МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное автономное образовательное учреждение**

**высшего образования**

**«Национальный исследовательский нижегородский университет им. Н.И. Лобачевского»**

**И.Д. Камскова**

**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ для выполнения практических заданий по дисциплине «ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ»**

Учебно-методическое пособие

Рекомендовано методической комиссией института экономики и предпринимательства ННГУ для обучающихся по специальности среднего профессионального образования 09.02.04 «Информационные системы (по отраслям)»

2017

УДК 681.3

ББК 32.81

Камскова И.Д. Методические рекомендации для выполнения практических заданий по дисциплине «Основы алгоритмизации и программирования»

Учебно-методическое пособие. - Нижний Новгород: Нижегородский государственный университет им. Н.И. Лобачевского, 2017. − 51 с.

Рецензент: к.т.н., доцент Штанюк А.А.

Методические рекомендации направлены на выполнение практических заданий по дисциплине «Основы алгоритмизации и программирования» для учащихся CПО по специальности 09.02.04 «Информационные системы (по отраслям)».

Главной задачей выполнения практических заданий является получение практических навыков программирования и решения алгоритмических задач.

Методические рекомендации разработаны в соответствии с программой по учебной дисциплине «Основы алгоритмизации и программирования».

Ответственный за выпуск:

председатель методической комиссии ИЭП ННГУ Летягина Е.Н.

УДК 681.3

ББК32.81

 © **Национальный исследовательский**

 **Нижегородский государственный**

 **Университет им. Н.И. Лобачевского, 2017**

Оглавление

[ПОЯСНИТЕЛЬНАЯ ЗАПИСКА 3](#_Toc512455596)

[ТЕМАТИЧЕСКОЕ ПЛАНИРОВАНИЕ 3](#_Toc512455597)

[МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ ЗАДАНИЙ 5](#_Toc512455598)

[Раздел 1. Основы алгоритмизации 5](#_Toc512455599)

[Раздел 2. Основы программирования в среде программирования 9](#_Toc512455600)

[Раздел 3. Структурированные типы данных 39](#_Toc512455601)

[Литература для учащегося 51](#_Toc512455602)

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Данные методические рекомендации направлены на реализацию практической работы по учебной дисциплине «Основы алгоритмизации и программирования» для студентов по специальности CПО по специальности 09.02.04 «Информационные системы (по отраслям)».

Практическая работа студента является одним из основных методов приобретения навыков программирования и решения алгоритмических задач. Главной задачей практической работы является развитие общих и профессиональных компетенций, умений приобретать научные знания путем выполнения практических заданий, формирование активного интереса и творческого подхода к решению задач.

Методические рекомендации по  выполнению практической работы разработаны в соответствии с программой по учебной дисциплине «Основы алгоритмизации и программирования».

При реализации программы у студентов формируются компетенции:

ОК 1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес

ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество

ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность

ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития

ОК 5. Использовать информационно - коммуникационные технологии в профессиональной деятельности

ОК 6. Работать в коллективе и команде, эффективно общаться с коллегами, руководством, потребителями

ОК 7. Брать на себя ответственность за работу членов команды (подчинённых), результат выполнения заданий

ОК 8. Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием, осознанно планировать повышение квалификации

ОК 9. Ориентироваться в условиях частой смены технологий в профессиональной деятельности

ПК 1.2. Взаимодействовать со специалистами смежного профиля при разработке методов, средств и технологий применения объектов профессиональной деятельности

ПК 1.3. Производить модификацию отдельных модулей информационной системы в соответствии с рабочим заданием, документировать произведённые изменения ПК

ПК 2.2 Программировать в соответствии с требованиями технического задания.

ПК 2.3 Применять методики тестирования разрабатываемых приложений.

# ТЕМАТИЧЕСКОЕ ПЛАНИРОВАНИЕ

|  |  |
| --- | --- |
| **Раздел / тема учебной дисциплины** | **Кол-во часов практических занятий** |
| **2017** | **2016** | **2015** | **2014** |
| **Раздел 1. Основы алгоритмизации** |
| Раздел 1. Тема 1.1. Алгоритм и его свойства | 0 | 0 | 0 | 0 |
| Раздел 1. Тема 1.2. Основные алгоритмические конструкции | 8 | 8 | 8 | 8 |
| **Раздел 2. Основы программирования в среде программирования** |
| Раздел 2. Тема 2.1.  Эволюция языков программирования | 0 | 0 | 0 | 0 |
| Раздел 2. Тема 2.2. Программирование на языке Паскаль | 26 | 26 | 26 | 26 |
| **Раздел 3. Структурированные типы данных** |
| Раздел 3. Тема 3.1. Понятие массива | 14 | 14 | 14 | 14 |
| Раздел 3. Тема 3.2. Строковый тип данных | 14 | 14 | 14 | 14 |
|  | **64** | **64** | **64** | **64** |

# МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ ЗАДАНИЙ

## Раздел 1. Основы алгоритмизации

**Тема 1.2. Основные алгоритмические конструкции**

**Методические рекомендации по выполнению практических заданий**

Для выполнения практических заданий студентам необходимо знать понятия линейного, циклического алгоритмов и алгоритма ветвления. В процессе освоения материала студенты должны понять суть задач, решаемых с помощью различных алгоритмических структур.

Студент должен изучить назначения алгоритмов, знать графический способ представления алгоритмических структур, знать категории задач, решаемых при помощи разных алгоритмических структур.

Примеры решения задач представлены ниже в тексте.

**Основные алгоритмические конструкции**

**Алгоритмы линейной структуры**

Линейным называется вычислительный процесс, в котором операции выполняются последовательно в естественном порядке. Блоки в этой структуре располагаются на схеме в том же порядке, в каком должны быть выполнены предписываемые ими действия. Такой порядок исполнения действий, называется естественным

**Алгоритмы разветвляющейся структуры**

Разветвляющимся называют вычислительный процесс, в котором операции выполняются в одном из направлений в зависимости от заданных условий. В общем случае число ветвей в алгоритме разветвляющейся структуры не обязательно равно двум.

**ЗАДАЧА. Разработать алгоритм вычисления наибольшего числа из двух чисел x и y.**

Из курса математики известно, если x > y, то наибольшее число x, если x < y, то наибольшее число y, если x = y, то число x равно числу y.

Входными данными являются значения чисел x и y. Выходным данными являются: наибольшее число или любое из чисел, если числа равны

Для решения задачи нам необходимо знать значения x и y.

**Словесное описание алгоритма**

1. Начало алгоритма.
2. Ввод значений x и y.
3. Сравниваем x и y. Если x = y, то переход к шагу 4, иначе к шагу 5.
4. Вывод информации: числа x и y равны. Переход к шагу 8.
5. Сравниваем x и y. Если x > y, то переход к шагу 6, иначе к шагу 7.
6. Вывод информации: число x больше y. Переход к шагу 8.
7. Вывод информации: число y больше x. Переход к шагу 8.
8. Конец алгоритма.

**Блок-схема**



**Алгоритмы циклической структуры**

Циклическим называют вычислительный процесс, в котором многократно повторяются некоторые этапы вычислений.

Часто при решении задач приходится многократно вычислять значения по одним и тем же математическим зависимостям для различных значений входящих в них величин. Такие многократно повторяемые участки вычислительного процесса называются циклами. Использование циклов позволяет существенно сократить объем схемы алгоритма и длину соответствующей ей программы. Различают циклы с заданным и неизвестным числом повторений. К последним относятся итерационные циклы, характеризующиеся последовательным приближением к искомому значению с заданной точностью.

Для организации цикла необходимо выполнить следующие действия:

Задать перед циклом начальное значение переменной, изменяющейся в цикле;

Изменять переменную перед каждым новым повторением цикла;

Проверять условие окончания или повторения цикла;

Управлять циклом, т.е. переходить к его началу, если он не закончен, или выходить из него по окончании.

Переменная, изменяющаяся в цикле, называется параметром цикла. В одном цикле может быть несколько параметров.

Переменную, значения которой вычисляются машиной и хранятся в одной и той же ячейке памяти, называют простой переменной а переменную, являющуюся элементом массива переменной с индексом. Следует иметь в виду, что параметром цикла является при использовании простой переменной сама переменная, а при использовании переменной с индексом – ее индекс.

**ЗАДАЧА. Разработать алгоритм вычисления суммы натуральных чисел от 1 до 100.**

Обозначим сумму натуральных чисел через S. Тогда формула вычисления суммы натуральных чисел от 1 до 100 может быть записана так:



где Xi – натуральное число X c номером i, который изменяется от 1 до n, n=100 – количество натуральных чисел.

Входными данными являются натуральные числа: 1, 2, 3, 4, 5, …, 98, 99, 100.

Выходные данные – значение суммы членов последовательности натуральных чисел.

Параметр цикла – величина, определяющая количество повторений цикла. В нашем случае i – номер натурального числа.

Подготовка цикла заключается в задании начального и конечного значений параметра цикла.

начальное значение параметра цикла равно 1,

конечное значение параметра цикла равно n,

шаг цикла равен 1.

Для корректного суммирования необходимо предварительно задать начальное значение суммы, равное 0.

Тело цикла. В теле цикла будет выполняться накопление значения суммы чисел, а также вычисляться следующее значение параметра цикла по формулам:

S=S+i;

I=I+1;

Условие продолжения цикла: цикл должен повторяться до тех пор, пока не будет добавлен последний член последовательности натуральных чисел, т.е. пока параметр цикла будет меньше или равен конечному значению параметра цикла.

Введем обозначения: S – сумма последовательности, i – значение натурального числа.

Начальное значение цикла i=1, конечное значение цикла i =100, шаг цикла 1.

**Словесное описание алгоритма**

1. Начало алгоритма.
2. Подготовка цикла: S:=0; i=1; n= 100;
3. Проверка условия. Если i <=n , то перейти к шагу 4, иначе к шагу 6.
4. Накопление суммы: S:=S+i;
5. Вычисление следующего значения параметра цикла: i:=i+1;
6. Вывод информации: сумма натуральных чисел – S.
7. Конец алгоритма.

**Блок-схема**



## Раздел 2. Основы программирования в среде программирования

**Тема 2.2. Программирование на языке Паскаль**

**Методические рекомендации по выполнению практических заданий**

Для выполнения практических заданий студентам необходимо знать операторы присваивания, ввода, вывода, типы данных: целые, логические и вещественные. Студенты должны выучить функции преобразования типов, стандартные математические функции Паскаль, арифметические, логические операции и операции отношения. Условный оператор If и оператор выбора Case. Три оператора цикла For, While, Repeat. Студенты должны знать синтаксис и семантику вышеперечисленных операторов.

Примеры решения задач представлены ниже в тексте.

**Линейные вычислительные процессы**

**Линейный вычислительный процесс** представляет собой последовательность действий, выполняемых один за другим. Основу программы линейного вычислительного процесса составляют оператор присваивания и операторы ввода-вывода данных.

**Оператор присваивания**

**Оператор присваивания** служит для вычисления выражения и записи результата в память компьютера.

Общий вид записи оператора:

**переменная := выражени>;**

Знак **:=** читается как “**присвоить**”. **Выражение** – это переменные, константы, функции и формулы. Конец любого оператора в Турбо Паскале фиксируется точкой с запятой.

**A:=3\*4.8;**

**Su:=X+X\*4.78;**

**C:=C+1;**

При выполнении оператора присваивания происходит замена текущего значения переменной, стоящей слева от знака присваивания, новым значением, полученным в результате вычисления выражения. Например, после выполнения оператора

**k : = k+1**; текущее значение переменной k увеличится на 1.

При записи оператора присваивания необходимо строго следить за типами переменной и выражения Тип переменной в левой части оператора присваивания обычно должен совпадать с типом значения выражения в правой части. Возможны случаи несовпадения типов, например, когда слева переменная вещественного типа, а справа выражение целого типа:

**Операторы ввода данных с клавиатуры**

**READ(список переменных);**

**READLN(список переменных);**

Переменные в списке перечисляются через запятую.

Операторы ввода обеспечивают чтение данных с клавиатуры, преобразование из внешнего представления во внутреннее и присваивание их значений переменным, указанным в списке.

При выполнении команды Read или Readln выполнение программы останавливается и компьютер ждет, пока пользователь не введет с клавиатуры нужное количество значений для переменных.

Вводятся только **значения** переменных. При вводе с клавиатуры нескольких значений, они отделяются друг от друга одним или несколькими **пробелами**. Ввод заканчивается нажатием клавиши ENTER.

Readln отличается от Read тем, что после его выполнения автоматически осуществляется переход на следующую строку.

**Операторы вывода данных на экран**

**WRITE(список выражений);**

**WRITELN(список выражений);**

Выражения в списке перечисляются через запятую. В качестве простейших выражений могут быть константы, переменные, символы, строки. Символы и строки выводятся в апострофах. Если необходимо вывести значения переменных или констант (любых, в том числе и символьных), то они выводятся без апострофов.

Оператор **WRITELN** отличается от оператора **WRITE** лишь тем, что после вывода значения последнего выражения из списка осуществляется автоматический переход курсора к началу следующей строки.

**Writeln –** выводит пустую строку

**Управление выводом данных**

При стандартной форме вывода в Object Pascal вещественные числа отображаются на экране с порядком. Мантисса выводится в нормализованном виде - с одной значащей цифрой в целой части и с десятью цифрами в дробной части. На порядок отводится четыре позиции: первая позиция - под букву Е, вторая позиция - под знак порядка, третья и четвертая позиции – под цифры порядка. В PascalABC.Net стандартная форма вывода вещественного числа – десятичная форма.

Если программиста не устраивает стандартная форма вывода, то он может использовать форматированный вывод, предусмотренный в Паскале. При форматированном выводе чисел используются два формата:

а) ширина поля, определяющая число позиций отводимых для вывода всего числа, включая целую, дробную части, знак и десятичную точку;

б) точность представления вещественного числа, определяющая число позиций в дробной части.

Вещественные числа с указанием формата всегда выводятся в десятичной форме. Для целых чисел используется только формат ширины поля. В операторах вывода форматы записываются после выражения и отделяются друг от друга двоеточием. Например, при записи оператора

 **writeln(‘K =’, 8+5:3, ‘ y =’, 7/10:5:1);**

результаты отображаются на экране в следующем виде:

 **K = 13 y = 0.7**

В приведенном примере форматы 3 и 5 задают ширину поля, формат 1 – точность выводимого числа.

**Типы данных**

Понятие типа – одно из фундаментальных понятий в Паскале. Описать или определить идентификатор – это означает указать тип связанного с этим идентификатором объекта программы: константы или переменной. Тип определяет:

* Способ внутреннего представления объекта для компьютера
* Действия, которые разрешаются над ним выполнять.

В программе перед использованием переменной в специальном разделе описаний переменных должно стоять зарезервированное слово VAR. Однотипные переменные в разделе перечисляются через запятую и отделяются от объявленного типа двоеточием. После определения типа ставится точка с запятой.

 **VAR** идентификатор [,идентификатор,…]**:**тип**;**

[идентификатор [,идентификатор,…]**:**тип**;**…]

Например,

 **VAR A : INTEGER;**

 **B, C: REAL;**

**Простые типы данных**

Простой тип данных в определяет конечное линейное множество значений параметра.

Рассмотрим первые 4 типа

**Логический тип**

Логический тип данных- это тип **Boolean**, принимающий значения True и False. Логическое значение в памяти занимает 1 байт. На самом деле, конечно, достаточно и одного бита, но оперировать ячейками меньше байта, мы, к сожалению, не можем. False< True.

**Символьный тип**

Символьные типы обеспечивают хранение отдельных символов. Основной тип данных - **Char**, который содержит символы с кодами 0..255. Занимает в памяти 1 байт. Для кодирования символов используется кодировка Windows-1251 и, что реже – Unicode.

**Вещественный тип**

Вещественные числа могут быть записаны в виде десятичной дроби, привычном для школьника (пример, 0.1 или -5.8125) – дробная часть отделяется от целой точкой. А также в форме с плавающей запятой (с указанием порядка)

**Пример:** даны вещественные числа, необходимо записать, как они выглядят в десятичной форме, в форме с плавающей точкой и в экспоненциальной форме.

|  |  |  |  |
| --- | --- | --- | --- |
| **Вещественное число** | **Десятичная дробь** | **Форма с плавающей точкой** | **Экспоненциальная форма** |
| 3,14\*105 | 314000,0 | 3,14\*105 | 3.14Е5 |
| 42.5\*10-3 | 0,0425 | 4,25\*10-2 | 4.25 Е-2 |
| -15,37 | -15,37 | -1,537\*101 | -1.537Е1 |
| 23 | 23,0 | 2,3\*101 | 2.3Е1 |
| 0.505Е-2 | 0,00505 | 5,05\*10-3 | 5.05Е-3 |
| -28.8Е+4 | -288000,0 | 2,88\*105 | -2.88Е+5 |

Наиболее популярный вещественный тип – **Real** – диапазон значений

От -1,8Е308 до 1,8Е308. Занимает 8 байт памяти.

**Некоторые функции преобразования типа**

Эти функции предназначены для преобразования типов величин, например – символа в целое число, вещественное в целое и пр.

**Chr(a: byte): char** - Преобразует код аргумента **а** в символ в кодировке Windows 1251. Аргумент должен быть в диапазоне **0 .. 255.**

**Ord(a: char): byte** - Преобразует символ в код в кодировке Windows 1251

**chr(33)=!**

**ord(z)=122**

**Round(x:real):integer** – округление значения **вещественного (real)** аргумента **x** до ближайшего **целого.**

**Round(36.6)=37**

**Round(36.4)=36**

**Trunc(x:real):integer** - Выделение целой части **вещественного (real)** аргумента **x**. Тип результата **целый.**

**Trunc(36.6)=36**

**Trunc(36.4)=36**

**Int(x:real):real -** Выделение целой части **вещественного (real)** аргумента **x**. Тип результата **вещественный (real)**

**Int(36.6)=36.0**

**Frac(x:real):real -** Выделение дробной части **вещественного (real)** аргумента **x**. Тип результата **вещественный (real)**

**Frac(36.6)=0.6**

**Арифметические выражения**

**Арифметические выражения** состоят из операндов и операций. Операции находятся между операндами и обозначают действия, которые выполняются над операндами. В качестве операндов выражения можно использовать: переменную, константу, функцию или другое выражение.

**Унарная операция** относится к одному операнду, и ее знак записывается перед операндом, например, -х. Унарный плюс не меняет операнд. +5 – так и останется 5. А унарный минус меняет знак операнда на противоположный.

**Бинарная операция** выражает отношение между двумя операндами, и ее знак записывается между операндами, например, Х+У.

**Арифметические операции**

В бесскобочных арифметических выражениях операции выполняются слева направо в соответствии с их приоритетом.

Приоритет операторов влияет на порядок их выполнения. При вычислении значения выражения в первую очередь выполняются операторы с более высоким приоритетом. Если приоритет операторов в выражении одинаковый, то сначала выполняется тот оператор, который находится левее.

**Приоритет арифметических операций.**

\* (умножение)

/ (деление)

\_DIV\_ (деление целых чисел нацело) 5 div 2 = 2;

\_MOD\_ (выделение остатка от деления целых чисел). 5 mod 2 = 1

Эти перечисленные операции имеют одинаковый приоритет

+ (сложение) и - (вычитание) – тоже имеют одинаковый приоритет, но он более низкий, чем у предыдущих операций.

Изменить порядок выполнения операций можно с помощью круглых скобок. Выражение, заключенное в круглые скобки, выполняется в первую очередь. Например, выражению a/b\*c соответствует математическая запись , а выражению а/(b\*с) соответствует запись .

**Логические операции**

Логические операции применяются к величинам логического типа, результат операции- тоже логический. Имеется одна унарная логическая операция not (логическое отрицание) и три бинарные – and (логическое и), or (логическое или), xor (логическое исключающее или).

|  |  |
| --- | --- |
| **х** | **Not x** |
| FalseTrue | True False |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **x** | **y** | **X and Y** | **X or Y** | **X xor Y** |
| False | False | False | False | False |
| False | True | False | True | True |
| True | False | False | True | True |
| True | True | True | True | False |

**Операции отношения**

Операции отношения предназначены для сравнения двух величин. Результат сравнения имеет логический тип.

= - равно

<> - не равно

< - меньше

> - больше

<= - меньше или равно

>= - больше или равно

К логическим и операциям отношения также применимы круглые скобки для изменения приоритета операций.

**Стандартные функции**

Часто используемые в арифметических выражениях элементарные математические функции оформлены в виде стандартных функций Паскаля.

Для правильного обращения к стандартной функции, необходимо записать имя функции, за которым в круглых скобках следует аргумент (параметр). Функцию можно использовать в качестве операнда выражения, например в инструкции присваивания.

Приоритет вычисления функций выше, чем приоритет арифметических операций.

 Так, чтобы вычислить квадратный корень, достаточно записать k:=Sqrt(n), где Sqrt() — функция вычисления квадратного корня, n — переменная, которая содержит число, квадратный корень которого надо вычислить.

 Ниже представлен набор стандартных функций с указанием типов функции и аргумента. В таблице приняты обозначения: I-целый тип, R- вещественный тип.

У тригономертических функций аргумент вводится в радианах.

|  |  |  |  |
| --- | --- | --- | --- |
| **Математическая запись** | **Запись в Турбо Паскале** | **Тип аргумента** | **Тип функции** |
| **| x |** | **Abs(x)** | IR | IR |
| **x2** | **Sqr(x)** | IR | IR |
| **sin x****cos x****arctg x** | **Sin(x)****Cos(x)****Arctan(x)** | I или R**Аргумент по умолчанию вводится в радианах** | R |
| **ex****ln x** | **Exp(x)****Ln(x)** | I или R | R |
|  | **Sqrt(x)** | I или R | R |
| **Число Пи - π** | **Pi** |  | R |
| **Число Экспонента - Е** | **Е** |  | R |
| **Дополнение к стандартным функциям в PascalABC.Net** |
| **xy** | **Power(x,y)** | I или R | R |
| **tg(x)** | **Tan(x)** | I или R | R |
| **Log2(x)****Lg(x)****LogN(x)** | **Log2(x)****Log10(x)****LogN(n,x)** | I или R | R |
| **Возвращает знак числа** | **Sign(x)**Sign(-4)=-1Sign(7)=1 | I или R | I |
| **Перевод радиан в градусы** | **RadToDeg(x)** | I или R | R |
| **Перевод градусов в радианы** | **DegToRad(x)** | I или R | R |

**Приоритет действий при вычислении выражений**

|  |  |  |
| --- | --- | --- |
| Группа | Тип действия | Операции или элементы |
| 1 | Вычисления в круглых скобках | () |
| 2 | Вычисление значений функций | Функции |
| 3 | Унарные операции | Not, ун+, ун- |
| 4 | Операции типа умножения | \* / div mod and shl shr |
| 5 | Операции типа сложения | + - or xor |
| 6 | Операции отношения | = <> < > <= >= in |

**Задача. Вычислите значение выражения (a+b5)(a−3b)+a2.**

var a,b:integer;
var x1:real;

begin
writeln('Введите значения а и b');
readln(a,b);
x1:=(a+power(b,5))\*(a-3\*b)+a\*a;
writeln;
write('Значение выражения:');
writeln(x1);
end.

**Задача. Сосчитать**

**a div b;**

**c mod ( d div a)**

var a,b,c,d,x1,y1:integer;
begin
writeln('Введите значение а');
readln(a);
writeln('Введите значение b');
readln(b);
writeln('Введите значение c');
readln(c);
writeln('Введите значение d');
readln(d);
x1:=a div b;
y1:= c mod (d div a);
writeln(a,' div ',b,'=',x1);
writeln(c,' mod (',d,' div ',a,')=',y1);
end.

**Задача. Дано целое трехзначное число. Используя операции div и mod, найти сумму его цифр.**

var a,b,c:byte;
var x1,s:integer;
begin
writeln('Введите трёхзначное целое число');
readln(x1);
a:=x1 mod 10;
b:=(x1 div 10) mod 10;
c:=x1 div 100;
s:=a+b+c;
writeln('Сумма:',s);
end.

**Задача. Даны две переменных с некоторыми значениями. Поменять местами значения этих переменных. Использовать два алгоритма: с использованием и без использования вспомогательной переменной)**

**Без использования вспомогательной переменной**

var x,y:integer;
begin
writeln('Введите значения x');
readln(x);
writeln('Введите значения y');
readln(y);
x:=x+y;
y:=x-y;
x:=x-y;
writeln('х=',x,' y=',y);
end.

**С использованием вспомогательной переменной**

var x,y,z:integer;
begin
writeln('Введите значение x');
readln(x);
writeln('Введите значение y');
readln(y);
z:=x;
x:=y;
y:=z;
writeln('х=',x,' y=',y);
end.

**Задача. Дано вещественное число. Округлить это число до ближайшего целого. Выделить из этого числа дробную и целую часть. Определить знак числа.**

var a,b,c,d,k:real;
begin
writeln('Введите вещественное число');
readln(k);
a:=sign(k);
b:=round(k);
c:=int(k);
d:=frac(k);
if x1>0 then writeln('Число положительное')
else writeln('Число отрицательное');
writeln('Оруглённое число:',b);
writeln('Целая часть:',c,' Дробная часть:',d);
end.

**Условные операторы языка Pascal**

В Паскале различают два вида условных операторов (оператора ветвления): неполный (короткий) и полный.

**Неполный (короткий) оператор ветвления (условный оператор)**

**IF <логическое выражение> THEN <оператор P>;** где **оператор P** – любой оператор.

Сначала вычисляется логическое выражение (ЛВ), и если оно имеет значение **TRUE**, то выполняется оператор **P**, стоящий за служебным словом **THEN**. В противном случае оператор **P** игнорируется и сразу переходим к оператору, который следует за условным оператором.

**Полный оператор ветвления (условный оператор)**

**IF <логическое выражение> THEN <оператор P1>**

 **ELSE <оператор P2>;**

где **оператор P1, оператор P2** – любые операторы.

Вычисляется логическое выражение, и если оно имеет значение **TRUE**, то выполняется оператор **P1**, стоящий после служебного слова **THEN**. В противном случае выполняется оператор **P2**, стоящий после служебного слова **ELSE** .

**Графическая интерпретация оператора.** В блок-схемах полному условному оператору соответствует структура **ЕСЛИ – ТО – ИНАЧЕ**.

**Замечание***.*

По определению в конструкцию короткого условного оператора может входить только один оператор P, а в конструкцию полного условного оператора может входить только один оператор Р1 и один оператор Р2.

**Задача. Вычислить и вывести на экран значения функции F в зависимости от введенного х**

****

**где x, a, b – вещественные числа – задаются с клавиатуры**

**var**

F,x,a,b:real;

**begin**

writeln ('введите числа a,b,x:');

 readln (a,b,x);

 **if** (x>=0) **and** (x=-10) **then**

 **if** x=0 **then** writeln ('Делить на ноль нельзя!')

 **else begin**

F:=(sin(x/2))/x;

 writeln ('F= ', F:0:2);

 **end**;

 **if** (x=-1) **and** (x<=-7) **then**

 **if** cos(x-7)=0 **then** writeln ('tg не существует')

 **else begin**

F:=tan(x-7);

 writeln ('F= ', F:0:2);

 **end**

 **else if** x-1=0 **then** writeln ('решений нет!')

 **else begin**

F:=sqr(sqrt(abs(ln(x-1))));

 writeln ('F= ', F:0:2);

 **end**;

**end**.

**Ответы**

**При а=-7, b=-4.6**

|  |  |
| --- | --- |
| **х** | **F** |
| **-10** | **-0.096** |
| **-7** | **-7.245** |
| **-3** | **Решений нет** |
| **-1** | **6.800** |
| **0** | **Деление на 0** |
| **5** | **0.120** |
|  |  |

**Задача. Определить значение U по формуле:**

**Значения x, y, z – вещественные - задаются с клавиатуры.**

**var**

U,x,y,z,max,min:real;

**begin**

writeln('введите числа x,y,z:');

 readln(x,y,z);

 min:=x;

 max:=x;

 **if** y>max **then** max:=y;

 **if** y<min **then** min:=y;

 **if** z>max **then** max:=z;

 **if** z<min **then** min:=z;

 **if** x+y-(z/2)=0 **then** writeln ('Делить на ноль нельзя!')

 **else begin**

U:=(max-5\*min)/x+y-(z/2);

 writeln ('U= ', U);

 **end**;

 **end**.

**Оператор выбора**

При многократном вложении условных операторов программа становится громоздкой и ее трудно понять. Считается, что число вложений не должно превышать двух–трех вложений. При большем числе вложений рекомендуется использовать оператор выбора CASE.

 **CASE <селектор> OF**

 **<список 1>: <оператор 1>;**

 **………………………………………………**

 **<список n>: <оператор n>;**

 **ELSE**

**END;**

**Селектор** – это выражение целого или символьного типа.

**Список** – всевозможные значения селектора, в списке перечисляются через запятую.

По вычисленному значению селектора выбирается для исполнения **оператор** той строки, в списке которой содержится константа, равная значению селектора.

После выполнения выбранного оператора управление передается на конец оператора **CASE**.

Списки могут быть заданы перечислением значений через запятую, например, 2, 5, 10. Или интервалом значений, например, 1..4, 15..67.

**Пример:**

|  |  |
| --- | --- |
| **Вид списка**  |  |
| Список, состоящий из одного значения | ‘a’ 8  |
| Список, содержащий перечисление значений | 1, 4, 5 ‘a’, ‘b’,’\*’,’@’  |
| Список, содержащий интервалы значений | 1**..**5, 20**..**30 ‘a’ **..**’z’  |
| Список, содержащий смешанную форму представления значений | 1 **..** 5, 15, 55‘a’ **..**’z’, ’A’, ’B’  |

**Задача. В восточном календаре принят 60-летний цикл, состоящий из 12-летних подциклов, обозначаемых названиями цвета: зеленый, красный, желтый, белый и черный. В каждом подцикле годы носят названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, курицы, собаки и свиньи. По номеру года определить его название, если 1984 год — начало цикла: «год зеленой крысы».**

**var**

N:integer;

**begin**

write ('Введите год: ');

 readln(N);

 Write ('год ');

**case** (N **mod** 10) **of**

0,1: write ('бел');

 2,3: write ('черн');

 4,5: write ('зелен');

 6,7: write ('красн');

 8,9: write ('желт');

**end**;

 **case** (N **mod** 12) **of**

0..5,9..11: write('ой ');

 6..8: write ('ого ');

 **end**;

 **case** ((N+8) **mod** 12) **of**

0: write ('крысы');

 1: write ('коровы');

 2: write ('тигра');

 3: write ('зайца');

 4: write ('дракона');

 5: write ('змеи');

 6: write ('лошади');

 7: write ('овцы');

 8: write ('обезьяны');

 9: write ('курицы');

 10: write ('собаки');

 11: write ('свиньи');

 **end**;

**end**.

**Задача. Даны два числа и арифметическая операция. Вывести на экран результат этой операции.**

var a,b,s:real;

 z:char;

begin

writeln ('Vvedite A = ');

readln(a);

writeln ('Vvedite znak: + - / \* : ' );

readln(z);

writeln ('Vvedite B = ');

readln(b);

case z of

'+': begin s:=a+b; writeln(a,' + ',b,' = ',s); end;

'-': begin s:=a-b; writeln(a,' - ',b,' = ',s); end;

'\*': begin s:=a\*b; writeln(a,' \* ',b,' = ',s); end;

'/': begin s:=a/b; writeln(a,' / ',b,' = ',s); end;

Else

writeln('error: incorrect operation symbol!!!');

end;

readln;

end.

**Циклы Pascal-Паскаль**

При решении задач может возникнуть необходимость повторить одни и те же действия несколько или множество раз. В программировании блоки кода, которые требуется повторять не единожды, оборачиваются в специальные конструкции – ***циклы***.

**Циклы позволяют многократно выполнять одну или группу команд, причем в тексте программы нет необходимости записывать эти команды несколько раз.**

Можно выделить три **оператора цикла**, присутствующих в том или ином виде во всех языках программирования:

1. **арифметический оператор цикла** (с параметром с шагом 1 и с параметром произвольного шага),
2. **итерационный оператор цикла с предусловием**,
3. **итерационный оператор цикла с постусловием**.

**Арифметический оператор цикла For (цикл с параметром)**

Рассмотрим задачу вычисления суммы первых n членов гармонического ряда:

Y= 1+ 1/2+ 1/3 + …+ 1/ n

При n=5 требуемые вычисления можно задать с помощью оператора присваивания вида:

Y:= 1+1/2+1/3+1/4+1/5

Если значение n заранее не известно и вводится в процессе выполнения программы, то аналогичный оператор присваивания записать невозможно, так как запись вида Y:=  1+1/2+1/3+…+1/ n в языках программирования недопустима.

На самом деле вычисление этой суммы можно осуществить по очень простому и компактному алгоритму: предварительно положим y=0 (с помощью оператора присваивания y:=0), а затем выполним оператор присваивания y:= y+1/ i для последовательных значений i= 1,2,…, n. При каждом очередном выполнении этого оператора к текущему значению y будет прибавляться очередное слагаемое. Как видно, в этом случае процесс вычислений будет носить циклический характер: оператор **y:= y+1/i** должен выполняться многократно, т.е. циклически, при различных значениях i.

Этот пример циклического вычислительного процесса является весьма типичным; его характерные особенности состоят в том, что

* число повторений цикла известно к началу его выполнения (в данном случае оно равно значению **n**, которое предполагается заданным к этому времени);
* управление циклом осуществляется с помощью переменной порядкового типа, которая в этом циклическом процессе принимает последовательные значения от заданного начального до заданного конечного значений (в нашем случае – это целочисленная переменная i, принимающая последовательные значения от 1 до n).

Для компактного задания подобного рода вычислительных процессов и служит **арифметический** **оператор цикла с параметром For**.

Цикл **For** существует в двух формах:

**For** **параметр:=**начальное**\_**значение **to** конечное\_значение **do**

 тело\_цикла;

**For** **параметр:=** начальное\_значение **downto** конечное\_значение **do**

 тело\_цикла;

**For** (для), **to** (увеличиваясь к) и **do** (выполнять, делать) – служебные слова,

**параметр** – переменная порядкового типа, первоначально принимающая начальное значение. Если между начальным и конечным значением указано служебное слово **to**, то на каждом шаге цикла значение параметра будет увеличиваться на единицу. Если же указано **downto**, то значение параметра будет уменьшаться на единицу. Иногда параметр цикла называют **счетчик** цикла

Начальные и конечные значения могут быть не только числами, но и выражениями.

**Тело цикла** - любые операторы Паскаль. Заметим, что в Паскале после **do** должен стоять один оператор, если необходимо выполнить несколько действий, то они должны быть объединены в один составной оператор путем заключения в операторные скобки.

**Особенностью арифметического цикла является то, что число повторений операторов цикла должно быть известно заранее. Решение о выполнении или невыполнении в очередной раз тела цикла принимается до начала его прохождения, поэтому может случиться так, что тело цикла не будет выполнено ни разу.**

Алгоритм выполнения цикла For

1. Вычисляются начальное и конечное значения
2. Параметру цикла присваивается начальное значение
3. Если полученное значение параметра меньше (при to) (больше меньше при downto) конечного значения параметра цикла, то выполняется тело цикла, в противном случае выполнение цикла прекращается.
4. Значение параметра цикла увеличивается (уменьшается) на 1 и переходим к пункту 3.

Пример:

С использованием **оператора цикла с параметром** вычислить суммы первых n членов ряда Y= 1+ 1/2+ 1/3 + …+ 1/ n

Var i:Word;

N,y:integer;

begin

Readln(n);

Y:= 0;

For i:= 1 to n do y:= y+1/i;

Если начальное и конечное значения равны, то тело цикла выполнится один раз. Количество повторений тела цикла можно вычислить по формуле

Конечное\_значение – Начальное\_значение + 1

Количество выполнений тела цикла определяется до первого выполнения тела цикла, поэтому, если в выражении n2 используются переменные, которые изменяются в теле цикла, это никак не повлияет на количество выполнений тела цикла.

**Например**, в следующей последовательности операторов:

n := 10;

For i := 1 to n do

 Begin

 Write (i);

 n := 100;

 End;

тело цикла будет выполняться 10 раз, хотя уже после первого выполнения тела цикла конечное значение параметра цикла станет равным 100.

Заметим так же, что **параметр цикла** может и не использоваться в теле цикла, так что основное его назначение – это управление числом повторений цикла.

Значение счетчика цикла может использоваться в выражениях, входящих в операторы тела цикла, но изменение значения счетчика цикла этими операторами недопустимо

Правильно записаны фрагменты под номерами 1 и 4. Во фрагментах 2 и 3 в операторах тела цикла производится изменение значение счетчика цикла.

**Задача. Вывести на экран значения N!.**

Var i:Word;

N,y:integer;

begin

Readln(n);

Y:= 1;

For i:= 1 to n do y:= y\*i;

Writeln(y:5);

**Задача. Вывести на экран ряд чисел в строку 1001,  1002,  1003,  ... 1025.**

var

 i:integer;

begin

 for i:=1001 to 1025 do write(' ',i);

end.

**Задача. Вывести на экран числа 20, 19, 18, ... до последнего положительного включительно.**

var

 i:integer;

Begin

 For i:=20 downto 1 do

 Writeln(‘ ‘,i);

end.

**Задача. Даны два целых числа A и B (A < B). Вывести все целые числа, расположенные между данными числами (включая сами эти числа), в порядке их возрастания.**

var

 A,B,i:integer;

begin

 writeln('введите A,B ');

read(A, B);

 for i := A to B do write(' ',i);

 writeln;

end.

**Задача. Даны натуральные числа от 40 до 150.Напечатать те из них, которые делятся на 7, но не делятся на 5.**

var

 i:integer;

begin

 for i:=40 to 150 do

 if (i mod 7=0) and (not(i mod 5=0)) then writeln(' ',i);

end.

**Задача. Задано натуральное число N. Найти количество натуральных чисел, не превосходящих N и не делящихся ни на одно из чисел 6 и 3. Выведите эти числа на экран**

var

 n,i,k:integer;

s:real;

Begin

Writeln('Введите n');

Readln(n);

k:=1;

for i:= 1 to n do

If (i mod 6<>0)and(i mod 3<>0) then

k:=k+1;

Writeln('Количество чисел, не превосходящих N и не делящихся ни на одно из чисел 6 и 3 =',k);

End.

**Задача. Составьте программу вывода на экран в порядке убывания всех делителей данного числа. Подсчитайте их количество**

var i, n, k: integer;

begin

 write('n = ');

 readln(n);

 n := abs(n);

 if (n = 0) then

 write('0 имеет бесконечное множество делителей')

 else begin

 k := 1;

 writeln('делители числа в порядке убывания:');

 write(n, ' ');

 for i := (n div 2) downto 1 do

 if (n mod i = 0) then begin

 write(i, ' ');

 end;

writeln;

write('количество делителей числа: ', k)

end;

end.

**Задача. Вычислить сумму квадратов чисел от А до В.**

var

 i,a,b:integer;

 s,el:real;

begin

writeln('Введите a и b');

readln(a,b);

s:=0;

for i:=a to b do begin

 el:=sqr(i);

 s:=s+el;

 end;

writeln('сумма квадртаных корней=',s);

end.

**Задача. Найдите количество прямоугольных треугольников с целочисленными сторонами, меньшими 900.**

var

 a,b,c,k: integer;

begin

for a:=3 to 900 do

for b:=a+1 to 900 do

 begin

 c:=trunc(sqrt(a\*a+b\*b));

 If (c<=100)and(a\*a+b\*b=c\*c) then

 k:=k+1;

 end;

writeln(k);

end.

**Задача. Вычислить и вывести на экран сумму ряда: 1+(1/2)2+(1/3)2+…+(1/n)2.**

var

 i,n:integer;

 s,el:real;

begin

writeln('Введите n');

readln(n);

s:=0;

for i:=1 to n do begin

 el:=power(1/i,3);

 s:=s+el;end;

writeln('Cуммa=',s:0:3);

end.

**Задача. Среди всех двузначных чисел найти те, сумма цифр которых равна n (0<n≤18) и количество этих чисел.**

var n, i, k, p1, p2:integer;

begin

k:=0;

writeln (‘N=’);

readln (n);

for i:=10 to 99 do

 begin

 p1:=i div 10;

 p2:= i mod 10;

 if (p1+ p2) =n then

 begin

 writeln (i);

 k:=k+1;

 end;

 end;

 writeln (‘Kolichestvo=’, k);

readln;

end.

**Итерационные циклы Паскаль**

При составлении программ часто возникают ситуации, когда

1. Заранее не известно количество повторений цикла;

2. Переменная – счетчик цикла должна изменяться с шагом ≠ 1.

В таких случаях используют итерационные циклы.

Итерационные циклы отличаются от циклов с параметром тем, что в них заранее неизвестно число повторений.

Итерационные циклы - это циклическое повторение блока команд, пока выполняется (или не выполняется) некоторое условие. Отсюда получаются два варианта реализации итерационных циклов: с предусловием и с постусловием.

В цикле с предусловием сначала проверяется условие, а потом делается шаг. В цикле с постусловием – сначала шаг, а потом проверка.

**При использовании логических циклов необходимо всегда заботиться о том, чтобы переменная, использующаяся в условии завершения цикла, каким-то образом обязательно изменялась в теле цикла. Это может быть команда присваивания или команда ввода.**

**Оператор цикла Паскаля с предусловием**

**While <** логическое выражение **> do Р,**

где **while** (пока), **do** (делать, выполнять) – служебные слова, Р – оператор. Здесь оператор Р выполняется ноль или более раз, но перед каждым очередным его выполнением вычисляется значение логического выражения, и оператор Р выполняется только в том случае, когда значение логического выражения =true.

Выполнение оператора цикла завершается, когда логическое выражение впервые принимает значение false. Если это значение выражение В принимает при первом же его вычислении, то оператор S не выполнится ни разу.

Оператор цикла Паскаля с предусловием можно считать наиболее универсальным – с использованием таких операторов можно задать и циклические процессы, определяемые операторами цикла с параметром и постусловием.

**Оператор цикла Паскаля с постусловием**

Рассмотрим теперь математическую задачу. Пусть нам необходимо вычислить сумму первых членов гармонического ряда, удовлетворяющих условию 1/i>= K, где 0< K<1, а i=1,2,3…...

Эту задачу можно решить по следующему алгоритму:

положить предварительно y=0 и i=0,

затем в цикле увеличивать i на 1,

к значению y добавлять очередное слагаемое 1/ i до тех пор, пока текущее значение 1/ i впервые окажется больше заданного значения 0< K<1.

Очевидно, что число повторений этого цикла заранее не известно. В подобного рода случаях мы можем лишь сформулировать условие, при выполнении которого процесс добавления к сумме очередного слагаемого должен завершиться. Для задания таких вычислительных процессов и служит оператор цикла Паскаля с постусловием. Этот оператор имеет вид:

**Repeat P1; P2;…; Pi until < логическое выражение >,**

где **repeat** (повторять) и **until** (до) – служебные слова, через **Si** обозначен любой оператор Паскаля.

При выполнении этого оператора цикла последовательность операторов, находящихся между словами **repeat** и **until**, выполнится один или более раз. Этот процесс завершается, когда после очередного выполнения заданной последовательности операторов логическое выражение примет (**впервые**) значение **true**. Таким образом, с помощью логического выражения задается условие завершения выполнения оператора цикла. Поскольку в данном случае проверка условия производится после выполнения последовательности операторов (тела цикла), этот оператор цикла и называется оператором цикла с постусловием.

С использованием этого вида оператора цикла Паскаля задача о суммировании первых членов гармонического ряда, удовлетворяющих заданному условию, может быть реализована следующим образом:

Какой алгоритм выбрать? Это зависит от конкретной задачи.

Если, сделав шаг без проверки, можно получить ошибку, то лучше проверка вначале. Ну, а если шаг без проверки вас не пугает, то можно отложить ее до завершения шага.

Нужно также проанализировать событие, которого мы ожидаем. Если оно может случиться до первого шага, то нужен цикл с предусловием. А если событие не может случиться до первого шага, то нужен цикл с постусловием.

**Сравнение циклов**

|  |  |
| --- | --- |
| **Цикл с предусловием While (пока условие истинно)** | **Цикл с постусловием Repeat (до истинности условия)** |
| * До начала цикла должны быть сделаны начальные установки переменных, управляющих условием цикла, для корректного входа в цикл
 |
| 1. В теле цикла должны присутствовать операторы, изменяющие переменные условия так, чтобы цикл через некоторое число итераций завершился
 |
| 1. Проверка условия выполнения цикла находится в начале цикла
 | 1. Проверка условия выполнения цикла находится в конце цикла
 |
| 1. Цикл работает пока условие истинно (пока True)
 | 1. Цикл работает пока условие ложно (пока False)
 |
| 1. Цикл завершается, когда условие становится ложным (до False)
 | 1. Цикл завершается, когда условие становится истинным (до True)
 |
| 1. Цикл может не выполниться ни разу, если исходное значение условия при входе в цикл False
 | 1. Цикл обязательно выполнится как минимум один раз
 |
| 1. Если в теле цикла требуется выполнить более одного оператора, то необходимо использовать составной оператор
 | 1. Независимо от количества операторов в теле цикла, использование составного оператора не требуется
 |
| **Цикл со счетчиком (с параметром) For** |
| * Начальная установка переменной счетчика цикла до заголовка не требуется
 |
| * Изменение в теле цикла значений переменных, стоящих в заголовке не допускается
 |
| * Количество итераций цикла неизменно и точно определяется значениями нижней и верхней границ и шага приращения
 |
| * Нормальный ход работы цикла может быть нарушен оператором goto или процедурами Break и Continue
 |
| * Цикл может не выполниться ни разу, если шаг цикла будет изменять значение счетчика от нижней границы в направлении, противоположном верхней границе
 |

**Задача. Алгоритм Евклида**

Это алгоритм вычисления наибольшего общего делителя (НОД) двух целых положительных чисел. Он основан на следующих свойствах величин: пусть a и b-одновременно ≠ 0, и пусть a≥ b.

Если b=0, то НОД(a,b) = a,

Если b ≠ 0, то для чисел a,b,c, где c- остаток от деления a на b (c=a mod b) выполнено равенство: НОД(a,b)=НОД(b,c),

Например:

НОД(15,6)=НОД(6,3)=НОД(3,0)=3.

Var a,b,c: integer;

Begin

Writeln (‘Enter a, b. (a>b)’);

readln( a, b);

While b>0 do

 Begin

 c:= a MOD b;

 a:= b;

 b:= c;

 end;

Writeln (‘NOD(a,b)= ‘,a);

Readln;

end.

**Задача. Вывести на экран значения х и функции y(х)=3x2-10 в виде таблицы для каждого значения х. Х меняется на интервале от х1 до х2 с шагом h . Значения х1, х2, и h вводятся с клавиатуры.**

var x,y,x1,x2,h: real;

BEGIN

Readln(x1,x2,h);

x:=x1;

writeln('----------------------------');

writeln('| X | Y | ');

writeln('----------------------------');

writeln;

repeat

  y:=3\*sqr(x)-10;

  writeln('        |   ',x:5:2,'   |    ',y:5:2,'    | ');

  x:=x+h;

until x>x2;

writeln('        ---------------------------');

readln;

END.

**Задача. С клавиатуры вводится натуральное число. Вывести в столбик каждую цифру этого числа и ее номер, а также количество цифр в числе.**

var x, y, c: longint;

 r, k: integer;

Begin

clrscr;

Writeln ('Enter zeloe chislo do 10 snakov');

Readln (x);

writeln;

y:=x;

k:=0;

Writeln ('Zifry zadannogo chisla:');

 repeat

 r:=y mod 10;

 c:=y div 10;

 k:=k+1;

 y:=c;

 Writeln (k,' zifra = ',r);

 until y<1;

Writeln ('Chislo - ',x,' coderjit ', k, ' zifr');

readln;

end.

**Задача. Использовать While.  Дано целое число N (> 1). Вывести наименьшее из целых чисел K, для которых сумма 1+2+… +K будет больше или равна N, и саму эту сумму.**

**var** n, k, s:integer;

**begin**

write('Введите число N: ');

readln(n);

 k:=0;

 s:=0;

 **while** s<n **do**

 **begin**

k:=k+1;

 s:=s+k;

 **end**;

writeln('Число: ', k);

writeln('Сумма: ', s);

**end**.

**Задача. Сосчитать сумму ряда  с точностью разницы абсолютных значений соседних членов ряда 0<ε<1. Вывести номер последнего члена ряда**

**var** el,elpr,eps,s:real; i:integer;

**begin**

write('Введите число E, 0<ε<1: ');

readln(eps);

 i:=2;

 s:=1/2;

 el:=1/4;

 **while** abs(abs(el)-abs(elpr))>=eps **do**

 **begin**

s:=s+el;

 elpr:=el;

 i:=i+1;

 el:=1/(2\*i);

 **end**;

writeln('Сумма: ', s:0:2);

writeln('Номер последнего члена ряда: ', i);

**end**.

**Задача. Вывести на экран в виде таблицы значения х и функции y(х)=5 cos2(x) для каждого значения х. Х меняется на интервале от х1 до х2 с шагом h . Значения х1, х2, и h вводятся с клавиатуры. Использовать цикл While**

**Var** y,x,x1,x2,h:real;

**begin**

write('Введите число X1: ');

 readln(x1);

write('Введите число X2: ');

 readln(x2);

write('Введите шаг h: ');

 readln(h);

writeln(' x | y');

writeln('----------+------------');

 x:=x1;

 **while** x<=x2 **do**

 **begin**

y:=5\*sqr(cos(x));

 writeln(' ',x:5:1,' | ',y:6:2);

 x:=x+h;

 **end**;

**end**.

**Задача. Дано целое число N (> 1). Вывести наименьшее целое K, при котором выполняется неравенство 3K > N, и сами значения K и 3К. Использовать два способа решения: с помощью цикла While и Repeat.**

**var** n,k:integer;

**begin**

Write('Введите число N, N>1: ');

readln(n);

writeln('Первый способ:');

k:=0;

**repeat**

k:=k+1;

**until** 3\*k>n;

writeln('K = ',k);

writeln('3K = ',3\*k);

writeln('Второй способ:');

k:=1;

**while** 3\*k<=n **do**

k:=k+1;

writeln('K = ',k);

writeln('3K = ',3\*k);

**end**.

**Задача. Написать программу, которая находит все двузначные числа и их количество, которые делятся на n или содержат цифру n (0<n<10). Использовать два способа решения: с помощью цикла While и Repeat.**

**var** i,n,k:integer;

**begin**

write('Введите цифру n(0<n<10): ');

readln(n);

writeln('Первый способ:');

i:=10; k:=0;

**repeat**

 **if** (i **mod** n=0) **or** (i **div** 10=n) **or** (i **mod** 10=n) **then**

 **begin**

write(i, ' ');

 k:=k+1;

 **end**;

 i:=i+1;

**until** i>99;

writeln;

writeln('Количество: ',k);

writeln('Второй способ:');

i:=10; k:=0;

**while** i<=99 **do**

 **begin**

 **if** (i **mod** n=0) **or** (i **div** 10=n) **or** (i **mod** 10=n) **then**

 **begin**

write(i, ' ');

 k:=k+1;

 **end**;

 i:=i+1;

 **end**;

writeln;

writeln('Количество: ',k);

**end**.

**Задача. Выведите на экран таблицу умножения для чисел от 1 до 10. Использовать два способа решения: с помощью цикла While и Repeat.**

**var**

i,j:integer;

**begin**

writeln('Первый способ:');

write(' |');

 i:=1;

 **repeat**

write(i:4);

 i:=i+1;

 **until** i>10;

 writeln;

 i:=1;

 **repeat**

write('----');

 i:=i+1;

 **until** i>11;

 writeln;

 i:=1;

 **repeat**

write(i:3,'|');

 j:=1;

 **repeat**

write(i\*j:4);

 j:=j+1;

 **until** j>10;

 writeln;

 i:=i+1;

 **until** i>10;

writeln;

writeln('Второй способ:');

 write(' |');

 i:=1;

 **while** i<=10 **do begin**

write(i:4);

 i:=i+1;

 **end**;

 writeln;

 i:=1;

 **while** i<=11 **do begin**

write('----');

 i:=i+1;

 **end**;

 writeln;

 i:=1;

 **while** i<=10 **do**

 **begin**

write(i:3,'|');

 j:=1;

 **while** j<=10 **do begin**

write(i\*j:4);

 j:=j+1;

 **end**;

 writeln;

 i:=i+1;

 **end**;

**end**.

**Задача. Вывести на экран значения х,у и функции z(х,y)=5x+y2 в виде таблицы для каждого значения х и у. Х меняется на интервале от х1 до х2 с шагом 1 . Y меняется на интервале от y1 до y2 с шагом h .Значения х1, х2, y1, y2 и h вводятся с клавиатуры.**

**Var** x,x1,x2:integer; y,y1,y2,h,z:real;

**begin**

write('Введите число X1: ');

 readln(x1);

write('Введите число X2: ');

 readln(x2);

 write('Введите число Y1: ');

 readln(y1);

write('Введите число Y2: ');

 readln(y2);

write('Введите шаг h: ');

 readln(h);

writeln(' x | y | z');

writeln('-------+---------+---------');

x:=x1;

**repeat**

y:=y1;

**repeat**

z:=5\*x+2\*y;

 writeln(' ',x:3,' | ',y:5:1,' | ',z:6:2);

 y:=y+h;

**until** y>y2;

 x:=x+1;

**until** x>x2;

**end**.

**Операторы завершения цикла**

Данные операторы применяются внутри операторов цикла с параметром или условием. Операторы имеют вид: **Continue**; - ограничение цикла, **Break**; - прерывание цикла.

Операторы Continue и Break позволяют производить действия не для всех операторов внутри цикла. Действие оператора Continue заключается в передаче управления на начало цикла, при этом контролируется условие выхода из цикла. Действие оператора Break заключается в передаче управления оператору, следующему за последним оператором цикла, при этом не контролируется условие выхода из цикла.

Во вложенных циклах операторы Continue и Break действуют только на цикл в котором они записаны. Приведем пример использования операторов для блокировки несанкционированного доступа в программу.

**Задача. Выписать все простые числа, не превышающие данного числа N.**

**Var**

X, N, i:integer; // проверяемое число, указанное число , счетчик

b:boolean;{true-число простое false-число не простое}

**Begin**

x:=2; //задаём начальное значение

Write('N=');

readln(N);

Writeln(1); // выводим 1

**While** x<=N **do begin**

b:=true; // предпологаем что число простое}

**for** i:=2 **to** x-1 **do** // перебираем все числа от 2 до x

**begin**

**if** (x **mod** i)=0 **then** {проверяем остаток от деления если число делится без остатка значит оно не простое и цикл можно прервать}

 **begin**

b:=false;

 **break**;

 **end**;

**end**;

**if** b=true **then** Writeln(x);

X:=X+1;

**end**

**end**.

## Раздел 3. Структурированные типы данных

**Тема 3.1.  Понятие массива**

**Методические рекомендации по выполнению практических заданий**

Для выполнения практических заданий студентам необходимо знать понятия: одномерный массив, двумерны массив, индекс.

Примеры решения задач представлены ниже в тексте.

**ОДНОМЕРНЫЕ МАССИВЫ**

Предположим, что программа работает с большим количеством однотипных данных. Скажем около ста разных целых чисел нужно обработать, выполнив над ними те или иные вычисления. Как вы себе представляете 100 переменных в программе? И для каждой переменной нужно написать одно и тоже выражение вычисления значения? Это очень неэффективно.

Есть более простое решение. Это использование такой структуры (типа) данных как **массив**. Массив представляет собой последовательность ячеек памяти, в которых хранятся однотипные данные. При этом существует всего одно имя переменной связанной с массивом, а обращение к конкретной ячейке происходит по ее индексу (номеру) в массиве.

Нужно четко понимать, что индекс ячейки массива **не** является ее содержимым. Содержимым являются хранимые в ячейках данные, а индексы только указывают на них. Действия в программе над массивом осуществляются путем использования имени переменной, связанной с областью данных, отведенной под массив.

Итак, **Массив — это структура данных, представляющая собой набор переменных одинакового типа, имеющих общее имя.**

Массив – это именованная группа однотипных данных, хранящихся в последовательных ячейках памяти. Каждая ячейка содержит элемент массива. Элементы нумеруются по порядку, но необязательно начиная с единицы (хотя в языке программирования Pascal чаще всего именно с нее). Порядковый номер элемента массива называется индексом этого элемента.

Массивы состоят из ограниченного числа компонент. Все элементы одного массива имеют один и тот же тип. У разных массивов типы данных могут различаться. Например, один массив может состоять из чисел типа integer, а другой – из чисел типа real.

 Индексы элементов массива обычно целые числа. Индексировать можно как константами и переменными, так и выражениями, результат вычисления которых дает значение перечислимого типа.

Массив в Delphi обозначается как **array**. Чтобы объявить переменную типа массива нужно описать её в разделе **var** следующим образом:

var Имя\_массива: array [длина массива] of тип данных;

где:

имя — имя массива;

array — зарезервированное слово языка Delphi, обозначающее, что объявляемое имя является именем массива;

длина массива — количество элементов (размер) массива, задается в [], в которых объявляется начальный и конечный индекс массива.

тип — тип элементов массива.

Примеры объявления массивов:

temp:array[1..31] of real;

кf:array [0. .2] of integer;

name:array[1..30] of char;

Обращение к определенному элементу массива осуществляется путем указания имени переменной массива и в квадратных скобках индекса элемента. Номер элемента массива называется **индексом. Индекс** – это значение порядкового типа, определенного, как тип индекса данного массива. Очень часто это целочисленный тип.

С помощью индекса массива можно обращаться к отдельным элементам любого массива, как к обычной переменной: можно получать значение этого элемента, отдельно присваивать ему значение, использовать его в выражениях.

 Индекс массива в Паскале не обязательно задавать в явном виде. В качестве индекса массива можно использовать переменную или выражение, соответствующее индексному типу. Иначе говоря, индексы можно вычислять.

Этот механизм – весьма мощное средство программирования. Но он порождает распространенную ошибку: результат вычислений может оказаться за пределами интервала допустимых значений индекса, то есть будет произведена попытка обратиться к элементу, которого не существует. Эта типичная ошибка называется «выход за пределы массива».

Простой массив является одномерным. Он представляет собой линейную структуру.

**Ввод массива Паскаля**

 Для того чтобы ввести значения элементов массива, необходимо последовательно изменять значение индекса, начиная с первого до последнего, и вводить соответствующий элемент. Для реализации этих действий удобно использовать цикл с заданным числом повторений, т.е. простой арифметический цикл, где параметром цикла будет выступать переменная – индекс массива Паскаля. Значения элементов могут быть введены с клавиатуры или определены с помощью оператора присваивания.

Пример фрагмента программы ввода массива Паскаля

program massiv1;

Var

 A : array [1..10] of integer ;

 I : byte; {переменная I вводится как индекс массива}

 Begin

 for i:=1 to 10 do readln (a[i]);

 for i := 1 to 10 do write (a[i]:3);

readln

end.

Рассмотрим теперь случай, когда массив Паскаля заполняется автоматически случайными числами, для этого будем использовать функцию **random (N)**.

Процедура **Randomize** используется в Паскаль для включения генератора случайных чисел. Функция **Random** определяет диапазон случайных чисел. Процедура Randomize и функция Random очень часто используются для демонстрации работы массивов в Паскаль.

Пример фрагмента программы заполнения массива Паскаля случайными числами

program massiv2;

var

 A : array [1..15] of integer ;

 I : byte; {переменная I вводится как индекс массива}

begin

randomize;

 for i:=1 to 15 do

 begin

 a[i]:=random(10); { i -му элементу массива присваивается «случайное» целое число в диапазоне от 0 до 10}

 writeln(‘a[‘,I,’]=’,a[i]:2);

 end;

readln

end.

**Основные действия с массивами Паскаля**

Как известно, определение типа данных означает ограничение области допустимых значений, внутреннее представление в ЭВМ, а также набор допустимых операций над данными этого типа. Мы определили тип данных как массив Паскаля. Какие же операции определены над этим типом данных? **Единственное действие, которое можно выполнять над массивами целиком, причем только при условии, что массивы однотипны, – это присваивание**. Если в программе описаны два массива одного типа, например,

 Var

 a , b : array [1..10] of real ;

то можно элементу массива a присвоить значение соответствующего элемента массива b

 a := b;

**Все остальные действия над массивами Паскаля производятся поэлементно (это важно!).**

**Вывод массива Паскаля**

Вывод массива в Паскале осуществляется также поэлементно, в цикле, где параметром выступает индекс массива, принимая последовательно все значения от первого до последнего.

 Вывод можно осуществить и в столбик с указанием соответствующего индекса. Но в таком случае нужно учитывать, что при большой размерности массива все элементы могут не поместиться на экране и будет происходить скроллинг, т.е. при заполнении всех строк экрана будет печататься очередной элемент, а верхний смещаться за пределы экрана.

При выполнении программы massiv2 на экране мы увидим, к примеру, следующие значения:

 a [1]=2

 a [2]=4

 a [3]=1 и т.д.

Функция sizeof, примененная к имени массива или имени массивного типа, возвращает количество байтов, отводимое под массив.

Пример программы с ошибкой массива Паскаля

 Program massive\_error ;

var

 A : array [1..6] of integer ;

 I : byte; {переменная I вводится как индекс массива}

begin

 i:=3;

 a[i\*3]:=25;

 end .

Хотя данная программа полностью соответствует синтаксису языка, и транслятор «пропустит» ее, на стадии выполнения произойдет ошибка выхода за пределы массива Паскаля. При i:=5 выражение i\*3=9, компьютер сделает попытку обратиться к элементу массива a [9], но такого элемента нет, поскольку описан массив размерностью 6.

Будем считать, что хорошая программа должна выдавать предупреждающее сообщение в случае попытки обращения к несуществующим элементам массива. Не лишним будет проверять возможный выход как за правую, так и за левую границы массива, ведь не исключено, что в результате вычисления значения выражения получится число, находящееся левее границы массива Паскаля.

Из всего этого следует сделать вывод: программисту надо быть очень аккуратным при работе с индексами массива.

**Задача. Требуется заполнить массив числами, которые вводит пользователь, и вычислить их сумму. Если пользователь вводит ноль или превышен размер массива, то запросы на ввод должны прекратиться.**

**Алгоритм решения задачи:**

Для решения подобного рода задач иногда бывает уместно использовать цикл с постусловием (repeat).

Программа на языке Паскаль:

const n = 10;

var

 arr: array[1..n] of integer;

 sum: integer;

 i: byte;

begin

 i := 1;

 sum := 0;

 repeat

 write (I,'-e число = ');

 readln(arr[i]);

 sum := sum + arr[i];

 i := i + 1

 until (arr[i-1] = 0) or (i > n);

 writeln

 for i := 1 to n do

 write (arr[i],' ');

 writeln;

 writeln ('sum = ', sum);

readln

end.

**Задача Дан массив из 10 чисел, которые вводятся с клавиатуры. Вывести на экран значения элементов массива. Сосчитать произведение всех отрицательных элементов массива.**

**var**

i,kol:integer;

a:**array** [1..10] **of** integer;

**begin**

kol:=0;

writeln('заполните массив:');

**for** i:=1 **to** 10 **do**

**begin**

readln(a[i]);

**if** a[i]<0 **then** kol:=kol+1;

**end**;

writeln(a,' кол-во отриц. элементов=',kol);

**end**.

**Задача Дан М[1..10], Элементы массива вычисляются по формуле: М[i]=. Выведите все элементы массива на экран. Сосчитайте количество элементов массива стоящих на четных позициях (с четными индексами) и которые больше 3-х.**

**var**

i,kol:integer;

m:**array** [1..10] **of** real;

**begin**

kol:=0;

**for** i:=1 **to** 10 **do**

**begin**

m[i]:=sqrt(2\*i);

**if** (i **mod** 2=0) **and** (m[i]>3) **then** kol:=kol+1;

writeln('a[',m[i]:0:4,']');

**end**;

writeln('кол-во элементов массива стоящих на четных позициях (с четными индексами) и которые больше 3-х=',kol);

**end**.

**Задача Дан массив из 15 чисел. Вывести на экран первоначальные значения элементов массива. Отсортировать значения элементов массива по возрастанию простым выбором. Вывести на экран новые значения элементов массива**

**var**

i,mi,j,j2:integer;

a:**array** [1..15] **of** integer;

**begin**

**for** i:=1 **to** 15 **do**

a[i]:=random(101);

 writeln(a,'-первоначальное значение масива');

 **for** i:=1 **to** 14 **do begin**

mi:=a[i];

 j2:=i;

 **for** j:=i+1 **to** 15 **do**

 **if** a[j]<mi **then**

 **begin**

mi:=a[j];

 j2:=j;

 **end**;

 a[j2]:=a[i];

 a[i]:=mi;

 **end**;

 writeln(a,'-полученное значение массива');

 **end**.

**Двумерные массивы**

Двумерный массив в Паскале трактуется как одномерный массив, тип элементов которого также является массивом (массив массивов). Положение элементов в двумерных массивах Паскаля описывается двумя индексами. Их можно представить в виде прямоугольной таблицы или матрицы.

Каждый элемент имеет свой номер, как у одномерных массивов, но сейчас номер уже состоит из двух чисел – номера строки, в которой находится элемент, и номера столбца. Таким образом, номер элемента определяется пересечением строки и столбца. Например, a[2,1] – это элемент, стоящий во второй строке и в первом столбце.

Описание двумерного массива Паскаля.

Var

 m: array [1..5,1..10] of <тип\_элементов>;

Обращение к элементам двумерного массива имеет вид: M [i,j]. Это означает, что мы хотим получить элемент, расположенный в i-й строке и j-м столбце. Тут главное не перепутать строки со столбцами, а то мы можем снова получить обращение к несуществующему элементу. Например, обращение к элементу M [10, 5] имеет правильную форму записи, но может вызвать ошибку в работе программы.

**Основные действия с двумерными массивами Паскаля**

Все, что было сказано об основных действиях с одномерными массивами, справедливо и для матриц. Единственное действие, которое можно осуществить над однотипными матрицами целиком – это присваивание. Т.е., если в программе у нас описаны две матрицы одного типа, например,

type

 matrix= array [1..5, 1..10] of integer;

 var

 a , b : matrix ;

 то в ходе выполнения программы можно присвоить матрице a значение матрицы b (a := b). Все остальные действия выполняются поэлементно, при этом над элементами можно выполнять все допустимые операции, которые определены для типа данных элементов массива.

Это означает, что если массив состоит из целых чисел, то над его элементами можно выполнять операции, определенные для целых чисел, если же массив состоит из символов, то к ним применимы операции, определенные для работы с символами.

**Ввод двумерного массива Паскаля.**

Для последовательного ввода элементов одномерного массива мы использовали цикл **for**, в котором изменяли значение индекса с 1-го до последнего. Но положение элемента в двумерном массиве Паскаля определяется двумя индексами: номером строки и номером столбца. Это значит, что нам нужно будет последовательно изменять номер строки с 1-й до последней и в каждой строке перебирать элементы столбцов с 1-го до последнего. Значит, нам потребуется два цикла for , причем один из них будет вложен в другой.

Пример программы ввода двумерного массива Паскаля с клавиатуры

 type

 matrix= array [1..5, 1..10] of integer;

 var

 a, : matrix;

 i, j: integer; { индексы массива }

 begin

 for i :=1 to 5 do {цикл для перебора всех строк}

 for j :=1 to 10 do {перебор всех элементов строки по столбцам}

 readln ( a [ i , j ]); {ввод с клавиатуры элемента, стоящего в i -й строке и j -м столбце}

end,

Двумерный массив Паскаля можно заполнить случайным образом, т.е. использовать функцию random (N), а также присвоить каждому элементу матрицы значение некоторого выражения. Способ заполнения двумерного массива Паскаля выбирается в зависимости от поставленной задачи, но в любом случае должен быть определен каждый элемент в каждой строке и каждом столбце.

**Вывод двумерного массива Паскаля на экран.**

Вывод элементов двумерного массива Паскаля также осуществляется последовательно, необходимо напечатать элементы каждой строки и каждого столбца. При этом хотелось бы, чтобы элементы, стоящие в одной строке, печатались рядом, т.е. в строку, а элементы столбца располагались один под другим. Для этого необходимо выполнить следующую последовательность действий (рассмотрим фрагмент программы для массива, описанного в предыдущем примере):

Пример программы вывода двумерного массива Паскаля

program Mas2;

type

 matrix=array [1..3, 1..4] of integer;

var

 a:matrix;

 i,j:integer;

begin

randomize;

for i:=1 to 3 do

 begin

 for j :=1 to 4 do

 begin

 a[i,j]:=random(20);

 write('a[',i,',',j,']=',a[i,j]:2,' ');

 end;

 writeln;

 end;

 readln

end.

Замечание (это важно!): очень часто в программах студентов встречается ошибка, когда ввод с клавиатуры или вывод на экран массива пытаются осуществить следующим образом: readln (a), writeln (a), где а – это переменная типа массив. При этом их удивляет сообщение компилятора, что переменную этого типа невозможно считать или напечатать. Может быть, вы поймете, почему этого сделать нельзя, если представите N кружек, стоящих в ряд, а у вас в руках, например, чайник с водой. Можете вы по команде «налей воду» наполнить сразу все кружки? Как бы вы ни старались, но в каждую кружку придется наливать отдельно. Заполнение и вывод на экран элементов массива также должно осуществляться последовательно и поэлементно, т.к. в памяти ЭВМ элементы массива располагаются в последовательных ячейках.

**Задача. Дана матрица 10х10. Заполнить эту матрицу значениями от 1 до 100 последовательно по столбцам, начиная с позиции [1,1] сверху-вниз с 1-го по 10-й столбец. Например в матрице 4х4 это выглядит так:**

**1 5 9 13**

**2 6 10 14**

**3 7 11 15**

**4 8 12 16**

**const** m=10;

**type** Myarray=**array**[1..m,1..m]**of** integer;

**var**

i,j,l:integer;

Mar:Myarray;

{--------------------------------------------------}

**procedure** Input(A:Myarray; m:integer);

**begin**

**var** i,j,l:integer;

**for** j:=1 **to** m **do**

**for** i:=1 **to** m **do**

 **begin**

Mar[i,j]:=l;

 l:=l+1;

 **end**;

**end**;

{-----------------------------------}

**procedure** Output(A:Myarray; m,n:integer);

**begin**

**var** i,j:integer;

write('I\J ');

**for** j:=1 **to** n **do**

write(j:4);

writeln;

writeln(' -----------------------------------------');

**for** i:=1 **to** m **do**

 **begin**

write(i:3,' |');

 **for** j:=1 **to** n **do**

write(A[i,j]:4);

 writeln;

 **end**;

**end**;

{----------------------}

**begin**

{Формирование массива}

Input(Mar,m);

{Вывод массива}

Output(Mar,m,m);

**end**.

**Задача. Дан двумерный массив случайных чисел от -5 до 14 из 5-ти строк и 6-ти столбцов. Вывести на экран первоначальные значения элементов массива в виде матрицы. Все отрицательные элементы массива заменить на 0 в каждом четном столбце матрицы. Вывести на экран новые значения элементов массива в виде матрицы.**

**const** m=5;

n=6;

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

**var**

i,j:integer;

Z:Myarray;

{--------------------------------------------}

**procedure** Fill\_Random(A:Myarray; m,n:integer);

**begin**

**var** i,j:integer;

**for** i:=1 **to** m **do**

**for** j:=1 **to** n **do**

Z[i,j]:=random(-5,14);

**end**;

{-----------------------------------}

**procedure** Output(A:Myarray; m,n:integer);

**begin**

**var** i,j:integer;

write('I\J ');

**for** j:=1 **to** n **do**

write(j:4);

writeln;

writeln(' -----------------------------');

**for** i:=1 **to** m **do**

 **begin**

write(i:3,' |');

 **for** j:=1 **to** n **do**

write(A[i,j]:4);

 writeln;

 **end**;

**end**;

{------------------------------------}

**begin**

{Формирование массива}

Fill\_Random(Z,m,n);

Output(Z,m,n);

writeln;

{Обработка массива}

**for** i:=1 **to** m **do**

 **for** j:=1 **to** 3 **do**

 **if** Z[i,2\*j]<0 **then**

 Z[i,2\*j]:=0;

Output(Z,m,n);{Новые значения массива}

**end**.

**Тема 3.2. Строковый тип данных**

**Методические рекомендации по изучению лекций и литературы**

Ключевыми для данной темы является следующие понятия: символ, строка, длина строки.

Студенты должны выучить функции преобразования типов, функции и процедуры работы со строками, их синтаксис и семантику.

**Строки**

Строки используются для хранения последовательностей символов. Она представляет собой особую форму одномерного массива, состоящего из символов. Но строка отличается от массива тем, что:

1. Массив имеет фиксированную длину:

 var m : array[1..20] of char;

 и у каждого элемента есть свое значение или 0.

2. Строка символов имеет две разновидности длины:

-общая длина строки, которая определяет величину памяти, выделяемый строке при описании;

-текущая длина строки (всегда **≤** общей длины) количество смысловых символов строки в данный момент времени.

-max длина строки 255 символов.

Под каждый символ отводится по одному байту, в котором хранится код символа. Еще один байт отводится под фактическую длину строки. Для коротких строк использовать стандартную строку неэффективно, поэтому в язык введена возможность самостоятельно задавать максимальную длину строки, например:

**type**

 str4 = string [4];

Здесь описан собственный тип данных с именем str4. Переменная этого типа занимает в памяти 5 байт. Длина строки должна быть константой или константным выражением.

Примеры описания строк:

**var**

 slovo : string; { Если в объявлении переменной описании длина строки не указывается, то она =255 – это строка стандартого типа }

 s1 : str4; { строка типа str4, описанного выше }

 s2 : string [15]; { описание типа задано при описании переменной }

Например, присвоим переменной значение:

**S2:= ‘весна’;**

В большинстве случаев переменные типа **String** исполняются для хранения слов и различных сообщений.

**Операции над строками**

Строки можно **присваивать** друг другу. Если максимальная длина результирующей строки меньше длины исходной, лишние символы справа отбрасываются:

S2:= ‘весна’;

s1:= s2; { в s1 будут помещены символы "весн" }

Строки можно **складывать** между собой с помощью операции конкатенации, которая обозначается знаком +, например:

s1 := 'ком';

s2 := s1 + 'пот'; { результат - "компот" }

или

s1 := 'ком';

s2 := 'пот';

slovo:=s1+s2; { результат - "компот" }

Строки можно сравнивать друг с другом с помощью **операций сравнения**. При сравнении строки рассматриваются **посимвольно слева направо**, при этом сравниваются коды соответствующих пар символов. Строки равны, если они имеют одинаковую длину и посимвольно эквивалентны:

'abc' > 'ab'

'abc' = 'abc'

'abc' < 'abc '

Имя строки может использоваться в операторах ввода **read и readln** и вывода **write и writeln**:

readln (s1, s2);

write (s1);

**При вводе** в строку считывается из входного потока количество символов, равное длине строки или меньшее, если символ перевода строки (Enter) встретится раньше.

**При выводе** под строку отводится количество позиций, равное ее фактической длине.

Переменная типа **String** состоит из цепочки символов, т.е. элементов типа **Char**

К отдельному символу строки можно обращаться как к элементу **массива символов**, например, **s1[4]**. Один символ строки имеет тип **char**, их можно использовать в выражениях одновременно, например:

s2:= ‘весна’;

тогда s1[3] = c

Таким образом существует 2 способа присваивать значения переменным типа строка:

1. **Как единый объект**

slovo:=‘университет’;

присваиваем сразу целую строку

stroka:=‘ННГУ’+’\_’+’им.Н.И.Лобачевского’

word:= slovo1 + slovo2;

1. **Строка- это совокупность символов (элементов типа Char)**, которые при обработке доступны каждый в отдельности. Доступ к отдельному элементу строки аналогичен доступу к отдельному элементу массива.

s2:= ‘весна’;

s2[4]:=‘т’**-** четвертому символу переменной s2 присвоили значение – т, получили s2 = ‘веста’;

Т.о., для доступа к отдельному элементу строки необходимо указать имя этой сроки и номер элемента в строке в [ ].С отдельным элементом строки возможно проделывать все те операции, что и с переменными типа **Char**.

**Стандартные функции работы со строками**

**Length()**

Функция length(str) - определяет фактическую длину строки str. результат имеет тип byte.

переменная := length (строка)

переменная – имя переменной типа integer

строка – имя строки, длина которой определяется.

Program one;

var N: integer; word: string;

begin

word:=‘информатика’;

n:=length(word);

write (N);

end.

**Copy()**

Функция **Copy(s, start, len)** возвращает подстроку длиной **len**, начинающуюся с позиции **start** строки **s**.

**Str:=Copy(s, start, len);**

**s** - исходная строка,

**start** - позиция, начиная с которой будет копироваться фрагмент,

**len** - число копируемых символов,

**str** - полученный фрагмент строки

Параметры len и start должны быть целого типа, str и s строковые.

**Пример:**

program p2;

var word; w1, w2, w3: string;

begin

word:=‘ картографирование’;

w1:=copy (word, 6,4);

writeln (w1);

w2:=copy (word, 2,3);

writeln (w2);

w3:=copy (word, 11, 3);

writeln (w3);

end.

**Pos()**

Функция **Pos(subs, s)** ищет вхождение подстроки subs в строку s и возвращает номер первого символа subs в s или 0, если subs не содержится в s.

program p3;

var word, w: string; N: integer;

begin

word:=‘электрификация’;

writeln (‘введите фрагмент для поиска’);

readln (w);

N= pos (w, word);

if N < > 0 then writeln (‘позиция =’,N);

else writeln (‘такого фрагмента нет’);

end.

**Concat()**

Функция **Concat(s1, s2, ..., sn)** возвращает строку, являющуюся слиянием строк s1, s2, ..., sn. Ее действие аналогично операции конкатенации.

**Стандартные процедуры работы со строками**

**Insert ()**

Процедура **Insert(subs, s, start)** вставляет в строку s подстроку subs, начиная с позиции start.

**Delete ()**

Процедура **Delete(s, start, len)** удаляет из строки s, начиная с позиции start, подстроку длиной len.

Program P4;

var word, word1: string;

begin

word:=‘колледж;

writeln (word); //колледж

delete (word,1,3);

writeln (word); //ледж

delete (word,4,1);

writeln (word); //лед

word1:=‘п’;

insert (word1, word, 1);

writeln (word); //плед

word1:=‘аа’;

insert (word1, word, 3);

writeln (word); //плааед

end.

**Процедуры преобразования типов в Паскале**

**Str()**

**Str (x, S)** преобразует число x в строковый формат. Здесь x – любое числовое выражение, S – строковая переменная. В процедуре есть возможность задавать формат числа x. Например, str(x: 8: 3, S), где 8 – общее число знаков в числе x, а 3 – число знаков после запятой.

Str (sin(1):6:4, S) – S=‘0.0175’

Str (3456, S) – S=‘3456’

**Val()**

**Val(S, x, kod)** преобразует строку символов S в число x. Здесь S – строковое выражение, x – числовая переменная (именно туда будет помещен результат), kod – целочисленная переменная (типа integer), которая равна номеру позиции в строке S, начиная с которой произошла ошибка преобразования, если преобразование прошло без ошибок, то переменная kod равна 0.

Тип X Оператор процедуры Значение X Значение kod

Real Val(’12.34’, x, kod) 12.34 0

Integer Val(’12.34’, x, kod) 12 3

**Задача. Дана строка, состоящая из русских слов, набранных заглавными буквами и разделенных пробелами (одним или несколькими). Найти количество слов, которые содержат хотя бы одну букву, введенную с клавиатуры.**

**var** a:string;

 c,help:char;

 i,x:integer;

**begin**

write('Введите строку=');

 readln(a);

 write('Введите букву=');

 readln(c);

 x:=0;

 help:=' ';

 i:=1;

 **while** i<length(a) **do**

 **begin**

 **if** (a[i]=c) **or** (a[i]=uppercase(c)) **then**

 **begin**

x:=x+1;

 **if** i<>length(a) **then**

 **repeat**

i:=i+1;

 **until** a[i]=help;

 i:=i+1;

 **end**

 **else** i:=i+1;

 **end**;

 writeln('Слов в предложении с буквой ',c,'=',x);

**end**.

**Задача Дана непустая строка S. Вывести строку, содержащую символы строки S, между которыми вставлено по одному пробелу**

**var** s1,s2,help:string;

 i,j:integer;

**begin**

write('Введите строку=');

 readln(s1);

 j:=1;

 j:=1;

 help:=' ';

 **for** i:=1 **to** length(s1) **do**

 **begin**

s2:=s2+s1[i]+help;

 **end**;

 writeln(s2);

**end**..

**Задача. Даны строки S, S1 и S2. Заменить в строке S первое вхождение строки S1 на строку S2.**

**var**

p:integer;

 s,s1,s2:string;

**begin**

write('Введите строку s=');

readln(s);

write('Введите строку s1=');

readln(s1);

write('Введите строку s2=');

readln(s2);

p:=pos(S1,S);

delete(s,p,length(s1));

insert(s2,s,p);

writeln('S=',s);

**end**.

**Задача. С клавиатуры вводится строка, состоящая из слов, между которыми один, два, три и более пробелов. Удалить в строке все лишние пробелы. Лишними считаются пробелы, следующие непосредственно за пробелами. Т.е. между словами всегда должен находиться один пробел.**

**var**

s:string;

 i:byte;

**begin**

write('Введите строку:');

readln(s);

i := 1;

**repeat**

 **if** copy(s,i,2)=' ' **then**

s := copy(s,1,i) + copy(s,i+2,length(s))

 **else**

i := i + 1

 **until** (i > length(s));

writeln('Итог: ', s);

**end**.

# Литература для учащегося

**Основные источники:**

1. 1. Семакин И.Г. Основы алгоритмизации и программирования: учебник для студ.учреждений СПО / И.Г.Семакин, А.П.Шестаков. – 3-е изд., - М.: Издательский дом «Академия», 2012. – 400с. Допущено Министерством образования и науки РФ в качестве учебника для студентов образовательных учреждений среднего профессионального образования. - Режим доступа: <http://znanium.com/catalog.php?bookinfo=507976>
2. 2. Голицына О.Л. Языки программирования: Учебное пособие / О.Л. Голицына, Т.Л. Партыка, И.И. Попов. - 3-e изд., перераб. и доп. - М.: Форум: ИНФРА-М, 2015. - 400 с. Допущено Министерством образования и науки РФ в качестве учебника для студентов образовательных учреждений среднего профессионального образования - Режим доступа: <http://znanium.com/catalog.php?bookinfo=507976>

**Дополнительные источники:**

1. Программирование и основы алгоритмизации: Для инженерных специальностей технических университетов и вузов. /А.Г. Аузяк, Ю.А. Богомолов, А.И. Маликов, Б.А. Старостин. Казань: Изд-во Казанского национального исследовательского технического ун-та - КАИ, 2013, 153 с.

2. Канцедал С.А. Алгоритмизация и программирование : Учебное пособие / С.А. Канцедал. - М.: ИД ФОРУМ: НИЦ Инфра-М, 2013. - 352 с.: ил.

3. Колдаев В.Д. Основы алгоритмизации и программирования: Учебное пособие / В.Д. Колдаев; Под ред. Л.Г. Гагариной. - М.: ИД ФОРУМ: ИНФРА-М, 2012. - 416 с.: ил.;

4. В.Б. Попов В.Б. Turbo Pascal для школьников: Учебно-методическое пособие / В.Б. Попов. - 2-e изд. - М.: ИЦ РИОР: ИНФРА-М, 2012. - 373 с.

**Ресурсы INTERNET**

1. [Электронный ресурс]/(http://www.ict.edu.ru/catalog/index.php).
2. [Электронный ресурс]/(http://artishev.com/texnologii/setevaya-os.html).
3. [Электронный ресурс]/(http://inoblogger.ru/2010/03/31/operacionnaya-sistemanterneta/).
4. [Электронный ресурс]/(http://www.tver.mesi.ru/e-lib/res/648/14/1.html).
5. [Электронный ресурс]/ <http://learnpascal.ru>.
6. [Электронный ресурс]/ [www.pas1.ru](http://www.pas1.ru)

**Инна Дмитриевна Камскова**

**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ для самостоятельной работы по дисциплине «ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ»**

Учебно-методическое пособие

Федеральное государственное автономное образовательное
 учреждение высшего образования

Нижегородский государственный университет имени Н.И. Лобачевского

603950, Нижний Новгород, пр. Гагарина, 23.