Министерство образования Московской области

Государственное бюджетное профессиональное

образовательное учреждение Московской области

Воскресенский колледж

**Задания и методические указания по выполнению**

**контрольной работы для студентов заочного отделения**

по предмету

Основы программирования

Наименование специальности

09.02.03Программирование в компьютерных системах

Квалификация выпускника

Техник-программист

Воскресенск 2015 г.

**Разработчик(и):**

**ГБПО МО**

**«Воскресенский**

**колледж» преподаватель Вострякова А.Н**

(место работы) (занимаемая должность) (инициалы, фамилия)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(место работы) (занимаемая должность) (инициалы, фамилия)

|  |
| --- |
| Одобрено на заседании предметно-цикловой комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ дисциплин  Протокол №\_\_\_\_\_\_\_ от «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_ 20\_\_\_\_г.  Председатель ПЦК \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_ / |

**Оглавление**

**Пояснительная записка 4**

**Указания по выполнению контрольной работы 5**

**Требования к оформлению контрольной работы. 5**

**Содержание контрольной работы 6**

**Задания контрольной работы по вариантам 6**

**Основные критерии оценки выполнения контрольной работы 12**

**Список литературы 13**

**Приложение 14**

**Пояснительная записка**

Учебная дисциплина «Основы программирования» входит в цикл общепрофессиональных предметов.

В результате изучения данной дисциплины студент должен

знать:

- основные модели алгоритмов;

- методы построения алгоритмов;

- методы вычисления сложности работы алгоритмов.

уметь:

- разрабатывать алгоритмы для конкретных задач;

- определять сложность работы алгоритмов.

А также иметь представление о прикладном характере дисциплины в рамках специальности. При изучении соблюдается единство терминологии и обозначений в соответствии с действующими государственными стандартами.

Данная контрольная работа составлена в соответствии с рабочей программой по данной дисциплине и является практик ориентированной.

Выполнение практических заданий является обязательным компонентом процесса обучения программированию, стимулирует усвоение знаний и формирует устойчивые умения и навыки.

Методические рекомендации предназначены для студентов заочного отделения специальности «Программирование в компьютерных системах»

для возможности более эффективного изучения предмета в целом, повышении степени глубины знаний по предложенным темам, повышение уровня сформированности практических умений и навыков, развития способности комплексного применения знаний в знакомых и варьирующихся ситуациях.

Методические рекомендации содержит:

1. материалы по теории программирования на языке Турбо Паскаль ( описание разделов программ, описание структуры программы, описание операторов и операторных конструкций, описание стандартных алгоритмов применяемых при разработке программ и т.д. );

2. примеры решения задач на языке турбо паскаль;

3. рекомендации по выполнению и оформлению контрольной работы;

4. перечень заданий;

5. дополнительная литература.

**Указания по выполнению контрольной работы**

В соответствии с тематикой контрольной работы, требуется:

1. Ознакомится с заданием и подобрать нужные материалы для его выполнения.
2. Разработать алгоритм программы в соответствии с заданием.
3. Разработать интерфейс программы.
4. Разработать алгоритм программы на языке программирования в соответствии с разработанным алгоритмом программы.
5. Выполнить отладку программы.
6. Выполнить тестирование программы.
7. Осуществить оптимизацию программного кода.
8. Оформить содержание контрольной работы в MS Word.

**Требования к оформлению контрольной работы.**

* Контрольная работа выполняется на одной стороне листа с соблюдением межстрочных интервалов.
* Контрольная работа должна быть выполнена на стандартных листах бумаги А4 (287\*210). При размещении текста на листе соблюдаются следующие требования ГОСТ 6.38-90:

Размер левого поля - 30 мм,

Правого - 15мм,

Верхнего - 20 мм,

Нижнего - 20 мм.

* Шрифт Times New Roman, 14 пт, межстрочный интервал 1,5, выравнивание «по ширине», величина абзацного отступа 1,25 мм.
* Листы должны быть пронумерованы. Нумерация сквозная – на титульном листе номер не ставится, на последующих страницах номер проставляют в нижней части листа (справа).
* Разделы нумеруются согласно требований ГОСТ 1.5-68 арабскими цифрами и разделяются точками.
* Заголовки разделов выполняются с выравниванием абзаца «по центру» (абзацный отступ 0 мм), начиная с нового листа. Расстояние от текста до следующего заголовка должно быть 12 пунктов, а от заголовка до следующего за ним текста - 6 пунктов.
* Таблицы, рисунки, формулы нумеруются последовательно арабскими цифрами в пределах раздела, если в отчете есть на них ссылки.

**Содержание контрольной работы**

Готовая контрольная работа должна содержать текст задачи соответствующего варианта (Вариант работы определяется последний цифрой личного кода студента), листинг готовой программы с подробными комментариями, скриншоты работающей программы (первый скриншот отображает вид окна при запуске программы, второй отображает результат выполнения программы). К контрольной работе должен прилагаться цифровой носитель, содержащий исходные коды программы.

**Задания контрольной работы по вариантам**

**Задание 1.** Разработать алгоритм линейной программы на языке программирования высокого уровня вычисляющий данное выражение.

Вариант: 1

Вариант: 2

Вариант: 3

Вариант: 4

Вариант: 5

Вариант: 6

Вариант: 7

Вариант: 8

Вариант: 9

Вариант: 10

**Задание 2.** Разработать разветвляющийся алгоритм для решения задач на языке высокого уровня.

**Варианты задач** (номер задачи соответствует номеру варианта):

1. Среди трех чисел найти среднее. Остальные два сложить. Если среди чисел есть равные, вывести сообщение "Ошибка".

2. Пользователь вводит целое число. Надо вывести на экран сколько в этом числе цифр и положительное оно или отрицательное.

Например: "это однозначное положительное число".

Пусть достаточно будет определить, является ли число однозначным, двухзначным или трехзначным и более.

1. Дано число определить является ли оно простым.
2. Даны координаты точки и радиус круга с центром в начале координат. Определить, принадлежит ли данная точка кругу.
3. Треугольник существует только тогда, когда сумма любых двух его сторон больше третьей.

Дано: a, b, c – стороны предполагаемого треугольника.

Требуется сравнить длину каждого отрезка с суммой двух других. Если хотя бы в одном случае отрезок окажется больше суммы двух других, то треугольника с такими сторонами не существует.

1. Всем известна прямоугольная (декартова) система координат, в которой две перпендикулярные оси делят плоскость на четверти. В первую четверть попадают точки, у которых обе координаты (x и y) больше нуля. Во вторую: x < 0, y > 0; третью: x < 0, y < 0; четвертую: x > 0,

y < 0. Допустим, требуется написать программу, определяющую по координатам точки, в какой четверти она находится.

1. Дано трёхзначное число. Определить, является ли оно палиндромом (палиндром – число, которое читается одинаково слева на право, и справа на лево) если да, то найти сумму его цифр.
2. Даны три числа. Вывести их квадраты по убыванию.
3. Даны 4 числа. Вывести их по убыванию.
4. Даны два числа. Посчитать сколько раз в их сумме уложится их разность.

**Задание 3.** Разработать циклический алгоритм программы на языке высокого уровня.

**Варианты задач** (номер задачи соответствует номеру варианта)**:**

1. Составить таблицу значений функции y = 5 - x2/2 на отрезке [-5; 5] с шагом 0.5.
2. Вывести на экран кубы чисел от A до B.
3. Дан диапазон чисел вывести все числа, делящиеся на 4 и 3.
4. Дан диапазон чисел найти сумму всех чисел, делящихся на три.
5. Дан диапазон чисел сложит друг с другом каждое третье число.
6. Дан диапазон чисел нужно по три числа вывести в рамочку.

Рамочку можно нарисовать любыми символами.

1. Дан диапазон чисел. Нужно каждое третье число вывести в рамочку в строку.
2. Дан ряд чисел. Сложить по пять чисел между собой и вывести на экран.
3. Дан числовой ряд найти сумму всего ряда и определить на каком числе сумма превысит число, введенное пользователем специально. Вывести это число на экран.
4. Программа запрашивает числа и выводит на экран на сколько число больше или меньше предыдущего. Выход из цикла 0.

**Задание 4.** Разработать алгоритм программы для обработки одномерного массива.

**Варианты задач** (номер задачи соответствует номеру варианта)**:**

1. Даны два массива. Сложить элементы первого массива и прибавить сумму к каждому элементу второго массива.
2. Даны два массива. Прибавить каждый элемент первого массива к соответствующему элементу второго массива.
3. Даны два массива. Определить в каком массиве больше четных чисел.
4. Даны два массива. Определить в каком из них больше чисел превышающих 50.
5. Даны два массива. найти разность соответствующих элементов и сохранить ответы в третьем массиве.
6. Даны два массива нужно наибольший элемент первого прибавит ко всем элементам второго массива.
7. Дан массив. Создать массив 2(динамически) из четных чисел первого массива.
8. Даны два массива. Определить, сколько раз данное число встречается в двух массивах одновременно.
9. Даны два массива. В массиве 1 найти количество элементов, больших среднего арифметического массива 2.
10. Даны два массива. Определить, сколько раз данное число встречается в двух массивах одновременно.

**Задание 5.** Разработать функцию и вынести ее в отдельный модуль.

**Варианты задач** (номер задачи соответствует номеру варианта):

1. Функция подсчитывает площадь прямоугольника.
2. Находит произведение трех чисел и возводит в квадрат.
3. Написать функцию возведения числа в произвольную степень.
4. Написать функцию вычисляющую n!
5. Функция получает три числа, находит среднее арифметическое.
6. Функция находит тангенс числа x.
7. Функция находит котангенс числа x.
8. Функция получает 3 числа и находит наибольшее из них.
9. Функция определяет делится ли число на 2 и 3 одновременно. Если делится возвращает 1, иначе 0.
10. Функция вычисляет выражения x=(a\*b/c)^2.

**Задание 6.** Разработать алгоритм обработки строк на языке высокого уровня.

**Варианты задач** (номер задачи соответствует номеру варианта):

1. Определить сколько слов в строке.
2. Определить сколько раз встречается в сроке предлог, заданный пользователем.
3. Дана строка удалить из нее каждый третий пробел.
4. Дана строка и дана подстрока из трех символов. Определить сколько раз подстрока встречается в строке.
5. Посчитать все гласные в строке.
6. Дана строка найти символ который находится по середине.
7. Дана строка удалить все буквы а и записать в конце строки.
8. Первое слово строки записать в конце.
9. Последние слово строки написать первым.
10. Удалить слова в которых букв меньше 5.

**Основные критерии оценки выполнения контрольной работы**

|  |  |  |
| --- | --- | --- |
| **№** | **Критерии оценки** | **оценка** |
| **1** | Контрольная работа должна быть правильно оформлена, тексты программ не должны содержать логических ошибок, учитывается сложность алгоритм. | зачет |
| **2** | Во всех остальных случаях работа признается невыполненной. | незачет |

**Список литературы**

**Основные источники:**

1. Голицина О.Л., Попов И.И. Основы алгоритмизации и программирования : учебное пособие / Голицина О.Л., Попов И.И. – 3-е изд., испр. И доп. – М. : ФОРУМ, 2010. – 432 с.
2. Колдаев И. Д. Основы алгоритмизации и программирования : учебное пособие / Под редакцией проф. Л. Г Гагариной – М. : ФОРУМ, 2010. – 416 с.

**Перечень электронных источников:**

1. <http://www.programmer-lib.ru/pascal.php>
2. <http://pascal.proweb.kz/index.php?page=2>
3. <http://schools.keldysh.ru/sch887/pascal.htm>
4. <http://wiki.iteach.ru/images/b/b8/%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D0%B8_-_%D0%9F%D0%B0%D1%81%D0%BA%D0%B0%D0%BB%D1%8CFom.pdf>
5. <http://tpdn.ru/guide/pf/31/?PAGEN_1=4&sid=31>
6. <http://pascal.net.ru/%D0%A3%D1%80%D0%BE%D0%BA+%D0%B4%D0%BB%D1%8F+%D0%BD%D0%B0%D1%87%D0%B8%D0%BD%D0%B0%D1%8E%D1%89%D0%B8%D1%85>

**приложение 1**

ТУРБО ПАСКАЛЬ

**Язык программирования** - это средство реализации алгоритмов задач в виде программ на компьютере.

В алфавит Паскаля (Pascal) входит:

латинские буквы, арабские цифры, специальные символы, такие как +,-,\*, /,:, ', <, >, =, (,), {,}, [,] и т. д., а также служебные слова.

**Идентификатор** ( имя объекта) - служит для обозначения меток, констант, типов, переменных, процедур, функций.

Служебное слово –последовательность символов зарезервированных в Паскале несущие установленный смысл (and, array, var, и т.д.).

**Правила задания идентификаторов**

Идентификатор записывается:

1. латинскими буквами и цифрами но начинаться запись должна с буквы;
2. нельзя использовать знаки препинания и пробелы.
3. Можно использовать символ нижнего подчеркивания «\_». Этот символ можно использовать в начале идентификатора, в середине или в конце.

**Пример: \_DFG56; fgs\_34;**

1. Если идентификатор содержит несколько слов то их можно разделить заглавными буквами.

**Пример: StrToInt;**

**Переменная**

**Переменная** – основной элемент языка, характеризуется идентификатором (именем), значением и типом. Бывают имена (переменные) со стандартными типами и с типами даваемые программистами.

Переменная описывается в разделе **VAR:**

VAR

< идентифекатор>:<тип данных>;

Пример:

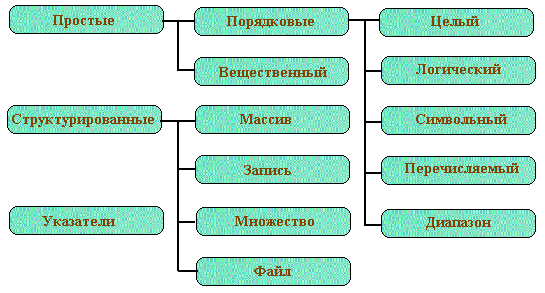
VAR

asd, b : integer;

**Типы данных**

Любые объекты, т.е. константы, переменные, значения функций или выражения, в Паскале характеризуются своими типами. Тип определяет множество допустимых значений того или иного объекта, а также множество операций, которые к нему применимы. Кроме того, тип определяет формат внутреннего представления данных в памяти ЭВМ.

  Паскаль характеризуется разветвленной структурой типов данных:



**Простые типы**

**Порядковые типы** отличаются тем, что каждый из них имеет конечное число возможных значений. Эти значения можно определенным образом упорядочить и, следовательно, с каждым из них можно сопоставить некоторое целое число - порядковый номер значения. В Паскале есть следующие порядковые типы:

**целый тип** представляет собой подмножество множества целых чисел, определяемое конкретной реализацией.

|  |  |  |
| --- | --- | --- |
| **Тип** | **Диапазон** | **Размер в байтах** |
| **Integer Byte**  **Shortint**  **Word**  **Longint** | от -32768 до +32767,  0 ... 255   -128 ... 127   0 ... 65535   -2147483648 ... 2147483647 | 2  1   1   2   4 |

**Char** - символьный тип, представляет собой набор символов, определяемый конкретной реализацией. Во внутреннем представлении он занимает 1 байт, множество значений этого типа фиксировано и упорядочено. Все символы считаются перенумерованными, начиная с нуля. Такой набор символов определен в каждой вычислительной системе. Он необходим, по крайней мере, для связи системы с внешним миром.

Константой символьного типа является один из допустимых символов, взятый в апострофы. Если апостроф сам является символом, то  апостроф, являющийся значением константы, записывается дважды, например   **‘7’ ,  ‘+’ , ‘F’ , ‘’’’ , ‘j’ , ‘?’ .**

**Boolean** - логический тип, определяет диапазон логических значений, который содержит два элемента **False (ложь) и True (истина)**. Во внутреннем представлении он занимает 1 байт.

**Перечисляемый тип** - задается перечислением тех значений, которые он может получать. Каждое значение именуется некоторым идентификатором и располагается в списке, обрамленном круглыми скобками, например:   
**Type  colors = (red, white, blue, black);**

**Тип-диапазон** - подмножество своего базового типа, в качестве которого может быть любой порядковый тип, кроме типа-диапазон. Тип-диапазон задается границами своих значений внутри базового типа

***<минимальное значение> . .<максимальное значение>***

При определении типа-диапазона нужно руководствоваться следующими правилами:

-".."  рассматриваются как один символ, поэтому между точками пробелы недопустимы;

- левая граница не должна превышать правую границу.

Пример:    **Type month = 1..12;**

**lat = ’a’ .. ’z’;**

**Вещественный тип** (дробное число)тоже имеет конечное число значений, которое определяется форматом внутреннего представления вещественного числа. Однако количество возможных значений вещественного числа настолько велико, что сопоставить с каждым из них целое число  не представляется возможным.

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип** | **Диапазон** | **Знач. цифры** | **Размер в байтах** |
| **Real Single   Double   Extended** | 1 от 2.9Е-39 до 1.7Е+38.5E-45 ... 34E38   5E-324 ... 1.7E308   1.6E-4951...1.1E4932 | 11-12  7-8   15-16   19-20 | 6  4   8   10 |

**Операции**

В таблице приведены арифметические операции, которые определены  для целого и вещественного типов:

|  |  |  |  |
| --- | --- | --- | --- |
| **Знак**  **операции** | **Операция** | **Типы**  **операндов** | **Тип**  **результата** |
| **+** | Сложение | Целый или  вещественный | Целый, если оба операнда целого типа, в противном случае - вещественный. |
| **-** | Вычитание | Целый или  вещественный |
| **\*** | Умножение | Целый или  вещественный |
| **/** | Деление | Целый или  вещественный | Вещественный |
| **Div** | Целочисленное деление | Целый | Целый |
| **Mod** | Остаток от целочисленного деления | Целый | Целый |

**В Паскале определены следующие логические операции:**

**Not**  -   логическое  **НЕ**;

**And** -   логическое  **И**;

**Or**   -   логическое  **ИЛИ**;

**Xor**  -   исключительное **ИЛИ**.

Данные операции определяются следующим образом:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **P** | **Q** | **Not P** | **P And Q** | **P Or Q** | **P Xor Q** |
| True   True   False   False | True   False   True   False | False   False   True   True | True   False   False   False | True   True   True   False | False   True   True   False |

 В Паскале используются следующие операции отношения: **=, <>, >, <, <=, >=**. Операции сравнения применимы к данным простых типов. Сравнивать можно между собой значения одного типа: целые, вещественные, символьные, логические, значения перечисляемого типа. Допустимо также сравнивать между собой целые и вещественные числа.

При вычислении выражений любого типа приоритет вычислений определяется расставленными скобками, а при их отсутствии по следующей таблице (в порядке убывания приоритета).

|  |  |
| --- | --- |
| **Приоритет** | **Операция** |
| 1  2   3   4 | Not  \*, /, Div, Mod, And  +, -, Or, Xor =,<>, <, >, <=, >= |

**Функции.**

Все  функции можно разделить на два вида:  не выводящие за пределы типа, т. е. аргументы и результат относятся к одному типу, и те, у которых тип результата не совпадает или может не совпадать с типом результата. Ниже приводится перечень часто используемых функций  Паскаля.

|  |  |  |  |
| --- | --- | --- | --- |
| Вызов функции | Тип аргумента | Тип  результата | Назначение |
| Abs(x) | целый  вещественный | целый  вещественный | Возвращает абсолютное значение x |
| Pi | - | вещественный | Возвращает значение числа ПИ |
| Sin(x) | целый вещественный | вещественный | Возвращает синус x радиан |
| Cos(x) | целый вещественный | вещественный | Возвращает косинус x радиан |
| ArcTan(x) | целый вещественный | вещественный | Возвращает арктангенс x радиан |
| Sqrt(x) | целый вещественный | вещественный | Возвращает квадратный корень из x>=0 |
| Sqr(x) | целый  вещественный | целый  вещественный | Возвращает значение квадрата x |
| Exp(x) | целый  вещественный | вещественный | Возвращает значение e в степени x |
| Ln(x) | целый  вещественный | вещественный | Возвращает натуральный логарифм x, x>0 |
| Trunc(x) | вещественный целый | целый | Возвращает целую часть значения x |
| Frac(x) | вещественный целый | вещественный | Возвращает дробную часть значения x |
| Int(x) | вещественный целый | вещественный | Возвращает целую часть значения x |
| Round(x) | вещественный целый | целый | Округляет x до ближайшего целого |
| Odd(x) | целый | логический | Проверяет нечетность значения x |
| Ord(x) | порядковый | целый | Возвращает порядковый номер, соответствующий значению x |
| Chr(x) | символьный | целый | Возвращает символ с заданным порядковым номером x |
| Pred(x) | порядковый | порядковый | Возвращает предшествующее значение x |
| Succ(x) | порядковый | порядковый | Возвращает последующее значение x |

**Составные части программы.**

Программа на Pascal состоит из двух частей: раздела описаний и раздела операторов. Для реализации программы всегда используются несколько базовых ключевых слов: (в следующем порядке):

**Program** (имя программы)

**Label** (Список меток)

**Const** (Список постоянных значений)

**Type** (Описание сложных типов данных)

**Var**  (Описание переменных программы)

**Begin** (Начало программного блока)

(Алгоритм из операторов и операндов)

**End.** (Конец программы)

**Оператор присваивания**

Оператор присваивания используется для задания значения переменных и имеет следующий синтаксис:

***имя-переменной : = выражение;***

Вычисляется выражение, стоящее в правой части оператора, после чего его значение записывается в переменную, имя которой стоит слева. Тип выражения и тип переменной должны быть совместимы, т.е. множество допустимых значений для типа выражения содержится во множестве допустимых значений для типа переменной.

Пример:

x:=3.14;

a:=b+c;

i:=i+1; (значение ячейки памяти i увеличивается на 1)

**Операторы ввода и вывода информации**

Взаимодействие устройств обработки и хранения информации с внешней средой (хотя бы с пользователем) является совершенно необходимым. За такой интерфейс в языке Паскаль отвечают операторы ввода-вывода информации. Эти инструкции позволяют ввести аргументы, параметры расчетов во время выполнения программы (а не на этапе ее написания), осуществить вывод рассчитанных данных в понятном человеку виде.

Сначала операторы ввода (форматы операторов):  
**Read**(<Список ввода>);  
**Readln**(<Список ввода>);

В таком формате эти команды позволяют вводить данные в переменные во время выполнения программы с клавиатуры. Элементами списка ввода могут быть имена переменных, которые должны быть заполнены значениями, введенными с клавиатуры.

Выполнение операторов ввода происходит так: ход программы приостанавливается, на экран выводится курсор, компьютер ожидает от пользователя набора данных для переменных, имена которых указаны в списке ввода. Пользователь с клавиатуры вводит необходимые значения в том порядке, в котором они требуются списком ввода, нажимает Enter. После этого набранные данные попадают в соответствующие им переменные и выполнение программы продолжается.

**Примечание***: данные при вводе разделяются пробелами или вводятся через Enter.*

Пример:

Read(a);

read(x1,x2,y);

Разница между работой процедур Read и Readln (от Read line) состоит в следующем: после выполнения Read значение следующего данного считывается с этой же строчки, а после выполнения Readln - с новой строки.

Для вывода информации в Паскале также есть две команды:  
**Write**(<Список вывода>);  
**Writeln**(<Список вывода>);

Такой формат использования Write и Writeln позволяет выводить на экран монитора данные из списка вывода. Элементами списка вывода могут являться имена переменных, выражения, константы. Прежде чем вывести на экран, компьютер значения выражений сначала вычислит. Элементы списка, также как и в операторах ввода, разделяются запятыми.

Различие между двумя операторами вывода таково: после выполнения оператора Writeln (от Write line) происходит переход на новую строчку, а после выполнения инструкции Write, переход на новую строчку не происходит и печать по последующим командам вывода Write или Writeln будет происходить на той же строчке. При вызове оператора Writeln без параметров просто происходит переход на новую строчку.

Пример:

Write('x=',x);

writeln('x=',x:6:2);

{число 6 означает количество позиций на изображение всего числа, а цифра 2 количество изображений дробной части}

writeln(x,2\*a-a, sqrt(x), y);

Существует так называемый форматированный вывод. Можно задать количество позиций, отводимых под число. Для целых – после выражения или переменной через двоеточие указывается меньше какого количества позиций не может быть выделено значению. Для вещественных – дополнительно через двоеточие можно указать количество цифр в дробной части. При этом происходит округление в ближнюю сторону.

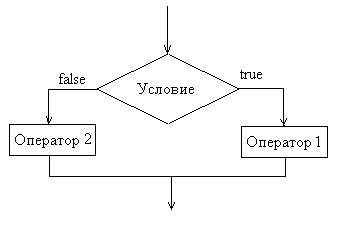
|  |  |  |  |
| --- | --- | --- | --- |
| **Процедура**  **(функция)** | **Назначение** | **Пример вызова** | **Примечания** |
| **read(*список-ввода*)** | ввод  данных (клвиатура) | read( a, b, c ) |  |
| **readln(*список-ввода*)** | ввод данных, пропуск маркера конца строки | readln( a, b, c ) |  |
| **write(*список-вывода*)** | Вывод  данных (экран) | write( ‘n = ’,n:10 )  write( ‘n = ‘,n:p:q ) | p,q: величины целого типа |
| **writeln(*списоквывода*)** | Вывод  данных, вывод маркера конца строки | writeln( ‘?’:s+2 )  writeln( ‘’:spaces ) | s,spaces: величины целого типа |
| **clrscr** | очистить весь экран (окрасить в цвет фона) | clrscr | модуль crt  (смотри приложение) |
| **textcolor(*цвет*)** | выбрать цвет символа | textcolor(magenta) | модуль crt  (смотри приложение) |
| **textbackground(*цвет*)** | выбрать цвет фона | textbackground(0) | модуль crt  (смотри приложение) |
| **window(*x1,y1,x2,y2*)** | окно (x1,y1)-(x2,y2) | window(1,1,40,10) | модуль crt  (смотри приложение) |
| **wherex** | позиция курсора x - номер колонки | x := wherex | модуль crt  (смотри приложение) |
| **wherey** | позиция курсора y - номер строки | y := wherey | модуль crt  (смотри приложение) |
| **gotoxy(*x,y*)** | установить курсор в положение ( x, y );  x - колонка [1..80];  y - строка [1..25]; | gotoxy(40,10)  gotoxy(50,wherey) | модуль crt  (смотри приложение) |

**Условный оператор**

Одной из основных алгоритмических структур является ветвление (условный оператор). С помощью условного оператора выбирается и выполняется один из операторов, входящих в его состав.

Условный оператор имеет два варианта записи.

Полная форма:



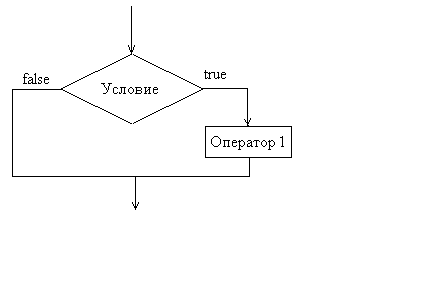
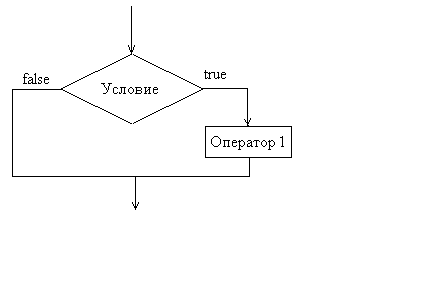
**If** <логическое выражение> **Then** <оператор1> **Else** <оператор2>**;**

В этом случае в зависимости от значения логического выражения выбирается для выполнения оператор1 или оператор2

Неполная форма:

**If** <логическое выражение> **Then** <оператор> **;**

В этом случае в зависимости от значения логического выражения выполняется записанный после **Then** оператор или не производится никаких действий.



Если после **Then** или **Else** необходимо выполнить действия, описанные несколькими операторами, то эти операторы должны быть заключены в операторные скобки **Begin - End**. Такая конструкция называется составным оператором.

С помощью условного оператора выбирается и выполняется один из операторов, входящих в его состав.

**Замечание**: *Знак «точка с запятой» не ставится перед служебным словом Else, но операторы в группах, естественно, отделяются друг от друга этим знаком.*

В программах на языке Паскаль условия представляют собой выражения, значением которых является величина логического (**Boolean**) типа. Это может быть как просто переменная указанного типа, так и сложная последовательность высказываний, связанных логическими операциями.

В простых условиях могут применяться знаки операций сравнения: >(больше), <(меньше), =(равно), <>(не равно), >=(больше или равно), <=(меньше или равно).

Примеры простых условий:

 A=5 {Значение переменной А равно 5}

(C+D3)>=(D1\*(45-2)) {Значение выражения в левой части больше либо равно значению выражения из правой части}  
    S<>'ABC' {Значение переменной S не равно строковой константе 'ABC'}

В сложных условиях добавляются логические операторы AND, OR, NOT, XOR. В таких случаях простые части условия заключаются в скобки.

Пример:

If **(a>4) AND (a<20) OR (a=0)** Then <оператор1> Else <оператор2>;

пример:

**"Из двух чисел выбрать наибольшее".**

На первый взгляд решение очевидно, но оно не столь тривиально, как кажется.

**Program** Example;  
**Var** A,B,C : **Real**; {A,B - для хранения аргументов, C - результат}  
**Begin**

**Writeln**('Введите два числа');  
 **Readln**(A,B);                            {Вводим аргументы с клавиатуры}  
 **If** A>B **Then** C:=A **Else** C:=B; {Если A>B, то результат - A, иначе результат - B}  
**Writeln**(C);                               {Выводим результат на экран}

**End.**

**Оператор выбора**

Если у вас не два возможных варианта выполнения программы, а больше, то может использоваться оператор выбора **CASE**. Структура этого оператора в Турбо Паскале:

**CASE** <ключ\_выбора> **OF**

C1 : <оператор1>;

C2 : <оператор2>;

. . .

CN : <операторN>;

[**ELSE** <оператор0>;]

**END**;

Здесь <ключ\_выбора> - это выражение порядкового типа, в зависимости от значения которого принимается решение; C1,…,CN – значения, с которыми сравнивается значение <ключа>; <оператор1>,…, <операторN> - оператор (возможно составные), из которых выполняется тот, с константой которого происходит первое совпадение значения <ключа>, <оператор0> выполнится, если значение ключа не совпадает ни с одной из констант C1,…,CN.

Ветвь **Else** не обязательна, и в отличие от оператора **if**, перед ней можно ставить точку с запятой. Если для нескольких значений <ключа> действия совпадают, то эти константы можно перечислить через запятую перед двоеточием или даже задать диапазон значений (нижняя граница .. верхняя граница).

Пример:

Вводится целое число, если это цифра, то определить четная она или нет, а если число, то определить, попадает ли оно в диапазон от 10 до 100, если нет, то выдать соответствующее сообщение*.*

**Program** chislo;

**var** i:integer;

**begin**

**write**('Введите целое число: ');

**readln**(i);

**case** i **of**

0,2,4,6,8 : **writeln**('Четная цифра');

1,3,5,7,9 : **writeln**('Нечетная цифра');

10…100,200 : **writeln**('Число от 10 до 100 или 200');

**else** **writeln**('Число либо отрицательное, либо > 100, но не 200');

**end**;

**readln**

end. **Циклические алгоритмы**

Циклом называется многократное повторение однотипных действий. Телом же цикла будем называть те самые действия, которые нужно многократно повторять.

**Существует три вида цикла, имеющих собственные операторы на языке Паскаль для их записи.** Эти виды имеют собственные условные названия: "**Пока**", "**До**", "**С параметром**". Друг от друга они несколько отличаются и используются каждый для своего класса задач.

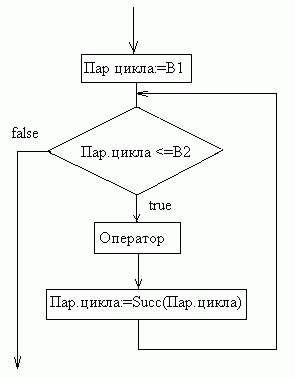
**Цикл "С параметром". Оператор цикла с параметром**

Оператор цикла с параметромимеет такую структуру:

**For** <имя\_переменной> **:=** <выр1> **To** <выр2>**Do** <оператор> **;**

<имя переменной> - параметр цикла   
<выражение1> - начальное значение параметра цикла   
<выражение2> - конечное значение параметра цикла   
Все три указанных выше объекта должны быть одного порядкового типа.   
<оператор> - тело цикла.

Этот оператор цикла выполняется следующим образом:



В данном случае параметром будет являться целочисленная переменная, которая будет изменяться на единицу при каждой итерации цикла. Таким образом, задав начальное и конечное значения для такой переменной, можно точно установить количество выполнений тела цикла.

Читается данная структура так: "Для переменной (далее следует ее имя) от начального значения до конечного выполнять оператор (являющийся телом цикла)". Иногда цикл с параметром даже называют "Для" или "For". В первом случае параметр с каждой итерацией увеличивается на единицу, во втором - уменьшается.

Выполняется этот цикл по следующему алгоритму:  
1. переменной-параметру присваивается начальное значение;  
2. выполняется тело цикла;

3. переменная-параметр автоматически увеличивается на 1 (в первом случае формата);

4. если параметр превышает конечное значение, то происходит выход из цикла, иначе - переход к пункту 2.

*Примечание*:

при использовании **Downto** параметр автоматически уменьшается на 1, а выход из цикла происходит тогда, когда параметр становится меньше конечного значения.

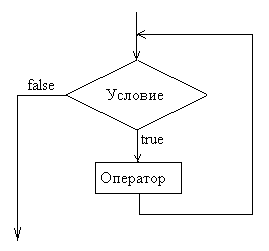
Таким образом, в отличие от первых двух видов цикла, этот цикл используется тогда, когда известно необходимое количество выполнений тела цикла.

Вообще говоря, цикл "Пока" является универсальным, то есть любая задача, требующая использования цикла, может быть решена с применением этой структуры. Циклы "До" и "С параметром" созданы для удобства программирования.

**Цикл "ПОКА". Оператор цикла с предусловием**

**While <**логическое выражение> **Do <**оператор> **;**

Здесь тело цикла выполняется до тех пор, пока логическое выражение возвращает значение **True**.



Группа операторов, называемая "телом цикла", судя по этой схеме, будет выполняться пока истинно условие цикла. Выход из цикла произойдет, когда условие перестанет выполняться.

Если условие ложно изначально, то тело цикла не будет выполнено ни разу. Если условие изначально истинно и в теле цикла нет действий, влияющих на истинность этого условия, то тело цикла будет выполняться бесконечное количество раз. Такая ситуация называется "зацикливанием". Прервать зациклившуюся программу может либо оператор (нажав Ctrl+C), либо аварийный останов самой программы, в случае переполнения переменной, деления на ноль и т.п., поэтому использовать структуру цикла следует с осторожностью, хорошо понимая, что многократное выполнение должно когда-нибудь заканчиваться.

На языке Pascal структура цикла "Пока" записывается следующим образом:

**While** <условие> **Do** <оператор>;

По-русски можно прочитать так: "Пока истинно условие, выполнять оператор". Здесь, так же как в формате условного оператора, подразумевается выполнение только одного оператора. Если необходимо выполнить несколько действий, то может быть использован составной оператор. Тогда формат оператора принимает такой вид:

**While** <условие> **Do**   
**Begin**

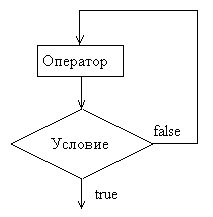
<оператор #1>;  
<оператор #2>;  
<оператор #3>;  
. . .

**End**;

**Цикл "ДО". Оператор цикла с постусловием**

**Repeat**   
            <операторы>   
**Until <**логическое выражение> **;**

Здесь тело циклавыполняется до тех пор, пока логическое выражение возвращает значение **False.**



Этот вид цикла отличается от предыдущего в основном тем, что проверка условия повторения тела цикла находится не перед ним, а после. Поэтому цикл "До" называют циклом "с постусловием", а "Пока" - "с предусловием".

Обратите также внимание на то, что новая итерация (повторное выполнение тела цикла) происходит не тогда, когда условие справедливо, а как раз тогда, когда оно ложно. Поэтому цикл и получил свое название (выполнять тело цикла до выполнения соответствующего условия).

Интересно, что в случае, когда условие цикла изначально истинно, тело цикла все равно будет выполнено хотя бы один раз. Именно это отличие "до" от "пока" привело к тому, что в программировании они не подменяют друг друга, а используются для решения задач, к которым они более подходят.

Формат цикла на языке Pascal:

**Repeat**

<оператор #1>;  
<оператор #2>;  
<оператор #3>;  
. . .

**Until** <условие>;

Читается так: "Выполнять оператор #1, оператор #2. : до выполнения условия".

Здесь не требуется использование составного оператора, потому что сами слова Repeat и Until являются операторными скобками.

Пример**:**

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

Решим эту задачу с использованием всех трех видов циклов.

***I. С использованием цикла "С предусловием".***

**Program Ex1**;  
**Var**   
   A : Integer;  
   S : Longint;  
**Begin**

A:=1; S:=0;  
**While** A<=100 Do  
**Begin**

S:=S+A\*A;  
A:=A+1

E**n**d;  
**Writeln**(S)

**End.**

***II. С использованием цикла "С постусловием".***

**Program Ex2;**  
**Var**   
    A : Integer;  
    S : Longint;  
**Begin**

A:=1; S:=0;  
Repeat

S:=S+A\*A;  
A:=A+1

Until A>100;  
Writeln(S)

**End.**

***III. С использованием цикла "С параметром".***

**Program Ex3;**  
**Var**   
    A : Integer;  
    S : Longint;  
**Begin**

S:=0;  
For A:=1 To 100 Do S:=S+A\*A;  
Writeln(S)

**End.**

**Процедуры и функции**

При решении сложных объемных задач часто целесообразно разбивать их на более простые. Метод последовательной детализации позволяет составить алгоритм из действий, которые, не являясь простыми, сами представляют собой достаточно самостоятельные алгоритмы. В этом случае говорят о вспомогательных алгоритмах или подпрограммах. Использование подпрограмм позволяет сделать основную программу более наглядной, понятной, а в случае, когда одна и та же последовательность команд встречается в программе несколько раз, даже более короткой и эффективной.

В языке Паскаль существует два вида подпрограмм: процедуры и функции, определяемые программистом. Процедурой в Паскале называется именованная последовательность инструкций, реализующая некоторое действие. Функция отличается от процедуры тем, что она должна обязательно выработать значение определенного типа.

Процедуры и функции, используемые в программе, должны быть соответствующим образом описаны до первого их упоминания. Вызов процедуры или функции производится по их имени.

Подпрограммы в языке Паскаль могут иметь параметры (значения, передаваемые в процедуру или функцию в качестве аргументов). При описании указываются так называемые формальные параметры (имена, под которыми будут фигурировать передаваемые данные внутри подпрограммы) и их типы. При вызове подпрограммы вместе с ее именем должны быть заданы все необходимые параметры в том порядке, в котором они находятся в описании. Значения, указываемые при вызове подпрограммы, называются фактическими параметрами.

**Формат описания процедуры:  
Procedure** <Имя процедуры> (<Имя форм. параметра 1>:<Тип>;   
< Имя форм. параметра 2>:<Тип>);  
<Раздел описаний>  
**Begin**  
<Тело процедуры>  
**End**;

Раздел описаний может иметь такие же подразделы, как и раздел описаний основной программы (описание процедур и функций - в том числе). Однако все описанные здесь объекты "видимы" лишь в этой процедуре. Они здесь локальны также, как и имена формальных параметров. Объекты, описанные ранее в разделе описаний основной программы и не переопределенные в процедуре, называются глобальными для этой подпрограммы и доступны для использования.

Легко заметить схожесть структуры программы целиком и любой из ее процедур. Действительно, ведь и процедура и основная программа реализуют некий алгоритм, просто процедура не дает решения всей задачи. Отличие в заголовке и в знаке после End.

**Формат описания функции:  
Function** <Имя функции> (<Имя форм. параметра 1>:<Тип>;  
< Имя форм. параметра 2>:<Тип>?) : <Тип результата>;  
<Раздел описаний>  
**Begin**  
<Тело функции>  
**End**;

В теле функции обязательно должна быть хотя бы команда присвоения такого вида: <Имя функции>:=<Выражение>;

Указанное выражение должно приводить к значению того же типа, что и тип результата функции, описанный выше.

Вызов процедуры представляет в программе самостоятельную инструкцию:

<Имя процедуры>(<Фактический параметр 1>, < Фактический параметр 2>);

Типы фактических параметров должны быть такими же, что и у соответствующих им формальных.

Вызов функции должен входить в выражение. При вычислении значения такого выражения функция будет вызвана, действия, находящиеся в ее теле, будут выполнены, в выражение будет подставлено значение результата функции.

пример :

**"Найти максимальное из трех введенных чисел".**

Для решения воспользуемся описанием функции, принимающей значение максимального из двух чисел, которые передаются в нее в виде параметров.

**Program** Fn;  
**Var**  
A,B,C :Real;  
**Function** Max(A,B:Real):Real; {Описываем функцию Max с формальными}  
**Begin** {параметрами A и B, которая принимает }

If A>B Then Max:=A {значение максимального из них }  
Else Max:=B {Здесь A и B - локальные переменные }

**End**;  
**Begin**

Writeln('Введите три числа');  
Readln(A,B,C);  
Writeln('Максимальным из всех является ', Max(Max(A,B),C))

**End.**

Обратите внимание на краткость тела основной программы и на прозрачность действий внутри функции. Формальные параметры A и B, используемые в подпрограмме, не имеют никакого отношения переменным A и B, описанным в основной программе.

Существует два способа передачи фактических параметров в подпрограмму: по значению и по ссылке. В первом случае значение переменной фактического параметра при вызове подпрограммы присваивается локальной переменной, являющейся формальным параметром подпрограммы. Что бы потом ни происходило с локальной переменной, это никак не отразится на соответствующей глобальной. Для одних задач это благо, но иногда требуется произвести в подпрограмме действия над самими переменными, указанными в качестве фактических параметров. На помощь приходит второй способ. Происходит следующее: при обращении к подпрограмме не происходит формирования локальной переменной формального параметра. Просто на время выполнения подпрограммы имя этой локальной переменной будет указывать на ту же область памяти, что и имя соответствующей глобальной переменной. Если в этом случае изменить локальную переменную, изменятся данные и в глобальной.

Передача параметров по ссылке отличается тем, что при описании подпрограммы перед именем переменной формального параметра ставится служебное слово **Var**. Теперь использование в качестве фактических параметров выражений или непосредственных значений уже не допускается - они должны быть именами переменных.

пример:

**"Расположить в порядке неубывания три целых числа".**

**Program Pr**;  
**Var**  
     S1,S2,S3 :Integer;  
**Procedure** Swap(Var A,B: Integer);{Процедура Swap с параметрами-переменными}  
**Var** C : Integer; {C - независимая локальная переменная}  
**Begin**   
      C:=A; A:=B; B:=C {Меняем местами содержимое A и B}  
**End**;  
**Begin**

Writeln('Введите три числа');  
Readln(S1,S2,S3);  
If S1>S2 Then Swap(S1,S2);  
If S2>S3 Then Swap(S2,S3);  
If S1>S2 Then Swap(S1,S2);  
Writeln('Числа в порядке неубывания:V',S1,S2,S3)

**End.**

**Задачи.**

1). Найти среднее арифметическое двух чисел. (PROCEDURE)

Var x,y,sr:real;

Procedure Serd(a,b:real;VAR S:REAL);…{список формальных параметров}

Begin

S:=(a+b)/2; {тело процедуры}

End;

Begin

Readln(x,y);

Sred(x,y,sr); …{список фактических параметров}

Writeln(sr);

End.

2).Найти :. (PROCEDURE)

Var x,y,z:integer:

Sr1,sr2,sr3,SR:real;

Procedure Serd(a,b:real;VAR S:REAL);…

Begin

S:=(a+b)/2;

End;

Begin

Readln(x,y);

Sred(x,y,sr1);

Sred(x,y,sr2);

Sred(x,y,sr3);

SR:=(sr1+sr2)/sr3;

Writeln(sr);

End.

3).Найти аn. (PROCEDURE)

var n,a,rez:integer;

procedure step(x,y:integer; var st:integer);

var I:integer;

begin

st:=1;

for I:=1 to y do st:=st\*x;

end;

begin

writeln(‘введите число а’);

readln(a);

writeln(‘введите степень’);

readln(n);

step(a,n,rez);

writeln(rez);

end.

4). Получить все делители введенного числа. Посчитать их количество. (PROCEDURE)

Var z,b,c,p,i:integer;

Procedure delit(n:integer; var k:integer);

Begin

Readln(n);

For I:=1 to n do begin

P:=n mod I:

If p=0 then begin

Writeln(‘делители’,I);

K:=k+1;

End;

End;

End;

Writeln(‘введите n’);

Readln(z);

Delit(z,b);

Writeln(‘количество делителей”,b);

End.

5). Найти сумму двух чисел. (FUNCTION)

Var a,b:integer;

Function summa(x,y:integer):integer;

Begin

Summa:=x+y;

End;

begin

Writeln(‘ введите два числа’);

Readln(a,b);

Writeln(summa(a,b));

End.

6). Найти площадь треугольника. . (FUNCTION)

Var x,y,z:intger; p:real;

Unction treug(a,b,c:integer):real;

Begin

P:=(a+b+c)/2;

Treug:=sqrt(p\*(p-a)\*(p-b)\*(p-c));

End;

Begin

Read(x,y,z);

Writeln(treug(x,y,z));

End.

**Строковые операции**

Все языки программирования высокого уровня имеют средства работы с литерными величинами. Паскаль - не исключение. В стандарте языка описаны два типа переменных для литерных величин. Это - **String** и **Char**. Переменная типа **Char** может содержать в себе только один единственный символ, тип **String** предназначен для хранения строковых величин до 255 символов длиною. При описании переменной типа **String** можно самим указать максимальное число символов, которое можно занести в нее. Конечно же, это число не должно превышать 255.

пример:

**Var** S : **String[30]**;

Для чего это нужно?

Дело в том, что при компиляции для каждой переменной отводится свой участок памяти. Если мы будем выделять для всех переменных типа Strin **String** g по 256 байт, то это приведет к тому, что при использовании достаточно большого их количества, памяти может и не хватить. Но если в переменной мы собираемся хранить, например, фамилию пользователя, то тридцати символов (тридцати байт) для этого вполне достаточно. Таким образом, экономится память и увеличивается быстродействие программ.

Переменным строкового типа можно присваивать строковые величины (внутри программы они заключаются в апострофы), значения выражений, которые приводят к строковым величинам. Значения можно также вводить с клавиатуры. При этом апострофы не используются. Как вам известно, в числовую переменную нельзя ввести строковую величину. Сделать наоборот - возможно, однако число, находящееся в строковой переменной, представляет собой просто последовательность символов (цифр), поэтому в арифметических выражениях участвовать не может.

Также новым для вас явится то, что при использовании строковой переменной к каждому ее символу можно обратиться отдельно. Необходимо только знать номер нужного символа от начала строки. Его достаточно поставить после имени переменной типа **String** в квадратных скобках.

Пример: S[5] - пятый символ строки S.

С отдельным символом строки можно производить все действия, которые можно производить с любой символьной переменной (ввод, присвоение, вывод на экран, участие в выражениях и т.д.).

 Обратите внимание на то, что нумерация символов в строке начинается с единицы. Внутри квадратных скобок вместо числа может находиться выражение, результатом которого является целое число. Главное, чтобы символ с таким номером в строке существовал. Но как же узнать, сколько символов в данный момент находится в строковой переменной? Для этого существует специальная функция, которая возвращает длину строковой переменной в символах. Это функция **Length**.

Ее формат: **Length(S)**

Здесь **S** - либо строковая величина, либо строковая переменная.

Приведенная далее программа выводит на экран длину введенной пользователем строковой величины.

**Program** Str1;  
**Var**   
S : String;  
**Begin**

Writeln('Введите последовательность символов');  
Readln(S);  
Writeln('Вы ввели строку из ',Length(S), ' символов')

**End.**

     Решим задачу:

**"Введенную строку вывести на экран по одному символу в строке экрана".**

**Program** Str2;  
**Var**

S : String;  
I : Byte;

**Begin**

Writeln('Введите строку');  
Readln(S);  
For I:=1 to Length(S) do {организуем цикл, начиная с первого символа}  
Writeln(S[I])                   {строки, до последнего (номер последнего}  
{совпадает с количеством символов строки S) }

**End.**

Какие же еще действия можно выполнять с переменными строкового типа?

Две строковые величины можно состыковывать. Эта операция называется конкатенацией и обозначается знаком "+".

Например, результатом выполнения следующих команд:  
   R:= 'kadabra';  
   H:= 'abra';  
   S:=H+R;  
в переменной S будет значение 'abrakadabra'.

Для конкатенации результат зависит от порядка операндов (в отличие от операции сложения). Следует помнить о том, какой максимальной длины может быть результирующая переменная, так как в случае превышения значением выражения числа, указанного после **String** в описании переменной, "лишние" символы в переменную не попадут.

Строковые величины можно сравнивать между собой. Это относится также и к строковым переменным. Но как же компьютер определяет, какая строка больше:

* та, которая длиннее?
* та, которая содержит больше заглавных букв?

На самом деле такая проверка проходит довольно сложно: компьютер сравнивает сначала первые символы строк. Большим из двух считается тот, код которого больше (вспомните, что такое код символа). Если равны первые символы, то так же анализируется следующая пара до тех пор, пока не будет найдено различие. Если начало строк совпадает, а одна из них кончается раньше, то вторая автоматически называется большей.

Код символа в Паскале можно определить при помощи функции **Ord**.

Ее формат: **Ord(C)**, где С - либо непосредственно указанный символ, либо переменная символьного типа, либо один символ строковой переменной. Вообще, функция **Ord** имеет более глубокий смысл, но об этом - позже. Есть и обратная функция, которая возвращает символ по известному коду. Это функция **Chr**(N), где N - выражение, приводящее к целому числу в интервале от 0 до 255 (возможные значения кода символа). Очевидно, что **Chr**(Ord(C))=C, **Ord**(Chr(N))=N.

 Следующая маленькая программа выводит на экран кодовую таблицу:

**Program** Str3;  
**Var**  
    I : Byte;  
**Begin**

For I:=32 to 255 do  
Write('VV',I:4, '-',Chr(I))

**End.**

Цикл в программе начинается с 32 потому, что символы с кодами от 0 до 31 являются управляющими и не имеют соответствующего графического представления.

Задача:

**"Определить, является ли введенная строка "перевертышем". Перевертышем называется такая строка, которая одинаково читается с начала и с конца. Например, "казак" и "потоп" - перевертыши, "канат" - не перевертыш".**

Поступим следующим образом: из введенной строки сформируем другую строку из символов первой, записанных в обратном порядке, затем сравним первую строку со второй; если они окажутся равны, то ответ положительный, иначе - отрицательный. Естественно, предложенный способ решения не является единственно возможным.

**Program** Str4;  
**Var**   
     S,B : String;  
     I : Byte;  
**Begin**

Writeln('Введите строку');   
Readln(S);  
B:='';              {Переменной B присваиваем значение "пустая строка"}  
For I:=1 to Length(S) do  
B:=S[I]+B;     {Конкатенация. Символы строки S пристыковываются к}  
{переменной B слева. Самым левым окажется последний.}  
If B=S Then Writeln('Перевертыш') Else Writeln('Не перевертыш')

**End.**

Число, записанное в строковую переменную, естественно числом не является, но очень часто требуется его все же использовать в качестве числа. Для этого нужно произвести преобразование типа. Перевод строкового представления числа в числовое выполняет в Паскале оператор **Val**.

Его формат:  
**Val(**S,X,C**)**;

Здесь

S - строка, содержащая число,

X - числовая переменная, в которую будет помещен результат,

С - переменная целочисленного типа, в которую помещается первого встреченного в S отличного от цифры символа.

Если после выполнения оператора **Val** переменная С имеет значение 0, то это означает, что преобразование типа прошло совершенно успешно и в строке нецифровых символов не встретилось.

Противоположное действие осуществляет оператор **Str**. Формат оператора:

**Str(**X,S**)**;

X - число (либо арифметическое выражение),

S - строковая переменная.

В переменную S попадает строковое представление числа X. Это нужно, например, при необходимости выводить на экран числа в графическом режиме (будет изучено позже), так как стандартные процедуры вывода на экран там работают только со строковыми величинами.

Для иллюстрации рассмотрим такую задачу:

**"Найти сумму цифр введенного натурального числа".**

Используя только числовые переменные, решить ее можно, но предлагаемое здесь решение, по-моему, проще.

**Program** Str5;  
**Var**

S : String;  
I,X,A,C : Integer;

**Begin**

Writeln('Введите натуральное число');  
Readln(S); {Число вводится в строковую переменную}  
A:=0;  
For I:=1 To Length(S) Do  
Begin

Val(S[I],X,C); {Цифровой символ превращается в число}  
A:=A+X {Цифры суммируются}

End;  
Writeln('Сумма цифр равна ',A)

**End.**

Теперь рассмотрим еще несколько действий над строками:

* оператор **DELETE(S,I,C)** из строковой переменной S удаляет C символов, начиная с I-того;
* оператор **INSERT(SN,S,I)** вставляет подстроку SN в строковую переменную S перед символом с номером I;
* функция **COPY(S,I,C)** возвращает подстроку строки S из C символов, начиная с символа с номером I;
* функция **Pos(SN,S)** возвращает номер символа, с которого в строке S начинается подстрока SN (позицию первого вхождения подстроки в строку). Если такой подстроки нет, то возвращается ноль.

Пример их использования:

**"Во введенной строке заменить все вхождения подстроки 'ABC' на подстроки 'KLMNO'".**

**Program** Str6**;**  
**Var**

S : String;  
A : Byte;

**Begin**

Writeln('Введите строку');  
Readln(S);  
While Pos('ABC',S)<>0 Do  
Begin

A:= Pos('ABC',S);  
Delete(S,A,3);  
Insert('KLMNO',S,A)

End;  
Writeln(S)

**End.**

**Определение типов**

В языке Паскаль возможно определять новые типы переменных. После определения этот тип становится доступным для описания переменных, также как и стандартные типы.

Новый тип перед первым его использованием должен быть описан в соответствующем разделе описаний. Его заголовок - служебное слово **Type**.

**Type**     <Имя типа> = <Описание типа>;

Есть несколько способов описания. Иногда говорят даже о видах типов (как бы это странно ни звучало).

Итак, первым рассмотрим так называемый **перечисляемый** **тип**.

**Перечисляемый** **тип** используется для повышения наглядности программ, позволяя записывать в переменные этого типа названия разнообразных объектов, исследуемых программой. Этот тип представляет собой набор идентификаторов, с которыми могут совпадать значения параметров.

Формат описания следующий:

**<Имя типа> = (<Ид.1>, <Ид.2>,? <Ид.N>);**

Далее можно определить любое число переменных уже описанного типа. Каждый идентификатор может участвовать в описании только одного перечисляемого типа.

Этим переменным можно присваивать только значения из списка, определенного при описании типа. Эти значения не являются ни числами, ни строковыми величинами, ни даже величинами логического типа, поэтому они не могут участвовать в арифметических, строковых, логических выражениях, а также не могут быть выведены на экран или принтер. Величины перечисляемого типа можно сравнивать между собой, над их множеством в языке Паскаль определены несколько функций:

**Ord(X)** - порядковый номер значения переменной X в списке идентификаторов.

**Succ(X)** - следующее значение для величины Х.

**Pred(X)** - предыдущее значение данного типа.

Для функции **Ord** нумерация среди значений идет, начиная от нуля. Для последнего значения нельзя применять функцию **Succ**, для первого - **Pred**.

Переменные различных перечисляемых типов несовместимы друг с другом.

Множество стандартных порядковых типов в языке Паскаль на самом деле определены как перечисляемые. Это типы **Char**, **Integer**, другие. Достоинства стандартных порядковых типов лишь в том, что над каждым из них уже определены специфические действия. Например, тип **Boolean** описан так:

**Type Boolean = (False, True)**;

Единственное его отличие от перечисляемых типов, определяемых программистом, состоит в том, что значения типа **Boolean** можно выводить на экран. Можете проверить, **Ord**(False)=0.

Интересно, что переменная перечисляемого типа может быть счетчиком в цикле "с параметром".

пример:

**Program** T1;  
**Type**  
Colors = (Black, Blue, Green, Cyan, Red, Magenta, Brown, Yellow, White);  
**Var**  
    C1,C2 : Colors;  
**Begin**

C1:=Green;  
C2:=Red;  
Writeln(Ord(C1), Ord(Succ(C2)))

**End.**

Во время выполнения на экране появятся числа "2" и "5", что соответствует номерам значений Green и Magenta.

**Массивы**

**Массивом** будем называть упорядоченную последовательность данных одного типа, объединенных под одним именем.

Положение ячейки в таблице должно однозначно определяться набором координат (индексов). Самой простой является линейная таблица, в которой для точного указания на элемент данных достаточно знания только одного числа (индекса).

Описание типа линейного массива:

**Type** <Имя типа>=**array** [<Диапазон индексов>] **Of** <Тип эл-ов>;

В качестве индексов могут выступать переменные любых порядковых типов. При указании диапазона начальный индекс не должен превышать конечный. Тип элементов массива может быть любым (стандартным или описанным ранее).

Описать переменную-массив можно и сразу (без предварительного описания типа) в разделе описания переменных:

**Var** <Переменная-массив> : **Array** [<Диапазон индексов>] **Of** <Тип элементов>;

Примеры описания массивов*:*

**Var**

S, BB : **Array** [1..40] Of Real;  
N : **Array** ['A'..'Z'] Of Integer;  
R : **Array** [-20..20] Of Word;  
T : **Array** [1..40] Of Real;

Теперь переменные S, BB и T представляют собой массивы из сорока вещественных чисел; массив N имеет индексы символьного типа и целочисленные элементы; массив R может хранить в себе 41 число типа Word.

Единственным действием, которое возможно произвести с массивом целиком - присваивание. Для данного примера описания впоследствии допустима следующая запись:

*S:=BB;*

Однако присваивать можно только массивы одинаковых типов. Даже массиву T присвоить массив S нельзя, хотя, казалось бы, их описания совпадают, произведены они в различных записях раздела описания.

Никаких других операций с массивами целиком произвести невозможно, но с элементами массивов можно работать точно так же, как с простыми переменными соответствующего типа. Обращение к отдельному элементу массива производится при помощи указания имени всего массива и в квадратных скобках - индекса конкретного элемента. Например:

R[10] - элемент массива R с индексом 10.

Фундаментальное отличие компонента массива от простой переменной состоит в том, что для элемента массива в квадратных скобках может стоять не только непосредственное значение индекса, но и выражение, приводящее к значению индексного типа. Таким образом, реализуется косвенная адресация:

BB[15] - прямая адресация;

BB[K] - косвенная адресация через переменную K, значение которой будет использовано в качестве индекса элемента массива BB.

Такая организация работы с такой структурой данных, как массив, позволяет использовать цикл для заполнения, обработки и распечатки его содержимого.

 Переменные типа **String** очень близки по своим свойствам массивам типа **Char**. Отличия в следующем: строковые переменные можно было вводить с клавиатуры и распечатывать на экране (с обычным массивом это не проходит); длина строковой переменной была ограничена 255 символами (255 B), а для размера массива критическим объемом информации является 64 KB.

**ОДНОМЕРНЫЙ МАССИВ (ВЕКТОР)**

Направление изменения индекса.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | … | n |
| **А** |  |  |  |  |  |  |  |

**Идентификатор**:

**ARRAY** [диапазоны индексов] **OF** тип элемента массива;

**Описание**.

**Var** A:**array**[1..5] **of** integer;

**const** n=5;

**Var** A:**array**[1..n] **of** integer;

**const** n=5;

**type** vector=**array**[1..n] **of** integer;

**var** a:vector;

**Ввод и вывод массива**

**Var** A:**array**[1..5] of integer;

i:integer;

**Begin**

**For** i:=1 **to** 5 do **read**(a[I]); ввод элементов массива.

**For** i:=1 **to** 5 do **write**(a[I]); вывод элементов массива.

В дальнейшем для учебных целей мы будем использовать массивы, заданные с помощью генератора случайных чисел. В языке Паскаль случайные числа формирует функция Random. Числа получаются дробными, равномерно расположенными в интервале от 0 до 1. Выражение, дающее целое случайное число в интервале [-50,50], будет выглядеть так:

Trunc(Random\*101)-50

пример:

**Зададим и распечатаем случайный массив из сорока целых чисел:**

**Program** M4;  
**Const**  
N=40; {Константа N будет содержать количество элементов массива}  
**Var**

A : Array [1..N] Of Integer;  
I : Integer;

**Begin**

**For** I:=1 **To** N **Do**  
**Begin**

A[I]:= Trunc(Random\*101)-50  
Write(A[I],'VVV')

**End**

**End.**

пример:

**Заполнить массив значениями квадратов индексов элементов.**

**Program** M3;  
**Const**  
N=50; {Константа N будет содержать количество элементов массива}  
**Var**

A : Array [1..N] Of Integer;  
I : Integer;

**Begin**

For I:=1 To N Do  
A[I]:=I\*I  
For I:=1 To N Do  
Write(A[I],'VVV')

**End.**

**ДВУМЕРНЫЕ И МНОГОМЕРНЫЕ МАССИВЫ**

**Двумерный массив (матрица)**

Направление изменения второго индекса.

Направление изменения первого индекса

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | . | . |  | n |
| 1 |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| n |  |  |  |  |  |  |  |  |  |

##### **A[2,5] или**

##### **А[I,j] если**

##### **i=5, j=2**

Идентификатор:

**ARRAY** [диапазоны индексов] **OF** тип элемента массива;

Представьте себе таблицу, состоящую из нескольких строк. Каждая строка состоит из нескольких ячеек. Тогда для точного определения положения ячейки нам потребуется знать не одно число (как в случае таблицы линейной), а два: номер строки и номер столбца. Структура данных в языке Паскаль для хранения такой таблицы называется двумерным массивом.

**ОПИСАНИЕ**.

1. **Var** A:**array**[1..5:1..4] **of** integer;
2. **const** n=5; m=4

**Var** A:**array**[1..n;1..m] **of** integer;

1. **const** n=5; m=4

**type** vector=**array**[1..n;1..m] **of** integer;

**var** a:vector;

**Ввод и вывод массива**

**Var** A:**array**[1..5;1..4] **of** integer;

I,j : integer;

**Begin**

**For** I:=1 **to** 5 **do**

**For** j:=1 **to** 4 **do** **read**(a[I,j]); ввод элементов массива.

**For** I:=1 **to** 5 **do** **begin**

**For** j=1 **to** 4 **do** **write**(a[I,j]); вывод элементов массива.

**Writeln**;

**End**;

|  |
| --- |
| Для иллюстрации способов работы с двумерными массивами решим задачу:  **"Задать и распечатать массив 10X10, состоящий из целых случайных чисел в интервале [1,100]. Найти сумму элементов, лежащих выше главной диагонали."**  При отсчете, начиная с левого верхнего угла таблицы, главной будем считать диагональ из левого верхнего угла таблицы в правый нижний. При этом получается, что элементы, лежащие на главной диагонали, будут иметь одинаковые индексы, а для элементов выше главной диагонали номер столбца будет всегда превышать номер строки. Договоримся также сначала указывать номер строки, а затем - номер столбца.  **Program M5**; **Var**  A : Array[1..10,1..10] Of Integer; I, K : Byte; S : Integer;  **Begin**  S:=0; For I:=1 To 10 Do Begin For K:=1 To 10 Do Begin  A[I,K]:=Trunc(Random\*100)+1; Write(A[I,K]:6); If K>I Then S:=S+A[I,K]  End; Writeln End; Writeln('Сумма элементов выше гл. диагонали равнаV',S)  **End**. |

**Работа с файлами**

Тип-файл представляет собой последовательность компонент одного типа, расположенных на внешнем устройстве (например, на диске). Элементы могут быть любого типа, за исключением самого типа-файла. Число элементов в файле при описании не объявляется. Работа с физическими файлами происходит через так называемые файловые переменные.

Для задания типа-файла следует использовать зарезервированные слова **File** и **Of**, после чего указать тип компонент файла.

*Пример:***Type**

N = File **Of** Integer; {Тип-файл целых чисел}  
C = File **Of** Char; {Тип-файл символов}

Есть заранее определенный в Паскале тип файла с именем **Text**. Файлы этого типа называют текстовыми.

Введя файловый тип, можно определить и переменные файлового типа:

**Var**

F1 : N;  
F2 : C;  
F3 : Text;

Тип-файл можно описать и непосредственно при введении файловых переменных:

**Var**  
      Z : File Of Word;

Файловые переменные имеют специфическое применение. Над ними нельзя выполнять никаких операций (присваивать значение, сравнивать и т.д.). Их можно использовать лишь для выполнения операций с файлами (чтение, запись и т.д.).

Элементы файла считаются расположенными последовательно, то есть так же, как элементы линейного массива. Отличие же состоит в том, что, во-первых, размеры файла могут меняться, во-вторых, способ обращения к элементам совсем другой: невозможно обратиться к произвольному элементу файла; элементы его просматриваются только подряд от начала к концу, при этом в каждый момент времени доступен только один элемент. Можно представить себе, что для каждого файла существует указатель, показывающий в данный момент на определенный компонент файла. После проведения операции чтения или записи указатель автоматически передвигается на следующий компонент.

Перед тем, как осуществлять ввод-вывод, файловая переменная должна быть связана с конкретным внешним файлом при помощи процедуры **Assign**.

**Формат:  
Assign**(<Имя файловой переменной>,<Имя файла>);

Имя файла задается либо строковой константой, либо через переменную типа **String**. Имя файла должно соответствовать правилам работающей в данный момент операционной системы. Если строка имени пустая, то связь файловой переменной осуществляется со стандартным устройством ввода-вывода (как правило - с консолью).

После этого файл должен быть открыт одной из процедур:

**Reset**(<Имя файловой переменной>);

Открывается существующий файл для чтения, указатель текущей компоненты файла настраивается на начало файла. Если физического файла, соответствующего файловой переменной не существует, то возникает ситуация ошибки ввода-вывода.

**Rewrite**(<Имя файловой переменной>);

Открывается новый пустой файл для записи, ему присваивается имя, заданное процедурой **Assign**. Если файл с таким именем уже существует, то он уничтожается.

После работы с файлом он, как правило, должен быть закрыт процедурой **Close**.

**Close**(<Имя файловой переменной>);

Это требование обязательно должно соблюдаться для файла, в который производилась запись.

Теперь рассмотрим непосредственную организацию чтения и записи.

Для ввода информации из файла, открытого для чтения, используется уже знакомый вам оператор **Read**. Правда, в его формате и использовании вы заметите некоторые изменения:

**Read**(<Имя файловой переменной>, <Список ввода>);

Происходит считывание данных из файла в переменные, имена которых указаны в списке ввода. Переменные должны быть того же типа, что и компоненты файла.

Вывод информации производит, как можно догадаться, оператор **Write**(<Имя файловой переменной>, <Список вывода>);

Данные из списка вывода заносятся в файл, открытый для записи.

Для текстовых файлов используются также операторы **Readln** и **Writeln** с соответствующими дополнениями, относящимися к файловому вводу-выводу. Любопытно, что вывод данных на монитор и ввод с клавиатуры в языке Паскаль тоже являются действиями с файлами. Они даже имеют свои предопределенные файловые переменные текстового типа: **Output** и **Input** соответственно. Переменная **Output** всегда открыта для записи, **Input** - для чтения. Если не указывать файловые переменные в операторах ввода-вывода (придем к формату, рассмотренному в теме "Операторы ввода-вывода"), то в случае записи по умолчанию выбирается файл **Output**, в случае чтения - **Input**.

Как вы знаете, любой файл конечен и продолжать чтение из него информации можно лишь до определенного предела. Как этот предел установить? Проверить, окончен ли файл, можно вызовом стандартной логической функции **Eof**(<Имя файловой переменной>). Она вырабатывает значение **True**, если файл окончен, и **False** - в противном случае.

Решим следующую задачу:

**"Написать программу, которая вводит с клавиатуры список фамилий учащихся, а затем распечатывает его, кроме тех учащихся, у которых фамилия начинается с буквы 'Ш'".**

Так как заранее количество данных не известно, то для их хранения используем файл. Тип элементов - строковый.

**Program** L;  
**Var**

I,N : Integer;  
F : File Of String;  
S : String;

**Begin**

Assign(F,'Spis.lst'); {Связываем переменную F с файлом Spis.lst}  
Writeln('Введите количество учащихся');  
Readln(N); {Вводим количество учащихся}  
Rewrite(F); {Создаем файл для записи в него данных}  
For I:=1 To N Do {Для всех учащихся}  
Begin

Writeln('Введите фамилию');  
Readln(S);  
Write(F,S)

End;  
Close(F);   
Reset(F);   
Writeln; Writeln('Список учащихся:');  
While Not(Eof(F)) Do   
Begin

Read(F,S);   
If S[1]<>'Ш' Then  
Writeln(S)

End;  
Close(F)

End.