению на ЭВМ требует последовательного поэтапного её рассмотрения с целью:

1) постановки и математической формализации;

2. разработки математической модели и выбора метода численного решения;
3. разработки алгоритма и определения структуры данных;
4. реализации алгоритма на языке программирования (составление программы), ввода программы;
5. отладки и тестирования программы;
6. оформления результата вычислений, составление программного продукта.

На первом этапе производится анализ формальной постановки задачи, и на его основе определяют количество и тип исходных данных для решения задачи, а также количество и тип результатов, которые должны быть получены при решении задачи. Всем данным дают имена, чтобы затем использовать их в алгоритме и программе.

Метод решения задачи определяется в зависимости от сложности задачи.

В математических и других расчетных задачах метод представляет собой совокупность формул, которые позволят из исходных данных получить результат.

Разработка алгоритма должна строиться по определенным правилам:

- использовать для всех данных имена, которые определили на этапе математической постановки задачи; если необходимо - ввести новые переменные для промежуточных результатов;
- использовать формулы, найденные для метода решения задачи;
- разбить метод решения задачи на этапы (команды исполнителю); каждый этап; должен содержать одно действие, понятное исполнителю;
- указать порядок следования этапов;
- указать признак окончания процесса решения задачи;

- в любом алгоритме надо указать, что является результатом решения задачи. предусмотреть выдачу результатов или сообщений об их отсутствии.

Четвертый этап заключается в том, что алгоритм переводится на конкретный язык программирования. При выборе языка программирования обычно руководствуются возможностями языка программирования и своими знаниями данного языка.

В процессе отладки устраняются синтаксические и семантические ошибки в программе. Синтаксические - это ошибки в инструкциях языка, семантические - это ошибки несоответствия типов данных и введенных в них значений. В процессе тестирования программу запускают на исполнение с допустимыми и недопустимыми исходными данными. При работе программы с допустимыми данными проверяют правильность выполнения контрольных примеров. Недопустимые исходные данные берутся для проверки реакции на них программы. Тестирование может выявить ошибки в программе, но не может гарантировать их отсутствие.

В состав программного продукта входят:

- формальная и математическая постановка задачи;
- метод решения задачи;
- описание алгоритма выполнения задачи;
- текст программы с комментариями;
- описание результатов тестирования;
- правила эксплуатации программы.

Понятие алгоритма

Одним из фундаментальных понятий в информатике является понятие алгоритма.

Алгоритм — это последовательность определенных действий со строгими правилами выполнения для получения нужного результата.

Свойства алгоритмов:
1. Дискретность (алгоритм должен состоять из конкретных действий, следующих в определенном порядке);
2. Детерминированность (любое действие должно быть строго и недвусмысленно определено в каждом случае);
3. Конечность (каждое действие и алгоритм в целом должны иметь возможность завершения);
4. Массовость (один и тот же алгоритм можно использовать с разными исходными данными);
5. Результативность (отсутствие ошибок, алгоритм должен приводить к правильному результату для всех допустимых входных значениях).

Виды алгоритмов:
1. Линейный алгоритм (описание действий, которые выполняются однократно в заданном порядке);
2. Циклический алгоритм (описание действий, которые должны повторятся указанное число раз или пока не выполнено задание);

3. Разветвляющий алгоритм (алгоритм, в котором в зависимости от условия выполняется либо одна, либо другая последовательность действий)
4. Вспомогательный алгоритм (алгоритм, который можно использовать в других алгоритмах, указав только его имя).

Циклическими называются алгоритмы, в которых часть этапов повторяется несколько раз.

Различают три вида циклов:

- **Цикл с предусловием («Пока»);**

На АЯ он имеет следующий вид

Пока условие, **повторять**

Нц

Серия

Кц

Выполнение серии команд (тела цикла) повторяется пока условие цикла истинно

- **Цикл с постусловием («До»);**

На АЯ он имеет следующий вид :

Повторять

Серия

До условие

Здесь используется условие окончания цикла, т.е. когда оно становится истинным, цикл заканчивает работу.

Цикл с предусловием (ПОКА) Цикл с постусловием (ДО)

Цикл с параметром.

содержит специальную переменную (***счетчик***, параметр цикла), которая последовательно изменяет свое значение от заданного ***начального*** до ***конечного значения*** с

некоторым ***шагом,*** при этом для каждого значения счетчика тело цикла исполняется один раз

Тема 1.2 История и классификация языков программирования

История развития языков программирования.

Классификация языков программирования: по уровням, специализации.

Язык программирования – это способ записи программ решения различных задач на ЭВМ в «понятной» для компьютера форме.

В развитии инструментального программного обеспечения (т.е. программного обеспечения, служащего для создания программных средств в любой проблемной области) рассматривают **пять поколений языков программирования (ЯП).**

ЯП *первого* поколения представляли собой набор машинных команд в двоичном (бинарном) или восьмеричном формате, который определялся архитектурой конкретной ЭВМ. Каждый тип ЭВМ имел свой ЯП, программы на котором были пригодны только для данного типа ЭВМ. Программирование в машинных адресах требует знания системы команд конкретной ЭВМ и их адресности. При этом реализация даже довольно несложных вычислений требует разложения их на простые операции, что значительно увеличивает общий объем программы и затрудняет ее чтение и отладку.

Второе **поколение *ЯП*** характеризуется созданием языков ассемблерного типа *(ассемблеров, макроассемблеров),* позволяющих вместо двоичных и других форматов машинных команд использовать их имена.При этом ассемблерные программы все так же затруднительны для чтения, трудоемки при отладке и требуют больших усилий для переноса на другие типы ЭВМ.

В эпоху ЭВМ т***ретьего*** **поколения ЯП** распространение получил язык PL/, разработанный фирмой IBM. Это был первый язык, преиендовавший на универсальность,т.е. на возможность решать любые задачи : вычислительные, обработки текстов, накопления и поиска информации. Однако язык PL/1 оказался слишком сложным языком. Для машин типа IBM 360/370 транслятор с языка PL/1 оказался недостаточно оптимальным и содержал ряд не выявленых ошибок.

Значительным событием в истории языков программирования стало создание в 1971г. Паскаля **как учебного языка** структурного программирования ***– языка четвертого поколения ЭВМ.***

Широкое распространение языку Паскаль обеспечили ПК. Фирма Borland (США) разработала для них систему программирования Турбо Паскаль. Турбо Паскаль – это не только язык и транслятор с него, но еще ***интегрированная среда программирования***, обеспечивающая пользователю удобство работы на языке Паскаль. В силу своих достоинств Паскаль стал источником многих современных языков программирования.

Язык программирования СИ создавался как инструментальный язык для разработки операционных систем, трансляторов, баз данных и других системных и прикладных программ. В отличие от Паскаля в нем заложены возможности непосредственного

обращения к некоторым машинным командам и определенным участкам памяти компьютера.

ЭВМ будущего – ***пятого*** – поколения называют машинами искусственного интеллекта.

В конце 20 века новым значительным направлением в развитии программного обеспечения ЭВМ стал **объектно-ориентированный подход**. **Объекты** – это структуры, объединяющие в единое целое данные и программы их обработки.

В последнее время возникло новое направление в технологии программирования – **визуальное**. Стали создаваться системы визуального программирования, использование которых позволяет легко и быстро программировать сложный графический интерфейс.

Тема 1.3. Структура программы на Паскале

Принципы структурного программирования.

Элементы языка Паскаль: алфавит, синтаксис, идентификаторы

Структура программы: заголовок, разделы описаний, раздел операторов.

Разделы описаний констант, типов, переменных.

В основе **структурного программирования** лежит **утверждение**:

Алгоритм для решения любой логической задачи можно составить только из структур Следование, Ветвление, Цикл, называемых **базовыми алгоритмическими структурами.**

Еще одним важнейшим технологическим приемом структурного программирования является **декомпозиция решаемой задачи на подзадачи**, т.е. более простые для программирования части исходной задачи.

Возможны **два подхода** к построению алгоритма :

- сверху вниз – сначала строится основной алгоритм, а затем – вспомогательные;

- снизу вверх – сначала составляются вспомогательные алгоритмы, а затем - основной

Соблюдение правил в языке программирования должно быть более строгим, чем в разговорном языке. Человеческая речь содержит значительное количество избыточной информации, т.е. не расслышав какое-то слово, можно понять смысл фразы в целом. Слушающий или читающий человек может додумать, дополнить, исправить ошибки в воспринимаемом тексте.

Компьютер же – это автомат, воспринимающий все «всерьез». В текстах программ нет избыточности, и компьютер сам не исправит даже очевидной (с точки зрения человека) ошибки. Он может лишь указать место, которое «не понял» и вывести замечаниео предполагаемом характере ошибки. Исправить же ошибку должен программист.

Язык Паскаль разрабатывался специально для целей обучения и с него рекомендуется начинать освоение программирования.

Запись программы на языке Паскаль похож на английский перевод алгоритма, записанного на алгоритмическом языке.

Структура программы на языке Паскаль. По определению стандартного языка Паскаль программа состоит из *заголовка* и *тела (блока)*, в конце которого следует ***точка*** – признак конца программы. В свою очередь, блок содержит *разделы описаний* и *раздел операторов*:

Program <имя программы**>;**

Label <раздел меток**>;**

Const <раздел констант**>;**

Type <раздел типов**>;**

Var <раздел переменных**>;**

Procedure (Function) <раздел подпрограмм**>;**

Begin

<раздел операторов>

End.

Заголовок программы начинается со слова **Program** и предпочтительней, чтобы оно отражало назначение программы, т.е. несло смысловую нагрузку.

Раздел описания переменных начинается со слова **Var**, за которым следует список переменных. **Тип** указывается после двоетчиясловом **Integer (целый)**. Начало и конец **раздела операторов** программы отмечаются словами **Begin (начало)** и **End (конец)**. ***В конце программы обязательно ставится точка***.

Необходимо строго соблюдать правила правописания – синтаксис программы. В частности, в Паскале однозначно определено назначеие знаков пунктуации :

- точка с запятой (;) ставится в конце заголовка программы, в конце раздела описания переменных, после каждого оператора. Перед словом End точку с запятой можно не ставить;

- запятая (,) является разделителем элементовво всевозможных списках (списке переменных в разделе описания, списке вводимых и выводимых величин).

Алфавит языка состоит из следующих символов:

• Заглавные и строчные латинские буквы и символ 'подчерк':

A, B, C..., X, Y, Z, a, b, c,.. , x, y, z, _

Буквы используются для формирования идентификаторов и служебных слов.

• Десять арабских цифр от 0 до 9:

0,1,2,3,4,5,6,7,8,9

Цифры используются для записи чисел и идентификаторов.

• Двадцать два специальных символа:

+ - * / = >< . , ; :

@ ' ( ) [ ] { } # $ ^

Идентификаторы **(имена)**. Идентификаторы вводятся для обозначения в программе переменных, констант, типов, меток, процедур и функций. Это любая последовательность букв и цифр, начинающаяся с бувы. Строчные и прописные буквы в идентификаторах и служебных словах **не различаются.**

Комментарии – это любая конструкция вида **{** любой текст **}** и **(***любой текст ***)**

Понятие **типов данных** является одним из центральных в любом языке программирования. С типом величины связаны **три ее свойства** :

1. Форма внутреннего представления
2. Множество принимаемых значений
3. Множество допустимых операций

Ниже показан **пример описания переменных**.

Var a : integer;

Index : 0..5;

S : char;

Sinus : real;

Semaphore : (Opened, Closed);

Каждая переменная, описанная в блоке, должна упоминаться в описаниях НЕ БОЛЕЕ ОДНОГО раза. Это относится не только к переменным, а вообще ко всем идентификаторам (констант, типов, меток, подпрограмм), которые объявляются в блоке.

Тема 1.4.Типы данных

Типы данных языка Паскаль: простые и структурированные.

Арифметические и логические выражения, правила их написания и вычисления.

Использование встроенных математических функций.

В языке Pascal любая переменная характеризуется своим *типом.* Под типом в данном случае понимается множество *значений,* которые может принимать переменная и, как следствие, множество *операций,* допустимых над данной переменной. Тип переменной определяется при ее описании и не может быть изменен. Переменная может участвовать только в операциях, допустимых ее типом.

Базовыми в системе типов являются ***простые*** типы. Составные типы по определенным правилам строятся из простых**.** ***Ограниченные*** **(или** ***интервальные***) типы формируются из простых типов путем сужения их области допустимых значений.

Рис. 2.2. Структура типов данных ТурбоПаскаля

Таблица 2.1

Типы данных ТурбоПаскаля

Идентификатор	Длина, байт	Диапазон (множество) значений
Целые		
Integer	2	–32768..32767
Byte	1	0..255
Word	2	0..65535
Shortint	1	–128..127
Longint	4	–2147483648..2147483647
Вещественные		
Real	6	$2{,}9 \cdot 10^{-39} — 1{,}7 \cdot 10^{38}$ (11—12)
Single	4	$1{,}5 \cdot 10^{-45} — 3{,}4 \cdot 10^{38}$ (7—8)
Double	8	$5 \cdot 10^{-324} — 1{,}7 \cdot 10^{308}$ (15—16)
Extended	10	$3{,}4 \cdot 10^{-4932} — 1{,}1 \cdot 10^{4932}$ (19—20)
Логический		
Boolean	1	True, False
Символьный		
Char	1	Все символы кода ASCII

Значения **целых** типов могут изображаться в программе двумя способами: в десятичном виде (традиционно, в виде последовательности цифр) и в шестнадцатеричном виде. Максимально допустимый диапазон изображений десятичного числа есть

-2147483648 .. +2147483747

Над **целыми значениями** допустимы следующие операции:

• четыре арифметических действия, имеющие привычный смысл. Эти действия обозначаются символами:

+ сложение;

- вычитание;

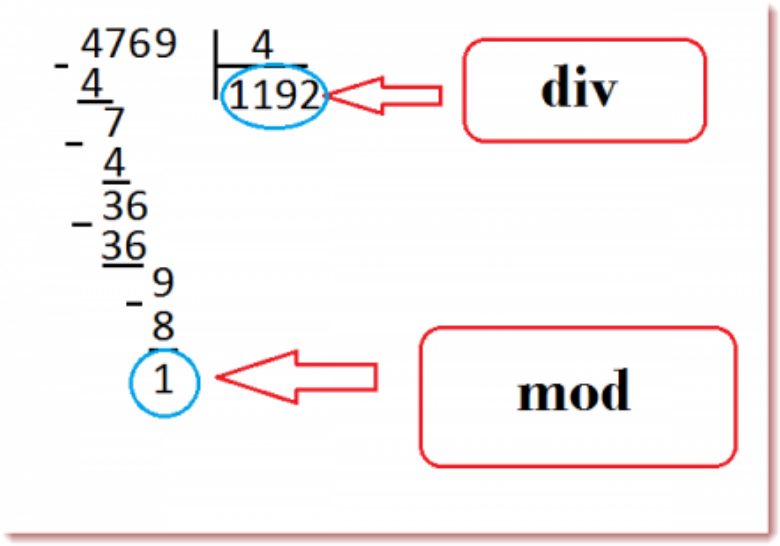
* умножение;

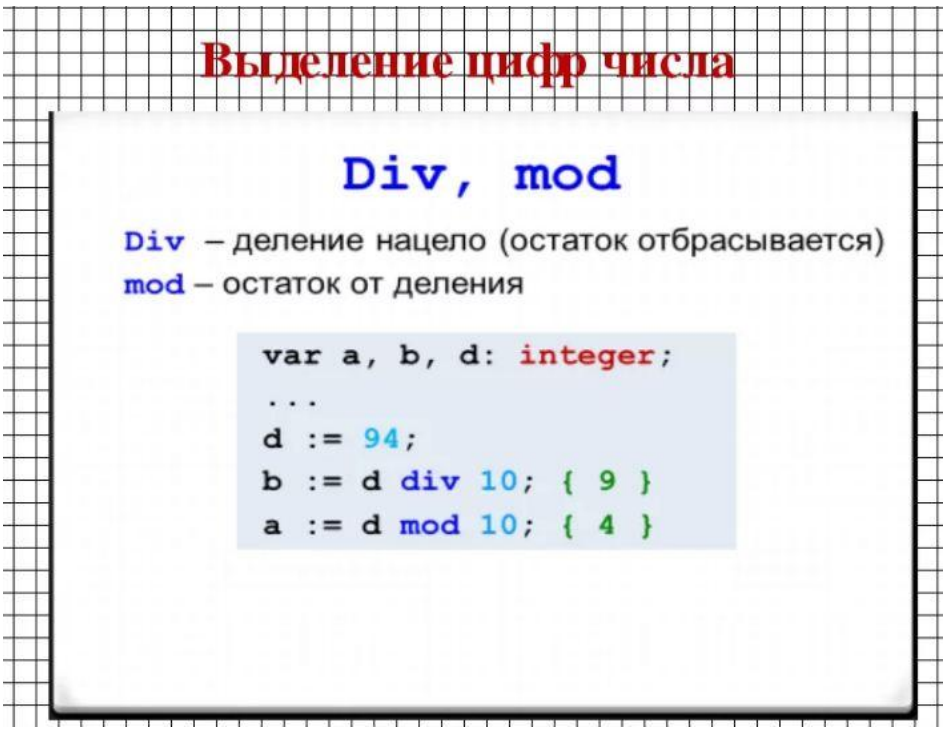
/ деление.

• две дополнительные операции "типа деления", обозначаемые служебными словами:

div деление нацело (с отбрасыванием дробной части)

mod взятие остатка от целочисленного деления (отбрасывается целая часть)





При применении к целым значениям все перечисленные операции (кроме деления) дают целый результат. Операция деления **всегда дает вещественный результат**.

Вещественные значения могут изображаться в форме с **фиксированной точкой** и в форме **с плавающей точкой**. В первом случае целая и дробная части вещественного числа отделяются друг от друга символом '.' (точка). Обе эти части должны обязательно присутствовать, например:

17.384 0.5

Вещественное число в форме с плавающей точкой записывается как пара вида

<мантисса> E<порядок>

Такое обозначение понимается как **"мантисса, умноженная на 10 в степени, равной порядку"**. Например, **7E-2 означает $7x10^{-2}$, 12.25E+6 или 12.25E6 оба обозначают $12.25x10^{+6}$**. Мантисса представляется в виде целого числа или как вещественное с фиксированной точкой; порядок обозначается целым числом; допускаются как положительные, так и отрицательные значения порядка. Над значениями **вещественных типов допустимы четыре арифметических операции (аналогично целым значениям). Все они дают вещественный результат, если хотя бы один операнд вещественный.**

Значениями **символьного типа** являются символы из множества ASCII. Это множество состоит из 256 различных символов, упорядоченных определенным образом, и содержит символы заглавных и строчных букв, цифр, и различных других символов, включая специальные управляющие символы. Служебное слово для описания символьного типа – **char**.

Символьное значение имеет графическое представление и оно изображается апострофом. **Над значениями символьного типа возможны операции сравнения.**

Логический тип.

Описывается служебным словом **Boolean.**

Имеется два значения булевского типа, представляющие логические истинностные значения (истина/ложь). Эти значения обозначаются посредством стандартных идентификаторов

true (истина) **false** (ложь)

Над значениями булевского типа допустимы операции сравнения, причем считается, что

false<true

Кроме того, имеются четыре стандартные логические операции, обозначаемые служебными словами:

not логическое отрицание (логическое "Не")

and логическое умножение (логическое "И").

Выражение ИСТИННО, если оба операнда ИСТИННЫ.

or логическое сложение (логическое "Или").

Выражение ЛОЖНО, если оба операнда ЛОЖНЫ.

Таблица истинности

А	Б	А и Б	А или Б	не А
И	И	И	И	Л
И	Л	Л	И	Л
Л	И	Л	И	И
Л	Л	Л	Л	И

Эти операции перечислены в порядке убывания приоритета, но все они имеют более высокий приоритет, чем операции сравнения с помощью знаков =, <=, <, >, >=, <>.

Интервальные типы

Самым простым способом образования новых типов из уже существующих является ограничение (а именно, сужение) допустимого диапазона значений некоторого стандартного скалярного типа. Это ограничение определяется заданием минимального и максимального значений диапазона, например:

1..10

-100..100

'a'..'z'

Перечислимые типы

Можно определить новый тип путем явного перечисления всех возможных его значений, причем каждое такое значение будет определяться только именем.

Примеры перечислимых типов:

(Red, Yellow, Green)

(Monday, Tuesday, Wednesday, Thursday,Friday, Saturday, Sunday)

(On, Off)

(Left, Up, Right, Down).

Стандартные функции.

Язык ТП 7.0 имеет набор стандартных функций, которые облегчают программирование (см. таблицу 2.3.4).

Для обращения к стандартной математической функции надо указать её имя и аргумент в круглых скобках. Аргумент может быть в виде значения, имени переменной или выражения. Независимо от вида, его тип должен соответствовать типу аргумента для данной функции

Стандартные математические функции TurboPascal 7.0

Abs(x)	Возвращает абсолютное значение числа X.
Cos(x)	Возвращает косинус числа x, где x – угол в радианах.
Sin(x)	Возвращает синус числа x, где x – угол в радианах.
Arctan(x)	Возвращает арктангенс числа x, где x – угол в радианах.
Exp(x)	Возвращает число, равное e в степени x.
Ln(x)	Возвращает число, равное натуральному логарифму от числа x.
Pi	Число Пи.
Sqr(x)	Возвращает число, равное квадрату числа X.
Sqrt(x)	Возвращает число, равное корню квадратному из X.
Trunc(x)	Возвращает число, равное целой части числа X (округление происходит путем отбрасывания дробной части числа X); усеченное число имеет тип LongInt.
Frac(x)	Возвращает число, равное дробной части числа X. Например, Frac(2.4) возвращает 0.4, Frac(3.99) возвращает 0.99.
Int(x)	Возвращает число, равное целой части числа X. Например, Int(2.4) возвращает 2.0, Int(0.99) возвращает 0.0.
Round(x)	Возвращает число, равное целой части числа X (округление происходит по всем правилам математики, т.е. к ближайшему целому). Процедура является универсальной, поскольку результат может быть как целым, так и вещественным числом.

Random(x)	Возвращает случайное число от 0 до X. Функция может использоваться без параметров. В этом случае будут генерироваться случайные числа от 0 до 1.
Ord(x)	Определяет четность числа. Возвращает значение True, если число X нечетно.
Inc(x,y)	Увеличивает значение числа X на Y. Процедура может быть задана без аргумента Y. В этом случае значение числа X будет увеличено на 1.
Dec(x,y)	Уменьшает значение числа X на Y. Процедура может быть задана без аргумента Y. В этом случае значение числа X будет уменьшено на 1.
Randomize	Обеспечивает несовпадение последовательностей случайных чисел, генерируемых функцией Random. Делает выбор случайных чисел еще более случайными.

Пример:

Программа нахождения среднего арифметического двух введенных целых чисел.

```
Program SrednArifm;

Var

X, Y: integer;

Sred: real;

Begin

Writeln('Введите числа X и Y');

Readln(x,y);

Sred:=(X+Y)/2;

Writeln('Среднее арифметическое чисел ', X, ' и ',Y,' равно ', Summa);

End.
```

В данном примере переменная Sred вещественного типа, поскольку в правой части присутствует операция деления.

Задачи

1. Ввести число купленных тетрадей и карандашей. Вычислить стоимость покупки, если цена одной тетради 5 рублей, а карандаша 2 рубля.

2. Ввести расстояния до дачи, количество бензина, которое автомашина потребляет в среднем на 100 км и стоимость бензина. Вычислить стоимость поездки (туда и обратно).

3. Ввести длину дистанции, на которую бежал бегун и его время. Вычислить среднюю скорость бегуна.

4. Ввести расстояние в километрах. Перевести расстояние в версты (1 верста – это 1066,8 м).

5. Ввести вес в килограммах. Перевести вес в фунты (1 фунт – это 405,9 граммов).

6. Ввести время, затраченное на выполнение ломашнего задания в часах и минутах. Перевести это время в минуты.

7. Ввести три целых числа. Найти их среднее геометрическое.

8. Ввести радиус окружности, подсчитать длину окружности.

9. Ввести радиус окружности, подсчитать площадь круга.

10. Ввести катеты a и b прямоугольного треугольника. Найти гипотенузу c. (Указание: использовать встроенную функцию вычисления квадратного корня из числа x - Sqrt(x)).

11. Ввести длины сторон произвольного треугольника и угол a между ними в градусах. Найти третью сторону c.

12. Ввести длины сторон произвольного треугольника a, b и c. Найти площадь треугольника.

13. Вычислить объем шара радиуса R. Длина R вводится с клавиатуры.

14. Ввести координаты двух точек на плоскости. Вычислить расстояние между ними.

15. Ввести длину грани куба. Вычислить его площадь боковой поверхности и объем.

16. Ввести коэффициенты и значения правых частей системы уравнений. Найти ее решение. Вывести на экран вид системы уравнений и ответ.

$$\begin{cases} a_1 x + b_1 y = c_1 \\ a_2 x + b_2 y = c_2 \end{cases}$$

17. Ввести число a. Для данного числа напечатать таблицу вида:

a^2

a^3 a^6

a^6 a^3 a.

12. Ввести целые числа x и y. Вычислить $\dfrac{|x - y|}{1 + |xy|}$.

13. Ввести длины катетов прямоугольного треугольника. Найти его гипотенузу и площадь.

14. Смешано V_1 литров воды температуры t_1 и V_2 литров воды температуры t_2. Найти объем и температуру образовавшейся смеси.

15. Найти площадь равнобедренной трапеции с основаниями a и b и углом при большем основании a. Значения a, b и a вводятся с клавиатуры.

16. Ввести длины сторон треугольника. Вычислить:

а) длины высот;

б) длины биссектрис;

в) длины медиан.

17. Ввести вещественные числа x и y. Вычислить расстояние от точки плоскости с координатами (x, y) до ближайшей границы квадрата с вершинами:

а) (-0,5; -0,5), (-0,5; 0,5), (0,5; 0,5), (0,5; -0,5);

б) (0;0), (0;1), (1;1), (1;0).

18. Ввести целые (либо вещественные) числа x_1, y_1, x_2, y_2, …, x_n, y_n. Выяснить, найдутся ли среди точек с координатами (x_1; y_1), (x_2; y_2), …, (x_n; y_n) четыре таких, которые являются вершинами квадрата.

В TurboPascal 7.0 **отсутствует функция для вычисления значения степени**. Так, для вычисления значения a^b можно применить формулу

$$\exp(b \bullet 1n(a)).$$

При отрицательном значении основания степени *a* вычисление невозможно!

Арифметические выражения.

Для верной записи арифметических выражений следует соблюдать определенные правила.

1. Все символы писать в строку, т.е. на одном уровне. Проставлять все знаки операций, не пропуская знак «*».
2. Не допускать записи двух знаков операций подряд, т.е. нельзя писать A +-B, следует писать

A + (-B).

3. Операции с более высоким приоритетом выполняют раньше операций с меньшим приоритетом.

Порядок убывания приоритетов операций следующий:

- вычисление функции;
- унарная операция смены знака (-);
- *, /, div, mod;
- **+, -**

4. Несколько записанных подряд операций с одинаковым приоритетом выполняют последовательно слева направо.
5. Часть выражения, заключенная в скобки, вычисляется в первую очередь. (Например, в выражении (A + B) * (C - D) умножение производится после сложения и вычитания.)
6. Не следует записывать выражения, не имеющие математического смысла, например: деление на нуль, логарифм отрицательного числа и т.п.

$$(1+y)\frac{2x+\sqrt{y}-(x+y)}{y+\frac{1}{x^2-4}}.$$

Приведем пример. Арифметическое выражение, записанное по указанным правилам (цифрами в кружке указан порядок выполнения операций),

Тема 1.5. Программирование линейных задач

Операции ввода и вывода информации.

Структура и использование оператора присваивания.

Форматирование при выводе данных на экран.

Линейные программы на языке Паскаль состоят из команд, которые выполняются в порядке их написания, поочередно.

Оператор присваивания :

В результате присваивания переменная получает конкретное значение.

Формат

имя переменной:=выражение;

Символ ":=" является оператором присваивания языка Паскаль.

Оператор присваивания выполняется следующим образом: сначала вычисляется значение выражения, находящегося справа от символа ":=", а затем полученное значение записывается в переменную, имя которой стоит слева от символа ":=".

Примеры: X:=A+0.5;

SUMMA:=X+N[1];

Линейная программа может содержать операторы присваивания, процедуры ввода переменных с клавиатуры и вывода.

Примеры:

X:=A+0.5;

Ввод данных — это передача информации от внешних устройств в оперативную память. Вводятся, как правило, исходные данные решаемой задачи.

Вывод данных — это передача данных из оперативной памяти на внешние носители (печать, дисплей, магнитные устройства и т.д.). Результаты решения любой задачи должны быть выведены.

Операторы ввода с клавиатуры :

Для ввода (чтения) данных используют операторы **read, readln («читай», «читай строку»).**

READ (R1,R2,..Rn);

READLN (R1,R2,..Rn);

В операторах **READ** (R1,R2,..Rn) и **READLN** (R1,R2,..Rn) в списке параметров должны стоять только имена переменных. Эти переменные могут быть одного или разных типов.

Отличие оператора **READ** (R1,R2,..Rn) от оператора **READLN** (R1,R2,..Rn) в том, что после оператора **READ** (R1,R2,..Rn) курсор на экране выполнения программы остается в той же строке после последнего набранного значения, а при использовании оператора **READLN** (R1,R2,..Rn) курсор переносится на следующую строку.

Оператор вывода на экран (обращение к стандартной процедуре вывода) имеет следующий формат:

Write (<список вывода>)

Здесь элементами списка вывода могут быть выражения различных типов (в частности, константы и переменные), например:

Write (234); {Выводится целая константа)

Write (A + B-2); {Выводится результат вычисления выражения}

Не рекомендуется это использовать при расчете рекурсивных или сложных математических функций !

Write (X, Summa, Argl, Arg2); {Выводятся значения переменных}

Оператор вывода на экран также может иметь вид

WriteLn (<список вывода>)

Здесь WriteLn (от Write line) — записать строку. Действие этого оператора отличается от Write тем, что после вывода последнего в списке значения происходит перевод курсора в

начало следующей строки. Оператор WriteLn, записанный без параметров, выполняет перевод строки.

Форматы вывода. Формат определяет представление выводимого значения на экране и отделяется от соответствующего ему элемента двоеточием. Если указатель формата отсутствует, то машина выводит значение по определенному правилу, предусмотренному по умолчанию.

Далее кратко в справочной форме приведем правила и примеры бесформатного и форматированного вывода величин различных типов. Для представления списка вывода используем следующие обозначения:

I, P(O — целочисленные выражения;

R — выражение вещественного типа;

B — выражение булевского типа;

Ch — символьная величина;

S — строковое выражение;

цифра;

* знак «+» или «-»;
_ пробел.

Форматы оператора Write

1:P — выводится десятичное представление величины I в крайние правые позиции поля шириной P.

I — выводится десятичное представление величины I, начиная с позиции расположения курсора:

Значение I	Оператор	Результат
134	Write(I:6)	_ _ _ 134
312	Write((I + I):7)	_ _ _ _ 624

R — в поле шириной 18 символов выводится десятичное представление величины R в формате с плавающей точкой

Значение R	Оператор	Результат
715.432	Write(R)	_ _ 7.1543200000E+02
-1.919E+01	Write(R)	_ -1.9190000000E+01

R:P — в крайние правые позиции поля шириной P символов выводится десятичное представление значения R в нормализованном формате с плавающей точкой. Минимальная длина поля вывода для положительных чисел составляет семь символов, для отрицательных — восемь. После точки выводится по крайней мере одна цифра:

Значение R	Оператор	Результат
511.04	Write(R:15)	5.110400000E+02
46.78	Write(-R:12)	-4.67800E+01

R:P:Q — в крайние правые позиции поля шириной P символов выводится десятичное представление значения R в формате с фиксированной точкой, причем после десятичной точки выводится Q цифр ($0 < Q < 24$), представляющих собой дробную часть числа (если $Q = 0$, то ни дробная часть, ни десятичная точка не выводятся; если $Q > 24$, то при выводе используется формат с плавающей точкой):

Значение R	Оператор	Результат
511.04	Write(R:8:4)	511.0400
-46.78	Write(R:7:2)	_-46.78

Ch:P — в крайнюю правую позицию поля шириной P выводится значение Сп:

Значение Ch	Оператор	Результат
'X'	Write(Ch:3)	_ _X
'!'	Write(Ch:2,Ch:4)	_!_ _ _!

S — начиная с позиции курсора выводится значение S:

Значение S	Оператор	Результат
'Day N'	Write(S)	Day N
'RRDD'	Write(S,S)	RRDDRRDD

S:P — значение S выводится в крайние правые позиции поля шириной P символов:

Значение S	Оператор	Результат
'Day N'	Write(S:10)	_ _ _ _ _Day N
'RRDD'	Write(S:5,S:5)	_RRDD_RRDD

B — выводится результат выражения B (True или False), начиная с текущей позиции курсора:

Значение B	Оператор	Результат
True	Write(B)	True
False	Write(B, Not B)	FalseTrue

B:P — в крайние правые позиции поля шириной P символов пыводится результат булевского выражения:

Значение B	Оператор	Результат
True	Write(B:6)	_ _True
False	Write(B:6, Not B:7)	_False_ _ _True

Тема 1.6. Управление символьным выводом

Режимы экрана при работе в Паскале.

Задание координат позиции курсора на экране.

Управление цветом фона и шрифта в Паскале.

Использование для вывода на экран только процедур Write и Writeln обеспечивает программисту очень мало возможностей по управлению расположением на экране выводимого текста.

Дополнительные возможности управления выводом на экран текста (фон, цвет и пр.) обеспечивают процедуры и функции модуля **CRT**.

Для установления связи пользовательской программы с этим модулем перед разделами описаний должна быть поставлена строка **Uses CRT**

Необходимая дополнительная информация по управлению расположением на экран выводимого текста (окна, цвет фона и букв и пр.) находится в пункте меню PascalABC **Помощь – Справка – Стандартные модули – Модуль GraphABC**

Рассмотрим **понятия,** необходимые при работе с модулем CRT: режимы экрана, координаты позиции на экране, текстовое окно, цветфона и цвет символа.

Режимы экрана. Во-первых, вывод на экран может происходить в текстовом или графическом виде.

Координаты позиции. Каждая символьная позиция на экране определена двумя координатами (X,Y). Координата X – позиция в строке. Координата Y– номер строки, в которой находится символ. Строки нумеруются сверху вниз.

Например, в режиме 80x25 символ в верхнем левом углу экрана имеет координаты (1; 1), символ В нижнем правом углу экрана— (80; 25); символ в середине экрана — (40; 13).

Для установления курсора на экране в позицию с координатами (X, Y) в модуле CRT существует процедура **GoToXY(X,Y).**

Приведем для примера программу, которая очищает экран и выставляет в центре экрана символ «*».

```
Uses CRT;
Begin
    ClrScr;
    GoToXY(40,13);
    Write('*')
End.
```

Для выполнения очистки экрана используется процедура **ClrScr**

Текстовое окно. Прямоугольное пространство на экране, в которое производится вывод символов, называется текстовым окном. Положение этого окна определяется координатами верхнеголевого и нижнего правого углов прямоугольника. Если окно занимает весь экран, то в режиме 80 x 25 его координаты (I; I) и (80; 25). Это исходное окно, изменить положение и размер которого можно с помощью процедуры

Window(XI, Y1, X2, У2)

Здесь аргументы — величины типа Byte; (XI, YI) — координаты верхнего левого угла окна; (X2, Y2) — координаты правого нижнего угла окна.

Процедура назначения цвета фона

Цвет задается латинскими буквами как называется в английском языке (red, yellow, green и пр.).

TextBackGround(Color)

Здесь аргумент — величина типа Byte, задающая номер цвета. Цвет фона надо задавать после очистки текстового окна процедурой **ClrScr**

Процедура назначения цвета символа

TextColor (Color)

Весьма полезной является функция **KeyPressed** из модуля CRT, при исполнении которой происходит опрос клавиатуры и определяется, не нажата ли какая-нибудь клавиша. В результате эта функция выдает логическое значение True, если нажата любая клавиша, и значение False— в противном случае. Часто данную функцию используют для организации задержки окна результатов на экране (после выполнения программы Паскаль вызывает на экран окно редактора), для чего перед концом программы записывают следующий оператор:

Repeat Until KeyPressed;

Это пустой цикл, который «крутится на месте» до нажатия какой-либо клавиши. В это время на экране выведено окно результатов. После нажатия клавиши значение KeyPressedстанет равно True, цикл завершится, исполнение выйдет на конец программы и на экран вернется окно редактора. Этот прием можно использовать для задержки выполнения программы в любом ее месте.

Команда DELAY(x); задержка изображения в милисекундах x

Конкретные варианты реализации указанных процедур надо искать в справочной информации PascalABC.

Тема 1.7. Использование средств графики в Паскале

Графический режим экрана, графические координаты

Графические примитивы, их использование при построении изображения.

Задание цвета фона и рисунка

В состав АВС Паскаля входит **модуль GraphABC**, который содержит процедуры, функции, а также встроенные типы и константы, предназначенные для работы в графическом режиме. Чтобы воспользоваться возможностями модуля **GraphABC**, в начале программы необходимо разместить оператор использования.

Необходимая дополнительная информация по **модулю GraphABC** находится в пункте меню PascalABC **Помощь – Справка – Стандартные модули – Модуль GraphABC**

Uses GraphABC; - подключение графического модуля к программе

Графические примитивы модуля GraphABC

Procedure SetPixel(x,y: integer; c: Color);
Закрашивает пиксел с координатами (x,y) цветом c

Procedure PutPixel(x,y: integer; c:Color);
Закрашивает пиксел с координатами (x,y) цветом c

Function GetPixel(x,y: integer): Color;
Возвращает цвет пиксела с координатами (x,y)

Procedure MoveTo(x,y: integer);
Устанавливает текущую позицию рисования в точку (x,y)

Procedure LineTo(x,y: integer);
Рисует отрезок от текущей позиции до точки (x,y). Текущая позиция переносится в точку (x,y)

Procedure LineTo(x,y: integer; c:Color);
Рисует отрезок от текущей позиции до точки (x,y) цветом c. Текущая позиция переносится в точку (x,y)

Procedure Line(x1,y1,x2,y2: integer);
Рисует отрезок от точки (x1,y1) до точки (x2,y2)

Procedure Line(x1,y1,x2,y2: integer; c:Color);
Рисует отрезок от точки (x1,y1) до точки (x2,y2) цветом c

Procedure FillCircle(x,y,r: integer);
Заполняет внутренность окружности с центром (x,y) и радиусом r

Procedure DrawCircle(x,y,r: integer);
Рисует окружность с центром (x,y) и радиусом r

Procedure FillEllipse(x1,y1,x2,y2: integer);
Заполняет внутренность эллипса, ограниченного прямоугольником, заданным координатами противоположных вершин (x1,y1) и (x2,y2)

Procedure DrawEllipse(x1,y1,x2,y2: integer);
Рисует границу эллипса, ограниченного прямоугольником, заданным координатами противоположных вершин (x1,y1) и (x2,y2)

Procedure FillRectangle(x1,y1,x2,y2: integer);

Заполняет внутренность прямоугольника, заданного координатами противоположных вершин (x1,y1) и (x2,y2)

Procedure FillRect(x1,y1,x2,y2: integer);

Заполняет внутренность прямоугольника, заданного координатами противоположных вершин (x1,y1) и (x2,y2)

Procedure DrawRectangle(x1,y1,x2,y2: integer);

Рисует границу прямоугольника, заданного координатами противоположных вершин (x1,y1) и (x2,y2)

Procedure FillRoundRect(x1,y1,x2,y2,w,h: integer);

Заполняет внутренность прямоугольника со скругленными краями; (x1,y1) и (x2,y2) задают пару противоположных вершин, а w и h – ширину и высоту эллипса, используемого для скругления краев

Procedure DrawRoundRect(x1,y1,x2,y2,w,h: integer);

Рисует границу прямоугольника со скругленными краями; (x1,y1) и (x2,y2) задают пару противоположных вершин, а w и h – ширину и высоту эллипса, используемого для скругления краев

Procedure Circle(x,y,r: integer);

Рисует заполненную окружность с центром (x,y) и радиусом r

Procedure Ellipse(x1,y1,x2,y2: integer);

Рисует заполненный эллипс, ограниченный прямоугольником, заданным координатами противоположных вершин (x1,y1) и (x2,y2)

Procedure Rectangle(x1,y1,x2,y2: integer);

Рисует заполненный прямоугольник, заданный координатами противоположных вершин (x1,y1) и (x2,y2)

Procedure RoundRect(x1,y1,x2,y2,w,h: integer);

Рисует заполненный прямоугольник со скругленными краями; (x1,y1) и (x2,y2) задают пару противоположных вершин, а w и h – ширину и высоту эллипса, используемого для скругления краев

Procedure Arc(x,y,r,a1,a2: integer);

Рисует дугу окружности с центром в точке (x,y) и радиусом r, заключенной между двумя лучами, образующими углы a1 и a2 с осью OX (a1 и a2 – вещественные, задаются в градусах и отсчитываются против часовой стрелки)

Procedure FillPie(x,y,r,a1,a2: integer);

Заполняет внутренность сектора окружности, ограниченного дугой с центром в точке (x,y) и радиусом r, заключенной между двумя лучами, образующими углы a1 и a2 с осью OX (a1 и a2 – вещественные, задаются в градусах и отсчитываются против часовой стрелки)

Procedure DrawPie(x,y,r,a1,a2: integer);

Рисует сектор окружности, ограниченный дугой с центром в точке (x,y) и радиусом r,

заключенной между двумя лучами, образующими углы a1 и a2 с осью OX (a1 и a2 – вещественные, задаются в градусах и отсчитываются против часовой стрелки)

Procedure Pie(x,y,r,a1,a2: integer);
Рисует заполненный сектор окружности, ограниченный дугой с центром в точке (x,y) и радиусом r, заключенной между двумя лучами, образующими углы a1 и a2 с осью OX (a1 и a2 – вещественные, задаются в градусах и отсчитываются против часовой стрелки)

Procedure DrawPolygon(points: **array of** Point);
Рисует замкнутую ломаную по точкам, координаты которых заданы в массиве points

Procedure FillPolygon(points: **array of** Point);
Заполняет многоугольник, координаты вершин которого заданы в массиве points

Procedure Polygon(points: **array of** Point);
Рисует заполненный многоугольник, координаты вершин которого заданы в массиве points

Procedure Polyline(points: **array of** Point);
Рисует ломаную по точкам, координаты которых заданы в массиве points

Procedure Curve(points: **array of** Point);
Рисует кривую по точкам, координаты которых заданы в массиве points

Procedure DrawClosedCurve(points: **array of** Point);
Рисует замкнутую кривую по точкам, координаты которых заданы в массиве points

Procedure FillClosedCurve(points: **array of** Point);
Заполняет замкнутую кривую по точкам, координаты которых заданы в массиве points

Procedure ClosedCurve(points: **array of** Point);
Рисует заполненную замкнутую кривую по точкам, координаты которых заданы в массиве points

Procedure TextOut(x,y: integer; s: string);
Выводит строку s в прямоугольник к координатами левого верхнего угла (x,y)

Procedure TextOut(x,y: integer; n: integer);
Выводит целое n в прямоугольник к координатами левого верхнего угла (x,y)

Procedure TextOut(x,y: integer; r: real);
Выводит вещественное r в прямоугольник к координатами левого верхнего угла (x,y)

Procedure DrawTextCentered(x,y,x1,y1: integer; s: string);
Выводит строку s, отцентрированную в прямоугольнике с координатами (x,y,x1,y1)

Procedure DrawTextCentered(x,y,x1,y1: integer; n: integer);
Выводит целое значение n, отцентрированное в прямоугольнике с координатами (x,y,x1,y1)

Procedure DrawTextCentered(x,y,x1,y1: integer; r: real);

Выводит вещественное значение r, отцентрированное в прямоугольнике с координатами (x,y,x1,y1)

Procedure FloodFill(x,y: integer; c:Color);

Заливает область одного цвета цветом c, начиная с точки (x,y).

Тема 1.8. Программирование циклических структур

Определение циклической программы. Виды циклов

Программирование циклов с условием.

Программирование циклов с параметром.

Операторы цикла служат для организации повторяющихся действий в программе.
Различают следующие виды циклов по способу выполнения :

- цикл с предусловием;

- цикл с постусловием;

- цикл с параметром.

Цикл с предусловием.

Формат оператора цикла с предусловием:

While <условие> do <тело цикла>;

While, do – зарезервированные слова («пока», «делать»);

While… do – заголовок цикла

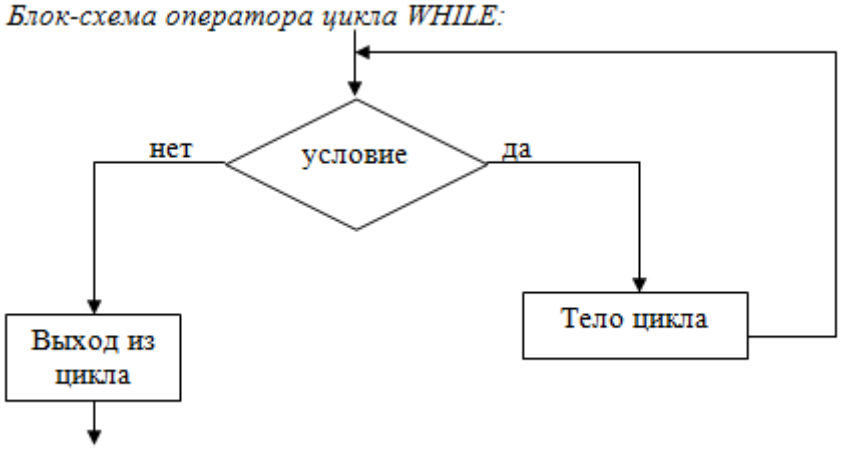
<условие> – выражение логического типа (условие продолжения цикла);

<тело цикла> – произвольный оператор Паскаля (может быть составным).

Порядок работы оператора WHILE

Сначала проверяется <условие>. Если оно равно TRUE (истина), то выполняется тело цикла, и снова проверяется <условие>. Так продолжается до тех пор, пока значение условия не станет равно FALSE (ложь), т.е. пока условие не нарушится. Если значение условия равно FALSE, то происходит выход из цикла и переход к следующему после While оператору.

Блок-схема оператора цикла WHILE:



Примерами оператора **цикла с предусловием** могут служить:

1. **while** M[i] <> 0 **do** i := i + 1;
2. **while** X <= N **do**

begin

Fact :=Fact * X;

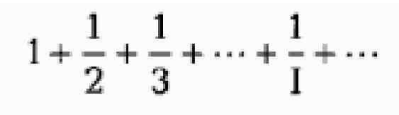
X := X + 1

end;

Цикл допускает только один оператор (**простой или составной**).

Составной оператор – это несколько простых, заключенных в операторные скобки **begin – end** (в примере 2 составной оператор имеет два простых).

Для примера приведем фрагмент программы на Паскале, вычисляющий с заданной точностью ε сумму гармонического ряда



Суммирование прекращается, когда очередное слагаемое становится меньше ε или когда целая переменная I достигает значения Maxlnt:

```
S := 0;
I := 1;
While (1/I >= Eps) And (I < MaxInt) Do
Begin
   S := S + 1/I;
   I := I + 1
End;
```

Оператор цикла с постусловием

Оператора цикла REPEAT аналогичен оператору цикла WHILE, но отличается от него тем, что условие проверяется после очередного выполнения тела цикла. Поэтому оператор REPEAT называется циклом с постусловием. Таким образом, цикл REPEAT всегда выполняется хотя бы один раз. Кроме того, в операторе REPEAT задается условие выхода из цикла, а не продолжения цикла, как в операторе WHILE. За это оператор REPEAT называется также циклом «ДО», т.к. он прекращает выполняться, как только значение условия, записанного после слова UNTIL, становится равным TRUE (истина).

Формат оператора цикла с постусловием:

repeat

<тело цикла>

Until <условие выхода из цикла>;

repeat, until – зарезервированные слова («повторять», «до тех пор, пока»)

repeat – заголовок цикла

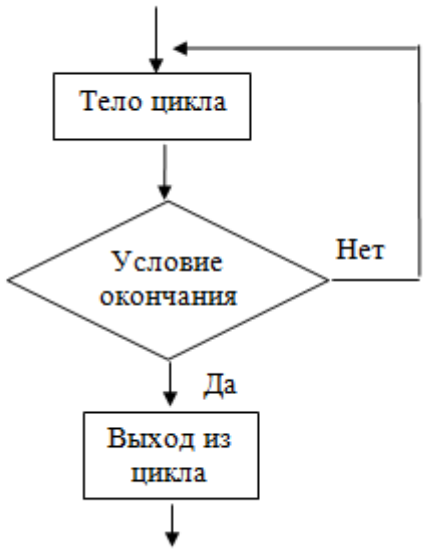
<тело цикла> – произвольный оператор Паскаля (может быть составным)

<условие выхода из цикла> – выражение логического типа

Порядок работы оператора REPEAT

Сначала выполняется <тело цикла>, затем проверяется значение <условия выхода из цикла>. В том случае, если оно равно FALSE (ложь), т.е. не выполняется – тело цикла повторяется еще раз. Так продолжается до тех пор, пока условие не станет TRUE (истина). Если значение условия равно TRUE, то происходит выход из цикла и переход к следующему после REPEAT оператору.

Блок-схема оператора цикла REPEAT:



Пример:
Вычислить функцию y=4*A*X для X=2,4,6,8,10.

```
Var A,x,y : integer;
begin
  x:=2;
  Repeat
     y:=4*A*x;
     writeln (x:4:2,'  ',y:4:2);
     x:=x+2;
  Until X>10;
End.
```

Предыдущая задача с использованием цикла с постусловием решается следующим образом :

```
S := 0;
I := 1;
Repeat
  S := S + 1/I;
  I := I + 1
Until (1/I < Eps) Or (I >= MaxInt);
```

Оператор цикла с параметром

В случаях, когда число повторений известно заранее, удобно использовать оператор цикла с параметром. В этом цикле важную роль играет переменная-параметр, которая на каждом шаге цикла автоматически изменяет свое значение ровно на единицу. Поэтому переменную-параметр называют также счетчиком.

Оператор цикла FOR можно использовать в двух вариантах.

Форматы оператора цикла с параметром:

Вариант 1 (с увеличением счетчика):

For <параметр>:=<начальное значение> to <конечное значение> do

<оператор>;

Вариант 2 (с уменьшением счетчика):

For <параметр>:=<начальное значение> downto <конечное значение> do

<оператор>;

For, to, downto, do - зарезервированные слова («для», «до», «уменьшая до», «делать»)

For … do – заголовок цикла

<параметр> – параметр цикла - переменная любого порядкового типа;

< начальное значение> – выражение того же типа, что и параметр;

<конечное значение> – выражение того же типа, что и параметр;

<оператор> - тело цикла. Тело цикла может быть произвольным оператором Pascal, в том числе составным.

Порядок работы оператора FOR

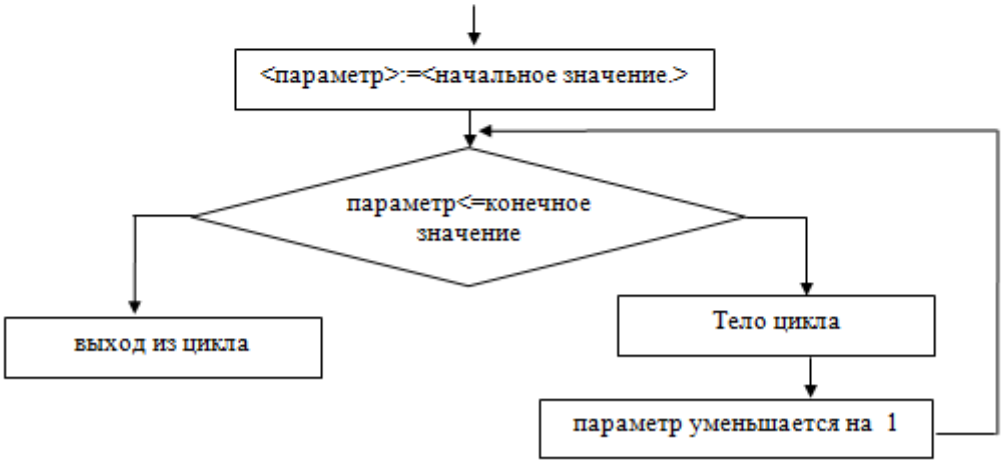
При первом обращении к оператору цикла FOR вначале вычисляются выражения <начальное значение> и <конечное значение> и осуществляется присваивание <параметр>:=<начальное значение>.

После этого циклически повторяются следующие действия:

1. Проверяется условие <параметр><конечное значение>
2. Если условие принимает значение TRUE (истина), то выполняется тело цикла, если условие принимает значение FALSE (ложь), то оператор FOR завершает работу, и происходит переход на оператор, следующий за циклом
3. Значение переменной-параметра изменяется на 1 (to) или – 1 (downto) и далее с п.1.

Блок-схема оператора For ...to

Блок-схема оператора For ... downto



Пример 1: Для значений X=1,2,...,10 вычислить значения функции Y=-5*X.

FOR X:=1 **TO** 10 **DO** Y:=-5*X;

Пример 2: Для значений X=10,9,...,1 вычислить значения функции Y=7*X.

FOR X:=10 **DOWNTO** 1 **DO** Y:=7*X;

Пример.

Посчитать значения функции $Y=5x^2+6$ ***для значений*** $x = 3,4,5,\ldots,20$. ***Все значения вывести на экран.***

```
Var x,y : integer;
Begin
    For x:=3 to 20 do
        Begin
        Y:=5*sqr(x)+6;
        Writeln('x= ', x, 'Y= ', Y);
        End;
End.
```

Тема 1.9. Программирование разветвленных структур

Определение разветвленной программы.

Операторы, использующиеся для разветвления.

Программирование операторов условия, безусловного перехода.

Программирование оператора выбора.

1. **Разветвляющейся программой** называется программа, в которой выбирается один из нескольких возможных путей вычислительного процесса. Ветвление в программах достигается использованием:

- операторов безусловного перехода;
- операторов условия;
- оператора выбора.

Оператор (безусловного) перехода вызывает передачу управления оператору, которому предшествует **метка.**

Метка представляет собой целое десятичное число в диапазоне от 1 до 9999. Допускается использование в качестве меток также и идентификаторов.

При использовании операторов перехода должны соблюдаться **следующие правила**:

1.
    1. Все метки, используемые в блоке, должны быть описаны, причем каждая не более одного раза.
    2. Метка, указанная в операторе перехода, должна помечать оператор, находящийся в том же блоке, что и сам оператор перехода. Таким образом, НЕ ДОПУСКАЮТСЯ переходы вовне процедур (функций) или внутрь них.

Переход внутрь структурного оператора может вызвать непредсказуемые эффекты, хотя компилятор не выдает сообщения об ошибке.

Пример структуры программы с оператором безусловного перехода:

Label M1;

Var a:real;

Begin

Readln(A);

……..

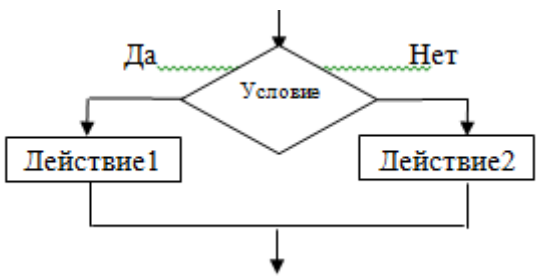
GOTO M1;

……

M1: writeln(A*2);

End.

2. **Условные операторы** предназначены для **выбора к исполнению одного из двух возможных операторов (ветвей) в зависимости от некоторого условия** (при этом одно из действий может быть пустым, то есть отсутствовать).



Данный оператор выполняется следующим образом. Сначала вычисляется выражение, стоящее после **служебного слова if**. Если значение выражения есть истина, то выполняется оператор, указанный после **служебного слова then**. Если результат вычисления выражения есть ложь, выполняется оператор, следующий за **служебным словом else**. **Когда в операторе присутствуют все служебные слова**, то говорят, что это **оператор в полной форме**. Если часть условного оператора, начиная со слова else, отсутствует, то управление немедленно передается оператору, следующему за условным, а сам оператор имеет **краткую форму** записи.

Пример условного оператора:

If X<У **then**

Max := Y

else

Max := X

После служебных слов **then** и **else** может стоять только один (**простой или составной**) оператор.

Пример задачи с оператором условия. Вычислить значения функции

$$y = \begin{cases} ax^2 - b, & x < 0 \\ bx^3 + ax^2 - x, & x = 0 \\ bx - a, & x > 0 \end{cases}$$

```
Var a,b,x,y : real;
begin
 writeln ('введи a,b,x');
 readln (a,b,x);
 if x<0 then y := a*sqr(x) – b  else
          if x=0 then y := b*x*x*x + a*x*x – x
 else  y := b*x – a;
 writeln ('y=',y);
 end.
```

Или так можно если не уверены,что правильно составите программу со вложенными условными операторами :

```
Var a,b,x,y : real;
Begin
   Writeln ('введи a,b,x');
  Readln (a,b,x);
   If  x<0 then y:= a*sqr(x)-b else y:=b*x-a;
   If  x=0 then  y:=b*x*sqr(x) + a*sqr(x)-x;
  Writeln ('x=',x:5:3,  'y=',y:5:3);
End.
```

Постановка задачи.

Разработать программу, вычисляющую значение функции f для произвольного значения x

$$f = \begin{cases} x^2, если X > 0 \\ x^3, если X \le 0 \end{cases}$$

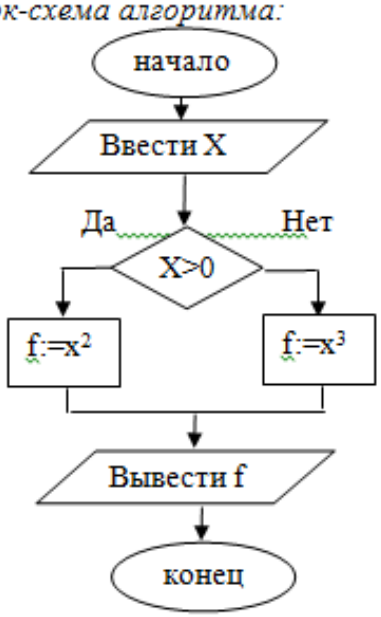
Список используемых переменных.

Исходные данные: x (вещественное число)

Результат: f (вещественное число)

Тип алгоритма: разветвляющийся.

Блок-схема алгоритма:



```
Program vetvlenie;

Var x,f:real;
Begin
Write('Ведите x ='); { вывод сообщения}
Readln(x); { ввод значения x}
If x>0 then f:=sqr(x) else f:=sqr(x)*x;{условный оператор – вычисление значения f по условию}
Writeln('f(',x:5:2,')=',f:7:2); {вывод результата}
End.
```

Тестовый пример:

1. x=2, f=4 (так как 2>0)
2. x=–2, f=–8 (так как -2 ≤ 0)

Протокол работы программы:

```
Ведите x =2
f( 2.00)=  4.00
Ведите x = –2
f( -2.00)= –8.00
```

Пример Требуется составить программу вычисления площади треугольники но длинам трех сторон *a. b. c.*

Для решения задачи используется формула Герона

$$\sqrt{p(p-a)(p-b)(p-c)},$$

где ***p*** **= (a+b+c*) / 2*** — полу периметр треугольника.

Исходные данные должны удовлетворять основному соотношению для сторон треугольника: длина каждой стороны должна быть меньше суммы длин двух других сторон.

Имея возможность в одном условном операторе записывать достаточно сложные логические выражения, можно сразу «отфильтровать» все варианты неверных исходных данных

```
Program Geron;
Var A, B, C, P, S : Real;
Begin
   WriteLn('Введите длины сторон треугольника:');
   Write('a = '); ReadLn(A);
   Write('b = '); ReadLn(B);
   Write('c = '); ReadLn(C);
   If (A > 0) And (B > 0) And (C > 0) And (A + B > C)
         And (B + C > A) And (A + C > B)
   Then  Begin
            P := (A + B + C)/2;
            S := sqrt(P * (P-A) * (P-B) * (P-C));
            WriteLn('Площадь = ', S)
         End
   Else WriteLn('Неверные исходные данные')
End.
```

Оператор выбора является **обобщением условного оператора** для случая произвольного числа альтернатив (ветвей). Конкретно, можно сопоставить различным значениям некоторого выражения соответствующие им операторы.

Простейший пример оператора варианта:

Case Color **of**

Red : X := Y+2;

Yellow : X := Y-2;

Green : X := Y

end;

Данный оператор будет выполняться следующим образом :

Сначала вычисляется текущее значение переменной Color. Затем это значение сравнивается с константами, записанными перед операторами. При совпадении значения переменной с одной констант будет выполнен оператор, "помеченный" данной константой. На этом выполнение оператора варианта будет завершено. Если значение переменной Color не совпадает ни с одной константой, то данный оператор не выполняет никаких действий. Для того, чтобы задать некоторые действия для случая такого несовпадения, можно использовать альтернативу **else**, например:

case (K+l)*2 **of**

2 : Add(A,B);

4 :Multiply(A,B);

0 :Subtract(A,B)

else

Writeln('Ошибка!')

end.

Оператор, следующий за служебным словом **else**, будет выполнен в том случае, если значение выражения (К+1) *2 не совпадет ни с одной из констант 2,4,0. Кроме одиночных констант в альтернативах оператора варианта могут быть заданы списки и/или диапазоны значений, которые в этом случае должны разделяться символом ',' (запятая), например:

case Switch **of**

1..2: Procl;

3,4,10..20: Proc2;

5, 6 : Proc3

else

Proc4

end

В последнем примере оператор процедуры Proc2 будет выполнен в том случае, когда переменная Switch будет иметь одно из следующих значений: 3,4,10,11,12, . . ., 19,20.

При использовании оператора варианта должны выполняться **следующие правила**:

1. Значение выражения, записанного после служебного слова **case**, должны принадлежать простому дискретному типу; для целого типа они должны лежать в диапазоне -32768. .32768.
2. Все константы, предшествующие операторам альтернатив, должны иметь тип, совместимый с типом выражения.
3. Все константы в альтернативах должны быть уникальны в пределах оператора варианта (то есть повторения констант в альтернативах не допускаются); диапазоны не должны пересекаться и не должны содержать констант, указанных в данной или других альтернативах.
4. В каждой альтернативе может быть только один **простой или составной оператор**.

Пример задачи с оператором выбора.

var d:'2'..'7';

Ввести значение d с клавиатуры. Вывести на экран название этой цифры (два, три, четыре, пять, шесть, семь).

В этой задаче уже дано описание переменной d и надо его использовать в программе.

```
var d:'2'..'7';
    begin
        writeln ('введи d);
        readln (d);
        case d of
                '2': writeln ('два');
                '3': writeln ('три');
                '4': writeln ('четыре');
                '5': writeln ('пять');
                '6': writeln ('шесть');
                '7': writeln ('семь');
    end;
        end.
```

Пример. Требуется составить программу перевода пятибалльной оценки в ее наименование: 5 – отлично, 4 – хорошо, 3 – удовлетворительно, 2 - неудовлетворительно

```
Program Marks;
Var N: Integer;
Begin
    WriteLn('Введите оценку:');
    ReadLn(N);
    Case N Of
          5: WriteLn('Отлично');
          4: WriteLn('Хорошо');
          3: WriteLn('Удовлетворительно');
          2: WriteLn('Неудовлетворительно')
     Else WriteLn('Нет такой оценки')
End;
```

Покажем на примере использование списка констант в операторе выбора. Следующая программа сообщает, сдал студент экзамен или не сдал, т.е. если он получил оценку 3,4 или 5, то экзамен сдан, а если – 2, то экзамен не сдан :

```
Case  N Of
      3, 4, 5 : WriteLn('Экзамен сдан')
            2 : WriteLn('Экзамен не сдан')
      Else WriteLn('Нет такой оценки');
```

Тема 1.10. Работа со структурированными данными

Понятие массива.

Описание массивов, допустимые типы для индексов и элементов массива.

Одномерные и двумерные массивы, их использование в программе.

Понятие записи, ее описание и использование в программе. Создание баз данных средствами языка Паскаль.

В повседневной и научной практике очень часто приходится работать с информацией, представленной в табличном виде. В Паскале аналогом таблиц является ***структурированный*** тип, который называется **массивом.**

Массив представляет собой совокупность пронумерованных однотипных значений, имеющих общее имя. Элементы масства обозначаются как переменные с индексами. Индексы записываются в квадратных скобках после имени массива. Например, T[1], T[6], H[I,j], H[5,1].

Массив, хранящий линейную таблицу, называется одномерным, а прямоугольную – двумерным.

Тип элементов массива является его ***базовым типом***.

Размер массива – это количество элементов в массиве.

Размерность массива – это количество индексов у элементов массива.

К каждому элементу можно обратиться по его номеру.

Описание одномерного массива

1-ый способ (в разделе описания переменных var)

var <имя массива>:**array** [<номер 1 эл-та>..<номер последнего эл-та>] **of** <тип элементов>;

2-ой способ (в разделе описания типов type):

Type <имя типа>= **array**[<номер 1 эл-та>..<номер последнего эл-та>] **of** <тип элементов>;

Var <имя массива>:<имя типа>;

В этом случае задается новый тип данных. Введение нового типа данных позволяет обрабатывать массивы в процедурах и функциях.

Обработка массивов происходит поэлементно. Чтобы получить доступ к нужному элементу массива, нужно указать имя массива и индекс (номер) элемента массива.

Пример:

Var A:array [1..5] of real;

…..

A[1]:=0;

A[3]:=9;

Write (A[3]);

Элементами массива могут быть значения любого типа.

В Паскале количество элементов массива всегда должно быть фиксировано, то есть определяться при трансляции программы. Это считается определенным недостатком языка, так как не во всех программах можно заранее предсказать необходимый размер массива.

Двумерный массив (матрица) – это таблица элементов одного типа, состоящая из нескольких строк с равным числом элементов в каждой.

Каждый двумерный массив определяется именем, числом строк и столбцов.

Описание двумерного массива:

Type

Matr=array[Нижняя граница индекса1..Верхняя граница индекса1,

Нижняя граница индекса2..Верхняя граница индекса2]

of < тип элементов >;

Var < Имя массива> :Matr;

Например, описание массива, состоящего из 4-х строк и 5-ти столбцов:

Type Matr=array[1..4,1..5]of integer;

Var A:Matr;

Ввод элементов матрицы:

Значение элементов двумерного массива выводятся на экран и вводятся с клавиатуры, как правило, по строкам, т.е. сначала все элементы первой строки, затем второй и т.д.

```
for i:=1 to 4 do

for j:=1 to 5 do

begin

write('A[',i,',',j,']=');

readln(A[i,j]);

end;
```

Вывод элементов матрицы А будет выглядеть так:

```
for i:=1 to 4 do

begin

for j:=1 to 5 do

write(A[i,j]:6);

writeln;

end;
```

При работе с массивами необходимо помнить об ограничении на размер общей памяти под переменные.

Каждый элемент массива обозначается именем массива с индексом. Индекс заключается в квадратные скобки, например A[1], B[i]. Имя элемента массива тоже является переменной, с которой можно выполнять различные операции, допустимые для её типа. Такая переменная называется переменной с индексом.

Элемент массива может стоять как в левой, так и в правой части оператора присваивания или в выражениях. Над элементами массива можно производить все операции, которые допустимы для данных его типа.

Для ввода и вывода элементов массива используется цикл с параметром.

Описание массива производится в разделах type и var, или только в разделе var, например:

```
Type Mass= array [1..10] of integer;

Var A,B : Mass;
```

C : array [0..20] of char;

ВНИМАНИЕ ! Циклы существенно влияют на быстродействие машины и объем оперативной памяти. Поэтому рекомендуется делать ввод и обработку элементов в одном цикле, а вывод результата – в другом. Надо стремиться к тому,чтобы в программе было как можно меньше циклов. В идеале ввод исходных данных, обработку элементов массива и вывод результата надо делать в одном цикле.

Пример1. Ввести массив из Nэлементов целого типа и посчитать сумму этих элементов

```
type  Mas = array [1..N] of integer;
var  Vector : Mas;  i,S:integer;
begin
 S:=0;
        writeln ('введиNчисел');
        for i:=1 to N do          {для ввода массива используем цикл с параметром}
   begin                            {так как сумму можно считать сразу же при вводе}
    readln (vector[i]);           {элементов, то это действие включено в этот же цикл}
    S:= S + vector[i];
    end;
  writeln ('сумма элементов массива равна', S);
   end.
```

Пример2. Ввести массив из N строк и M столбцов элементов целого типа и посчитать суммы элементов для каждой строки. Все суммы вывести на экран.

```
var  M: array [1..n, 1..m] of integer;
S : array [1..n] of integer;
i, j : integer;
begin
        writeln ('введи N строк по M чисел в строке: ');
  for i:=1 to n do begin
     S[i]:=0;
  For  j:=1 to m do       begin
          read(M[I,j]);   {числа надо вводить через пробел, в конце строки}
        S[i]:=S[i]+M[I,j]; end;       {нажимать  клавишу Enter}
                    end;
   for i:=1 to N  do write ( 'S[',I,']=',S[i], ' ');
end.
```

В примерах N и M можно задавать произвольные числа при описании массивов. Но они должны быть меньше запрашиваемой размерности массива от пользователя.

```
{Ввести массив, состоящий из N строк по M целых чисел в строке.
Найти суммы всех строк и вывести все суммы на экран.}
const
 n=10;
 m=10;
var
  A: array [1..n, 1..m] of real;
  S: array [1..n] of real;
  i,c,v,j:integer;
begin
  write('Введите количество строк=');
  readln(c);
  Write('Введите количество столбцов=');
  readln(v);
  for i:=1 to c do
  begin
    s[i]:=0;
    for j:=1 to v do
    begin
      read(A[i, j]);
      s[i]:= s[i] + a[i, j];
    end;
  end;
  for i:= 1 to c do write('S[', i, ']=', s[i], '');
end.
```

Рекомендуемый способ объявления массивов

1. Объявить константы, определяющие диапазон индексов.

2. Определить собственный тип данных для массива, использующий для определения диапазона объявленные на предыдущем шаге константы.

3. Объявить переменные созданного типа данных.

Например:

Const

LOW_BOUND=0;

UP_BOUND=10;

Type

MyArray=**Array**[LOW_BOUND..UP_BOUND] **Of** Integer;

Var

A, B :MyArray;

При таком способе объявления массивов, можно будет организовывать циклы по всем их

элементам, используя эти константы

For i:=LOW_BOUND **To** UP_BOUND **DO**

…

В связи с этим, разработка и отладка программы значительно облегчится, так как если в результате модификации программы потребуется изменить диапазон изменения индексов

массива, то будет достаточно изменить значения констант LOW_BOUND и UP_BOUND.

Вносить изменения в код при этом не потребуется.

Пример обработки массива

Составить программу, которая:

- вводит массив, состоящий из 5 целых чисел;
- вычисляет сумму элементов массива;
- определяет максимальный элемент массива;
- выводит на экран все четные элементы массива;
- выводит на экран все элементы массива с четными номерами;
- посчитывает количество элементов массива, которые больше числа 4;
- увеличивает на 1 все элементы массива, которые меньше числа 5;
- выводит на экран полученный массив.

Используемые алгоритмы

Циклический – для ввода, обработки и вывода массива.

Список используемых переменных

Входные данные:
A – массив, состоящий из 5 целых чисел;
Выходные данные:
S – сумма элементов массива A;
max – максимальный элемент массива A;
k – количество элементов массива A, которые больше числа 4;
A – измененный массив.

Текст программы на Pascal:

```
Program Massiv;
Uses CRT;
Type mas=array[1..5] of integer;
Var a:mas;
      S,I,k,P,min,max :integer;
      Begin
S:=0; k:=0;
{ввод массива}
Writeln('Ввод массива:');
for i:=1 to 5 do
Begin
   Write('Введите элемент массива A[',i,']=');
   ReadLn(a[i]);
   max:=a[1];
{суммирование элементов массива}
   S:=S+a[i];
   Writeln('Сумма элементов массива =',S:8);
{вывод на экран всех четных элементов массива}
  If a[i] mod 2=0 Then Write('Четные элементы массива:'a[i]);
{вывод на экран всех элементов массива с четными индексами}
  If i mod 2=0 Then Write('Элементы массива с четными индексами:' 'a[',i,']=',a[i],' ');
{подсчет количества элементов массива, которые больше числа 4}
  If a[i]>4 Then k:=k+1;
  Write('Количество элементов, больших числа 4=',k);
{увеличение на 1 всех элементов массива, которые меньше числа 5}
   If a[i]<5 Then a[i]:=a[i]+1;
End;
```

```
{поиск максимального элемента массива}
For i:=2 to 5 do
  If a[i]>max Then max:=a[i];
Writeln('Максимальный элемент=',max);
{вывод обработанного массива}
for i:=1 to 5 do
   Writeln('a[',i,']=',a[i]);
End.
```

Запись представляет собой совокупность ограниченного количества данных различного типа. В отличие от массива запись объединяет в одну группу данные разных типов, поэтому её называют также комбинированным типом.

Описание комбинированного типа представляет собой список описаний его элементов (которые называются также ***полями*** записи); каждое описание похоже на описание простой переменной. Список полей начинается служебным словом **record** и должен завершаться служебным словом **end.**

Структура записи состоит из фиксированного числа компонент (полей), каждое из которых имеет собственное (уникальное в пределах записи) имя и ПРОИЗВОЛЬНЫЙ тип. Это удобно, например, при работе с базами данных, которые можно организовывать с помощью записей. Запись в целом и её отдельные элементы обозначаются именами.

Например: ведомость на стипендию содержит различные данные: порядковый номер N-целое положительное число, фамилия, И.О. - массив символов, размер стипендии - действительное число. Для этой ведомости создадим запись:

- C - имя записи;
- N - номер в списке;
- FAM - фамилия, имя;
- ST - стипендия.

N, FAM, ST - называются полями записи. Все имена полей должны быть уникальными в пределах данной записи.

Вся ведомость состоит из стольких записей, сколько учащихся должно получить стипендию, на одного учащегося по одной записи.

К элементу записи можно обратиться с помощью уточненного имени, которое содержит имя записи и через точку - имя элемента записи. Для нашей записи :

C.N - обращение к номеру;

C.FAM - обращение к фамилии;

C.ST - обращение к стипендии.

Записи описываются в разделе описаний и используются в разделе операторов. **Описание записи имеет вид:**

TYPE имя-типа = **RECORD**

имя-элемента1: тип;

имя-элемента2: тип;

.....

имя-элемента N: тип

END;

VAR имя-записи : имя-типа;

Для ведомости на стипендию описание записи имеет вид:

TYPE T=**RECORD**

N: integer;

FAM: string[20];

ST: real

END;

VAR C1,C2 : T;

Ввод и вывод данных для списка производится отдельно для каждого элемента записи, который рассматривается как обычная переменная. Обращение к записи в целом, а не только к её элементам, допускается лишь в операторе присваивания. Слева и справа от знака присваивания должны быть имена записей одинакового типа: например, C1:=C2.

Пример.

Для каждого из 25 студентов группы известны фамилия и оценки (в баллах) по 5 дисциплинам. Требуется вычислить среднюю оценку каждого студента и выбрать человека, имеющего максимальный средний балл.

Данные для решения рассматриваемой задачи можно описать как запись следующим образом:

Type student=record

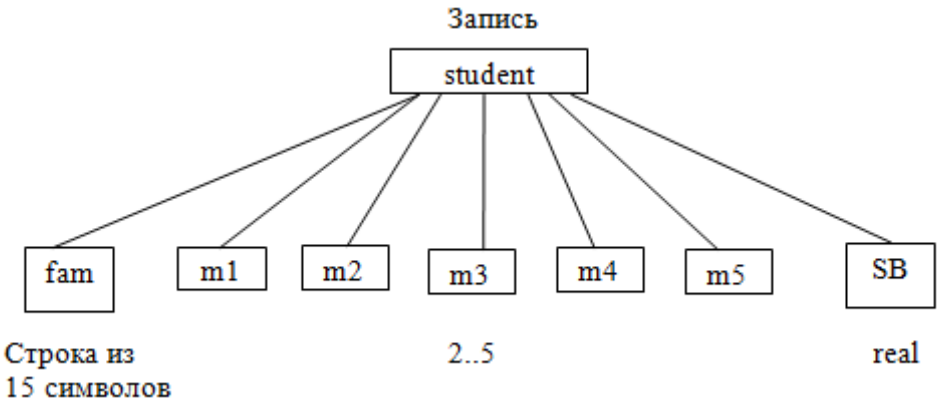
fam:string[15];

m1,m2,m3,m4,m5:2..5;

SB:real

end;

Var stud:student;



Доступ к полям записи можно осуществить 2 способами.

1-ый способ: указание имени переменной и имени поля.

Например, stud.fam – фамилия студента, stud.SB – средний балл студента.

2-ой способ: использование **оператора присоединения with.**

Общий вид оператора присваивания WITH:

With <имя записи > Do <оператор >;

Внутри оператора к компонентам записи можно обращаться, указывая только имя соответствующего поля.

Например:

with stud do

begin

```
Readln(fam);

ReadLn(m1,m2,m3,m4,m5);

SB:=(m1+m2+m3+m4+m5)/5;

end;
```

Пример задачи с записью.

Составить список из N учащихся, содержащий фамилию, имя, оценки по двум предметам. Вывести на экран список фамилий тех учащихся, которые имеют двойки по какому-либо предмету. Если таких нет, то вывести соответствующую надпись.

Особенности задачи:

1. В этой задаче надо описать массив из N записей, так как надо хранить сведения обо всех фамилиях и оценках студентов (т.е. получаем базу данных на языке Паскаль).
2. Для того чтобы отметить факт вывода на экран какой-то фамилии надо применить переменную Dv:=0 на начальном этапе, позволяющую ввести подсчет в нашем случае «двоек». Если она на выходе равна 0, то двоечников нет. В противном случае можно увидеть количество «двоек». Такая переменная называется **флагом**.

```
type stud = record
Fam: string[10];
  Oc1,oc2 : byte;
End;
Var Spis : array [1..N] of stud;
I,Dv :integer;
begin
  writeln ('введи  количество записей N: ');
   readln (N);
  writeln ('введите информацию для Ваших записей);
  for i:=1 to N do        {в первом цикле только вводим все данные}
begin
  write('Familia:  '); readln(Spis[i].Fam);
  write('Ocenka1:  ');readln(Spis[i].Oc1);
  write('Ocenka2:  ');readln(Spis[i].Oc2);
end;
Dv:=0;
for i:=1 to N do {во втором цикле анализируем оценки  и выводим фамилии}
  if(Spis[i].Oc1=2) OR (Spis[i].Oc2=2) then       {двоечников на экран}
begin
writeln('Получил 2', Spis[i].Fam);
Dv:=Dv+1;
End;
If Dv = 0 then writeln ('в группе двоек нет');
end.
```

Тема 1.11. Типовые приемы решения задач

Определение кратности чисел, суммы (произведения) числовой последовательности.

Поиск максимального (минимального) члена последовательности.

Удаление элементов из массива.

Вставка элементов в массив

Понятие сортировки, виды сортировок, сортировка элементов

массива

В этой теме рассмотрим решение типовых задач на языке Паскаль.

1.Определение кратности чисел.

Определение суммы или произведения членов числовой последовательности производится в цикле с заданным числом повторений или с предусловием (с постусловием), в зависимости от того, известно число членов этого ряда, входящих в сумму (произведение), или оно будет определяться каким-то условием, проверяемым каждый после очередного увеличения суммы (произведения). При этом предварительно, перед открытием цикла, для **вычисления суммы переменная, отслеживающая сумму, обнуляется (S:=0), а при вычислении произведения переменной-накопителю присваивается значение 1 (P:=1).**

Суммирование или умножение выполняют в цикле.

Если имена накопителей S – для суммы и P – для произведения, то формулы для накопления результатов :

S:=S+x – для нахождения суммы;

P:=P*x – для нахождения произведения.

Кратность чисел. Кратными называются числа, если одно число делится на второе без остатка. Поэтому для определения кратности чисел находят целочисленный остаток от

деления чисел и сравнивают его с нулем. Например, числа 8 и 2 кратны, т.к. остаток от деления 8 на 2 равен 0. Числа 7 и 2 не кратны, т.к. остаток отделения 7 на 2 равен 1.

Определение наибольшего (наименьшего) члена последовательности проще запрограммировать, если члены последовательности являются массивом. В этом случае вычисление наибольшего (наименьшего) значения выполняется по

следующему алгоритму:

1. после ввода элементов последовательности (массива) в отдельную переменную, предназначенную для наибольшего (наименьшего) члена последовательности занести первый элемент,
2. в цикле сравнивают эту переменную с остальными членами массива.
3. Если очередной член последовательности окажется меньше переменной, то в неё заносят этот член последовательности.

Пример. Ввести с клавиатуры массив из 10 чисел и найти наибольшее значение. Вывести его на экран.

```
Var M: array [1..N] of integer;
    Max, I: integer;
Begin
Writeln ('введи 10 чисел');
For I:=1 to 10  do readln (M[i]);            {ввод элементов массива}
Max:=a[1];                                   {занесение в ячейку Max первого значения}
For I:=2 to 10 do
Ifmax<M[i] then max :=M[i];   {сравнение Max с остальными элементами}
                                             {и занесение большего значения в Max}
Writeln ('Max= ', max);                      {вывод Max на экран}
Readln;
End.
```

Поиск максимального элемента и его номера

Переменная **max** хранит значением максимума, **K** – его номер в массиве

```
Max:=a[1]; k:=1;  {поиск начинаем с первого элемента}
For i:=2 to N do   {перебираем элементы, начиная со второго}
If a[i]>max then
  Begin
    Max:=a[i];  k:=I;  {запоминаем  значение и номер элемента, }
                       {который больше всех предыдущих}
  End;
```

Для вычисления остатка от деления одного числа на второе используют операцию mod:

8 mod 2 = 0;

7 mod 2 = 1.

Проверка кратности выполняется с помощью оператора условия. Например, проверка **четности** элементов массива A:

If A[i] mod 2 = 0 then …;

Проверка **на нечетность** элементов массива A:

If A[i] mod 2 = 1 then …;

Прием удаления элементов из массива так, чтобы оставшиеся элементы сохранили исходный порядок расположения в массиве, **выполняется с помощью алгоритма:**

- **ввод элементов массива в цикле с параметром;**
- **для индексов оставшихся элементов вводят дополнительную переменную. Вначале её значение равно нулю;**
- **в цикле с параметром проверяются значения всех элементов массива, и <u>если элемент надо оставить в массиве</u>, изменяют индекс на 1 и элементу с этим индексом присваивают значение текущего элемента. После выхода из цикла последнее значение индекса соответствует количеству элементов, оставшихся в массиве.**

Прием удаления элементов из массива так, чтобы оставшиеся элементы сохранили исходный порядок расположения в массиве.

Пример. Ввести с клавиатуры массив из 10 чисел и удалить из него все отрицательные элементы. Вывести полученный массив на экран.

```
Var M: array [1..10] of integer;
    k, I: integer;
Begin
 Writeln ('введи 10 чисел');
 For I:=1 to 10 do readln (M[i]);
 K:=0;
 For I:=1 to 10 do
   If M[i] >  0 then
   begin
     k:=k+1; M[k]:=M[i]
   end;                              {если число положительное, то его}
  For I:=1 to k do writeln(M[i],' ');     { оставляем в массиве с индексом k}
End.
```

Поиск элемента с заданным значением, например, найти первый элемент с нулевым значением

```
I:=0;   //номер элементов массива//
Repeat
  I:=i+1;
Until (a[i]=0)  //нашли требуемый элемент// or (i=n)  //массив закончился//
If a[i]=0 then  writeln ('Номер первого нулевого элемента =',i)
          Else writeln ('Таких элементов нет
```

Вставка элементов в массив должна выполняться так, чтобы не затирать значения исходных элементов. **Весь алгоритм должен разбиваться на три этапа:**

- **поиск места вставки;**
- **сдвиг элементов от места вставки до последнего элемента на количество вставляемых чисел;**
- **вставка чисел на освободившиеся места.**

Первый этап рационально выполнять циклом с предусловием. Если по условию задачи место вставки чисел известно, то первый этап может отсутствовать в алгоритме.

Второй этап – это цикл с параметром. В этом цикле управляющая переменная должна меняться с шагом –1, так как сдвиг должен выполняться, начиная с конечного элемента. При этом не затираются значения исходных элементов.

Третий этап очень прост и не требует дополнительных пояснений.

Вставка элементов в массив выполняется так, чтобы не затирать значения исходных элементов.

Пример. Ввести с клавиатуры массив из 10 чисел и вставить в него два числа A,B после первого из отрицательных элементов массива. Вставляемые числа ввести с клавиатуры. Вывести полученный массив на экран.

```
var M : array [1..12] of integer;
  A, B, i, j : integer;
begin
writeln ('Введи A, B: ');
readln (A, B);
writeln ('Введи 10 чисел: ');
for i := 1 to 10 do readln (M[i]);   {ввод элементов массива}
{Поиск места вставки чисел A, B в массив}
j := 1;
while (not M[j]<0) and (j < 10) do j := j + 1;
{Проверка: если в массиве нет отрицательных чисел, то выводится об этом
сообщение и вставки чисел не происходит; в противном случае числа
   вставляются в позиции j+1 и j+2}
if j>= 10 then writeln ('в массиве нет отрицательных чисел ')
else
begin
for i := 10 downto j do M[i+2] := M[i];   {сдвиг элементов массива с места
   вставки}
M[j+1] := A;
M[j+2] := B;
For i := 1 to 12 do write(M[i]:3);  {вывод полученного массива на экран}
end;
end.
```

Массив – это упорядоченная совокупность элементов одного типа с общим именем.

Массивы удобно использовать для хранения однородной по составу информации, например, элементов таблицы, коэффициентов уравнений, матриц.

Сортировка массива – это процесс перестановки элементов с целью упорядочивания их в соответствии с каким-либо критерием.

Например, если имеется массив целых чисел A, то после сортировки по возрастанию должно выполнятся условие:

A[1]<= A[2]<=...<= A[N], где N – наибольший индекс массива a.

Можно сортировать массивы типов INTEGER, REAL, CHAR, STRING, так как можно сравнивать переменные этих типов.

Задача сортировки распространена в информационных системах и используется как предварительный этап задачи поиска, так как поиск в упорядоченном массиве производится гораздо быстрей.

Сортировка методом прямого обмена (метод пузырька)

В основе алгоритма лежит обмен соседних элементов массива. Каждый элемент массива, начиная с первого, сравнивается со следующим и если он больше следующего, то элементы меняются местами. Таким образом, наиболее большие элементы массива как бы выдавливаются "наверх" (в конец массива), словно пузырьки газа из воды - отсюда и одно из названий этого алгоритма.

Этот процесс повторяется на единицу меньше раз, чем элементов в массиве.

© К. Поляков, 2006-2011

Сортировка методом выбора

1. Просматривая массив от первого элемента, найти минимальный элемент и поменять местами первый элемент и минимальный.
2. Просматривая массив от второго элемента, найти минимальный и поменять местами второй и минимальный.
3. И так далее до последнего элемента.

Пример 1

Постановка задачи

Отсортировать по возрастанию методом пузырька массив, состоящий из 5 целых чисел.

Используемые алгоритмы

Циклический – для ввода, сортировки и вывода массива.

Список используемых переменных

Входные данные:

a – целочисленный массив;

Выходные данные:

a – целочисленный массив, отсортированный по возрастанию;

Промежуточные величины:

i – номер шага сортировки;

k – индекс элемента массива, с которым сравнивается соседний элемент массива;

buf – используется при обмене местами двух элементов массива

Текст программы на Pascal:

```
Program Puzir;
Const N=5;
var a:array[1..N] of integer;
k:integer; { счетчик циклов }
i:integer; { текущий индекс элемента массива }
buf:integer;
begin
writeln('Сортировка массива методом пузырька');
write('Ввод массива');
for i:=1 to N do
begin
write(i' элемент->');
readln(a[i]);
end;
```

```
writeln('Сортировка');
for k:=1 to N-1 do
begin
for i:=1 to N-k do
if a[i]>a[i+1] then
begin
{обменяем местами i-й и i+1-й элементы}
buf:=a[i];
a[i]:=a[i+1];
a[i+1]:=buf;
end;
{вывод массива}
for i:=1 to N do write(a[i],' ');
writeln;
readln
end;
writeln('Массив отсортирован!');
readln
end.
```

Тестовый пример

Если исходный массив A: 12 -3 56 47 10

То отсортированный массив A: -3 10 12 47 56

1.

1. Протокол выполнения программы

Сортировка массива.

Ввод массива.

1 элемент -> 12

2 элемент -> -3

3 элемент -> 56

4 элемент -> 47

5 элемент -> 10

Сортировка

-3 12 47 10 56

-3 12 10 47 56

-3 10 12 47 56

-3 10 12 47 56

Массив отсортирован!

Пример 2

Постановка задачи

Отсортировать по возрастанию методом выбора массив, состоящий из 5 целых чисел.

Используемые алгоритмы

Циклический – для ввода, сортировки и вывода массива.

Список используемых переменных

Входные данные:

a – целочисленный массив;

Выходные данные:

a – целочисленный массив, отсортированный по возрастанию;

Промежуточные величины:

i – индекс элемента массива, с которого начинается поиск минимального;

j – индекс элемента массива, с которым сравнивается текущий минимальный элемент массива;

min – индекс минимального элемента

Текст программы на Pascal:

```
Program vibor;
const N=5;
var a:array[1..N] of integer;
i:integer; {номер элемента, от которого ведется поиск мин. эл-та}
min:integer; {индекс минимального элемента в части массива от i до верхней границы}
j:integer; {номер элемента, сравниваемого с минимальным}
buf:integer; {буфер, используемый при обмене элементов}
k:integer; {индекс элементов массива}

begin
writeln('Сортировка массива');
writeln('Ввод массива');
for k:=1 to N do
begin
write(k,' элемент ->');
read(a[k]);
end;
writeln('Сортировка ');
for i:=1 to N-1 do
begin
{поиск минимального элемента в части массива от a[i] до a[N]}
min:=i;
```

```
for j:=i+1 to N do
if a[j]
{меняем местами a[i] и a[min] }
buf:=a[i];
a[i]:=a[min];
a[min]:=buf;
{выведем массив}
for k:=1 to N do write(a[k],' ');
writeln;
readln
end;
writeln('Массив отсортирован!');
readln
end.
```

Тестовый пример

Если исходный массив A: 12 -3 56 47 10

То отсортированный массив A: -3 10 12 47 56

1.

2. Протокол выполнения программы

Сортировка массива.

Ввод массива.

1 элемент -> 12

2 элемент -> -3

3 элемент -> 56

4 элемент -> 47

5 элемент -> 10

Сортировка

-3 12 56 47 10

-3 10 56 47 12

-3 10 12 47 56

-3 10 12 47 56

Массив отсортирован!

Пример. *Упорядочивание элементов массива из 20 целых величин по возрастанию*

Для сортировки будем использовать *метод выбора*. Алгоритм состоит в том,что сначала выбирается наименьший элемент массива и меняется местами с первым элементом, затем просматриваются элементы, начиная со второго, и наименьший из них меняется местами со вторым элементом, и так далее n-1 раз. На последнем проходе цикла при необходимости меняются местами предпоследний и последний элементы массива.

```
program sort;                          { Сортировка выбором }
const n = 20;
var a : array [1 .. n] of integer;
    i, j, nmin, buf : integer;
begin
   writeln('Введите ', n, ' элементов массива');
   for i := 1 to n do read(a[i]);
   for i := 1 to n - 1 do begin    { просмотр массива n - 1 раз }
      nmin := i;
      for j := i + 1 to n do               { поиск минимума    }
         if a[j] < a[nmin] then nmin := j;
      buf     := a[i];                     { перестановка      }
      a[i]    := a[nmin];                  { двух              }
      a[nmin] := buf;                      { элементов массива }
   end;


   writeln('Упорядоченный массив:');
   for i := 1 to n do write(a[i]:5)
end.
```

Процесс обмена значениями между элементами массива с номерами I и nmin через буферную переменную buf показан на рисунке. Цифры около стрелок обозначают порядок действий.

Тема 1.12. Работа со строковым типом данных

Понятие строки и ее структура.

Процедуры и функции для работы со строками.

Строка – это последовательность символов определённой длины.

Для описания переменных строкового типа используется зарезервированное слово ***string***.

Объявление переменных типа string: Var Strl:String[31];

Str2:String;

В квадратных скобках указывается максимально допустимый размер строки (количество символов), которая может храниться в переменной. Если максимальный размер не указан, то он автоматически принимается равным 255.

Строки относятся к структурированным типам данных.

Строка похожа на одномерный массив символов: она имеет определённую длину (не больше некоторого числа), к каждому символу можно обратиться по его номеру (как в массиве):

Str1[i] – это обращение к i-му элементу строки Str1.

Переменные строкового типа вводятся и выводятся не поэлементно, как массивы, а сразу целиком, как переменные простых типов.

К символам строки применимы стандартные **символьные функции:**

- chr(x) – возвращает значение символа по коду;
- ord(A) – возвращает код заданного символа A;
- pred(A) – возвращает предыдущий символ;
- succ(A) – возвращает следующий символ;
- upcase(A) – преобразует строчную букву в заглавную (обрабатывает только латинские буквы).

Значением строки может быть любая последовательность символов, заключенная в одинарные кавычки (апострофы). Можно присваивать строке пустое значение, обозначаемое как '' (две одинарные кавычки подряд). При попытке записать в переменную строку длиннее, чем задано в описании,"лишняя" часть будет отсечена.

Строки можно присваивать, сливать и сравнивать.

Например:

```
Var st1,st2,st3,sts: string;

Begin

. . .
```

{ **Операции присваивания**}

st1:='Фамилия';

st2:='Имя';

st3:='Отчество';

{ **Операция слияния**}

sto:=st1+' '+st2+' '+st3;

{ В результате в строке sto будет 'Фамилия Имя Отчество'}

End; . . .

Сравнение строк основывается **на порядке расположения символов в таблице ASCII**.

Например: 'abcd' = 'abcd' - результат сравнения True (истина);

'abc'<'bcd' так как код символа 'a' меньше кода символа 'b' и т.п.

Для работы со строками в Паскале разработан ряд стандартных процедур и функций.

Длину строки можно определить, используя встроенную функцию Length(S), где S - строка типа string.

В ряде случаев возникает необходимость **преобразования числовых значений в строку и наоборот.** Для этого можно использовать две процедуры:

1) Str(X,S) - **преобразует числовое значение X в строковое S**.

Возможно задание формата для X в виде: X:F:n (для вещественных чисел, где F - общее число позиций выделяемых под число, включая десятичную точку, а n - число знаков в дробной части) или X:F (для целых чисел).

Например:

Str (55,s); - строковая переменная s принимает значение, равное '55'.

2) Val(S,X,ErrCode) - **преобразует строку S в числовое значение, если это возможно.** Параметр ErrCode содержит ноль, если преобразование прошло успешно, и тогда в X помещается результат преобразования, в противном случае он содержит номер позиции в строке S, где обнаружен ошибочный символ.

Например:

Val('125',K,kod) - в результате выполнения этой процедуры переменная К получает целое значение, равное 125, параметр kod=0;

Val(' 1.05',M,code) - M=1.05, code=0;

Val('100, ',N,code) - это ошибочный вызов, т.к. в исходной строке на 4-й позиции располагается недопустимый для числа символ ',' и поэтому параметр code=4, а переменная N остается без изменения.

Кроме перечисленных имеется **еще 5 функций и процедур**:

1. **Concat (S1,S2,…,Sn)** -функция, результат которой равен слиянию строк S1,S2,…,Sn.

Например: Ssum:=Concat(s1,s2,s3).

2) **Copy (S,Start,L)** - функция, результатом которой является подстрока длиной L, начинающаяся с позиции Start строки S.

Например:

Stcop:=Copy('TTTx1+++',4,2) - результатом является подстрока Stcop='x1'.

3) **Delete (S,Start,L)** - процедура, которая удаляет из S подстроку длиной L, начиная с позиции Start в строке S.

4) **Insert (S,Subs,Start)** - процедура, которая вставляет подстроку Subs в строку S начиная с позиции Start строки S. Например:

S:='Фамилия Адрес';

Insert(S,' Имя Отчество',9);

В результате строка S будет иметь вид: 'Фамилия Имя Отчество Адрес';

5) **Pos(Subs,S)** - функция поиска вхождения подстроки Subs в строку S; результатом поиска будет номер (адрес) первого символа подстроки Subs в S, если заданная подстрока найдена, или 0, если подстроки в строке нет.

Операции со строками

Склеивание **(сцепление, конкатенация, объединение строк)** – это последовательное объединение нескольких строк. Знак операции "+".

Пример:

Str1:='У Егорки';

Str2:=’отговорки';

Str3:=Str1+' '+Str2;

В результате строка Str3='У Егорки отговорки'

Склейку строк можно произвести с помощью функции Concat(Str1,Str2,...,StrN); например, Str3:=concat(Str1,Str2);

Сравнение

Сравнение строк производится посимвольно слева направо до первого несовпадающего символа; большей считается та строка, в которой код первого несовпадающего символа больше.

Две строки считаются равными, если они равны по длине и совпадают посимвольно.

Пример:

'Balkon'<'balkon' (Ord('B')

'balkon'>'balken' (Ord('o')>Ord('e'))

'balkon'>'balk' (длина первой строки больше)

'кошка '>'кошка'

'Кот'='Кот'

Стандартные процедуры и функции для работы со строками

Процедура **Delete(Str,Poz,m)** – удаляет m символов из строки Str, начиная с позиции Poz.

Пример:

Значение Str

Оператор

Результат Str

‘Мама мыла раму’

Delete(Str,6,5)

‘Мамараму’

Процедура **Insert(Str1,Str2, Poz)** – вставляет строку Str1 в строку Str2, начиная с позиции Poz, при этом Str1 не изменяется.

Пример:

Значение Str

Оператор

Результат Str

‘ЭВМ PC

Insert(’IBM- ’,Str,5)

‘ЭВМ IBM-PC’

Функция **Copy(Str, Poz,m)** – копирует (выделяет) из строки Str m символов, начиная с позиции Poz, при этом исходная строка не меняется.

Пример:

Значение Str

Оператор

Результат S

‘Мама мыла раму’

S:=Copy(Str,6,4)

‘мыла’

Функция **Length(Str)** - возвращает целое число, равное количеству символов в строке .

Пример:

Значение Str

Оператор

Результат X

‘Тест 5’

X:=Length(Str)

Функция **Pos(Str1,Str2)** – определяет первое появление (вхождение) в строке Srt2 подстроки Str1. Результатом этой функции является целое число; оно равно номеру той позиции, где находится первый символ подстроки Str1. Если в строке Str2 подстроки Str1 нет, то значение функции равно 0.

Пример:

Значение Str

Оператор

Результат X

‘Программа’

X:=Pos(‘ра’,Str)

5

‘Программа’

X:=Pos(‘б’,Str)

0

Процедура **Str(N,Str1)** – переводит числовое значение N встроковое и присваивает результат строке Str1, причём можно переводить как целые числа, так и вещественные, и есть возможность задавать формат числа, например: Str(f:6:3,Str1);

Пример:

Значение N

Оператор

Результат Str

1234

Str (N,Str)

‘1234’

Процедура **Val(Str,N,K)** – переводит строковое значение Strв числовое N. Если строка Str действительно является записью числа (целого или вещественного), то значение k=0, иначе K будет равно номеру первого ошибочного символа, а значение N не определено.

Пример:

Значение Str

Оператор

Результат N

Результат K

’123’

Val(Str,N,K)

123

0

’12-33’

Val(Str,N,K)

3

Пример. Ввести строки S и S1, символ C. Перед каждым вхождением символа C в строку S вставить строку S1. Полученную строку вывести на экран.

```
var s, s1 : string;
    c : char;
    i : integer;
begin
  writeln('Введи строку s и подстроку s1');
  readln(s);
  readln(s1);
  writeln('Введи символ “c”);
  readln(c);
  for i:=length(s) downto 1 do
      if s[i] = ‘c’ then  insert(s1,s,i+1);
      writeln (s);
end.
```

Пример. *Получение из слова ВЕЛИЧИНА слова НАЛИЧИЕ*

```
Program Slovo_1;
Var Sl1, Sl2 : String[10];
Begin
   Sl1 := 'ВЕЛИЧИНА';
   Sl2 := Copy(Sl1,7,2) + Copy(Sl1,3,4) + Sl1[2];
   WriteLn(Sl2)
End.
```

Пример. *Программа получения из слова СТРОКА слова СЕТКА*

```
Program Slovo_2;
Var Sl : String[10];

Begin
   Sl := 'СТРОКА';
   Delete(Sl,3,2);
   Insert('Е',Sl,2);
   WriteLn(Sl)
End.
```

Пример. *Формирование символьной строки, состоящей из N звездочек (где N – целое число, 1 ≤N≤255)*

```
Program Stars;
Var A : String;
   N, I : Byte;
Begin
   Write ('Введите число звездочек');
   ReadLn(N);
   A := '';
   For  I := 1 To N Do
        A := A + '*';
   WriteLn(A)
End.
```

Здесь строковой переменной А сначала присваивается значение пустой строки (‘ ‘), а затем к ней присоединяются звездочки.

Пример. *Подсчет в символьной строке числа цифр, предшествующих первому значению «!».*

```
Program C;
Var S : String;
    K, I : Byte;
Begin
  WriteLn('Введите строку');
  ReadLn(S);
  K := 0;
  I := 1;
  While (I <= Length(S)) And (S[I] <> '!') Do
  Begin
    If (S[I] >= '0') And (S[i] <= '9')
    Then K := K + 1;
    I := I + 1
  End;
  WriteLn ('Количество цифр до символа "!" равно ', K)
End.
```

В этой программе переменная К играет роль счетчика цифр, а переменная I – роль параметра цикла. Выполнение цикла закончится при первом же выходе на символ «!» или (если в строке такого символа нет) при выходе на конец строки. Символ S[I] является цифрой, если истинно отношение ‘0’≤S[I]≤’9’

Тема 1.13. Использование множеств в программе

Понятие множества. Множественный тип данных.

Выполнение операций с множествами

Под множеством в языке Паскаль понимают **ограниченный неупорядоченный набор различных элементов одинакового типа, логически связанных друг с другом**. Количество элементов, входящих в множество, может изменяться (в пределах от 0 до 255).

Множество, не содержащее элементов, называется *пустым*.

Множество имеет имя. Тип элементов, входящих в множество, называется базовым. В качестве базового типа можно использовать любой порядковый тип, кроме Word, Integer, Longint.

Множества должны быть объявлены либо в разделе Var, либо в разделах Type и Var, одновременно:

Var Имя множества: ***Set of*** базовый тип;

или

Type Имя типа=**Set of** базовый тип;

Var Имя множества: Имя типа;

Например:

```
Type

TM=Set of 1..100;

TS=Set of 'a'..'z';

Var Mch:TM; {Множество целых чисел от 1 до 100}

MSym:TS; {Множество строчных латинских букв}

M: Set of 1..10; {Множество целых чисел от 1 до 10}
```

Значения переменных множества задаются в разделе операторов с помощью **конструктора** множества, который представляет собой список элементов базового типа, заключенный в квадратные скобки.

Например:

```
Var M1,M2,M3:set of 1..99;

Begin . . .

M1:=[]; { Множество пустое}

M2:=[1,3,5,7,9]; { Множество нечетных чисел в первом десятке}

M3:=[2,4,6,8]; { Множество четных чисел в первом десятке}

. . .

End.
```

Конструктор множества. Конкретные значения множества задаются с помощью конструктора множества, представляющего собой список элементов, заключенный в квадратные скобки. Сами элементы могут быть либо константами, либо выражениями базового типа.

Приведем несколько примеров задания множеств с помощью конструктора:

[3, 4, 7, 9, 12] — множество из пяти целых чисел;

[1. .100] — множество целых чисел от 1 до 100;

[‘a’,’b’,’c’] — множество, содержащее три литеры a, b, c;

[‘ a' ..’z',' ?','!'] — множество, содержащее все строчные

латинские буквы, а также знаки ? и !.

Символом «[ ]» обозначают пустое множество, т.е. множество, не содержащее никаких элементов.

В качестве элементов в изображении множеств допускается использовать константы и выражения, тип которых совместим с базовым типом.

Типизированная константа - множество задается в виде правильного конструктора множества, например:

```
Type

Type_month=(Jn,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec);

TDays=Set of 1..31;

Tmonth=Set of 1..12;

Tsym=Set of 'A'..'Z';

Tmno=Set of Type_month;

Const

SymMno:Tsym=['A','E','I','O','U']; {подмножество гласных букв}

DaysMno:TDays=[1,8,15,22,29]; {подмножество выходных дней месяца}

Spring_Mes:Tmonth=[3,4,5]; {подмножество весенних месяцев года}

Spring_Month:Tmno=[Mar,Apr,May]; {тоже, что и предыдущее}
```

Над множествами определены следующие операции:

1. * - пересечение множеств: результат содержит элементы, общие для обоих множеств. Например: пусть имеется описание:

```
Var S1,S2,S3,S4,S5:Set of 1..10;

Begin

S1:=[1,3,4,6];

S2:=[2,4,5,1];
```

S3:=S1*S2; - в S3 будет содержаться [1,4].

2)+ - объединение множеств : результат содержит элементы первого множества, дополненные недостающими элементами из второго множества:

S4:=S1+S2; - в S4 будет содержаться [1,3,4,6,2,5].

3)− - разность множеств: результат содержит элементы из первого множества, которые не принадлежат второму: S5:=S1-S2; - в S5 будет содержаться [3,6].

4)= - проверка эквивалентности (или равенства): возвращает TRUE, если оба множества эквивалентны, т.е. содержат все одинаковые элементы.

5)<> - проверка неэквивалентности (или неравенства): возвращает TRUE, если оба множества неэквивалентны, т.е. содержат неодинаковые элементы.

6)<= - проверка вхождения: возвращает TRUE, если первое множество включено во второе (т.е. все элементы первого множества присутствуют также и во втором).

7)>= - проверка вхождения: возвращает TRUE, если второе множество включено в первое.

8) IN - проверка принадлежности элемента множеству. Эта операция возвращает результат TRUE, если элемент (или выражение), стоящий слева принадлежит множеству, указанному справа.

Пример проверки принадлежности элемента x хотя бы одному из трех множеств M11, M12, M13

```
if( x in M11) or ( x in M12) or ( xin M13)
 then writeln('Да')
 else writeln('Нет');
```

Пример проверки принадлежности элемента x всем трем множествам: M11, M12, M13.

```
if( x in M11) and ( x in M12) and ( xin M13)
 then writeln('Да')
 else writeln('Нет');
```

Пример Дана символьная строка. Требуется определить в ней количество знаков препинания (.— ,*:!?). Программа для решения данной задачи

```
Program P1;
Var S: String; I, K: Byte;
Begin
  ReadLn(S); K := 0;
  For I := 1 To Length(S) Do
  If S[I] In ['.', '-', ',', ';', ':', '!', '?']
  Then K := K + 1;
  WriteLn('Число знаков препинания равно ', K)
End.
```

Рассмотрим процедуру INSET, позволяющую вводить значения в множество :

```
Type SetChar : Set Of Char;
Procedure INSET(Var M : SetChar);
Var I, N : Byte; C : Char;
Begin
  Write('Укажите размер множества: '); ReadLn(N);
  M := [];
  For I := 1 To N Do
  Begin
      Write(I:1,'-й элемент: '); ReadLn(C);
      M := M + [C]
  End;
  WriteLn('Ввод завершен!')
End;
```

В основной программе для ввода значений в множество, например с именем SIM, достаточно записать оператор INSET(SIM). В результате произойдет диалоговый ввод значений

Пример Даны две символьные строки, содержащие только строчные латинские буквы. Требуется построить строку S3, в которую войдут только общие символы SI и S2 в алфавитном порядке и без повторений.

Программа для решения данной задачи:

```
Program P2;
Type Mset = Set Of 'a'..'z';
Var S1, S2, S3 : String;
    MS1, MS2, MS3 : Mset;
    C : Char;
Procedure SM(S : String; Var MS : Mset);
{Процедура формирует множество MS, содержащее все
символы строки S}
Var I: Byte;
Begin MS := [];
  For  I := 1 To Length(S) Do
       MS := MS + [S[I]]
End;
Begin {Ввод исходных строк}
  ReadLn(S1); ReadLn(S2);
{Формирование множеств MS1 и MS2 из символов строк
S1 и S2}
  SM(S1, MS1); SM(S2, MS2);
{Пересечение множеств - выделение общих элементов в
множество MS3}
  MS3 := MS1 * MS2;
{Формирование результирующей строки S3}
  S3 := '';
  For C := 'a' To 'z' Do
      If C In MS3 Then S3 := S3 + C;
  WriteLn('Результат: ', S3)
End.
```

Тема 1.14. Использование подпрограмм в программе

Понятие подпрограмм, виды подпрограмм. Правила описания и обращения к функции.

Правила описания и обращения к процедуре. Формальные и фактические параметры, их виды и использование.

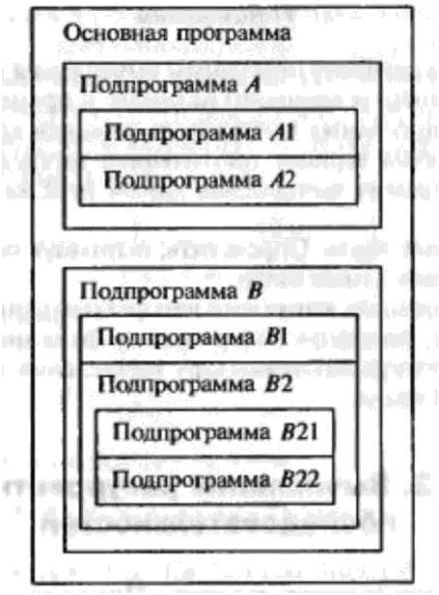
Подпрограмма - это часть программы, самостоятельно оформленная, снабженная именем, к которой можно неоднократно обращаться. "Вызов" подпрограммы, то есть выполнение действий, заданных в подпрограмме в виде операторов, может быть произведен в некоторой точке программы путем указания имени этой подпрограммы.

Любая подпрограмма может использоваться лишь и пределах области действия ее описания. Например, область действия подпрограмм *A* и *B* — основная программа, поэтому из основной программы можно обратиться к подпрограммам *A* и *B*. В свою очередь, в подпрограмме *B* могут быть обращения к подпрограмме *A,* однако из *A* нельзя обратиться к *B,* поскольку описание *A* предшествует описанию *B*. Подпрограммы *A1* и *A2* локализованы в под программе *A* и могут использоваться только в ней, причем из *A2* можно обратиться к *A1,* а из *A1* в *A2* нельзя.

Из подпрограммы *B1* можно обратиться к подпрограмме *A,* поскольку ее описание является глобальным по отношению к *B1,* по нельзя обратиться к *A1,*поскольку область действия описания *A1* не распространяется на блок подпрограммы *B*.

Из подпрограммы B22 можно обратиться только к подпрограммам B21, *B1* и A (Определите самостоятельно, почему.)

Если одно и то же имя описано во внешнем блоке (глобально) и внутреннем (локально), то последнее описание (локальное) перекрывает первое в пределах внутреннего блока.



Пример структуры программы

Структура подпрограммы почти буквально повторяет структуру всей Pascal-программы.

В языке Pascal имеется **два вида подпрограмм - процедуры и функции**. Имея один и тот же смысл и аналогичную структуру, процедуры и функции различаются назначением и способом их использования.

Функция – это подпрограмма, которая возвращает **одно значение** (т.е. используется для вычисления определенного значения).

Общий вид функции (структура функции):

Function <имя функции> [(<список параметров>)]:<тип результата>;

<раздел описаний>;

Begin

….

…. <тело функции>

<имя функции>:=результат;

End;

где Function – зарезервированное слово;

<список параметров> – перечень входных параметров или аргументов с указанием их типов; список параметров может отсутствовать;

<тип результата> – тип вычисляемого значения функции.

Результат выполнения функции **сохраняется с именем самой функции.** Поэтому в теле любой функции должен находиться, по крайней мере, один оператор, который присваивает имени функции значение результата этой функции:

Имя функции:=Результат;

Результат работы функции передается в основную программу.

Обращение к функции выполняется в каком-либо операторе. Для обращения к функции необходимо указать ее имя со списком фактических параметров (если они описаны в функции).

Формальные параметры функции — это переменные, которые указываются в заголовке функции.

Фактические параметры функции — это переменные и значения, которые указываются при вызове функции.

Составить программу вычисления функции: .

Список используемых переменных

Входные данные: x, u, z – произвольные числа (вещественного типа)

Выходные данные: y – значение функции (вещественного типа)

```
Program func;

Uses crt;

Var x,u,z,y:real;

Function step(a:real):real; {заголовок функции}

Begin {начало функции}

step:=sqr(a)-a

End; {конец функции}

{основная программа}

Begin

Clrscr;
```

Write(‘Введите числа x,u,z:’);

Readln(x,u,z);

Y:=exp(step(x)+1)+exp(step(u)+4)+exp(step(z)-2); {обращение к функции step}

Writeln(‘y=’,y:8:2);

Readln

End.

Тестовый пример

Если x=0,12; u=0,3; z=0,17, то значение функции будет равно 46,81979.

Протокол выполнения программы

Введите числа x, u, z: 0.12 0.3 0.17

y=46.82

Например, функцию определения наибольшего из двух заданных вещественных чисел можно описать следующим образом:

```
Function Max(X, Y : Real): Real;
Begin
  Max := X;
  If X > Y Then Exit Else Max := Y
End;
```

Вычисление факториала и возведение в степень.

Function stepen (a: real;n:word):***real***;

Begin

{ Если степень числа а равна 0, то результат – число 1.}

If n=0 ***then*** stepen: =1

Else

{ Иначе для возведения в степень число а надо умножить на а в степени n-1.}

Stepen: = a*stepen(a, n-1)

End;

Function factorial (n:word) :***real***;

Begin

{Если N=0 или 1, то факториал равен 1.}

If n <=1 ***then*** factorial: =1

Else

{Иначе факториал числа N рассчитывается так N*factorial(N-1).}

Factorial: = N*factorial (N-1)

End;

Процедура – это подпрограмма, которая может рассчитывать **любое количество** значений или выполнять определенные действия.

Общий вид процедуры (структура процедуры):

Procedure <имя> [(<список формальных параметров>)];

<раздел описаний>;

Begin

<тело процедуры>

End;

где **Procedure** – зарезервированное слово;

<список формальных параметров> – перечень формальных параметров с указанием их типов; список параметров может отсутствовать.

Вызов процедуры оформляется в виде отдельного оператора обращения к процедуре, который имеет вид:

<имя процедуры> [(<список фактических параметров>)];

Формальные параметры процедуры — это переменные, которые указываются в заголовке процедуры.

Фактические параметры процедуры — это переменные и значения, которые указываются при вызове процедуры.

В программе различают **глобальные** и **локальные** переменные.

Глобальные переменные – это переменные, которые действуют (доступны) во всей программе (в том числе и в подпрограммах).

Глобальные переменные объявляются в описании основной части программы.

Локальные переменные – это переменные, которые действуют (доступны) в пределах той подпрограммы, в которой они описаны. В других частях программы они не существуют.

Переменная существует с момента ее описания.

Процедуры и функции обязательно должны быть описаны в **разделе описаний**. Структура подпрограммы повторяет структуру программы, т.е. состоит из заголовка, блока описаний и тела процедуры или функции.

Например, заголовок: **Procedure** Max (a,b :real; var c:real);

Function Factorial (n:integer): integer;

В заголовке подпрограммы может быть задан **список формальных параметров**. Он указывается в круглых скобках после имени процедуры или функции**. При обращении к процедуре или функции формальные параметры заменяются фактическими**. Типы формальных параметров обязательно обозначаются идентификаторами. Например, Max (5,y,m) – обращение к процедуре; R:=Factorial(6) – обращение к функции.

Правила соответствия между формальными и фактическими параметрами следующие : **соответствие по количеству, по последовательности и по типам**

Пример. Описать процедурой вычисление суммы квадратов двух чисел. Ввести с клавиатуры x,y,z,t. Вычислить S1=x^2+y^2, S2=z^2+t^2, вывести S1 и S2 на экран.

```
var x,y,z,t,S1,S2:real;
procedure SUM_KV(a,b:real; var sum:real);
BEGIN
  sum:=sqr(a)+sqr(B);
END;
begin
  writeln( 'введи x,y,z,t ');
  readln(x,y,z,t);
  SUM_KV(x,y,S1);
  SUM_KV(z,t,S2);
  Writeln ('S1=',S1:6:2,' S2=',S2:6:2);
end.
```

Тема 1.15. Вычисление рекуррентных последовательностей

Понятие рекуррентной последовательности.

Правила вычисления рекуррентной последовательности.

Рекурсия может использоваться в различных задачах, например при вычислении сумм числовых рядов различных видов. Её смысл в том, что для получения результата надо вычислять следующий член ряда через предыдущий.

Рассмотрим ***вычисление суммы бесконечно убывающего ряда с заданной точностью.*** Бесконечно убывающий ряд – это такой, в котором каждый следующий член меньше предыдущего. Для вычисления значения следующего члена ряда надо определить закон изменения членов ряда.**Для этого записывают формулу n-го и n–1-го членов ряда, а затем вычисляют приращение (коэффициент) как частное от деления n-го на n–1-й член:**

Вид формул для этих членов ряда находят из формулы самого ряда.

Для вычисления суммы ряда используют **цикл с предусловием,** так как неизвестно, на каком члене ряда закончатся вычисления для заданной точности.

Подробные выкладки следующих примеров см. в учебнике И.Г. Семакин, А.П. Шестаков «Основы алгоритмизации и программирования», Москва, ИЦ Академия, 2017 стр. 106-109

Пример 2.4. Требуется вычислить *n*-й элемент арифметической прогрессии (2.1).

Программа, решающая данную задачу, следующая:

```
Var N, I: 0..MaxInt;
  A : Real;
Begin
  Write('N = ');
  ReadLn(N);
  A := 1;
  For  I := 2 To N Do
       A := A + 2;
  WriteLn('A(', N:1, ')= ', A:6:0)
End.
```

Рекуррентная формула $a_i = a_{i-1} + 2$ перешла в оператор $A := A + 2$.

$$a_1 = 1,\ a_2 = 3,\ a_3 = 5,\ a_4 = 7,\ a_5 = 9,\ \ldots; \qquad (2.1)$$

Пример 2.5. Требуется просуммировать первые n элементов геометрической прогрессии (2.2), не используя формулу для суммы прогрессии.

Программа решения данной задачи следующая:

```
Var N, I: 0..MaxInt;
  A, S : Real;
Begin
  Write('N = '); ReadLn(N);
  A := 1;
  S := A;
  For I := 2 To N Do
  Begin
      A := 2 * A;
      S := S + A
  End;
  WriteLn('Сумма равна ', S:6:0)
End.
```

$$a_1 = 1,\ a_2 = 2,\ a_3 = 4,\ a_4 = 8,\ a_5 = 16,\ \ldots \qquad (2.2)$$

Пример 2.6. Требуется вывести на печать первые $n (n \geq 3)$ чисел Фибоначчи и подсчитать, сколько среди них четных чисел.

Программа решения данной задачи следующая:

```
Var N, I, K, F, F1, F2 : 0..MaxInt;
Begin
  F1 := 1; F2 := 1;
  K := 0;
  WriteLn('F(1) = ', F1, ' F(2) = ', F2);
  For I := 3 To N Do
  Begin
       F := F1 + F2;
```

```
        WriteLn('F(', I : 1, ') = ', F);
        If Not Odd(F) Then K := K + 1;
        F1 := F2; F2 := F
    End;
    WriteLn('Количество четных чисел в последователь-
    сти равно ', K)
End.
```

Пример 2.7. Для заданного вещественного x и малой величины ε (например, $\varepsilon = 0{,}000001$) требуется вычислить сумму ряда

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots,$$

Если слагаемые в этом выражении обозначить

$$a_0 = 1;\ a_1 = x;\ a_2 = \frac{x^2}{2!};\ a_3 = \frac{x^3}{3!},\ \ldots,$$

то обобщенная формула для i-го элемента будет следующей:

$$a_i = \frac{x^i}{i!}.$$

Нетрудно увидеть, что между элементами данной последовательности имеется рекуррентная зависимость. Ее можно найти интуитивно, но можно и вывести формально. Правда, для этого требуется догадаться, что рекурсия — одношаговая, и что каждый следующий элемент получается посредством умножения предыдущего на некоторый множитель, т.е.

$$a_i = Ka_{i-1}.$$

Используя обобщенную формулу, запишем:

$$\frac{x^i}{i!} = K\frac{x^{i-1}}{(i-1)!},$$

откуда

$$K = \frac{x^i/i!}{x^{i-1}/(i-1)!} = \frac{x}{i}.$$

Действительно,

$$a_0 = 1;\ a_1 = a_0\frac{x}{2};\ a_3 = a_2\frac{x}{3}\ \ldots$$

Следовательно, данная рекуррентная последовательность может быть описана следующим образом:

$$a_i = \begin{cases} 1, & \text{если } i = 0; \\ a_{i-1}\dfrac{x}{i}, & \text{если } i > 0. \end{cases}$$

```
Var A, X, S, Eps : Real;
    I : Integer;
Begin
  Write('X = '); ReadLn(X);
  Write('Epsilon = '); ReadLn(Eps);
  A := 1; S := 0; I := 0;
  While Abs(A) > Eps Do
  Begin
    S := S + A;
    I := I + 1;
    A := A * X/I
  End;
  WriteLn('Сумма ряда равна ', S : 10 : 4)
End.
```

Степенной ряд имеет вид *y*=. Степенные многочлены при вычислении их суммы приводятся к виду ***по схеме Горнера.*** Используя эту схему, можно избежать возведения в степень переменных, заменив это действие умножением. Это повышает точность вычислений и упрощает алгоритмы. В вычислениях применяют прием рекурсии, который состоит в постепенном вычислении *y.* В алгоритме начальное значение , Для следующих значений используется формула .

По схеме Горнера многочлен вида приводится к виду . Для удобства работы с коэффициентами членов ряда их вводят как массив, количество элементов в массиве всегда на 1 больше наибольшей степени *x*. Если в степенном ряду нет каких-то значений степени *x*, то в ряд добавляют члены с недостающими степенями x и коэффициентами, равными нулю. Это необходимо для получения заданной степени *x*.

Пример. *Вычислить полином Pn(xf) по схеме Горнера при X={0.5, 1.5, 3.2, 4.1, 6.8}, A={-0.5, 0.8, 1.1, 2.7, 3.5, 1.9, 5.1}, j=1+5, n=6,*
Pn(x)=(…(A(1)X+A(2))X+A(3))X+…+A(n))X+A(n+1)).

```
const n=6;

a:array[1..n+1] of real=(-0.5,0.8,1.1,2.7,3.5,1.9,5.1);

function Pn(x:real):real;

var i:integer;

begin

Result:=a[1];
```

```
For i:=2 to n do
Result:=Result*x+a[i];
Result:=Result+a[n+1];
end;
var x:real;
begin
x:=0.5;
Writeln(x:1:1,' ',Pn(x):8:5);
x:=1.5;
Writeln(x:1:1,' ',Pn(x):8:5);
x:=3.2;
Writeln(x:1:1,' ',Pn(x):8:5);
x:=4.1;
Writeln(x:1:1,' ',Pn(x):8:5);
x:=6.8;
Writeln(x:1:1,' ',Pn(x):8:5);
Readln;
end.
```

```
{2. Изменить вид представления степенного ряда y = 6x^6 - 4x^4 + 5x^3 + 9x^2 + 12
в соответствии со схемой Горнера.
Составить программу для вычисления суммы степенного ряда по схеме Горнера.}
var y, x: real;
i: integer;
a: array[1..7] of real;
begin
  writeln('Введите коэффициенты уравнения');
  for i:= 1 to 7 do
  readln(a[i]);
  writeln('Введите целочисленный X');
  readln(x);
  y:=a[1];
  for i:= 2 to 7 do
  begin
    y:= y*x+a[i];
  end;
  writeln('Сумма степенного ряда:= ', y:0:2);
end.
```

Рекурсия – это прием вычисления значения задачи с помощью обращения её к самой себе. Этот прием можно применять к задачам, которые можно свести к подзадаче того же типа, но более низкого уровня. Например, такие задачи, как вычисление N! или X^N. Их можно свести к подзадачам с более низким уровнем: N! =(N-1)!*N; $X^N = X^{N-1}*X$.

Тема 1.16. Работа с файлами в Паскале

Понятие файла, создаваемого в среде Паскаля, виды файлов. Понятие файловой переменной, описание файлов и файловых переменных.

Операции с файлами: установочные, ввода-вывода информации, перемещения по файлу.

В языке Pascal **под ФАЙЛОМ понимается именованная область памяти на внешнем запоминающем устройстве, способная хранить некоторую совокупность информации.** В эту область внешней памяти можно как поместить определенные данные, так и извлечь их из нее. Эти действия имеют общее название ввод-вывод.

Файл в языке Pascal представляет собой последовательность элементов одного типа**. В отличие от массива длина файла не задается, место элемента не определяется индексом.** Данные могут храниться на диске как символьные или во внутреннем коде машины (двоичные файлы).

Любой файл имеет 3 характерные особенности:

1. файл имеет *имя*;
2. файл содержит *компоненты одного типа*; типом компонент файла может быть любой тип, кроме файлов;

3. *длина* вновь создаваемого файла никак не оговаривается при его объявлении и *ограничивается только емкостью устройств внешней п*амяти.

В Паскале понятие файла имеет два значения:

- поименованная информация на внешнем устройстве (внешний файл);
- переменная файлового типа в Паскаль-программе (внутренний файл).

В программе между этими объектами устанавливается связь. Вследствие этого все, что происходит в процессе выполнения программы с внутренним файлом, дублируется во внешнем файле. С элементами файла можно выполнять только две операции: считывать из файла, записывать в файл.

Файловый тип переменной — структурированный тип, представ-ляюший собой совокупность однотипных элементов, число которых заранее (до исполнения программы) не определено.

Структура описания файловой переменной имеет вид :

```
Var <имя переменной> : File Of <тип элемента>;
```

Здесь <тип элемента> может быть любым, кроме файлового. Например :

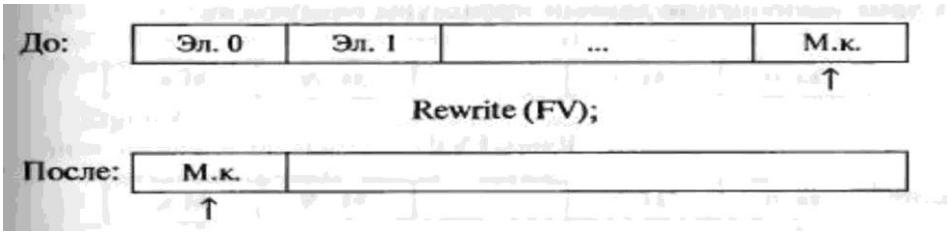
```
Var Fi : File Of Integer;
    Fr : File Of Real;
    Fc : File Of Char;
```

Файл можно представить как последовательную цепочку эле-нтов (Эл.), пронумерованных от О, заканчивающуюся специ-ьным кодом, называемым *маркером конца* (М.к.):

Число элементов, хранящихся в данный момент в файле, называется его *текущей длиной.* Существует специальная ячейка памяти, которая хранит адрес элемента файла, предназначенного для текущей обработки (записи или считывания). Этот адрес называется *указателем,* или *окном,* файла.

Для того чтобы начать запись в файл, его следует *открыть для юписи.* Это обеспечивает процедура Rewrite(FV); (где FV — имя файловой переменной). При этом указагсль устанавливается на начало файла. Если в файле до этого была информация, она исчезает.

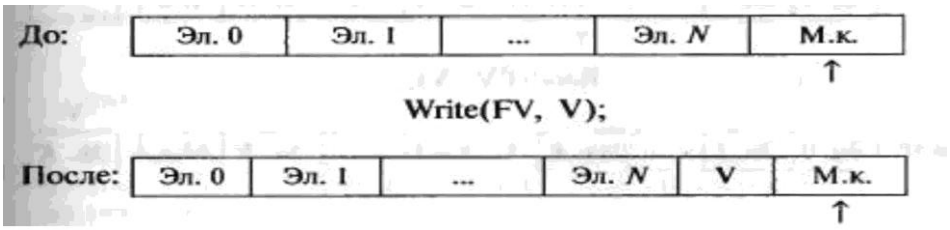
Схематически выполнение процедуры Rewrite можно представить в следующем виде:



Стрелкой снизу отмечается позиция указателя.

Запись в файл осуществляется процедурой Write (FV, V); (где V — переменная того же типа, что и файл FV). Запись происходит в том месте, на которое установлено окно (указатель файла). Сначала записывается значение, затем указатель смещается в следующую позицию. Если новый элемент вносится в конец файла, сдвигается маркер конца.

Схема выполнения данной процедуры следующая:



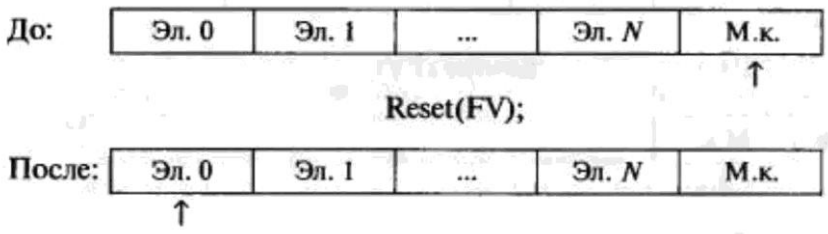
Пример Требуется в файловую переменную Fx внести 20 вещественных чисел, последовательно вводимых с клавиатуры.

Программа решения следующая:

```
Var Fx : File Of Real;
    X : Real; I: Byte;
Begin
  Rewrite(Fx);
  For I := 1 To 20 Do
  Begin
      Write('? '); ReadLn(X);
      Write(Fx, X)
  End
End.
```

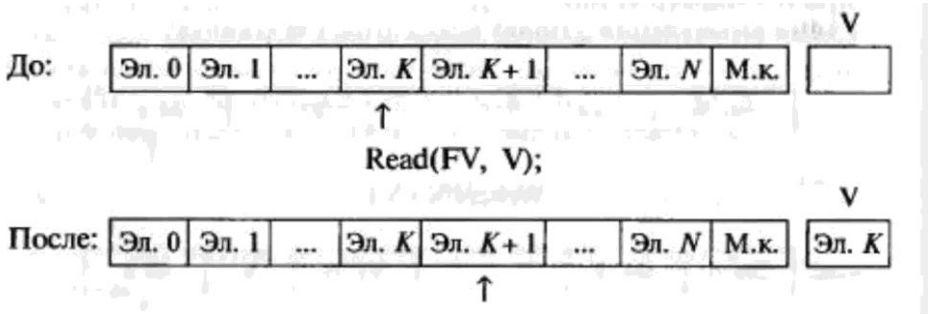
Для считывания элементов файла с его начала следует *открыть фай,1 для считывания,* что выполняется процедурой Reset (FV). В результате указатель устанавливается на начало файла. При этом вся информация в файле сохраняется.

Схема выполнения данной процедуры следующая:



Считывние из фай,ш осуществляется процедурой Read (FV, V); (где V — переменная того же типа, что и файл FV). При этом значение текущего элемента файла записывается в переменную V, а указатель смещается к следующему элементу.

Схема выполнения данной процедуры следующая:



Доступ к элементам файла может быть *последовательным* или *прямым.* В стандартном Паскале допускается только последовательный доступ к элементам.

Принцип последовательного доступа: для того чтобы прочитать л-ю запись файла, сначала следует прочитать все предыдущие записи с 1-й по (n- 1)-ю**.**

Пример Требуется в переменной *x* получить 10-й элемент вещественного файла *Fx.*

Программа решения следующая:

```
Program A;
Var Fx : File Of Real;
    X : Real;
Begin
  Reset(Fx);
  For I := 1 To 10 Do Read(Fx, X)
End.
```

Функция Eof(FV) проверяет маркер конца файла (end of file). Это логическая функция, которая получает значение True, если указатель установлен на маркер конца, и значение False — в противном случае.

Пример Требуется просуммировать все числа из фа или Fx, описанного в примере ранее

Программа решения следующая:

```
Reset(Fx);
Sx := 0;
While Not Eof(Fx) Do
Begin
  Read(Fx, X);
  Sx := Sx + X
End;
```

Для организации связи между файловой переменной и внешним файлом в ТурбоПаскале используется следующая *процедура назначения:*

```
Assign(<имя файловой переменной>,
       <идентификатор внешнего файла>).
```

Здесь <идентификатор внешнего файла> — строковая величина (константа или переменная). Например:

```
Assign(Fi, 'Number.dat');
```

После выполнения процедур Assign и Rewrite создается новый внешний файл, имя которого заносится в директорию.

Если файл открывается для считывания (Assign и Reset), то в указанном каталоге уже должен содержаться указанный внешний файл. В противном случае будет обнаружена ошибка.

```
Close(Fi).
```

Работа с файлом в программе завершается его *закрытием* с помощью следующей процедуры:

Итак, последовательность действий при создании и заполнении файла:

- описать файловую переменную;
- описать переменную того же типа, что и файл;
- произвести назначение (Assign);
- открыть файл для записи (Rewrite);
- записать в файл данные (Write);
- закрыть файл (Close).

Пример Требуется создать файл, содержащий среднесуточные температуры за некоторое количество дней. При этом не обязательно предварительно указывать количество чисел во вводимой информации. Можно договориться о каком-то условном лшчении, которое будет признаком конца ввода, например число 9999.

Программа решения следующая:

```
Program Task1;
Var Ft : File Of Real; T : Real;
Begin
  Assign(Ft, 'Temp.dat'); Rewrite(Ft);
  WriteLn('Вводите данные. Признак конца - 9999');
  ReadLn(T);
  While T <> 9999 Do
  Begin
      Write(Ft, T); Write('? '); ReadLn(T)
  End;
  WriteLn('Ввод данных закончен!');
  Close(Ft)
End.
```

В результате работы этой программы на диске будет создан файл с именем Temp.dat, ы котором сохранится введенная информация. **i**

Для последовательного считывания данных из файла требуется выполнить следующие действия:

- описать файловую переменную;
- описать переменную того же типа;
- выполнить назначение (Assign);
- открыть файл для считывания (Reset);
- в цикле считывать из файла (Read);
- закрыть файл (Close).

Пример Требуется определить среднюю температуру для значений, хранящихся в файле Temp.dat. Программа решения следующая:

```
Program Task2;
Var Ft : File Of Real;
    T, St : Real; N : Integer;
Begin Assign(Ft, 'Temp.dat');
  Reset(Ft);
  St := 0;
  While Not Eof(Ft) Do
  Begin
      Read(Ft, T);
      St := St + T
  End;
  N := FileSize(Ft);
  St := St/N;
  WriteLn('Средняя температура за ',
  N : 3, ' суток равна', St : 7 : 2,' градусов');
  Close(Ft)
End.
```

В этой программе использована следующая функция определения размера файла:

```
FileSize(<имя файловой переменной>);
```

Результат выполнения этой функции— целое число, равное текущей длине файла.

В программе файловая переменная текстового типа описывается следующим образом:

```
Var <идентификатор> : text;
```

Текстовый файл можно создать или преобразовать с помощью текстового редактора, его также можно просмотреть на экране дисплея или распечатать на принтере. В программах на Паскале для работы с текстовыми файлами наряду с процедурами Read и Write употребляются процедуры ReadLn и Write Ln.

Процедура, считывающая строку из файла с именем FV и помечающая прочитанное в переменные из списка ввода:

```
ReadLn(FV, <список ввода>).
```

Процедура, записывающая в файл FV значения из списка вывода и выставляющая маркер конца строки:

```
WriteLn(FV, <список вывода>).
```

Функция Eoln (FV) используется для обнаружения конца строки в текстовом файле (end of line — конец строки). Это логическая функция, которая принимает значение True, если указатель файла достиг маркера конца строки, и значение False — в противном случае.

Пример Пусть файл с именем Note.txt содержит некоторый текст. Требуется определить количество строк в этом тексте. Программа решения следующая

```
Var Note : Text; K : Integer;
Begin
  Assign(Note, 'Note.txt');
  Reset(Note);
  K := 0;
  While Not Eof(Note) Do
  Begin
    ReadLn(Note);
    K := K + 1
  End;
  WriteLn('Количество строк равно ', K);
  Close(Note)
End.
```

Пример Требуется в текстовом файле Note.txt определить длину самой большой строки. Программа решения следующая :

```
Var Note : Text;
    Max, K : Integer; C: Char;
Begin
  Assign(Note, 'Note.txt');
  Reset(Note);
  Max := 0;
  While Not Eof(Note) Do
  Begin
    K := 0;
    While Not Eoln(Note) Do
    Begin
      Read(Note, C);
      K := K + 1
    End;
    If K > Max Then Max := K;
    ReadLn(Note)
  End;
  WriteLn('Наибольшая строка имеет ', Max, ' знаков');
  Close(Note)
End.
```

Пример задачи с файлом прямого доступа. Считать из файла, элементы которого являются записи, содержащие целые и вещественные числа, а также строки, записи с четными номерами и посчитать, сколько в них положительных, отрицательных и нулевых целых чисел.

```
Type zap = record
        a : integer;
        b : real;
     c : string[20];
     end;
var n, z, p : byte;
        f : file of zap;
        el : zap;
begin
z := 0;
n := 0;
p := 0;
Assign (f, 'data.zap');
Reset (f);
While Not Eof (f) do
        Begin
        Seek(f, FilePos (f) + 1);
        Read (f, el);
        If el.a > 0 then p := p + 1 else
                if el.a < 0 then n := n + 1 else z := z + 1;
        end;
writeln (n + z + p : 10, 'считано чисел из файла');
writeln ('положительных: ':14,'отрицательных:':14, 'нулевых':14);
```

```
writeln (p : 14, n : 14, z : 14 );
close (f);
end.
```

Пример задачи с текстовым файлом.

В файле DAN1.dat записаны целые числа в нескольких строках. Вычислить сумму элементов каждой строки и результаты вместе с исходной информацией записать в файл DAN2.dat.

```
Var f1, f2 : text;
    X, sum : integer;
Begin
Assign (f1, 'DAN1.dat');
Assign (f2, 'DAN2.dat');
Reset (f1);
Rewrite (f2);
While not Eof (f1) do
 Begin
 Sum := 0;
  While not EoLn (f1) do
        Begin
        Read (f1, x);
        Write (f2, x:3);
        Sum := sum + x;
        End;
Write ('Результат находится в файле DAN2.dat');
Close (f1);
Close (f2);
End.
```

Постановка задачи

Дан текстовый файл, содержащий только целые числа; в каждой строке может быть несколько чисел, которые разделяются пробелами. Вывести на экран все числа с учетом разбиения на строки, подсчитать количество элементов в каждой строке и записать данные в новый файл.

Список используемых переменных:

Ish – файловая переменная для исходного внешнего файла

Result – файловая переменная для результирующего внешнего файла

i – номер текущей строки исходного файла (Ish)

X – число, содержащееся в исходном файле (Ish)

K – количество элементов (чисел) текущей строки в исходном файле (Ish)

Текст программы на Turbo Pascal:

```
Program DemoText;
var Ish,Result:text;
i,x,k:integer;
Begin
clrscr;
Assign(Ish,'fox.txt'); {связь с внешним файлом}
Assign(Result,'Res.txt');
reset(Ish); {открываем файл Ish для чтения}
rewrite(Result); {открываем файл Result для записи}
i:=0; {обнуляем номер строки}
while not Eof(Ish) do {пока не конец файла Ish}
begin
k:=0; {начальное количество элементов строки}
inc(i); {значение текущей строки}
while not Eoln(Ish) do {пока не конец строки}
begin
read(Ish,x); {считываем очередное число}
write(x,' '); {выводим число на экран}
inc(k); {увеличиваем счетчик элементов строки}
end;
writeln; {на экране переходим на новую строку}
writeln(Result,'В ',i,'-й строке ',k,' элементов');
{записываем сообщение о количестве элементов в строке
в файл Result}
```

```
readln(Ish); {переходим к следующей строке файла Ish}
end;
close(Result); {закрываем файл Result}
close(Ish); {закрываем файл Ish}
end.
```

Отладочный пример

Пусть в файле содержится следующая информация:

```
-32 16 0 8 7
4 5 9 13 11 -5 -8
6 -8 0 12
5 4 3 2 1 12
1 2
-1 -2 -4
-1 -2 4
```

Этот файл можно создать в среде Turbo Pascal таким образом:

- создайте новый файл (команда New меню File)
- запишите все числа по строкам через пробелы
- сохраните этот файл, например под именем ‘fox.txt’

Теперь данные из этого файла можно использовать в программе.

В файле результата должны быть следующие значения:

В 1-й строке 5 элементов

В 2-й строке 7 элементов

В 3-й строке 4 элементов

В 4-й строке 6 элементов

В 5-й строке 2 элементов

В 6-й строке 3 элементов

В 7-й строке 3 элементов

Протокол выполнения программы

-32 16 0 8 7

4 5 9 13 11 -5 -8

6 -8 0 12

5 4 3 2 1 12

1 2

-1 -2 -4

-1 -2 4

В файле с именем Res.txt следующие данные:

В 1-й строке 5 элементов

В 2-й строке 7 элементов

В 3-й строке 4 элементов

В 4-й строке 6 элементов

В 5-й строке 2 элементов

В 6-й строке 3 элементов

В 7-й строке 3 элементов

Файл Res.txt можно просмотреть в среде Turbo Pascal командой Open меню File.

Тема 1.17. Указатели и динамические структуры

Виды оперативной памяти, статические и динамические переменные. Понятие указателя, операции с указателями

Понятие списков, их разновидности.

Все переменные, использующиеся в программе и описанные в разделе **VAR**, размещаются в оперативной памяти перед выполнением программы и постоянно там находятся. Эти переменные называются ***статическими***. Статические переменные разделяются на глобальные (то есть переменные, описанные во внешнем блоке программы) и локальные (они описаны в подпрограммах).

Переменные, созданием и уничтожением которых может явно управлять программист, называются ***динамическими*** переменными.

В Паскале существует и другой способ выделения па мши под данные — динамический. *Динамическими называются величины, память под которые отводится во время выполнения программы.*

Раздел оперативной памяти, распределяемый статически, называется *статической* памятью; динамически распределяемый раздел памяти называется *динамической* памятью.

Использование динамических величин предоставляет программисту ряд дополнительных возможностей. Во-первых, подключение динамической памяти позволяет увеличить объем обрабатываемых данных. Во-вторых, если потребность в каких-то данных отпала до окончания программы, то занятую ими память можно освободить для другой информации. В-третьих, использование динамической памяти позволяет создавать структуры данных переменного размера.

Работа с динамическими величинами связана с использованием еще одного типа данных — ссылочного.

Указатели. Величины, имеющие ссылочный тип, называют *указателями.*

Указатель содержит в динамической памяти адрес поля, в котором хранится величина определенного типа. Сам указатель располагается в статической памяти.

Адрес — это номер первого байта поля памяти, в котором располагается величина. Размер поля однозначно определяется типом.

Величина ссылочного типа (указатель) описывается в разделе описания переменных следующим образом:

```
Var <идентификатор> : <ссылочный тип>;
```

В стандартном Паскале каждый указатель может ссылаться на исличину только одного определенного типа, который называется *базовым* для данного указателя. Имя базового типа указывается в описании в следующей форме:

Схема действия указателя

```
<ссылочный тип> := ^<имя типа>;
```

Приведем примеры описания указателей:

```
Type Massiv = Array[1..100] Of Integer;
Var  P1 : ^Integer;
     P2 : ^Char;
     Pm : ^Massiv;
```

Здесь P1 — указатель на динамическую величину целого типа; P2 — указатель на динамическую величину символьного типа; PM — указатель на динамический массив, тип которого задан в разделе Type.

Сами динамические величины не требуют описания в программе, поскольку во время компиляции память под них не выделяется. Во время компиляции память выделяется только под статические величины. Указатели — это статические величины, поэтому они требуют описания.

Память под динамическую величину, связанную с указателем, выделяется в результате выполнения стандартной процедуры NEW. Формат обращения к этой процедуре следующий:

```
NEW(<указатель>);
```

Считается, что выполнение этого оператора создает динамическую величину, имя которой имеет следующий вид:

```
<имя динамической величины> ::= <указатель>^.
```

Пусть в программе, в которой имеется приведенное ранее описание указателя, имеются следующие операторы:

```
NEW(P1); NEW(P2); NEW(PM);
```

Связь указателей с динамическими величинами

После их выполнения в динамической памяти оказывается выделенным место под три величины (две скалярные и один массив), имеющие соответственно идентификаторы **P1^, P2^, PM^**.

Кроме процедуры NEW значение указателя может определяться следующим оператором присваивания:

```
<указатель> := <ссылочное выражение>;
```

В качестве ссылочного выражения можно использовать:

• указатель;

- ссылочную функцию (т.е. функцию, значением которой является указатель);
- константу Nil.

Nil — это зарезервированная константа, обозначающая пустую ссылку, т.е. ссылку, которая ни на что не указывает. При присваивании базовые типы указателя и ссылочного выражения должны быть одинаковыми. Константу Nil можно присваивать указателю с любым базовым типом.

Ссылочная переменная до присваивания ей значения (с помощью оператора присваивания или процедуры NEW) является неопределенной.

Ввод и вывод указателей не допускается.

Если динамическая величина теряет свой указатель, она становится «мусором». В программировании так называют не нужную информацию, занимающую память

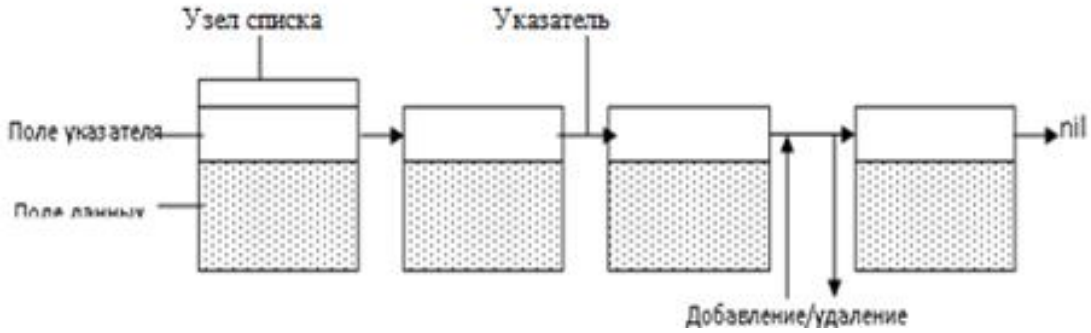
В Паскале имеется стандартная процедура, позволяющая освобождать память от данных, потребность в которых отпала. Ее формат

```
DISPOSE(<указатель>);
```

Например, если динамическая переменная P^ больше не требуется, оператор DISPOSE(P) удалит ее из памяти, после чего значение указателя P станет неопределенным. Особенно существенный эффект экономии памяти получают при удалении больших массивов данных.

Для определения размера необходимой памяти можно воспользоваться стандартной функцией **SizeOf**. Функция **MemAvail**, которая возвращает суммарный размер всех свободных областей динамической памяти. Функция **MaxAvail** определяет наибольшую свободную область.

С **динамическими переменными** можно создавать различные структуры, например, ***связные списки*****.** Связной список представляет **собой цепочку записей-узлов, в которой каждая запись содержит основные данные и ссылку на следующую запись в цепочке**. Во главе списка находится указатель («корень»), который указывает на первую запись в списке. Указателю в последней записи обычно присваивается значение nil, служащее признаком конца списка. Такая структура называется динамической, она может изменяться в процессе выполнения программы.



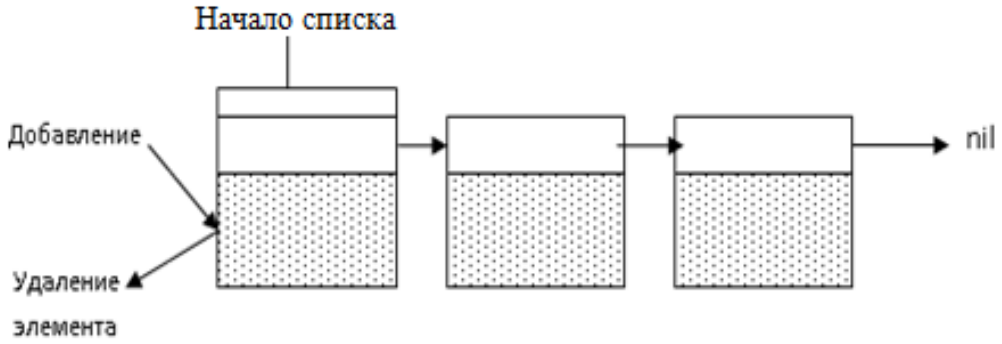
Основными операциями, которые можно выполнять над списками, являются операции включения записи в список и удаления ее из списка. Чтобы добавить в связный список новый узел, достаточно изменить один указатель, сами узлы при этом перемещаться не должны. Удаление узла также осуществляется изменением соответствующего указателя так, чтобы он ссылался на узел, следующий за удаляемым

Принято выделять следующие **типы связных списков**:

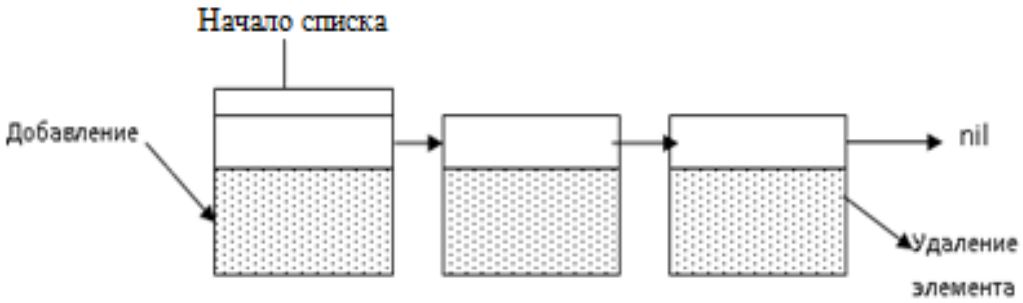
- односвязные линейные списки;
- односвязные циклические списки;
- двусвязные линейные списки;
- двусвязные циклические списки.

В циклическом списке последний узел содержит указатель на первый его элемент. Таким образом, получается замкнутая структура данных. Один из узлов в этом случае условно считается корневым, а пустого указателя в циклическом списке нет. В двусвязных списках каждый узел содержит два указателя — на предыдущий и последующий узлы.

Стек представляет собой частный случай списка, доступ к которому возможен только в корневой точке. Добавление или удаление нового элемента производится в начале списка.



Очередь, как и стек, является частным случаем списка. Доступ возможен только к первому и к последнему элементам списка. Данные могут быть добавлены в конец очереди и удалены из её начала. Элемент, который был добавлен в очередь первым, первым достигнет её начала. Принцип работы этой структуры: «Первый вошел – первый вышел».



Тема 1.18. Технология модульного программирования

Организация внешних подпрограмм.

Правила описания модулей, их структура. Использование модулей в

программах.

Введение внешних подпрограмм. В этом случае исходный текст каждой процедуры или функции хранится в отдельном файле и при необходимости с помощью специальной директивы компилятора включается в текст создаваемой программы.

Проиллюстрируем этот прием на примере следующей задачи целочисленной арифметики: дано натуральное число *n,* требуется найти сумму первой и последней цифр этого числа.

Для решения задачи используем функцию, вычисляющую число цифр в записи натурального числа. Например:

```
Function Digits(N : LongInt): Byte;
Var Kol : Byte;
Begin
  Kol := 0;
  While N <> 0 Do
  Begin
      Kol := Kol + 1;
      N := N Div 10
  End;
  Digits := Kol
End;
```

Сохраним этот текст в файле с расширением .inc (расширение внешних подпрограмм в Турбо Паскале), например digits.inc.

Теперь составим основную программу, решающую поставленную задачу с использованием описанных функций:

Опишем также функцию возведения натурального числа в натуральную степень (a^n):

```
Program Example1;
Var N, S : Integer;
{$I digits.inc} {Подключение внешней функции из файла
digits.inc, вычисляющей количество цифр в записи числа}
{$I power.inc} {Подключение внешней функции из файла
power.inc, вычисляющей результат возведения числа A в
степень N}
Begin
  Write('Введите натуральное число: '); ReadLn(N);
  {Для определения последней цифры числа N берется
  остаток от деления этого числа на 10, а для опре-
  деления первой цифры число N делится на число 10,
  возведенное в степень, на единицу меньшую, чем
  количество цифр в записи числа N (так как нумера-
  ция разрядов начинается с 0)}
  S := N Mod 10 + N Div Power(10, Digits(N) - 1);
  WriteLn('Искомая сумма: ', S)
End.
```

Директива компилятора {$I <имя файла>} позволяет в данное место текста программы вставить содержимое файла с указанным именем.

Файлы с расширением .inc можно накапливать на магнитном лиске, формируя таким образом личную библиотеку подпрограмм.

Внешние процедуры создаются и внедряются в использующие их программы так же, как и функции в рассмотренной задаче.

В нашей версии PascalABC данной возможности не предусмотрено. Поэтому материал носит ознакомительный характер.

Создание и использование модулей. Рассмотрим теперь структуру, разработку, компиляцию и использование модулей.

Модуль — это набор ресурсов (функций, процедур. констант, переменных, типов и т.д.), разрабатываемых и хранимых независимо от использующих их программ. В отличие от внешних подпрограмм, модуль может содержать достаточно большой набор процедур и функций, а также других ресурсов для разработки программ. В основе идеи модулыюсти лежат принципы струкгурного программирования. Существуют стандартные модули ТурбоПаскаля (SYSTEM. CRT, GRAPH и др.).

Модуль имеет следующую структуру:

```
Unit <имя модуля>; {Заголовок модуля}
Interface
  {Интерфейсная часть}
Implementation
  {Раздел реализации}
Begin
  {Раздел инициализации модуля}
End.
```

После служебного слова Unit записывается имя модуля, которое (для удобства дальнейших действий) должно совпадать с именем файла, содержащего данный модуль. Следовательно (как принято в MS DOS), имя не должно содержать более восьми символов.

В **разделе Interface** обделяются все ресурсы, которые будут в дальнейшем досгуины программисту при подключении модуля. Для подпрограмм здесь указывается лишь полный заголовок.

В **разделе Implementation** описываются все подпрограммы, которые были ранее объявлены. Кроме того, в нем могут содержаться свои константы, переменные, типы, подпрограммы, которые носят вспомогательный характер и используются для написания основных подпрограмм. В отличие от ресурсов, объявленных в разделе Interface, все, что дополнительно объявляется в Implementation, уже не будет доступно при подключении модуля. При описании основной подпрограммы достаточно указать ее имя, не переписывая полностью весь заголовок, а затем записать тело.

Раздел инициализации (зачастую отсутствующей) содержит операторы, которые должны быть выполнены сразу же после запуска программы, использующей модуль.

При разработке модуля рекомендуется следующий порядок:

1. спроектировать модуль, т.е. определить основные и вспомогательные подпрограммы и другие ресурсы;

1. описать компоненты модуля;

2. отладить каждую подпрограмму отдельно, после чего «вклеить» в текст модуля.

Например :

Сохраним текст разработанной программы в файле DROBY.PAS и откомпилируем модуль, используя внешний компилятор, поставляемый вместе с ТурбоПаскалем. Команда будет иметь вид TPC DROBY.PAS. Если в тексте нет синтаксических ошибок, получим файл DROBY.TPU, иначе будет выведено соответствующее сообщение с указанием строки, содержащей ошибку. Другой вариант компиляции: в меню системы программирования Турбо-Паскаль выбрать Compile/Destination Disk, а затем — Compile/Build.

Теперь можно подключить разработанный модуль к программе, где планируется его использование.

Как видно из приведенного примера, для подключения модуля используется служебное слово Uses, после которого указывается имя модуля. Данная строка записывается сразу же после заголовка программы. Если необходимо подключить несколько модулей, они перечисляются через запятую.

При использовании ресурсов модуля программисту совсем не обязательно знать, как работают вызываемые подпрограммы. Достаточно знать назначение подпрограмм и их спецификации, т.е. имена и параметры. По этому принципу осуществляется работа со всеми стандартными модулями. Следовательно, если программист разрабатывает модули не только для личного пользования, ему необходимо выполнить полное описание всех доступных при подключении ресурсов.

Приведем простейший пример модуля.

Данный пример касается **описаний**, связанных с датами, месяцами, днями и т.д. и не содержит разделов реализации и инициализации.

```
unit Calendar;
interface
type
Days =  (Mon,Tue, Wed,Thu,Fri, Sat, Sun);
WorScingDays = Mon.. Fri;
Months = (Jan, Fetb,Mar, Apr, May, June, July, Aug, Sept, Oct., Nov, Dec) ;
Summer = June. .Aug;
Autumn = Sep . . Nov;
Spring = Mar.. . May;
DayNo – 1.. . 31;
YearNo= 1900 . . 2000;
Date = record
Day : DayNo;
Month : Months ;
Year : YearNo
end ;
implementation
end.
```

Пример. Создать модуль, содержащий процедуру, выполняющую поиск минимального значения из двух переменных. В основной программе ввести значения A, B, C, D; найти минимальное значение из них и вывести его на экран.

```
Unit minXY;          {заголовок модуля}
Interface
Procedure min (x,y : real; var M : real);   {заголовок процедуры}
Implementation
Procedure min;              {реализация процедуры}
 Begin
If x < y then M := x else M := y;
End.
```

Основная программа:

```
Uses minXY;{подключение модуля к программе}
Var A, B, C, D, M1, M2, M3 : real;
Begin
Writeln (' введи A, B, C, D: ');
Readln (A, B, C, D);
Min (A, B, M1);     {1-е обращение к процедуре в модуле}
Min (C, D, M2);     {2-е  обращение к процедуре в модуле}
Min (M1, M2, M3); {3-е  обращение к процедуре в модуле}
Writeln ('min = ', M3 : 6 : 2);
End.
```

Имя модуля должно нести смысловую нагрузку и сообщать специалисту процесс, который он решает. Крайне нежелательно использовать в именах процедур или программ собственные человеческие имена или какую либо абракадабру.