Министерство образования Красноярского края

Краевое государственное бюджетное профессиональное образовательное

учреждение «Красноярский колледж радиоэлектроники и информационных технологий»



**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ**

По дисциплине Основы алгоритмизации и программирования

Курс \_\_\_\_

Для специальности

09.02.11 "Разработка и управление программным обеспечением”

Красноярск, 2025

Методические рекомендации составлены:

Преподавателем КГБПОУ СПО «ККРИТ» К.Н. Татарникова

Преподавателем высшей категории КГБПОУ СПО «ККРИТ» Татарников А.В.

РАССМОТРЕНО

на заседании цикловой комиссии преподавателей

профессионального цикла информационно-технического профиля

Протокол №\_\_\_ от «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2025 г.

Председатель ЦК \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В. Харитонова

Ответственный редактор: зам. директора по учебной работе М.А. Полютова

Одобрено Методическим советом КГБПОУ СПО «ККРИТ»

протокол № \_\_ от «\_\_\_» \_\_\_\_\_\_ 2025 г.

Председатель методического совета

Зам. директора по УР М.А. Полютова

***Практическое занятие №1***

**Наименование занятия: *Формы записей алгоритмов***

**Цель занятия:** Научиться составлять алгоритмы линейной и разветвляющейся структур и записывать их в различных формах. Формировать ОК-1, ОК-2, ОК-3, ОК-5, ОК-6, ОК-7,ОК-8 **Подготовка к занятию:** Повторить теоретический материал по теме «Основные понятия ал- горитмизации».

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

### Задание на занятие:

Составить алгоритмы решения задач в соответствии со своим вариантом. Описать ал- горитмы в словесной форме, с помощью псевдокода и блок-схем.

## ВАРИАНТ 1

1. Вычислить значение функции

*f* (*z*)  1 





1   sin *z*



*z* 2 

для *z* =3

1. Ввести два числа *a* и *b* . Если *a* < *b* , оба числа возвести в квадрат, если *a*  *b* - оста-

вить их без изменения. Если *a*  *b* , возвести эти числа в куб.

## ВАРИАНТ 2

1. Вычислить значение функции *f*(*x)*= sin(2*x*2 ) 

*x*

cos 0,1*x*

для *x* =1

1. Ввести два числа *а*1 и *а*2 если ни одно из них не равно нулю, то найти их произведе- ние, в противном случае вывести их сумму.

## ВАРИАНТ 3

1. Найти длину гипотенузы и площадь прямоугольного треугольника, зная длины двух ка- тетов.

5*x*  *x*

2

1. Вычислить значение функции

*у*  2

*x* 2

при условии, что

*x*  0 .

*x* 

3

## ВАРИАНТ 4

* 1. Найти объем куба и площадь его сторон по произвольному ребру куба *a* .
  2. Вычислить значение функции *u*  , при условии, что *b*  0 и

*a x*  sin *x*

*b*

*x*  0 .

## ВАРИАНТ 5

 *x* 2  *x*

1. Вычислить значение функции

*f* (*x*)  sin

 3

 

 cos1,1

для *x* =1

1. Ввести три числа, возвести в квадрат те из них, значения которых отрицательны.

## ВАРИАНТ 6

1. Вычислить и напечатать значение функции

*F* (*x*)  cos *x* 

sin *x*

для *x* = -1

1. Ввести три числа и удвоить значение тех из них, которые положительные.

*x*

## ВАРИАНТ 7

1. Вычислить значение функции *f*(*х*) = sin(1  *x*2 ) 

*x*  2 cos(6  *x*)

для *x* =1

1. Ввести два числа и определить из них минимальное и максимальное.

## ВАРИАНТ 8

* 1. Вычислить длины окружностей радиусов *R*1 и *R*2 с одним центром, а также пло-

щадь кольца, образованного этими окружностями.

* 1. Вычислить значение функции вывести на экран.

*f* (*x*) 

*x*  *a* sin *x a*  *x*

при условии, что

*a*  *x* . Результат

## ВАРИАНТ 9

1. Заданы координаты трех вершин треугольника. Найти его периметр и площадь.

*ax* 2 ,





1. Вычислить значение функции F=

*для*

*x*  0,

## ВАРИАНТ 10

(*ax*  *b*)2 *в*

*противном*

*случае*

1. Даны два числа. Найти среднее арифметическое кубов этих чисел и среднее геомет- рическое их модулей.
2. Вычислить значение функции

*x*  *y*,



*Z*  

*для*

*x*  0,

*x*  *y*  1 *в противном случае*

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Выполнить задание;
3. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание;
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Дайте определение понятию алгоритм
2. Перечислите формы записи алгоритмов
3. Перечислите базовые структуры алгоритмов, охарактеризуйте их.

# ПРИЛОЖЕНИЕ

***Алгоритмом*** называется точное описание, определяющее последовательность дей- ствий исполнителя, направленных на решение поставленной задачи.

#### Формы записи алгоритмов

На практике наиболее распространены следующие формы представления алгоритмов: словесная, графическая (блок-схемы), псевдокод и программа.

***Словесный способ*** записи алгоритмов представляет собой описание алгоритма на естественном языке.

Пример. Записать алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел.

Алгоритм может быть следующим:

* 1. задать два числа;
  2. если числа равны, то взять любое из них в качестве ответа и остановиться, в про- тивном случае продолжить выполнение алгоритма;
  3. определить большее из чисел;
  4. заменить большее из чисел разностью большего и меньшего из чисел;
  5. повторить алгоритм с шага 2.

Описанный алгоритм применим к любым натуральным числам и должен приводить к решению поставленной задачи.

Словесный способ не имеет широкого распространения по следующим причинам:

* + - такие описания строго не формализуемы;
    - страдают многословностью записей;
    - допускают неоднозначность толкования отдельных предписаний.

***Графический способ*** (или блок-схема) - описание структуры алгоритма с помощью последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометриче- ская фигура, представленная в виде блочного символа. Блочные символы соединяются лини- ями переходов, определяющими очередность выполнения действий. В таблице приведены наиболее часто употребляемые символы.

|  |  |  |
| --- | --- | --- |
| **Название символа** | **Обозначение и пример заполнения** | **Пояснение** |
| Процесс | 0007 | Вычислительное действие или по- следовательность действий |
| Решение | 0008 | Проверка условий |
| Модификация | 0009 | Начало цикла |
| Предопределенный процесс | 0010 | Вычисления по подпрограмме, стандартной подпрограмме |
| Ввод-вывод | 0011 | Ввод-вывод в общем виде |
| Пуск-останов | 0012 | Начало, конец алгоритма, вход и выход в подпрограмму |
| Документ | 0013 | Вывод результатов на печать |

Блок **"процесс"** применяется для обозначения действия или последовательности дей- ствий, изменяющих значение, форму представления или размещения данных. Для улучше- ния наглядности схемы несколько отдельных блоков обработки можно объединять в один блок. Представление отдельных операций достаточно свободно.

Блок **"решение"** используется для обозначения переходов управления по условию. В каждом блоке "решение" должны быть указаны вопрос, условие или сравнение, которые он определяет.

Блок **"модификация"** используется для организации циклических конструкций. (Слово модификация означает видоизменение, преобразование). Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра для каждого повторения.

Блок **"предопределенный процесс"** используется для указания обращений к вспомо- гательным алгоритмам, существующим автономно в виде некоторых самостоятельных моду- лей, и для обращений к библиотечным подпрограммам.

***Псевдокод*** - описание структуры алгоритма на естественном, частично формализо- ванном языке, позволяющее выявить основные этапы решения задачи, перед точной его за- писью на языке программирования. В псевдокоде используются некоторые формальные кон- струкции и общепринятая математическая символика.

Псевдокод занимает промежуточное место между естественным и формальным язы-

ками.

* Он близок к обычному естественному языку, поэтому алгоритмы могут на нем за-

писываться и читаться как обычный текст.

* В псевдокоде используются некоторые формальные конструкции и математическая символика, что приближает запись алгоритма к общепринятой математической записи.

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

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

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

***Программа*** - описание структуры алгоритма на языке алгоритмического программи- рования.

### Основные алгоритмические конструкции

Элементарные шаги алгоритма можно объединить в следующие алгоритмические конструкции: линейные (последовательные), разветвляющиеся, циклические.

Линейной называют алгоритмическую конструкцию, реализованную в виде последо- вательности действий (шагов), в которой каждое действие (шаг) алгоритма выполняется ров- но один раз.

Разветвляющейся называется алгоритмическая конструкция, обеспечивающая выбор между двумя альтернативами в зависимости от значения входных данных.

### Линейный алгоритм Разветвляющийся алгоритм

|  |  |
| --- | --- |
| Действие 1 | |
|  |  |
| Действие 2 | |
|  |  |

Да

Условие

Нет

Действие 2

Действие 1

Циклической называют алгоритмическую конструкцию, в которой некоторая, идущая подряд группа действий (шагов) алгоритма может выполняться несколько раз, в зависимости от входных данных или условия задачи.

Параметры цикла

условие

нет

да

Тело цикла

Параметры цикла

нет

условие

Тело цикла

### Примеры составления блок-схемы алгоритма

*x* sin *x*

*a*  *b*

3

2*ab*

*Пример 1.* Составить схему алгоритма вычисления значения: *Z* 

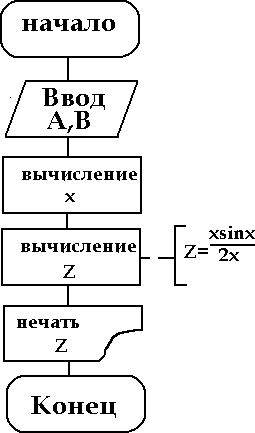
, *гдеx* 

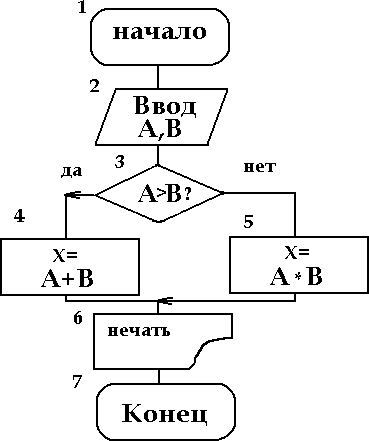
2*x*

Для построения блок – схемы алгоритма опишем последовательность действий, необ- ходимых для решения данной задачи:

* + начало
  + ввод чисел a,b
  + вычисление х
  + вычисление z
  + вывод результата
  + конец

Исходя из этого, составляем блок-схему алгоритма согласно ГОСТ, используя со- ответствующие блоки.



*Пример 2.* Составить схему алгоритма вычисления значения: x=a+b при a>b, x=a\*b, при a<=b.

***Практическое занятие №2***

**Наименование занятия: *Составление простейших программ на языке Pascal***

**Цель занятия:** Изучить порядок действий при вычислении выражений, научиться состав- лять линейные программы. Формировать ОК-1, ОК-4, ОК-5, ОК-6, ОК-7, ОК-8, ОК-9, овла- деть знаниями и умениями, необходимыми для освоения ПК 1.2.

**Подготовка к занятию:** Повторить теоретический материал по теме «Элементы языка Pas- cal. Операторы языка»

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:**

## ВАРИАНТ 1

1. Составьте программу для расчета по двум формулам. Предварительно подготовьте те- стовые примеры с помощью калькулятора. Отсутствующие в языке функции выразите через имеющиеся.

*Z*  sin 4  cos 2 ; *Z* 

*x*  3

*x*  3

1 1  cos 4 1  cos 2 2

1. Дано действительное число *а.* Не пользуясь ни какими другими арифметическими операциями, кроме умножения получить *а*4 за 2 операции.

## ВАРИАНТ 2

1. Составьте программу для расчета по двум формулам. Предварительно подготовьте те- стовые примеры с помощью калькулятора. Отсутствующие в языке функции выразите через имеющиеся.

sin   cos(2   )

*x* 2  2*x*  3  (*x*  1) *x* 2  9

*x* 2  2*x*  3  (*x* 1) *x* 2  9

*Z*1  cos  sin(2   ) ; *Z* 2 

1. Дано действительное число *а.* Не пользуясь ни какими другими арифметическими операциями, кроме умножения получить *а*6 за 3 операции.

## ВАРИАНТ 3

1. Составьте программу для расчета по двум формулам. Предварительно подготовьте те- стовые примеры с помощью калькулятора. Отсутствующие в языке функции выразите через имеющиеся.

*Z*1  2sin (3  2) cos2 (5  2); *Z* 2 

2*b*  2 *b*2  4

*b*2  4  *b*  2

2

1. Дано действительное число *а.* Не пользуясь ни какими другими арифметическими операциями, кроме умножения получить *а*7 за 4 операции.

## ВАРИАНТ 4

1. Составьте программу для расчета по двум формулам. Предварительно подготовьте те- стовые примеры с помощью калькулятора. Отсутствующие в языке функции выразите через имеющиеся.

1

*Z*  sin 2  sin 5  sin 3 ;

1 cos  1  2sin 2 2

*Z* 2 

1. Дано действительное число *а.* Не пользуясь ни какими другими арифметическими операциями, кроме умножения получить *а*8 за 3 операции.

*b*  2

## ВАРИАНТ 5

1. Составьте программу для расчета по двум формулам. Предварительно подготовьте те- стовые примеры с помощью калькулятора. Отсутствующие в языке функции выразите через имеющиеся.

*Z*  sin 2  sin 5  sin 3 ;

1 cos  cos 3  cos 5

*Z*  *m* 1 *m*  *n* 1 *n*

* + *nm*  *m*2  *m*

*m*3*n*

2

1. Дано действительное число *а.* Не пользуясь ни какими другими арифметическими операциями, кроме умножения получить *а*9 за 4 операции.

## ВАРИАНТ 6

1. Составьте программу для расчета по двум формулам. Предварительно подготовьте те- стовые примеры с помощью калькулятора. Отсутствующие в языке функции выразите через имеющиеся.

*Z*  cos 2  3      cos 2 11   ; *Z*  *m*  *n*

1     

8

4

8

4



  

  2 *m*

1. Дано действительное число *а.* Не пользуясь ни какими другими арифметическими операциями, кроме умножения получить *а*10 за 4 операции.

## ВАРИАНТ 7

1. Составьте программу для расчета по двум формулам. Предварительно подготовьте те- стовые примеры с помощью калькулятора. Отсутствующие в языке функции выразите через имеющиеся.

*Z*  cos  sin 

1 cos  sin 

*Z* 2 

(3*m*  2)2  24*m*

2

3 

*m*

*m*

1. Дано действительное число *а.* Не пользуясь ни какими другими арифметическими операциями, кроме умножения получить *а*13 за 5 операций.

## ВАРИАНТ 8

1. Составьте программу для расчета по двум формулам. Предварительно подготовьте те- стовые примеры с помощью калькулятора. Отсутствующие в языке функции выразите через имеющиеся.

*Z*1 

1  1

4 4

sin 5



 2

  8 ;





 *a*  2

*Z* 2  

2*a*



 *a*

2 

*a*  2

*a*  2

1. Дано действительное число *а.* Не пользуясь ни какими другими арифметическими операциями, кроме умножения получить *а*15 за 5 операций.

2*a*  2

  

*a*  2*a* 

## ВАРИАНТ 9

1. Составьте программу для расчета по двум формулам. Предварительно подготовьте те- стовые примеры с помощью калькулятора. Отсутствующие в языке функции выразите через имеющиеся.



*Z*  cos  cos  2  sin  sin  2 ;

1  *a*  *a* 2

*Z* 



 2 

1  *a*  *a* 2 1

5  2*a* 2 

1 2 



2*a*  *a* 2

2*a*  *a* 2 

1. Дано действительное число *а.* Не пользуясь ни какими другими арифметическими операциями, кроме умножения получить *а*21 за 6 операций.



## ВАРИАНТ 10

1. Составьте программу для расчета по двум формулам. Предварительно подготовьте те- стовые примеры с помощью калькулятора. Отсутствующие в языке функции выразите через имеющиеся.

*Z*  1  *tg* ;

1 1  *tg*

*Z* 2 

1. Дано действительное число *а.* Не пользуясь ни какими другими арифметическими операциями, кроме умножения получить *а*28 за 6 операций.

*a*  2

1

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Нарисовать блок-схему алгоритмов для своего варианта;
3. Составить программы на языке Pascal;
4. Ввести программы в ЭВМ и откомпилировать их;
5. Выполнить программы и занести результаты в отчет;
6. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (блок-схема алгоритмов, код программ, результаты выполнения программ);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Записать оператор присваивания.
2. Записать операторы ввода-вывода.
3. В чем отличие между операторами Read и Readln, Write и Writeln?
4. Как откомпилировать программу и запустить её на выполнение?

# ПРИЛОЖЕНИЕ

### Основные элементы языка Pascal

***Алфавит*** языка состоит из множества символов, включающих в себя буквы, цифры и специальные символы.

*Латинские буквы:* от А до Z (прописные) и от а до z (строчные), символ “подчеркива- ния”, который в языке считается буквой.

*Цифры*: арабские от 0 до 9 и шестнадцатеричные (первые 10 цифр от 0 до 9 - араб- ские, остальные шесть - латинские буквы: А, B, C, D, E, F).

*Специальные символы:* + - \* / = , . : ; < > [ ] ( ) { } ', $, пары <> <= >= := (\* \*) (. .), пробел (символы (. .) соответствуют символам [ ], несколько пробелов считаются одним).

К спецсимволам относятся также служебные слова - abs, and, array, begin, case, const, dir, do, downto, else, end, for, function, goto, if, int, label, mod, not, of, or, procedure, program, repeat, shr, then, to, type, var, while, with и др. Смысл зарезервированных слов фиксирован строго. При этом набор зарезервированных слов может меняться от версии к версии.

***Идентификатор*** - это последовательность букв, цифр и знаков подчеркивания, начинающихся не с цифры. Под идентификатором мы будем понимать ячейку памяти ЭВМ, которая имеет свое имя и в которой хранится информация. В Паскале строчные и прописные буквы в идентификаторах и служебных словах не различаются. Идентификаторы могут иметь произвольную длину, но значащими являются только первые 63 символа. Хорошим стилем является осмысленный выбор имени идентификатора. Зарезервированные слова не могут использоваться в качестве идентификаторов.

***Комментарии*** заключаются либо в фигурные скобки **{** комментарий 1 **}**, либо в сим- волы **(\*** комментарий 2 **\*)** и могут занимать любое количество строк. Последовательность из трех символов **(\*)** начинает комментарий до конца строки. Текст комментария игнорируется при компиляции, если это не директивы компилятора, которые имеют вид **{$ }**.

***Константа*** - это величина, которая в ходе выполнения программы принимает одно значение. Ее значение устанавливается еще до того, как программа начнет выполняться, а в ходе ее запуска сохраняет свое значение неизменной на всем протяжении работы програм- мы.

В качестве констант могут использоваться целые, вещественные, шестнадцатеричные числа, логические константы, символы, строки символов, множества:

а) целые числа записываются без дробной части со знаком или без него;

б) вещественные числа записываются со знаком или без него, с фиксированной или плавающей точкой (например, +3.14 или -19е-5);

в) логическая константа - либо false либо true (ложь или истина);

г) символьная константа - любой символ, заключенный в апострофы (например, '<>',

'Y');

д) строковая константа - любая последовательность символов, заключенная в апо-

строфы (например, 'это моя строка', ' "').

В Паскале существуют типизированные константы, представляющие собой перемен- ные простых или составных типов (кроме файлов) с начальным значением.

***Переменными*** называются параметры программы, которые могут менять свое значе- ние в процессе ее выполнения. Все без исключения переменные должны быть описаны в раз- деле программы, начинающемся со слова **VAR**. Затем следуют конструкции вида:

список идентификаторов переменных**:** тип1; список идентификаторов переменных**:** тип2;

В списке имена переменных перечисляются через запятую. Кроме базовых типов Турбо Паскаля здесь можно использовать свои типы (описанные ранее в разделе **Type**).

### Арифметические операции, функции, выражения

В Паскале определены следующие операции:

* Унарные: not (отрицание);
* Мультипликативные: \* (умножение), / (деление), div (деление нацело), mod (остаток от целочисленного деления), and (логическое “И”);
* Аддитивные: + (сложение), - (вычитание), or (логическое “ИЛИ”);
* Отношения: = (равно), <> (не равно), < (меньше), > (больше), <= (меньше или равно),

>= (больше или равно).

*Примечание*: в Паскале нет операции возведения в степень. Вместо этого для возведения в целую степень можно использовать операцию умножения, а в дробную – выражение, ис- пользующее операцию логарифмирования и потенцирования: xy=ey\*lnx, что на языке Пас- каль соответствует выражению exp(y\*ln(x)) с использованием двух стандартных функций exp(<выражение>) и ln(<выражение>).

#### Стандартные функции Паскаля

Для вычисления наиболее распространенных математических функций в Турбо- Паскале предусмотрены следующие стандартные функции:

|  |  |
| --- | --- |
| **Обращение** | **Функция** |
| Pi | Число π |
|  | Синус х (х в радианах) |
| Cos(x) | Косинус х (х в радианах) |
| Arctan(x) | Арктангенс х (х в радианах) |
| Exp(x) | еx – экспонента |
| Ln(x) | Натуральный логарифм x |
| Sqr(x) | Квадрат числа х |
| Sqrt(x) | Квадратный корень из числа х |
| Abs(x) | Абсолютная величина числа х |
| Trunc(x) | Ближайшее целое, не превышающее х по модулю |

|  |  |
| --- | --- |
| Frac(x) | Дробная часть числа х |
| Int(x) | Целая часть числа х |
| Round(x) | Округление числа до ближайшего целого |
| Succ(x) | Определение следующего по порядку элемента из списка |
| Pred(x) | Определение предыдущего элемента из списка |
| Random | Псевдослучайное число в интервале [0; 1) |
| Random(x) | Псевдослучайное число в интервале [0; х) |

.

***Выражение*** - это единица языка, которая определяет способ вычисления некоторого

значения. Выражения формируются из констант, переменных, функций, знаков операций и круглых скобок по определенным синтаксическим правилам.

В Паскале имеется большое количество встроенных функций для работы с данными каждого типа. Имена (указатели) этих функций с аргументом в круглых скобках могут также встречаться в выражениях.

Круглые скобки используются для изменения порядка вычисления частей выражения. Выражения без скобок вычисляются в порядке, соответствующем приоритету операций. Приоритеты расставлены таким образом:

* вычисления в круглых скобках;
* вычисление значений функций;
* унарные операции ( **not**,+,- );
* операции типа умножения ( \*,/,**div**,**mod**,**and** );
* операции типа сложения ( +,-, **or**, **xor** );
* операции отношения ( =, <>, <, >, <=, >= ).

В логическом выражении 2<=4 **and** 5>3 Паскаль выдаст ошибку, поскольку операция

**and** будет выполнена раньше операций сравнения. Верная запись - (2<=4) **and** (5>3).

### Структура программы

Программа, написанная на Паскале, состоит из заголовка и тела (блока), в конце кото- рого следует точка – признак конца программы. В свою очередь, блок содержит разделы описаний и раздел операторов:

**Program** <имя программы>; { Заголовок программы }

**Uses** … ; { Подключение модулей } **Label** ... ; { Раздел объявления меток } **Const** ... ; { Раздел объявления констант }

**Type** ... ; { Раздел объявления новых типов } **Var** ... ; { Раздел объявления переменных } **Procedure** ... ; { Описание своих процедур } **Function** ... ; { Описание своих функций } **Begin** { начало основной программы }

<раздел операторов>

### End.

**Операторы языка**

Оператор – это описание действий, которые будут выполнены при реализации алго- ритма. В Паскале существуют 2 вида операторов: простые и структурные.

Простые операторы: оператор присваивания, оператор перехода, процедуры и функции, пустой оператор. Структурные операторы: составной оператор, оператор условия, оператор варианта, оператор цикла.

### Пустой оператор

В программе может применяться пустой оператор, не выполняющий никакого дей- ствия, представляющий собой точку с запятой. Он может потребоваться для осуществления на него безусловного перехода или для более наглядного просмотра программы.

Goto M1;

. . . . M1:

. . . .

### Оператор присваивания

Оператор присваивания используется для задания значения переменных и имеет сле- дующий 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 содер- жится во множестве допустимых значений для типа переменной.

### Составной оператор

Составным оператором считается последовательность произвольных операторов, за- ключенная в операторные скобки - зарезервированные слова **begin** ... **end**. Допускается про- извольная глубина вложенности составных операторов. Составной оператор применяется там, где по синтаксическим правилам языка может стоять только один оператор, а нам надо выполнить несколько действий. В этом случае набор необходимых команд должен быть оформлен как составной оператор. По сути, все тело программы представляет собой один составной оператор.

### Простейший ввод и вывод

Рассмотрим простейшие процедуры ввода и вывода. По умолчанию ввод осуществля- ется с клавиатуры, а вывод на экран. К операторам ввода относятся:

**Read**(<список переменных через запятую>);

**Readln**(<список переменных>);

### Readln;

Второй отличается от первого тем, что после ввода переводит курсор на новую стро- ку, точнее, в конце своей работы считывает с клавиатуры код клавиши <**Enter**>. Третий опе- ратор используется для организации паузы - выполнение программы продолжится, как пра- вило, только после нажатия на клавиатуре клавиши <**Enter**>.

К операторам вывода относятся:

**Write**(<список вывода>); **Writeln**(<список вывода>); **Writeln**;

В списке вывода кроме имен переменных можно писать строковые константы (после- довательность символов в апострофах) и даже выражения (выводятся их значения). Второй оператор отличается от первого тем, что после вывода переводит курсор на новую строку. Третий оператор просто переводит курсор на новую строку.

Форма записи параметра в процедуре Write имеет следующий вид: Write(параметр, параметр:B1, параметр:B1:B2)

где В1 - ширина поля, B2 - длина дробной части. По умолчанию integer занимает 7 позиций, real - 13 позиций. В целой части числа записывается первая отличная от нуля цифра. Шесть последующих цифр составляют дробную часть мантиссы. Одну позицию занимает буква е, которая обозначает основание степени, которое соответствует числу 10, одна позиция отво- дится под знак порядка и две - для величины порядка.

Например, если даны три числа: А = 3.6, В = 7.4, С = -2.5 и напечатан оператор Write('A=',A:4:1,' B=',B,' C=',C:6:1), то в результате будет выведено: А= 3.6 В= 7.400000Е+00 С= -2.5

Рассмотрим в качестве примера программу определения координат материальной точки (х; y) для заданной секунды ее полета (t) по параболической траектории, вычисляемых по формулам:

х = v·t·cos(O), y = v·t·sin(O) - g·t2 / 2,

где v - начальная скорость точки, О - угол ее бросания, g - ускорение свободного падения, равное 9,81 м/с2.

program traekt;

Const g = 9.81;

Var x, y, v, O : real; t : integer;

Begin

read(v, O, t);

x := v \* t \* cos(O);

y := v \* t \* sin(O) - g \* t \* t /2; writeln('Координата х=',x,' Координата y=',y)

End.

***Практическое занятие №3***

**Наименование занятия: *Составление программ с применением условных операторов* Цель занятия:** Научиться составлять программы с применением условного оператора, опе- ратора выбора. Формировать ОК-2, ОК-3, ОК-4, ОК-5, ОК-6, ОК-8, овладеть знаниями и умениями, необходимыми для освоения ПК-1.3, ПК-2.2, ПК-2.3

**Подготовка к занятию:** Повторить теоретический материал «Операторы языка Pascal».

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:**

## ВАРИАНТ 1

1. Даны три действительных числа. Возвести в квадрат те из них, значения которых не- отрицательны и в четвертую степень – отрицательные.
2. По введенному натуральному числу *п (п*  *100),* обозначающему количество ворон, выве- сти надпись «На дереве *п* ворон», где ворона склоняется соответственно *п:* «ворон»,

«ворона», «вороны». В случае, если *п* > 100, вывести надпись «Ветка обломилась».

## ВАРИАНТ 2

1. Даны действительные числа *a*, *b*, *c*. Проверить выполняются ли неравенства *a* < *b* < *c*.
2. Написать программу, которая по набранным баллам (1-100) в результате некоторого те- стирования, сообщает полученную оценку:

0 ≤ б < 20 – плохо

20 ≤ б < 50 – неудовлетворительно 50 ≤ б < 75 – удовлетворительно 75 ≤ б < 93 – хорошо

93 ≤ б < 100 – отлично

## ВАРИАНТ 3

1. Даны действительные числа *x*, *y*, *z*. Найти *max*(*x*+*y*+*z*, *xyz*);
2. Для каждой введенной цифры (0 – 9) вывести соответствующее ей название на ан- глийском языке (0 – zero, 1 – one, …)

## ВАРИАНТ 4

1. Дано натуральное число *а*. Выяснить, является ли данное число четным.
2. Написать программу, которая по введенному номеру времени года (1 – зима, 2 – вес- на, 3 – лето, 4 – осень) выдает соответствующие этому времени года месяцы и количе- ство дней в каждом из месяцев.

## ВАРИАНТ 5

1. Даны три действительных числа. Выбрать из них те, которые принадлежать интерва- лу (1,0; 3,5).
2. Даны два действительных положительных числа *х* и *у*. Арифметические действия над числами пронумерованы следующим образом: 1 – сложение, 2 - вычитание, 3 – умножение, 4 – деление. Составить программу, которая по введенному номеру вы- полняет то или иное действия над числами.

## ВАРИАНТ 6

1. Даны действительные числа *x*, *y*. Меньшее из этих чисел заменить их полусуммой, а большее – удвоенным произведением.
2. Написать программу, которая по введенному числу определяет последнюю цифру его квадрата.

## ВАРИАНТ 7

1. Посчитать количество отрицательных чисел среди чисел *a*, *b*, *c*.
2. По введенному номеру месяца определить количество дней в месяце (год считать не ви- сокосным).

## ВАРИАНТ 8

1. Даны целые числа *a*, *b*, *c*. Определить, делителем каких из них является число *k.*
2. По введенному дню недели определить, рабочий или выходной день.

## ВАРИАНТ 9

1. Перераспределить значения переменных *х* и *у* так, чтобы в *х* оказалось большее из этих значений, а в *у* – меньшее.
2. Для натурального числа *k* напечатать фразу «Мы нашли *k* грибов в лесу», согласовав окончание слова «гриб» с числом *k.*

## ВАРИАНТ 10

1. Найти максимальное из трех заданных чисел.
2. По введенному номеру месяца определить название месяца.

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Нарисовать блок-схемы алгоритмов для своего варианта;
3. Составить программы на языке Pascal;
4. Ввести программы в ЭВМ и откомпилировать их;
5. Выполнить программы и занести результаты в отчет;
6. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (блок-схемы, код программ, результаты выполнения про- грамм);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Что называется составным оператором?
2. Записать полный и краткий формат условного оператора
3. По какому алгоритму работает условный оператор?
4. Оператор выбора, формат оператора выбора
5. Каким образом работает оператор выбора?
6. В каких случаях применяется оператор выбора?

# ПРИЛОЖЕНИЕ

### Разветвляющиеся алгоритмы

В Паскале имеется возможность нелинейного хода программы, т.е. выполнения опе- раторов не в том порядке, в котором они записаны. Такую возможность нам предоставляют разветвляющиеся алгоритмы. Они могут быть реализованы одним из трех способов: с ис- пользованием операторов перехода, условного оператора или оператора выбора.

### Условный оператор

Условный оператор **IF** позволяет изменить порядок выполнения команд в зависимо- сти от некоторого логического условия, т.е. он осуществляет ветвление вычислительного процесса. Условный оператор имеет вид:

**IF** <условие> **THEN** <оператор1> [**ELSE** <оператор2>];

В случае истинности логического выражения, стоящего в условии, выполняется <опе- ратор1>, а <оператор2> пропускается. При ложном значении логического выражения про- пускается <оператор1> и выполняется <оператор2>.

Оператор **IF** может быть полным (присутствуют обе ветви) или неполным (**Else**-ветви нет, при ложном условии ничего не делается). По правилам каждая из ветвей может содер- жать либо один выполняемый оператор, либо несколько, объединенных в составной. Точка с запятой перед **Else** считается ошибкой.

***Пример****. Ввести целое число. Вывести соответствующий ему символ ASCII- таблицы, либо сообщить, что такого символа нет (0-31 - управляющие коды, затем до 256*

*- печатаемые символы).* **program** ascii\_symbol; **var** i:word;

### begin

**write**('Введите целое число: '); **readln**(i); **if** (i>31) **and** (i<256) **then**

**writeln**('Соответствующий символ - ', Chr(i))

**else writeln**('Такого символа нет');

### readln end.

**Оператор выбора**

Оператор выбора является обобщением условного оператора: он дает возможность выполнить один из нескольких операторов в зависимости от значения некоторого выраже- ния, называемого селектором.

**CASE <селектор> OF**

<**список меток 1**> **:** <**оператор 1**>;

<**список меток 2**> **:** <**оператор 2**>;

. . . . . . . ;

<**список меток N**> **:** <**оператор N**>; **ELSE <оператор>**

## END;

*селектор* - выражение любого перечисляемого типа, кроме вещественного;

*оператор* - любой оператор языка, в том числе и составной;

*список меток* - список разделенных запятыми значений выражения 'селектор' или одно его значение; тип метки и селектора должен быть одинаков;

Оператор варианта выбирает для исполнения тот ОПЕРАТОР, одна из меток кото- рого равна текущему значению выражения СЕЛЕКТОР. Если ни одна из меток не равна те- кущему значению селектора, то никакие операторы не выполняются, либо выполняются операторы, следующие за зарезервированным словом ELSE (если такое имеется).

Ветвь **Else** не обязательна, и в отличие от оператора **if**, перед ней ставится точка с за- пятой. Если для нескольких значений <селектора> действия совпадают, то эти константы можно перечислить через запятую перед двоеточием или 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**.

***Практическое занятие №4***

**Наименование занятия: *Операторы цикла***

**Цель занятия:** Научиться составлять программы с применением операторов цикла на языке Pascal. Формировать ОК-2, ОК-3, ОК-4, ОК-5, ОК-6, ОК-8, овладеть знаниями и умениями, необходимыми для освоения ПК-1.3, ПК-2.2, ПК-2.3

**Подготовка к занятию:** Повторить теоретический материал по теме «Операторы цикла»

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:**

## ВАРИАНТ 1

* 1. Используя цикл с предусловием, составить программу табулирования функции

*f* (*x*)  *x*  sin *x* на отрезке [*a*, *b*] с шагом *h*. Результат представить в виде таблицы, первый

столбец которой – значения аргумента, второй – соответствующие значения функции.

* 1. Вводится последовательность ненулевых чисел, 0-конец последовательности. Подсчитать, сколько в ней отрицательных чисел.

## ВАРИАНТ 2

1. Используя цикл с постусловием, составить программу табулирования функции

*f* (*x*)  *tgx* на отрезке [*a*, *b*] с шагом *h*. Результат представить в виде таблицы, первый стол-

бец которой – значения аргумента, второй – соответствующие значения функции.

1. На выставке собак, где были представлены разные породы, отбор животных про- изводился по возрасту и высоте холки. Определить, сколько было боксеров 2-3-летнего возраста с высотой холки не менее 55 сантиметров.

## ВАРИАНТ 3

1. Используя цикл с предусловием, составить программу табулирования функции *f* (*x*)  2cos *x* 1 на отрезке [*a*, *b*] с шагом *h*. Результат представить в виде таблицы, первый столбец которой – значения аргумента, второй – соответствующие значения функции.
2. В очереди за билетами стоят мужчины и женщины. Какое количество мужчин стоит в начале очереди до первой женщины.

## ВАРИАНТ 4

1. Используя цикл с постусловием, составить программу табулирования функции *f* (*x*)  sin *x*  cos *x* на отрезке [*a*, *b*] с шагом *h*. Результат представить в виде таблицы, первый столбец которой – значения аргумента, второй – соответствующие значения функции.
2. Дано натуральное число *п.* Найти количество цифр данного числа, больших *а (а*

вводится с клавиатуры).

## ВАРИАНТ 5

1. Используя цикл с предусловием, составить программу табулирования функции

*f* (*x*)  sin 2 *x* на отрезке [*a*, *b*] с шагом *h*. Результат представить в виде таблицы, первый

столбец которой – значения аргумента, второй – соответствующие значения функции.

1. Найти сумму цифр заданного натурального числа *п.*

## ВАРИАНТ 6

1. Используя цикл с постусловием, составить программу табулирования функции

*f* (*x*)  *x* sin *x* на отрезке [*a*, *b*] с шагом *h*. Результат представить в виде таблицы, первый

столбец которой – значения аргумента, второй – соответствующие значения функции.

1. Составить программу нахождения минимального из *п* отрицательных чисел.

## ВАРИАНТ 7

1. Используя цикл с предусловием, составить программу табулирования функции

*f* (*x*)  sin 1   2

  

*x*

на отрезке [*a*, *b*] с шагом *h*. Результат представить в виде таблицы, первый

 

столбец которой – значения аргумента, второй – соответствующие значения функции.

1. Ввести *п* чисел. Определить, сколько среди них положительных.

## ВАРИАНТ 8

1. Используя цикл с постусловием, составить программу табулирования функции

*f* (*x*)  cos *x*  *ctgx* на отрезке [*a*, *b*] с шагом *h*. Результат представить в виде таблицы, первый

столбец которой – значения аргумента, второй – соответствующие значения функции.

1. Дано натуральное число. Найти произведение цифр этого числа.

## ВАРИАНТ 9

1. Используя цикл с предусловием, составить программу табулирования функции

*f* (*x*)  cos 2*x* на отрезке [*a*, *b*] с шагом *h*. Результат представить в виде таблицы, первый

столбец которой – значения аргумента, второй – соответствующие значения функции.

1. Дано натуральное число. Найти количество чётных цифр этого числа.

## ВАРИАНТ 10

1. Используя цикл с постусловием, составить программу табулирования функции *f* (*x*)  sin *x*  0,5cos *x* на отрезке [*a*, *b*] с шагом *h*. Результат представить в виде таблицы, пер- вый столбец которой – значения аргумента, второй – соответствующие значения функции.
2. Вводится последовательность ненулевых чисел, 0-конец последовательности. Найти сумму положительных чисел последовательности.

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Нарисовать блок-схему алгоритмов для своего варианта;
3. Составить программы на языке Pascal;
4. Ввести программы в ЭВМ и откомпилировать их;
5. Выполнить программы и занести результаты в отчет;
6. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (блок-схемы, код программ, результаты выполнения про- грамм);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Запишите формат цикла с параметром.
2. Запишите формат цикла с предусловием.
3. Запишите формат цикла с постусловием.
4. Чем отличается оператор цикла с предусловием от оператора цикла с постусловием?

# ПРИЛОЖЕНИЕ

### Циклические алгоритмы

Последовательность команд, выполняющихся несколько раз в зависимости от некоторого условия, называется циклом.

### Цикл с параметром

Если заранее известно число повторений цикла, то в программе используются циклы с параметром. Оператор цикла **For** организует выполнение одного оператора заранее опреде-

ленное число раз. Его еще называют цикл со счетчиком. В общем виде цикл с параметром задается следующим образом:

**FOR** <параметр> := <nz> **TO** <kz> **DO** <оператор>

Здесь параметр цикла (счетчик) представляет собой переменную порядкового типа;

<nz> и <kz> - выражения, определяющие начальное и конечное значение счетчика; <опера- тор> - один (возможно составной) оператор, который называют телом цикла, повторяемый определенное число раз.

На первом шаге цикла параметр принимает значение nz. После каждого выполнения тела цикла, если параметр цикла не равен kz, происходит увеличение параметра на единицу. Если nz > kz, то <тело цикла> не будет выполнено ни разу и выполнение цикла с параметром сразу же закончится.

Возможна другая форма цикла с параметром.

**FOR** <параметр> := < kz > **DOWNTO** < nz > **DO** <оператор>,

которая выполняется аналогичным образом, но значение <параметра> изменяется с шагом, равным -1.

***Рекомендации*:** Использовать цикл **for** при заранее известном количестве повторений. Не изменять параметр в теле цикла. При использовании кратных (вложенных) циклов при- менять разные переменные в качестве параметров. Определять до цикла значения всех ис- пользуемых в нем переменных. Не ставить точку с запятой после **do**.

***Пример 1.*** Программа *DemoFor1* выводит на экран таблицу перевода из градусов по шкале Цельсия(С) в градусы по Фаренгейту(Р) для значений от 15°С до 30°С с шагом 1 гра- дус. Перевод осуществляется по формуле: F = С\*1.8+32.

program DemoFor1; var

I: integer; F: real; begin

Writeln (' Температура ') ;

for I:= 15 to 30 do *{Заголовок цикла с параметром}*

begin *{Начало тела цикла}*

F:= I\*1.8+32;

Writeln('no Цельсию= ',I,' по Фаренгейту= ', F:5:2) end; *{Конец тела цикла}*

end.

В блоке описания переменных описаны параметр цикла *I* типа *integer* и переменная *F*

— температура по Фаренгейту типа *real*. Переменная *I,* помимо функций управляющей пе- ременной, является переменной, хранящей целочисленные значения температуры по шкале Цельсия. В начале выполнения программы на экран выводится надпись *' Температура '*, а затем оператором повтора выводится таблица соотношения температуры в шкалах Цельсия и Фаренгейта. Печать таблицы выполняется оператором *Writeln('По Цельсию= ',I,' по Фарен- гейту= ' , F: 5:2).*

Цикл выполняется следующим образом. При первом обращении к оператору *for* вы- числяются значения начального *(15)* конечного *(30)* параметров цикла, и управляющей пере- менной *I* присваивается начальное значение *15*.

Затем циклически выполняется следующее:

Проверяется условие *I<=30.*

Если оно соблюдается, то выполняется составной оператор в теле цикла, т.е. рассчиты- вается значение выражения *I\* 1.8+32*, затем оно присваивается переменной *F,* и на экран вы- водится сообщение: 'По Цельсию= ', I, ' по Фаренгейту= ', F:5:2.

Если условие *I<=30* не соблюдается, т. е. как только *I* станет *> 30,* оператор тела цик- ла не выполняется, а управление в программе передается за пределы оператора *for*, в нашем примере на оператор *end*. Программа завершает работу.

Значение параметра цикла *I* увеличивается на единицу, и управление передается в за- головок цикла *for* для проверки условия.

Далее цикл повторяется.

***Пример 2****: Вводятся 10 чисел, посчитать среди них количество положительных.*

program cycle\_for1;

var i,kn:byte; x:real; begin

kn:=0;

for i:=1 to 10 do begin

writeln('Введите ',i,' число: '); readln(x);

if x>0 then kn:=kn+1 {увеличиваем количество на 1} end;

writeln('Вы ввели ',kn,' положительных чисел.'); readln

end.

***Пример 3****: Напечатать буквы от 'Z' до 'A'.*

program cycle\_for2; var c:char;

begin

for c:='Z' downto 'A' do write(c); readln

end.

В данной программе применяется *цикл for с убыванием значения управляющей пере- менной* (используется указание *downto* - убывание).

### Циклы с условием

Если заранее неизвестно число повторений цикла, то используются циклы с условием.

В Паскале имеется два типа таких циклов. Это циклы с предусловием и постусловием.

**Цикл с предусловием WHILE** <логическое выражение> **DO** <оператор>;

Выполнение оператора цикла с предусловием начинается с проверки условия, запи- санного после слова while. Если оно соблюдается, то выполняется <тело цикла>, а затем вновь проверяется условие и т.д. Как только на очередном шаге окажется, что условие не со- блюдается, то выполнение <тела цикла> прекратится.

Если <тело цикла> состоит из нескольких операторов, то они объединяются опера- торными скобками.

В теле цикла обязательно должен быть оператор, влияющий на соблюдение условия, в противном случае произойдет зацикливание.

***Пример 4.*** Найти сумму 10 произвольно введенных целых чисел. program DemoWhile;

const

Limit =10; *{Ограничение на количество вводимых чисел}*

var Count, Item, Sum: integer; begin

Count:=0; *{Счетчик чисел}*

Sum:= 0; *{Сумма чисел}*

while (Count < Limit) do *{Условие выполнения цикла}*

begin

Count:= Count+1;

Write('Введите ', Count, ' - e целое число: '); Readln(Item);*{Ввод очередного числа с клавиатуры}*

Sum:= Sum+Item;

end.

end;

Writeln('Сумма введенных чисел равна ', Sum) ;

В данном примере в разделе описания констант описана константа *Limit=10,* задающая ограничение на количество вводимых чисел. В разделе описания переменных описаны пере- менные *Count, Item, Sum* целочисленного типа. В начале выполнения программы обнуляются значения счетчика введенных чисел *Count* и их суммы *Sum*. Затем выполняются цикл ввода 10 чисел и их суммирование. Вначале оператор *условия while* проверяет условие *Count < Limit*. Если условие верно, то выполняется составной оператор в теле цикла:

begin

Count:= Count+1;

Write('Введите ', Count, '-e целое число: '); Readln(Item) ;

Sum:= Sum+Item;

End;

в котором вводится значение очередного числа, и на это значение увеличивается значение суммы. После этого управление в программе вновь передается оператору цикла *while*, опять проверяется условие *Count < Limit*. Если условие верно, то выполняется составной оператор и т. д., пока значение переменной *Count* будет меньше 10. Как только значение *Count* станет равно 10 и условие *Count < Limit* не будет соблюдено, выполнение цикла завершится, а управление в программе будет передано на оператор, находящийся за словом *end*, т. e. пер- вый оператор за границей *while.* Это вызов процедуры *Writeln,* которая выведет сообщение *'Сумма введенных чисел равна'* и напечатает значение переменной *Sum*.

***Пример 5****: Нахождение наибольшего общего делителя двух целых чисел с помощью алгоритма Евклида.*

program Evklid;

var a,b,c:integer; begin

write('введите два целых числа : '); readln(a,b);

while b<>0 do begin

c:=a mod b; a:=b;

b:=c; end;

writeln('наибольший общий делитель = ',a); readln

end.

***Пример 6****: Пары неотрицательных вещественных чисел вводятся с клавиатуры. По- считать произведение для каждой пары и сумму всех чисел.*

program cycle\_while;

var x,y,sum:real; otv:char; begin

sum:=0; otv='Д';

while (otv='Д') or (otv='д') do begin

write('Введите числа x,y > 0 '); readln(x,y);

writeln('Их произведение = ',x\*y:8:3); sum:=sum+x+y;

write('Завершить программу (Д/Н)? '); readln(otv);

end;

writeln('Общая сумма = ',sum:8:3); readln

end**.**

**Цикл с постусловием**

## REPEAT

<оператор 1>

...

<оператор N>

**UNTIL** <логическое выражение>

Оператор **Repeat** организует повторяющееся выполнение нескольких операторов до тех пор пока не станет истинным условие, стоящее в **Until**-части. Тело цикла обязательно выполняется хотя бы один раз. Таким образом, в этом цикле логическое выражение - это условие выхода из цикла.

При создании циклических алгоритмов Турбо Паскаль позволяет использовать проце- дуры **Continue** и **Break**. Процедура **Continue** досрочно завершает очередной шаг цикла, пе- редает управление на заголовок. Процедура **Break** реализует немедленный выход из цикла.

***Рекомендации*:** Для того чтобы избежать зацикливания программы необходимо обес- печить изменение на каждом шаге цикла значения хотя бы одной переменной, входящей в условие цикла. После выхода из цикла со сложным условием (с использованием операций **and**, **or**, **xor**) как правило, необходима проверка того, по какому условию цикл завершен.

***Пример 7****: Пары неотрицательных вещественных чисел вводятся с клавиатуры. По- считать произведение для каждой пары и сумму всех чисел.*

program cycle\_repeat;

var x,y,sum:real; otv:char; begin

sum:=0; repeat

write('Введите числа x,y > 0 '); readln(x,y);

writeln('Их произведение = ',x\*y:8:3); sum:=sum+x+y;

write('Завершить программу (Д/Н)? '); readln(otv);

until (otv='Д') or (otv='д'); writeln('Общая сумма = ',sum:8:3); readln

end.

***Практическое занятие №5***

**Наименование занятия: *Вложенные циклы***

**Цель занятия:** Научиться составлять программы с применением вложенных циклов на язы- ке Pascal. Формировать ОК-2, ОК-3, ОК-4, ОК-5, ОК-6, ОК-8, овладеть знаниями и умения- ми, необходимыми для освоения ПК-1.3, ПК-2.2, ПК-2.3

**Подготовка к занятию:** Повторить теоретический материал по теме «Операторы цикла».

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:**

## ВАРИАНТ 1

Дано натуральное число *n*. Составить программу вычисления значения выражения 11 + 22 +...+ *nn*

## ВАРИАНТ 2

У гусей и кроликов вместе 64 лапы. Сколько может быть кроликов и гусей (указать все сочетания)?

## ВАРИАНТ 3

Дано натуральное число *n*. Можно ли его представить в виде суммы трёх квадратов натуральных чисел? Если да, то указать тройку *x*, *y*, *z* таких натуральных чисел, что *x*2 + *y*2 + *z*2 = *n*.

## ВАРИАНТ 4

Составить программу для графического изображения делимости чисел от 1 до *n* (*n* - исходное данное). В каждой строке надо печатать число и сколько плюсов, сколько делите- лей у этого числа. Например, если исходное данное - число 4, то на экране должно быть напечатано:

1+

2++

3++

4+++

## ВАРИАНТ 5

Дано натуральное число *n*. Составить программу вычисления S = 1! + 2! + 3! + …+ n!

## ВАРИАНТ 6

Даны натуральные числа *a*, *b* (*a* < b). Получить все простые числа *p*, удовлетворяю- щие неравенствам: *a* ≤ *p* ≤ *b*.

## ВАРИАНТ 7

Дано натуральное число *n*. Можно ли его представить в виде суммы трёх квадратов натуральных чисел? Если да, то указать все тройки *x*, *y*, *z* таких натуральных чисел, что *x*2 + *y*2 + *z*2 = *n*.

## ВАРИАНТ 8

Даны действительное число *х* и натуральное число *n*. Составить программу вычисле- ния выражения *S*  sin *x*  sin *x*2  sin *x*3  ...  sin *xn* .

## ВАРИАНТ 9

Составить программу возведения заданного числа в третью степень, используя сле- дующую закономерность:

13 = 1

23 = 3 + 5

33 = 7 + 9 + 11

43 = 13 + 15 + 17 + 19

53 = 21 + 23 + 25 + 27 + 29

## ВАРИАНТ 10

Даны действительное число *а* и натуральное число *n*. Составить программу вычисле-

ния выражения *S*  1  1

*a a* 2

 1  ... 

*a* 4

1

*a* 2*n*2

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Составить программу на языке Pascal;
3. Ввести программу в ЭВМ и откомпилировать её;
4. Выполнить программу и занести результаты в отчет;
5. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (код программы, результаты выполнения программы);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Какие циклы называют вложенными?

# ПРИЛОЖЕНИЕ

### Вложенные циклы

Если для решения задачи требуется организовать два цикла, один из которых помеща- ется внутрь другого, то такие конструкции называют вложенными циклами.

***Пример1.*** Даны натуральные числа n и k. Составить программу вычисления выраже- ния 1k + 2k +...+ nk.

Для вычисления указанной суммы целесообразно организовать цикл с параметром i, в котором, во-первых, вычислялось бы очередное значение y = ik и, во-вторых, осуществлялось бы накопление суммы прибавлением полученного слагаемого к сумме всех предшествующих (s = s + y).

Program Example;

Var n, k, y, i, s, m: Integer; Begin

Writeln ('Введите исходные данные n и k'); Readln(n,k);

s:=0;

For i:=1 To n Do Begin y:=1;

For m:=1 To k Do y:=y\*i; {нахождение степени k числа i} s:=s+y;

End;

Writeln('Ответ: ',s);

End.

***Пример 2.*** Модифицировать предыдущую программу так, чтобы она вычисляла сум- му 11 + 22 +...+ nn.

Данная задача отличается от предыдущей тем, что показатель степени очередного слагаемого совпадает со значением её основания, следовательно, параметры внутреннего цикла (цикла, в котором вычисляется очередное слагаемое) совпадают с параметрами внеш- него цикла.

Program Example\_14;

Var n, y, i, s, m: Integer;

Begin

Writeln('Введите начальное значение n '); Readln(n);

s:=s;

For i:=1 To n Do Begin y:=1;

For m:=1 To i Do y:=y\*i; {нахождение степени k числа i} s:=s+y;

End;

Writeln('Ответ: ',s);

End.

***Пример 3.*** Старинная задача. Сколько можно купить быков, коров и телят, если плата за быка 10 рублей, за корову - 5 рублей, за телёнка - полтинник (0,5 рубля), если на 100 руб- лей надо купить 100 голов скота.

Обозначим через b - количество быков; k - количество коров; t - количество телят. По- сле этого можно записать два уравнения: 10b + 5k + 0.5t = 100 и b + k + t = 100. Преобразуем их в 20b + 10k + t = 200 и b + k + t = 100

На 100 рублей можно купить:

не более 10 быков, т.е. 0<=b<=10 не более 20 коров, т.е. 0<=k<=20 не более 200 телят, т.е. 0<=t<=200.

Таким образом, получаем:

Program Example; Var b, k, t: Integer; Begin

For b:=0 To 10 Do For k:=0 To 20 Do For t:=0 To 200 Do

If (20\*b+10\*k+t=200) And (b+k+t=100) Then Writeln('быков ',b,' коров ',k,' телят ',t);

End.

Сколько раз будет проверяться условие в данной программе? Значение переменной b изменяется 11 раз (от 0 до 10), для каждого её значения переменная k изменяется 21 раз, а для каждого значения переменной k переменная t изменяется 201 раз. Таким образом, усло- вие будет проверяться 11\*21\*201 раз. Но если известно количество быков и коров, то коли- чество телят можно вычислить по формуле t = 100 - (b + k) и цикл по переменной t исключа- ется.

Program Example\_16;

Var b, k, t: Integer;

Begin

For b:=0 To 10 Do

For k:=0 To 20 Do Begin

t:=100-(b+k);

If (20\*b+10\*k+t=200) Then Writeln('быков,'b,' коров ',k,' телят',t); End;

End.

При этом решении условие проверяется 11\*21 раз.

Внутренний и внешний циклы могут быть любыми из трёх рассмотренных ранее ви- дов: циклами с параметром, циклами с предусловием или циклами с постусловием. Правила организации как внешнего, так и внутреннего циклов такие же, как и для простых циклов каждого из этих видов. Но при использовании вложенных циклов необходимо соблюдать следующее условие: внутренний цикл должен полностью укладываться в циклическую часть внешнего цикла.

***Практическое занятие №6***

**Наименование занятия: *Одномерные массивы***

**Цель занятия:** Научиться составлять программы на языке Pascal для обработки одномерных массивов. Формировать ОК-2, ОК-3, ОК-4, ОК-5, ОК-7, ОК-8, овладеть знаниями и умения- ми, необходимыми для освоения ПК-1.3, ПК-2.2, ПК-2.3

**Подготовка к занятию:** Повторить теоретический материал по теме «Одномерные масси- вы».

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:**

## ВАРИАНТ 1

* 1. Заполнить одномерный массив случайными числами, найти его наибольший эле-

мент.

* 1. В одномерном массиве, состоящем из n целочисленных элементов, вычислить сум-

му элементов массива, расположенных между первым и последним нулевыми элементами. Элементы массива вводятся с клавиатуры.

## ВАРИАНТ 2

1. Заполнить одномерный массив случайными числами, найти его наименьший эле-

мент.

1. В одномерном массиве, состоящем из n вещественных элементов, вычислить сумму

элементов массива, расположенных между первым и последним отрицательными элемента- ми. Элементы массива вводятся с клавиатуры.

## ВАРИАНТ 3

1. Заполнить одномерный целочисленный массив случайными числами; найти произ- ведение чисел массива, превышающих некое число m (если таковых нет – вывести соответ- ствующее сообщение).
2. В одномерном массиве, состоящем из n вещественных элементов, вычислить сумму элементов массива, расположенных до последнего положительного элемента. Элементы мас- сива вводятся с клавиатуры.

## ВАРИАНТ 4

1. Заполнить одномерный вещественный массив случайными числами; найти такие из них, которые входят в интервал [m, n] (произвольные m и n вводятся с клавиатуры).
2. В одномерном массиве, состоящем из n вещественных элементов, вычислить сумму элементов массива, расположенных между первым и последним положительными элемента- ми. Элементы массива вводятся с клавиатуры.

## ВАРИАНТ 5

1. Заполнить одномерный вещественный массив случайными числами, найти сумму отрицательных элементов массива.
2. Даны целочисленные массивы А(n) , В(n). Создать новый массив С(n), элементами которого являются произведения соответствующих элементов массивов А и В. Элементы массивов вводятся с клавиатуры.

## ВАРИАНТ 6

1. Заполнить одномерный целочисленный массив случайными числами, найти произ- ведение элементов массива с четными номерами.
2. Дан массив А(n). Создать новый массив, элементами которого являются произве- дение A(i) на максимальный элемент массива A. Элементы массива вводятся с клавиатуры.

## ВАРИАНТ 7

1. Заполнить одномерный целочисленный массив случайными числами, вычислить сумму элементов массива с нечетными номерами.
2. Дан массив А(n). Уменьшить все его элементы на минимальный элемент массива. Элементы массива вводятся с клавиатуры.

## ВАРИАНТ 8

1. Заполнить одномерный целочисленный массив случайными числами, все четные элементы заменить на 0, а нечетные на 1.
2. Дан массив А(n), состоящий из целых чисел. Найти количество и сумму тех эле- ментов, которые делятся на 5 и не делятся на 7. Элементы массива вводятся с клавиатуры.

## ВАРИАНТ 9

1. Заполнить одномерный целочисленный массив случайными числами, посчитать в нем количество нулей и единиц.
2. Дан массив С(n). Поменять знак на противоположный у всех отрицательных эле- ментов массива. Элементы массива вводятся с клавиатуры.

## ВАРИАНТ 10

1. Заполнить одномерный вещественный массив случайными числами, найти сумму всех положительных и произведение всех отрицательных элементов. Результат вывести на экран.
2. Массив А(n) вводится с клавиатуры. Найти среднее арифметическое его элементов с нечетными номерами.

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Составить программы на языке Pascal;
3. Ввести программы в ЭВМ и откомпилировать их;
4. Выполнить программы и занести результаты в отчет;
5. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (код программ, результаты выполнения программ);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Что такое одномерный массив?
2. Как описать массив?
3. Как определить местоположение элемента в массиве?
4. Ка получить доступ к элементу массива?

# ПРИЛОЖЕНИЕ

**Массивы**

С понятием "массив" приходится сталкиваться при решении научно-технических и экономических задач обработки совокупностей большого количества значений. В общем случае ***массив*** - это структурированный тип данных, состоящий из фиксированного числа элементов, имеющих один и тот же тип.

Название *регулярный тип* (или ряды) массивы получили за то, что в них объединены однотипные (логически однородные) элементы, упорядоченные по индексам, определяющим положение каждого элемента в массиве.

В качестве элементов массива можно использовать и любой другой ранее описанный тип, поэтому вполне правомерно существование массивов записей, массивов указателей, массивов строк, массивов и т.д. Элементами массива могут быть данные любого типа, вклю- чая структурированные. Тип элементов массива называется *базовым*. Особенностью языка Паскаль является то, что *число элементов массива фиксируется* при описании и в процессе выполнения программы не меняется.

Элементы, образующие массив, упорядочены таким образом, что каждому элементу соответствует совокупность номеров (индексов), определяющих его местоположение в об- щей последовательности. *Доступ* к каждому отдельному элементу осуществляется путем индексирования элементов массива. *Индексы* представляют собой выражения любого ска- лярного типа, кроме вещественного. Тип индекса определяет границы изменения значений индекса. Для описания массива предназначено словосочетание: *array of .*

Формат записи массивов: Type

<имя типа> = array [тип индекса] of <тип компонента>;

Var

<идентификатор> : <имя типа>;

Массив может быть описан и без представления типа в разделе описания типов данных: Var <идентификатор> : array [тип индекса] of <тип компонента>;

### Примеры описания одномерных массивов

Если в описании массива задан один индекс, массив называется *одномерным*. Одно- мерный массив соответствует понятию линейной таблицы (вектора).

*Одномерные массивы описываются следующим образом:*

Type

Var

Klass = (К1, К2, КЗ, К4) ;

Znak = array [1..255] of char;

Ml: Znak; {Тип Znak предварительно описан в разделе типов} М2: array[1..4] of Klass;

М3: array[1..60] of integer; {Прямое описание массива М3} Mas: array[1..4] of integer;

В Паскале количество элементов массива всегда должно быть фиксировано, т. е. определяться при трансляции программы.

### Действия над массивами

Для работы с массивом как единым целым используется идентификатор массива без указания индекса в квадратных скобках. Массив может участвовать только в операциях от- ношения *"равно", "не равно" и в операторе присваивания*. Массивы, участвующие в этих действиях, должны быть идентичны по структуре, т. е. иметь одинаковые типы индексов и одинаковые типы компонентов.

Например, если массивы А и В описаны как

*Var А, В: array[1..20] of real,*

то применение к ним допустимых операций даст следующий результат:

*Выражение Результат*

*А=В True*, если значение каждого элемента массива А равно со- ответствующему значению элемента массива В

*А<>В True*, если хотя бы одно значение элемента массива А не равно значению соответствующего элемента массива В

*А:=В* Все значения элементов массива В присваиваются соот- ветствующим элементам массива А. Значения элементов массива В остаются неизменны.

### Действия над элементами массива

После объявления массива каждый его элемент можно обработать, указав *идентифи- катор (имя) массива* и *индекс элемента* в квадратных скобках. Например, запись *Mas[2], Vector[10]* позволяет обратиться ко второму элементу массива *Mas* и десятому элементу мас- сива *Vector*.

Индексированные элементы массива называются *индексированными переменными* и могут быть использованы так же, как и простые переменные. Например, они могут нахо- диться в выражениях в качестве операндов, использоваться в операторах *for, while, repeat*, входить в качестве параметров в операторы *Read, Readin, Write, Writeln*; им можно присваи- вать любые значения, соответствующие их типу.

Рассмотрим типичные ситуации, возникающие при работе с данными типа *array*. Для этого опишем три массива и четыре вспомогательные переменные:

Var

A,D : array[l.,4] of real;

В : array[1..10,1..15] of integer; I, J, К : integer;

S : real;

*Инициализация (присваивание начальных значений) массива* заключается в присваива- нии каждому элементу массива одного и того же значения, соответствующего базовому ти- пу. Наиболее эффективно эта операция выполняется с помощью оператора for, например:

*Инициализация элементов одномерного массива А: for I := 1 to 4 do*

*A[I] := 0;*

### Ввод-вывод элементов массива

Паскаль не имеет средств ввода-вывода элементов массива сразу, поэтому ввод и вывод значений производится поэлементно. Значения элементам массива можно присвоить с по- мощью оператора присваивания, как показано в примере инициализации, однако чаще всего они вводятся с экрана с помощью оператора *Read* или *Readln* с использованием оператора организации *цикла for*:

*Ввод элементов одномерного массива А:*

*for I:=l to 4 do Readln(A[I]) ;*

В связи с тем, что использовался оператор *Readln*, каждое значение будет вводиться с новой строки. Можно ввести и значения отдельных элементов, а не всего массива. Так, опе- раторами:

*Read(A[3]);*

вводится значение третьего элемента массива А.

*Вывод значений элементов массива* выполняется аналогичным образом, но использу- ются операторы *Write* или *Writeln*:

*Вывод элементов одномерного массива А:*

for I := I to 4 do Writeln (A[I]);

*Копированием* массивов называется присваивание значений всех элементов одного массива всем соответствующим элементам другого массива. Копирование можно выполнить одним оператором присваивания, например *А:=D;* или с помощью оператора *for*:

for I := 1 to 4 do A[I] := D[I];

В обоих случаях значение элементов массива D не изменяется, а значения элементов массива А становятся равными значениям соответствующих элементов массива D. Очевид- но, что оба массива должны быть идентичны по структуре.

Иногда требуется осуществить *поиск в массиве* каких-либо элементов, удовлетворяю- щих некоторым известным условиям. Пусть, например, надо выяснить, сколько элементов массива А имеют нулевое значение.

К := 0;

for I := 1 to 4 do

if A[I] = 0 then К := К + 1;

После выполнения цикла переменная *К* будет содержать количество элементов массива

*А* с нулевым значением.

*Перестановка значений элементов массива* осуществляется с помощью дополнитель- ной переменной того же типа, что и базовый тип массива.

Например, так запишется фрагмент программы, обменивающий значения первого и пятого элементов массива А:

Vs:= A[5]; {Vs - вспомогательная переменная} А[5]:= А[1];

А[1]:= Vs;

### Примеры решения типовых задач на одномерные массивы

***Пример 1.*** *Сформировать и вывести на экран последовательность из п элементов одномерного массива, вводимых с клавиатуры.*

program primer1; var

mas:array [1..100] of integer; n,i:integer; BEGIN

writeln('Введите количество элементов массива не больше 100: '); readln(n);

for i:=1 to n do *{ввод элементов массива}*

begin

writeln('Введите ',i,' элемент массива'); readln(mas[i]);

end;*{конец ввода}*

writeln('Введенный массив: ');

for i:=1 to n do *{вывод элементов массива}*

begin

write(' ',mas[i],' ');

end; *{конец вывода}*

END.

***Пример 2.*** *Сформировать и вывести на экран последовательность из n элементов, заданных датчиком случайных чисел на интервале [-23, 34].*

Program posled;

Var a: array[1..100] of integer; i, n: integer;

Begin

Write (‘Сколько элементов? ’); Readln (n);

For i=1 to n do begin

a[i]:= Random(58)-23;

writeln (a[i],’ ‘); end;

End.

***Пример 3.*** *Найти сумму элементов одномерного массива. Размер произвольный. Эле- менты вводятся с клавиатуры.*

Program summa;

Var a: array[1..100] of real; i, n: integer;

s: real;

Begin

Write (‘n=’); Readln (n); s:=0;

For i:=1 to n do begin

write (‘введите число’); readln (a[i]); s:=s+a[i];

end;

writeln(‘сумма элементов равна ‘,s); End.

***Пример 4.*** *Найти номер наименьшего элемента в массиве, заданного датчиком слу- чайных чисел на интервале [-20, 25]. Размер произвольный.*

Program numberminim;

Var a: array[1..100] of integer; i, n, num, min: integer;

Begin

Write (‘n=’); Readln (n);

For i:=1 to n do begin

a[i]:= Random(46)-20;

writeln (a[i]); end;

min:=a[1]; num:=1;

For i:=2 to n do

If a[i]< min then begin

min:=a[i]; num:=i;

end;

Writeln(‘ номер наименьшего элемента: ‘,num); End.

***Пример 5.*** *Найти произведение элементов целочисленного одномерного массива с четными номерами, состоящего из n элементов. Элементы вводятся с клавиатуры.*

Program proizved\_chet;

Var a: array [1..100] of integer; i, n, p: integer;

Begin p:=1;

write ('n='); readln (n); for i:=1 to n do

begin

write ('a[',i,']='); readln (a[i]); if i mod 2=0 then p:=p\*a[i]; end;

Writeln ('произведение элементов массива с четными номерами равно ',p); End.

***Пример 6.*** *Сортировка целочисленного массива в порядке возрастания*

program z5\_3;uses crt; const n=10;

var x:array [1..n] of integer; i,j,h:integer; BEGIN

clrscr; randomize; writeln('исходный массив:'); for i:=1 to n do begin

x[i]:=random(n+1); write(x[i]:3); end;

writeln; writeln('упорядоч. массив:'); for i:=2 to n do begin

for j:=n downto i do begin if x[j-1]>x[j] then begin h:=x[j-1];

x[j-1]:=x[j]; x[j]:=h;

end; end; end;

for j:=1 to n do write(x[j]:3); END.

***Практическое занятие №7***

**Наименование занятия: *Двумерные массивы***

**Цель занятия:** Получить практические навыки в составлении и отладке программ на языке Pascal для обработки двумерных массивов. Формировать ОК-2, ОК-3, ОК-4, ОК-5, ОК-7, ОК-8, овладеть знаниями и умениями, необходимыми для освоения ПК-1.3, ПК-2.2, ПК-2.3 **Подготовка к занятию:** Повторить теоретический материал по теме «Двумерные массивы» **Литература:**

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:**

## ВАРИАНТ 1

Дана целочисленная квадратная матрица. Вывести ее на экран и определить сумму элементов в тех строках, которые не содержат отрицательных элементов.

## ВАРИАНТ 2

Дана целочисленная прямоугольная матрица. Вывести ее на экран и определить сум- му элементов в тех строках, которые содержат хотя бы один отрицательный элемент.

## ВАРИАНТ 3

Дана целочисленная квадратная матрица. Вывести ее на экран и определить сумму элементов тех столбцах, которые не содержат отрицательных элементов.

## ВАРИАНТ 4

Дана целочисленная квадратная матрица. Вывести ее на экран и определить произве- дение элементов в тех строках, которые не содержат отрицательных элементов.

## ВАРИАНТ 5

Дана целочисленная прямоугольная матрица. Вывести ее на экран и определить коли- чество столбцов, содержащих хотя бы один нулевой элемент.

## ВАРИАНТ 6

Дана целочисленная прямоугольная матрица. Вывести ее на экран и определить коли- чество строк, не содержащих ни одного нулевого элемента.

## ВАРИАНТ 7

Дана целочисленная прямоугольная матрица. Вывести ее на экран и определить коли- чество четных отрицательных элементов.

## ВАРИАНТ 8

Дана целочисленная прямоугольная матрица. Вывести ее на экран и определить коли- чество четных элементов в каждом столбце.

## ВАРИАНТ 9

Дана целочисленная прямоугольная матрица. Вывести ее на экран и определить коли- чество отрицательных элементов в каждой строке матрицы

## ВАРИАНТ 10

Дана целочисленная прямоугольная матрица. Вывести ее на экран и определить коли- чество положительных элементов в каждом столбце.

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Составить программу на языке Pascal;
3. Ввести программу в ЭВМ и откомпилировать её;
4. Выполнить программу и занести результаты в отчет;
5. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (код программы, результаты выполнения программы);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Что такое двумерный массив?
2. Как описывается двумерный массив?
3. Как получить доступ к элементам двумерного массива?

# ПРИЛОЖЕНИЕ

### Двумерные массивы

Двумерный массив можно представить как таблицу или матрицу. Для получения до- ступа к его элементам используются два индекса: номер строки и номер столбца. При описа- нии в типе индексов надо указать диапазон для двух индексов массива.

Type Var

*Примеры описания двумерных массивов:*

Vector = array[1..4] of integer; Massiv = array[1..4] of Vector;

Matrix : Massiv;

Ту же структуру можно получить, используя другую форму записи:

Var

Matrix : array[1..4,1..4] of integer ;

Для описания массива можно использовать предварительно определенные константы:

Const

G1 = 4; G2 = 6;

Var

MasY: array[1..Gl, l..G2] of real;

*Элементы массива располагаются в памяти последовательно. Элементы с меньшими значениями индекса хранятся в более низких адресах памяти. Многомерные массивы распо- лагаются таким образом, что самый правый индекс возрастает самым первым.*

Например, если имеется массив: A:array[1..5,1..5] of integer;

то в памяти элементы массива будут размещены по возрастанию адресов: А[1,1] А[1,2] … А[1,5] А[2,1] А[2,2] … А[5,5]

При работе с *двумерным массивом* указываются два индекса, с n-мерным массивом - n индексов. Например, запись *Matr[4,4]* делает доступным для обработки значение элемента, находящегося в *четвертой строке четвертого столбца* массива *Matr*.

### Процедуры обработки матриц

#### Сумма указанной строки:

S:=0; i:=3;

For J:=1 to m do S:=S+b[i,j];

#### Транспонирование квадратной матрицы:

For i:=1 to n-1 do For j:=1 to n do

Begin

P:= a[i,j]; a[i,j]:=a[j,i]; a[j,i]:=p; end;

#### Удаление строки из матрицы:

n:=n-1;

For i:=1 to n do

For j:=1 to m do b[i,j]:=b[i+1,j];

#### Включение строки в матицу:

i:=n;

while i>=k do begin

for j:=1 to m do b[i+1,j]:=b[i,j]; i:=i-1;

end;

for j:=1 to m do b[k,j]:=c[j]; n:=n+1;

#### Перестановка строк матрицы:

For k:=1 to k do c[k]:=a[i,k];

For k:=1 to k do a[i,k]:=a[j,k];

For k:=1 to k do a[j,k]:=c[k];

#### Поиск минимального элемента матрицы:

Min:= a[1,1]; k;=1; L:=1;

For i:=1 to n do For j:=1 to m do

IF min>a[i,j] then begin min:=a[i,j]; k:=i; l:=j; end;

#### Сложение строк матрицы:

For j:=1 to m do a[k,j]:=a[k,j]+a[l,j]\*b;

### Примеры решения типовых задач на двумерные массивы

***Пример 1.*** *Программа ввода-вывода двумерного массива*

program primer; var i,j,n,m:integer;

massiv:array[1..10,1..10] of integer; *{описан двумерный массив с именем massiv}*

BEGiN

write('Введите количество строк массива (не больше 10): '); read(n); *{n-количество строк}* write('Введите количество столбцов массива (не больше 10):'); read(m); *{m-количество столбцов}*

for i:=1 to n do *{ввод элементов двумерного массива}*

begin

for j:=1 to m do read(massiv[i,j]);

end; *{конец ввода}*

writeln('Введенный массив: ');

for i:=1 to n do *{вывод элементов двумерного массива}*

begin

for j:=1 to m do write(massiv[i,j]:5);

end; *{конец вывода}*

END.

***Пример 2.*** *Сформировать с помощью датчика случайных чисел и вывести на экран матрицу, размером МхN. Элементы задаются на интервале [-20, 25].*

Var a: array[1..50,1..50] of integer; i, j, n, m: integer;

Begin

Write(‘сколько строк?’); Readln(m); Write(‘сколько столбцов?’); Readln(n);

For i:=1 to m do begin

For j:=1 to n do begin

a[i,j]:=int(rnd\*46)-20;

write(a[i,j],’ ‘); end;

writeln; end;

End.

***Пример 3.*** *В двумерном массиве, состоящем из целых чисел, найти наименьший эле- мент и номер строки, в которой он находится. Элементы вводятся с клавиатуры. Размер MXN.*

Program minim;

Var a: array[1..50,1..50] of integer; i, j, m, n, min, K: integer;

Begin

Write(‘сколько строк?’); Readln(m);

Write(‘сколько столбцов?’); Readln(n); For i:=1 to m do

For j:=1 to n do

begin write(‘a[‘,i,’,’,j,’]=’); readln (a[i,j]); end; min:=a[1,1]; K:=1;

For i:=1 to m do For j:=1 to n do

If a[i,j]< min then begin

min:=a[i,j]; K:=i; end;

Writeln(‘наименьшее число ‘,min,’ находится в ‘, k , ‘ строке’); End.

***Практическое занятие №8***

**Наименование занятия: *Работа со строками***

**Цель занятия:** Научиться составлять программы с использованием строковых типов дан- ных. Формировать ОК-2, ОК-3, ОК-4, ОК-5, ОК-7, ОК-8, овладеть знаниями и умениями, не- обходимыми для освоения ПК-1.3, ПК-2.2, ПК-2.3

**Подготовка к занятию:** Повторить теоретический материал по теме «Строки»

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:**

## ВАРИАНТ 1

* 1. Дана строка символов, среди которых есть одна открывающаяся и одна закрывающа- яся скобки. Вывести на экран все символы, расположенные внутри этих скобок.
  2. Определить, сколько раз в строке встречается заданное слово.

## ВАРИАНТ 2

1. Дана строка символов *S.* Подсчитать, сколько раз среди данных символов встреча- ется буква *х.*
2. Дана строка. Преобразовать ее, удалив каждый символ \* и повторив каждый сим- вол, отличный от \*.

## ВАРИАНТ 3

1. Дана строка символов. Удалить в ней первый знак препинания.
2. Дана строка, заканчивающаяся точкой. Посчитать, сколько слов в строке.

## ВАРИАНТ 4

1. Дана строка символов. Выяснить имеются ли в данной строке рядом стоящие запя- тая и тире.
2. Дана строка. Указать те слова, которые содержат хотя бы одну букву *t*.

## ВАРИАНТ 5

1. Дана строка символов. Преобразовать строку, заменив в ней каждую точку много- точием.
2. В строке имеется одна точка с запятой (;). Посчитать, сколько символов до точки с

запятой и после нее.

## ВАРИАНТ 6

1. Дана строка символов*.* Подсчитать, сколько в ней букв *r, k, t.*
2. В строке заменить все двоеточия (:) точкой с запятой (;) и посчитать количество

замен.

## ВАРИАНТ 7

1. Дана строка символов, среди которых есть двоеточие (:). Определить, сколько сим- волов ему предшествует.
2. Удалить часть символьной строки, заключенной в скобки (вместе со скобками).

## ВАРИАНТ 8

1. Дана строка символов. Определить, сколько раз в нее входит группа *abc*.
2. В строке удалить все символы «двоеточие» (:) и посчитать количество удаленных символов.

## ВАРИАНТ 9

1. Дана строка символов. Преобразовать строку, заменив в ней все восклицательные знаки точками.
2. Дана строка, содержащая английский текст. Найти количество слов, начинающихся с буквы b.

## ВАРИАНТ 10

1. Дана строка символов. Удалить из неё все знаки препинания.
2. Дана строка, содержащая текст. Найти длину самого короткого и самого длинного

слова.

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Составить программы на языке Pascal;
3. Ввести программы в ЭВМ и откомпилировать их;
4. Выполнить программы и занести результаты в отчет;
5. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (код программ, результаты выполнения программ);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Какой тип данных называют строковым?
2. Что такое текущая длина и длина строки?
3. Какие стандартные процедуры и функции применяются при работе со строковыми ве- личинами?

# ПРИЛОЖЕНИЕ

**Строки**

***Строка*** *—* это последовательность символов кодовой таблицы персонального компь- ютера. При использовании в выражениях строка заключается в апострофы. Количество сим- волов в строке (длина строки) может динамически изменяться от 0 до 255. Для определения данных строкового типа используется идентификатор *String*, за которым следует заключен- ное в квадратные скобки значение максимально допустимой длины строки данного типа. Ес- ли это значение не указывается, то по умолчанию длина строки равна 255 байт.

Переменную строкового типа можно определить через описание типа в разделе опре- деления типов или непосредственно в разделе описания переменных. Строковые данные мо- гут использоваться в программе также в качестве констант.

Недопустимо применение строковых переменных в качестве *селектора* в операторе

*Case.*

Определение строкового типа устанавливает максимальное количество символов, ко-

торое может содержать строка.

### Формат описания строкового типа

*Type*

*Var*

*<имя типа> =String [максимальная длина строки];*

*<идентификатор, . . . > : <имя типа>;*

Переменную типа *String* можно задать и без описания типа:

*Var*

*<идентификатор, . . . > : String [максимальная длина строки];*

### Пример описания строковых данных

*Const*

*Type Var*

*Address = 'ул. Переверткина, 25'; {Строковая константа} Flot = string[125];*

*Fstr : Flot; {Описание с заданием типа)*

*St1 : String; {По умолчанию длина строки St1= 255} St2, St3 : string[50] ;*

*Nazv : string[280]; (Ошибка, длина строки Nazv превышает 255}*

Для ввода и вывода переменной типа STRING используются операторы READLN и WRITELN.

#### Операции над строками

1. Операции сравнения. Сравнение происходит посимвольно слева направо: сравни- ваются коды соответствующих символов, пока не нарушится равенство. Две строки счита- ются равными, если они равны по длине и совпадают посимвольно.
2. Строки можно объединять с помощью операции сцепления. Знак операции - «+».
3. Индексирование. Так как строку можно рассматривать как массив символов, то при помощи индексирования можно организовать доступ к его отдельным символам по их но- мерам.

### Стандартные строковые процедуры и функции

***Delete (Str,Poz,N)*** — удаление *N* символов строки *Str*, начиная с позиции *Poz*. Если зна- чение Poz > 255, возникает программное прерывание.

***Insert (Strl, Str2, Роz)*** –вставка строки *Str1* в строку *Str2*, начиная с позиции *Poz*. *Например:*

Var

Sl, S2 : string[ll] ;

…

S1 := ' ЕС ';

S2 := 'ЭВМ1841';

Insert(S1,S2,4) ;

В результате выполнения последнего выражения значение строки *S2* станет равным

*'ЭВМ ЕС 1841'*.

***Str (IBR,St)*** — преобразование числового значения величины *IBR* и помещение резуль- тата в строку *St*. После IBR может записываться формат, аналогичный формату вывода. Если в формате указано недостаточное для вывода количество разрядов, поле вывода расширяется автоматически до нужной длины.

***Val (St,IBR,Code)*** — преобразует значение *St* в величину целочисленного или веще- ственного типа и помещает результат в *IBR*. Значение *St* не должно содержать незначащих пробелов в начале и в конце. *Code* — целочисленная переменная. Если во время операции преобразования ошибки не обнаружено, значение *Code* равно нулю, если ошибка обнаруже- на (например, литерное значение переводится в цифровое), *Code* будет содержать номер по- зиции первого ошибочного символа, а значение *IBR* не определено.

***Copy (St,Poz,N)*** — выделяет из строки *St* подстроку длиной *N* символов, начиная с по- зиции *Poz.* Если *Poz > Length(St),* то результатом будет пробел; если *Poz > 255*, возникнет ошибка при выполнении. Функция *Length* описана ниже. *Poz, N* — целочисленные выраже- ния.

***Concat (Strl,Str2,...,StrN)*** — выполняет сцепление строк *Strl, Str2,..,StrN* в том порядке, в каком они указаны в списке параметров. Сумма символов всех сцепленных строк не должна превышать 255.

***Length (St)*** — вычисляет текущую длину в символах строки *St.* Результат имеет цело- численный тип.

***Pos (Strl,Str2)*** — обнаруживает первое появление в строке *Str2* подстроки *Strl*. Резуль- тат имеет целочисленный тип и равен номеру той позиции, где находится первый символ подстроки *Strl.* Если в *Str2* подстроки *Strl* не найдено, результат равен 0.

***UpCase (Ch)*** — преобразует строчную букву в прописную. Параметр и результат име- ют литерный тип. Обрабатывает буквы только латинского алфавита.

### Примеры программ работы со строковыми переменными

***Пример 1****. Составить программу, определяющую, является ли введенное слово пе- ревертышем. Перевертышем называется слово, которое одинаково читается как сначала, так и с конца, например: шалаш, казак.*

Как видно из определения, для выяснения, является ли слово перевертышем, необхо- димо сравнивать 1-й и последний символ в строке, 2-й и предпоследний, 3-й и предпредпо- следний символ, и т. д. до середины слова. Если в процессе сравнения будет установлено от- личие сравниваемых символов, т. е. выясняется, что слово читается слева направо иначе, чем справа налево, значит можно сделать вывод, что это слово не является перевертышем. Если в процессе сравнения не будет выявлено отличие сравниваемых символов, значит это слово — перевертыш. Введем следующие переменные: для хранения слов — *Word* типа *String* с мак- симальным размером слов 30 символов и переменную *I* целого типа, указывающую номер позиции сравниваемого символа от начала строки.

program Perev\_Word; *{Является ли введенное слово перевертышем?}*

Var

I : byte;

Word : string[30]; Begin

Write('Введите слово '); Readln(Word) ;

*{Проверяем символы поочередно от начала до середины слова}*

for I:=1 to Trunc (Length (Word)/ 2) do begin

end;

*{Если соответствующие символы не одинаковы}*

if Word[I]<>Word[Length(Word)-I+1] then begin

Writein (' Неперевертыш') ;

exit *{Выход из цикла и завершение программы, дальше не имеет смысла сравнивать }*

end;

Writeln (' Перевертыш'); end.

***Практическое занятие №9***

**Наименование занятия: *Множества***

**Цель занятия:** Научиться составлять программы с использованием множественного типа данных. Формировать ОК-2, ОК-3, ОК-4, ОК-5, ОК-7, ОК-8, овладеть знаниями и умениями, необходимыми для освоения ПК-1.3, ПК-2.2, ПК-2.3

**Подготовка к занятию:** Повторить теоретический материал по теме «Множества»

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:**

## ВАРИАНТ 1

Известны сорта роз, выращиваемые тремя цветоводами: «Анжелика», «Виктория»,

«Гагарин», «Ave Maria», «Катарина», «Юбилейная». Определить те сорта, которые имеются у каждого из цветоводов; которые есть хотя бы у одного из цветоводов; которых нет ни у од- ного из цветоводов.

## ВАРИАНТ 2

Дан текст, заканчивающийся точкой. Напечатать в алфавитном порядке все гласные русские буквы (а, е, и, о, у, ы, э, ю, я), входящие в этот текст.

## ВАРИАНТ 3

Имеется множество, содержащее натуральные числа из некоторого диапазона. Сфор- мировать два множества, первое из которых содержит все простые числа, а второе – все со- ставные.

## ВАРИАНТ 4

Задан некоторый набор товаров. Определить для каждого из товаров, какие из них имеются в каждом из *n* магазинов, какие товары есть хотя бы в одном магазине и каких това- ров нет ни в одном магазине.

## ВАРИАНТ 5

Есть список игрушек, некоторые из которых имеются в N детских садах. Определить игрушки из списка, которые есть в каждом детском саду, которых нет ни в одном из садов.

## ВАРИАНТ 6

Дан текст из строчных латинских букв, заканчивающийся точкой. Напечатать все буквы, входящие в текст не менее двух раз.

## ВАРИАНТ 7

Имеется список группы (все имена различны). Определить, есть ли в группе человек, который побывал в гостях у всех. (Для каждого студента составить множество побывавших у него друзей, сам он в это множество не входит).

## ВАРИАНТ 8

Из диапазона целых чисел m..n выделить множество чисел, делящихся на *k* или *l* без остатка и множество чисел делящихся на *k* · *l* без остатка.

## ВАРИАНТ 9

Дан текст из строчных латинских букв, заканчивающийся точкой. Напечатать все буквы, входящие в текст по одному разу.

## ВАРИАНТ 10

В озере водится несколько видов рыб. Три рыбака поймали рыб, представляющие не- которые из имеющихся видов. Определить: какие виды рыб есть у каждого рыбака; какие виды есть в озере, но нет ни у одного из рыбаков.

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Составить программу на языке Pascal;
3. Ввести программу в ЭВМ и откомпилировать её;
4. Выполнить программу и занести результат в отчет;
5. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (код программы, результат выполнения программы);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Что называется множеством?
2. Как описываются множества?
3. Перечислите операции над множествами.

# ПРИЛОЖЕНИЕ

### Множества

*Множество* – это структурированный тип данных, представляющий собой набор взаи- мосвязанных по какому-либо признаку или группе признаков объектов, которые можно рас- сматривать как единое целое. Каждый объект во множестве называется *элементом множе- ства*.

Все элементы множества должны принадлежать одному из скалярных типов, кроме вещественного. Этот тип называется *базовым типом множества*. Базовый тип задается диа- пазоном или перечислением. *Область значений* типа множество – набор всевозможных под- множеств, составленных из элементов базового типа.

В выражениях на языке Паскаль значения элементов множества указываются в квад- ратных скобках:[1,2,3,4], [‘a’,’b’,’c’], [‘a’..’z’]. Если множество не имеет элементов, оно называется *пустым* и обозначается, как [ ]. Количество элементов множества называется его *мощностью*. В отличие от массива порядок перечисления элементов во множестве не имеет значения, и количество элементов заранее не определено. Количество элементов, входя- щих во множество, может меняться от 0 до 256. Значения множества задаются в квадрат- ных скобках перечислением элементов через запятую.

Например,

[ ]- пустое множество;

[2, 3, 7,11] - множество из целых чисел; ['а', 'с', Т]; - множество из символов;

[1..10] - множество из элемента ограниченного типа;

[k,.2\*k] - элемент множества задастся текущим значением переменной k.

#### Формат записи множественных типов:

*Type*

<имя типа> = **set** of <элемент 1,…., элементN>;

*Var*

<идентификатор, ….> : <имя типа>;

Можно задать множественный тип и без предварительного описания:

*Var*

<идентификатор, …> : set of <элемент1, …>;

Например,

*Type*

Simply = **set** of ‘a’ ..’h’; Number = **set** of 1..31;

*Var*

Pr : Simply; N : Number;

Letter : **set** of char; {определение множества без предварительного описания в разделе типов}

В данном примере переменная *Pr* может принимать значения символов латинского ал- фавита от ‘a’ до ‘h’; *N* – любое значение в диапазоне 1..31; *Letter* – любой символ. Попытка присвоить другие значения вызовет программное прерывание.

Количество элементов множества не должно превышать 256, соответственно номера значений базового типа должны находиться в диапазоне 0..255.

### Операции над множествами

При работе с множествами допускается использование операций отношения “=”, “<>”, “>=”, “<=”, объединения, пересечения, разности множеств и операции in. Результатом выражения с применением этих операций является значение *True* или *False*.

**Операция “равно” (=).** Два множества А и В считаются равными, если состоят из од- них и тех же элементов. Порядок следования элементов в сравниваемых множествах значе- ния не имеет.

*Например:*

|  |  |  |  |
| --- | --- | --- | --- |
| *Значение А* | *Значение В* | *Выражение* | *Результат* |
| [1,2,3,4] [‘a’, ’b’, ’c’]  [‘a’..’z’] | [1,2,3,4]  [‘c’, ‘a’]  [‘z’..’a’] | A=B A=B A=B | True False True |

**Операция “не равно” (<>).** Два множества А и В считаются не равными, если они от- личаются по мощности или по значению хотя бы одного элемента.

*Например:*

|  |  |  |  |
| --- | --- | --- | --- |
| *Значение А* | *Значение В* | *Выражение* | *Результат* |
| [1,2,3] [‘a’..’z’]  [‘c’..’t’] | [3,1,2,4] [‘b’.. ‘z’]  [‘t’..’c’] | A<>B A<>B  A<>B | True True  False |

**Операция “больше или равно” (>=).** Операция “больше или равно” (>=) использует- ся для определения принадлежности множеств. Результат операции А >=В равен True, если все элементы множества В содержаться в множестве А. В противном случае результат равен False.

*Например:*

|  |  |  |  |
| --- | --- | --- | --- |
| *Значение А* | *Значение В* | *Выражение* | *Результат* |
| [1,2,3,4] [‘a’..’z’]  [‘z’,’x’,’c’] | [2,3,4] [‘b’.. ‘t’]  [‘c’,’x’] | A>=B  A>=B A>=B | True  True True |

**Операция “меньше или равно” (<=).** Эта операция используется аналогично преды- дущей операции, но результат выражения А<=В равен True, если все элементы множества А содержаться во множестве В. В противном случае результат равен False.

|  |  |  |  |
| --- | --- | --- | --- |
| *Значение А* | *Значение В* | *Выражение* | *Результат* |
| [1,2,3] [‘d’..’h’]  [‘a’,’v’] | [1,2,3,4] [‘z’.. ‘a’]  [‘a’,’n’,’v’] | A<=B  A<=B A<=B | True  True True |

**Операция in**. Операция in используется для проверки принадлежности какого-либо значения указанному множеству. Обычно применяется в условных операторах.

|  |  |  |
| --- | --- | --- |
| *Значение А* | *Значение В* | *Результат* |
| 2  ‘v’ X1 | if A in [1,2,3] then.. if A in [‘a’..’n’] then..  if A in [X0,X1,X2,X3] then.. | True True  True |

При использовании операции in проверяемое на принадлежность значение и множе- ство в квадратных скобках не обязательно предварительно описывать в разделе описаний. Операция in позволяет эффективно и наглядно производить сложные проверки условий, за- меняя иногда десятки других операций. Например, выражение if(a=1) or (a=2) or (a=3) or (a=4) or (a=5) or (a=6) then…можно заменить более коротким выражением

if a in [1..6] then… .

Часто операцию in пытаются записать с отрицанием: *X NOT in M.* Такая запись является ошибочной, так как две операции следуют подряд; правильная инструкция имеет вид*: NOT (X in M).*

**Объединение множеств (+).** Объединением двух множеств является третье множество, содержащее элементы обоих множеств.

*Например:*

|  |  |  |  |
| --- | --- | --- | --- |
| *Значение А* | *Значение В* | *Выражение* | *Результат* |
| [1,2,3] [‘A’..’D’]  [] | [1,4,5] [‘E’.. ‘Z’]  [] | A+B A+B  A+B | [1,2,3,4,5]  [‘A’..’Z’] [] |

**Пересечение множеств (\*).** Пересечением двух множеств является третье множество, которое содержит элементы, входящие одновременно в оба множества.

*Например:*

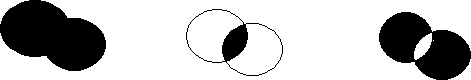
|  |  |  |  |
| --- | --- | --- | --- |
| *Значение А* | *Значение В* | *Выражение* | *Результат* |
| [1,2,3] [‘A’..’Z’]  [] | [1,4,2,5] [‘B’.. ‘R’]  [] | A\*B A\*B  A\*B | [1,2] [‘B’..’R’]  [] |

**Разность множеств (-).** Разностью двух множеств является третье множество, которое содержит элементы первого множества, не входящие во второе множество.

*Например:*

|  |  |  |  |
| --- | --- | --- | --- |
| *Значение А* | *Значение В* | *Выражение* | *Результат* |
| [1,2,3,4] [‘A’..’Z’]  [X1,X2,X3,X4]] | [3,4,1] [‘D’.. ‘Z’]  [X4,X1] | A-B  A-B  A-B | [2] [‘A’..’C’]  [X2,X3] |

Результат операций рад двумя множествами можно наглядно представить с помощью за- крашенных частей двух кружочков:



*Объединение Пересечение Разность*

Использование в программе данных типа **set** дает ряд преимуществ: значительно упро- щаются сложные операторы if , увеличивается степень наглядности программы и понимания алгоритма решения задачи, экономятся память, время компиляции и выполнения.

Имеются и отрицательные моменты, основной из них – отсутствие в языке Паскаль средств ввода-вывода элементов множества, поэтому программист сам должен писать соот- ветствующие процедуры.

#### Пример 1.

**Program** Dem\_Mno; {демонстрация операций над множествами}

***Type***

Digits=**set** of 0..9;

***Var***

D1, D2, D3, D:Digits;

***Begin***

D1:=[2,4,6,8]; {заполнение множеств} D2:=[0..3,5];

D3:=[1,3,5,7,9];

D:=D1+D2; {объединение множеств D1 и D2} D:=D+D3; {объединение множеств D и D3 } D:=D-D2; {разность множеств D и D2 } D:=D\*D1; {пересечение множеств D и D1}

**end**.

Сначала описан тип *Digits=set of 0..9,* затем описаны переменные *D1,D2,D3,D* этого ти- па. В первой части программы осуществляется заполнение множеств, а затем над множе- ствами выполняются операции объединения, пересечения, разности.

***Пример 2.*** *Описать множество М (1..50) и сделать его пустым. Вводя целые числа с клавиатуры, заполнить множество 10 элементами.*

**Program** Input\_Mno;

*Var*

M:**set** of 1..50; X,I:integer;

### Begin

M:= [ ]; {М – пустое множество}

**for** I:=1 **to** 10 **do begin**

write(‘введите ‘, I, ‘ –й элемент множества: ‘); readln (X);

**if** (X **in** M) **then** {если введенное число входит в множество М}

### begin

writeln(X,’ помещен в множество 1..50’); M:=M+[X];

### end;

**end;**

writeln;

### end.

В разделе описания переменных описано множество целых чисел от 1 до 50, перемен- ная Х целого типа, которая используется для считывания числа-кандидата в множество, и целая переменная I, используемая для подсчета количества введенных чисел. В начале про- граммы применена операция инициализации множества М, так как оно не имеет элементов и является пустым:

M:= [ ];

Заполнение множества элементами производится с использованием оператора повто- ра for, параметр которого I будет указывать порядковый номер вводимого элемента. Опера- ция заполнения множества записывается оператором присваивания:

M:=M+[X];

Контроль заполнения множества записан операцией проверки принадлежности in. Ес- ли условие X in M выполняется, выводится сообщение о том, что число Х помещено в мно- жество.

***Пример 3.*** Описывается множество гласных и согласных букв русского языка и опре- деляется количество гласных и согласных букв в предложении, введенном с клавиатуры пользователем.

**Program** Glasn\_Sogl;

*Type*

*Var*

*Begin*

Letters=set of ‘A’..’я’;

Glasn, Sogl:Letters;

Text:String;

I:Byte;

G,S:Byte;

Glasn:=[‘A’,’a’, ‘Е’,’е’,’И’,’и’,’О’,’о’,’У’,’у’,’Э’,’Ю’,’ю’, ‘Я’,’я’];

Sogl:=[‘Б’..’Д’,’б’..’д’,’Ж’,’ж’,’З’,’з’,’К’..’Н’,’к’..’н’,’П’..’Т’,’п’..’т’,’Ф’..’Щ’,’ ‘ф’..’щ’,’Ъ’,’ъ’,’Ь’,’ь’];

Write(‘Введите предложение’); Readln(Text);

G:=0;

S:=0;

***For*** *I:=1* ***to*** *Length(Text)* ***do***

Begin

**If** Text[I] **in** Glasn then G:=G+1;

**If** Text[I] **in** Sogl then S:=S+1;

### End;

Writeln(‘В предложении” ’,Text,’ ”‘,G,’гласных и ‘,S,’согласных букв’);

### End.

Зададим тип Letters –множество букв русского языка, затем опишем переменные этого типа: Glasn-множество гласных букв, Sogl-множество согласных букв. Вводимое с клавиату- ры предложение опишем переменной Text типа String. Для указания символа в строке Text применим переменную I типа Byte.Для подсчета количества гласных и согласных букв опи- шем переменные Gи S. Проверку принадлежности символов, составляющих предложение множествам гласных или согласных букв русского языка запишем с использованием цикла for , параметр I которого, изменяясь от 1 до значения длины предложения, будет указывать порядковый номер символа в предложении. Принадлежность очередного символа предложе- ния множеству гласных или согласных букв запишем операцией in. Если символ является гласной буквой (Text[I] in Glasn), то счетчик гласных букв G увеличивается на 1. Аналогично с согласными буквами.

***Практическое занятие №10***

**Наименование занятия: *Составление программ с использованием процедур***

**Цель занятия:** Научиться составлять программы с использованием процедур на языке Pascal. Формировать ОК-2, ОК-3, ОК-4, ОК-5, ОК-7, ОК-8, овладеть знаниями и умениями, необходимыми для освоения ПК-1.3, ПК-2.2, ПК-2.3

**Подготовка к занятию:** Повторить теоретический материал по теме «Подпрограммы».

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:**

## ВАРИАНТ 1

1. Написать программу, содержащую процедуру для вычисления значения функции *f*(*x*)

1, если *х*  1;

 *x*(*x*  1)

*f* (*x*)  



0,

2

если

, если

x  0.

0  x  1;

1. Написать процедуру перевода числа из градусной меры в радианную.

## ВАРИАНТ 2

1. Написать программу, содержащую процедуру для вычисления значения функции *f*(*x*)

ln(*x*), если x  0;



*f* (*x*) 

- 99999,

если x  0;

ln(*x*2 ), если x  0.



1. Написать процедуру, которая выводит на экран строку, состоящую из звездочек.

## ВАРИАНТ 3

1. Написать программу, содержащую процедуру для вычисления значения функции *f*(*x*)

 *x*  1 ,



 8

если

x  2;

*f* (*x*)   *x*  0,51, если - 2  x  2;

 3

*ex* 8 ,



если

x  -2.

1. Написать процедуру, заменяющую в исходной строке все символы-единицы на

символы-нули. Замена выполняется, начиная с заданной позиции строки.

## ВАРИАНТ 4

1. Написать программу, содержащую процедуру для вычисления значения функции *f*(*x*)

*x*  3 , если х  0,





*f* (*x*)  

7



8

3 , если

8

x  0,

ex ,



если x  0.

1. Написать процедуру нахождения площади треугольника по координатам его вершин

## ВАРИАНТ 5

1. Написать программу, содержащую процедуру для вычисления значения функции *f*(*x*)

1  *х*, если *х*  1;

ln(*x*  1)

*f* (*x*)  

 2

, если

0  x  1;

0, если x  0.

1. Написать процедуру, которая находит из двух целых чисел наибольшее число.

## ВАРИАНТ 6

1. Написать программу, содержащую процедуру для вычисления значения функции *f*(*x*)

 1 , если *х*  1;

*е*

 *х*



*f* (*x*)   *x* ,

 2



1,



если если

0  x  1;

x  0.

1. Написать процедуру нахождения суммы большего и меньшего из трех чисел.

## ВАРИАНТ 7

1. Написать программу, содержащую процедуру для вычисления значения функции *f*(*x*)

 *x* 2  *х*  9,

*f* (*x*)  

если x  8;

 1 , если x  8.

 *х* 4  6

1. Написать процедуру, результатом которой является true, если символ, заданный при обращении к процедуре - буква, и false в противном случае.

## ВАРИАНТ 8

1. Написать программу, содержащую процедуру для вычисления значения функции *f*(*x*)

3, если x  3;

*f* (*x*)  *х*2  3*х*  9, если x  3.



1. Написать процедуру, которая выводит на экран строку, состоящую из одинаковых символов.

## ВАРИАНТ 9

1. Написать программу, содержащую процедуру для вычисления значения функции *f*(*x*)

sin *x*,

*f* (*x*)  

если

x  0;

 0;

0,5 , если x

cos *x*, если x  0.



1. Написать процедуру, которая вычисляет объем и площадь поверхности параллелепи- педа.

## ВАРИАНТ 10

1. Написать программу, содержащую процедуру для вычисления значения функции *f*(*x*)

*x* 2  *x*, если x  5;

*f* (*x*)   *x* 2  1, если - 4  x  5;



lg *x* ,





если

x  4.

1. Написать процедуру, определяющую позицию самого правого вхождения заданного символа в исходную строку. Если строка не содержит символ, то результатом должна быть 1.

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Составить программы на языке Pascal;
3. Ввести программы в ЭВМ и откомпилировать их;
4. Выполнить программы и занести результат в отчет;
5. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (код программ, результат выполнения программ);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Что называется подпрограммой?
2. Что называется процедурой?
3. Как описывается процедура в программе на Паскале?
4. Какова последовательность вызова процедуры?
5. Что такое формальные и фактические параметры?

# ПРИЛОЖЕНИЕ

### Процедуры

Процедура – это подпрограмма, которая решает некоторую частную задачу или объ- единяет группу часто встречающихся операторов. Каждая процедура должна быть предвари- тельно описана в разделе описаний процедур и функций. Описание процедуры состоит из заголовка и тела процедуры. Описание процедуры содержит служебное слово procedure, имя процедуры и заключенный в скобки список формальных параметров с указанием их типов. Формальные параметры отделяются точкой с запятой (список однотипных параметров может быть перечислен через запятую).

Procedure <имя процедуры> (<список формальных параметров>);

После заголовка идут разделы описаний (констант, типов, переменных, процедур и функций, используемых в процедуре) и операторы языка Паскаль, реализующие алгоритм процедуры.

const ... ;

type . . . ;

var . . . ; begin

<операторы>

end ;

Формальные параметры фиктивно присутствуют в процедуре и определяют тип и дей- ствия, которые над ними производятся. Их нельзя описывать в разделе описаний процедуры.

Во время выполнения формальные параметры заменяются фактическими параметрами. Фактические параметры – это параметры, которые передаются процедуре при обращении к ней.

Формальные параметры делятся на параметры – значения и параметры - переменные. Параметры – значения используются в качестве входных данных подпрограммы, при обра- щении к которой фактические параметры передают свое значение формальным и больше не меняются. Параметры-значения задаются следующим образом:

<Имя параметра>:< тип параметра>.

Параметры – переменные могут использоваться как в качестве входных, так и в каче- стве выходных. В заголовке процедуры перед ними необходимо указывать служебное слово var:

Var <имя параметра>:<тип параметра>

При обращении к подпрограмме фактические параметры замещают формальные. В результате выполнения подпрограммы изменяются фактические параметры.

***Вызов процедуры.*** Для обращения к процедуре необходимо использовать оператор вызова процедуры. Он имеет следующий вид:

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

При вызове процедуры основная программа приостанавливает свою работу и передает управление в процедуру. При завершении процедуры передается управление на команду, следующую за вызовом процедуры.

Количество, типы и порядок следования формальных и фактических параметров должны совпадать.

***Пример.*** Подпрограмма должна заполнить массив исходными данными.

Program proba; Const k=100

Type Tm=array[1…k] of integer; Var M:TM ; i; fk; byte;

Procedure InputMas (Var M:TM ; i; fk; byte;) var i: byte;

begin

for i:=1 to10 do begin

Mas[i]:=round(random(50)-25);

{= = = = = = = = = = = = = = = = = = = = = = =}

begin

write(‘Введите количество элементов’); repeat read (fk); until fk<=k; inputMas(M,fk);

for i:=1 to fk do write(M[i]:5); end.

***Практическое занятие №11***

**Наименование занятия: *Составление программ с использованием функций***

**Цель занятия:** Научиться составлять программы с использованием функций на языке Pascal. Формировать ОК-2, ОК-3, ОК-4, ОК-5, ОК-7, ОК-8, овладеть знаниями и умениями, необхо- димыми для освоения ПК-1.3, ПК-2.2, ПК-2.3

**Подготовка к занятию:** Повторить теоретический материал по теме «Подпрограммы».

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:** Составить функцию нахождения факториала и вычислить значение вы- ражения:

## ВАРИАНТ 1

*y* 

( *p*  *m*)!*n*!

## ВАРИАНТ 6

*y*  *k*!

( *p*  *n*)!(*m*  *n*)! (*n*  *k*)!(*k*  *m*)!

## ВАРИАНТ 2 ВАРИАНТ 7

*y*  *n*! *p*!(*n*  *p*)!

 *m*!

*y*  ( *p*  *n*)!

*p*!(*m*  *n*)!

## ВАРИАНТ 3

*y*  *n*!

## ВАРИАНТ 8

*y* 

*m*!*n*!

*k*!(*n*  *k*)! ( *p*  *n*)!(*m*  *n*)!

## ВАРИАНТ 4

*y* 

*n*!*m*!

## ВАРИАНТ 9

*y*  *m*!

*m*!(*n*  *m*)! ( *p*  *n*)!(*m*  *n*)!

## ВАРИАНТ 5 ВАРИАНТ 10

*y*  (*k*  *m*)!*n*!

(*n*  *k*)!

*y*  *m*!*n*! *n*!(*m*  *p*)!

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Составить программу на языке Pascal;
3. Ввести программу в ЭВМ и откомпилировать её;
4. Выполнить программу и занести результат в отчет;
5. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (код программы, результат выполнения программы);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Как описывается функция в программе в языке Паскаль?
2. Последовательность вызова функций?
3. Чем отличается функция от процедуры?

# ПРИЛОЖЕНИЕ

### Функции

Функция, определенная пользователем, состоит из *заголовка* и тела *функции*.

*Заголовок* содержит зарезервированное слово *function,* имя функции, заключенное в круглые скобки, необязательный список формальных параметров и тип возвращаемого функцией значения. Тело функции представляет собой локальный блок, по структуре анало- гичный программе. В целом структура функции, определенной пользователем имеет вид:

*function <имя> (формальные параметры) : <тип результата>; const ...*

*type ... var begin*

*<операторы>*

*end;*

В разделе операторов должен находиться, по крайней мере, один оператор, *присваи- вающий имени функции значение*. В точку вызова возвращается результат последнего при- сваивания.

Обращение к функции осуществляется по имени с необязательным указанием списка аргументов. Каждый аргумент должен соответствовать формальным параметрам, указанным в заголовке, и иметь тот же тип.

***Пример.*** Составить программу вычисления выражения Z=( А5 + А \_3 )/2\* АМ , в ко- торой возведение в степень выполняется *функцией Step*.

program DemoFunc; Var

М : integer;

А,Z,R : real ;

*{Функция вычисления степени. N - степень, X – число, возводимое в данную степень. N, X — формальные параметры; результат, возвращаемый функцией в точку вызова, имеет веще- ственный тип}*

function Step(N : integer; X : real): real; Var

I : integer; Y *:* real; begin Y:=1;

for I:=1 to N do*{Цикл вычисления N—й степени числа X)*

Y:=Y\*X;

Step:=Y ; *{Присваивание функции результата вычисления степени}*

end; *{Конец функции}*

Begin *{Начало основной программы}* Write('Введите значение числа А и показатель степени М'); Readln(A,M) ;

Z:=Step(5,А) ; *{Вызов функции с передачей ей фактических параметров N=5, X=А}*

Z:=Z+ Step(3,l/A); *{Вызов функции с передачей ей фактических параметров N=3, X=1/А}*

if M=0 then R:=l *{если число возводится в нулевую степень, то результат всегда ра- вен 1}*

else if M>0 then R:=Step(M,A)*{Вызов функции Step с передачей ей фактических пара- метров М, А}*

else R:=Step(-M,A); {*Вызов функции с передачей ей фактических параметров: - М, отрицательная степень}*

end.

Z:=Z/(2\*R) ;

Writeln(' Для А= ',А,'М= ',М,' Значение выражения= ',Z);

В начале программы описываются переменная целого типа *М* и переменные веще- ственного типа *А, Z, R*, после этого описывается функция вычисления степени числа *Step* с формальными параметрами *N* и *X,* результат, возвращаемый функцией в точку вызова, - ве- щественного типа.

В описании функции вводятся две *локальных (местных) переменных I* и *Y*. Перемен- ная *I* служит для подсчета числа повторений цикла, а в *Y* накапливается значение степени как произведения *N* одинаковых сомножителей. В заключение функции присваивается значение вычисленного произведения.

В начале выполнения основной программы на экран выводится запрос *"Введите зна- чение числа А и показатель степени М"* и считывается с клавиатуры значение вещественного числа А и целого числа М.

*Затем выполняется оператор: Z:=Step(5,A);*

Осуществляется вызов функции *Step* с передачей ей фактических параметров *5, А*. Их значения присваиваются формальным параметрам функции *N* и *X*. По окончании вычисления степени числа значение функции *Step*, вычисленное для фактических параметров *5* и *А*, при- сваивается переменной *Z*. Аналогично в операторе:

*Z := Z + Step(3,l/A);*

сначала осуществляется вызов функции *Step* с передачей ей фактических параметров *3, 1/A,* после чего значение переменной *Z* увеличивается на величину возвращенного в основ- ную программу результата вычисления функции *Step*.

Операторы:

*if M=0 then R:=1*

*else if M>0 then R:=Step(M,A) else R:=Step(- M,A);*

проверяют условия *М=0, М>0* и в зависимости от их соблюдения либо присваивает пере- менной *R* значение 1 *(при М=О)*, либо выполняет вызов функции *Step* для *фактических зна- чений М, А* или *-М, А*, а после вычисления значения функции *Step* присваивает его перемен- ной R.

Оператор:

*Z:=Z/(2\*R);*

выполняет вычисление значения выражения, а затем присваивает вычисленное значение пе- ременной *Z*.

В заключение программы стандартная процедура *Writeln* выводит на экран сообщение о результате вычислений степени *М* числа *А*.

***Практическое занятие №12***

**Наименование занятия: *Работа с файлами***

**Цель занятия:** Получить практические навыки в работе с файловым типом данных. Форми- ровать ОК-2, ОК-3, ОК-4, ОК-5, ОК-8, овладеть знаниями и умениями, необходимыми для освоения ПК-1.2, ПК-1.3, ПК-2.2, ПК-2.3

**Подготовка к занятию:** Повторить теоретический материал по теме «Файлы»

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:**

## ВАРИАНТ 1

* 1. Дан файл *f*, элементы которого являются действительными числами. Найти сумму и произведение элементов файла.
  2. Описать процедуру traingle(t), формирующую текстовый файл t из 9 строк, в первой из которых - одна литера '1', во второй - две литеры '2' ..., в девятой - девять литер '9'.

## ВАРИАНТ 2

1. Дан файл *f*, элементы которого являются действительными числами. Найти сумму квад- ратов элементов файла и последний элемент файла.
2. Описать процедуру printlines(t), печатающую построчно содержимое текстового файла t.

## ВАРИАНТ 3

1. Дан файл *f*, элементы которого являются действительными числами. Найти разность первого и последнего элементов файла.
2. Пусть текстовой файл *t* разбит на непустые строки. Посчитать число строк, кото- рые начинаются с буквы d.

## ВАРИАНТ 4

1. Дан файл *f*, элементы которого являются действительными числами. Найти наиболь- шее из значений модулей элементов с нечетными номерами.
2. Пусть текстовой файл t разбит на непустые строки. Посчитать число строк, кото- рые оканчиваются буквой z.

## ВАРИАНТ 5

1. Дан файл *f*, элементы которого являются целыми числами. Найти количество четных чисел среди элементов.
2. Дан файл, содержащий текст на русском языке. Выяснить, входит ли заданное слово в

указанный текст, и если да, то сколько раз.

## ВАРИАНТ 6

1. Дан файл *f*, элементы которого являются целыми числами. Получить в файле *g* все элементы файла *f*, делящиеся на 3 и не делящиеся на 7.
2. Дан файл, содержащий текст на русском языке и некоторая буква. Посчитать, сколько слов начинается с указанной буквы

## ВАРИАНТ 7

1. Дан файл *f*, элементы которого являются целыми числами. Записать в файл *g* все четные числа файла *f*, а в файл *h* - все нечетные. Порядок следования чисел сохраняется.
2. Дан текстовый файл. Удалить из него все лишние пробелы, оставив между словами не более одного пробела. Результат поместить в новый файл.

## ВАРИАНТ 8

1. Дан файл *f*, элементы которого являются целыми числами. Получить файл *g*, обра- зованный из файла *f* исключением повторных вхождений одного и того же числа.
2. Дан файл, содержащий текст, записанный строчными буквами. Получить в другом файле тот же текст, записанный заглавными буквами.

## ВАРИАНТ 9

1. Дан файл *f*, элементы которого являются целыми числами. Получить в файле *g* все элементы файла *f*, являющиеся четными числами.
2. Дан файл, содержащий текст, включающий русские и английские слова. Посчитать, каких букв в тексте больше – русских или латинских.

## ВАРИАНТ 10

1. Дан файл *f*, элементы которого являются действительными числами. Найти сумму наибольшего и наименьшего из значений элементов.
2. Дан файл, содержащий текст на русском языке. Определить, сколько раз встречает-

ся в нем самое длинное слово.

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Составить программы на языке Pascal;
3. Ввести программы в ЭВМ и откомпилировать их;
4. Выполнить программы и занести результат в отчет;
5. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (код программ, результат выполнения программ);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Как описывается файловая переменная?
2. Назначение процедуры Reset?
3. Какая процедура закрывает файл?
4. Назначение процедуры Rewrite?
5. Какие процедуры используются для переименования и удаления файлов?

# ПРИЛОЖЕНИЕ

**Файлы**

Условно файл в Паскале можно изобразить как некоторую ленту, у которой есть нача- ло, а конец не фиксируется. Элементы файла записываются на эту ленту последовательно друг за другом:



где F – имя файла, а F1, F2, F3, F4 – его элементы. Файл во многом напоминает магнитную ленту, начало которой заполнено записями, а конец пока свободен. В программировании су- ществует несколько разновидностей файлов, отличающихся методом доступа к его компо- нентам: файлы последовательного доступа и файлы произвольного доступа.

### Работа с файлами в Паскале

Любой файл имеет три характерные особенности. Во-первых, у него есть имя, что да- ет возможность программе работать одновременно с несколькими файлами. Во-вторых, он содержит компоненты одного типа. Типом компонентов может быть любой тип Паскаля, кроме файлов. Иными словами, нельзя создать «файл файлов». В-третьих, длина вновь со- здаваемого файла никак не оговаривается при его объявлении и ограничивается только емко- стью устройств внешней памяти.

Файловый тип или переменную файлового типа в Паскале можно задать одним из трех способов:

Type <имя\_ф\_типа>=file of<тип\_элементов>;

<имя\_ф\_типа>=text;

<имя\_ф\_типа>=file;

Здесь <имя\_ф\_типа> – имя файлового типа (правильный идентификатор); File, of – зарезер- вированные слова (файл, из); <тип\_элементов> – любой тип Паскаля, кроме файлов.

*Пример описания файлового типа в Паскале*

Type

Product= record Name: string; Code: word;

End;

Text80= file of string[80]; Var

F1: file of char;

F2: text;

F3: file;

F4: Text80;

F5: file of Product;

В зависимости от способа объявления можно выделить три вида файлов Паскаля:

* типизированные файлы Паскаля(задаются предложением file of..);
* текстовые файлы Паскаля(определяются типом text);
* нетипизированные файлы Паскаля(определяются типом file).

### Основные процедуры и функции для работы с файлами

1. До начала работы с файлами в Паскале необходимо установить связь между фай- ловой переменной и именем физического дискового файла:

Assign(<файловая\_переменная>, <имя\_дискового\_файла>)

Следует помнить, что имя дискового файла при необходимости должно содержать путь доступа к этому файлу, включая имя дисковода. При этом имя дискового файла – стро- ковая величина, т.е. должна быть заключена в апострофы. Например:

Assign (chf, 'G:\Home\ Student\ Lang\ Pascal\ primer.dat');

Если путь не указан, то программа будет искать файл в своем рабочем каталоге и по указанным путям в autoexec.bat.

1. После окончания работы с файлами на Паскале, они должны быть закрыты. Close(<список файловых переменных>);

При выполнении этой процедуры закрываются соответствующие физические файлы и фиксируются сделанные изменения.

1. Подготовка к записи в файл Паскаля Rewrite(<имя\_ф\_переменной>);

Процедура Rewrite(f) (где f – имя файловой переменной) устанавливает файл с именем f в начальное состояние режима записи, в результате чего указатель устанавливается на

первую позицию файла. Если ранее в этот файл были записаны какие-либо элементы, то они становятся недоступными.

1. Запись в файл Паскаля Write(<имя\_ф\_переменной>, <список записи>);

При выполнении процедуры write(f, x) в ту позицию, на которую показывает указа- тель, записывается очередная компонента, после чего указатель смещается на следующую позицию. Естественно, тип выражения х должен совпадать с типом компонент файла.

1. Подготовка файла к чтению Паскаля Reset(<имя\_ф\_переменной>);

Эта процедура ищет на диске уже существующий файл и переводит его в режим чте- ния, устанавливая указатель на первую позицию файла. Если происходит попытка открыть для чтения не существующий еще на диске файл, то возникает ошибка ввода/вывода, и вы- полнение программы будет прервано.

1. Чтение из файла в Паскале Read(<имя\_ф\_переменной>,<список переменных>);

Для типизированных файлов при выполнении процедуры read() последовательно счи- тывается, начиная с текущей позиции указателя, число компонент файла, соответствующее числу переменных в списке, а указатель смещается на это число позиций.

В большинстве задач, в которых используются файлы, необходимо последовательно перебрать компоненты и произвести их обработку. В таком случае необходимо иметь воз- можность определять, указывает ли указатель на какую-то компоненту файла, или он уже вышел за пределы файла и указывает на маркер конца файла.

1. Функция определения достижения конца файла в Паскале Eof(<имя\_ф\_переменной>);

Название этой функции является сложносокращенным словом от end of file. Значение этой функции имеет значение true, если конец файла уже достигнут, т.е. указатель стоит на позиции, следующей за последней компонентой файла. В противном случае значение функ- ции – false.

1. Изменение имени файла в Паскале Rename(<имя\_ф\_переменной>, <новое\_имя\_файла>);

Здесь новое\_ имя\_ файла – строковое выражение, содержащее новое имя файла, воз- можно с указанием пути доступа к нему.

Перед выполнением этой процедуры необходимо закрыть файл, если он ранее был от-

крыт.

крыт.

1. Уничтожение файла в Паскале Erase(<имя\_ф\_переменной>);

Перед выполнением этой процедуры необходимо закрыть файл, если он ранее был от-

1. Уничтожение части файла от текущей позиции указателя до конца в Паскале Truncate(<имя\_ф\_переменной>);
2. Файл Паскаля может быть открыт для добавления записей в конец файла Append(<имя\_ф\_переменной>);

### Типизированные файлы Паскаля

Длина любого компонента типизированного файла строго постоянна, т.к. тип компо- нент определяется при описании, а, следовательно, определяется объем памяти, отводимый под каждую компоненту. Это дает возможность организовать прямой доступ к каждой ком- поненте (т.е. доступ по порядковому номеру).

Перед первым обращением к процедурам ввода/вывода указатель файла стоит в его начале и указывает на его первый компонент с номером 0. После каждого чтения или записи указа- тель сдвигается к следующему компоненту файла. Переменные и выражения в списках ввода и вывода в процедурах read() и write() должны иметь тот же тип, что и компоненты файла

Паскаля. Если этих переменных или выражений в списке несколько, то указатель будет сме- щаться после каждой операции обмена данными на соответствующее число позиций.

Для облегчения перемещения указателя по файлу и доступа к компонентам типизированного файла существуют специальные процедуры и функции:

- функция Паскаля, определяющая число компонентов в файле: fileSize(<имя\_ф\_переменной>)

* функция Паскаля, значением которой является текущая позиция указателя: filePos(<имя\_ф\_переменной>)
* процедура Паскаля, смещающая указатель на компоненту файла с номером n: seek(<имя\_ф\_переменной>,n)

Так, процедура seek(<имя\_ф\_переменной>,0) установит указатель в начало файла, а процедура seek(<имя\_ф\_переменной>, FileSize(<имя\_ф\_переменной>)) установит указатель на признак конца файла.

### Текстовые файлы Паскаля

Текстовые файлы предназначены для хранения текстовой информации. Именно в та- ких файлах хранятся, например, исходные тексты программ. Компоненты текстовых файлов могут иметь переменную длину, что существенно влияет на характер работы с ними. Доступ к каждой строке текстового файла Паскаля возможен лишь последовательно, начиная с пер- вой. К текстовым файлам применимы процедуры assign, reset, rewrite, read, write и функция eof. Процедуры и функции seek, filepos, filesize к ним не применяются. При создании тексто- вого файла в конце каждой записи (строки) ставится специальный признак EOLN(end of line

* конец строки). Для определения достижения конца строки существует одноименная логи- ческая функция EOLN(<имя\_ф\_переменной>), которая принимает значение true, если конец строки достигнут.

Форма обращения к процедурам write и read для текстовых и типизированных файлов одинакова, но их использование принципиально различается.

Кроме процедур read и write при работе с текстовыми файлами используются их раз- новидности readln и writeln. Отличие заключается в том, что процедура writeln после записи заданного списка записывает в файл специальный маркер конца строки. Этот признак вос- принимается как переход к новой строке. Процедура readln после считывания заданного списка ищет в файле следующий признак конца строки и подготавливается к чтению с нача- ла следующей строки.

***Пример 1****. Создать простой текстовый файл с именем WORK.TXT:*

… Var

F: text; begin

Assign(F, ' WORK. ТХТ') ;

Rewrite(F) ;

Write(F, 'Простой текстовый файл'); Close(F) ;

end.

***Пример 2.*** *Пусть нам необходимо сформировать текстовый файл с помощью Паска- ля, а затем переписать из данного файла во второй только те строки, которые начинают- ся с буквы «А» или «а».*

Пояснения: нам понадобятся две файловые переменные f1 и f2, поскольку оба файла текстовые, то тип переменных будет text. Задача разбивается на два этапа: первый – форми- рование первого файла; второй – чтение первого файла и формирование второго.

Для завершенности решения задачи есть смысл добавить еще одну часть, которая в задаче явно не указана – вывод на экран содержимого второго файла.

Program primer; Var f1,f2:text; I,n: integer;

S: string; Begin

{формируем первый файл}

Assign(f1, ‘file1.txt’); {устанавливаем связь файловой переменной с физическим файлом на диске}

Rewrite(f1); {открываем файл для записи} Readln(n) {определим количество вводимых строк} for i:=1 to n do

begin

readln(s); {вводим с клавиатуры строки}

writeln(f1,s); {записываем последовательно строки в файл} end;

close(f1); {заканчиваем работу с первым файлом, теперь на диске существует файл с име- нем file1.txt, содержащий введенные нами строки. На этом программу можно закончить, ра- боту с файлом можно продолжить в другой программе, в другое время, но мы продолжим}

{часть вторая: чтение из первого файла и формирование второго} Reset(f1); {открываем первый файл для чтения}

Assign(f2, ‘file2.txt’); {устанавливаем связь второй файловой переменной с физическим файлом}

Rewrite(f2); {открываем второй файл для записи}

{Дальше необходимо последовательно считывать строки из первого файла, проверять вы- полнение условия и записывать нужные строки во второй файл. Для чтения из текстового файла рекомендуется использовать цикл по условию «пока не конец файла»}

While not eof(f1) do Begin

Readln(f1,s);{считываем очередную строку из первого файла} If (s[1]=’A’) or (s[1]=’a’) then

Writeln(f2,s); {записываем во второй файл строки, удовлетворяющие условию} End;

Close(f1,f2); {заканчиваем работу с файлами}

{часть третья: выводим на экран второй файл} Writeln;

Writeln(‘Второй файл содержит строки:’); Reset(f2); {открываем второй файл для чтения} While not eof(f2) do {пока не конец второго файла} Begin

Readln(f2,s);{считываем очередную строку из второго файла} Writeln(s); {выводим строку на экран}

End; End.

***Практическое занятие №13***

**Наименование занятия: *Динамические структуры данных***

**Цель занятия:** Научиться составлять программы с использованием динамических структур. Формировать ОК-2, ОК-3, ОК-4, ОК-5, ОК-6, ОК-7, ОК-8, овладеть знаниями и умениями, необходимыми для освоения ПК-2.2, ПК-2.3

**Подготовка к занятию:** Повторить теоретический материал по теме «Указатели и динами- ческие структуры».

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:**

## ВАРИАНТ 1

Составить программу, которая вставляет в список *L* новый элемент *F* за каждым вхождением элемента *E*.

## ВАРИАНТ 2

Составить программу, которая вставляет в список *L* новый элемент *F* перед первым вхождением элемента *E*, если *E* входит в *L*.

## ВАРИАНТ 3

Составить программу, которая удаляет из списка *L* все элементы *E*, если таковые имеются.

## ВАРИАНТ 4

Составить программу, которая удаляет из списка *L* все отрицательные элементы.

## ВАРИАНТ 5

Составить программу, которая проверяет, есть ли в списке *L* хотя бы два одинаковых элемента.

## ВАРИАНТ 6

Составить программу, которая вставляет в список *L* за первым вхождением элемента

*E* все элементы списка *L*, если *E* входит в *L.*

## ВАРИАНТ 7

Составить программу, которая формирует список *L,* включив в него по одному разу элементы, которые входят одновременно в оба списка *L*1 и *L*2*.*

## ВАРИАНТ 8

Составить программу, удаляет из списка *L* все положительные элементы.

## ВАРИАНТ 9

Составить программу, удаляет из списка натуральных чисел все элементы, кратные заданному числу *k*.

## ВАРИАНТ 10

Составить программу подсчета различных элементов списка целых чисел.

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Составить программу на языке Pascal;
3. Ввести программу в ЭВМ и откомпилировать её;
4. Выполнить программу и занести результат в отчет;
5. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (код программы, результат выполнения программы);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Что называется динамической величиной?
2. Что называется динамической памятью?
3. Что называется указателем?
4. Что такое связанный список?

# ПРИЛОЖЕНИЕ

### Динамические структуры

#### Линейные списки (однонаправленные цепочки)

***Списком*** называется структура данных, каждый элемент которой посредством указате- ля связывается со следующим элементом.

Каждый элемент связанного списка, во-первых, хранит какую-либо информацию, во- вторых, указывает на следующий за ним элемент. Так как элемент списка хранит разнотип- ные части (хранимая информация и указатель), то его естественно представить записью, в которой в одном поле располагается объект, а в другом – указатель на следующую запись такого же типа. Такая запись называется *звеном*, а структура из таких записей называется *списком или цепочкой*.

Лишь на самый первый элемент списка (голову) имеется отдельный указатель. Послед- ний элемент списка никуда не указывает.

#### Описание списка

Type ukazat= ^S;

S= record

Inf: integer;

Next: ukazat;

End;

В Паскале существует основное правило: перед использованием какого-либо объекта он должен быть описан. Исключение сделано лишь для указателей, которые могут ссылаться на еще не объявленный тип.

#### Формирование списка

Чтобы список существовал, надо определить указатель на его начало. Type ukazat= ^S;

S= record

Inf: integer;

Next: ukazat;

End; Var u,x: ukazat;

Создадим первый элемент списка:

New (u); {выделяем место в памяти} u^. Next:= nil; {указатель пуст} u^.Inf:=3;

Продолжим формирование списка. Для этого нужно добавить элемент либо в конец списка, либо в голову.

А) *Добавим элемент в голову списка*. Для этого необходимо выполнить последователь- ность действий:

* 1. получить память для нового элемента;
  2. поместить туда информацию;
  3. присоединить элемент к голове списка.

New(x); Readln(x^.Inf); x^.Next:=u; u:=x;

Б) *Добавление элемента в конец списка*. Для этого введем вспомогательную перемен- ную, которая будет хранить адрес последнего элемента. Пусть это будет указатель с именем hv (хвост).

x:= hv;

New(x^.next); {выделяем память для следующего элемента} x:= x^.next;

x^.next:= nil; x^.inf:= 5; hv:=x;

#### Просмотр списка

While u<> nil do Begin

Writeln (u^.inf); u:= u^.next; end;

#### Удаление элемента из списка

А) *Удаление первого элемента*. Для этого во вспомогательном указателе запомним пер- вый элемент, указатель на голову списка переключим на следующий элемент списка и осво- бодим область динамической памяти, на которую указывает вспомогательный указатель.

x:= u;

u:= u^.next; dispose(x);

Б) *Удаление элемента из середины списка*. Для этого нужно знать адреса удаляемого элемента и элемента, стоящего перед ним. Допустим, что digit – это значение удаляемого элемента.

x:= u;

while (x<> nil) and (x^.inf<> digit) do begin

dx:= x;

x:= x^.next; end;

dx:= x^.next:

dispose(x);

В) *Удаление из конца списка*. Для этого нужно найти предпоследний элемент. x:= u; dx:= u;

while x^.next<> nil do begin

dx:= x; x:= x^.next; end;

dx^.next:= nil; dispose(x);

#### Прохождение списка

Важно уметь перебирать элементы списка, выполняя над ними какую-либо операцию.

Пусть необходимо найти сумму элементов списка. summa:= 0;

x:= u;

while x<> nil do begin

summa:= summa+ x^.inf; x:= x^.next;

end;

### Пример программы создания и использования связанного списка

Пусть требуется создать связанный список из записей, содержащих сведения об авто- мобилях, а также реализовать операции со связанным списком: запись первым в список, удаление первого объекта из списка, просмотр всего списка, удаление объекта, следующего за указанным.

*{Пример использования указателей для обработки связанного списка}*

Program point;

Uses Crt;

Type

NameStr = String [20]; Link = ^Auto;

Auto = record

Name : NameStr; *{Марка автомобиля}*

Speed : real; *{Скорость}*

Next : Link; *{Поле для связи со следующим объектом в списке}*

end;

Var

Р,First : Link; *{Указатели на запись: текущую, первую}*

NamFind : NameStr; *{Марка автомобиля для поиска}*

V : 0..4; *{Селектор меню}*

EndMenu : boolean; *{Окончание вывода меню}*

*{Поиск объекта с именем FN, по результатам поиска возвращает указатель на найденный объект или Nil, если объект не найден}*

Function FindName(FN:NameStr) : Link; Var

Curr : Link; begin

Curr:=First; *{Установить указатель на первом объекте в списке }*

*{Повторять пока не дойдем до конца списка}*

while Curr <> Nil do

if Curr^.Name=FN then *{Если нашли заданный объект}*

begin

FindName:=Curr; *{Возвращаем значение указателя на него}*

Exit; *{Завершаем функцию}*

end

else

Curr:=Curr^ .Next; *{Перейти к следующей записи}*

FindName:=Nil; *{В списке нет искомого объекта}*

end; *{Конец FindName}*

*{Добавление записи первой в связанный список}*

procedure AddFirst(A:Link); begin

A^. Next:=First; *{Новый объект первый в списке}*

First:=А; *{Голова списка ссылается на новый объект}*

end; *{Конец AddFirst}*

*{Удаление первого объекта из списка}*

procedure DelFirst(var A:Link); begin

A:=First;

First:=First^. Next; *{Теперь First указывает на тот объект, на который ранее ссылал- ся объект А}*

end; *{Конец DelFirst}*

*{Удаление из списка объекта, стоящего после объекта Old}*

procedure DelAfter(Old:Link; var A:Link); begin

A:=Old^.Next; *{Переменной А присваивается значение указателя на удаляемый объект}*

Old^.Next:=Old^.Next^.Next; *{Теперь Old указывает на тот объект, на который ранее ссылался следующий за ним объект, а объект А исключен из связанного списка}*

end; *{Конец DelAfter}*

*{Ввести данные об объекте}*

procedure InpAvto; begin

P:=New(Link); *{Создать очередной объект типа Auto}*

Write('Введите марку автомобиля :'); Readln(P^.Name) ; Write('Максимальная скорость :'); Readln(Р^.Speed);

AddFirst(Р); *{Вызов процедуры добавления записи, на которую ссылается указа- тель Р (Р- фактический параметр, А - формальный параметр-значение) }*

end; *{Конец InpAvto}*

*{Вывести на экран все объекты из связанного списка}*

procedure MyList; var

Curr : Link; *{Локальная переменная - указатель на очередной объект}*

begin

Curr:=First; *{Установить указатель на первом объекте в списке}*

*{Повторять, пока не дойдем до конца списка}*

while Curr <> Nil do begin

Writeln( 'Марка : ' , Curr^. Name,' скорость : ', Curr^. Speed) ; Curr:=Curr^.Next; *{Перейти к очередному объекту связанного списка}* end ;

Write('Вывод списка окончен. Нажмите Enter'); Readln;

end; *{Конец MyList}*

Begin *{Основная программа}*

New(P); *{Создать новую динамическую переменную и установить на нее перемен- ную-указатель}*

EndMenu:=False ;

repeat *{Очищать экран и выводить меню до тех пор, пока EndMenu<>True}*

CIrScr;

Writeln('Укажите вид работы:'); Writein('1. Запись первым в список');

Writeln('2. Удаление первого объекта из списка'); Writein('3. Просмотр всего списка') ;

Writein('4. Удаление объекта, следующего в списке за указанным') ; Writein('0. Окончание работы');

Readin(V) ;

Case V of *{Вызов разных процедур в зависимости от выбора пункта меню}*

1. : InpAvto; *{Ввод данных об объекте}*
2. : DelFirst(P); *{Удаление первого в списке}*
3. : MyList; *{Вывод списка всех элементов связанного списка}*
4. : begin *{Удаление объекта, следующего за указанным}*

Write('Введите марку автомобиля, за которым следует удаляемый из списка :'); Readln(NamFind) ;

DelAfter(FindHame(NamFind),P); *{Вызов процедуры*

*DelAfter с параметрами: функцией FindName(NamFind) и указателем Р}*

end

else

EndMenu:=True; *{Завершить вывод меню}*

end;

end.

until EndMenu; *{Если EndMenu=True, то завершить вывод меню на экран}*

Dispose(Р); *{Уничтожить динамическую переменную Р и освободить память в куче}*

***Практическое занятие №14***

**Наименование занятия: *Работа в графическом режиме***

**Цель занятия:** Получить практические навыки работы в графическом режиме. Формировать ОК-2, ОК-3, ОК-4, ОК-5, ОК-8, овладеть знаниями и умениями, необходимыми для освоения ПК-1.3, ПК-2.2

**Подготовка к занятию:** Повторить теоретический материал по теме «Модули».

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:**

## ВАРИАНТ 1

Написать программу, которая рисует на экране снеговика.

## ВАРИАНТ 2

Написать программу, которая рисует на экране бабочку.

## ВАРИАНТ 3

Написать программу, которая выводит на экран автомобиль.

## ВАРИАНТ 4

Написать программу, которая рисует на экране новогоднюю елку.

## ВАРИАНТ 5

Написать программу, которая рисует на экране вазу с цветами.

## ВАРИАНТ 6

Написать программу, которая рисует на экране паровоз.

## ВАРИАНТ 7

Написать программу, которая рисует на экране домик.

## ВАРИАНТ 8

Написать программу, которая выводит на экран флаг олимпийских игр.

## ВАРИАНТ 9

Написать программу, которая рисует на экране кораблик.

## ВАРИАНТ 10

Написать программу, которая рисует на экране гусеницу.

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Составить программу на языке Pascal;
3. Ввести программу в ЭВМ и откомпилировать её;
4. Выполнить программу и занести результат в отчет;
5. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (код программы, результат выполнения программы);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Как запустить графическую систему?
2. Как вывести точку на экран?
3. Как вывести линию на экран?
4. Как вывести текст на экран?
5. Как установить цвет?

# ПРИЛОЖЕНИЕ

### Графика на Паскале

Запуск и завершение работы в графической системе осуществляется следующим об-

разом:

1. Подключить модуль Graph (библиотеку графических процедур): **uses Graph;**
2. Установить графический режим:

* описываем переменные, которые определяют графический драйвер (gd) и монитор(gm):

### var gd, gm: integer;

* задаем команду ПК для самовыбора значений переменных:

**gd:=Detect;** (значение gm после команды gd:=detect; определяется автоматически)

* инициализируем графический режим:

### InitGraph( gd, gm,’ указывается путь к драйверу, чем подробнее, тем лучше’).

С этого момента все графические средства доступны пользователю.

1. Завершить работу в графической системе: **CloseGraph;**

Например,

**Uses graph; {**подключение графического модуля}

### Var gd,gm:integer;

**Begin**

**gd:=detect; {**определение графического драйвера**} InitGraph(gd,gm,’ ‘); {**инициализация графики**}**

**. . . {** текст программы**} CloseGraph;** {закрытие графики}

### End.

**Базовые процедуры и функции**

Для построения изображений на экране используется система координат. Отсчет начинается от верхнего левого угла экрана, который имеет координаты (0,0). Значение Х (столбец) увеличивается слева направо, значение Y (строка) увеличивается сверху вниз. Что- бы строить изображения, необходимо указывать точку начала вывода. В текстовых режимах эту точку указывает курсор, который присутствует на экране. В графических режимах види- мого курсора нет, но есть невидимый текущий указатель CP (Current Pointer). Фактически это тот же курсор, но он невидим.

**Процедуры модуля *Graph***

|  |  |  |
| --- | --- | --- |
| **Процедура** | **Формат** | **Действие** |
| SetColor | SetColor (a: word); | Устанавливает цвет, которым будет осу-  ществляться рисование |
| SetBkColor | SetBkColor (a: word); | Устанавливает цвет фона |
| SetFillStyle | SetFillStyle (a,b: word);  a – стиль закраски, b – цвет | Устанавливает стиль и цвет закраски |
| SetLineStyle | SetLineStyle (a,b,c: word);  а – стиль линии, b- образец построения линии (может  устанавливаться пользова- телем), с-толщина линии | Устанавливает стиль и толщину линии |

|  |  |  |
| --- | --- | --- |
| SetTextStyle | SetTextStyle (a,b,c: word); | Устанавливает шрифт, стиль и размер  текста |
| SetFillPattern | SetFillPattern (Pattern: Fill- patternType; Color:word);  Pattern- маска | Выбирает шаблон заполнения, опреде- ленный пользователем |
| ClearDivice | ClearDivice | Очищает экран и устанавливает текущий  указатель в начало |
| SetViewPort | SetViewPort (x1, y1 , x2, y2 : integer, Clip:boolean); | Устанавливает текущее окно для графи-  ческого вывода |
| ClearViewPort | ClearViewPort | Очищает окно |
| PutPixel | PutPixel (a,b,c :integer); | Рисует точку цветом с в (x,y) |
| Line | Line(x1, y1 , x2,y2 :integer); | Рисует линию от (x1, y1) к (x2,y2) |
| Rectangle | Rectangle (x1, y1 , x2, y2:integer ); | Рисует прямоугольник с диагональю от  (x1, y1) к (x2, y2) |
| Bar | Bar (x1, y1 , x2, y2:integer); | Рисует закрашенный прямоугольник |
| Bar3D | Bar3D (x1, y1 , x2,y2, d:integer, a:boolean); | Рисует трехмерную полосу (параллеле-  пипед) |
| Circle | Circle (x,y,r: word); | Рисует окружность радиуса r с центром в  точке (x, y) |
| Arc | , - начальный и конеч- ный углы в градусах | Рисует дугу из начального угла к конеч- ному, используя (x,y) как центр |
| Ellipse | Elli x, Ry:  integer);  , - начальный и конеч- ный углы в градусах | Рисует эллиптическую дугу от начально- го угла к конечному, используя (x, y) как центр |
| FillEllipse | FillEllipse (x, y, Rx, Ry:integer);  Rx, Ry – вертикальная и го- ризонтальная оси | Рисует закрашенный эллипс |
| MoveTo | MoveTo (x, y:integer); | Передвигает текущий указатель в (x, y) |
| MoveRel | MoveRel(x, y : integer); | Передвигает текущий указатель на за- данное расстояние от текущей позиции  на x по горизонтали и на y по вертикали |
| OutText | OutText (text: string); | Выводит текст от текущего указателя |
| OutTextxy | OutTextxy(x, y: integer, text:  string); | Выводит текст из (x, y) |
| Sector | x, Ry: in-  teger);  , - начальный и конеч- ный углы в градусах | Рисует и заполняет сектор эллипса |

### Функции модуля Graph

|  |  |
| --- | --- |
| GetBkColor | Возвращает текущий фоновый цвет |
| GetColor | Возвращает текущий цвет |
| GetX | Возвращает координату X текущей позиции |
| GetY | Возвращает координату Y текущей позиции |
| GetMaxX | Возвращает максимальное значение по оси X |
| GetMaxY | Возвращает максимальное значение по оси Y |
| GetPixel | Возвращает цвет точки в (x, y) |

**Цветовая шкала**

|  |  |  |  |
| --- | --- | --- | --- |
| Цвет | Код | Цвет | Код |
| Black – черный | 0 | DarkGray – темно-серый | 8 |
| Blue – синий | 1 | LightBlue – голубой | 9 |
| Green - зеленый | 2 | LightGreen – ярко-зеленый | 10 |
| Gyan – бирюзовый | 3 | LightGyan – ярко-бирюзовый | 11 |
| Red – красный | 4 | LightRed – ярко-красный | 12 |
| Magenta – малиновый | 5 | LightMagenta – ярко-малиновый | 13 |
| Brown – коричневый | 6 | Yellow – желтый | 14 |
| LightGray – светло-серый | 7 | White – белый | 15 |

***Пример 1.*** *Написать программу, которая вычерчивает треугольник красной линией в центре экрана.*

Program treug;

uses graph; { подключение библиотеки графических процедур}

var gd,gm: integer; {описание переменных, определяющих графический драйвер и монитор} begin

gd:=detect; {определение значений переменных по выбору ПК} initgraph(gd, gm, ‘ c/bp’); {инициализация графического режима} SetColor(4); {задание цвета линии}

SetLineStyle(1,0,3); {задание стиля линии} Line(320, 240, 320, 180);

Line(320, 240, 390, 240);

Line(390, 240, 320, 180);

end.

***Пример 2***. *Изобразить в левой верхней части экрана четырехугольную звезду зелено- го цвета*

program g; uses crt, graph; const

Star: array[1..18] of integer = (75, 0, 100, 50, 150, 75, 100, 100, 75, 150, 50, 100, 0, 75, 50, 50, 75,

0);

var

gd, gm: integer; begin

gd:= detect;

initgraph(gd, gm, ‘ c/bp’); SetFillStyle(1,2);

FillPoly(9,Star); {9 – количество пересечений + 1} CloseGraph;

end.

### Построение дуг и окружностей

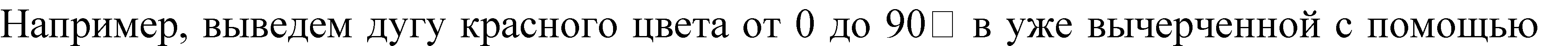
Процедура вычерчивания окружности текущим цветом имеет следующий формат: Cicrle(x, y, r: word), где x, y – координаты центра окружности, r – ее радиус.

Например, фрагмент программы обеспечит вывод ярко-зеленой окружности с радиу- сом 50 пикселей и центром в точке (450, 100):

SetColor(LightGreen); Circle(450, 100, 50);

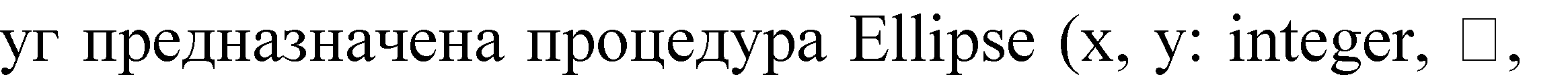
Дуги можно вычертить с помощью процедуры Arc(x, y: integer, , , R:integer), где x, y- центр окружности, , - начальный и конечный углы в градусах, R – радиус. Для задания углов используется полярная система координат.



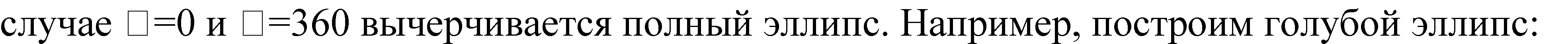


Circle(450, 100, 50) окружности:

SetColor(Red); Arc(450, 100, 0, 90,50);

Для построения эллиптических д

В



x, Ry: integer), где x, y – центр эллипса, Rx, Ry: горизонтальная и вертикальная оси.

SetColor(9);

Ellipse(100, 100, 0, 360, 50,50);

Фон внутри эллипса совпадает с фоном экрана. Чтобы создать закрашенный эллипс, используется специальная процедура FillEllipse (x, y: integer, Rx, Ry: integer). Закраска эллип- са осуществляется с помощью процедуры SetFillStyle (a, b: word), где а – стиль закраски, b – цвет закраски.

Например, нарисуем ярко-красный эллипс, заполненный редкими точками зеленого

цвета:

SetFillStyle(WideDotFill, Green); { установка стиля заполнения} SetColor(12); {цвет вычерчивания эллипса}

FillEllipse(300, 150, 50, 50);

**Стандартные стили заполнения**

|  |  |  |
| --- | --- | --- |
| Константа | Значение | Маска |
| EmptyFill | 0 | Заполнение цветом фона |
| SolidFill | 1 | Заполнение текущим цветом |
| LineFill | 2 | Заполнение символами --, цвет – color |
| LtslashFill | 3 | Заполнение символами // нормальной толщины,  цвет – color |
| SlashFill | 4 | Заполнение символами // удвоенной толщины,  цвет – color |
| BkslashFill | 5 | Заполнение символами \\ удвоенной толщины,  цвет – color |
| LtbkSlahFill | 6 | Заполнение символами \\ нормальной толщины,  цвет – color |
| HatchFill | 7 | Заполнение вертикально-горизонтальной штри-  ховкой тонкими линиями, цвет – color |
| XhatchFill | 8 | Заполнение штриховкой крест-накрест по диа-  гонали «редкими» тонкими линиями, цвет – col- or |
| InterLeaveFill | 9 | Заполнение штриховкой крест-накрест по диа- гонали «частыми» тонкими линиями, цвет – col-  or |
| WideDotFill | 10 | Заполнение «редкими» точками |
| CloseDotFill | 11 | Заполнение «частыми» точками |
| UserFill | 12 | Заполнение по определенной пользователем  маске заполнения, цвет – color |

***Практическое занятие №15***

**Наименование занятия: *Составление и отладка программ с использованием модулей* Цель занятия:** Получить практические навыки в составлении и отладке программ с исполь- зованием модулей на языке Pascal. Формировать ОК-2, ОК-3, ОК-4, ОК-5, ОК-8, овладеть знаниями и умениями, необходимыми для освоения ПК-1.3, ПК-2.2

**Подготовка к занятию:** Повторить теоретический материал по теме «Модули»

### Литература:

1. Лобачева М.Е. Основы алгоритмизации и программирования. Учебное пособие, 2015

**Задание на занятие:**

## ВАРИАНТ 1

Напишите программу, используя модуль, которая обнуляет из одномерного массива целых чисел наибольший элемент.

## ВАРИАНТ 2

Напишите программу, используя модуль, которая в одномерном массиве, состоящем из обыкновенных дробей, находит сумму всех элементов массива. Результат представить в виде несократимой дроби.

## ВАРИАНТ 3

Напишите программу, используя модуль, которая обнуляет из одномерного массива целых чисел наименьший элемент.

## ВАРИАНТ 4

Напишите программу, используя модуль, которая в одномерном массиве, состоящем из обыкновенных дробей, находит произведение всех элементов массива. Результат предста- вить в виде несократимой дроби.

## ВАРИАНТ 5

Напишите программу, используя модуль, которая обнуляет из одномерного массива целых чисел все отрицательные элементы.

## ВАРИАНТ 6

Напишите программу, используя модуль, которая в одномерном массиве, состоящем из обыкновенных дробей, находит среднее арифметическое всех элементов массива. Резуль- тат представить в виде несократимой дроби.

## ВАРИАНТ 7

Напишите программу, используя модуль, которая обнуляет из одномерного массива целых чисел все положительные элементы.

## ВАРИАНТ 8

Напишите программу, используя модуль, которая в одномерном массиве, состоящем из обыкновенных дробей, находит сумму элементов массива, стоящих на четных местах.

## ВАРИАНТ 9

Напишите программу, используя модуль, которая обнуляет из одномерного массива целых чисел все двузначные элементы.

## ВАРИАНТ 10

Напишите программу, используя модуль, которая в одномерном массиве, состоящем из обыкновенных дробей, находит произведение элементов массива, стоящих на нечетных местах.

### Порядок проведения занятия:

1. Получить допуск к работе;
2. Составить программу на языке Pascal;
3. Ввести программу в ЭВМ и откомпилировать её;
4. Выполнить программу и занести результат в отчет;
5. Оформить отчет.

### Содержание отчета:

1. Наименование, цель занятия, задание;
2. Выполненное задание (код программы, результат выполнения программы);
3. Ответы на контрольные вопросы;
4. Вывод о проделанной работе.

### Контрольные вопросы для зачета:

1. Что такое модуль?
2. Из каких частей состоит модуль?
3. Какие разделы могут быть включены в исполнительную часть модуля?
4. Как объявить модуль в программе?

# ПРИЛОЖЕНИЕ

**Модули**

Модуль — это набор ресурсов (функций, процедур, констант, переменных, типов и т.д.), разрабатываемых и хранимых независимо от использующих их программ. В отличие от внешних подпрограмм модуль может содержать достаточно большой набор процедур и функций, а также других ресурсов для разработки программ. Обычно каждый модуль содер- жит логически связанные между собой программные ресурсы.

В основе идеи модульности лежат принципы структурного программирования. Суще- ствуют стандартные модули Turbo Pascal, которые обычно описываются в литературе по данному языку.

Модуль имеет следующую структуру:

Unit <имя модуля>; {заголовок модуля} Interface

{интерфейсная часть} Implementation

{раздел реализации} Begin

{раздел инициализации модуля} End.

После служебного слова Unit записывается имя модуля, которое (для удобства даль- нейших действий) должно совпадать с именем файла, содержащего данный модуль. Поэтому (как принято в MS DOS) имя не должно содержать более 8 символов.

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

В разделе Implementation реализуются все подпрограммы, которые были ранее объяв- лены. Кроме того, здесь могут содержаться свои константы, переменные, типы, подпрограм- мы и т.д., которые носят вспомогательный характер и используются для написания основных подпрограмм. В отличие от ресурсов, объявленных в разделе Interface, все, что дополнитель- но объявляется в Implementation, уже не будет доступно при подключении модуля. При написании основных подпрограмм достаточно указать их имя (т.е. не нужно полностью пе- реписывать весь заголовок), а затем записать тело подпрограммы.

Раздел инициализации (который часто отсутствует) содержит операторы, которые должны быть выполнены сразу же после запуска программы, использующей модуль.

***Пример1.*** *Модуль для работы с одномерными массивами до 100 целых чисел.*

{модуль описаний, глобальных для основной программы и всех модулей}

**Unit** Globals;

### Interface

**const** Len=100;

**type** Vector = **array**[1..Len] **of** integer;

### Implementation End.

**Unit** Vectors;

### Interface

**uses** Globals;

{находит максимальный элемент массива}

**function** Max\_V(A:Vector; n:byte):integer;

{поэлементное сложение двух векторов}

**procedure** Add\_V(A,B:Vector; n:byte; **var** C:Vector);

{скалярное произведение векторов}

**function** Scal\_V(A,B:Vector; n:byte):integer;

### Implementation

**function** Max\_V; {заголовок без параметров}

**var** i,max:integer;

### begin

max:=A[1];

**for** i:=2 **to** n **do if** A[i]>max **then** max:=A[i]; Max\_V:=max;

**end**;

**procedure** Add\_V; **var** i:integer; **begin**

**for** i:=1 **to** n **do** C[i]:=A[i]+B[i];

**end**;

**function** Scal\_V(A,B:Vector; n:byte):integer;

{заголовок из interface} **var** s:integer; i:byte; **begin**

s:=0;

**for** i:=1 **to** n **do** s:=s+A[i]\*B[i]; Scal\_V:=s;

**end**;

**End**. {раздел инициализации модуля отсутствует}

***Пример 2*.** *Разработать личную библиотеку, включив в нее процедуры:*

* *ввода элементов числовой матрицы размером N\*N;*
* *транспонирования матрицы;*
* *вывода результирующей матрицы.*

*В основной программе ввести размер матрицы N.*

Начнем разработку модуля, который будет носить название Matrix. Программно это будет выглядеть так:

**Unit** Matrix;

{Зарезервированное слово *Unit* служит для указания имени библиотеки. Это имя должно совпадать с именем PAS-файла библиотеки (т.е библиотека Matrix должна находить- ся с файле *Matrix.Pas*), а иначе компилятор даст ошибку при попытке использования такой библиотеки}

### Interface

{Секция *Interface* содержит описания общедоступных типов данных, констант, проце- дур и функций. Т.е. все, что будет здесь находиться можно будет использовать при подклю- чении данной библиотеки.}

### Type

TMatrix = array [1..10,1..10] of Integer; { Квадратная матрица } procedure MatrInput (Var m : TMatrix; n : Integer); { ввод матрицы } procedure MatrOutput (Var m : TMatrix; n : Integer); { вывод матрицы } procedure MatrTransp (Var m : TMatrix; n : Integer); { транспонирование }

### Implementation

{Секция *Implementation* содержит реализацию тел процедур и функций, описанных в *Interface*. Также здесь могут содержаться типы данных, константы, процедуры и функции, необходимые для работы, но которые не будут видны программе при подключении библио- теки.}

{Процедура обмена местами двух элементов матрицы (x1,y1) и (x2,y2).

Эта процедура используется при транспонировании матрицы, но ее нельзя вызвать при подключении библиотеки, т.к. она не объявлена в секции *Interface*.}

procedure Swap (Var m : TMatrix; x1,y1,x2,y2 : Integer); var

temp : Integer; begin

temp := m[x1,y1]; m[x1,y1] := m[x2,y2]; m[x2,y2] := temp;

end;

{Ввод матрицы с клавиатуры. Параметры процедуры здесь не указаны, т.к. они есть в секции Interface }

procedure MatrInput; var

i,j : Integer; begin

for i:=1 to n do begin

Write(i:3,'-я строка : ');

for j:=1 to n do Read(m[i,j]); ReadLn;

end; end;

{Транспонирование матрицы.} procedure MatrTransp;

var

i,j : Integer; begin

for i:=1 to n-1 do for j:=i+1 to n do Swap (m,i,j,j,i);

end;

{Вывод матрицы на экран.} procedure MatrOutput;

var

i,j : Integer;

begin

for i:=1 to n do begin

Write(i:3,'-я строка : ');

for j:=1 to n do Write (m[i,j]:4); WriteLn;

end; end;

{Эта секция может использоваться для инициализации работы библиотеки.} Begin

End.

Создание модуля закончено.

Теперь необходимо создать файл, который будет содержать текст основной программы, в которой будет подключаться разработанный выше модуль.

{Это отдельный файл, содержащий основную программу} Uses

Crt, { Библиотека стандартных процедур управления экраном и клавиатурой }

Matrix; {Наш разработанный модуль-библиотека работы с квадратными матрицами (личная)}

Var

m : TMatrix; { Объявляем матрицу - максимальный размер 10\*10 }

n : Integer; { Размер матрицы } Begin

{ Повторяем ввод размера, пока не будет введено корректное значение} repeat

ClrScr;

Write('Введите размер матрицы (1..10) : '); ReadLn(n);

until (n >= 1) and (n <= 10); WriteLn;

WriteLn('Введите матрицу размера',n,'\*',n,'по строкам:');

MatrInput (m,n); {вызов процедуры ввода матрицы, определенной в модуле Matrix}

{Транспонируем ее }

MatrTransp (m,n); {вызов процедуры транспонирования матрицы, определенной в мо- дуле Matrix}

{ Выведем результат на экран } WriteLn;

WriteLn('Транспонированная матрица:');

MatrOutput (m,n); {вызов процедуры вывода матрицы, определенной в модуле Matrix} End.

Все модули хранятся в файлах с расширением TPU, имена которых совпадают с име- нами модулей. Для создания на диске файла с именем Display.tpu нужно выполнить компи- ляцию этого модуля (клавиши Alt+F9), предварительно установив опцию Compile/Distination/Disk.