Владимирский государственный университет

С. Б. НАУМОВА

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PascalABC

Учебно-практическое пособие

Владимир 2022

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

С. Б. НАУМОВА

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PascalABC

Учебно-практическое пособие

Электронное издание

Владимир 2022

ISBN 978-5-9984-1629-3

УДК 004
ББК 16.23

Рецензенты:

Кандидат физико-математических наук
доцент кафедры функционального анализа и его приложений
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
О. В. Крашенинникова

Кандидат физико-математических наук
доцент кафедры информационных технологий Российской академии
народного хозяйства и государственной службы
при Президенте Российской Федерации (Владимирский филиал)
А. А. Жукова

Наумова, С. Б.

Программирование на языке PascalABC [Электронный ресурс] : учеб.-практ. пособие / С. Б. Наумова ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2022. – 286 с. – ISBN 978-5-9984-1629-3. – Электрон. дан. (5,83 Мб). – 1 электрон. опт. диск (CD-ROM). – Систем. требования: Intel от 1,3 ГГц; Windows XP/7/8/10; Adobe Reader; дисковод CD-ROM. – Загл. с титул. экрана.

Доступно и всесторонне рассмотрены ключевые операторы языка программирования PascalABC, основные понятия, приведены многочисленные примеры программ. Предложены способы решения типовых заданий, разнообразные задания для формирования навыка программирования и формирования алгоритмического стиля мышления, а также задания для самостоятельной работы студентов.

Предназначено для студентов 1 – 2-го курсов высших учебных заведений, обучающихся по направлению подготовки 44.03.05 – Педагогическое образование, а также может быть полезно широкому кругу учащихся, абитуриентов, студентов педагогических вузов, учителей.

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Ил. 57. Библиогр.: 11 назв.

УДК 004
ББК 16.23

ISBN 978-5-9984-1629-3

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Алгоритмизация и программирование – один из самых важнейших разделов информатики и ИКТ. Этот раздел находит применение в самых различных областях науки.

В учебно-практическом пособии изложены теоретические и практические основы программирования на языке PascalABC. Весь материал пособия разбит на разделы и подразделы, в которых приведены основные теоретические сведения (определения, синтаксис, блок-схемы и формат записи основных операторов языка), примеры и задачи с подробными решениями, а также варианты для самостоятельного решения типовых задач. Такое изложение материала позволит студентам, изучающим программирование, овладеть стандартными приёмами и навыками и впоследствии творчески применять их в решении сложных задач.

Первая глава посвящена понятию алгоритма и его свойствам. Рассмотрены различные способы записи алгоритмов.

Во второй главе рассматривается структура программы в языке PascalABC, описываются типы данных.

В третьей главе начинается изучение первых операторов: ввода и вывода (рассматривается также форматный вывод данных), оператора присваивания. Студенты учатся составлять самые простые линейные алгоритмы и записывать их на языке программирования.

В четвертой главе рассматриваются оператор безусловного перехода, команды ветвления с различными логическими выражениями, а также оператор выбора.

Пятая глава посвящена изучению циклических алгоритмов, а именно циклов с предусловием и постусловием, приводятся примеры и решения типовых задач.

В шестой главе рассмотрен циклический оператор с параметром.

Седьмая глава знакомит обучающихся с линейными массивами, рассматриваются способы ввода и вывода элементов массива. Приводится формула для случайного генерирования чисел.

В восьмой главе собраны несколько наиболее распространенных алгоритмов сортировок элементов массивов: простой выбор, простой обмен, простые вставки, гномья сортировка, сортировка Шелла и шейкерная сортировка. Приводятся сами алгоритмы, а также программы.

Девятая глава посвящена алгоритмам поиска в массиве. Рассматриваются алгоритм простого перебора и метод бинарного поиска.

В десятой главе изучаются двумерные массивы, рассматриваются способы ввода и вывода элементов массива, печать такого массива в виде матрицы.

Одиннадцатая глава посвящена организации подпрограмм в языке PascalABC: функций и процедур.

В двенадцатой главе рассказывается о рекурсивных подпрограммах.

В тринадцатой главе речь о символьном типе данных. Описываются функции для работы с символьным типом.

Четырнадцатая глава посвящена строковому типу данных и функциям для работы с этим типом.

В пятнадцатой главе описываются алгоритмы шифрования: по ключевой фразе, шифр Цезаря, наоборот, код Шеннона и др.

В шестнадцатой главе вводится еще один тип данных: запись. Рассматриваются одиночные записи и массивы записей. Приводятся примеры программ.

Семнадцатая глава посвящена типу данных – множество. Рассматриваются способы построения множеств, действия над множествами, способ вывода элементов множества на экран.

В восемнадцатой главе мы рассматриваем приемы работы с файлами. Студенты научатся записывать информацию в файл разными способами (создание нового файла и дозаписи в конец существующего), считывать текстовые и числовые данные из файлов. Рассматриваются типизированные и текстовые файлы.

В девятнадцатой главе рассказывается о модуле CRT, описываются процедуры, позволяющие регулировать процесс вывода информации на экран.

В двадцатой главе рассматриваются основные графические процедуры, которые используются при подключении модуля GraphABC. Описывается организация анимации в языке PascalABC.

ГЛАВА 1. ПОНЯТИЕ АЛГОРИТМА. СВОЙСТВА АЛГОРИТМА. СПОСОБЫ ЗАПИСИ АЛГОРИТМОВ

Определение. Алгоритм является одним из основных понятий информатики, поэтому строгого определения нет. Можно дать лишь интуитивное определение: это понятное и точное предписание исполнителю совершить конечную последовательность действий, направленных на достижение указанной цели или решение поставленной задачи. Синонимы к слову алгоритм: инструкция, правило, предписание, рецепт.

В 9 веке Аль-Хорезми (Algorizmi или Algorizmus), один из крупнейших средневековых ученых, первым описал "алгоритм" арифметических действий над многозначными числами. Позднее имя автора стало нарицательным, так стали называть арифметику, основанную на десятичной позиционной системе счисления. Еще позже математики в Европе стали называть так любое вычисление, выполняемое по строго определённым правилам.

Еще одно основное понятие информатики: **исполнитель алгоритма** - это устройство или одушевленное существо, способное понять и выполнить команды, составляющие алгоритм. Исполнитель характеризуется средой, где он работает, и системой команд. Отказы исполнителя возникают при вызове команды в недопустимом для данной команды состоянии среды.

ИСПОЛНИТЕЛИ	
ФОРМАЛЬНЫЕ Не понимают смысла команд, но умеют реагировать на них строго определенным образом (ЭВМ). Одну и ту же команду всегда выполняют одинаково.	НЕФОРМАЛЬНЫЕ Могут понимать команды по-своему и действовать разными способами при одних и тех же исходных данных (служебная собака, человек). Могут отказаться выполнять команду.

Свойства алгоритма

1. Понятность: все команды должны входить в систему команд исполнителя. Он должен знать способ действий при получении этой команды.

2. Дискретность: алгоритм представляет собой последовательность шагов.

3. Определенность (детерминированность): каждая команда может быть выполнена только одним способом.

4. Результативность (конечность, финитность): за конечное число шагов исполнение алгоритма должно закончиться, алгоритм должен привести к решению или сообщить о невозможности решения.

5. Массовость и вариативность: алгоритм должен быть применим для целого класса однотипных задач. Должны быть предусмотрены все возможные случаи.

Разрабатывать алгоритмы на сегодняшний день может только человек, так как это сложная интеллектуальная задача, требующая глубоких разносторонних знаний.

Способы записи алгоритма

Словесный способ записи алгоритмов

Вербальный или словесный способ – это запись алгоритма на естественном языке. Он очень удобен, если нужно кратко и понятно для всех описать суть алгоритма. Однако, этот способ имеет недостатки, при словесном описании не всегда удается ясно и точно отобразить порядок действий.

Применим словесный способ записи алгоритма для описания процесса вычисления площади прямоугольника

$$S=x*y,$$

где S – площадь прямоугольника; x, y – длины его сторон.

Очевидно, что x, y должны быть заданы заранее, иначе задачу решить невозможно.

Словесный способ записи алгоритма выглядит так:

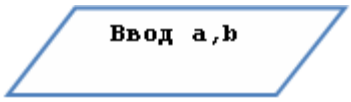
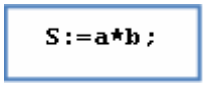
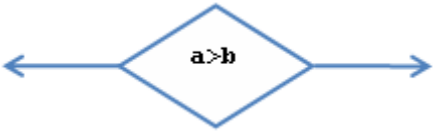
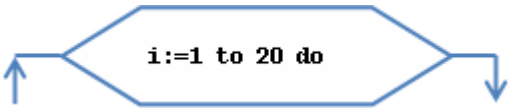
- Начало алгоритма.

- Задать численное значение стороны x.
- Задать численное значение стороны y.
- Вычислить площадь S прямоугольника по формуле S=x*y.
- Вывести результат вычислений.
- Конец алгоритма.

Графический способ описания алгоритмов

Существует несколько способов графического описания алгоритмов. Наиболее широко используемым на практике графическим описанием алгоритмов является использование **блок-схем**. Несомненное достоинство блок схем – наглядность и простота записи алгоритма.

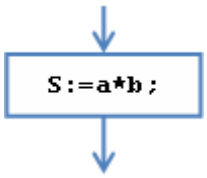
Каждому действию алгоритма соответствует геометрическая фигура (блочный символ). Перечень наиболее часто употребляемых символов приведен в таблице:

Название символа	**Обозначение и пример заполнения**	**Пояснения**
Пуск-останов		Начало, завершение алгоритма или подпрограммы
Ввод-вывод данных		Ввод исходных данных или вывод результатов
Процесс		Внутри прямоугольника записывается действие, например, расчетная формула
Решение		Проверка условия, в зависимости от которого меняется направление выполнения алгоритма
Модификация		Организация цикла

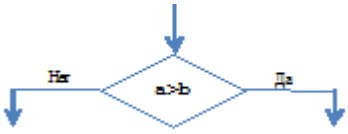
Название символа	Обозначение и пример заполнения	Пояснения
Предопределенный процесс		Использование ранее созданных подпрограмм
Комментарий		Пояснения

Пояснения:

- **блок Процесс** обозначает вычислительный процесс и применяется для обозначения действия или последовательности действий, изменяющих значения переменных или данных

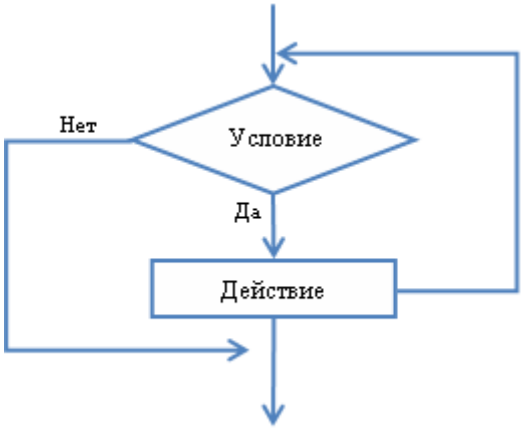


- **блок Решение** обозначает проверку условия



Если условие выполняется, то есть a>b, то следующим выполняется действие по стрелке «Да». Если условие не выполняется, то осуществляется переход по стрелке «Нет».

- **блок Модификация** используется для организации циклических (повторяющихся) действий.



- **блок Предопределенный** процесс используется для указания обращений к ранее созданным алгоритмам и программам, в том числе и библиотечным подпрограммам.
- **блок Ввод-Вывод**. При решении задачи на компьютере ввод исходных данных может осуществляться различными способами, например, с клавиатуры, с жесткого диска, с флэш-карты т. д. Задание численных значений исходных данных называется вводом, а отображение результатов расчета на экране монитора или с помощью принтера на бумаге – выводом. Если ввод-вывод не привязан к конкретному устройству, то обозначается параллелограммом. Если необходимо указать конкретное устройство ввода или вывода, то используются специальные геометрические фигуры.

		
устройство ввода или вывода	дисплей	магнитный диск

Псевдокод

Это полуформальное описание на языке, являющемся синтезом естественного языка, языка программирования и общепринятых математических выражений. В псевдокоде используются служебные слова, смысл которых определен однозначно:

алг	алгоритм
дано	данные
надо	что мы должны получить
нач	начало
кон	конец

Псевдокод - это последовательное вербальное описание кода, которое можно постепенно перенести в язык программирования. Это нестрогий план, инструмент для обдумывания проблемы и средство общения, позволяющее передавать ваши мысли другим людям.

Псевдокод используется для демонстрации того, как компьютерный алгоритм может и должен работать. Это своего рода промежуточный этап в программировании, между стадией планирования и стадией написания работающего кода. Хороший псевдокод может превратиться в комментарии к финальной версии программы и будет помогать исправлять ошибки в будущем и корректировать код.

В псевдокоде часто используется синтаксис, похожий на синтаксис языка Паскаль. Это объясняется тем, что Паскаль создавался как язык для обучения программированию и его синтаксис особенно приспособлен для восприятия человеком. Используются также Си, Фортран, Алгол и др.

Псевдокод субъективен и не стандартизирован. Понятность - основной критерий псевдокода!

Программный способ записи алгоритмов

Алгоритм должен быть записан на каком-то промежуточном языке, с точными и однозначными правилами и отличном от естественного языка и языка блок-схем, но понятном компьютеру. Такой язык принято называть **языком программирования**.

Программный способ записи алгоритма – это запись алгоритма на языке программирования, позволяющем на основе строго определенных правил формировать последовательность предписаний, однозначно отражающих смысл и содержание алгоритма, с целью его последующего исполнения на компьютере.

> *Запись алгоритма на языке программирования называется **компьютерной программой**.*

Языки программирования - это формальные знаковые системы (буквы, цифры, символы), предназначенные для написания программ.

Языки программирования представляют собой системы обозначений для описания алгоритмов и структур данных.

Языки программирования отличаются функциональными возможностями доступными типами данных и т.д.

Формальное исполнение алгоритма

Исполнитель решает задачу формально, не вникает в смысл того, что он делает, и вместе с тем, получает верный результат. Наличие алгоритма формализовало процесс решения задачи, дало возможность механически исполнять команды, а следовательно, исполнение можно поручить автомату, машине, ЭВМ!

Исполнителя можно представлять в виде устройства с кнопочным управлением. Каждая кнопка соответствует одной команде исполнителя, а нажатие означает вызов этой команды, в результате чего исполнитель совершает соответствующее элементарное действие. Управление исполнителем заключается в последовательном вызове команд. В простейшем случае можно считать, что это делает человек. Однако, человек может ошибиться при наборе длинной последовательности команд, неверно проанализировать обстановку, не успеть в критической ситуации. Возникает идея посредника - управляющего устройства, которое, получив от человека инструкции, самостоятельно вызывает команды исполнителя. В роли такого посредника может выступать ЭВМ.

Человек заранее составляет полный план (программу) управления, то есть запись, при каких условиях, какие действия и в какой последовательности должно будет выполнить устройство. А затем

устройство выполняет эту программу автоматически. Такой вид деятельности называется **программированием** или **алгоритмизацией**.

Программирование как вид человеческой деятельности требует специфического алгоритмического стиля мышления. В отличие от управления тут не так важна скорость реакции или наблюдательность человека. На первый план выступает способность планировать длинные последовательности действий, четко записывать их в виде программ, умение предвидеть все возникающие последствия и предусмотреть все условия, которые могут возникнуть при выполнении алгоритмов.

Секрет могущества ЭВМ в том, что она может быстро выполнять длинные последовательности очень простых команд и обрабатывать большие объемы информации.

Описание того, какие действия, в каком порядке и над какими порциями информации должна произвести ЭВМ, называется **программой для ЭВМ**.

Для того чтобы ЭВМ могла выполнить программу, эта программа должна быть записана по строгим правилам, в виде, доступном для обработки на ЭВМ. Такой набор правил называется **языком программирования** или **алгоритмическим языком**.

Воспринимая программу, написанную на языке программирования, ЭВМ некоторым образом ее преобразует, помещает в собственную память, а затем выполняет команды, соответствующие данной программе.

Ошибки составления алгоритмов

Из принципа формального исполнения следует, что ни исполнитель, ни ЭВМ не могут совершать ошибок. Все ошибки, связанные с управлением исполнителем, совершает человек. Компьютер не будет за нас думать и проверять программы.

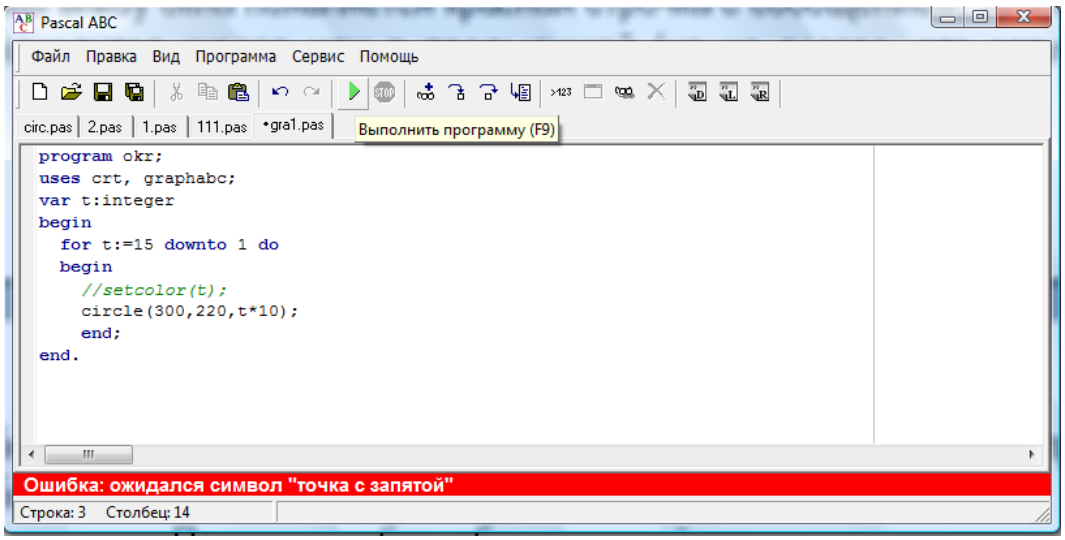
1. Синтаксические ошибки. Вызов команды, не входящей в систему команд исполнителя. Эти ошибки обнаруживает ЭВМ. Она не передает исполнителю синтаксически неверную команду и сообщает человеку о допущенной ошибке.

2. Семантические (семантика - смысл). Вызов команды в ситуации, когда эта команда не может быть исполнена. Эти ошибки приводят к отказу исполнителя (робот-чертежник не может двигаться влево, потому что там стенка).

3. Логические. ЭВМ выполнила алгоритм, исполнитель выполнил все команды, но цель, известная только человеку, не достигнута. Эти ошибки не фиксируются ни ЭВМ, ни исполнителем, поэтому важно понимать, что безотказное выполнение алгоритма еще не означает его правильности. (Например, человек использует неправильную формулу для нахождения объема пирамиды, но в ней нет ни деления на 0, ни извлечения корня из отрицательного числа, т.е. с точки зрения ЭВМ все хорошо.)

Принцип формального исполнения распространяется и на ЭВМ. Это требует от человека точной, не допускающей двусмысленности формы записи алгоритма. Такую форму обеспечивают формальные языки программирования.

Пример сообщения об ошибке



Классификация языков программирования

I. Языки программирования низкого уровня (машинно-ориентированные). Например Ассемблер. Это языки, средства которых существенно зависят от особенностей конкретной ЭВМ.

Достоинства:

-высокое качество создаваемых программ с точки зрения их компактности и скорости выполнения;

- возможность прямого использования конкретных аппаратных ресурсов;

- учет особенностей функционирования данной ЭВМ.

Недостатки:

- трудоемкость процесса составления программ

- низкая скорость программирования

II. Языки высокого уровня (машинно-независимые). Это средства описания алгоритмов решения задач и информации, подлежащей обработке, которые не требуют от программиста знания особенностей функционирования конкретной ЭВМ. К ним относятся почти все используемые сегодня языки. Программы представляют собой последовательности операторов, структурированные в соответствии с определенными правилами. Операторы языка описывают действия, которые должна выполнить система после трансляции программы на машинный язык.

Глава 2. СТРУКТУРА ПРОГРАММЫ В ЯЗЫКЕ PASCAL. ТИПЫ ДАННЫХ

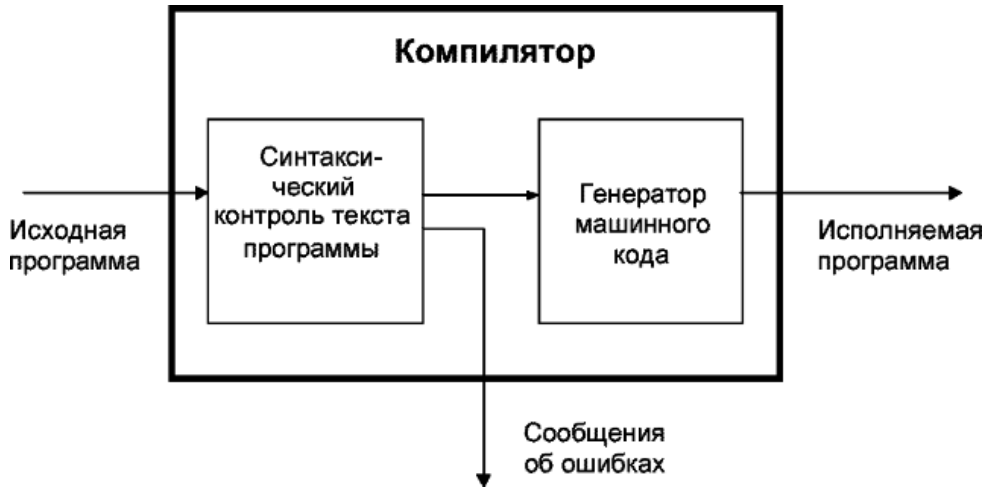
Программу, выполняющую одни и те же действия, можно написать на различных языках программирования. Какой же из них лучше? Ответ не так прост. Язык Паскаль был специально разработан для обучения программированию швейцарским ученым Никлаусом Виртом, и поэтому можно сказать, что он лучше других подходит для начальных этапов знакомства с процессом написания программ. После можно с легкостью перейти на программирование на другом языке.

В настоящее время существуют различные среды программирования для Паскаля: Turbo Pascal, Borland Pascal, Free Pascal, Pascal ABC, PascalABC.NET. Большинство сред включают в себя следующие инструменты разработки:

- текстовый редактор;

- транслятор (специальная программа, переводящая текст, написанный на языке программирования, в машинный код), трансляторы бывают двух видов: интерпретаторы и компиляторы.

- отладчик, для поиска и исправления ошибок в программе.



Замечание. Генерация выполняемой программы при компиляции происходит только в том случае, если в тексте программы нет синтаксических ошибок!

СТРУКТУРА ПРОГРАММЫ

```
Program <Имя программы>;    //  Заголовок
<Раздел подключения модулей>
<Раздел описаний>  //  Декларативная часть
Begin
<Тело программы>  //   Раздел операторов
End.
```

Раздел подключения модулей. Ключевое слово USES указывает программе, что необходимо подключить дополнительный внешний модуль (файл с описанием и определение различных функций). Например, Uses CRT, Uses graphABC.

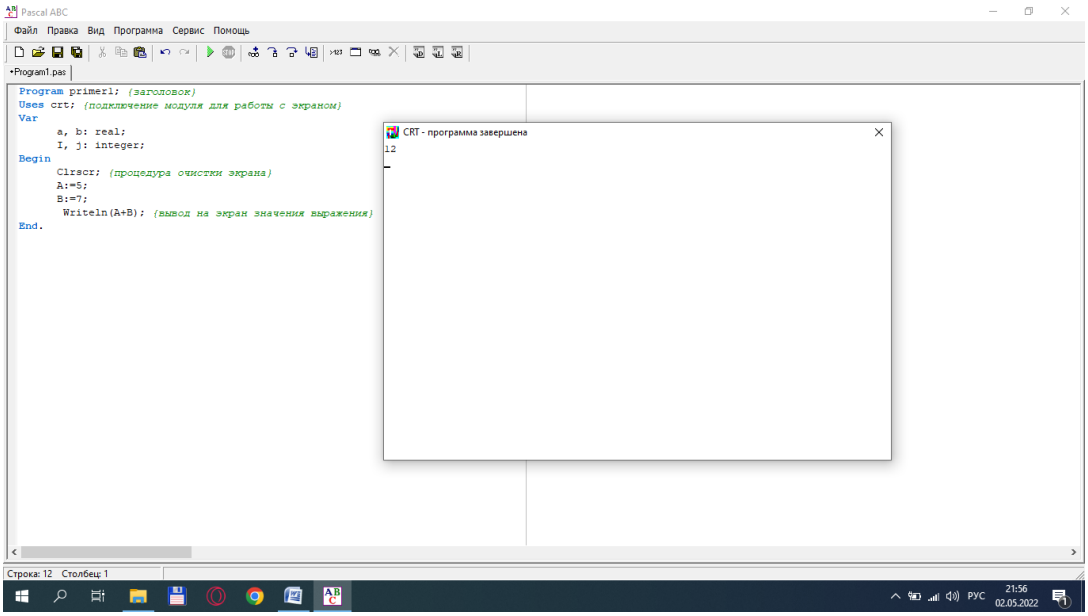
Декларативная часть. Может состоять из пяти разделов.

1. Описание меток Label. Вообще, не рекомендуется использовать метки, равно как и оператор GOTO, в программе, соблюдая принципы структурного программирования.
2. Описание типов переменных Type, вводимых программистом в дополнение к предопределенным типам.
3. Раздел описания констант Const.
4. Раздел описания переменных Var.
5. Раздел, где описываются процедуры и функции Procedure, Function.

Замечание. Порядок разделов произвольный, но желательно придерживаться указанного порядка, для быстрого нахождения нужного раздела.

Предложения в языке заканчиваются ; и только после последнего оператора End точка (.).

```
Program primer1; {заголовок}
Uses crt; {подключение модуля для работы с экраном}
Var
    c, d: real;
    k, x: integer;
Begin
    Clrscr; {процедура очистки экрана}
    C:=5;
    D:=7;
     Writeln(C+D); {вывод на экран значения выражения}
End.
```



АЛФАВИТ ЯЗЫКА

1. Буквы латинского алфавита.
2. Арабские цифры.
3. Специальные знаки . , ; : <> <= >= <> # $ @ ^ () {} [] ‘ * / + - := .. (указание диапазона), пробел.

Большинство трансляторов не различают буквы разных регистров: program=PROGRAM. Но русская А не совпадает с латинской A!

ОСНОВНЫЕ ЛЕКСЕМЫ

Идентификаторы, используемые для обозначения констант, переменных, типов, процедур, функций и т.д.

- Должны быть уникальными (для разных объектов – разные имена).

- Имеют ограничение по длине (зависит от конкретной реализации языка).

- Могут состоять только из латинских букв, цифр и знака подчеркивания_; не могут начинаться с цифры.

- Не могут совпадать с зарезервированными словами, такими как begin, var и т.д.

Замечание. Желательно, чтобы идентификаторы несли смысловую нагрузку, но и слишком длинными быть не должны.

ЗАРЕЗЕРВИРОВАННЫЕ СЛОВА ЯЗЫКА

And	Asm	Array	Begin	Case
Const	Constructor	Destructor	Div	do
Downto	Else	End	Exports	file
For	Function	Goto	If	implementation
In	Inherited	Inline	Interface	label
Library	Mod	Nil	Not	object
Of	Or	Packed	Procedure	program
Record	Repeat	Set	Shl	shr
String	Then	To	Type	unit
Until	uses	var	while	with
xor				

Переменные характеризуются типом и значением. Тип определяет, какие значения может принимать переменная и какие операции с ней можно выполнять. Значение переменной может быть присвоено с помощью констант.

Константы:

1) **Числовые** – целые, вещественные и шестнадцатеричные. *Целые* состоят из цифр и знака + или -. Например, 1 +16 -12. *Вещественные* могут быть с фиксированной и плавающей точкой: 1e-2 ($1x10^{-2}$); 1E+2; 12.5; 0.01; +12.7; - 34.7. *Шестнадцатеричные* начинаются со знака $ и содержат цифры 0, 1, …9, A, B, C, D, E, F. Например, $ABC, $1F.

2) Символьные константы заключаются в апострофы. Если символа нет на клавиатуре, его можно задать по коду: ‘a’ ‘A’ #32

3) Строковые константы также заключаются в апострофы: ‘Программа’, ‘program’.

```
Const
Pi=3.14;
Name1=’Татьяна’;
Name2=’Виктор’;
R21=6.33187E+03;
w_w_w=934122;
```

Договоримся, для удобства отладки и повышения понятности кода:

- Давать «говорящие» имена переменным, если находим сумму, например, называть переменную Sum, summa. Среднее арифметическое – SrArifm и так далее.

- Снабжать программу комментариями.

- Использовать текст подсказок, указаний пользователю при вводе и выводе информации.

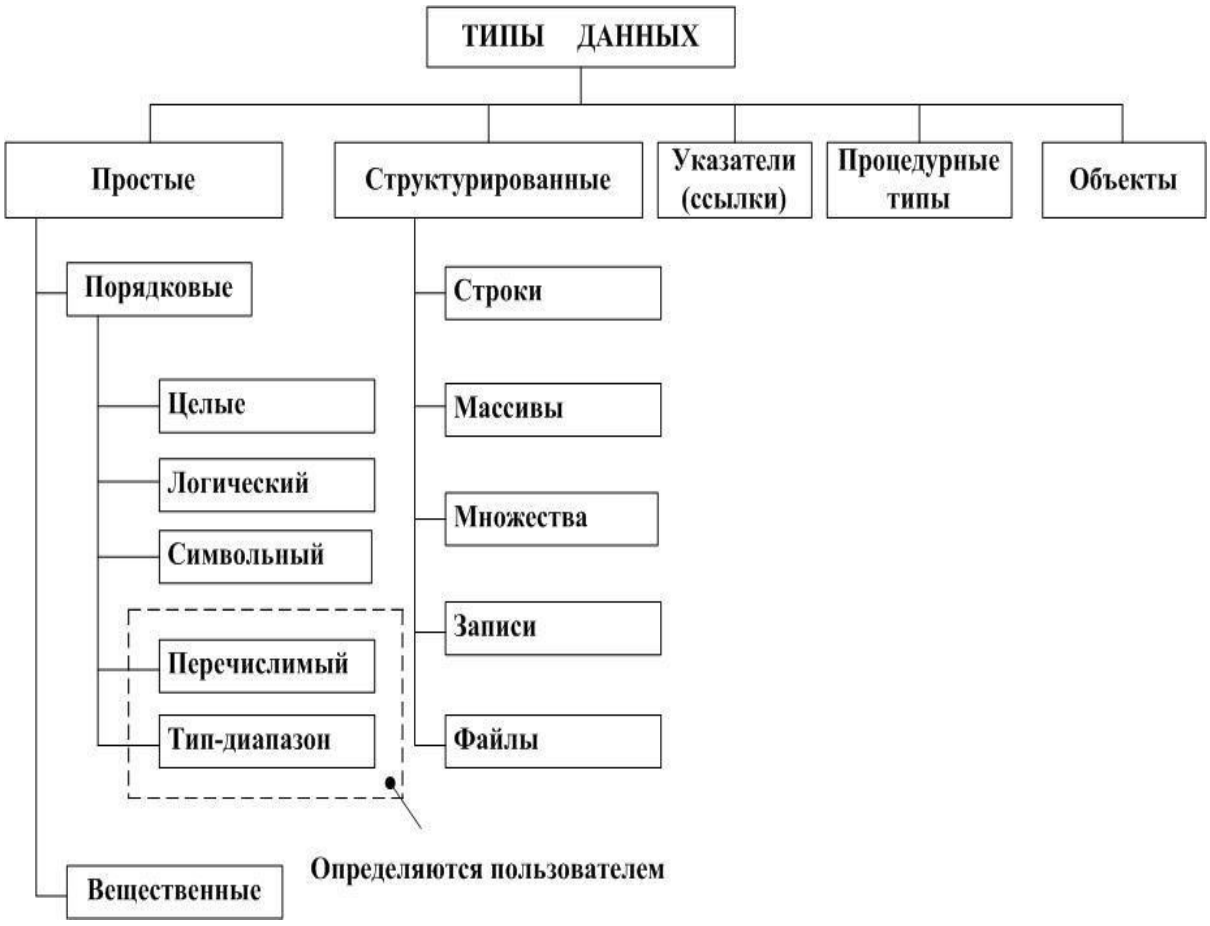
- Использовать отступы для выделения блоков.

ТИПЫ ДАННЫХ В ЯЗЫКЕ ПАСКАЛЬ

Переменная – это именованный участок памяти для хранения данных определенного типа. Значение переменной может меняться при выполнении программы. Значение константы изменено быть не может!

A
3.14

Summa
7341



К базовым типам относятся:

- Тип целых чисел **Integer**
- Тип «длинных» целых чисел **Longint**
- Тип неотрицательных целых чисел от 0 до 255 **Byte**

- Тип неотрицательных целых чисел от 0 до 65535 **Word**
- Тип действительных чисел (с дробной частью) **Real**
- Символьный тип **Char** имеет длину 1 байт
- Строковый тип **String,** строка длиной до 255 символов
- Логический тип **Boolean,** может принимать два значения: true/false.

На физическом уровне типы данных отличаются друг от друга количеством ячеек памяти (байтов), выделяемых для хранения соответствующей переменной.

- **Пользовательские типы**

перечисление	поддиапазон
Type Col=(red, black, white)	Type Alf='A'..'Z' Day=1..31

```
Var
A, B, H, K_22, Angle: Real;
Name3: String;
Flag: Boolean;
I, J, K, Cout: Word;
```

ФУНКЦИИ ДЛЯ РАБОТЫ С ПОРЯДКОВЫМ ТИПОМ

ORD(X) – возвращает порядковый номер элемента порядкового типа

PRED(X) – возвращает предыдущее значение

SUCC(X) – возвращает следующее значение

Ord('y')=121 Pred('c')='b' Succ('c')='d'

CHR(X) – возвращает символ по его порядковому номеру

Иногда возникает необходимость присвоить значение **вещественного** типа переменной **целого** типа, тогда используются функции: TRUNC(X) и ROUND(X). Первая возвращает целую часть своего аргумента, вторая – округляет аргумент до ближайшего целого значения.

Основные типы данных в Паскале		
Тип	**Диапазон**	**(байт)**
Целые типы		
Byte;	0..255	1
Shortint;	-128..127	1
Integer;	-32768..32767	2
Word;	0..65535	2
Longint;	-2147483648..2147483647	4
Вещественные типы		
Real;	$2.9*10^{-39}..1.7*10^{38}$	6
Single;	$1.5*10^{-45}..3.4*10^{38}$	4
Double;	$5.0*10^{-324}..1.7*10^{308}$	8
Extended;	$1.9*10^{-4951}..1.1*10^{4932}$	10
Comp;	$-2*10^{63}+1..2*10^{63}-1$	8
Логический тип		
Boolean;	0..255 (0- True, 1..255-False)	1
Символьный тип		
Char;	(любой из 256 Символов ASCII)	1
Литерный (строковый) тип		
String;	Строка не более 255 символов	1

ГЛАВА 3. ОПЕРАТОРЫ ВВОДА, ВЫВОДА И ПРИСВАИВАНИЯ. ЛИНЕЙНЫЕ АЛГОРИТМЫ

Стандартные математические функции Паскаля			
Обращение	**Тип аргумента**	**Тип результата**	**Примечание**
Abs(x)	Real, integer	Тип аргумента	Модуль аргумента
ArcTan(x)	Real, integer	Real	Арктангенс (значение в радианах)
Cos(x)	Real, integer	Real	Косинус, угол в радианах
Exp(x)	Real, integer	Real	Экспонента
Frac(x)	Real	Real	Дробная часть числа
Int(x)	Real, integer	Real	Целая часть числа
Ln(x)	Real, integer	Real	Логарифм натуральный
Pi	Нет	Real	3,141592653
Power(x,y)	Real, integer	Real	X в степени y
Sin(x)	Real, integer	Real	Синус, угол в радианах
Sqr(x)	Real, integer	Тип аргумента	Квадрат аргумента
Sqrt(x)	Real, integer	Real	Корень квадратный
Random	Нет	Real	Псевдослучайное число в интервале [0, 1]

Random(N)	Integer	Integer	Псевдослучайное число в интервале [0, N]
Round(x)	Real	Integer	Округление до ближайшего целого
Trunc(x)	Real	Integer	Отбрасывание дробной части числа

- Для возведения в степень можно использовать известное математическое тождество: $\mathbf{x^y = e^{ylnx}}$
- в Паскале можно вычислить только значение натурального логарифма, для вычисления логарифма с произвольным основанием используется математическое тождество: $\mathbf{\log_a b = \ln b / \ln a}$

Пример вычисления степени числа.

```
var x, n, z: real;
begin
readln(x,n);
   z:=power(x,n);
   writeln(z);
end.
```

АРИФМЕТИЧЕСКИЕ ВЫРАЖЕНИЯ

```
Counter:=0;
D:=b*b-4*a*c;
Pi:=3.1415;
z:=(r1+r2)/(r1*r2);
```

Выражение состоит из **операндов** (константы, переменные) и **операторов** (действия над операндами). Тип выражения диктуется типами операндов, входящих в выражение, и операциями, выполняемыми над ними.

Для операций **сложения, вычитания и умножения** тип результата в зависимости от типа операнда будет таким:

Операнд 1	Операнд 2	Результат
Integer	Integer	Integer
Integer	Real	Real
Real	Integer	Real
Real	Real	Real

Для операции **деления** тип результата в зависимости от типа операнда будет таким:

Операнд 1	Операнд 2	Результат
Integer	Integer	Real
Integer	Real	Real
Real	Integer	Real
Real	Real	Real

В Pascal есть операции **целочисленного деления** и **нахождения остатка от деления**. При выполнении целочисленного деления (операция **div**) остаток от деления отбрасывается. Их операнды должны иметь тип Integer.

Например, 19 div 3 = 6; 38 div 5 = 7; 126 div 10 = 12, 9 div 10 = 0.

Операция **mod** находит остаток от деления одного целого числа на другое.

Например, 15 mod 3 = 0; 18 mod 5 = 3; 123 mod 10 = 3, 7 mod 10 = 7.

Порядок вычислений в выражениях следующий:

1. Вычисляются подвыражения, заключенные в скобки;
2. Затем выполняются операции с наибольшим приоритетом; обычно используются следующие уровни приоритетов (в порядке убывания):
   - возведение в степень;
   - мультипликативные операции: * , / , div , mod;
   - унарные операции: + , - , abs , not;
   - аддитивные операции: +, -;
   - операции отношения: = , <>, <, >, <=, >=;
   - логические операции: and, or, not;
3. Операции с одинаковым приоритетом выполняются слева направо.

Хотя нет ограничений на сложность выражений, однако выражения, содержащие более 7 операндов, трудны для чтения и понимания и поэтому такие выражения не рекомендуется использовать.

Математическое выражение: $x^{3/2} - 7x + tg\,(x+2)$

Выражение на Паскале: **exp(3*ln(x)/2)-7*x+sin(x+2)/cos(x+2)**

ОПЕРАТОРЫ ДЕЙСТВИЯ

Операторы действия - это средства языка, позволяющие изменять в процессе выполнения программы состояние вычислений. Самый простой оператор действия - **оператор присваивания**.

<имя_переменной>:=<выражение>

Выражение справа должно приводить к значению того же типа, какого и сама переменная, или совместимого типа, Real:=Integer (Word), но не наоборот!

Сначала вычисляется значение выражения справа, потом результат заносится в ячейку памяти.

Пример оператора присваивания

p:=18+235;{переменной a присваиваем математическое выражение. Стоит заметить, что переменная p должна быть числового типа}

Оператор присваивания, несмотря на кажущуюся простоту, имеет очень важное алгоритмическое значение.

Удобно считать, что выполнение присваивания переменной некоторого значения означает помещение этого значения в выделенный для переменной ящик.

Операторы языка	
Простые: присваивание и вызов процедуры	Структурные: составной оператор, ветвление, циклы

СОСТАВНОЙ ОПЕРАТОР

Begin

<оператор 1>;

<оператор 2>;

…..

<оператор N> {; перед end можно не ставить}

End;

Составной оператор используется там, где по синтаксису можно выполнить **один** оператор, а требуется выполнить **несколько**.

ВВОД И ВЫВОД ДАННЫХ

Ввод данных с клавиатуры организуется с помощью стандартной процедуры

read(<список ввода>)

или ее разновидности

readln(<список ввода>).

Элементы списка ввода - идентификаторы (имена) переменных, перечисляемые через запятую. При выполнении этого оператора от пользователя требуется ввести с клавиатуры последовательность значений, **разделяя их пробелами**. Программа приостанавливает свою работу и ждет, пока нужные данные будут набраны и нажат ENTER. Введенные значения присваиваются переменным в том порядке, в каком их имена указаны в скобках после read.

Пример ввода данных с клавиатуры

Read (x,y,u); 14 13 6.17

X=14 Y=13 U=6.17

Readln отличается от Read тем, что после выделения очередного числа из введенной с клавиатуры строки и присваивания его последней переменной из списка инструкции **Readln,** оставшаяся часть строки теряется и следующая инструкция **Read** или **Readln** будет требовать нового ввода.

Readln (a,b);

Read (c); 4 23 0.17

a=4 b=23 c=??

Программа будет ожидать ввода нового числа, чтобы присвоить его переменной c.

Чтобы «подсказать» пользователю, какие данные ожидает от него программа, перед каждой командой ввода следует располагать инструкцию **write.**

Вывод данных на экран производится с помощью стандартной процедуры

write(<список вывода>)

или ее разновидности

writeln(<список вывода>).

Список вывода может содержать константы, переменные, выражения, формат вывода. Выражения в списке вывода разделяются запятыми.

Пример вывода данных на экран

write(a,b,c);{где a,b,c - переменные. После вывода данных на экран, курсор останется на последнем символе}

writeln(a,b,c);{где a,b,c - переменные. После вывода данных на экран, курсор перейдет на новую строку)}

Окончание **ln** в имени процедуры подобно ситуации с операторами ввода, означает, что курсор автоматически будет переведен в начало следующей строки экрана.

При вызове оператора **Writeln** без параметров (без списка вывода) происходит переход на новую строку. Применяется, например, при выводе матрицы в привычном в математике виде в виде таблицы.

Элементы списка ввода - идентификаторы (имена) переменных, выражения, константы, перечисляемые через запятую. Надо учитывать, что объекты выводятся **без пробелов**! Необходимо добавлять пробелы вручную.

Пример.

```
Write(Summa);
Writeln('Корни уравнения   x1=', x1, '   x2=', x2);
```

ФОРМАТНЫЙ ВЫВОД ДАННЫХ

1. Для переменной **целого** типа: write(d:8) – для вывода значения переменной отводится 8 позиций. Выравнивание по правой границе поля.

write(‘Всего изделий:’, kol:8);

Всего изделий: 35.

2. Чтобы задать формат для дробной переменной, надо задать два целых числа, разделенных двоеточием. Нужно определить ширину поля вывода, а также число цифр справа от десятичной точки. Если задать только ширину поля, на экране появится число в формате с плавающей точкой.

Пример.

x1=13.25 x2=-0.3401

write(‘x1=’, x1:5:2, ‘ x2=’, x2:12);

x1=13.25 x2=-3.40100E-01

Если ширины поля недостаточно для ввода значения переменной, выводится число в формате с плавающей точкой и десятью цифрами после запятой (все поле вывода занимает 17 позиций).

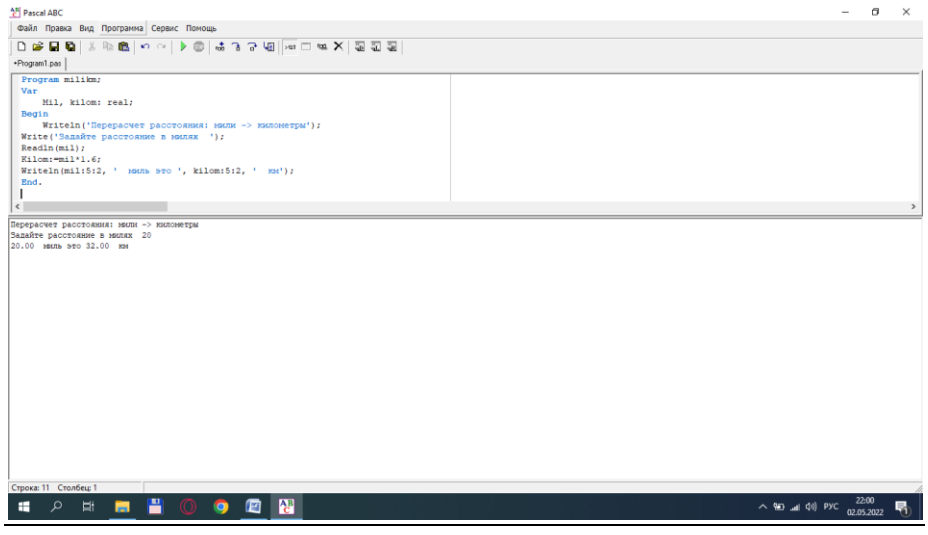
Пример.

```
Writeln(‘Введите информацию:’);
Write(‘Стоимость одного изделия ’);
Readln(C);
Write(‘Количество изделий’);
Readln(K);
Write(‘Скидка ’);
Readln(Sk);
```

Если тип данных не соответствует или не может быть приведен к типу переменных, указанных в инструкции read, программа аварийно завершает работу и выводит сообщение об ошибке.

Задача. Перевести расстояние в милях в расстояние в километрах (1 миля примерно равна 1,6 км).

```
Program milikm;
Var
        Mil, kilom: real;
Begin
  Writeln('Перерасчет расстояния: мили -> километры');
  Write('Задайте расстояние в милях  ');
  Readln(mil);
  Kilom:=mil*1.6;
  Writeln(mil:5:2, '  миль это ', kilom:5:2, '  км');
 End.
```



Задачи для решения

1. Найти площадь параллелограмма со стороной a и высотой h.

2. Вычислить площадь круга радиуса R.

3. Найти площадь кольца, образованного кругами с радиусами R1 и R2.

4. Вычислить площадь треугольника, заданного координатами вершин (x1,y1), (x2,y2) и (x3,y3).

5. Даны координаты двух точек. Составить уравнение прямой, проходящей через эти точки в виде y=kx+b.

Задачи для самостоятельного решения

1. Написать программу для вычисления расстояния между двумя точками с координатами x1, y1, x2, y2, используя формулу из геометрии. Результат вывести на экран с комментариями.

2. Составить программу, которая запрашивает имя пользователя и его год рождения, затем подсчитывает количество лет, дней и минут, прожитых, примерно, этим человеком. Результаты вывести на экран.

3. Составить программу, вычисляющую по закону Ома значение силы тока (I=U/R) для участка цепи, запрашивая с клавиатуры значения напряжения на концах участка и его сопротивления.

4. Составить программу, которая, используя генератор случайных чисел, выводит на экран случайное число, значение которого лежит в пределах от 7 до 10.

5. Составить программу, которая, используя генератор случайных чисел, выводит на экран случайное число, значение которого лежит в пределах от 17 до 100.

6. Составить программу, которая, используя генератор случайных чисел, выводит на экран случайное число, значение которого лежит в пределах от 37 до 110 и имеет 2 знака в дробной части.

7. Эти задачи предназначены для приобретения навыков реализации линейных алгоритмов. Вычислить значения заданных функций.

1	$S = \ln\left(x + \sqrt{x^2 + 1}\right)$ $Z = ae^{-ax}\sin(bx)$	4	$Y = \sin^3(x^2 + a^2) - \sqrt{\frac{x}{b}}$ $Z = \frac{x^2}{a} + \cos(x+b)^3$
2	$Y = \frac{2\cos(x - \frac{\pi}{6})}{0{,}5 + \sin^2 y}$ $Z = 1 + \frac{b^2}{3 + \frac{b^2}{5}}$	5	$Y = x^2(x+1) - \sin^2(x+a)$ $Z = \sqrt{\frac{xb}{a}} + \cos^2(x+b)^3$
3	$Y = ax^{\frac{3}{2}}$ $Z = \frac{6a}{\sqrt{x}(4 + 9a^2 x)}$	6	$Z = \frac{-t\sqrt{t^2 + 8a^2}}{4a}$ $W = (x^2 + y^2 - ax)^2$
7	$Y = at^3$ $Z = \frac{1}{27a^2}\left((4 + 9a^2 x)^{\frac{3}{2}} - 8\right)$	16	$Z = c^2\cos x + \sqrt{c^4\cos^2 2x}$ $W = (x^2 + y^2)^2 - 2c(x^2 - y^2)$
8	$Y = e^{-bt}\sin(at + b) + \sqrt{\lvert bt + a \rvert}$ $S = b\sin(at^2\cos 2t) - 1$	17	$Z = \frac{4a(x+a)}{2a+x} + \sin(\frac{x}{2a})$ $W = \pi a^2\frac{3x + 2a}{x}$
9	$S = x^3\mathrm{tg}(x+b)^2 + a$ $Z = \frac{bx^2 - a}{e^{bx} - 1}$	18	$Y = (a+x)^3\cos x + \ln\cos x$ $Z = x^{\frac{2}{3}} + e^{(a-x)^2}$
10	$Y = \frac{at^3}{1 + t^2}$ $Z = \frac{a\sin^2 x}{\cos x}$	19	$Y = \frac{ae^{\frac{x}{a}} + e^{-\frac{x}{a}}}{2}$ $Z = \cos\left(a + \frac{x^2}{2a}\right)^{\frac{3}{2}}$
11	$Y = at(t^2 - 1)^{\frac{3}{2}}$ $Z = (e^{tx} + \pi)t^2 + 1$	20	$Y = \frac{ax\sqrt{bx} + ax^2}{(b-x)^2}$ $Z = (a+b) + ba\sin^2\left(\frac{b+a}{x}\right)$

12	$Y = at\dfrac{t^2-1}{t^2+1}$ $Z = 2a^2 - 0{,}5\pi a^2$	21	$Y = e^{\cos x + \ln^2(x+a)}$ $Z = \sqrt{-bx^2 + abx - c}$
13	$Z = (x-a)^2(x^2+y^2) - \ln x$ $W = \dfrac{\sqrt{x^2-a^2}}{2}$	22	$Z = \dfrac{x^2}{a^2} + \dfrac{y^2}{b^2} + \dfrac{xy}{ab}$ $W = -\pi + \operatorname{arctg}\left(\dfrac{x-y}{ab+xy}\right)$
14	$Z = \dfrac{\sin x}{\sqrt{1+t^2\sin^2 x}}$ $W = e^{-tx}\sqrt{x+1} + e^{tx}\sqrt{x+1.5}$	23	$e = \operatorname{arctg}\left(x^t + a\right) - ax$ $Z = e^{-ax} + \ln^2(tx-a)^2$
15	$y = \dfrac{a^2x - e^{-x}\cos ax}{ax - e^{-x}\sin ax}$ $Z = \ln(a+x) + e^{2x}\ln(a^2+x^2)$	24	$y = \left\lvert ax^2 - x\sin^3 a \right\rvert$ $Z = \sqrt{\operatorname{arctg}\dfrac{a^2+x^2}{ax}}$

ГЛАВА 4. КОМАНДА ВЕТВЛЕНИЯ. ЛОГИЧЕСКИЕ ВЫРАЖЕНИЯ. ОПЕРАТОР ВЫБОРА

Вспомним структуру простой программы (линейный алгоритм):

```
Program Имя;
Var
        {объявление переменных}
Begin
        {инструкции программы}
End.
```

ОПЕРАТОР БЕЗУСЛОВНОГО ПЕРЕХОДА

Некоторые строки в программе могут быть снабжены специальными метками, в качестве такой метки может быть использовано целое неотрицательное число. Описывают имеющиеся в программе метки в разделе описания меток **Label**.

```
Label 1, 2, 8;
…..
2:  writeln(a);
```

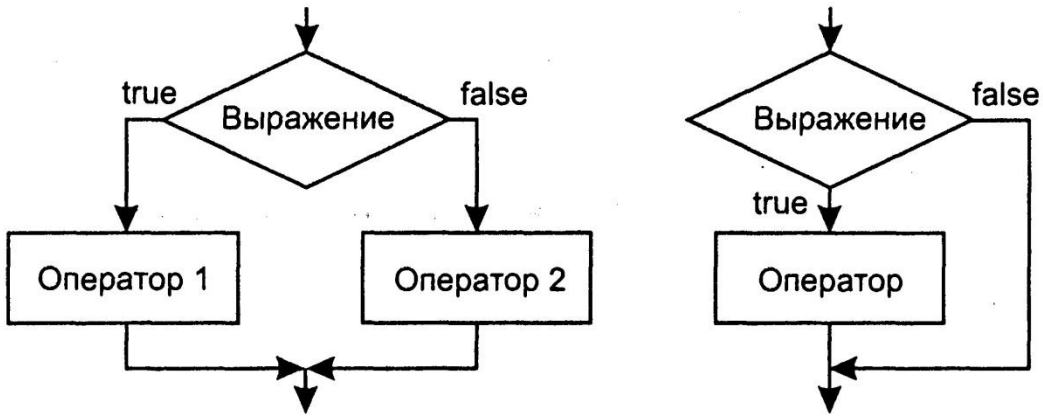
Если требуется, чтобы последовательное выполнение программы было нарушено, применяют оператор, передающий управление в произвольную ее точку. Это команда **безусловного перехода**:

GOTO <МЕТКА>

Вообще, использование данной команды считается среди программистов дурным тоном. Всегда есть возможность обойтись без нее.

УСЛОВНЫЙ ОПЕРАТОР

Используется, когда необходимо выполнить различные последовательности действий в зависимости от истинности или ложности условия.



Условие – это логическое выражение, значение которого может быть либо истинно, либо ложно. Простое условие состоит из двух **операндов** и **оператора сравнения**: <, >, =, <>, >=, <=.

Summa<100

Score>=HBound

Sim=Chr(13)

Операндами могут быть: переменная, константа, функция или выражение.

Из простых условий можно составлять **сложные** (составные), с помощью **логических операторов NOT, AND** – логическое И и **OR** – логическое ИЛИ.

Операнд 1	Операнд 2	and	or	not
л	**л**	**л**	**л**	**и**
л	**и**	**л**	**и**	**и**
и	**л**	**л**	**и**	**л**
и	**и**	**и**	**и**	**л**

Пример составного условия.

(Summa > 500) and (Day=7)

((Summa>500) and (Day=7)) or (Summa >1000)

Выбор действий в зависимости от выполнения условия может быть реализован при помощи инструкций IF и CASE.

```
IF  <условие>
        THEN
            BEGIN
                {инструкции, выполняемые, если условие истинно}
            END
        ELSE
            BEGIN
                { инструкции, выполняемые, если условие ложно}
            END;
```

Если нужно выполнить только одну инструкцию, то begin и end можно не писать.

Пример. if t=1

```
        then
            z:=r1+r2
        else
            z:=(r1+r2)/(r1*r2);
```

ОПЕРАТОР ВЕТВЛЕНИЯ В НЕПОЛНОЙ ФОРМЕ

```
IF  <условие>
        THEN
            BEGIN
                {инструкции, выполняемые, если условие истинно}
            END;
```

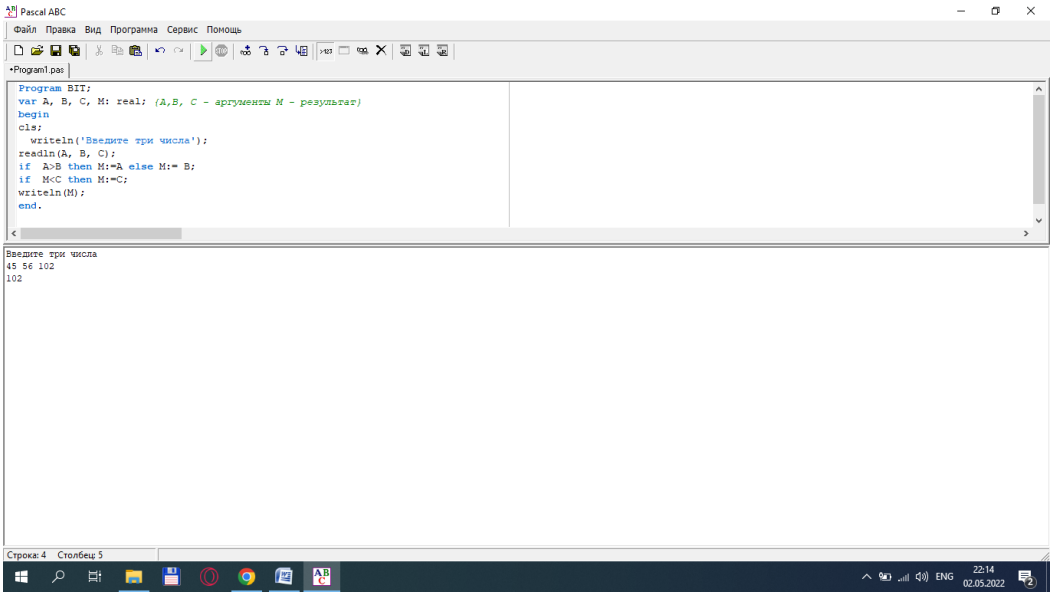
Замечание. Не следует писать сложных логических выражений. Разбивайте их на части с помощью переменных типа BOOLEAN.

Пример (Большее из двух чисел).

```
Program BID;
var A, B, C: real; {A,B - аргументы C - результат}
begin
   writeln('Введите два числа');
   readln(A, B);
   if A>B then C:=A else C:= B;
   writeln(C);
end.
```

Пример (Большее из трех чисел).

```
Program BIT;
var A, B, C, M: real; {A,B, C - аргументы M - результат}
begin
   writeln('Введите три числа');
   readln(A, B, C);
   if  A>B then M:=A else M:= B;
   if  M<C then M:=C;
   writeln(M);
end.
```

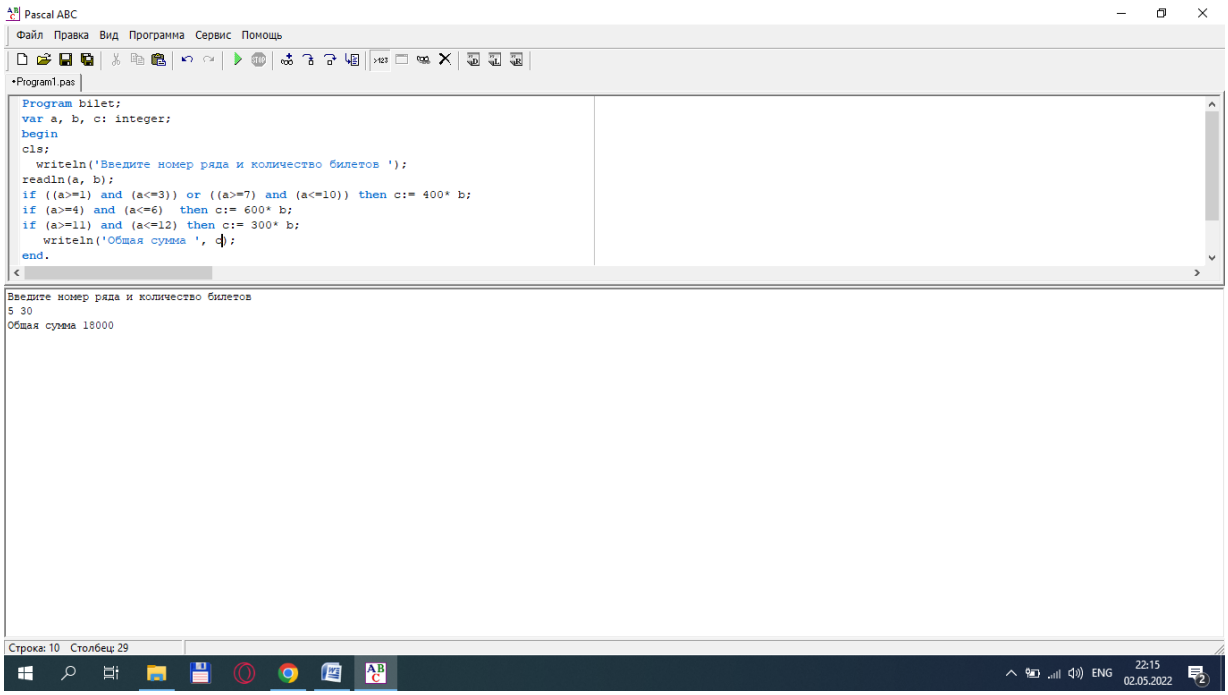


Пример (Максимальное и минимальное числа).

```
Program minmax;
var x, max, min: integer; {A,B - аргументы C - результат}
begin
   writeln(‘Введите первое число ‘);
   readln(max);
   min:=max
   write(‘Введите второе число ‘);
   readln(x);
   if x>max then max:=x else min:=x;
   write(‘Введите третье число ’);
   readln(x);
   if x>max then max:=x;
   if x<min then min:=x;
   writeln(‘Наименьшее число ’, min);
   writeln(‘Наибольшее число ’, max);
end.
```

Задача (Цена билета). Написать программу, которая по номеру ряда и количеству мест выдает общую цену за билеты. 1-3 ряд: 500 руб, 4-7 ряд: 800 руб, 8-10 ряд: 500 руб, 11-13 ряд: 350 руб.

```
Program bilet;
var kol, n, cena: integer;
begin
  writeln(‘Введите количество билетов и номер ряда‘);
  readln(kol, n);
  if ((n>=1) and (n<=3)) or ((n>7) and (n<11)) then c:= 500* kol;
  if (n>=4) and (n<=7)  then cena:= 800* kol;
  if (n>=11) and (n<=13) then cena:= 350* kol;
 writeln(‘Общая стоимость ’, cena);
end.
```



КОМАНДА ВЫБОРА (CASE)

Позволяет реализовать **множественный выбор** и в общем виде записывается так:

```
CASE  <выражение> OF
        Список констант 1: BEGIN
                {последовательность инструкций 1}
                END;
    Список констант 2: BEGIN
                {последовательность инструкций 2}
                END;
    …….
    Список констант N: BEGIN
                {последовательность инструкций N}
                END;
        ELSE
            BEGIN
                { последовательность инструкций, выполняемая, если значение выражения не попало ни в один из списков констант}
            END;
END;
```

- Выражение – выражение, от значения которого зависит дальнейший ход программы;
- Список констант – константы, разделенные запятыми.
- Если между begin и end только один оператор, то операторные скобки можно не использовать.

Пример. I способ.

```
Case day of
        1, 2, 3, 4, 5: write(‘Рабочий день’);
        6: write(‘Суббота!’);
        7: write(‘Воскресенье!’);
    End;
```

II способ.

```
Case day of
        1..5: write(‘Рабочий день’);
        6: write(‘Суббота!’);
        7: write(‘Воскресенье!’);
    End;
```

III способ.

```
Case day of
        6: write(‘Суббота!’);
        7: write(‘Воскресенье!’);
    Else write(‘Рабочий день’);
    End;
```

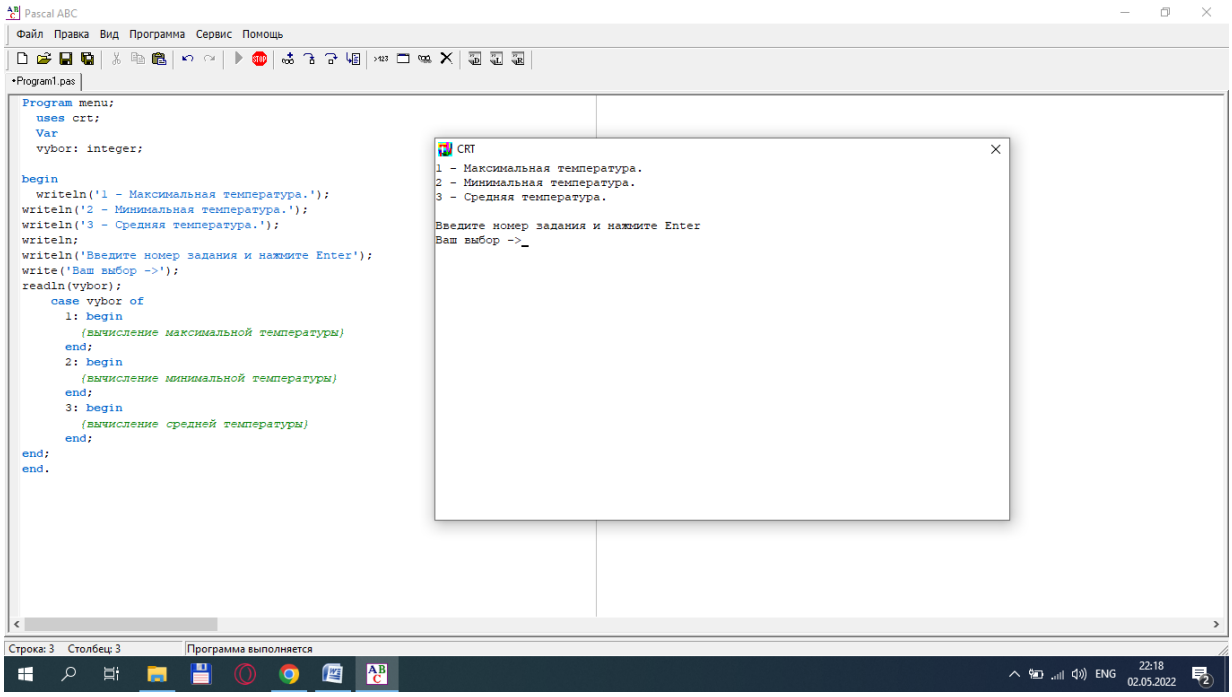
Замечание 1. Между служебными словами else и end могут находиться несколько операторов, применять составной оператор (операторные скобки) не надо.

Замечание 2. Оператор case может быть заменен группой операторов if, но его использование более наглядно и более гибко при модификациях.

ИСПОЛЬЗОВАНИЕ ОПЕРАТОРА CASE ДЛЯ ОРГАНИЗАЦИИ МЕНЮ

Оператор выбора удобно использовать для организации меню (выбора действий в зависимости от введенного значения переменной). Например, это можно сделать, как в данном фрагменте программы.

```
Program menu;
Var
    vybor: integer;
begin
    writeln(‘1 – Максимальная температура.’);
writeln(‘2 – Минимальная температура.’);
writeln(‘3 – Средняя температура.’);
writeln;
writeln(‘Введите номер задания и нажмите Enter’);
write(‘Ваш выбор ->’);
readln(vybor);
            case vybor of
                1: begin
                {вычисление максимальной температуры}
                end;
                2: begin
                    {вычисление минимальной температуры}
                end;
                3: begin
                    {вычисление средней температуры}
                end;
    end;
end
```



.

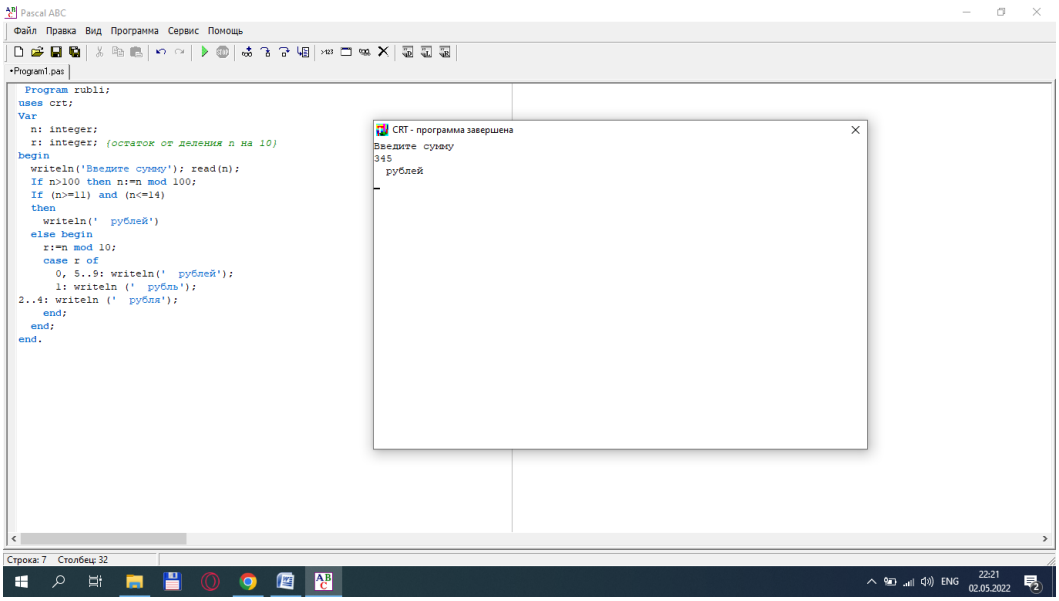
Задача. Вывод числовой информации с поясняющим текстом.

В зависимости от последней цифры числа, должно быть напечатано:

0, 5, 6, 7, 8, 9	рублей
1	рубль
2, 3, 4	рубля
Исключение 11, 12, 13, 14	рублей

```
Program rubli;
Var
    n: integer;
    r: integer; {остаток от деления n на 10}
begin
    {организовать ввод n}
    If n>100 then n:=n mod 100;
    If (n>=11) and (n<=14)
```

```
        then
                writeln(‘  рублей’)
        else begin
                r:=n mod 10;
                case r of
                        0, 5..9: writeln(‘  рублей’);
                        1: writeln (‘  рубль’);
                2..4: writeln (‘  рубля’);
                end;
        end;
end.
```



Вопросы и задачи для решения

1. Что напечатает программа при x=7 и y=11?

```
If x+16<=y then
        begin
                x:=x*12-2*y;
```

```
            y:=-12*x;
        end
    else
        begin
            x:=x-2*y;
            y:=8*x-7
        end;
    writeln(x,y);
```

2. Что напечатает программа при x=20 и y=9?

```
    If x>y  then
        begin
            x:=x+y;
            y:=2*y+4;
        end
    else
        begin
            x:=2*(x+y);
            y:=sqr(5*y)
        end;
    writeln(x,y);
```

3. Какие значения (истина или ложь) принимают составные условия?

a) (7>3) and (8<15);

b) (7>3) or (8<15);

c) (7>3) or not (8<15);

d) Not (7>3) and (8<15);

e) (2<>3) and (14>14);

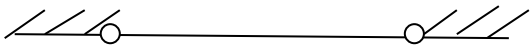
f) Not (2<>3) and (14>14);

g) Not ((2<>3) and (14>14));

4. Запишите с помощью условного оператора принадлежность x промежуткам:

a) $x \in [-2;5]$;

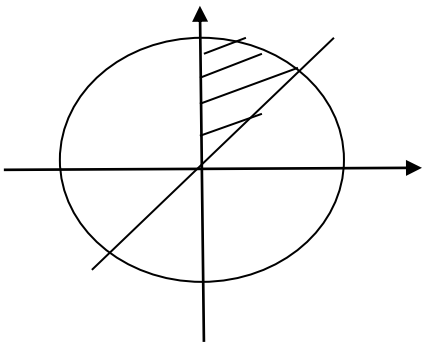
b) $y \in (1;10)$;

c) 

1 10

5. Написать программу, проверяющую, принадлежит ли точка области.

$$\begin{cases} x^2 + y^2 < 4 \\ x > 0 \\ y > x \end{cases}$$



6. Решить задачу про билеты с использованием оператора выбора (case).

7. Написать программу, решающую квадратное уравнение $ax^2+bx+c=0$.

8. Написать программу, решающую квадратное неравенство $ax^2+bx+c<0$.

<table>
<tr><td colspan="3">a>0</td><td colspan="3">a<0</td></tr>
<tr><td>D>0</td><td>D=0</td><td>D<0</td><td>D>0</td><td>D=0</td><td>D<0</td></tr>
<tr><td>x1<x<x2</td><td colspan="2">Решений нет</td><td>x<x1 и x>x2</td><td>x – любое число, кроме x1</td><td>x – любое число</td></tr>
<tr><td colspan="6">a=0</td></tr>
<tr><td colspan="2">b>0</td><td colspan="2">b=0</td><td colspan="2">b<0</td></tr>
<tr><td colspan="2">bx+c<0
x< -c/b</td><td colspan="2">c<0
если c>=0, решений нет;
если c<0, то x любое</td><td colspan="2">bx+c<0
x>-c/b</td></tr>
</table>

Неравенства для проверки

5x^2+9x-2<0	(-2; 0,2)
-x^2+2x+15<0	(-беск; -3) и (5; +беск)
2x^2+13x-7>0	(-беск; -7) и (0,5; +беск)
25x^2+30x+9<0	Решений нет
x^2+4>0	Любое число
3x-9<0	x<3
6<0	Решений нет

Задачи для самостоятельного решения

1. Составить программу, чтобы компьютер запросил имя пользователя и его год рождения, затем подсчитал возраст человека, в зависимости от возраста разработайте вариант диалога с пользователем (еще не учишься, учишься в таком-то классе (использовать формулу!), уже не учишься).

2. Написать программу вычисления y в зависимости от значения *x*

y = 1/x при x < 0, y = 2*x^2 при других x.

3. Написать программу вычисления y в зависимости от значения *x*

y = 1/ x^2 при x > 0, y = x/6 при других x.

4. Запросить с клавиатуры координаты точки (X, Y) и горизонтального отрезка прямой (Xn, Xk, Yn) и определить, лежит точка на отрезке прямой или нет. Сообщение об этом вывести на экран.

*Подсказка***.** Если координата Y точки не равна координате Y прямой, то НЕ лежит, если координата X точки не находится в пределах между Xn начала и Xk конца прямой, то точка НЕ лежит на прямой.

5. Написать программу вычисления подоходного налога по формулам:

а) при сумме менее 3000 рублей налог не взимается,

б) от 3000 до 20000 руб, берется 13% от суммы,

в) при сумме более 20000 руб берется 1400 рублей плюс 20% от суммы превышающей 20000 рублей.

6. Запросить с клавиатуры 3 стороны треугольника и по ним определить, является ли он прямоугольным, сообщение вывести на экран.

*Подсказка***.** Для каждой стороны применить теорему Пифагора и проверить, выполняется ли она. Если выполняется, то треугольник является прямоугольным. Переменные должны быть целыми числовыми. Функция квадрата - sqr(x), корня квадратного - sqrt(x). Программу проверить при сторонах 3, 4, 5 - прямоугольный, а 4, 5, 6 - не прямоугольный.

7. Запросить радиус круга R и сторона квадрата A. Определить, поместится ли круг в квадрате. Круг поместится в квадрате, если диаметр круга меньше или равен стороне квадрата.

8. Запросить радиус круга R и сторона квадрата A. Определить, поместится ли квадрат в круге. Квадрат поместится в круге, если диагональ квадрата меньше или равна диаметру окружности.

9. Написать программу для определения подходящего возраста для вступления в брак, используя следующее соображение: возраст девушки равен половине возраста мужчины плюс 7, возраст мужчины определяется соответственно как удвоенный возраст девушки минус 14. Данные для проверки работы программы задать самостоятельно.

10. Написать программу, контролирующую знание закона Ома. Обучаемый вводит формулу закона Ома в символьную переменную, которая далее сравнивается с правильным ответом, хранящимся в другой символьной переменной.

11. Написать программу вычисления значения функции y

y = x², если -2 <= x <= 2, y = 4 в остальных случаях.

12. Задать с помощью условного оператора следующие действия:

а) меньшее из двух значений переменных вещественного типа x и y заменить 0, а в случае их равенства заменить нулями оба;

б) наибольшее их трех различных значений переменных x, y, z уменьшить на 0.3

13. Найти сопротивление цепи из двух соединенных проводников. Сопротивления проводников и тип соединения запрашивать с клавиатуры. При последовательном соединении проводников R = R1 + R2, при параллелельном соединении проводников R = R1 * R2/(R1 + R2)

ГЛАВА 5. ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ. КОМАНДЫ ЦИКЛА С ПРЕДУСЛОВИЕМ И ПОСТУСЛОВИЕМ

Циклы используются, если одно и то же действие необходимо выполнить несколько раз. В языке Паскаль есть три возможности организовать цикл.

<table>
<tr><td colspan="3">Виды циклов</td></tr>
<tr><td>Цикл с предусловием
«ПОКА»
While… do</td><td>Цикл с постусловием
«ДО»
Repeat… until</td><td>Цикл
с параметром, известно точное число повторений</td></tr>
<tr><td> 
</td><td> 
</td><td>For i:=1 to N do
Begin
…
End;</td></tr>
</table>

ЦИКЛ «ПОКА» (WHILE)

Группа операторов (тело цикла) будет выполняться, ПОКА истинно условие цикла.

- Если условие изначально ложно, то тело цикла не будет выполнено ни разу.
- Если условие изначально истинно, и в теле цикла нет никаких действий, влияющих на истинность условия, то тело будет выполняться бесконечно, произойдет зацикливание.

```
WHILE <условие>  DO
Begin
```

<операторы>

End;

Пример1. Var x, y: real;

Begin

X:=5;

While x>5 do

Writeln(y); {цикл ни разу не выполнится}

While x=5 do

Writeln(y); {бесконечный цикл}

End.

Цикл while используется, когда неизвестно количество повторений. Для его организации нужно:

1. Правильно задать начальное значение переменной цикла.
2. Не забывать, что она должна изменяться во время выполнения цикла.
3. Правильно записать условие повторения.

Пример2. Var x, y: real;

Begin

X:=2;

While x<10 do

Begin

Y:=x*x;

X:=x+0.1

End;

End.

Пример3. While x<x1 do

Begin

Y:=a*x*x+b;

```
            Writeln(x, ‘  ‘, y);
            X:=x+dx;
        End;
```

Наиболее часто циклы с предусловием применяют для вычислений с заданной точностью, для организации поиска в массиве или файле.

Условие цикла может быть составным.

Задача. Покупатель имеет в распоряжении сумму S. В магазине он совершает покупки, стоимость которых последовательно вводится с клавиатуры. Определить, сколько покупок может совершить покупатель на имеющуюся сумму и сколько денег у него останется.

```
дано S (сумма)
        A (стоимость)
надо  K (количество покупок)
        O (остаток)
алг
        нач
            ввод S, A
            K:=0
            пока S>A
                S:=S-A
                K:=K+1
                ввод A
            кон
        вывод S, K
```

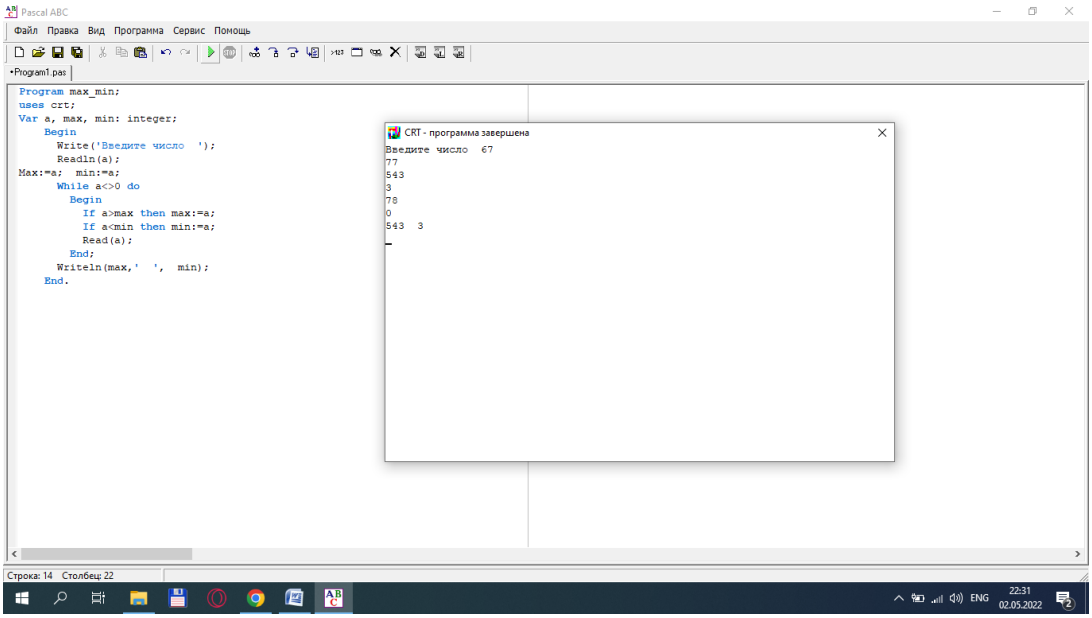
```
Program pokupki;
Var s, a, k: real;
```

```
Begin
    Writeln('Введите общую сумму');
    Readln(s);
Writeln('Введите стоимость покупки');
    Readln(a);
    K:=0;
    While s>=a do
        Begin
            s:=s-a;
            k:=k+1;
        Write('Введите стоимость покупки');
            Readln(a);
        End;
Writeln('Количество покупок  ', k);
Writeln('Оставшаяся сумма  ', s);
End.
```

Задача. С клавиатуры вводится последовательность чисел. Ввод заканчивается, когда пользователь введет 0. Определить max и min из введенных чисел.

```
Program max_min;
Var a, max, min: integer;
    Begin
        Write('Введите число  ');
        Readln(a);
       Max:=a;  min:=a;
        While a<>0 do
            Begin
              If a>max then max:=a;
              If a<min then min:=a;
```

```
        Read(a);
    End;
Writeln(max, min);
End.
```



ЦИКЛ «ДО» (REPEAT - UNTIL)

Проверка условия осуществляется после выполнения последовательности операторов (тела цикла). Действия будут выполняться до тех пор, пока условие ЛОЖНО. Как только оно станет истинным, цикл завершится.

```
REPEAT
        <операторы>
UNTIL <условие>;
```

- Составной оператор begin…end не требуется, так как сами слова repeat и until служат операторными скобками.
- Если условие изначально истинно, тело цикла все равно будет выполнено, хотя бы один раз. Именно это отличие циклов

«ДО» и «ПОКА» привело к тому, что они не подменяют друг друга, а используются каждый по своему назначению.

- Инструкцию until удобно использовать при разработке программ, обрабатывающих ввод с клавиатуры или из файла.

Пример 1.
```
Repeat
        Writeln(i);
        I:=i+1;
    Until i=10;
```

Пример 2.
```
Repeat
        read(n);
        sum:=sum+n;
    Until n=0;
```

Задача. Проверить введенное число на простоту.

```
Program prost;
Var n, d, r: integer;
    Begin
        Write('Введите целое число  ');
        Readln(n);
       d:=2;  {сначала будем делить на 2}
        repeat
            r:=n mod d;
            if r<>0 then d:=d+1 {n не разделилось нацело на d}
        until r=0; {пока не найдется число, на которое делится n}
        if d=n
       then
        writeln(n, '- простое')
       else
```

```
        writeln(n, '- составное');
    End.
```

Задачи для решения

1. Найти сумму цифр натурального числа N.

2. Протабулировать функцию с шагом 0,5: $f(x)=x^3 - 3x+11$, $-3<=x<=3$.

x	f(x)
-3	29
-2.5	…
-2	…
-1.5	…
…	…
3	11

3. С клавиатуры вводится последовательность чисел, в конце 0. Определить количество четных и нечетных.

4. Найти сумму положительных чисел, введенных с клавиатуры. Программа завершает свою работу, как только будет введено неположительное число.

5. Напечатать числа Фибоначчи, не превосходящие числа N. (1, 1, 2, 3, 5, 8, 13, 21,…).

6. Найти НОД(a, b) с помощью алгоритма Евклида. Например, (48, 15)=(3, 15)=(3, 0), значит, НОД(48, 15)=3.

Задачи для самостоятельного решения

1. Самостоятельно измените программу так, чтобы компьютер запрашивал желаемое количество попыток, и при превышении этого числа сообщал загаданное число и заканчивал программу.

2. В этой программе реализована игровая ситуация для проверки вашей реакции.

```
program press;
uses crt;
var q,w:integer;
begin
  writeln('После надписи "Жми" быстрей нажми любую клавишу');
  delay (2000);                          {задержка времени}
  writeln('Жми!!!');
  q:=0;
  while not keypressed do
    q:=q+1;                              {Если не нажата любая клавиша, то наращивать q}
  writeln('Пока вы соображали, прошло ',q,' циклов');
end.
```

Измените программу так, чтобы использовать цикл repeat.

ГЛАВА 6. ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ. ЦИКЛ С ПАРАМЕТРОМ

Цикл с параметром используется для повторения последовательности действий, если известно **точное число повторений**.

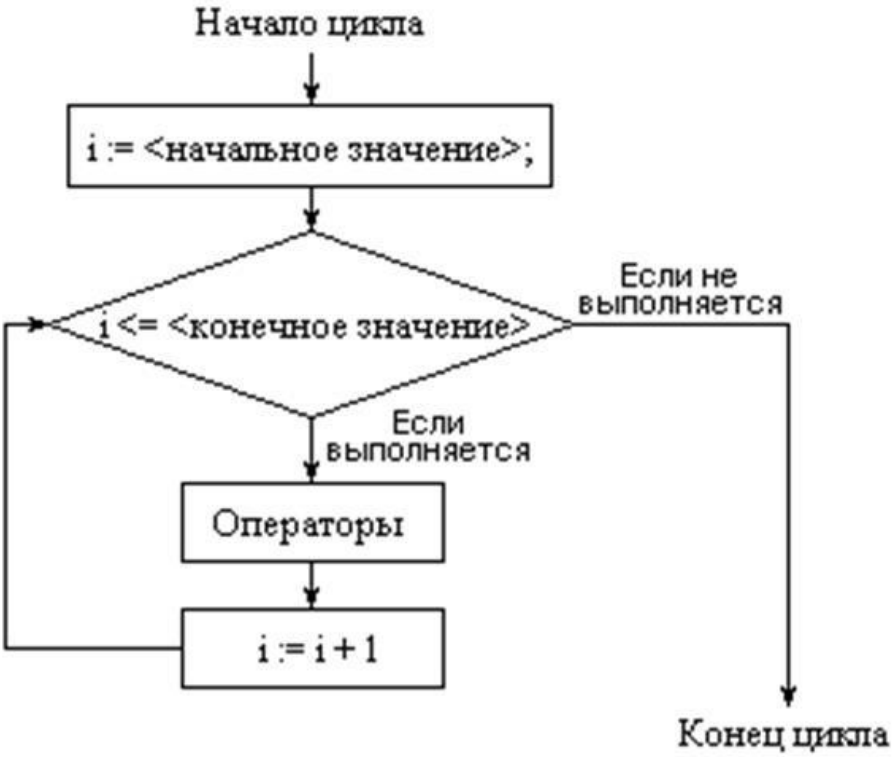
Имеет две основные формы:

```
FOR счетчик:= начальное значение  TO конечное значение DO
    Begin
            <операторы>
    End;
```

```
FOR счетчик:= начальное значение  DOWNTO конечное значение DO
    Begin
            <операторы>
    End;
```

- Переменная-счетчик имеет порядковый тип. Она меняется от начального значения до конечного с шагом 1 (-1).
- Основное использование данного оператора – обработка массивов.
- Если начальное значение счетчика больше конечного значения, то тело цикла ни разу не выполнится.
- Цикл ПОКА является универсальным, то есть любая задача, требующая повторений, может быть решена с его помощью. Циклы ДО и с параметром созданы для удобства программирования.

БЛОК-СХЕМА, СООТВЕТСТВУЮЩАЯ ИНСТРУКЦИИ FOR



Задача 1. Протабулировать функцию $y=2x^2-10$ на отрезке [-3; 3] с шагом 0,5.

```
x:=-3;
for i:=1 to 12 do begin
        y:= 2*x*x-10;
   x:=x+0.5;
        writeln(x, ‘  ‘, y)
   End;
```

x	y
-3	8
-2,5	2,5
-2	-2
-1,5	-5,5
…	…
2,5	2,5
3	8

Задача 2. Найти сумму квадратов натуральных чисел от 1 до 20.

ПОКА

```
Program poka;
    Var a, s: integer;
      Begin
          a:=1;
         s:=0;
          While a<=20 do
                Begin
                s:=s+a*a;
                a:=a+1
              end;
       writeln(s)
     End.
```

ДО

```
Program do;
    Var a, s: integer;
      Begin
          a:=1;
         s:=0;
          repeat
                s:=s+a*a;
              a:=a+1
          until a>20;
       writeln(s)
     End.
```

Цикл с параметром

```
Program parametr;
  Var a, s: integer;
    Begin
       s:=0;
       for a:=1 to 20 do
            s:=s+a*a;{один оператор, поэтому операторные скобки не нужны}
       writeln(s)
    End.
```

Задачи для решения

1. Вывести на экран числа **от 20 до 10 в строку**.

2. **Что выведется** на экран в результате работы программы? Заполнить таблицу.

a:=0; b:=1;
for i:= 3 to 6 do begin
a:=a+I; b:=b*i
end;
write(a, ‘ ‘, b);
	i	a	b

		0	1
	3		
	4		
	5		
	6		

3. Написать программу для вычисления суммы **пяти** элементов ряда: 1+1/2+1/3+1/4+1/5. Использовать форматный вывод. (Ответ: примерно 2.28).

4. Найти **ошибки** в программе. Что выведет программа при a=4?

```
Program factorial;
Var a, f: real;
i: integer;
Begin
        Write(‘Введите число =>’); readln(a);
        For  i:=1 to a do
              f:=f*i;
        Writeln(‘f= ’, f);
End.
```

5. С клавиатуры вводится **n** чисел. Найти среднее арифметическое **четных** чисел.

6. Дано число N. Выяснить, имеются ли среди чисел N, N+1,…2N **числа-близнецы** (простые числа, разность между которыми равна 2). Программа должна выводить пары этих чисел.

7. Натуральное число называется **совершенным**, если оно равно сумме всех своих делителей, за исключением самого себя. Напечатать все совершенные числа, меньшие N.

8. На отрезке [1, N] найти натуральное число, имеющее максимальную сумму делителей (само число в качестве делителя не учитывать).

9. Составить программу, определяющую, являются ли два числа взаимно простыми.

Задачи для самостоятельного решения

1. Вычислить сумму первых 10 натуральных чисел. Использовать счетчик типа S=S+A, т.е. счетчик суммы. Использовать управляющую переменную цикла.

2. Напечатать таблицу соответствия между весом в фунтах и весом в кг для значений от 1 до 10 фунтов с шагом 1 фунт. 1 фунт=400 г.

3. Напечатать таблицу перевода расстояний в дюймах в сантиметры (1 дюйм=2.54 см) для значений от 1 до 10 дюймов с шагом 1.

4. Напечатать таблицу перевода температуры по Фаренгейту в градусы по Цельсию от 15 до 30. Перевод осуществляется по формуле F=1.8*C+32.

5. Запросить с клавиатуры в цикле 5 любых целых чисел, найти их сумму и среднее арифметическое, результаты вывести на экран.

6. Напечатать все нечетные натуральные числа от 1 до 50 в столбик. Использовать управляющую переменную цикла.

7. Напечатать все четные, натуральные числа в диапазоне, заданном пользователем с клавиатуры в строку. Использовать управляющую переменную цикла.

8. Составить таблицу умножения для числа 12 в виде:

12 * 2 = 24

12 * 2 = 24

12 * 3 = 36 и т.д. до

12 * 10 = 120

9. Вычислить сумму квадратов первых 7 натуральных чисел.

ГЛАВА 7. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ. ЛИНЕЙНЫЕ МАССИВЫ

Определение: массив – это пронумерованный набор однотипных элементов, объединенных общим именем.

Массив объединяет компоненты одного типа в новый тип с помощью механизма индексации. Проще всего представлять массив в виде таблицы, где каждая величина находится в собственной ячейке.

СПОСОБЫ ОПИСАНИЯ МАССИВА

I. TYPE <Имя типа> = ARRAY[<Диапазон индексов>] OF <Тип элементов>;

- Индексы – это переменные порядкового типа.
- Начальный индекс не должен превышать конечного.
- Тип элементов массива может быть любым.

II. VAR <Переменная-массив>: ARRAY[<Диапазон индексов>] OF <Тип элементов>.

Примеры. Var

S, BB: array[1..40] of real;

N: array [‘A’.. ‘Z’] of integer;

R: array[-20..20] of string;

- Единственное действие, которое возможно произвести с массивом целиком, - присваивание. Например, S:=BB.
- Но с отдельными элементами массива можно работать так же. как с обычными переменными соответствующего типа.
- Обратиться к элементу массива можно при помощи указания имени всего массива и в квадратных скобках – индекса конкретного элемента. Например, R[10], BB[i], a[k].
- Возможность использования в качестве индекса значение любого порядкового типа – **мощное средство языка**. Если, например, необходимо связать информацию с буквами алфавита, то можно непосредственно использовать их в каче-

стве индексов, а не создавать параллельный массив с цифровой индексацией.

ЗАПОЛНЕНИЕ МАССИВА

Массиву может быть присвоено начальное значение с помощью типизированных констант. В отличие от обычных констант, их значение может быть изменено в программе.

Примеры. Type

```
  Col=(red, yellow, green);
Const
  Mas: array[1..10] of byte=(0, 45, 12, 5, 4, 9, 34, 89, 2, 1);
  Mas1: array [Boolean, col] of char=((‘a’, ‘b’, ‘c’), (‘1’, ‘2’, ‘3’));
```

ТИПИЧНЫЕ ДЕЙСТВИЯ С МАССИВАМИ

- Ввод массива.
- Вывод массива.
- Сортировка массива.
- Поиск в массиве заданного элемента.
- Поиск в массиве максимального или минимального элемента.

Пример.

```
Program vvod;
  var b: array[1..100] of integer;
  begin
    b[1]:=7;
    b[2]:=32;
    b[3]:=123;
    ……  { Трудоемкая задача? }
```

```
writeln(b[1]);
writeln(b[2]);
writeln(b[3]);
……
End.
```

В этом случае никакого преимущества массива перед обычными переменными не видно. Поэтому для работы с массивами используем другой способ.

```
Program vvod2;
var b: array[1..100] of integer;
    i: integer;
begin
    for i:=1 to 100 do
    readln(b[i]);
    for i:=1 to 100 do
    write(b[i], ‘  ‘);
    End.
```

Кстати, введение в язык Паскаль цикла с параметром было обусловлено во многом необходимостью обработки информационных последовательностей, то есть массивов.

ГЕНЕРАТОР СЛУЧАЙНЫХ ЧИСЕЛ

Удобно задавать массивы с помощью генератора случайных чисел. Случайные числа формирует функция RANDOM.

A:=random(x) – случайное число (если аргумент не указан, то результат real – число в интервале от 0 до 1, если x целое число от 0 до 65535, то результат случайное целое число в интервале от 0 до x-1;

Выражение, дающее целое случайное число в интервале [-50; 50], выглядит так:

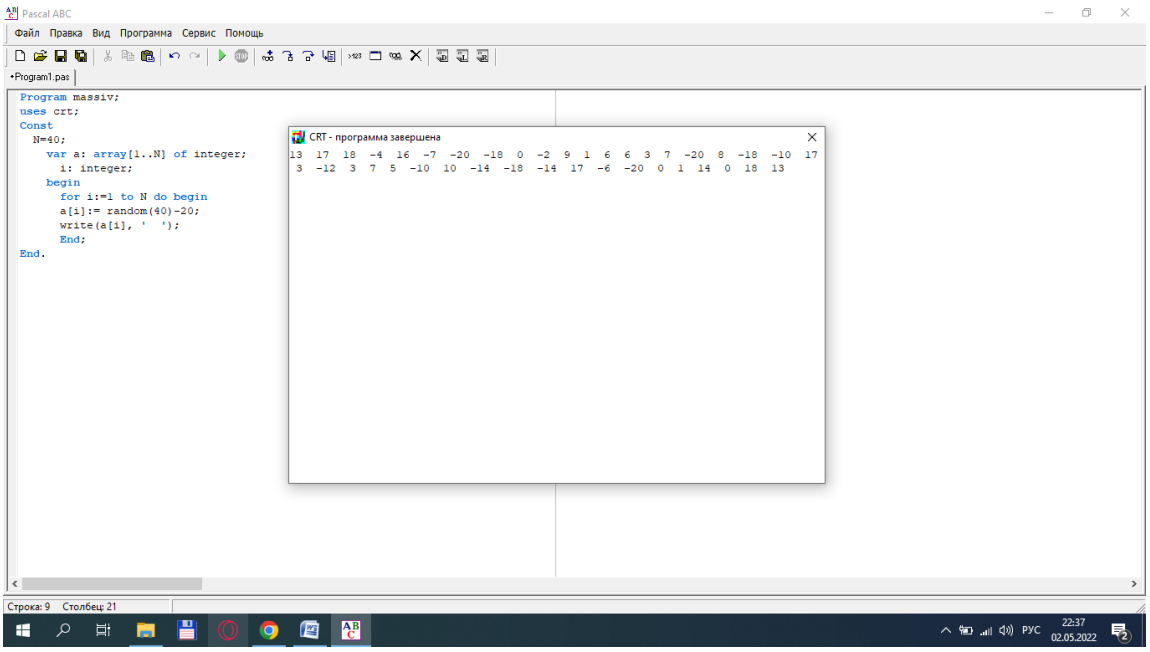
```
RANDOM(101)-50
```

Общая формула для генерации случайного числа из отрезка [A; B]:

X:=RANDOM(B-A+1)-A

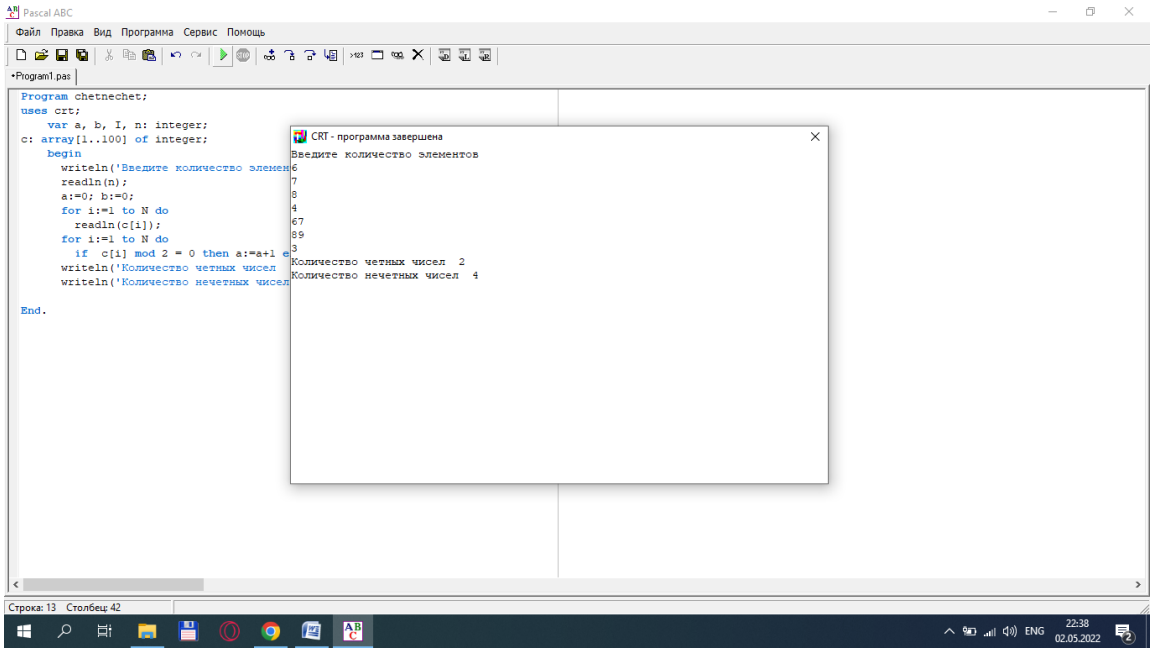
Зададим и распечатаем случайный массив из 20 чисел.

```
Program massiv;
Const
        N=20;
var a: array[1..N] of integer;
        i: integer;
begin
        for i:=1 to N do begin
        a[i]:= random*50-30;
        write(a[i], ‘  ‘);
        End;
End.
```



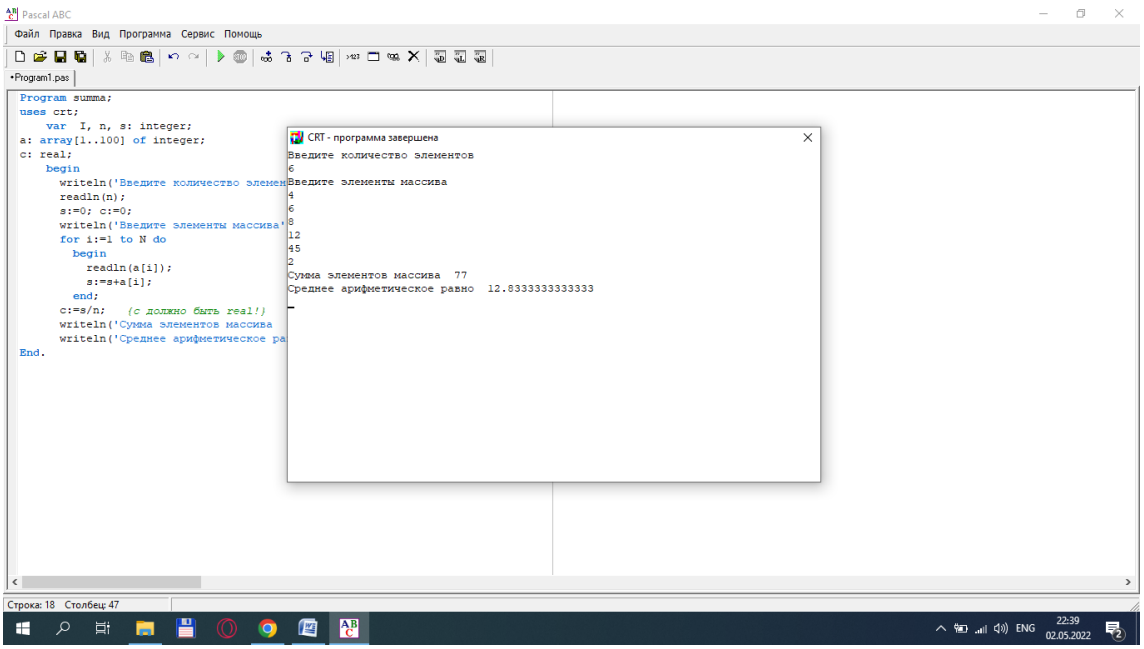
Задача. Подсчитать количество четных и нечетных элементов массива.

```
Program chetnechet;
var a, b, I, n: integer;
c: array[1..100] of integer;
begin
    writeln(‘Введите количество элементов’);
    readln(n);
    a:=0; b:=0;
    for i:=1 to N do
        readln(c[i]);
    for i:=1 to N do
        if  c[i] mod 2 = 0 then a:=a+1 else b:=b+1;
    writeln(‘Количество четных чисел  ’, a);
    writeln(‘Количество нечетных чисел  ’, b);

End.
```



Задача. Найти сумму и среднее арифметическое элементов массива.

```
Program summa;
var  I, n, s: integer;
a: array[1..100] of integer;
c: real;
begin
    writeln(‘Введите количество элементов’);
    readln(n);
    s:=0; c:=0;
    writeln(‘Введите элементы массива’);
    for i:=1 to N do
        begin
            readln(a[i]);
            s:=s+a[i];
        end;
    c:=s/n;   {c должно быть real!}
    writeln(‘Сумма элементов массива  ’, s);
    writeln(‘Среднее арифметическое равно  ’, c);
End.
```



Замечание.

Вид экрана при вводе элементов массива	Соответствующий фрагмент программы
A(1)= 2 <Enter> A(2)=5 <Enter> …	For i:=1 to n do begin Write(‘a**(**’, i, ‘**)**=’); readln(a[i]) End;

Задачи для решения

1. Ввести массив с клавиатуры. Найти минимальный и максимальный элементы массива (см. программу из Лекции 5).

2. Осуществить реверс массива.

3. Организовать ввод элементов массива с использованием ГСЧ. Найти наименьшее четное и наибольшее нечетное число в массиве.

4. Осуществить циклический сдвиг массива вправо.

2	7	6	5	4	**9**
9	2	7	6	5	4

5. Найти среднее арифметическое положительных элементов массива, стоящих на четных местах.

6. Дана целочисленная таблица A(N). Подсчитать наибольшее число одинаковых идущих подряд элементов. (1, 1, 1, 2, 3, 15, -3, 0, 0, 0, 0, 1, 7, 8).

7. Даны три целочисленные таблицы A(k), B(n), C(m). Известно, что существуют числа, встречающиеся во всех трех таблицах. Напечатать **одно** из таких чисел.

Задачи для самостоятельного решения

1. Описать числовой массив на 5 элементов и заполнить его присваиванием любыми числами, распечатать содержимое элементов массива

2. а) в столбик

3. б) в строку.

4. Описать числовой массив на 5 элементов и заполнить его в цикле с клавиатуры любыми числами, распечатать содержимое элементов массива.

5. Описать символьный массив на 5 элементов и заполнить его присваиванием именами, распечатать содержимое элементов массива.

6. Описать символьный массив на 5 элементов и заполнить его в цикле с клавиатуры именами, распечатать содержимое элементов массива.

7. Описать числовой массив на 25 элементов и заполнить его случайными целыми числами, каждое из которых лежит в пределах от 10 до 50, распечатать содержимое элементов массива в строку.

8. Описать числовой массив на 15 элементов и заполнить его случайными целыми числами, каждое из которых лежит в пределах от 10 до 100, распечатать содержимое элементов массива в строку, рассчитать среднее арифметическое элементов и вывести его на экран с поясняющим текстом, распечатать содержимое тех элементов массива, значение которых больше среднего арифметического.

9. Найти сумму элементов массива с четными номерами, содержащего 10 чисел.

10. Найти сумму положительных элементов заданного массива, содержащего 5 чисел.

11. Задано 2 массива, содержащих по 5 чисел. Сформировать новый массив, включая в него сначала все элементы первого массива, затем все элементы второго массива.

12. Задан массив, содержащий 10 чисел. Найти значение и индекс максимального (минимального) элемента.

13. Информация о температуре воздуха за месяц задана в виде массива. Определить, сколько раз температура опускалась ниже 0^0C.

14. Занести в массив карту расположения кораблей в игре "Морской бой" и смоделировать игру.

15. Задан массив, содержащий 10 чисел. Сформировать 2 массива, включая в массив первый четные (по номеру) элементы массива в порядке их следования, а во второй массив – нечетные.

16. Запросить с клавиатуры 5 слов, занести их в массив, определить количество в каждом из них знаков, занести их в другой массив, напечатать на экране содержимое обоих массивов в табличной форме.

17. Запросить с клавиатуры 5 слов, занести в массив только те слова, количество букв в которых равно четырем. Распечатать на экране содержимое массива.

18. Запросить с клавиатуры слово, определить количество в нем знаков, разрезать слово на отдельные буквы, которые занести в другой массив, распечатать на экране слово *справа налево*.

19. В заданном двумерном массиве поменять местами следующие два элемента:

a. Минимальный и максимальный.

b. Два наименьших.

c. Первый и последний положительные.

d. Два наибольших.

e. Два первых положительных.

f. Два последних отрицательных.

g. Два первых четных.

h. Два последних нечетных.

i. Минимальный положительных и максимальный отрицательный.

ГЛАВА 8. СОРТИРОВКИ МАССИВОВ

Сортировка – это упорядочивание элементов массива по возрастанию или убыванию.

АЛГОРИТМЫ СОРТИРОВОК

1. Простой выбор.
2. Простой обмен (Пузырьковая сортировка).
3. Простые вставки.

СОРТИРОВКА «ПРОСТОЙ ВЫБОР»

Дан массив A={a_1, a_2, …., a_n}. Расположим элементы массива по возрастанию. Алгоритм:

1. Рассмотрим элементы a_1, a_2, …., a_n. Среди них найдем наименьший элемент, запомнив его позицию k.
2. Произведем обмен значений элементов a_1 и a_k, то есть поставим наименьший элемент на первое место массива.
3. Рассмотрим элементы a_2, a_3, …., a_n. Среди них найдем наименьший элемент, запомнив его позицию k.
4. Произведем обмен значений элементов a_2 и a_k, то есть поставим наименьший элемент на второе место массива.

И т.д. пока не рассмотрим элементы a_{n-1} и a_n.

Пример. I. Находим минимальный элемент и меняем его местами с первым: **3**, 6, 2, 4, **1**, 2

II. 1, 6, 2, 4, **3**, 2

III. 1, 2, **6**, 4, 3, **2**

IV. 1, 2, 2, **4**, **3**, 6

V. 1, 2, 2, 3, 4, 6

```
Program Vybor;
const m=20;
```

```
var a: array[1..m] of integer;
I, j, x, n, k, min: integer;
begin
  writeln('Сортировка массива по возрастанию методом Простой выбор');
  write('Введите количество элементов массива  ');
  readln(n);
  //  Ввод элементов массива
  for i:=1 to n do
        begin
              write('A(', I, ')=');
              readln(a[i]);
        end;
  // Сортировка
  for i:=1 to n-1 do
        begin
        {поиск минимального элемента в части массива от a[i] до a[n]}
        k:=I;
              for j:=i+1 to n do
                    if a[j]<a[k] then k:=j;
        // Меняем местами a[k] и a[j]
        min:=a[i];
        a[i]:=a[k];
        a[k]:=min;
        // Выводим массив после каждого шага
              for x:= 1 to n do
                    write(a[x], '  ')
              writeln;
        end;
```

```
writeln('Массив отсортирован');
end.
```

СОРТИРОВКА «ПРОСТОЙ ОБМЕН (ПУЗЫРЬКОВАЯ)»

Дан массив A={a_1, a_2, …., a_n}. Расположим элементы массива по возрастанию. В основе алгоритма лежит обмен значений соседних элементов массива. Сортировать начинаем с конца массива. Каждый элемент, начиная с последнего, сравнивается с предыдущим, и если он меньше предыдущего, элементы меняются местами. Таким образом, элементы с меньшим значением продвигаются к началу массива («всплывают»), а элементы с большим значением – к концу массива («тонут»). Поэтому метод и получил свое второе название, метод пузырька. Этот процесс повторяется n-1 раз.

I. Рассмотрим элементы a_1, a_2, …., a_n.

1 шаг: Рассмотрим элементы a_{n-1} и a_n. Если $a_{n-1} > a_n$, то произведем обмен значений.

2 шаг: Рассмотрим элементы a_{n-2} и a_{n-1}. Если $a_{n-2} > a_{n-1}$, то произведем обмен значений.

И т.д.

n-1шаг: Рассмотрим элементы a_1 и a_2. Если $a_1 > a_2$, то произведем обмен значений.

В итоге всплыл один пузырек.

II. Рассмотрим элементы a_2, …., a_n.

1 шаг: Рассмотрим элементы a_{n-1} и a_n. Если $a_{n-1} > a_n$, то произведем обмен значений.

2 шаг: Рассмотрим элементы a_{n-2} и a_{n-1}. Если $a_{n-2} > a_{n-1}$, то произведем обмен значений.

И т.д.

n-1шаг: Рассмотрим элементы a_2 и a_3. Если $a_2 > a_3$, то произведем обмен значений.

……

N-1. Рассмотрим элементы a_{n-1} и a_n.

1 шаг: Рассмотрим элементы a_{n-1} и a_n. Если $a_{n-1} > a_n$, то произведем обмен значений.

В итоге всплыл n-1 пузырек. На этом алгоритм будет закончен, так как n-ый пузырек уже стоит на своем месте.

I=1	3	3	3	3	**1**
I=2	2	2	2	**1**	3
I=3	4	4	**1**	2	2
I=4	5	**1**	4	4	4
I=5	**1**	5	5	5	5

```
Program Obmen;
const m=20;
var a: array[1..m] of integer;
I, j, n, k, min: integer;
begin
 writeln('Сортировка массива по возрастанию методом Простой обмен');
 write('Введите количество элементов массива  ');
 readln(n);
 //  Генерация случайным образом элементов массива и вывод на экран
 for i:=1 to n do
        begin
              a[i]:=random(50)-20;
              write(a[i], '  ');
        end;
        writeln;
 // Сортировка
 for i:=2 to n do
```

```
begin
    for j:=n downto i do
        if a[j]<a[j-1] then
            begin
            // Меняем местами a[j] и a[j-1]
                min:=a[j];
                a[j]:=a[j-1];
                a[j-1]:=min;
            end;
end;
// Выводим отсортированный массив
writeln('Массив отсортирован');
for i:= 1 to n do
    write(a[i], ' ')
end.
```

СОРТИРОВКА «ПРОСТЫЕ ВСТАВКИ» (ВКЛЮЧЕНИЯ)

1. При сортировке вставкой сначала упорядочиваются два элемента массива.
2. Затем делается **вставка** третьего элемента в соответствующее место по отношению к первым двум элементам.
3. Затем делается **вставка** четвертого элемента в список из трех элементов.
4. Процесс повторяется до тех пор, пока все элементы не будут упорядочены.

Плюсы данного алгоритма сортировки:

- простота реализации,
- эффективность метода на частично упорядоченных последовательностях.

8 **5** 0 3 7

5 8 **0** 3 7

0 5 8 **3** 7

0 3 5 8 **7**

0 3 5 7 8

```
Program Vstavki;
var mas: array[1..100] of integer;
I, j, n, temp, nom: integer;
begin
  writeln('Сортировка массива по возрастанию методом Простые вставки');
  write('Количество элементов массива=>');
  readln(n);
  //  Ввод с клавиатуры элементов массива
  for i:=1 to n do
        begin
              write('Введите элемент номер  ', I, '  ');
              readln(mas[i]);
        end;
        writeln;
  // Сортировка
  for i:=1 to n-1 do
        begin
              nom:=i+1;
              temp:=mas[nom];
                    for j:=i+1 downto 2 do
                          begin
                                if temp<mas[j-1] then
                                      begin
```

```
                                    mas[j]:=mas[j-1];
                                    nom:=j-1;
                                    end;
                            end;
                        mas[nom]:=temp;
        end;
        // Выводим отсортированный массив
        writeln('Массив отсортирован');
        for i:= 1 to n do
                write(mas[i], '  ')
end.
```

ГНОМЬЯ СОРТИРОВКА

Гномья сортировка впервые была предложена 2 октября 2000 года Хамидом Сарбази-Азадом (Hamid Sarbazi-Azad). Он назвал ее «Глупая сортировка, простейший алгоритм сортировки всего с одним циклом…». И действительно, глупый этот метод или нет, но в нем задействован, не как в большинстве сортировок – два или более циклов, а только один. Позже, голландский ученый Дик Грун, на страницах одной из своих книг, привел для гномьей сортировки следующую аналогию.

Гномья сортировка основана на технике, используемой обыкновенным голландским садовым гномом. Вот как садовый гном сортирует ряд цветочных горшков. По существу, он смотрит на два соседних цветочных горшка, если они находятся в правильном порядке, то

он переходит на один горшок вперед, иначе он меняет их местами и возвращается на один горшок назад. Граничные условия: если позади нет ни одного горшка – он шагает вперед, а если нет следующего горшка, тогда он закончил.

Так описал основную идею алгоритма гномьей сортировки Дик Грун. Заменим гнома и горшки на более формальные объекты, например на указатель (номер текущего элемента) и элементы массива, и рассмотрим принцип работы алгоритма еще раз.

Вначале указатель ставится на 2-ой элемент массива.

Затем происходит сравнение двух соседних элементов A[i] и A[i-1], т. е. сравниваются первый элемент (i-1) и второй (i). Если те стоят на своих позициях, то сдвигаем указатель на позицию i+1 и продолжаем сравнение с нового положения; иначе меняем элементы местами, и, поскольку в какой-то момент может оказаться, что элементы в левом подмассиве стоят не на своих местах, сдвигаем указатель на одну позицию влево, осуществляя следующее сравнение с новой позиции. Так алгоритм продолжает выполняться до тех пор, пока весь массив не будет упорядочен.

Здесь следует выделить два важных момента.

Во-первых, процесс упорядочивания заканчивается тогда, когда не выполняется условие i<n, где n – размер массива.

Во-вторых, как было сказано, указатель перемещается как вперед по списку, так и назад, и в том случае если он окажется над первым элементом, его сразу же следует сместить на одну позицию вправо, т. к. в противном случае придется сравнивать два элемента, одного из которых не существует.

Кажется немного необычным, тот факт, что алгоритм сортировки использует всего один цикл. Плюс к этому, на практике гномья сортировка не уступает в скорости сортировки вставками.

Код программы на Pascal:

```
program Gnome_sort;
uses crt;
```

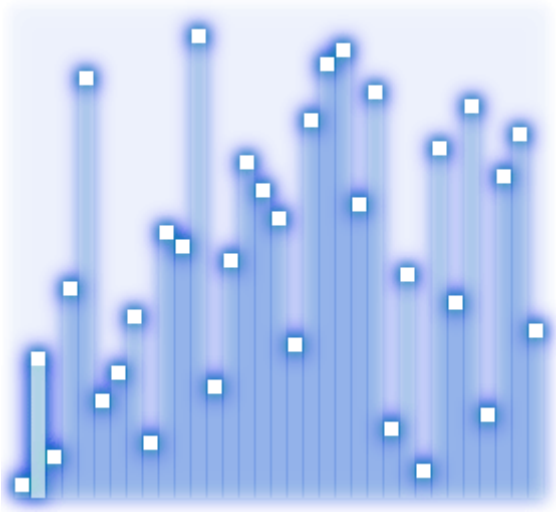
```
type massiv=array[1..100] of integer;
var k, m, n: integer;
a: massiv;
procedure PGnome_sort(i, j: integer; mas: massiv);
var t: integer;
begin
while i<=n do
begin
if mas[i—1]<=mas[i] then begin i:=j; j:=j+1; end
else
begin
t:=mas[i];
mas[i]:=mas[i—1]; mas[i—1]:=t;
i:=i—1;
if i=1 then begin i:=j; j:=j+1; end;
end;
end;
write(‘Отсортированный массив: ‘);
for i:=1 to n do write(mas[i], ‘ ‘); {вывод массива}
end;
begin
clrscr;
write(‘Размер массива > ‘); read(n);
for k:=1 to n do {ввод массива}
begin
write(k, ‘ элемент > ‘); read(a[k]);
end;
k:=2; m:=3;
```

```
PGnome_sort(k, m, a); {вызов процедуры сортировки}
readkey;
end.
```

ШЕЙКЕРНАЯ СОРТИРОВКА (ПЕРЕМЕШИВАНИЕ)

Рассматриваемый алгоритм имеет несколько непохожих друг на друга названий. Среди них: сортировка перемешиванием, двунаправленная пузырьковая сортировка, шейкерная сортировка, пульсирующая сортировка (ripple sort), трансфертная сортировка (shuttle sort), и даже сортировка «счастливый час» (happy hour sort).

Второй вариант (двунаправленная пузырьковая сортировка) наиболее точно описывает процесс работы алгоритма. Здесь, в его название, довольно-таки удачно включен термин «пузырьковая». Это действительно альтернативная версия известного метода, модификации в котором заключаются, по большей части, в реализации, упомянутой в названии, двунаправленности: алгоритм перемещается, не как в обменной (пузырьковой) сортировке – строго снизу вверх (слева направо), а сначала снизу вверх, потом сверху вниз.



http://kvodo.ru/shaker-sort.html

Перестановка элементов в шейкерной сортировке выполняется аналогично той же в пузырьковой сортировке, т. е. два соседних элемента, при необходимости, меняются местами. Пусть массив требуется упорядочить по возрастанию. Обозначим каждый пройденный путь

от начала до конца последовательности через Wi, где i – номер пути; а обратный путь (от конца к началу) через —Wj, где j – номер пути.

Тогда после выполнения Wi, один из неустановленных элементов будет помещен в позицию справа, как наибольший из еще неотсортированных элементов, а после выполнения —Wj, наименьший из неотсортированных, переместится в некоторую позицию слева. Так, например, после выполнения W1 в конце массива окажется элемент, имеющий наибольшее значение, а после —W1 в начало отправится элемент с наименьшим значением.

Код программы на Pascal:

```
program CocktailSort;
uses crt;
type Massiv=array[1..100] of integer;
var
n, k: integer;
A: Massiv;
{процедура шейкерной сортировки}
procedure ShakerSort(Mas: Massiv; Start, m: integer);
var Left, Right, temp, i: integer;
begin
Left:=Start;
Right:=m;
while Left<=Right do
begin
for i:=Right downto Left do
if (Mas[i-1]>Mas[i]) then
begin
temp:=Mas[i];
Mas[i]:=Mas[i-1];
Mas[i-1]:=temp;
```

```
end;
Left:=Left+1;
for i:=Left to Right do
if Mas[i-1]>Mas[i] then
begin
temp:=Mas[i];
Mas[i]:=Mas[i-1];
Mas[i-1]:=temp;
end;
Right:=Right-1;
end;
for i:=1 to M do write(' ', Mas[i]);
end;
{основной блок программы}
begin
clrscr;
write('Размер массива > '); read(n);
for k:=1 to n do
begin
write(k, ' элемент > '); read(A[k]);
end;
write('Результирующий массив: ');
ShakerSort(A, 2, n);
end.
```

СОРТИРОВКА ШЕЛЛА

В 1959 году американский ученый Дональд Шелл опубликовал алгоритм сортировки, который впоследствии получил его имя – «Сортировка Шелла». Этот алгоритм может рассматриваться и как обобщение пузырьковой сортировки, так и сортировки вставками.

Идея метода заключается в сравнение разделенных на группы элементов последовательности, находящихся друг от друга на некотором расстоянии. Изначально это расстояние равно d или N/2, где N — общее число элементов. На первом шаге каждая группа включает в себя два элемента расположенных друг от друга на расстоянии N/2; они сравниваются между собой, и, в случае необходимости, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние d сокращается на d/2, и количество групп, соответственно, уменьшается. Постепенно расстояние между элементами уменьшается, и на d=1 проход по массиву происходит в последний раз.

Далее, на примере последовательности целых чисел, показан процесс сортировки массива методом Шелла. Для удобства и наглядности, элементы одной группы выделены одинаковым цветом.

Исходный массив	5	3	8	0	7	4	9	1	6	2
d=5	4	3	1	0	2	5	9	8	6	7
d=2	1	0	2	3	4	5	6	7	9	8
d=1	0	1	2	3	4	5	6	7	8	9

Ссылка на демонстрацию алгоритма сортировки Шелла посредством танца:

https://forkettle.ru/vidioteka/programmirovanie-i-set/algoritmy-i-struktury-dannykh/108-sortirovka-i-poisk-dlya-chajnikov/1008-sortirovka-shella.

Первое значение, соответствующее расстоянию d равно 10/2=5. На каждом шаге оно уменьшается вдвое. Элементы, входящие в одну группу, сравниваются и если значение какого-либо элемента, стоящего левее того с которым он сравнивается, оказывается больше (сортировка по возрастанию), тогда они меняются местами. Так, элементы путем внутригрупповых перестановок постепенно становятся на свои позиции, и на последнем шаге (d=1) сортировка сводится к проходу по одной группе, включающей в себя все N элементов массива. При этом число требуемых обменов оказывается совсем небольшим.

Псевдокод

```
ЗАДАЧА Шелл(a=: РЯД ИЗ ЦЕЛ);
ПЕР
  b,i,j,k,h: ЦЕЛ;
УКАЗ
  b:=РАЗМЕР(a);
  k:=b ДЕЛИТЬ 2;
  ПОКА k>0 ВЫП
    ОТ i:=1 ДО b-k ВЫП
      j:=i;
      ПОКА (j>=1) И (a[j]>a[j+k]) ВЫП
        h:=a[j];
        a[j]:=a[j+k];
        a[j+k]:=h;
        УМЕНЬШИТЬ(j);
      КОН;
    КОН;
    k:=k ДЕЛИТЬ 2
  КОН
КОН Шелл;
```

Код программы на Pascal:

```
program ShellSorting;
uses crt;
type massiv=array[1..100] of integer;
var i, j, n, d, count: integer;
A: massiv;
procedure Shell(A: massiv; n: integer); {сортировка Шелла}
```

```
begin
d:=n;
d:=d div 2;
while (d>0) do
begin
    for i:=1 to n-d do
        begin
            j:=i;
                while ((j>0) and (A[j]>A[j+d])) do
                        begin
                          count:=A[j];
                          A[j]:=A[j+d];
                          A[j+d]:=count;
                          j:=j-1;
                        end;
        end;
        d:=d div 2;
end;
writeln;
for i:=1 to n do write(' ', A[i]); {вывод массива}
end;
{основной блок программы}
begin
write('Размер массива > '); read(n);
for i:=1 to n do {ввод массива}
begin write(i, ' элемент > '); readln(A[i]);
end;
```

Задачи для решения

1. Заполнить таблицу для алгоритма «Простые вставки».
2. Массив: 8, 5, 0, 3, 7. Как будет изменяться массив?

i	nom	temp	j
1	2	5	2
…	…	…	…

3. Подсчитать количество различных чисел, встречающихся в целочисленной таблице.
4. Организовать генерацию элементов массива случайным образом. Какой элемент встречается в массиве чаще остальных?

ГЛАВА 9. ПОИСК В МАССИВЕ

При решении многих задач возникает необходимость установить, содержит ли массив определенную информацию или нет.

Например, проверить, есть ли в массиве фамилий студентов – Петров. Задачи такого типа называются **поиском в массиве**.

Для организации поиска могут быть использованы различные алгоритмы. Наиболее простой – это алгоритм **простого перебора**. Поиск осуществляется последовательным сравнением элементов массива с образцом до тех пор, пока не будет найден элемент, равный образцу, или не будут проверены все элементы. Алгоритм простого перебора применяется, если элементы массива не упорядочены.

АЛГОРИТМ ПРОСТОГО ПЕРЕБОРА

Перебор элементов осуществляет инструкция REPEAT, в теле которой инструкция IF сравнивает текущий элемент массива с образцом и присваивает переменной found значение TRUE, если текущий элемент равен образцу.

Цикл завершается, если в массиве обнаружен элемент, равный образцу, или если проверены все элементы массива. По завершению цикла по значению переменной found можно определить, успешен поиск или нет.

```
Program Poisk_perebor;
var massiv: array[1..20] of integer;
I, obrazec: integer;
Found: Boolean;
begin
    writeln('Простой перебор');
    //  Ввод элементов массива
    for i:=1 to n do
        begin
            write('A(', I, ')=');
```

```
                readln(a[i]);
            end;
    write('Введите образец для поиска (целое число) ');
    readln(obrazec);
    // поиск простым перебором
    Found:=FALSE;
    I:=1; { проверяем с первого элемента массива }
            Repeat
                If massiv[i]=obrazec then found:= TRUE else
i:=i+1;
        Until (i>20) or (found); {завершаем, если совпадение с об-
    разцом или проверены все элементы массива}
    If found then
        writeln('Совпадение с элементом номер', i:3)
    else
        writeln ('Совпадений нет');
    end.
```

1. Очевидно, что чем больше элементов в массиве и чем дальше расположен нужный элемент от начала массива, тем дольше будет программа искать нужный элемент.

2. Данный алгоритм может использоваться для поиска как в числовых, так и в строковых массивах.

На практике часто проводится поиск в массиве, элементы которого **упорядочены** по некоторому критерию. Например, массив фамилий упорядочен по алфавиту, массив данных о погоде упорядочен по датам наблюдения.

Для поиска в упорядоченном массиве применяются другие, более эффективные по сравнению с методом простого перебора, алгоритмы, например, **метод бинарного поиска**.

МЕТОД БИНАРНОГО ПОИСКА

Суть метода: выбирается средний (по номеру) элемент упорядоченного массива, и с этим элементом сравнивается образец.

verh	1	**-5**
	2	-1
	3	0
	4	2
sred	**5**	**13**
	6	44
	7	70
	8	75
niz	9	**91**

	1	-5
	2	-1
	3	0
	4	2
	5	13
verh	6	**44**
sred	**7**	**70**
	8	75
niz	9	**91**

1) Если средний элемент равен образцу, то задача решена.

2) Если средний элемент **больше** образца, то искомый элемент расположен ВЫШЕ среднего элемента (между элементами с номерами verh и srerd).

3) Если средний элемент **меньше** образца, то искомый элемент расположен НИЖЕ среднего элемента (между элементами с номерами srerd и niz).

После того как определена часть массива, в которой может располагаться искомый элемент, поиск проводят в этой части, выделяя новый средний элемент. Номер среднего элемента вычисляется по формуле

(niz-verh)/2 + verh

```
Program Poisk_BINAR;
var a: array[1..9] of integer;
I, obrazec, sred, verh, niz, n: integer;
Found: Boolean;
begin
```

```
        writeln('Бинарный поиск в массиве);
            for i:=1 to 9 do
            begin
                write('A(', I, ')=');
                readln(a[i]);
            end;
    write('Введите образец для поиска (целое число) ');
    readln(obrazec);
    verh:=1;  niz:=9;  n:=0;
    Found:=FALSE;
    Writeln('  verh  niz  sred  ');
            Repeat
                Sred:=(niz-verh) div 2 + verh;
                Writeln(verh:6, niz:6, sred:6);
                N:=n+1;
                If a[sred]= obrazec then found:= TRUE
                Else begin
                        If obrazec<a[sred] then niz:=sred-1
                        Else verh:=verh+1;
                    End;
        Until (verh> niz) or found; {завершаем, если совпадение с образцом или проверены все элементы массива}
    If found then
    writeln('Совпадение с элементом номер', sred, '  Выполнено  ', n, ' сравнений.')
    else
    writeln ('Образец не найден.);
    end.
```

ПОИСК МИНИМАЛЬНОГО (МАКСИМАЛЬНОГО) ЭЛЕМЕНТА МАССИВА

Алгоритм поиска минимального (максимального) элемента в неупорядоченном массиве довольно очевиден: делается предположение, что первый элемент является минимальным (максимальным), затем остальные элементы последовательно сравниваются с этим элементом. Если во время очередной проверки обнаруживается, что проверяемый элемент меньше (больше) принятого за min (max), то этот элемент принимается на минимальный (максимальный) и продолжается проверка оставшихся элементов.

```
Program Poisk_min;
Const granica=10;
var a: array[1..granica] of integer;
min, I: integer;
begin
    //  Ввод элементов массива
    for i:=1 to n do
        begin
            write('A(', I, ')=');
            readln(a[i]);
        end;
min:=1;
for i:=2 to granica do
    if a[i]<a[min] then min:=I;
writeln('Мин. элемент массива: ', a[min]);
writeln('Номер элемента: ', min);
end.
```

ГЛАВА 10. ДВУМЕРНЫЕ МАССИВЫ

Определение: массив – это пронумерованный набор однотипных элементов, объединенных общим именем.

Массив объединяет компоненты одного типа в новый тип с помощью механизма индексации. Проще всего представлять массив в виде таблицы, где каждая величина находится в собственной ячейке.

СПОСОБЫ ОПИСАНИЯ МАССИВА

TYPE <Имя типа> = ARRAY[<Диапазон индексов>] OF <Тип элементов>;

- Индексы – это переменные порядкового типа.
- Начальный индекс не должен превышать конечного.
- Тип элементов массива может быть любым.

VAR <Переменная-массив>: ARRAY[<Диапазон индексов>] OF <Тип элементов>.

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

Рассмотрим двумерный массив Паскаля размерностью 3*3, то есть в ней будет три строки, а в каждой строке по три элемента:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Каждый элемент имеет свой номер, как у одномерных массивов, но сейчас номер уже состоит из двух чисел – номера строки, в кото-

рой находится элемент, и номера столбца. Таким образом, номер элемента определяется пересечением строки и столбца. Например, a21 – это элемент, стоящий во второй строке и в первом столбце.

ОПИСАНИЕ ДВУМЕРНОГО МАССИВА

Существует несколько способов объявления двумерного массива Паскаля.

Мы уже умеем описывать одномерные массивы, элементы которых могут иметь любой тип, а, следовательно, и сами элементы могут быть массивами. Рассмотрим следующее описание типов и переменных:

Пример описания двумерного массива Паскаля

```
Type
Vector = array [1..5]
Matrix= array [1..10] of vector;
Var m: matrix;
```

Мы объявили двумерный массив Паскаля m, состоящий из 10 строк, в каждой из которых 5 столбцов. При этом к каждой i -й строке можно обращаться m [i], а каждому j -му элементу внутри i -й строки – m [i , j].

Определение типов для двумерных массивов Паскаля можно задавать и в одной строке:

```
Type
Matrix= array [1..5] of array [1..10] of < тип элементов >;
```

или еще проще:

```
type
matrix = array [1..5, 1..10] of <тип элементов>;
Var m: matrix;
```

Описание массива в разделе переменных:

```
const
   M = 10;
   N = 5;
var
   a: array [1..M, 1..N] of integer;
```

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

ОСНОВНЫЕ ДЕЙСТВИЯ С ДВУМЕРНЫМИ МАССИВАМИ

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

```
type
matrix= array [1..5, 1..10] of integer;
var
  a , b : matrix ;
```

то в ходе выполнения программы можно присвоить матрице A значение матрицы B (A:= B). Все остальные действия выполняются **поэлементно**, при этом над элементами можно выполнять все допустимые операции, которые определены для типа данных элементов массива. Это означает, что если массив состоит из целых чисел, то над его элементами можно выполнять операции, определенные для целых чисел, если же массив состоит из символов, то к ним применимы операции, определенные для работы с символами.

ВВОД ДВУМЕРНОГО МАССИВА В ПАСКАЛЕ

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

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

Рассмотрим пример ввода двумерного массива Паскаля с клавиатуры:

Пример программы ввода двумерного массива Паскаля с клавиатуры

```
type
  matrix= array [1..5, 1..10] of integer;
var
  a: matrix;
  i, j: integer; { индексы массива }
begin
  for i :=1 to 5 do {цикл для перебора всех строк}
     for j :=1 to 10 do {перебор всех элементов строки по столбцам}
      readln ( a [ i , j ]); {ввод с клавиатуры элемента, стоящего в i -й строке и j -м столбце}
end.
```

ВЫВОД ДВУМЕРНОГО МАССИВА НА ЭКРАН

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

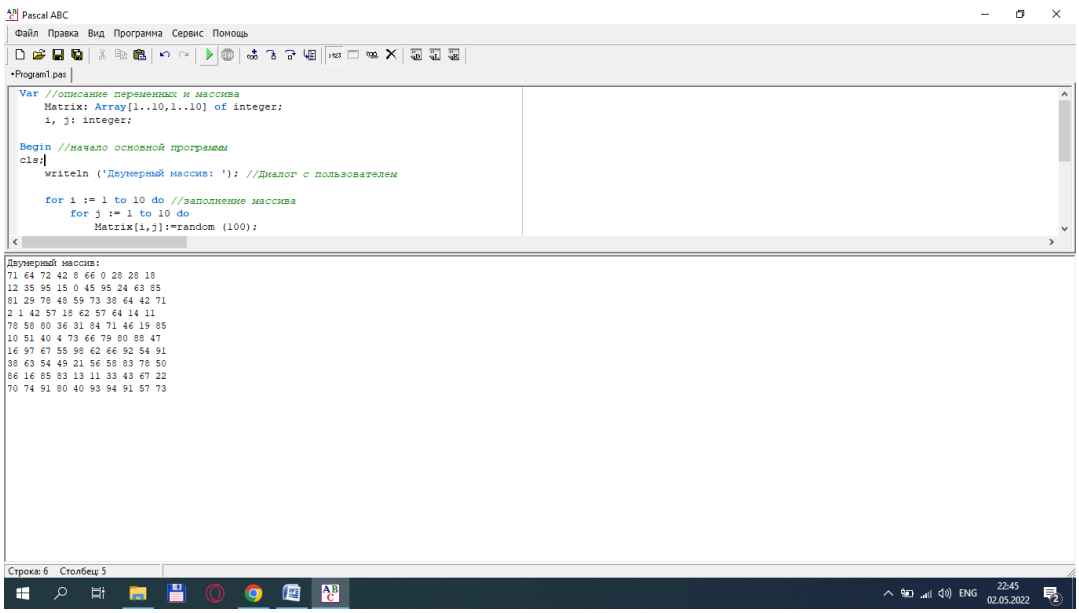
Пример программы вывода двумерного массива

```
for i :=1 to 5 do {цикл для перебора всех строк}
begin
  for j :=1 to 10 do {перебор всех элементов строки по столбцам}
    write ( a [ i , j ]:4); {печать элементов, стоящих в i -й строке матрицы в одной экранной строке, при этом для вывода каждого элемента отводится 4 позиции}
    writeln; {прежде, чем сменить номер строки в матрице, нужно перевести курсор на начало новой экранной строки}
end ;
```

Пример программы с использованием двумерного массива, в котором мы заполняем массив случайными числами и выводим его на экран:

```
Var //описание переменных и массива
  Matrix: Array[1..10,1..10] of integer;
  i, j: integer;

Begin //начало основной программы
  writeln ('Двумерный массив: '); //Диалог с пользователем
```

```
    for i := 1 to 10 do //заполнение массива
        for j := 1 to 10 do
            Matrix[i,j]:=random (100);

    for i := 1 to 10 do
        begin //Вывод массива
            for j := 1 to 10 do
                write (matrix[i,j], ' ');
            writeln

            end; //Конец программы
end.
```

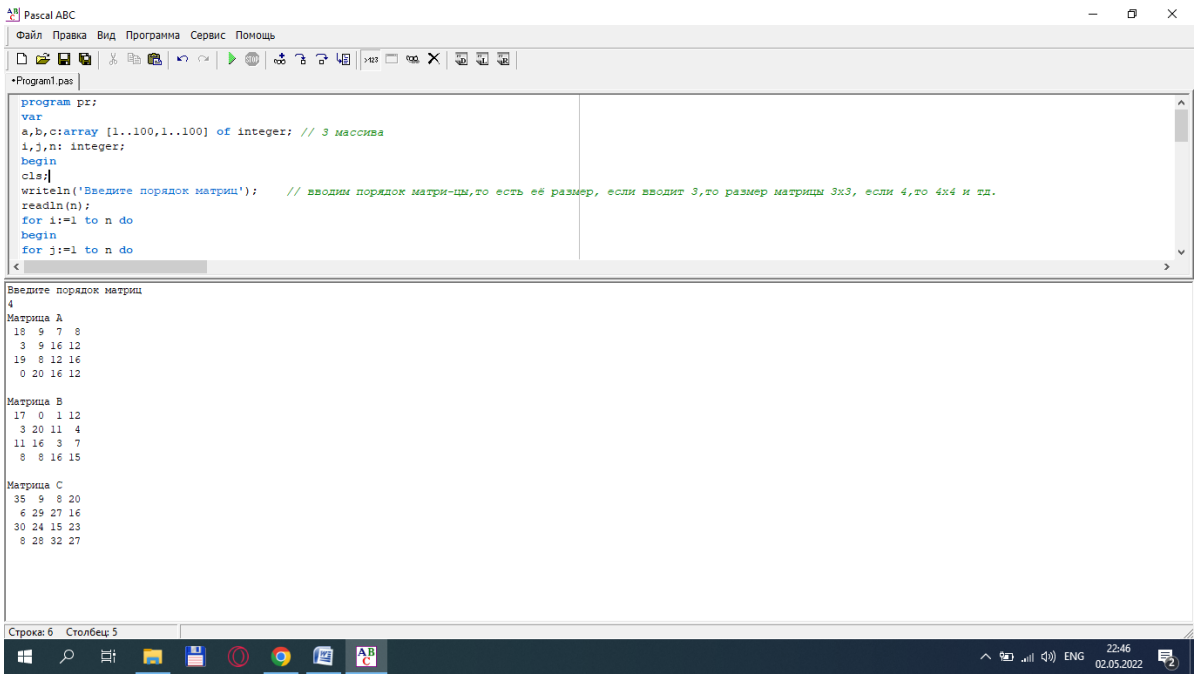


Сумма матриц

```
program pr;
var
a,b,c:array [1..100,1..100] of integer; // 3 массива
```

```
i,j,n: integer;
begin
writeln('Введите порядок матриц');     // вводим порядок матрицы,то есть её размер, если вводит 3,то размер матрицы 3x3, если 4,то 4x4 и тд.
readln(n);
for i:=1 to n do
begin
for j:=1 to n do
   a[i,j]:=random(21);   // матрица заполнена случайными числами от 0 до 20
end;
writeln('Матрица A');
for i:=1 to n do
begin                  // выводим матрицу на экран
   for j:=1 to n do
      write(a[i,j]:3);
   writeln;
end;
for i:=1 to n do       // так же и со 2 матрицей
begin
   for j:=1 to n do
   b[i,j]:=random(21);
end;
writeln;
writeln('Матрица B');
for i:=1 to n do
   begin
      for j:=1 to n do
```

```
            write(b[i,j]:3);
        writeln;
    end;        // теперь создадим 3 матрицу,каждый элемент которой равно сумме соответ эл. 2 других матриц
    for i:=1 to n do
      for j:=1 to n do
        c[i,j]:=a[i,j]+b[i,j];
    writeln;
    writeln('Матрица C');  // выводим 3 матрицу как предыдущие
    for i:=1 to n do
      begin
      for j:=1 to n do
            write(c[i,j]:3);
        writeln;
      end;
    end.
```



Задачи для решения

1. Дан массив 4*4 случайных, целых чисел от -50 до 50.

a) Вывести сумму всех элементов массива.

b) Напечатать количество отрицательных элементов каждого столбца.

c) Вывести максимальное число каждой строки.

d) Вывести на экран измененный массив, в котором останутся только простые числа, а остальные заменить на 0.

e) Заменить в и исходном массиве все элементы выше главной диагонали на 0.

2. Даны два двумерных массива 4*4 целых чисел. Вывести третий массив, где на местах совпавших элементов, останутся эти элементы, а на всех остальных - сумма соответствующих элементов.

3. Даны матрицы A и B, размерностей N*M и M*K соответственно. Вывести на экран матрицу C=A*B (произведение матриц A и B).

4. Дана матрица N*M, состоящая из натуральных чисел. Найти в ней максимальный элемент. Если их несколько, то вывести на экран положение каждого из них.

5. Упорядочить элементы столбцов матрицы N*M по возрастанию.

6. Дана прямоугольная матрица A(K*L). Получить новую матрицу путем умножения всех элементов данной матрицы на наименьший по абсолютной величине.

ГЛАВА 11. ПОДПРОГРАММЫ: ПРОЦЕДУРЫ И ФУНКЦИИ

Определение: Подпрограмма - специальным образом оформленный блок программы, для дальнейшего его многократного использования.

Подпрограммы решают очень важные задачи, значительно облегчая программирование:

1) избавляют от необходимости многократно повторять в тексте программы аналогичные фрагменты;

2) улучшают структуру программы, облегчая ее понимание при разборе;

3) повышают устойчивость к ошибкам программирования и непредвиденным последствиям при модификациях.

СТРУКТУРА ТЕКСТА ПОДПРОГРАММЫ

соответствует структуре текста основной программы:

```
<заголовок подпрограммы>;
<раздел описаний>;
begin
<тело подпрограммы>
end;
```

Следует обратить внимание на два отличия в описании основной программы и подпрограммы:

- подпрограмма начинается с заголовка, содержащего имя подпрограммы, передаваемые в нее и возвращаемые от нее параметры. Заголовок подпрограммы отличается от заголовка основной программы;
- подпрограмма заканчивается точкой с запятой, а не точкой.

Подпрограммы бывают двух видов:

1) Функции

2) Процедуры

Функция - это подпрограмма, которая возвращает какое-либо значение.

Процедура - это подпрограмма, которая сразу изменяет значения и выполняет операции с числами, операторами и т.д.

Заголовки процедуры и функции имеют вид:

```
function <имя_функции> (<список формальных параметров>): <тип результата>;
procedure <имя_процедуры> (<список формальных параметров>);
```

Список формальных параметров – это имена переменных с указанием их типов, над которыми подпрограмма осуществляет действия.

Пример заголовков процедуры и функции:

```
procedure primer1 (x, y: real; z: integer);
function primer2 (n, m: byte): integer;
```

У функций и процедур существуют **параметры** (переменные, которые передают какое - либо значение). Они тоже бывают двух видов:

1) **Формальны**е - те, которые находятся в описании подпрограммы

2) **Фактически**е - те, которые передаются из основной программы в функцию или процедуру

Также у подпрограммы существую переменные, с которыми она в дальнейшем работает. Они делятся опять же на два типа:

1) **Глобальные** переменные, то есть действующие во всей программе.

2) **Локальные** - те, которые действуют только в процедуре или функции.

Переменные, объявленные в основной программе (глобальные), доступны всем инструкциям программы, в том числе и инструкциям процедур и функций.

Переменные, объявленные в подпрограммах (локальные), доступны только инструкциям той подпрограммы, в которой они объявлены.

Имена локальных переменных могут совпадать с именами глобальных переменных. Если в подпрограмме описана (локализована) переменная с тем же именем, что и глобальная, то в этом случае на время действия подпрограммы локальная переменная «закрывает» глобальную и делает ее недоступной, т.е. значение глобальной переменной на время действия подпрограммы «замораживается».

Память для локальных, т.е. описанных в подпрограмме, переменных выделяется на время действия подпрограммы в специальной области, называемой **стеком**. При завершении подпрограммы память освобождается, поэтому все внутренние результаты работы подпрограммы не сохраняются от одного обращения к другому.

Вызов подпрограммы происходит при каждом употреблении ее имени в основной программе. При вызове подпрограммы начинают выполняться ее операторы до завершающего слова **end** или специального выхода из подпрограммы – **exit**. После выхода из подпрограммы управление передается команде, следующей за обращением к подпрограмме.

При вызове подпрограммы (процедуры или функции) после ее имени в скобках указываются **фактические параметры.** Фактическими параметрами могут быть как конкретные значения, так и переменные или даже результат выражения. Последовательность фактических параметров, их количество и их тип должны точно совпадать с формальными параметрами подпрограммы. Значения фактических параметров на момент обращения к подпрограмме должны быть определены.

ФУНКЦИИ

Функции - подпрограммы, которые возвращают значение, то есть не изменяют переменные в основной программе, а выполняет действие, которое в дальнейшем можно использовать многократно.

Рассмотрим структуру функции на примере:

```
function Prim(a, b, c: integer): integer; //Задание переменных в функцию из программы
var
  M: integer; //Описание локальной/ных переменных
begin
  A := B + C;
  M := A / B; //Присваивание локальной переменной значение A/B
  A := A * C * B * M;
  Prim := A; //Возвращение значения
end; //Конец функции
```

Данная функция выполняет простые операции с числами, в предпоследней строчке мы видим, как происходит **возвращение значения**.

ПАРАМЕТР-ЗНАЧЕНИЕ И ПАРАМЕТР-ПЕРЕМЕННАЯ

Передаваться значения параметров могут по **значению** и по **ссылке** (параметр-переменная).

При передаче параметров по значению в подпрограмме создаются переменные, в соответствии с объявлениями в заголовке. Эти переменные существуют только на время работы подпрограммы.

При вызове процедуры фактические параметры, передаваемые **по значению**, могут быть записаны как константы, переменные или выражения.

В начале выполнения подпрограммы копии значений этих констант, переменных и выражений присваиваются временным перемен-

ным подпрограммы. Когда подпрограмма завершается, используемые в ней переменные не сохраняются. Это значит, что передача параметров **по значению** имеет смысл для передачи данных в подпрограмму, но не для передачи полученных результатов из нее.

Пример описания формальных параметров, передаваемых **по значению**:

```
procedure primer (x, y: integer; z: real);
```

При передаче параметров по ссылке основная программа передает в подпрограмму не значение переменной, а ссылку на нее, следовательно, память под передаваемую по ссылке переменную не отводится. Подпрограмма, производя некоторые действия с переменной, принявшей ссылку на место памяти основной программы, где расположена переменная, в действительности производит действия с переменной основной программы. Поэтому после завершения процедуры изменения, совершенные с переменной основной программы, сохраняются. Этот механизм используется для получения результатов от подпрограммы.

При вызове процедуры параметры, передаваемые по ссылке, не могут быть заданы ни константой, ни выражением, т.к. они не имеют адреса для передачи. Параметры, передаваемые по ссылке, при вызове процедуры задаются **только переменной.**

В заголовке процедуры список параметров, принимающих значения по ссылке, начинается служебным словом **var**.

Пример описания формальных параметров, передаваемых **по ссылке**:

```
procedure primer1 (var k,l: byte; var d: integer);
```

КАК ВЫЗВАТЬ ФУНКЦИЮ В ПРОГРАММЕ

Для того, чтобы вызвать функцию, мы должны либо присвоить её значение какой-либо переменной, либо совершить с ней какую-то операцию, либо вывести с помощью write/writeln.

Пример вызова для нашей вышеописанной функции:

```
k:=Prim(a, b, c); // Присваивание значения функции переменной, где k-любая переменная, описанная в программе
k:=Prim(a,b,c)+365; // Совершение операции с функцией, к значению функции прибавляем 365
write (Prim(a,b,c)); // Вывод значения функции
```

ПРОЦЕДУРЫ

Процедуры - подпрограммы, которые не возвращают значение, являются отрезком программы для дальнейшего его использования.

Рассмотрим структуру процедуры на примере:

```
procedure Prim(var a, b, c: integer);
var
  M: integer; //Описание локальной/ных переменных
begin
  A := B + C;
  M := A / B; //Присваивание локальной переменной значение A/B
  A := A * C * B * M;
  write(A); // Вывод значения переменной в процедуре
end; //Конец процедуры
```

Главные отличия процедуры от функции:

- Нет описания типа возвращаемого значения (в первой строке после скобок стоит сразу точка с запятой);
- Не возвращает значение, а значит, имени процедуры ничего не присваивается.

Передаваться значения могут как по ссылке, так и по значению.

ВЫЗОВ ПРОЦЕДУРЫ ИЗ ОСНОВНОЙ ПРОГРАММЫ

Для вызова процедуры нам не нужно присваивать её чему-либо, выводить с помощью writeln и т.д., ведь у процедуры нет определённого значения. Для вызова процедуры необходимо написать имя процедуры и формальные параметры, указанные в первой строке в том же количестве и того же типа! Пример:

```
Prim(a,b,c);
```

Для вызова процедуры важно, чтобы параметры были тех же типов и шли в том же порядке.

Например:

```
var
  a, k, m: integer;

begin
  read(a);
  k := 1;
  m := 1;
  Prim(a, k, m);
end.
```

В данном случае мы значение a присваиваем a, значение k присваиваем b, а значение m присваиваем c. Если бы k было типа real, то и b должно быть типа real.

Пример программы, использующей функцию и процедуру вместе, а также с разными типами:

```
program primer;

var
  a, k: integer; // Описание глобальных переменных
  m: real;
```

```
function PrimFun(a, b: integer; c: real): real;
var
  M: real; //Описание локальной переменной
begin
  A := B + A;
  M := A / B; //Присваивание локальной переменной значения A/B
  C := A * C * B * M;
  PrimFun := C; //Возвращение значения
end; //Конец функции

procedure PrimProc(var a, b: integer; c: real);
var
  M: real; //Описание локальной/ных переменных
begin
  A := B + A;
  M := A / B; //Присваивание локальной переменной значения A/B
  C := A * C * B * M;
end; //Конец процедуры

begin
  read(a, k);
  m := 1;
  m := PrimFun(a, k, m); //Присваивание значения функции переменной
m
  PrimProc(a, k, m); // Вызов процедуры
  writeln(a, k, m); //Вывод полученных значений
end.
```

Приступая к решению задач, следует помнить, что:

- *Для передачи данных в подпрограмму следует использовать параметры. Глобальные переменные применять не рекомендуется;*

- *При написании подпрограммы-функции среди ее инструкций должен быть хотя бы один оператор присваивания вида <имя_функции>:= выражение;*

- *Тип каждого фактического параметра (константы, выражения или переменной) при вызове подпрограммы должен совпадать с типом формального параметра, указанного в заголовке подпрограммы;*

- *Если в заголовке подпрограммы перед именем формального параметра не стоит слово var, то при вызове подпрограммы в качестве фактического параметра можно использовать константу, переменную или выражение соответствующего типа;*

- *Если параметр подпрограммы используется для возврата результата в основную программу, то в заголовке перед ним ставится слово var;*

- *Если в заголовке подпрограммы перед именем формального параметра стоит слово var, то при вызове подпрограммы в качестве фактического параметра можно использовать только переменную соответствующего типа;*

Решение задачи с использованием подпрограмм

Задача: треугольник задан координатами своих вершин. Составить программу вычисления его площади.

Решение этой задачи удобно представить в виде следующих этапов:

1. Ввод последовательно координат трех вершин: x1,y1,x2,y2,x3,y3.
2. Вычисление длины первой стороны

3. Вычисление длины второй стороны
4. Вычисление длины третьей стороны
5. Вычисление площади по формуле Герона
6. Вывод на экран полученного значения площади.

Из приведенного выше плана решения задачи видно, что в пунктах 2-4 будут выполняться одни и те же действия только с разными координатами вершин. Удобно оформить эти вычисления в виде **подпрограммы**. Так как длина стороны – это вещественное число, то ее можно вычислять с помощью **подпрограммы-функции**. При этом для начала можно продумать лишь название функции, ее параметры и тип результата, а детали реализации записать чуть позже. Иначе говоря, следует пока написать «пустую» функцию, чтобы понять для начала ход решения основной задачи.

Итак, пусть наша функция называется storona. Для вычисления стороны треугольника нужно передать в подпрограмму координаты двух вершин. Пусть формальные параметры называютсяa1,b1,a2,b2 и имеют вещественный тип (real). Тип результата функции также будет вещественным.

Тогда начальный вариант программы будет выглядеть следующим образом:

```
program treugolnik;


var
  S, a, b, c, p: real; {описываем переменные для вычисления площади (S), сторон (a,b,c), периметра (p)}
  X1, y1, x2, y2, x3, y3: real;{описываем переменные – координаты вершин}


function storona(a1, b1, a2, b2: real): real;{заголовок функции, в которой будет вычисляться длина стороны треугольника}
begin
```

```
end;{конец функции. Пока тело функции пусто, детали вычислений запишем чуть позже. Зато к этой функции уже можно обращаться из основной программы}


begin{начало основной программы}
  Writeln('Введите координаты первой вершины');
  Readln(x1, y1);
  Writeln('Введите координаты второй вершины');
  Readln(x2, y2);
  Writeln('Введите координаты третьей вершины');
  Readln(x3, y3);
  A := storona(x1, y1, x2, y2); {вычисляем длину первой стороны}
  B := storona(x2, y2, x3, y3); {вычисляем длину второй стороны}
  C := storona(x3, y3, x1, y1); {вычисляем длину третьей стороны}
  P := a + b + c; {вычисляем периметр}
  S := sqrt(p / 2 * (p / 2 - a) * (p / 2 - b) * (p / 2 - c));
  Writeln('площадь треугольника равна:', s);
end.
```

После того, как в основной программе выделены подзадачи, продуман план решения, решен основной вопрос, какие данные передаются в подпрограмму и что должно быть получено в результате ее работы, можно детально продумать алгоритм работы подпрограммы.

Для вычисления длины стороны в функции можно описать локальную (определенную только внутри функции) переменную вещественного типа, пусть она называется d.

```
function storona(a1, b1, a2, b2: real): real;
var
  d: real;{описание переменной, которая будет использоваться исключительно внутри функции}
begin
```

```
  d := sqrt(sqr(a1 - a2) + sqr(b1 - b2));
  Storona := d; {обязательный в теле функции оператор присваивания, когда с именем функции связывается вычисленное значение}
end;
```

Вот теперь наша задача полностью решена.

Эту же задачу можно решить с использованием **подпрограммы-процедуры**. Начинать решение задачи рекомендуется так же, как и в предыдущем случае. Сначала определить название процедуры, передаваемые в нее параметры, способ получения результата из процедуры. Тело процедуры пока можно сделать пустым. Затем написать основную программу, в которой определить точки обращения к процедуре и значения фактических параметров.

Пусть наша процедура называется dlina. Для вычисления длины стороны треугольника в процедуру необходимо передать значения координат двух вершин, поэтому в списке формальных параметров в заголовке процедуры нужно описать 4 параметра-значения вещественного типа: a1,b1,a2,b2. Для передачи полученного значения в основную программу нужно описать в заголовке параметр-переменную, принимающий ссылку, назовем этот параметр d (необходимо помнить, что в заголовке процедуры перед этим параметром ставится служебное слово var).

Тогда начальный вариант программы будет выглядеть следующим образом:

```
program treugolnik;


var
  S, a, b, c, p: real; {описываем переменные для вычисления площади (S), сторон (a,b,c), периметра (p)}
  X1, y1, x2, y2, x3, y3: real;{описываем переменные – координаты вершин}
```

```
procedure dlina(a1, b1, a2, b2: real; vard: real);{заголовок процедуры, в которой будет вычисляться длина стороны треугольника}
begin
end;{конец процедуры. Пока тело процедуры пусто, детали вычислений запишем чуть позже. Зато к этой процедуре уже можно обращаться из основной программы}


begin{начало основной программы}
  Writeln('Введите координаты первой вершины');
  Readln(x1, y1);
  Writeln('Введите координаты второй вершины');
  Readln(x2, y2);
  Writeln('Введите координаты третьей вершины');
  Readln(x3, y3);
  dlina(x1, y1, x2, y2, a); {вычисляем длину первой стороны, при этомx1,y1,x2,y2 будут передавать копии своих значений в процедуру, а переменная a будет принимать полученный результат}
  dlina(x2, y2, x3, y3, b); {вычисляем длину второй стороны, при этомx2,y2,x3,y3 будут передавать копии своих значений в процедуру, а переменная b будет принимать полученный результат }
  dlina(x3, y3, x1, y1, c); {вычисляем длину третьей стороны, при этомx3,y3,x1,y1 будут передавать копии своих значений в процедуру, а переменная c будет принимать полученный результат}
  P := a + b + c; {вычисляем периметр}
  S := sqrt(p / 2 * (p / 2 - a) * (p / 2 - b) * (p / 2 - c));
  Writeln('площадь треугольника равна:', s);
end.
```

Обратите еще раз внимание на различие в вызове подпрограммы-функции и подпрограммы-процедуры. Обращение к функции использовалось в операторе присваивания как выражение, а обращение к процедуре – это оператор.

Теперь можно наполнить содержанием саму процедуру:

```
procedure dlina(a1, b1, a2, b2: real; var d: real);
begin
  D := sqrt(sqr(a1 - a2) + sqr(b1 - b2));
end;
```

Задачи для решения

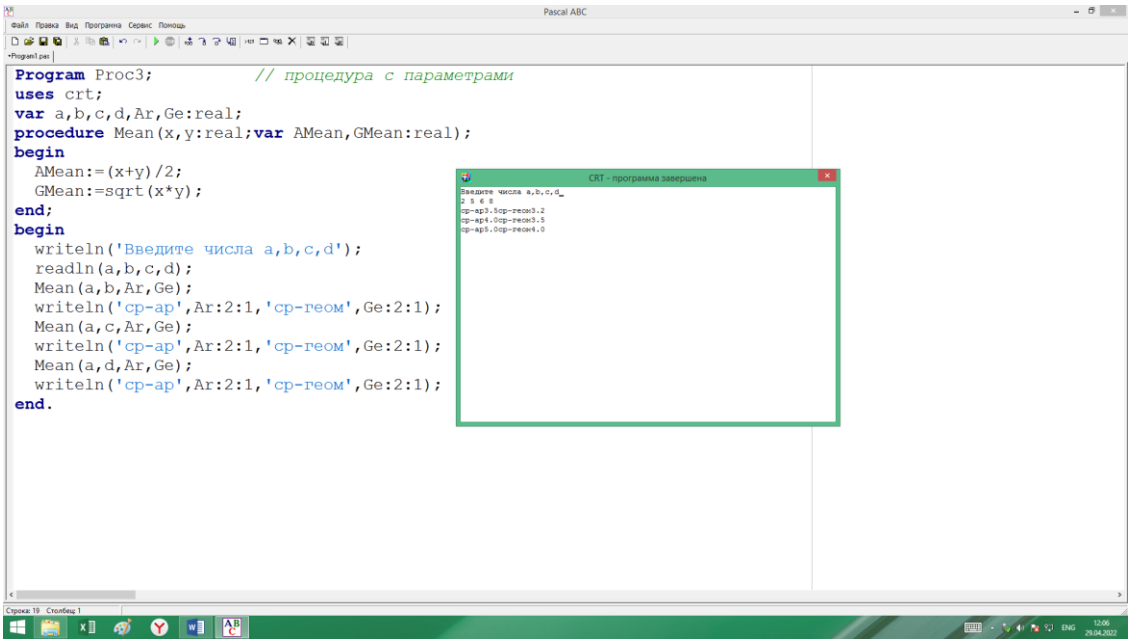
1. На данном отрезке [N, 2N] распечатать все числа-близнецы. Функция проверяет, является ли число простым (возвращает значение логического типа, истина/ложь).
2. В линейном массиве найти элемент с максимальной суммой делителей (функция возвращает сумму делителей своего аргумента).
3. В линейном массиве подсчитать количество различных элементов. Использовать процедуру ввода элементов массива, процедуру сортировки массива (любой алгоритм сортировки) и функцию, возвращающую количество различных элементов массива.
4. Дана прямоугольная матрица. Из данной матрицы получить новую путем умножения на максимальный по модулю элемент (процедуры ввода и ывода матрицы, функция, возвращающая максимальный по модулю

Пример решения задач из Электронного задачника:

```
Program Proc1;      // процедура с параметрами
uses crt;
var i: integer;
   t ,c:real;
procedure powerA3(a:real;var b:real); {формальные параметры процедуры}
begin
  b:=power(a,3);       {a - число, c - степень}
  end;
begin                  {основная программа}
```

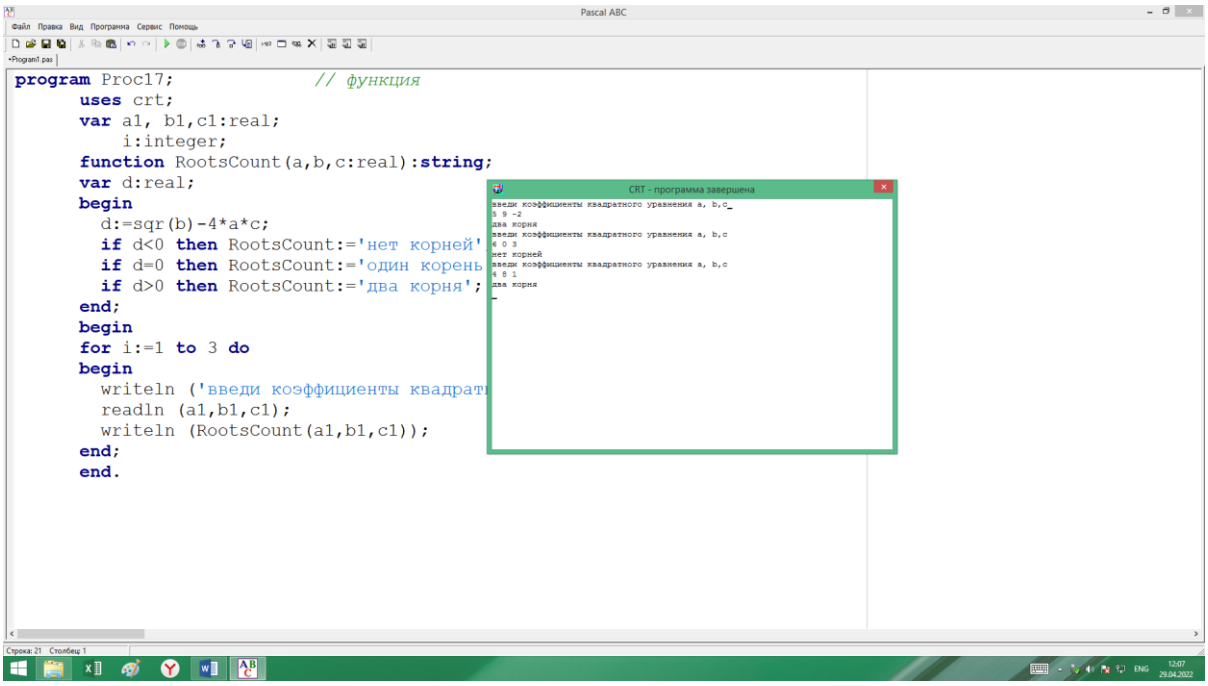
```
  writeln('Вычисление 3-й степени числа:');
  for i:=1 to 5 do
  begin
    write('введите число  ');
    readln(t);
    powerA3(t,c);       {фактические параметры процедуры}
  writeln(3-я степень числа:',c:2:1);
  end;
end.

Program Proc3;          // процедура с параметрами
uses crt;
var a,b,c,d,Ar,Ge:real;
procedure Mean(x,y:real;var AMean,GMean:real);
begin
  AMean:=(x+y)/2;
  GMean:=sqrt(x*y);
end;
begin
  writeln('Введите числа a,b,c,d');
  readln(a,b,c,d);
  Mean(a,b,Ar,Ge);
  writeln('ср-ар',Ar:2:1,'ср-геом',Ge:2:1);
  Mean(a,c,Ar,Ge);
  writeln('ср-ар',Ar:2:1,'ср-геом',Ge:2:1);
  Mean(a,d,Ar,Ge);
  writeln('ср-ар',Ar:2:1,'ср-геом',Ge:2:1);
end.
```

```
program Proc16;         // функция
    uses crt;
    var a, b:real;
    function sign(x:real):integer;
    begin
    if x<0 then sign:=-1;
    if x=0 then sign:=0;
    if x>0 then sign:=1;
    end;
    begin
    writeln ('vv a i b');
    readln (a,b);
    writeln ('sign(a)+sign(b) = ',sign(a)+sign(b));
    end.
```




```
program Proc17;          // функция
    uses crt;
    var a1, b1,c1:real;
```

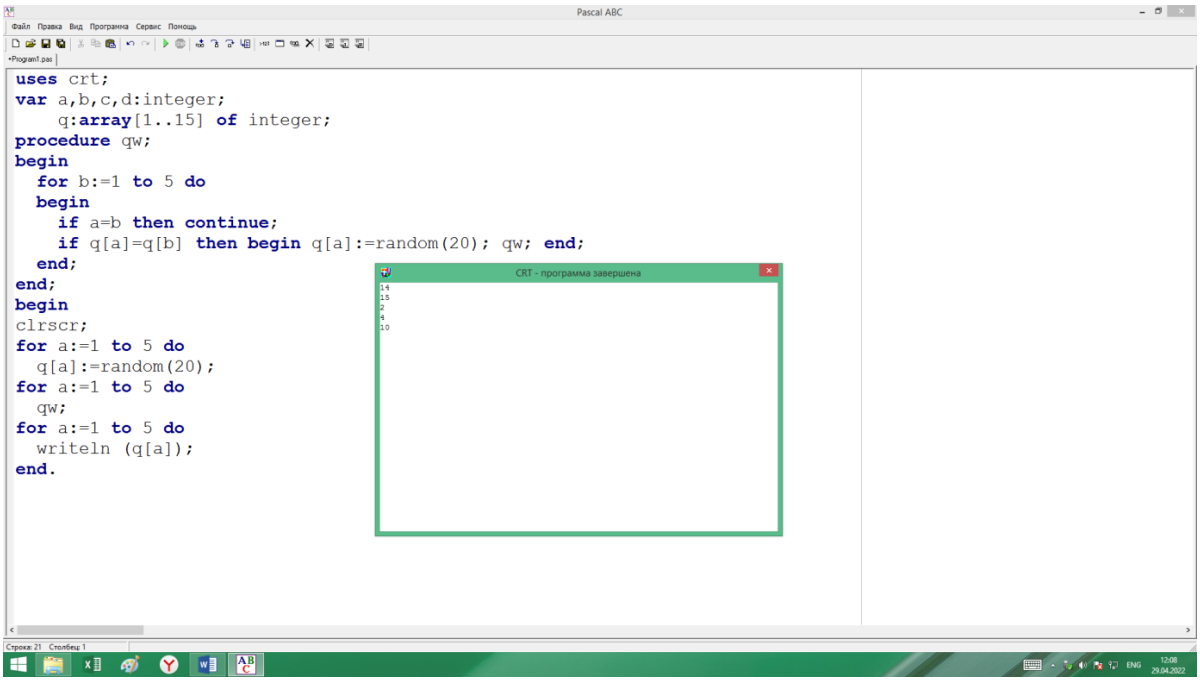
```
    i:integer;
function RootsCount(a,b,c:real):string;
var d:real;
begin
  d:=sqr(b)-4*a*c;
  if d<0 then RootsCount:='нет корней';
  if d=0 then RootsCount:='один корень';
  if d>0 then RootsCount:='два корня';
end;
begin
for i:=1 to 3 do
begin
  writeln ('введи коэффициенты квадратного уравнения a, b,c');
  readln (a1,b1,c1);
  writeln (RootsCount(a1,b1,c1));
end;
end.
```



Наберите и отладьте следующие программы, разберитесь в их работе:

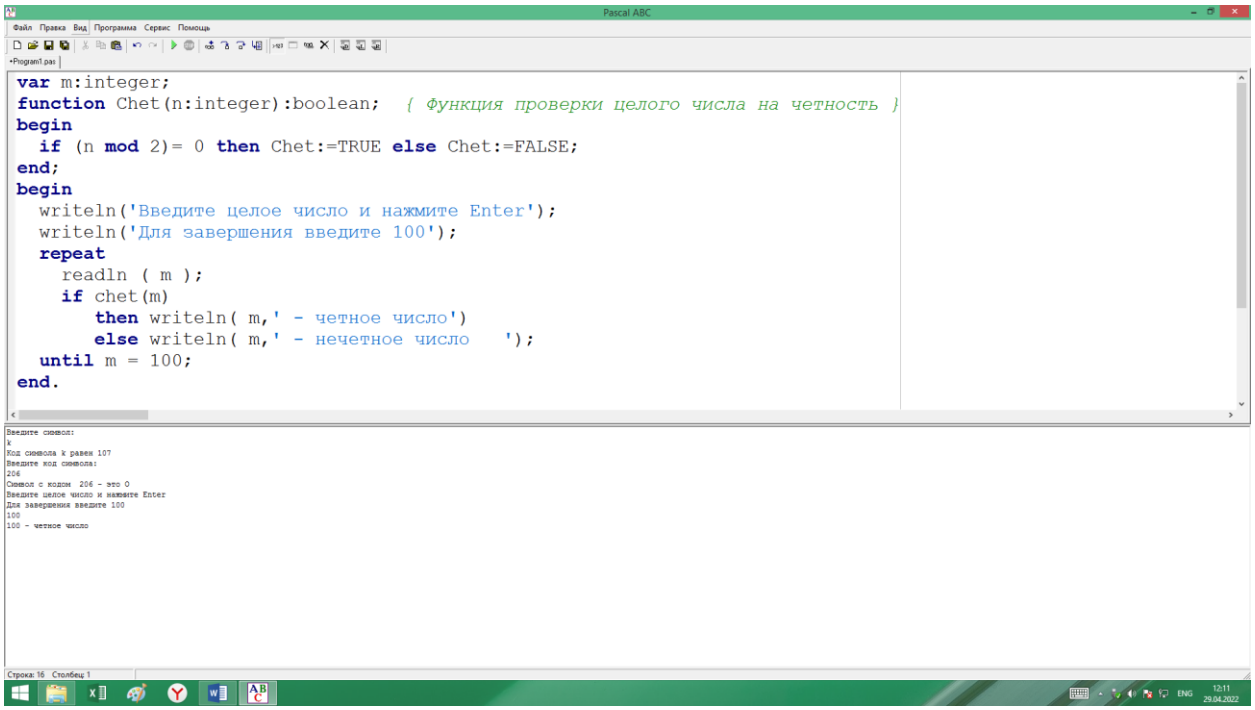
1. Программа "Неповторяющиеся случайные числа" с использованием массива и процедуры без параметров.

```
uses crt;
var a,b,c,d:integer;
    q:array[1..15] of integer;
procedure qw;
begin
  for b:=1 to 5 do
  begin
    if a=b then continue;
    if q[a]=q[b] then begin q[a]:=random(20); qw; end;
  end;
end;
begin
clrscr;
for a:=1 to 5 do
  q[a]:=random(20);
for a:=1 to 5 do
  qw;
for a:=1 to 5 do
  writeln (q[a]);
end.
```



2. Проверка целого числа на четность (использование функции).

```
var m:integer;
function Chet(n:integer):boolean;  { Функция проверки целого числа на четность }
begin
  if (n mod 2)= 0 then Chet:=TRUE else Chet:=FALSE;
end;
begin
  writeln('Введите целое число и нажмите Enter');
  writeln('Для завершения введите 100');
  repeat
    readln ( m );
    if chet(m)
       then writeln( m,' - четное число')
       else writeln( m,' - нечетное число   ');
  until m = 100;
end.
```



3. Вычисление длины и площади окружности (использование процедуры с параметрами).

```
uses crt;
var t,l,s:real;{глобальные переменные радиус, длина и площадь окружности}
procedure SqLeOkr(r:real);{процедура, r формальный параметр процедуры}
begin
  s:=pi*r*r;{r - радиус, s - площадь круга, l - длина окружности}
  l:=2*pi*r;
end;
begin {основная программа}
  writeln('Вычисление длины окружности и площади круга:');
  write('Задайте радиус и нажмите Enter  ');
  readln(t);
  l:=0;
  s:=0;
  SqLeOkr(t); {фактический параметр процедуры}
  writeln('Радиус окружности: ',t:3:1);
  writeln('Длина окружности: ',l:3:1,',площадь: ',s:3:1);
end.
```

ГЛАВА 12. РЕКУРСИЯ

Подпрограммы в Паскале могут обращаться сами к себе. Такое обращение называется ***рекурсией***.

Для того чтобы такое обращение не было бесконечным, в тексте подпрограммы должно быть условие, по достижению которого дальнейшее обращение к подпрограмме не происходит.

Рекурсивностью в Паскале могут обладать как функции, так и процедуры.

Рекурсия может быть бесконечной. Но, как и любой другой алгоритм, она обязана выдавать результат своей работы за некое определенное количество операций.

Поэтому важно: На каждом шаге выполнения рекурсивного тела процедуры или функции обязательно либо изменение параметра данной функции, либо наличие условия при котором она больше себя не будет вызывать!

Рассмотрим простой пример использования рекурсивной процедуры:

Пример: Напечатать последовательность чисел в обратном порядке, используя рекурсивный вызов процедуры. Например:

row (5) = 5 4 3 2 1

Из условия задачи ясно, что условием завершения рекурсии будет сам аргумент функции, который следует уменьшать на единицу, пока он >= 1.

```
procedure row(n: integer);
begin
  if n >= 1 then begin
    write(n, ' ');
    row(n - 1)
  end;
end;
```

```
begin
  row(10);
end.
```

Теперь рассмотрим более сложный пример использования рекурсии в Паскаль.

Пример: Написать рекурсивную процедуру, выводящую цифры, переданного ей в качестве фактического параметра числа, в обратном порядке.

Например: при переданном функции числе 3078, должно в итоге вернуть 8703
Использовать операции div и mod

```
procedure reverse(n: integer);
begin
  write(n mod 10);
  if (n div 10) <> 0 then
    reverse(n div 10)
end;

begin
  writeln;
  reverse(3078);
end.
```

А теперь посмотрим, как используется рекурсия при вычислении факториала в Паскаль.

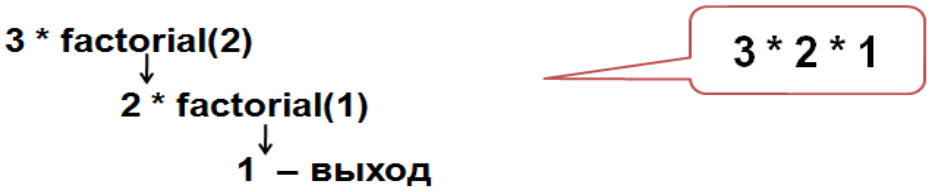
Пример: Создать рекурсивную функцию для вычисления факториала числа

Подсказка:

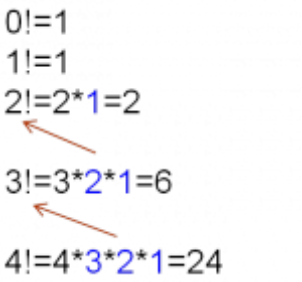
2!=2*1=2

3!=3*2*1=6

Выводим формулу a!=a*((a-1)!)




```
var
  x: integer;

function fact(a: integer): integer;
begin
  if (a <= 1) then
    a := 1
  else
    a := a * (fact(a - 1));
  fact := a;
end;

begin
  writeln('Число?');
  readln(x);
  writeln(fact(x));
end.
```



Пример.

Рассмотрим математическую головоломку из книги Ж. Арсака «Программирование игр и головоломок».

Построим последовательность чисел следующим образом: возьмем целое число i>1. Следующий член последовательности равен i/2, если i четное, и 3i+1, если i нечетное. Если i=1, то последовательность останавливается.

Математически конечность последовательности независимо от начального i не доказана, но на практике последовательность останавливается всегда.

Применение рекурсии позволило решить задачу без использования циклов, как в основной программе, так и в процедуре.

Пример программы с использованием рекурсии

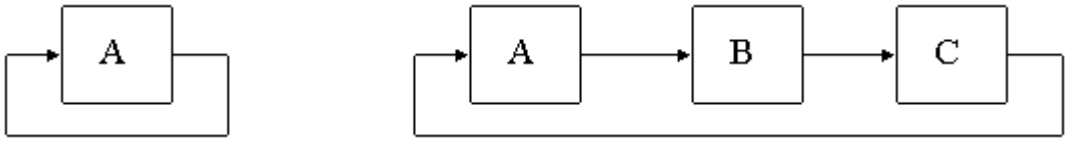
```
Program Arsac;
Var first: word;
Procedure posledov (i: word);
Begin
  Writeln (i);
  If i=1 then exit;
  If odd(i) then posledov(3*i+1) else posledov(i div 2);
End;
Begin
  Write (' введите первое значение '); readln (first);
  posledov (first);
  Readln ;
End.
```

Программист разрабатывает программу, сводя исходную задачу к более простым. Среди этих задач может оказаться и первоначальная, но в упрощенной форме. Например, для вычисления F(N) может понадобиться вычислить F(N-1). Иными словами, частью алгоритма вычисления функции будет вычисление этой же функции.

Алгоритм, который является своей собственной частью, называется ***рекурсивным.*** Часто в основе такого алгоритма лежит рекурсивное определение.

Любое рекурсивное определение состоит из двух частей. Одна часть определяет понятие через него же, другая часть – через иные понятия.

Процедура является **рекурсивной**, если она обращается сама к себе прямо или косвенно (через другие процедуры).



Заметим, что при косвенном обращении все процедуры в цепочке – рекурсивные.

Все сказанное о процедурах целиком относится и к функциям.

Рекурсия изнутри

Это может показаться удивительным, но самовызов процедуры ничем не отличается от вызова другой процедуры. Что происходит, если одна процедура вызывает другую? В общих чертах следующее:

- в памяти размещаются параметры, передаваемые процедуре (но не параметры-переменные!);
- в другом месте памяти сохраняются значения внутренних переменных вызывающей процедуры;
- запоминается адрес возврата в вызывающую процедуру;
- управление передается вызванной процедуре.

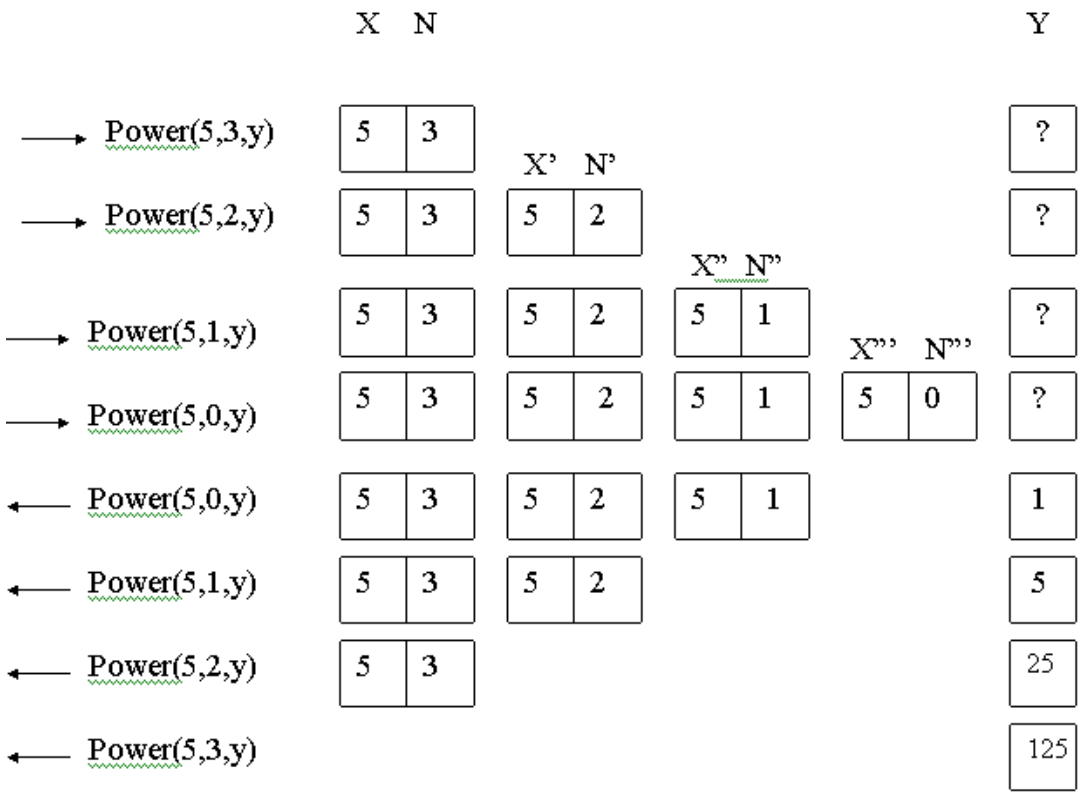
Если процедуру вызвать повторно из другой процедуры или из нее самой, будет выполняться тот же код, но работать он будет с другими значениями параметров и внутренних переменных. Это и дает возможность рекурсии.

Пусть рекурсивная процедура Power(X, N, Y) возводит число X в степень N и возвращает результат Y .

Пример рекурсивной процедуры, возводящей число в степень

```
procedure Power(X: real; N: integer; var Y: real);
begin
  if N = 0 then
    Y := 1
  Else begin
    Power(X, N - 1, Y);
    Y := Y * X;
  end;
end;
```

Проследим за состоянием памяти в процессе выполнения вызова данной процедуры Power(5,3, Y). Стрелка «->» означает вход в процедуру, стрелка «<-» означает выход из нее.



Число копий переменных, одновременно находящихся в памяти, называется ***глубиной рекурсии***. Как видно из примера, сначала она растет, а затем сокращается.

Подведем итог.

- рекурсивной называется такая процедура или функция, которая вызывает сама себя;
- для завершения процесса рекурсии в алгоритме рекурсивной функции (процедуры) обязательно должно быть условие, обеспечивающее непосредственное завершение функции (процедуры).

ПРИМЕР. Напишем программу вычисления функции, заданной следующим образом:

```
F(1)=1; F(2n)=F(n); F(2n+1)=F(n)+F(n+1)
```

Решение: из определения видно, что вычисление функции от аргумента, сводится к вычислению этой же функции от меньшего аргумента, и процесс уменьшения аргумента продолжается до тех пор, пока в качестве аргумента не получится единица. Значение функции от единицы определено.

Таким образом, работа алгоритма будет состоять из некоторого количества шагов, на каждом из которых будут выполняться два действия:

- Определение четности или нечетности аргумента. От этого зависит выбор формулы вычислений;
- Выполнение вычислений. Фактически это будет сводиться к определению нового аргумента функции.

Пример рекурсивной программы вычисления функции

```
program primer;

Uses crt;

var
  N, a: integer;

function f(n: integer): integer;
begin
  if n = 1 then
    f := 1 {условие завершения рекурсии}
   Else
  begin
    if odd(n){проверка на нечетность числа}
    then begin
      n := n div 2;
      f := f(n) + f(n + 1)
    end
    else begin
```

```
        n := n div 2;
        f := f(n)
      end;
    end;
end;

begin{начало основной программы}
  clrscr;
  write(' Введите число – ');
  readln(n);
  a := f(n);
  write(' результат – ', a);
end.
```

Задачи для решения

1. Написать рекурсивную функцию sumTo(n), которая для данного n вычисляет сумму чисел от 1 до n, например:

```
sumTo(1) = 1
sumTo(2) = 2 + 1 = 3
sumTo(3) = 3 + 2 + 1 = 6
...
```

2. Найти НОД методом Эвклида. Использовать рекурсивную процедуру.

Пример для чисел 3430 и 1365:

остаток от деления 3430 на 1365	3430 mod 1365 = 700
остаток не равен нулю, повторим то же действие, подставив вместо первого числа второе, а вместо второго – остаток	1365 mod 700 = 665
остаток также не нуль, поэтому еще одно деление	700 mod 665 = 35
остаток также не нуль, поэтому еще одно деление	665 mod 35 = 0
остаток нуль	НОД равен 35

ГЛАВА 13. СИМВОЛЬНЫЙ ТИП ДАННЫХ. ФУНКЦИИ ДЛЯ РАБОТЫ С СИМВОЛЬНЫМ ТИПОМ

Символьный тип данных Char — тип данных, значениями которого являются одиночные символы. Данный тип может содержать всего один любой символ (Например: «*», «/», «.», «!» и другие). Каждый такой символ занимает 8 бит памяти, всего существует 256 восьмибитовых символов. Все символы, используемые символьным типом Char записаны в таблице символов ASCII (American Standart Code for Information Interchange) или Американский стандарт кода для обмена информацией.

Символьные константы заключаются в апострофы, например '.', '*', '7', 's'. Также символьную константу можно записать с помощью символа — «решетки», например #185 — выведет символ под номером 185 из таблицы ASCII (это символ '№').

К типу Char применимы следующие встроенные функции:

1) Функция Chr (x: byte): char. Возвращает символ, соответствующий ASCII - коду числа x;

2) Функция Ord (x: char):byte. Возвращает число (код), соответствующее символу x в ASCII - таблицы;

3) Функция UpCase (x: char): char. Преобразует символы из строчных букв в прописные, но распространяется только на литеры латинского алфавита, русские просто игнорируются;

4) Функция Pred (x: char): char. Возвращает символ, который предшествует в ASCII таблице символу x,

5) Функция Succ (x; char): char. Возвращает символ, который следует в ASCII - таблице за символом x.

Например:

- ord ('A') = 65
- chr (128) = 'Б'
- pred ('Б') = 'А'

- succ ('Г') = 'Д'
- upcase ('n') = 'N'

Пример программы на Паскаль с использованием функции Ord:

```
var
  x: char; // Описание переменных (x - символьный тип)

begin //Начало программы
  readln(x); //Считывание переменной
  writeln(ord(x)); //Вывод номера в таблице ASCII
end. //Конец программы
```

Пример программы на Паскаль с использованием функции Chr:

```
var
  x: integer; // Описание переменных (x - целочисленный тип)

begin //Начало программы
  readln(x); //Считывание переменной
  writeln(chr(x)); //Вывод символа по номеру в таблице ASCII
end. //Конец программы
```

Пример программы на Паскаль с использованием функций Pred и Succ:

```
var
  x: char; // Описание переменных (x - символьный тип)

begin //Начало программы
  readln(x); //Считывание переменной
  writeln(pred(x)); //Вывод предыдущего символа в таблице ASCII
```

```
  writeln(succ(x)); //Вывод следующего символа в таблице ASCII
end. //Конец программы
```

Пример программы на Паскаль с использованием функции UpCase:

```
var
  x: char; // Описание переменных (x - символьный тип)

begin //Начало программы
  readln(x); //Считывание переменной
  writeln(upcase(x)); //Вывод английской буквы верхнего регистра
end. //Конец программы
```

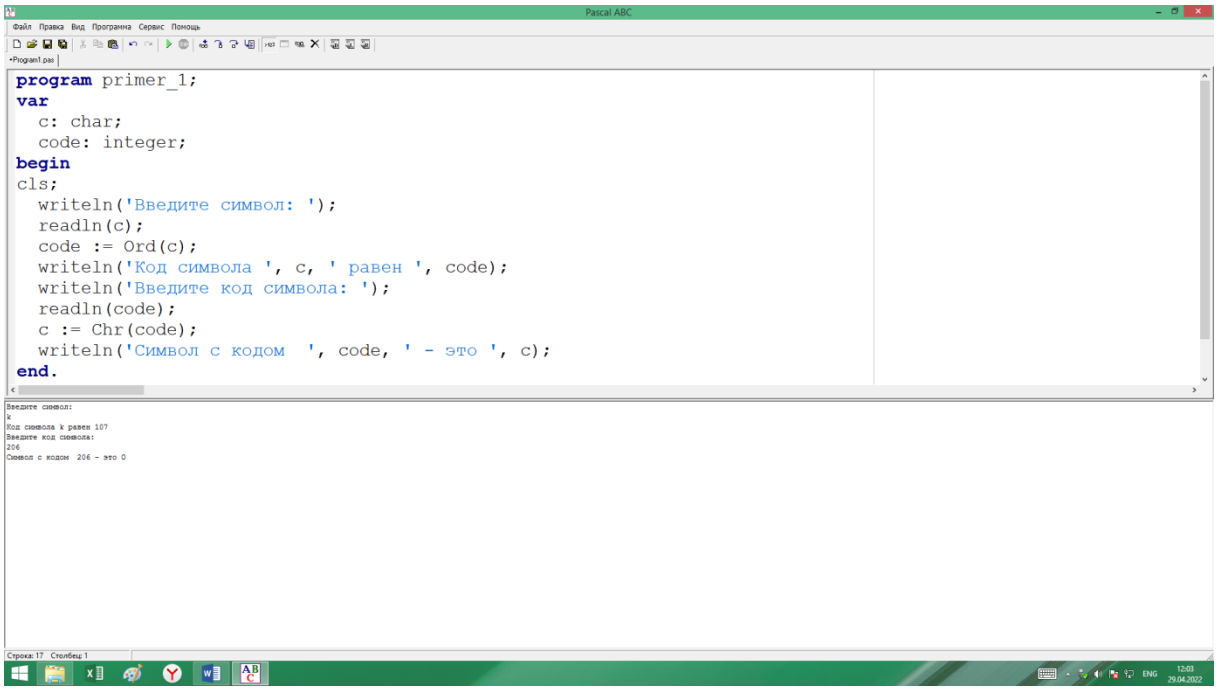
Пример описания величины символьного типа:

```
 Var
   letter, symbol, ch: char;
```

Пример 1. Определение кода по символу и символа по коду.

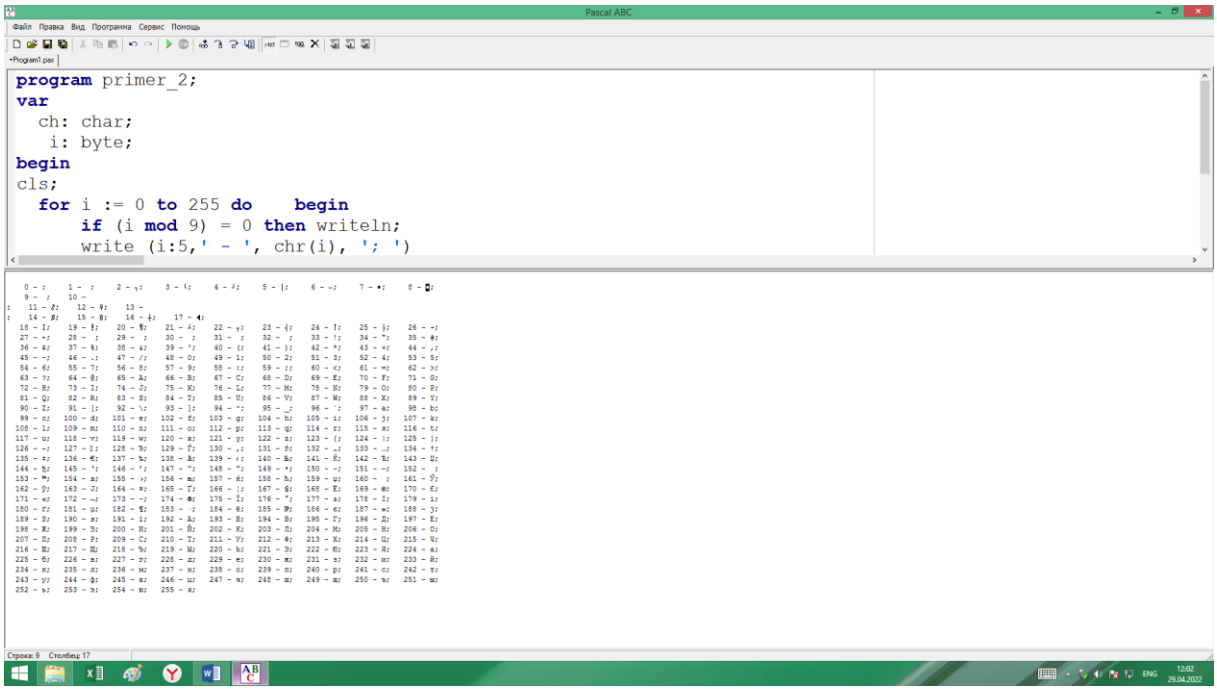
```
program primer_1;
var
  c: char;
  code: integer;
begin
cls;
  writeln('Введите символ: ');
  readln(c);
  code := Ord(c);
  writeln('Код символа ', c, ' равен ', code);
  writeln('Введите код символа: ');
```

```
  readln(code);
  c := Chr(code);
  writeln('Символ с кодом  ', code, ' - это ', c);
end.
```



Пример 2. Напишите программу для вывода на экран всех символов таблица ASCII.

```
 program primer_2;
var
  ch: char;
   i: byte;
begin
  for i := 0 to 255 do    begin
      if (i mod 5) = 0 then writeln;
      write (i:3,' - ', chr(i), '; ')
  end;
end.
```



Пример 3. Вывести в одну строку: ABBCCC…ZZ…Z

Переменные:

i – переменная цикла; определяет, какая буква будет выводиться;

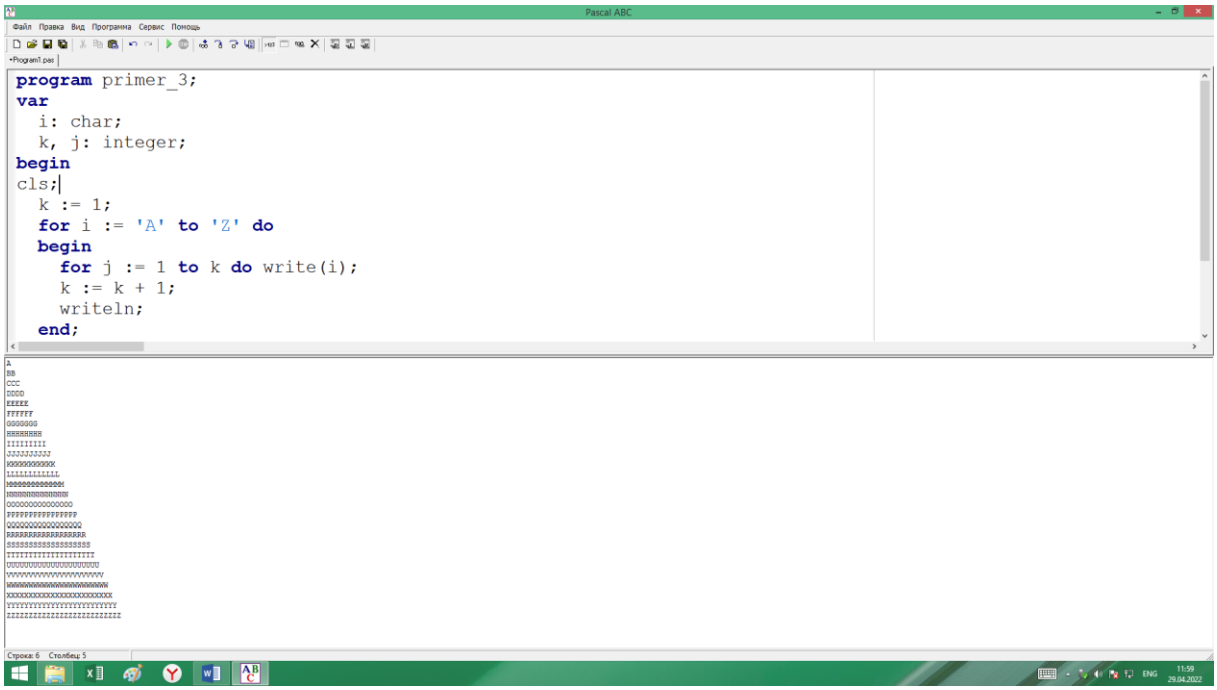
k – количество повторений буквы;

j – переменная цикла, счетчик количества уже выведенных букв.

```
program primer_3;


var
  i: char;
  k, j: integer;
begin
cls;
  k := 1;
  for i := 'A' to 'Z' do
  begin
    for j := 1 to k do write(i);
```

```
    k := k + 1;
   writeln;
  end;
end.
```

Внешний цикл определяет, какую букву выводим на печать, внутренний – сколько раз будет печататься буква.



ГЛАВА 14. СТРОКОВЫЙ ТИП ДАННЫХ. ФУНКЦИИ ДЛЯ РАБОТЫ СО СТРОКОВЫМИ ВЕЛИЧИНАМИ

Строка — это последовательность символов. Максимальное количество символов в строке *(длина строки)* может изменяться от 1 до 255. Переменную строкового типа можно определить через описание типа в разделе определения типов или непосредственно в разделе объявления переменных.

Переменные: **string** [максимальная длина строки];

Если максимальная длина строки не указывается, то она равна 255 байт.

При вводе строки количество символов в ней определяется автоматически, при этом автоматически заполняется нулевой байт. Для получения длины строки имеется функция length, которая возвращает значение нулевого байта строки. К любому символу в строке можно обратиться, как к элементу массива: S[i], S – строка.

Разумеется, никто не запрещает вводить и выводить строки по отдельным символам, используя любой из операторов цикла. Однако такая необходимость возникает редко. Посимвольный ввод строки стоит реализовать только в том случае, если необходимо совместить ввод и какую-то обработку символов. Посимвольный вывод также удобно использовать для какой-либо нестандартной формы вывода.

Операции над строками:

1) Операция сцепления (+)

Применяется для соединения нескольких строк.

Например: Выражение: 'Turbo' + ' Pascal'+'7.0'

Результат: 'Turbo Pascal 7.0'

2) Операции отношения

Операции =, < >, >, <, >=, <= выполняют сравнение двух строковых операндов и используются в основном при проверке условий. Сравнение строк производится слева направо до первого несовпадающего символа, и та строка считается большей, в которой код перво-

го несовпадающего символа больше. Такой способ сравнения строк называют лексикографическим. Строки считаются равными, если они совпадают по длине и содержат одни и те же символы.

Например:

Выражение	Результат
'Иванов' = 'Иванов И.И.'	false
'Иванов' < 'Иванов И.И.'	true
'program'='PROGRAM'	true

3) Процедура delete(st, poz, n) — удаление n символов строки st, начиная с позиции poz. Если значение poz больше, чем размер строки, ничего не удаляется.

При удалении символов оставшаяся часть строки подтягивается к началу, чтобы занять образовавшуюся после удаления "дырку".

Например:

```
st: = 'река Волга'; delete(st, 1, 5); { в результате st='Волга' }
```

Следующий фрагмент программы удаляет все пробелы из начала строки st (пробелы в начале строки называются ведущими пробелами):

```
while st[l]=' ' do delete(st,1,1) ;
```

4) Процедура insert(str1, str2, poz) — вставка строки str1 в строку str2, начиная с позиции poz. Не перепутайте: первый параметр - что вставляем, второй - куда.

Например:

```
s1:= 'Я разрабатываю программы'; s2 := 'хорошие ';
insert (s2,s1, 16) ;
{ в результате s1='Я разрабатываю хорошие программы' }
```

При вставке строка раздвигается, чтобы вместить вставляемые символы.

При вставке в начало и конец строки действие процедуры insert аналогично выполнению операции конкатенации. Так, например, для вставки звездочек в конец строки можно использовать

```
s:=s+'*' на insert ('*', s, length (s)+1);
```

Обратите внимание — при вставке символов обязательно нужно контролировать длину полученной строки, чтобы не потерять последние символы. При этом никакого сообщения об ошибке не выдается даже при включенной директиве {$R+}.

5) Функция length(st) — вычисляет текущую длину в символах строки st. Результат имеет целочисленный тип.

```
Например, n:=length('123456789'); { n=9 }
```

6) Функция copy(st, poz, n) — выделяет из строки st подстроку длиной n символов, начиная с позиции poz. Если poz больше длины строки, то результатом будет пустая строка.

Например:

```
s1: = 'Turbo Pascal';
s2:=copy (s1, 1,5) ;
s3:=copy(sl,7,3) ;
{ в результате s2='Turbo'; s3='Pas' }
```

7) Функция concat(str1, str2,..., strn) — выполняет сцепление строк str1, str2, ... strn в том порядке, в каком они указаны в списке параметров.

Например:

```
s:=concat('AA', 'XX', 'Y'); ( в результате s='AAXXY'; }
```

Функция concat выполняет те же действия, что и операция конкатенации. Например, для приведенного случая то же самое можно было записать так: s: = 'AA' + 'XX' + 'Y';.

8) Функция pos(strl, str2) — обнаруживает первое появление в строке str2 подстроки str1. Результат имеет целочисленный тип и равен номеру той позиции, где находится первый символ подстроки str1. Если в str2 подстроки str1 не найдено, результат равен 0.

Например:

```
s1:= 'Turbo Pascal';
n1:=pos{'Pascal',s1); n2:=pos('pascal', s1);
{В результате n1=7; n2=0 (pascal и Pascal - это разные строки)}
```

9) Процедура str(number, st) — преобразование числового значения величины number в строку st. После number может записываться формат, аналогичный формату вывода. Если в формате указано недостаточное для вывода количество разрядов, поле вывода расширяется автоматически до нужной длины. Удобно использовать процедуру str для вставки числовых данных в какой-либо текст, т. к. операция конкатенации и процедура insert могут работать только со строковыми данными.

Например:

```
var
  Sl, S2, s3, s4: string;
  num1: integer;
  num2: real;
...
num1: =5; num2:=5.78;
str (num1, S1) ;str (num1:3,s2) ; str (num2, s3) ; str (num2: 3:1, s4);
{ в результате s1='5'; S2=' 5'; s3='5.780000000000E+00'; s4='5.8'; }
```

10) Процедура val(st, number, code) — преобразует значение st в величину целочисленного или вещественного типа и помещает результат в numder. Code - целочисленная переменная. Если во время операции преобразования ошибки не обнаружено, значение code равно нулю, если ошибка обнаружена (строковое значение не переводится в цифровое), code будет содержать номер позиции первого ошибочного символа, а значение number не определено.

Например:

```
s1:= '5.78'; s2:= '5,78';
val(s1,num1, cod1);
val(s2,num2,cod2);
{ в результате cod1=0, cod2=2 - второй символ ошибочный }
```

Обратите внимание — использование функции val позволяет избежать неприятной ошибки выполнения Invalid numeric format, возникающей при неправильном вводе числовых данных. Приходится прибегать к такой хитрости — вводить вместо числовой переменной строковую (в ней разрешены любые символы). После этого введенная строка преобразуется в число и, если преобразование невозможно, об этом выводится сообщение на русском языке. Чаще всего в таких случаях не прерывают программу, а просят пользователя повторить ввод. Здесь очень удобно использовать цикл repeat, т. к. гарантируется выполнение тела цикла хотя бы один раз.

Например:

```
var
  s: string;
  n, error: integer;

begin
  repeat
    write('Введите число '); { просим ввести чиспо, }
    readln(s); { а вводим строку }
    val(s, n, error); { преобразуем строку в число }
    if error > 0 then writeln('Неверный символ № ', error);
  until error = 0;
... { продолжение программы }
end.
```

Задачи

1. Дано натуральное число N. Используя функции и процедуры:

а) выяснить, входит ли цифра 3 в запись числа N.

б) переставить первую и последнюю цифры числа N.

в) определить сумму его цифр.

г) определить, сколько цифр в числе.

д) приписать по "1" в начало и в конец числа.

е) поменять порядок числа на обратный.

2. Написать программу перевода чисел из десятичной системы счисления в римскую.

3. Задача "Меню". Дана непустая последовательность слов из строчных русских букв, между словами запятая, в конце точка.

1) определить максимальную длину слова в последовательности.

2) напечатать в алфавитном порядке все гласные буквы, которые содержатся в каждом слове.

3) определить количество слов в предложении.

4) каждую глухую согласную заменить на соответствующую звонкую и наоборот.

5) напечатать заданную фразу в обратном порядке.

Например для фразы "наступило утро":

1) 9

2) о у

3) 2

4) наздубило удро

5) орту олипутсан

Задачи для самостоятельного решения

1. Слова и фразы, которые можно читать справа налево и слева направо. Такие слова или фразы называются перевертыши или палиндромы. Например, ШАЛАШ, РОТОР.

Данная программа проверяет, является ли слово палиндромом. При вводе выражения пробелы не вводить.

```
program palindrom;
uses crt;
var     q:integer;
   a,b,c:string;
begin
 clrscr;
 writeln('Введите слово - палиндром ');
 readln(a);
 for q:=1 to length(a) div 2 do      {проверка до середины слова}
 begin
  b:=copy(a,q,1);                  {очередная буква слева}
  c:=copy(a,length(a)+1-q,1);         {соответствующая буква справа}
  if b<>c then
  begin
   write('Это не палиндром !!!');
   exit;
  end;
 end;
 write('Правильно.');
end.
```

Переделайте программу так, чтобы в ней не использовался exit.

2. Если выражение состоит из нескольких слов, то нужно, чтобы компьютер анализировал наличие пробелов. Это реализовано в следующей программе:

```
program palindrom2;
uses crt;
var q:integer;
   a:string;
begin
 clrscr;
 write('Введите фразу ');
 readln(a);
 {==========Алгоритм удаления пробелов==========}
 for q:=1 to length(a) do
 if copy(a,q,1)=' ' then delete(a,q,1);
 {=============================================}
 for q:=1 to length(a) div 2 do
  begin
   if copy(a,q,1)<>copy(a,length(a)+1-q,1) then
   begin
    write('Это не палиндром, вот пример палиндрома: "аргентина манит негра"');
    exit;
   end;
  end;
 write('Правильно.');
end.
```

Переделайте программу так, чтобы в ней не использовался exit.

3. Из числовых переменных, равных 4, 7, 9 с помощью строковых функций получить число 794 и вывести его на экран.

4. Запросить с клавиатуры слово и число до 10000, определить количество в них знаков, напечатать результат на экране.

5. Запросить с клавиатуры три числа, сложить их, затем определить количество знаков в сумме, напечатать результат на экране.

6. Составить программу, чтобы компьютер сгенерировал 10 случайных целых трехзначных чисел. Длину определять с помощью строковых функций.

7. Из слова "ВЕЛИКОЛЕПНЫЙ" с помощью строковых функций получить слова: ВЕЛИК, КОЛ, ЛИК, ЛЕВ, ВЕК и вывести их на экран.

8. Из своих фамилии и имени с помощью строковых функций получить 10 слов и вывести их на экран.

9. Составить программу, чтобы компьютер сгенерировал случайное целое трехзначное число, с помощью строковых функций определил, сколько в нем единиц, десятков и сотен, вывел эту информацию на экран.

10. Составить программу, чтобы компьютер сгенерировал три случайных целых однозначных чисел и из них сформировал трехзначное число, все числа вывел на экран.

ГЛАВА 15. ИСПОЛЬЗОВАНИЕ ЭВМ ДЛЯ ШИФРОВКИ И ДЕШИФРОВКИ СООБЩЕНИЙ

Сообщение – это упорядоченная совокупность символов некоторого алфавита.

Алфавит – любое множество символов.

Ключ – алгоритм, позволяющий расшифровать сообщение.

Криптостойкость шифра – способность быть нерасшифрованным без знания ключа.

Методы шифрования

1. *Шифрование с помощью ключевой фразы.*

Ключевая фраза – некоторое сообщение, в котором должны содержаться все символы алфавита и знаки препинания. Например, можно использовать такую фразу: «умею ли я находить с помощью электронно-вычислительной машины значения функций, а также объемы многогранников?!.»

В этой фразе содержатся все буквы русского алфавита и знаки препинания, кроме кавычек.

Алгоритм шифрования

1) Ввести с клавиатуры секретное сообщение.
2) Каждый символ секретного текста заменить на его номер в ключевой фразе.
3) Вывести на экран зашифрованное сообщение.

Для дешифровки осуществляется обратный процесс, каждый номер заменяется на соответствующий символ из ключевой фразы.

Пример. Слово «информатика» шифруется так: 7 11 72…

Номер шифрограммы	Шифрограмма
1.	1 5 6 1 33 14 2 14 35 18 9 5 15 1 90 5 63 3 6 3 11 42 54
2.	2 14 54 5 15 9 15 9 79 5 20 12 2 42 13 5 43 3 20 17 11 42 13 5 22 35 12 41 7 6

3.	9 5 22 14 2 11 4 5 43 1 15 11 14 3 5 2 99 11 14 41 3 11 18 3
4.	6 4 90 6 4 5 99 35 14 63 1 5 41 5 11 12 43 12 6 3 5 2 12 9
5.	41 5 6 3 20 1 5 35 14 15 7 6 12 20 18 5 3 6 14 43 33 12
6.	15 3 35 3 41 11 9 79 5 99 15 3 5 20 33 1 43 12 6 5 3 41 99 3 11 7 54 79 5 90 42 6 12 5 22 35 3 6 3 20 17 11 42 54 5 1 99 14 6 14 33
7.	2 42 5 41 20 3 5 1 43 7 6 7 20 18 5 22 14 11 3 2 11 14 99 1
8.	41 5 17 14 17 5 99 14 15 5 14 20 3 11 11 9 9 5 22 14 99 14 15 12 5 20 17 14 9 6 12 5 15 14 6 99 14 5 11 12 5 15 41 14 35 3
9.	41 20 3 5 22 35 14 54 15 3 17 79 5 33 12 33 5 20 5 90 3 6 42 13 5 9 90 6 14 11 18 5 15 42 2
10.	20 33 12 63 33 12 5 6 14 86 18 79 5 15 12 5 41 5 11 3 54 5 11 12 2 0 33 111 5 15 14 90 35 42 2 5 2 14 6 14 15 76 12 2 5 1 35 14 33 111
11.	12 5 41 42 5 11 14 33 17 4 35 11 5 20 42 99 35 12 17 18 5 20 2 14 99 6 7 5 90 42 5 11 12 5 72 6 3 54 17 3 5 41 14 15 14 20 17 14 43 11 42 13 5 17 35 1 90 110
12.	9 5 33 5 41 12 2 5 22 7 58 1 5 40 5 43 3 99 14 5 86 3 5 90 14 6 3 110
13.	63 7 2 12 111 112 112 5 33 35 3 20 17 18 9 11 7 11 79 5 17 14 35 86 3 20 17 41 1 9
14.	3 20 6 7 5 63 41 3 63 15 42 5 63 12 86 7 99 12 4 17 20 9 79 5 63 11 12 43 7 17 79 5 30 17 14 5 33 14 2 1 40 11 7 90 1 15 18 5 11 1 86 11 14
15.	1 86 5 22 14 6 11 14 43 18 5 90 6 7 63 7 17 20 9 79 5 12 5 99 3 35 2 12 11 12 5 41 20 3 5 11 3 17
16.	20 2 3 58 12 6 7 20 18 5 41 5 33 1 43 1 5 33 14 11 7 79 5 6 4 15 7 112 112 112
17.	9 5 63 11 12 4 79 5 99 14 35 14 15 5 90 1 15 3 17 111 5 9 5 63 11 12 4 79 5 20 12 15 1 5 76 41 3 20 17 18 111
18.	9 5 11 12 5 22 35 12 41 1 4 5 35 1 33 1 5 11 12 15 3 6 12 5 22 3 35 43 12 17 33 1 5 20 5 6 3 41 14 54 5 35 1 33 7
19.	9 5 22 12 2 9 17 11 7 33 5 20 3 90 3 5 41 14 63 15 41 7 99 5 11 3 35 1 33 14 17 41 14 35 11 42 54
20.	11 14 43 18 79 5 1 6 7 76 12 79 5 72 14 11 12 35 18 79 5 12 22 17 3 33 12 112 112 112

21.	2 14 35 14 63 5 7 5 20 14 6 11 76 3 79 5 15 3 11 18 5 43 1 15 3 20 11 42 54 111
22.	2 14 20 33 41 12 111 5 33 12 33 5 2 11 14 99 14 5 41 5 30 17 14 2 5 63 41 1 33 3

2. Вставка.

Данный алгоритм шифрования заключается в следующем. После каждого символа (буквы) вставляется некоторая буква, полученная случайным образом.

Алгоритм шифрования

1) Ввести с клавиатуры секретное сообщение.
2) Формирование зашифрованного текста с использованием генератора случайных чисел RND: чтобы вставить случайную букву, генерируем число из интервалов кодов таблицы ASCII, соответствующих русским буквам.
3) Вывод полученного текста на экран.

Для дешифровки необходимо удалить вставленные символы.

Пример. Фраза «миру мир» шифруется так: «**м**р**и**л**р**е**у**ы **м**й**и**я**р**э».

3. Наоборот.

Алгоритм шифрования

Переворачивается каждое слово сообщения.

Пример. Фраза «учись студент» шифруется так: «ьсичу тнедутс».

Дешифровка – аналогично.

4. Тарабарский язык.

Алгоритм шифрования

Глухие буквы секретного текста заменяются на звонкие и наоборот.

А=”бвгдзж”

B=”пфктсш”

Пример. Фраза «сегодня студент, а завтра доцент» шифруется так: «зекотня здутенд, а сафдра тоценд».

Алгоритмы шифровки и дещифровки совпадают.

5. *Шифр Цезаря.*

Пусть $A=\{a_1,a_2,\ldots a_n\}$ – алфавит, $S=\{s_1,s_2,\ldots s_k\}$ – сообщение, s_i принадлежит А.

Будем называть **кодом символа** число, равное порядковому номеру этого символа в алфавите. **Ключом** для шифра Цезаря является некоторое число n, которое назовем **сдвигом.**

Алгоритм шифрования

1) Ввести с клавиатуры секретное сообщение.
2) Вырезать символ.
3) Определить его код, т.е. порядковый номер в алфавите.
4) К коду прибавить сдвиг и по полученному числу восстановить букву.
5) Вывод полученного текста на экран.

Замечание 1. Если полученное после прибавления сдвига число выходит за пределы алфавита (т.е. оно больше 32), заменяем его остатком от деления на длину алфавита 32 (буква ё не учитывается).

Пример. Шифруем слово «информатика», n=2.

Код(и)=9, 9+2=11, на 11-м месте в алфавите стоит буква «л». И т.д.

Код(я)=32, 32+2=34, 34 mod 32=2, на 2-м месте в алфавите стоит буква «б».

Замечание 2. Для удобства кириллицу можно задать какой-то переменной: А=”абвгдежзийклмнопрстуфцчшщъыьэюя”.

Алгоритм дешифровки отличается тем, что к коду надо прибавлять значение, противоположное сдвигу (вычитать сдвиг).

Замечание 3. Если полученное после вычитания сдвига число выходит за приделы алфавита слева (т.е. оно меньше или равно 0), заменяем его по формуле k=k mod 32+32.

Пример. Алфавит располагается по кругу «…эюяабвгд…». Если при дешифровке получилось, что код=0, ему соответствует буква «я». Действительно, k=0 mod 32+32, k=32, на 32-м месте в алфавите стоит буква «я».

Если код=-33, то по формуле k=-33 mod 32+32, k=-1+32=31, на 31-м месте в алфавите стоит буква «ю».

Номер шифрограммы	Зашифрованный текст	Шаг сдвига
1.	Гыюяхънм Аъъи Пнпшыпъи ъндншн ьыътщъыра ъньышъмяйюм. Пэхтвншн пиюенм фъняй Птятэоаэрн, шлсх юнщит энфъыэысъит ьы пыфэнюянщ х внэнчятэнщ, ъы ысхънчыпит ьы ыожтюяпа, п чнчыщ пют ухшх; ьэхтвншн сыдй чъмфм Внюхшхм, чэнюнпхгн Эштъ.	45
2.	Пдьщйфя юбуыр Иггвяьж, епб юбуыу Вфеьяьу, е Мвджщафдва, ювжвдвчв вб гдщшежфцья; гдьщйфя ь фххфж Мвдьв ь абвчьщ шдзчьщ.	148
3.	Вгч хагдъ гафчвктэъ аувсц бвъфчдгдфафтяъс яъьаюе ячъщфчгдяаы, яъьаюе ячъядчвчгяаы ъ ячяешяаы дчдекьъ. Аяят Птфэафят г хвегдяню, давшчгдфчяяню ейтгдъчю гэчцъэт щт ъз бвъфчдгдфъсюъ, юаэйтэъфа ацаувсс ъз.	18
4.	Слщйбгще б кгмрщчтбе, ейщржфе ездзэфе дчэше, кезлйюысбе жщ жюю, гщащдзкх, рлз зжб кщеб эюдщчлкш изозяб жщ жюю, изъфы б изьзызйбы жюкгздхгз ыйюеюжб к жюв.	25

5.	Виь жеымеыящпяь, яю жзявяояц дь щтбчютщчц жеижьпдеийя, и окщийщег ешвьъоьдяц яижевдьддеа йцэьвеа ешцючддеийя еймеыявя ей ийчзкпбя, ойешт кэ щьиу щьоьз дя зчюк дь жеые-айя б дьа.	87
6.	Кхс жсесуло ф рим л елзио тул нгйзсп фосеи ии феихоцб цоюдсънц л доифхвьли диоюи кцдю, нсхсуюи елзриолфя дифтуифхгррс, хсх зцпго, ъхс ср сфсдиррс рюръи обдикир. И ахс зцпго нгйзюм.	99
7.	Пдьщйфя юбуыр Иггвяьж, епб юбуыу Вфеьяьу, е Мвджщафдва, ювжвдвчв вб гдщшежфцья; гдьщйфя ь фххфж Мвдьв ь абвчьщ шдзчьщ.	116
8.	Врйнпд онркд лякдмыйни ймювзмз бнчдк ляррзбмъи, снкрсъи лнкнгни цдкнбдй р рспзедмнэ внкнбни, б нцйяф, рбдскъф оямсякнмяф он снвгячмди лнгд, р бърнйзл еяан з б йнпзцмдбнл упяйд.	31
9.	Нй, иамзйнлъ иы шнй игвуаа кй мэйазо мйлно клгэанмнэга, клг эгяа эйуаяуаюй Пчалы э жгса Аииц Пыэжйэиц гвйьлывгжймч ьамкйейдмнэй г мнлыр, кйяйьицд нйзо, ейнйлцд эцлыбыанмъ клг эгяа таюй-игьояч мжгуейз йюлйзийюй г иам-эйдмнэаиийюй замно.	91
10.	Хювп, фхщбвтшвхымэю, Пмха слы эхбъюымъю сюымих фагуше ьгцзшэ т ъюьэрвх, эю нвюв бваре ьюу ювэюбшвмбп вюымъю ъ вюьг гьэюьг ш тьхбвх аюсъюьг, эрсыофрвхымэюьг ш хбвхбвтхээюьг тчуыпфг, ювышзртихьг хую юв тбхе т нвющ уюбвшэющ.	16
11.	Пъпроп шч, чп ошыхэвкм ыхшм ышлпыпочтае, эвпх; ьпщпъж шч шыькчшмтх ымштц ъксншмшъшц ышлпыпочтаэ, фшьшъшу чэрчш лехш шь чпнш эуьт. Оч, чкнчэм ншхшмэ т ъкыыькмтм лшхжвтп чшнт, ыькх ошфксемкьж Аччп Пкмхшмчп, щшбпцэ шч щшхкнкх, бьш щхкч кллкьк лех ятцпък.	74
12.	Гыюяхънм Аъъи Пнпшыпъи ъндншн ьыътщъыра ъньышъмяйюм. Пэхтвншн пиюенм фъняй Птятэоаэрн, шлсх юнщит энфъыэысъит ьы пыфэнюянщ х внэнчятэнщ, ъы ысхънчыпит ьы ыожтюяпа, п чнчыщ пют ухшх; ьэхтвншн сыдй чъмфм Внюхшхм, чэнюнпхгн Эштъ.	77

13.	Виь жеымеыящпяь, яю жзявяояц дь щтбчютщчц жеижьпдеийя, и окщийщег ешвьъоьдяц яижевдьддеа йцэьвеа ешцючддеийя еймеыявя ей ийчзкпбя, ойешт кэ щьиу щьоьз дя зчюк дь жеые-айя б дьа.	55
14.	Птыж ездчйжбюб нид-ид гыедгхигды ю ездъдбьцб дисзаюшцит нид-ид щбцэцвю. Ог жцъдзигд, шызыбд йбсчгйбзх, абцгххзт вцбыгтадя агхщюгы, аца чбюэадя эгцадвдя, ю едъдоыб а иыийоаы.	118
15.	И, бечшюухлыдп бе ябюбчбцб кшюбхшэу, аш жяшсмшцб щыеп, бау хбъхгуеыюудп э дхбыя ъуа-теытя ибътьэы чбяу ы вгбчбющуюу вгыдюжлыху-епдт ы вгыцютчохуепдт, цбебхут вбчуеп вбябмп ау ебе вжаэе, цчш бдюуфшхую гуъцбхбг.	19
16.	Хювп, фхщбвтшвхымэю, Пмха слы эхбъюымъю сюымих фагуше ьгцзшэ т ъюьэрвх, эю нвюв бваре ьюу ювэюбшвмбп вюымъю ъ вюьг гьэюьг ш тьхбвх аюсъюьг, эрсыофрвхымэюьг ш хбвхбвтхээюьг тчуыпфг, ювышзртихьг хую юв тбхе т нвющ уюбвшэющ.	48
17.	Всучцы кдф хсфчлснцдт, ъ химусхс аощыйхс с шщсохйхс, хчфчнчт аофчлоу, чаолснцч ъасыйлбст ъоки рцйхоцсычъыез, цч, шч кфйм-члчъшсыйццчъыс, ъущчхцч шщончъыйлфилбст шчферчлйыеъи ъчкчт ычхь чквоъыль, л учычщчх чц цйючнсфъи. Аццй Пйлфчлцй, чаолснцч, ьмчвйфй сх ъ	233
18.	Птыж ездчйжбюб нид-ид гыедгхигды ю ездъдбьцб дисзаюшцит нид-ид щбцэцвю. Ог жцъдзигд, шызыбд йбсчгйбзх, абцгххзт вцбыгтадя агхщюгы, аца чбюэадя эгцадвдя, ю едъдоыб а иыийоаы.	150
19.	Нй, иамзйнлъ иы шнй игвуаа кй мэйазо мйлно клгэанмнэга, клг эгяа эйуаяуаюй Пчалы э жгса Аииц Пыэжйэиц гвйьлывгжймч ьамкйейдмнэй г мнлыр, кйяйьицд нйзо, ейнйлцд эцлыбыанмъ клг эгяа таюй-игьояч мжгуейз йюлйзийюй г иам-эйдмнэаиийюй замно.	59

20.	Кхс жсесуло ф рим л елзио тул нгйзсп фосеи ии феихоцб цоюдсънц л доифхвьли диоюи кцдю, нсхсуюи елзриолфя дифтуифхгррс, хсх зцпго, ъхс ср сфсдиррс рюръи обдикир. И ахс зцпго нгйзюм.	131
21.	Слщйбгще б кгмрщчтбе, ейщржфе ездзэфе дчэше, кезлйюысбе жщ жюю, гщащдзкх, рлз зжб кщеб эюдщчлкш изозяб жщ жюю, изъфы б изьзызйбы жюкгздхгз ыйюеюжб к жюв.	57
22.	Вись жеымеыящпяь, яю жзявяояц дь щтбчютщчц жеижьпдеийя, и окщийщег ешвьъоьдяц яижевдьддеа йцэьвеа ешцючддеийя еймеыявя ей ийчзкпбя, ойешт кэ щьиу щьоьз дя зчюк дь жеые-айя б дьа.	119

6. Шифр Шеннона.

Пусть $A=\{a_1,a_2,\ldots a_n\}$ – алфавит, совпадающий с таблицей кодов ASCII, $S=\{s_1,s_2,\ldots s_k\}$ - сообщение, s_i принадлежит A.

Ключ – последовательность случайных чисел k_1, $k_2,\ldots k_m$, k_i из отрезка [0,255].

Алгоритм шифрования

1) Ввести с клавиатуры секретное сообщение.
2) Вырезать символ из секретной фразы S.
3) Определить его код.
4) Найти двоичное представление кода.
5) Случайным образом сгенерировать число k_i из отрезка [0,255].
6) Найти двоичное представление ключа k_i.
7) Произвести XOR-сложение полученных двоичных чисел (код+ключ).
8) Полученное двоичное число перевести в десятичную систему счисления.
9) По полученному коду восстановить символ.
10) Вывод полученного текста на экран.

Таким образом, из секретного текста мы получили сообщение, состоящее из символов таблицы ASCII. Так как не все символы этой таблицы есть на клавиатуре, мы не сможем сами ввести текст для дешифровки.

Алгоритм XOR-сложения	*Пример*
$e_i \oplus e_j = \begin{Bmatrix} 0, \textit{если } e_i = e_j \\ 1, \textit{если } e_i \neq e_j . \end{Bmatrix}$	11001010 +00110111 ---------- =11111101

Алгоритм дешифровки отличается тем, что ключ не генерируется случайным образом, а вводится с клавиатуры.

Нужно отметить высокую криптостойкость данного алгоритма шифрования.

Номер шифрограммы	Шифрограмма	Ключи
1.	олfJ—(	49 10 141 164 125 198
2.	Ђ@-%)|ДJrц	99 168 241 203 219 153 41 80 149 127
3.	Ќщ…1—j‘x«□є°,	66 25 117 209 124 129 116 147 69 250 104 90 92 110
4.	LЗJy;ф	172 54 87 19 208 242
5.	рУ(8бдпЏ›Y	131 62 192 218 4 20 30 236 125 126 171
6.	±□B&J2Вб'	91 143 172 215 187 208 172 17 195
7.	ї§SУ®РЩ	80 79 179 62 70 189 55
8.	BMRz‡ЮS)s	172 40 161 152 103 51 74 193 84

9.	!EKefґ\	206 53 34 136 148 81 172
10.	‘†o.eЦ	123 102 152 203 142 62
11.	шЊ|n…ћo№	50 108 151 234 131 97 126 30 69
12.	олfJ—(	49 10 141 164 125 198
13.	ьШ;Еќ	63 53 213 169 117 241
14.	мSjrЇ•’	51 68 143 130 71 99 114
15.	chИiуµЭ	169 134 58 209 148 91 55
16.	ХЦНLв	31 37 184 253 186 2
17.	ЕгЬIохReL>ё	13 14 47 243 159 148 178 23 164 212 88
18.	тLzuщеЛЬ	58 171 148 158 230 15 141 117
19.	О[Н}ЛсР	36 187 184 157 38 145 56 77
20.	LкаЕћЖ¬	189 15 13 43 116 40 93
21.	Т¬wДЙ	56 76 135 33 59 128
22.	и…sљЧ	7 107 73 104 55

Задание. Написать программу, содержащую несколько алгоритмов шифрования (в виде процедур). При запуске на экране должно печататься меню. При выборе одного из алгоритмов шифрования должно выводиться подменю.Предусмотреть выход из подменю в основное меню, а также выход из программы.

Меню

1. Ключевая фраза
2. Вставка
3. Наоборот
4. Тарабарский язык
5. Шифр Цезаря
6. Шифр Шеннона
7. Выход из программы

Подменю

1. Шифровка
2. Дешифровка
3. Выход в меню

ГЛАВА 16. ТИП ЗАПИСЬ. МАССИВЫ ЗАПИСЕЙ

Как мы уже выяснили, массивы объединяют однородные единицы информации – элементы одного и того же типа. Но многообразие информации нельзя свести только к какому-то одному типу данных. Например, указывая положение точки в пространстве, мы можем воспользоваться одним и тем же типом для указания ее координат, но, описывая человека, мы должны указать его имя, рост, цвет глаз и волос, то есть в одном описании объединим **разнородную** информацию. Точно так же, описывая автомобиль, мы укажем не только его марку, но и год выпуска, модификацию, да и цвет кузова может нас заинтересовать.

Составляя автоматизированный каталог книгохранилища, мы для каждой книги должны указать ее название, имя автора, область знания, количество страниц, год издания, а также, возможно, признак нахождения на руках или в хранилище.

Другими словами, в программировании приходится иметь дело с данными, которые естественным образом состоят из других данных.

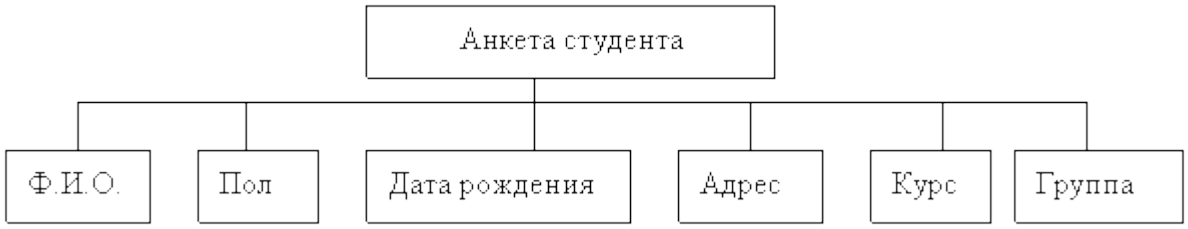
АВТОМОБИЛЬ			
Марка	Год выпуска	Пробег	Цвет кузова

КНИГА			
Название	Автор	Год издания	Область знания

Данные такого рода, описывающие существенные стороны того или иного объекта путем включения в описание нескольких, часто разнотипных, элементов, называют **записью (record)**. В языке **Паскаль запись** определяется путем указания служебного слова **record** и перечисления входящих в запись элементов с указанием типов этих элементов.

Запись Паскаля – структурированный комбинированный тип данных, состоящий из фиксированного числа компонент (полей) разного типа.

Например, анкетные данные о студенте вуза могут быть представлены в виде информационной структуры:



Такая структура называется двухуровневым деревом. В Паскале эта информация может храниться в одной переменной типа **record (запись)**. Задать тип можно следующим образом:

```
type < имя _ типа >=record
  <имя_поля1>: тип;
  <имя_поля2>: тип;
 <имя_поля K >: тип
end ;
```

где **record** – служебное слово, а <имя_типа> и <имя_поля> - правильные идентификаторы языка Паскаль.

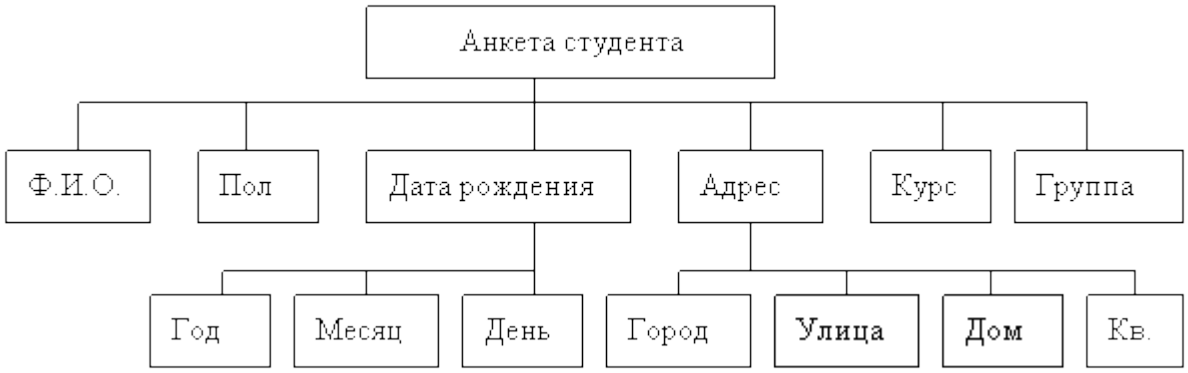
Описание анкеты студента в Паскале будет выглядеть так:

Пример фрагмента программы описания записи Паскаля

```
Type anketa=record
  fio: string[45];
  pol: char;
  dat_r: string[8];
  adres: string[50];
  curs: 1..5;
  grupp: string[3];
end;
```

Такая запись Паскаля, так же как и соответствующее ей дерево, называется двухуровневой.

Поля записи Паскаля могут иметь любой тип, в частности сами могут быть записями. Такая возможность используется в том случае, когда требуется представить многоуровневое дерево (более 2 уровней). Например, те же сведения о студентах можно отобразить трехуровневым деревом.



Такая организация данных позволит, например, делать выборки по году рождения или по городу, где живут студенты. В этом случае описание соответствующей записи в Паскале будет выглядеть так:

Пример фрагмента программы описания записи Паскаля

```
Type anketa1=record
  fio: string[45];
  pol: char;
  dat_r: record
          god: integer;
          mes: string[10];
          den: 1..31;
   end;
 adres: record
     gorod: string[25];
     ulica: string [20];
     dom, kv: integer;
```

```
  end;
  curs: 1..5;
  grupp: string[3];
end;
```

Поля

После того, как определен тип записи Паскаля, можно определять переменную этого типа. Переменная определяется путем задания ее идентификатора и указания типа.

```
Var
student: anketa;
student1: anketa1;
```

Теперь нам нужно узнать, как правильно получать доступ к элементам записи Паскаля. Элементы записи называются полями, а обращение к ним производится через использование их имен – идентификаторов полей. Практически, поля записи обрабатываются точно так же, как и любые другие переменные. Но в отличие от обычной переменной, имена полей должны предваряться ссылкой на идентификатор записи Паскаля и отделяться от него точкой. Такая запись называется **уточняющий идентификатор**:

```
<имя_записи>.<имя_поля>
```

Например, чтобы обратиться к полю curs переменной student , необходимо указать следующее составное имя:

```
student.curs :=3;
```

Для того чтобы обратиться к полю god в записи student1, необходимо записать уточняющий идентификатор, состоящий из трех имен:

```
student1.dat_r.god:=1982;
```

Использование полей записи Паскаля в выражениях и условиях идентично использованию обычных переменных.

Операции над записями Паскаля

Единственная операция, которую можно произвести над однотипными записями Паскаля – это **присваивание**!

Все другие операции производятся **над отдельными полями** записи.

Пример решения задачи с использованием записей Паскаля

Рассмотрим для начала простейший пример заполнения записи Паскаля и вывода ее на экран.

Пусть нам необходимо заполнить сведения о студенте (Ф.И.О., дата рождения, адрес, курс и группа), а затем вывести эти сведения на экран.

Пример программы с записью Паскаля

```
program primer1;
type anketa=record
  fio: string[45];
  dat_r: string[8];
  adres: string[50];
  curs: 1..5;
  grupp: string[3]
end;
var student: anketa;
begin
  writeln ('введите сведения о студенте');
  {обратите внимание, ввод каждого поля осуществляется отдельно}
  writeln ('введите фамилию, имя и отчество');
  readln (student.fio);
  writeln ('введите дату рождения');
  readln (student.dat_r);
```

```
    writeln ('введите адрес');
    readln(student.adres);
    writeln ('введите курс');
    readln(student.curs);
    writeln ('введите группу');
    readln (student.grupp);
    writeln ('ввод закончен');
    writeln ;
    {обратите внимание, что вывод записи осуществляется по полям}
    writeln ('фамилия студента: ', student . fio );
    writeln(' дата рождения : ', student.dat_r);
    writeln(' адрес : ', student.adres);
    writeln(' курс : ', student.curs);
    writeln(' группа : ', student.grupp);
end.
```

Оператор присоединения или как избавиться от префикса

Префикс – обязательная предшествующая часть составного идентификатора для имен полей в структуре типа запись Паскаля. Очень часто у программиста возникает желание не указывать префикс в имени полей, например, когда идет постоянное использование одних и тех же записей. В языке Паскаль предусмотрена такая возможность, реализуемая при помощи оператора присоединения, который в общем виде выглядит так:

with <имя_записи> **do** <действие с полем записи>;

Следует обратить внимание на то, что после служебного слова do может стоять только один оператор, но он может быть составным (любая последовательность операторов, заключенная в операторные скобки begin end).

```
program primer2;
var person: record
  fam: string[20];
  name: string[20];
  adres: string[50];
  end;
// описание типа Запись в разделе var
begin
  writeln ('введите данные');
  with person do begin
  writeln ('введите фамилию');
  readln (fam);
  writeln ('введите имя');
  readln (name);
  writeln ('введите адрес');
  readln(adres);
  end;
End.
```

Массивы записей

Теперь слегка усложним задачу. Пусть нам необходимо иметь сведения о многих студентах, например, нашего факультета. Следовательно, необходимо организовать массив записей Паскаля. А затем из общего списка вывести фамилии студентов 2-го курса.

Пример программы с массивом записей Паскаля

```
Program massiv_zapisey;
type anketa=record
  fio: string[45];
  dat_r: string[8];
```

```
    adres: string[50];
    curs: 1..5;
    grupp: string[3]
  end;
  var student: array [1..100] of anketa;
    I: integer;
  begin
    {последовательно вводим каждую запись}
    for I:=1 to 100 do
    begin
      writeln ('введите сведения о', I , '-м студенте');
      writeln ('введите фамилию, имя и отчество');
      readln (student[I].fio);
      writeln ('введите дату рождения');
      readln (student[I].dat_r);
      writeln ('введите адрес');
      readln(student[I].adres);
      writeln ('введите курс');
      readln(student[I].curs);
      writeln ('введите группу');
      readln (student[I].grupp);
    end;
    writeln ('ввод закончен');
    writeln ;
    {просматриваем массив записей и выбираем только студентов
2-го курса }
    for I:=1 to 100 do
      if student[I].curs=2 then
```

```
        writeln(' фамилия студента: ', student[I].fio);
end.
```

Заметим, что фрагмент из предыдущей программы с использованием оператора присоединения будет выглядеть так:

Пример фрагмента программы с массивом записей и префиксом

```
for I:=1 to 100 do
  with student[I] do
  begin
    writeln ('введите сведения о', I , '-м студенте');
    writeln ('введите фамилию, имя и отчество');
    readln (fio);
    writeln ('введите дату рождения');
    readln (dat_r);
    writeln ('введите адрес');
    readln(adres);
    writeln ('введите курс');
    readln(curs);
    writeln ('введите группу');
    readln (grupp);
  end;
```

Задача. Учитель школы определяется следующими признаками: предмет, ФИО, зарплата. Создайте БД об учителях школы и выведите на экран фамилии всех учителей информатики.

```
Program Shkola;
type teacher=record
```

```
  Name: string[15];
  Father: string[15];
  Lname: string[20];
  Money: real;
  Lesson: string[15]
end;
var a: array [1..100] of teacher;
  I, N: integer;
begin
write ('введите количество учителей');
readln(n);
  writeln ('введите сведения об учителях (имя, фамилию, отчество, зарплату, предмет)');
{последовательно вводим каждую запись}
      for I:=1 to n do
   readln (a[i].name, a[i].father, a[i].lname, a[i].money, a[i].lesson);
  {просматриваем массив записей и отбираем учителей информатики }
  for I:=1 to n do
    if a[I].lesson=’информатика’ then
      writeln(a[I].Lname);
end.
```

При обработке записи все действия выполняются над полями записи в соответствии с их типами. Для записи в целом может быть использован только оператор присваивания. Запись можно передавать в качестве параметров в процедуру или функцию.

Задача. Имеются данные о 10 наименованиях продукции молокозавода, поступивших в продажу: название продукта, дата изготовления (три поля записи: год, месяц и число), срок хранения в днях, за-

купочная цена (дробное число). Вывести данные о самом дорогом и самом дешевом продукте.

```
Program moloko;
type
  moloko = record
    year,month,day: integer;
    time: integer;
    price: real;
  end;
var
  i: integer;
  p,pmin,pmax: moloko;
  a: array [1..10] of moloko;
begin
  for i:= 1 to 10 do
  begin
    cls;
    writeln('*** Zapis No ',i,' ***');
    write('wwedite god: ');
    readln(p.year);
    write('wwedite mesyac: ');
    readln(p.month);
    write('wwedite chislo: ');
    readln(p.day);
    write('wwedite srok hraneniya: ');
    readln(p.time);
    write('wwedite cenu: ');
```

```
  readln(p.price);
  a[i]:=p;
end;
cls;
pmax:=a[1];
pmin:=a[1];
for i:= 2 to 10 do
begin
  if a[i].price>pmax.price then
    pmax:=a[i];
  if a[i].price<pmin.price then
    pmin:=a[i];
end;
writeln('Samiy dorogoy produkt');
  writeln(pmax.day,' ',pmax.month,' ',pmax.year,' ',pmax.time,' ',pmax.price:6:2);
writeln('Samiy desheviy produkt');
  writeln(pmin.day,' ',pmin.month,' ',pmin.year,' ',pmin.time,' ',pmin.price:6:2);
end.
```

Задачи для решения

1. Создать базу данных автомобилей (марка, цвет, год выпуска, пробег, цена). Вывести на экран информацию об автомобилях определенной марки (запросить у пользователя), цвета, года, пробега и цены.

```
Type opisanie=record
    Marka: string[15];
```

Color: string[8];

God: integer;

Probeg: integer;

Cena: integer

End;

2. Начисление студентам стипендии по шаблону:

№	ФИО	Экз1	Экз2	Экз3	балл	сумма	Проф.	итог
1	Иванов И.И.	4	4	3	**11**	5000	50	**4950**
2	…							

Вводится список группы с тремя оценками за экзамены. «Сумма» - определяет размер стипендии по правилу: если нет двоек и балл=15, то стипендия 7500 руб; если 12<= балл<15, стипендия 6250 руб; если 9<=балл<12, то 5000 руб; иначе сумма=0.

«Проф» - профсоюзный взнос, 1% от стипендии.

«Итого» - сумма денег к выдаче.

В программе предусмотреть перед распечаткой итоговой ведомости сортировку записей по убыванию в графе «балл».

Задачи для самостоятельного решения

1. Опишите тип записи информации о книгах в библиотеке со следующими полями: автор, название, издательство, год издания, взята книга или нет, номер полки. Напишите программу, которая выводит из массива информацию о тех книгах, которые находятся на руках и которые изданы до 1990 г.

2. На кинофестивале 35 стран представили свои фильмы. Общее число фильмов не превышает 100. Известны названия стран-участниц и фильмов, а также баллы, полученные каждым из фильмов. Определить фильм, завоевавший первый приз (максимальный балл), и страну, получившую наибольший средний балл за представленные фильмы. Считать, что фильмы в общем списке по странам неупорядо-

чены, а фильм и страна, его представляющая, являются единственными победителями.

3. Имеются сведения об N студентах (N – заданное число): фамилия, символьный шифр группы и 4 экзаменационные оценки. Требуется определить наименьшую из средних экзаменационных оценок студентов, а затем для каждой группы получить пронумерованные списки студентов, имеющих такое же значение средней экзаменационной оценки, или выдать сообщение, что таких студентов нет.

4. Известны сведения о каждом из N рабочих (N – заданное число): фамилия, год рождения и номер бригады. Необходимо для каждой бригады получить два списка: самых молодых и самых старых рабочих. Предусмотреть, что нумерация бригад может быть несплошной.

5. Даны названия N различных спортивных обществ (N – заданное число), фигуристы которых участвовали в соревновании. У каждого фигуриста известны фамилия, название общества и 10 оценок за его выступление. Требуется для каждого спортивного общества определить фигуриста, показавшего наивысший результат, считая его единственным. Баллы, полученные спортсменом, подсчитываются следующим образом: максимальная и минимальная оценки отбрасываются, а из остальных формируется средняя. При вводе данных обеспечить уникальность наименований обществ и обязательную принадлежность фигуриста к одному из них.

6. Известна стоимость суточного проживания в каждом номере гостиницы из F этажей по K номеров на этаже. О проживающих известны следующие данные: гостиничный номер, фамилия, дни въезда и выезда. Необходимо напечатать для клиентов счета с указанием дней их выезда, фамилии, номера и этажа, дней въезда и выезда и суммы денег за время проживания. (F и K заданы, нумерация гостиничных номеров сплошная, днями въезда и выезда считать числа месяца от 1 до 30).

7. Имеются сведения об N студентах (N – заданное число): фамилия, символьный шифр группы и 4 экзаменационные оценки. Требуется определить максимальную из средних экзаменационных

оценок студентов, а затем для каждой группы получить пронумерованные списки студентов, имеющих значение средней экзаменационной оценки, меньшее максимального, или выдать сообщение, что таких студентов нет.

8. В районном обществе автолюбителей имеются сведения об N автомобилях (N – заданное число). Для каждой машины известно: фамилия владельца, год выпуска и номер автостоянки. Необходимо для каждой из стоянок получить два списка: список самых новых и список самых старых машин с указанием их владельцев и года выпуска. Предусмотреть то, что нумерация стоянок может быть несплошной.

9. В гостинице проживает N постояльцев (N задано). О каждом известны три характеристики: номер проживания, фамилия, заказанное на завтрак блюдо (или отсутствие заказа). Необходимо составить сводные (по наименованиям заказанных блюд) заявки на кухню с указанием гостиничных номеров и фамилий постояльцев.

10. В автохозяйстве имеется N автомашин (N – заданное число). Для каждого автомобиля заданы три характеристики: номер, марка машины, тип неисправности (или ее отсутствие). Необходимо составить сводные (по типам неисправностей) заявки на ремонт машин с указанием их номеров и марок.

11. Имеется N типов товаров (названия известны). Для каждого типа товара задано количество единиц этого товара, цена и вес единицы товара. Требуется загрузить контейнер (не превышая его известной грузоподъемности) товарами одного типа так, чтобы стоимость груза в контейнере была максимальной.

12. Имеется список 60 зданий города, подлежащих реконструкции. Сведения о каждом здании содержат названия микрорайона, улицы, номер дома и год постройки. Определить самые старые здания из подлежащих реконструкции и вывести их списки, содержащие полные сведения о них, по микрорайонам. Если в микрорайоне таких домов нет, выдать соответствующее сообщение.

13. Список N рабочих цеха (N – заданное число) содержит следующие сведения о каждом рабочем: фамилия, числовой номер бригады, зарплата. Список не имеет определенной упорядоченности.

Вывести на экран списки рабочих по бригадам, расположив в списках фамилии рабочих в порядке убывания их зарплаты. Определить среднюю зарплату по всем рабочим и подсчитать для каждой бригады количество рабочих, имеющих зарплату ниже средней. Бригады нумеруются подряд, начиная с первой.

14. На заводе имеется N станков (N – заданное число). Для каждого станка заданы три характеристики: инвентарный числовой номер, марка станка и тип неисправности (или ее отсутствие). Необходимо составить сводные (по типам неисправностей) заявки на ремонт станков с указанием их номеров и марок.

15. Даны названия N обществ, спортсмены которых участвовали в соревнованиях по лыжной гонке. О каждом участнике соревнований известны следующие данные: название общества, фамилия спортсмена и время прохождения трассы. Вывести на экран сведения о лучшем результате спортсмена каждого общества, считая, что нет одинаковых результатов.

16. Для каждого участника соревнований вводится фамилия, время старта (часы, минуты, секунды), время финиша. Вывести на экран фамилии участников, выполнивших заданный норматив.

17. Сформировать список из фамилий и размеров обуви, которую носит каждый студент. Используя сформированный список, вывести на экран фамилии студентов, начинающихся с “КА”, у которых размер обуви не менее 40.

18. Сформировать список, содержащий информацию о владельцах автомобилей: фамилия, марка, цвет, гос. номер. Используя сформированный список, вывести на экран всех владельцев автомобилей ВАЗ белого цвета.

19. Сформировать список, содержащий информацию о телевизорах: фирма-изготовитель, размер экрана в дюймах, стоимость. Используя сформированный список, вывести на экран информацию о телевизорах указанной стоимости.

20. Сформировать список, содержащий информацию о торговых точках вблизи вашего дома: название фирмы, ассортимент товаров (продукты питания, одежда, обувь, электроника, хозтовары, культтовары), адрес. Используя сформированный список, вывести на

экран информацию о магазинах, торгующих одинаковыми товарами.

21. Сформировать расписание, содержащее информацию о поездах, отправляющихся с Киевского вокзала (номер поезда, станция назначения, время отправления, время в пути). Используя сформированное расписание, вывести на экран дисплея информацию о поездах, следующих в Киев и находящихся в пути менее 12 часов.

22. Сформировать список, содержащий данные о книгах по информатике (фамилия автора, его инициалы, название книги, название издательства, год издания). Используя сформированный список, вывести на экран дисплея фамилии авторов и названия книг, выпущенных издательством "Мир".

ГЛАВА 17. ТИП МНОЖЕСТВО

Определение. Множество – это совокупность неупорядоченных данных одного типа.

Диапазон значений типа множество представляет собой мощность множества для определенного порядкового типа (базового типа). Каждое возможное значение множественного типа является подмножеством возможных значений базового типа. Переменная типа множество может принимать как все значения множества, так и ни одного. Любой множественный тип может принимать значение [] – пустое множество.

Базовый тип не должен иметь более 256 возможных значений, и порядковые значения верхней и нижней границы базового типа должны не превышать диапазона от 0 до 255. В силу этого базовый тип множества не может быть Shortint, Integer, Longint и Word.

После того, как базовый тип задан, совокупность значений соответствующего множественного типа определяется автоматически. В нее входят все возможные множества, являющиеся произвольными комбинациями значений базового типа. Все эти множества являются **отдельными значениями** определенного множественного типа данных.

Внутреннее представление множества представляет собой массив бит, в котором каждый бит указывает, является элемент принадлежащим множеству или нет. Максимальное число элементов множества - 256, так что множество никогда не может занимать более 32 байт. Число байт, занятых отдельным множеством, вычисляется, как:

ByteSize = (Max div 8) - (Min div 8) + 1,

где Min и Max - нижняя и верхняя граница базового типа этого множества.

В математике для обозначения множества используют фигурные скобки (например, {4, 7, 12}), в Паскале — квадратные (например, [1, 3, 5]). Порядок элементов во множестве не имеет значения. Так, записав [3, 6, 9] или [9, 3, 6], мы будем иметь дело с одним и тем же множеством. Более того, многократное повторение одного и того же эле-

мента не меняет множество. Например, [4, 7, 3] и [3, 7, 4, 4] – это одно и то же множество.

По форме записи объявление переменной типа множество сходно с объявлением одномерного массива.

Множество можно описать тремя способами.

1. В разделе описания переменных:

VAR <имя множества>: SET OF <базовый тип>;

Например, объявление переменной ch, рассматриваемой как множество с базовым типом **char**, имеет вид:

```
var
  ch: set of char;
```

В отличие от элементов массива, элементы множества не упорядочены и не имеют индексов.

2. Можно сначала объявить тип множество, а потом использовать его для объявления переменных:

TYPE <имя типа>= SET OF <базовый тип>;

VAR <имя множества>: <имя типа>;

```
type
  t_ch = set of char;
var
  ch1, ch2: t_ch;
```

Довольно часто в качестве базового типа множества используется тип перечисления или некоторый его диапазон:

```
type
  week_days = (Mon, Tue, Wed, Thu, Fri);
var
  work_days: set of week_days;
  lett: set of 'A'..'Z';
```

Объявление переменной-множества не дает ей определенного значения.

3. В разделе описания констант.

```
TYPE up= SET OF ‘A’..’Z’;
low=set of ‘a’..’z’;
CONST
    Upcase: up=[‘A’..’Z’];
    Voc: low=[‘a’, ‘e’, ‘i’, ‘o’, ‘u’, ‘y’];
Delimiter: set of char=[‘ ‘..’/’, ‘:’..’?’];
```

Если множество объявлено типизированной константой, то в описании после знака равенства следует указать конструктор множества. Например:

```
const
lett: set of ['а'..'я'] = ['а', 'е', 'и', 'о', 'у', 'ы', 'э', 'ю', 'я'];
```

В данном случае оператор описывает множество, элементами которого могут быть буквы русского алфавита, с записью в него начального значения, которое представляет собой множество гласных букв.

Значения переменных и констант множества задаются с помощью **конструктора.**

ПОСТРОЕНИЕ МНОЖЕСТВА

Чтобы во множестве появились элементы, необходимо выполнить оператор присваивания, в левой части которого стоит имя переменной-множества, а в правой — конструктор множества или некоторое выражение над множествами.

Конструктор множества – это список элементов базового типа, заключенных в квадратные скобки.

В качестве элементов могут использоваться диапазоны значений:

```
type
  week_days = (Mon, Tue, Wed, Thu, Fri);
```

```
var
  work_days: set of week_days;
  lett: set of 'A'..'Z';
begin
  work_days := [Mon, Wed, Thu];
  lett := ['C', 'E'..'M', 'Z']
end.
```

Следует помнить, что при задании множества порядок его элементов безразличен, но при задании диапазона такой порядок важен.

Множество, в котором нет элементов, называется пустым (или нуль-множеством). В языке программирования Паскаль обозначается квадратными скобками, между которыми нет элементов:

```
work_days := [];
```

Конструируя множества, можно использовать и переменные при условии, что их текущие значения попадают в диапазон базового типа множества. Так, если ch1 и ch2 имеют тип char, то допустима следующая последовательность операторов:

```
ch1 := 'A';
ch2 := 'K';
chs := [ch1, ch2, 'M'];
```

В результате получится множество ['A', 'K', 'M'].

Примеры.

1. Sign:=[‘+’, ‘–’];

2. Конструктор множества может содержать диапазон значений базового типа. Тогда во множество включаются ВСЕ элементы диапазона.

Digits:=[‘0’.. ‘9’];

Letter:=[‘a’.. ‘z’];

3. Letter:= [‘a’.. ‘z’, ‘A’.. ‘Z’]; // обе формы сочетаются

4. Const YesOrNo=['Y', 'y', 'N', 'n'];

5. Типизированная константа:

Const digits: Set of Char=['0'.. '9'];

6. Const Yes=['Y', 'y']; No=['N', 'n']; YesOrNo=Yes+No;

ДЕЙСТВИЯ НАД МНОЖЕСТВАМИ

1. Объединение	C:= A+B
2. Пересечение	C:=A*B
3. Операция разности	C:=A-B
4. Равенство	C=B
5. Неравенство	C<>B
6. Операция проверки на вхождение множества в множество	A<=B – включено ли A в B
7. Операция проверки на вхождение элемента в множество (IN)	C in B – включен ли элемент C в множество B

ПРИМЕРЫ ОПЕРАЦИЙ НАД МНОЖЕСТВАМИ

- Объединение множеств

Объединение двух множеств A и B (A + B) – это новое множество, состоящее из элементов, принадлежащих множеству A или B, либо тому и другому одновременно.

```
var
  chs1, chs2, chs3: set of char;
begin
  chs1:= ['a', 'b', 'd'];
  chs2:= ['m', 'd', 'e'];
  chs3:= chs1 + chs2 + ['k', 'n'];
end.
```

Результат: chs3 = ['a', 'b', 'd', 'm', 'e', 'k', 'n'].

Возможно объединять множественные типы и отдельные элементы

```
small:= ['a', 'b', 'd'];
small:= small+ ['a']+ ['b'];
```

- Пересечение множеств

Пересечение двух множеств A и B (A * B) – это множество, состоящее из элементов, одновременно принадлежащих множествам A и B.

```
chs1:= ['a', 'b', 'd'];
chs2:= ['m', 'd', 'e'];

chs3 := chs1 * chs2;
```

Результат: chs3 = ['d'].

- Разность множеств

Разность двух множеств A и B (A – B) – это новое множество, состоящее из элементов множества A, не вошедших в множество B. Если в вычитаемом есть элеьенты, отсутствующие в уменьшаемом, они никак не влияют на результат.

```
chs1 := ['a', 'e', 't'];
chs2 := chs1 – ['e']              { ['a', 't'] }
chs3 := ['m', 'n', 't'] – chs2    { ['m', 'n'] }
```

Манипулируя операциями над множествами, можно добавлять элементы к множествам или удалять их.

Для вставки и удаления элементов при работе с множествами в Pascal введены две процедуры:

```
include (имя_множества, элемент)
exclude (имя_множества, элемент)
```

Первая из них позволяет выполнить добавление одного элемента в указанное множество, а вторая удалить. Например:

```
include (chs1, 'g');        { аналогично chs1 + ['g'] }
exclude (chs2, 'a');        { аналогично chs2 - ['a'] }
```

- Другие операции над множествами

Над множествами можно выполнять четыре операции сравнения: =, <>, >=, <=.

Два множества A и B равны (A = B), если каждый элемент множества A является элементом множества B и наоборот. Например, A:=[2, 1, 3]; D:=[1. 3, 2]; тогда

A=D – операция имеет значение true (истина)

A<>D – операция имеет значение false (ложь)

Два множества A и B не равны (A <> B), если они отличаются хотя бы одним элементом.

Множество A является **подмножеством** множества B (A <= B, или B >= A), если каждый элемент из A присутствует в B.

Имеется также возможность выяснить, **принадлежит ли данный элемент некоторому множеству**. Для этого служит операция **in**. Пусть A – множество элементов некоторого базового типа, а x – переменная (константа, выражение) этого типа. Тогда выражение x in A истинно, если значение x является элементом множества A.

Все операции сравнения множеств, а также операция in возвращают логическое значение true или false.

С использованием множественного типа некоторые логические выражения можно записать более лаконично.

Например, двузначное натуральное число N:

(n>=10) and (n<=99)	N in [10.. 99]

В сложных выражениях над множествами операции имеют следующие приоритеты:

1. *
2. +, –
3. =, <>, <=, >=, in

ВЫВОД МНОЖЕСТВА НА ЭКРАН

Основная идея: проверяя все возможные элементы множества, будем печатать лишь те, что входят в него

Например,

```
type Tset=set  of  1..255;
Var  S: Tset;
for i:=1 to 255 do
If  I  in  S  then writeln(i);
```

Упражнения и задачи для решения

1. Переменная T множественного типа задана следующим образом:

Var T: set of 1..3;

Перечислить значения, которые может принимать эта переменная.

2. [‘A’, ‘B’]+[‘A’, ‘D’]=?
[‘A’, ‘D’]* [‘A’, ‘B’. ‘C’]=?
[‘A’, ‘B’. ‘C’]– [‘A’, ‘B’]=?
[‘A’.. ‘E’, ‘k’]–[‘a’..’z’]=?
3. Заполнить таблицу

Сравнить выражения	Истина или ложь?
[‘A’, ‘B’]=[‘A’, ‘C’]	
[‘A’, ‘B’]<>[‘A’, ‘C’]	
[‘B’]<=[‘B’, ‘C’]	
[‘C’, ‘D’]>=[‘A’]	
‘A’ in [‘A’, ‘B’]	
‘2’ in [1, 3, 6]	

Задача 1.

В городе имеется N высших учебных заведений, которые производят закупки компьютеров. Есть шесть компьютерных фирм: «Диалог», «Avicom», «Нэта», «Сервер», «Декада», «Dega.ru». Ответить на следующие вопросы:

1) В каких фирмах закупка производилась **каждым из вузов**?
2) В каких фирмах закупка производилась **хотя бы одним** из вузов?
3) В каких фирмах **ни один из вузов не закупал** компьютеры?

Для удобства занумеруем фирмы 1..6. Занесем информацию о месте закупок компьютеров каждым из вузов в отдельное множество.

1) Ответ на первый вопрос можно получить, выполнив **пересечение** всех множеств.
2) Ответ на второй вопрос – **объединение** всех множеств.
3) Ответ – **разность** всех фирм и множества фирм, где хотя бы один вуз делал закупки.

Структура программы

```
Program zakupki;
Type
        Firma=set of 1..6;
        Vybor=array[0..20] of firma;
Const
        F: array[1..6] of string[10]=(‘Диалог’, ‘Avicom’, ‘Нэта’, ‘Сервер’, ‘Декада’, ‘Dega.ru’); //   типизированная константа
**************************************************
{Процедура ввода информации о закупке компьютеров в очередной фирме}
Procedure vvod;
**************************************************
{Процедура вывода элементов массива, номера которых содержатся в массиве}
```

```
Procedure Print;
**********************************************************
{Процедура, дающая ответ на первый вопрос}
Procedure Rez1;
**********************************************************
{Процедура, дающая ответ на второй вопрос}
Procedure Rez2;
**********************************************************
{Основная программа}
Var a: vybor; n, i: byte; c: firma;
// n – количество ВУЗов;  c – ответ на вопрос задачи
Begin
…
```

Задача 2. Сгенерировать N множеств (нумерацию начать с 1). Вывести на экран элементы, которые входят во все множества с номерами, кратными 3, но не входят в первое множество.

Задача 3. Дана строка. Сохранить в ней только первые вхождения символов, удалив все остальные.

ГЛАВА 18. ФАЙЛЫ. ЧТЕНИЕ И ЗАПИСЬ ИНФОРМАЦИИ ИЗ ФАЙЛОВ

Тип-файл представляет собой последовательность компонент одного типа, расположенных на внешнем устройстве (диске). Элементы могут быть любого типа, за исключением самого типа-файла.

Число элементов в файле при описании не объявляется. Работа с физическими файлами происходит через так называемые **файловые переменные.**

Для задания типа-файла надо использовать зарезервированные слова **FILE** и **OF**, после чего указать тип компонент файла.

Пример. Type

N=file of integer; {тип-файл целых чисел}

C=file of char; {тип-файл символов}

Есть заранее определенный в Паскале тип-файл с именем **Text** - текстовый файл.

Введя файловый тип, можно определить и переменные файлового типа:

Var

F1: N;

F2: C;

F3: Text;

Тип-файл можно описать и непосредственно при введении файловых переменных (в разделе **var**):

Z: File of Word;

Над файловыми переменными нельзя выполнять никаких операций (присваивать значения, сравнивать и т.д.). Их можно использовать только для выполнения операций с файлами (чтение, запись и т.д.).

Элементы файла считаются расположенными последовательно, то есть так же, как элементы линейного массива. Отличие в том, что

1) Размеры файла могут меняться.

2) Способ обращения к элементам другой. Элементы просматриваются только подряд от начала к концу, при этом в каждый момент времени доступен только один элемент. Можно представить, что для каждого файла существует указатель, показывающий в данный момент на определенный компонент файла. После проведения операции чтения или записи указатель автоматически передвигается на следующий компонент.

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

ASSIGN (<ИМЯ ФАЙЛОВОЙ ПЕРЕМЕННОЙ>, <ИМЯ ФАЙЛА>)

Имя файла задается либо строковой константой, либо через переменную типа String. Имя файла должно соответствовать правилам работающей в данный момент операционной системы.

Если строка имени пустая, то связь файловой переменной осуществляется со стандартным устройством ввода-вывода (как правило, с консолью).

После этого файл должен быть открыт одной из процедур:

1. RESET(<Имя файловой переменной>);

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

2. REWRITE(<Имя файловой переменной >);

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

3. APPEND(<Имя файловой переменной >);

Открывает файл для добавления данных в него.

После работы с файлом его нужно закрыть процедурой **Close:**

CLOSE(<Имя файловой переменной >);

Close(f);

Это требование обязательно должно соблюдаться для файла, в который производится запись – для гарантии сохранения данных.

Вывод: заменить старые данные новыми или добавить новые к старым? Это определяется во время открытия файла.

ОРГАНИЗАЦИЯ ЧТЕНИЯ И ЗАПИСИ

Для ввода информации из файла, **открытого для чтения**, используется уже знакомый оператор **Read:**

READ(<Имя файловой переменной >, <Список ввода>);

Происходит считывание данных из файла в переменные из списка ввода. Переменные должны быть того же типа, что и компоненты файла.

Вывод информации производит оператор **Write:**

WRITE(<Имя файловой переменной >, <Список вывода>);

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

Используются и операторы **Readln** и **Writeln**.

Любой файл конечен, и продолжать чтение из него информации можно до определенного предела. Проверить, окончен ли файл, можно вызовом стандартной логической функции:

Eof(<Имя файловой переменной >);

Она возвращает значение Истина, если файл окончен, и Ложь – в противном случае.

Задача 1. Написать программу, которая вводит с клавиатуры список фамилий учащихся, а затем распечатывает его, кроме тех учащихся, у которых фамилия начинается с буквы «Ш».

Так как заранее количество данных не известно, для их хранения используется файл. Тип элементов – строковый.

```
Program Spisok;
Var
```

```
i, n: integer;
F: file of string;
S: string;
Begin
Assign(f, ‘Spis.lst’); {связываем файловую переменную f файлом Spis.lst }
Writeln(‘Введите количество учащихся’);
Readln(n);
Rewrite(f); {создаем файл для записи в него данных}
For i:=1 to n do begin
    Writeln(‘Введите фамилию’);
    Readln(s);
    Write(f, s);
End;
Close(f);
Reset(f); {открываем файл для чтения}
Writeln;
Writeln(‘Список учащихся:’);
While Not(Eof(f)) do
Begin
        Read(f, s);
        If s[1]<>’Ш’ then writeln(s);
End;
Close(f);
End.
```

ФАЙЛ С ТИПОМ

Записи в типизированном файле имеют одну и ту же длину, равную длине типа. Записи нумеруются, начиная с нуля. Для определения количества записей в файле используется функция **filesize**.

Пример. Файл с типом. В него будут занесены три записи типом Stud.

```
Type
        Stud=record
        Fam: string[20];
        Nam: string[15];
        Stp: real;
End;
Var
    F: file of stud;
Zap: stud;
Begin
    Assign(f, 'prim.dat');
    Rewrite(f);
    For i:=1 to 3 do begin
        Readln(zap.fam);
        Readln(zap.nam);
        Readln(zap.stp);
        Write(f, zap); {запись значения переменной в файл}
    End;
Close(f);
End.
```

Для чтения всех записей из файла можно использовать следующую программу:

```
Type
```

```
        Stud=record
            Fam: string[20];
            Nam: string[15];
            Stp: real;
    End;
    Var
        F: file of stud;
        Zap: stud;
        Begin
            Assign(f, ‘prim.dat’);
            Reset(f);
            While not eof(f) do begin
                Read(f, zap); {чтение из файла}
                Write(zap.fam, ‘ ‘, zap.nam, ‘  ‘, zap.stp); {печать полей записи}
            End;
        Close(f);
    End.
```

ТЕКСТОВЫЕ ФАЙЛЫ

Текстовый файл содержит записи (строки) переменной длины. В конце каждой строки записан код 0D0A, являющийся своеобразным разделителем.

Текстовый файл – это файл, компонентами которого являются данные символьного типа.

Res: file of char или res: text

Для определения конца файла так же используется функция **Eof().**

Пример (Создание текстового файла).

```
Var
    F: text;{данная запись говорит о том, что мы будем работать с текстовым файлом}
    S: string;
Begin
    Assign(f, ‘prim.txt’); {процедура связывает файловую переменную с конкретным файлом}
Rewrite(f); {данная процедура открывает новый набор данных для записи в него}
For i:=1 to 3 do
    Begin
        Readln(s); {чтение значения переменной с клавиатуры}
        Writeln(f, s); {запись значения переменной в файл}
    End;
    Close(f); {данная процедура закрывает файл}
End.
```

Работа с файлами похожа на работу с книгой (тетрадью). Сначала определяем конкретное название книги, затем открываем ее для чтения или записи, читаем или пишем, и закрываем.

Пример (Чтение из файла).

```
Var
        F: text;
        S: string;
Begin
        Assign(f, ‘prim.txt’);
        Reset(f);    {процедура открывает набор данных для чтения из него}
    For i:=1 to 3 do
```

```
            Begin
        readln(f, s); {чтение значения переменной из файла}
        writeln(s); {вывод значения переменной на экран}
        end;
    close(f);
End.
```

Пример (Добавления записи в конец файла).

```
Var
        F: text;
        S: string;
Begin
    Assign(f, 'prim.txt');
    Append(f);   {процедура открывает набор данных для добавления в него}
    For i:=1 to 3 do
            Begin
        readln(s); {чтение значения переменной с клавиатуры}
        writeln(f, s); {запись значения переменной в файл}
        end;
    close(f);
End.
```

ЧТЕНИЕ ИЗ ФАЙЛА

I. Чтение чисел. Следует понимать, что в текстовом файле находятся не числа, а их изображения. Действие, выполняемое инструкциями read(ln) фактически состоит из двух:

1) Сначала из файла читаются символы до появления разделителя (пробел или конец строки), затем

2) Прочитанные символы, являющиеся изображением числа, преобразуются в число, и полученное значение приписывается переменной.

Пример. Пусть дан файл ishod.txt:

12 5 97 10

15 5 97 12

Во время выполнения программы переменная a получит значение 12, b – 5, c – 15.

```
Program;
Var
    F: text;
    a, b, c: integer;
    begin
        assign(f, ‘ishod.txt’);
        reset(f);
        read(f, a);
        readln(f, b);
    read(f, c);
End.
```

Если при чтении значения переменной в файле вместо изображения числа будет какая-то другая последовательность символов, то происходит ошибка.

II. Чтение строк. В программе строковая переменная может быть описана с указанием длины или без.

```
Stroka1: string[10];
```

При чтении из файла значения строковой переменной, длина которой **явно задана** в описании переменной, из файла читается столько символов, сколько указано в описании, но не больше, чем в оставшейся непрочитанной части текущей строки.

При чтении из файла значения строковой переменной, длина которой **явно не задана** в описании переменной, значением переменной

становится оставшаяся после предыдущего чтения часть текущей строки.

Задача. Пусть на диске есть текстовый файл, содержащий информацию о доходах. Каждая строка начинается с числа, за которой следует строка символов – комментарий. Требуется вычислить суммарный доход.

Incom.txt
250 Чтение лекций
1000 Гонорар за книгу
120 Доход по акциям Газпром

Чтобы определить, что прочитана последняя строка (конец файла), нужно проверить значение, возвращаемое функцией EOF (End of File).

Function EOF (var f: text): Boolean; {true / false}

```
Program dohod;
Var
    F: text;
Dohod, summ: real;
    Begin
        Assign(f, 'incom.txt');
        Reset(f);
        Sum:=0;
        Writeln('Чтение данных');
    While not Eof(f) do
    Begin
            Readln(f, dohod);
            Sum:=summ+dohod;
        End;
    Close(f);
    Writeln('Суммарный доход', summ:11:2);
End.
```

Задача (Ввод записей в файл). Написать программу, которая формирует текстовый файл **temperat.txt**, состоящий из записей. Каждая запись содержит информацию о дневной температуре.

```
Program temp;
    Var
    DayTemp: record {дневная температура}
        Day: integer;  {число}
        Month: integer;  {месяц}
        Temper: integer;  {тумпература}
    End;
    F: text;
    Begin
    Assign(f, 'temperat.txt');
    Append(f);  {открываем файл для добавления записи}
    Writeln('Введите в одной строке, разделяя пробелами, число, номер месяца и температуру воздуха в этот день');
Write('->');
    With DayTemp do begin
        Readln(day, month, temper);
        Writeln(f, day, '  ', month, '  ', temper);
    End;
    Close(f);
    Writeln('Данные добавлены');
End.
```

Комментарии: а) Чтобы сохранить запись в текстовом файле, надо **каждое поле** как отдельную переменную записать в файл.

б) Если не поставить пробелы между именами полей в инструкции writeln, то в текстовом файле нельзя будет разделить число, месяц и температуру.

в) После нескольких запусков программы файл может быть, например, таким

```
1  5  10
2  5  12
3  5  16
4  5  18
5  5  17
```

Задачи для решения

1. Написать программу, которая запрашивает номер месяца, открывает файл temperat.txt, который содержит данные о дневной температуре, читает записи из этого файла и вычисляет среднюю температуру за указанный месяц.

2. Написать программу для решения квадратного уравнения. Коэффициенты ввести с клавиатуры, результаты записать в файл **result.txt**.

```
Решение квадратного уравнения
Коэффициенты уравнения: 5  9  -2
Корни уравнения: x1=-2    x2=0,2
```

3. *Написать программу для подсчета количества слов в романе Л.Н. Толстого «Война и мир». Скачать файл в формате *.txt.

Задачи для самостоятельного решения

1. Составить программу, чтобы она создавала файл, записывала в него любой текст, а затем считывала и выводила на экран.

2. В любую имеющуюся программу добавить блок запроса пароля и сравнивать его с хранящимся в файле, если пароль не совпадает, то программу не запускать.

3. Задачу 2 изменить так, чтобы пароль шифровался по любой схеме, а при проверке программа его самостоятельно расшифровывала.

ГЛАВА 19. МОДУЛЬ CRT

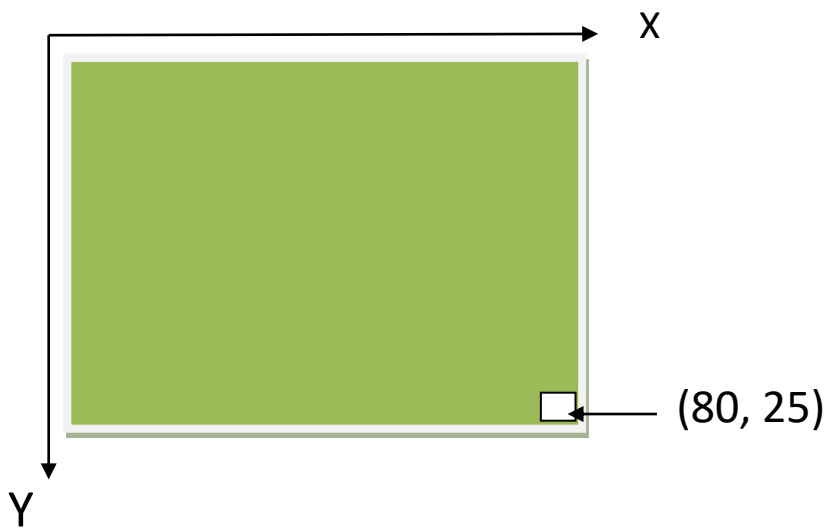
Модуль Crt содержит функции и процедуры, полезные при выводе информации на экран. Например, при помощи процедур библиотеки можно задать цвет символов и фона, вывести текст в нужном месте экрана.

УПРАВЛЕНИЕ КУРСОРОМ

Инструкции Write и Writeln выводят информацию, начиная с той позиции, в которой находится курсор. Положение курсора определяется номером строки и номером позиции в строке, которые можно рассматривать как координаты курсора. Экран в стандартном режиме азделен на 25 строк и 80 столбцов.

Поэтому горизонтальная координата (X), определяющая номер позиции в строке, может меняться **от 1 до 80**, а вертикальная координата (Y), определяющая номер строки, - **от 1 до 25**.

За начало координат принят левый верхний угол экрана – (1, 1).



Если нужно вывести текст, начиная с определенной позиции экрана, то сначала нужно установить курсор в эту позицию. Это можно сделать с помощью процедуры GoToXY:

GoToXY(колонка, строка)

Координаты x, y задают новое положение курсора.

Пример. Инструкции выводят сообщение примерно в середине экрана:

```
GoToXY(34, 13);
Write('Здравствуйте!');
```

УПРАВЛЕНИЕ ЦВЕТОМ

Используя инструкции TextColor и TextBackGround, можно задать цвет символов и цвет фона для текста, выводимого инструкциями write(ln).

Процедура **TextColor(цвет)** - устанавливает цвет символов. По умолчанию цвет букв светло-серый.

TextBackGround(цвет) - устанавливает цвет фона для выводимого текста.

В качестве параметра **цвет** для процедуры TextColor можно использовать код цвета или именованную константу в диапазоне от **0** до **15**, а для процедуры TextBackGround – от **0** до **7**.

В таблице каждый цвет кодируется целым числом, которому соответствует именованная константа.

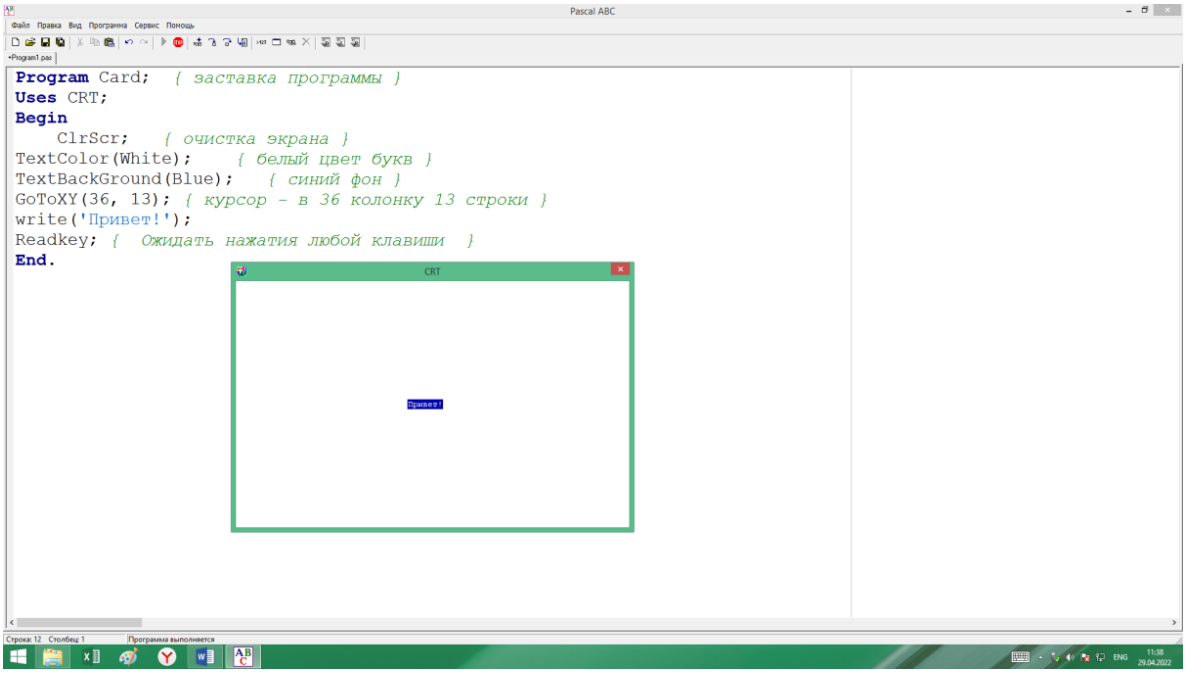
Цвет	**Код**	**Константа**
Черный	0	Black
Синий	1	Blue
Зеленый	2	Green
Бирюзовый	3	Cyan
Красный	4	Red
Сиреневый	5	Magenta
Коричневый	6	Brown
Белый (светло-серый)	7	lightGray
Серый	8	darkGray
Голубой	9	lightBlue

Светло-зеленый	10	lightGreen
Светло-бирюзовый	11	lightCyan
Светло-красный (алый)	12	lightRed
Светло-сиреневый	13	lightMagenta
Желтый	14	Yellow
Белый (яркий)	15	White

Модуль CRT эмулирует текстовый терминал первых персональных компьютеров. Графические объекты (прямые, окружности и т.д.) в текстовом окне не допустимы.

Подключение библиотеки к программе осуществляется предложением **USES**:

```
Program Card;  { заставка программы }
Uses CRT;
Begin
        ClrScr;   { очистка экрана }
    TextColor(White);    { белый цвет букв }
    TextBackGround(Blue);   { синий фон }
    GoToXY(36, 13); { курсор – в 36 колонку 13 строки }
    write(‘Привет!’);
    Readkey; {  Ожидать нажатия любой клавиши  }
End.
```



Рассмотрим еще несколько полезных команд.

WINDOW(X1, Y1, X2, Y2) – создание окна вывода. Если окно задано, все операторы write(ln) выводят только в него.

WhereX, WhereY – эти функции позволяют узнать координаты местонахождения курсора.

Пример. GoToXY(WhereX+10, WhereY+2);

ОЧИСТКА ЭКРАНА

Процедура **ClrScr (от англ. Clear Screen)** – удаляет все символы с экрана или текущего окна. При этом экран закрашивается текущим цветом фона, заданным процедурой TextBackGround. Если цвет фона в программе не задавался, то экран закрасится **черным** цветом. Кроме того, курсор устанавливается в точку экрана с координатами (1, 1), то есть в начало первой строки.

Пример. TextBackGround(Blue);

ClrScr; { экран очищается и закрашивается синим цветом }

ВВОД СИМВОЛА С КЛАВИАТУРЫ

Функция **Readkey** (читать клавишу) – возвращает символ (тип char), соответствующий нажатой клавише. Если ни одна клавиша не нажата, то функция ждет до тех пор, пока какая-либо клавиша будет нажата. Символ, соответствующий нажатой клавише, на экран не выводится.

Используя функцию Readkey, можно обрабатывать не только нажатия алфавитно-цифровых клавиш, но и служебных (функциональные клавиши <F1>-<F12>, клавиши перемещения курсора и др.).

1) При нажатии служебной клавиши Readkey возвращает 0.
2) Чтобы получить номер нажатой служебной клавиши, нужно еще раз вызвать Readkey.

Пример. Программа, позволяющая узнать код нажатой клавиши. Программа завершает работу при нажатии клавиши <Esc>.

```
Program Kod;
    Uses crt;
    Var ch: char;
    Begin
        Repeat
            Ch:=ReadKey;
            If ch=chr(0) then  { нажата служебная клавиша }
                Begin
                    Writeln('Служебная клавиша');
                    Ch:= ReadKey;
                End;
            Writeln(ord(ch));
        Until ord(ch)=27; { пока не нажата клавиша Esc}
    End.
```

Задача. Нарисовать на экране рамку. При вызове процедуры задаются координаты левого верхнего угла рамки, ширина и высота.

```
Procedure Frame(l: integer; t: integer; w: integer; h: integer);
    Var i, x, y: integer;
c1, c2, c3, c4, c5, c6: char; {символы, из которых рисуется рамка}
    Begin
        C1:= chr(218); { символ левого верхнего угла }
        C2:= chr(196); { символ горизонтальной линейки }
        C3:= chr(191); { символ правого верхнего угла }
        C4:= chr(179); { символ вертикальной линейки }
        C5:= chr(192); { символ левого нижнего угла }
    C6:= chr(217); { символ правого нижнего угла }
    GotoXY(l, t);
    Write (c1);
```

```
For i:=1 to w-2 do { верхняя граница }
        Write(c2);
Write(c3);
Y:=t+1;
X:= L+w-1;
For i:=1 to h-2 do { левая и правая границы }
        Begin
                GotoXY(l, t);
                Write(c4);
                GotoXY(x, y);
                Write(c4);
                Y:=y+1;
        End;
GotoXY(l, y);
Write(c5);
For i:=1 to w-2 do { нижняя граница }
        Write(c2);
Write(c6);
End; { конец процедуры }
```

Пример. Фрагмент меню. **Названия пунктов** выводятся на синем фоне светло-серым цветом, а **номера пунктов** – ярко-белым.

```
TextBackGround(Blue);
TextColor(15);
Write('1. ');
TextColor(7);
Writeln('Вывод на экран.');
TextColor(15);
```

```
Write(‘2. ’);
TextColor(7);
Writeln(‘Вывод на принтер.’);
```

Стандартные процедуры read и readLn. Стандартные функции readKey и KeyPressed; их применение в циклах

Этот пункт мы посвятим вопросам программированию обменов с клавиатурой компьютера. Паскаль содержит несколько простых и ясных средств, которые позволяют организовать эффективное управление программы посредством клавиатуры.

Самая простая и часто применяемая техника организации приема информации основывается на использовании уже знакомых Вам процедур read и readln. Расширим знания о них. Эти процедуры работают со стандартным входным файлом, который отождествлен с “консолью”, т. е. с клавиатурой и экраном дисплея. На практике это означает, что информация, введенная с клавиатуры, помимо обработки процедурами, будет отображаться на экране.

Удобством указанных процедур является автоматическое преобразование ими вводимой цепочки символов в значение заданного типа. Так, если в разделе описания переменных имеется описание вида

```
Var
Chislo : integer;
```

то выполнение оператора readln (Chislo) будет происходить следующим образом. Программа будет приостановлена в ожидании ввода с клавиатуры символов, изображающих целое число. После ввода этих изображений они будут автоматически преобразованы в соответствующие двоичные значения и присвоены переменной Chislo. Аналогично организован прием значений действительного, символьного и строкового типа. Если read(readln) не может выполнить преобразования, то генерируется ошибка № 106 – Invalid numeric format (Неверный формат числовых данных) и выполнение программы прекращается. Это является стандартной реакцией, которую выполняет программа, взявшая на себя обработку ошибок. Приведем пример применения этих процедур ввода при организации циклов.

```
Program Useread;
Var
  X, Y : Byte;
  Stop : String;
Begin
  TextBackGround;
  Randomize;
  repeat
    X:=Random (76);
    Y:=Random (23);
    GoToXY (X, Y);
    TextColor (Random(15));
    write('***');
    X:=1;
    Y:=24;
    write ('Для остановки программы наберите "Стоп"');
    write ('Для продолжения – любую клавишу ');
    readln(Stop);
  until (Stop='Стоп') or (Stop ='стоп');
  readln;
End.
```

Примечание. Здесь использованы следующие процедуры:

GoToXY (X, Y:Byte) - перемещает курсор к элементу экрана с заданными координатами, учитывая, что размер экрана в текстовом режиме 25 строк по 80 символов.

TextBackGround (Color : Byte) – задает цвет фона.

TextColor (Color : Byte) – задает цвет символов.

Однако, несмотря на простоту и удобство, стандартные процедуры read и readln не обеспечивают всеж потребностей, возникающих

при работе с клавиатурой. Их важнейший недостаток в том, что вместе с приемом символов они выполняют их отображение на экран (так называемое “эхо на монитор”). В большинстве случаев это либо не нужно, либо недопустимо.

Например, если программа реализует некоторый оконный интерфейс, то вывод вводимых символов испортит изображение. Кроме того, они рассчитаны только на ввод относительно небольшого подмножества символов (буквы, цифры, знаки препинания) и частичного использования специальных клавиш (например, Backspace для отмены только что введенного символа).

Эти процедуры не могут распознать нажатие функциональных или редактирующих клавиш и их сочетаний с управляющими клавишами Ctrl, Alt, Shift. В силу указанных причин процедуры read и readln редко используются в серьезных программах.

Стандартная функция readKey

Более универсальным средством взаимодействия с клавиатурой является стандартная функция readKey из системного модуля Crt. Функция вызывается без параметров, возвращает значение символьного типа и работает следующим образом. Организуется задержка выполнения с ожиданием нажатия клавиши. После того, как нажатие произведено, функция завершает работу, возвращая код нажатой клавиши. Полученное значение можно использовать далее в программе. Тривиальный пример работы с функцией readKey, не требующий комментариев, может выглядеть так:

```
Program UsereadKey;
Uses Crt;
Var
  Sym : Char;
Begin
  ClrScr;
  while true do
    begin
      write (‘Введите букву - ‘);
```

```
    Sym := readKey;
    writeln (‘Вы ввели букву - ‘, Sym);
    if Sym = ‘q’
      then
        Exit
   end
End.
```

Примечание. Здесь использована процедура Exit, которая позволяет досрочно выйти из программы. Применение этой процедуры является плохим стилем программирования.

Функция readKey не отображает введенный символ на экран, благодаря чему она широко используется для организации управления в различных диалоговых программах. В дополнение к этому readKey позволяет отслеживать нажатие более широкого множества клавиш, опознавая функциональные и редактирующие клавиши и их сочетания с управляющими клавишами Ctrl, Alt, Shift.

Если говорить более подробно, функция readKey исходит из того, что все множество клавиш и их сочетаний с управляющими клавишами разбито на два подмножества, которые обычно называют основным и расширенным наборами.

В основной набор входят клавиши букв, цифр, разделителей и знаков препинания, их комбинации с клавишей Shift (или, что то же самое, при включенном переключателе CapsLock), а также клавиши Tab, BackSpace, Enter и Esc. Если нажата одна из перечисленных клавиш, то readKey возвратит обычный ASCII-код соответствующего символа.

В расширенном наборе содержатся некоторые (не все) клавиши из основного набора в комбинации с клавишами Ctrl и Alt, а также функциональные и редактирующие клавиши. Если нажимается одна из клавиш расширенного набора, то функция readKey возвращает символ с кодом 0 (его представление в программе – chr(0) или #0). В этом случае повторное обращениек readKey вернет код клавиши из расширенного набора.

Коды клавиш из основного и расширенного наборов в виде, удобном для включения в программы, приведены в приложении.

Схема использования функции readKey для общего случая может выглядеть так:

```
Program UsereadKey2;
Uses Crt;
Var
  Sym : Char;
Begin
  ClrScr;
  while true do
    begin
      write (‘Нажмите клавишу‘);
      Sym := readKey;
      if Sym <> #0
        then
          begin {основной набор}
            case Sym of
              #8 : writeln (‘Вы нажали BackSpace‘);
              #9 : writeln (‘Вы нажали Tab‘);
              #13 : writeln (‘Вы нажали Enter‘);
              #27 : writeln (‘Вы нажали Esc‘);
            else
              writeln (‘Вы ввели символ ‘,Sym);
          end;
        if Sym = #27
          Then
            Exit
          end
```

```
        else
          begin {расширенный набор}
            Sym := readKey;{повт. чтение: берем расширенный код}
            writeln ('Вы нажали клавишу с кодом ', Ord (Sym)));
          end
        end
    End.
```

Большинство прикладных диалоговых программ использует описанную технику взаимодействия с клавиатурой. Однако встречаются случаи, когда возможностей функции readKey оказывается недостаточно. На самом деле функция readKey воспринимает нажатия не всех клавиш: достаточно попробовать, запустив вышеприведенную программу, нажать клавиши F11, F12, ввести комбинацию Alt+Esc и т.д.

Стандартная функция KeyPressed

Второй базовой функцией взаимодействия с клавиатурой является функция KeyPressed. В отличие от readKey, она предназначена не для приема кода нажатой клавиши, а для простой проверки, была ли нажата какая-либо клавиша. Эта функция вызывается без параметров и возвращает значение булевого типа: True, если было нажатие, и False в противном случае.

Важно понять, что KeyPressed не производит никаких действий с кодом нажатой клавиши, но код может быть далее прочитан функцией readKey, например,

```
    . . .
if KeyPressed
  then
    S := readKey;
. . .
```

Соотношение этих функций станет более понятным, если рассмотреть их внутреннюю организацию несколько подробнее. В системной области DOS имеется небольшой буфер, в который операционная система помещает коды нажатых клавиш. буфер организован в виде очереди, причем помещение кодов производится в ее хвост, а

считывание из головы. Таким образом, каждое обращение к функции readKey извлекает из головы очереди один содержащийся там код. Если буфер пуст, то организуется задержка выполнения до тех пор, пока в нем не появится код (появление кода соответствует нажатию клавиши). Если же к моменту вызова readKey нажатие уже произошло, то есть буфер содержит хотя бы один код, то никакой задержки не будет. Буфер очень невелик и рассчитан на хранение максимум 15 кодов, что соответствует 15 нажатиям. Кстати говоря, иногда встречается такая ситуация, когда та или иная программа “не успевает” выбирать коды клавиш из буфера (то есть нажатия производятся чаще). Ситуация переполнения буфера индицируется звуковым сигналом, после чего коды вновь нажимаемых клавиш будут пропадать.

Коды специальных клавиш

Клавиша	Что возвращает ReadKey?
Escape	#27
Enter	#13
Ctrl+**Enter**	#10
Back**sp**ace	#8
Ctrl+**B**ack**sp**ace	#127
Tab	#9
Shift+**Tab**	#15
F1	#0, #59
F2	#0, #60
F3	#0, #61

F4	#0, #62
F5	#0, #63
F6	#0, #64
F7	#0, #65
F8	#0, #66
F9	#0, #67
F10	#0, #68
←	#0, #75
↑	#0, #72
→	#0, #77
↓	#0, #80
Delete	#0, #83
Insert	#0, #82
Home	#0, #71
End	#0, #79
Pa**g**e **Up**	#0, #73
Pa**g**e **D**o**w**n	#0, #81
NumPad 5[1]	#0, #76

Shift+F1	#0, #84
Shift+F2	#0, #85
Shift+F3	#0, #86
Shift+F4	#0, #87
Shift+F5	#0, #88
Shift+F6	#0, #89
Shift+F7	#0, #90
Shift+F8	#0, #91
Shift+F9	#0, #92
Shift+F10	#0, #93
Ctrl+F1	#0, #94
Ctrl+F2	#0, #95
Ctrl+F3	#0, #96
Ctrl+F4	#0, #97
Ctrl+F5	#0, #98
Ctrl+F6	#0, #99
Ctrl+F7	#0, #100
Ctrl+F8	#0, #101

Ctrl+**F9**	#0, #102
Ctrl+**F10**	#0, #103
Alt+**F1**	#0, #104
Alt+**F2**	#0, #105
Alt+**F3**	#0, #106
Alt+**F4**	#0, #107
Alt+**F5**	#0, #108
Alt+**F6**	#0, #109
Alt+**F7**	#0, #110
Alt+**F8**	#0, #111
Alt+**F9**	#0, #112
Alt+**F10**	#0, #113
Ctrl+←	#0, #115
Ctrl+→	#0, #116
Ctrl+**Home**	#0, #119
Ctrl+**End**	#0, #117
Ctrl+**Page Up**	#0, #132
Ctrl+**Page Down**	#0, #118

Ctrl+**2**	#0, #3
Ctrl+**A**	#1
Ctrl+**B**	#2
Ctrl+**C**[2]	#3
Ctrl+**D**	#4
Ctrl+**E**	#5
Ctrl+**F**	#6
Ctrl+**G**	#7
Ctrl+**H**[3]	#8
Ctrl+**I**	#9
Ctrl+**J**	#10
Ctrl+**K**	#11
Ctrl+**L**	#12
Ctrl+**M**	#13
Ctrl+**N**	#14
Ctrl+**O**	#15
Ctrl+**P**	#16
Ctrl+**Q**	#17

Ctrl+**R**	#18
Ctrl+**S**	#19
Ctrl+**T**	#20
Ctrl+**U**	#21
Ctrl+**V**	#22
Ctrl+**W**	#23
Ctrl+**X**	#24
Ctrl+**Y**	#25
Ctrl+**Z**	#26
Ctrl+**[**	#27
Ctrl+\	#28
Ctrl+**]**	#29
Ctrl+**6**	#30
Ctrl+-	#31
Alt+**Q**	#0, #16
Alt+**W**	#0, #17
Alt+**E**	#0, #18
Alt+**R**	#0, #19

Alt+**T**	#0, #20
Alt+**Y**	#0, #21
Alt+**U**	#0, #22
Alt+**I**	#0, #23
Alt+**O**	#0, #24
Alt+**P**	#0, #25
Alt+**A**	#0, #30
Alt+**S**	#0, #31
Alt+**D**	#0, #32
Alt+**F**	#0, #33
Alt+**G**	#0, #34
Alt+**H**	#0, #35
Alt+**J**	#0, #36
Alt+**K**	#0, #37
Alt+**L**	#0, #38
Alt+**Z**	#0, #44
Alt+**X**	#0, #45
Alt+**C**	#0, #46

Alt+**V**	#0, #47
Alt+**B**	#0, #48
Alt+**N**	#0, #49
Alt+**M**	#0, #50
Alt+**1**	#0, #120
Alt+**2**	#0, #121
Alt+**3**	#0, #122
Alt+**4**	#0, #123
Alt+**5**	#0, #124
Alt+**6**	#0, #125
Alt+**7**	#0, #126
Alt+**8**	#0, #127
Alt+**9**	#0, #128
Alt+**0**	#0, #129
Alt+-	#0, #130
Alt+=	#0, #131

Задачи для решения

1. Написать программу «Визитная карточка», которая выводит на экран хорошо оформленные ваши личные данные: ФИО, дату

рождения, адрес, телефон (можно вымышленные). Использовать процедуру, рисующую рамку, разные цвета, хорошо скомпонуйте информацию.

2. Составьте программу движения круга вверх, вниз, влево, вправо в зависимости от нажатия клавиш управления курсором.

3. Составьте программу движения заданного слова сверху вниз и обратно, для остановки движения запрограммируйте нажатие какой-либо клавиши.

4. Составьте программу движения человечка, для остановки движения запрограммируйте нажатие какой-либо клавиши..

5. Составьте программу движения маятника, для остановки движения запрограммируйте нажатие какой-либо клавиши.

6. Составьте программу движения бегущей строки.

7. Составьте программу вывода слова на экран и поочередное мерцание его букв.

8. Составьте программу падания букв из введенного слова.

9. Составьте программу случайного вывода звездочек разного цвета на экран, для вывода запрограммируйте нажатие какой-либо клавиши.

10. Составьте программу рисования надувающихся пузырей и их лопания по достижению заданного радиуса.

11. Составить программу для вывода на экран бегущей надписи, например: "Для остановки нажми Ctrl+Stop".

ГЛАВА 20. ГРАФИКА В ПАСКАЛЬ АВС

Основные графические процедуры

Для работы в графическом режиме необходимо подключение модуля **GraphABC.** Первой инструкцией программы должна быть инструкция:

uses GraphABC;

По умолчанию размеры графического экрана 640 на 400 точек.

Все графические объекты имеют определенный цвет. Каждому цвету соответствует название:

clBlack – черный

clPurple – фиолетовый

clWhite – белый

clMaroon – темно-красный

clRed – красный

clNavy – темно-синий

clGreen – зеленый

clBrown – коричневый

clBlue – синий

clSkyBlue – голубой

clYellow – желтый

clCream – кремовый

clAqua – бирюзовый

clOlive – оливковый

clFuchsia – сиреневый

clTeal – сине-зеленый

clGray – темно-серый

clLime – ярко-зеленый

clMoneyGreen – цвет зеленых денег

clLtGray – светло-серый

clDkGray – темно-серый

clMedGray – серый

clSilver – серебряный

Также можно задать цвет номером, например random(16777215) – случайный цвет из всей палитры цветов Паскаля

Ниже указаны некоторые процедуры модуля GraphABC, применяющиеся для построения примитивов.

Строка	*результат*
setpixel(x,y,c)	построить точку (x,y) цветом c
setpencolor(c)	устанавливает текущий цвет рисунка
lineto(x,y)	рисует отрезок от текущего положения пера до точки (x,y)
line(x1,y1,x2,y2)	соединить две точки отрезком
rectangle(x1,y1,x2,y2)	построить прямоугольник с заданными концами диагонали и сторонами, параллельными осям координат
circle(x,y,r)	построить окружность с центром (x,y) и радиусом R
arc(x,y,a,b,r)	построить дугу окружности: a,b – начальный и конечный углы в градусах
ellipse(x1,y1,x2,y2)	нарисовать эллипс, заданный описанным прямоугольником с вершинами (x1,y1) и (x2,y2)
floodfill(x,y,c)	заливает область цветом c, начиная с точки (x,y)

Рассмотрим перечисленные и другие процедуры более подробно.

Управление экраном

SetWindowWidth(w) - устанавливает ширину графического окна;

SetWindowHeight(h) - устанавливает высоту графического окна;

Очистка графического окна

ClearWindow; - очищает графическое окно белым цветом.

ClearWindow(color); - очищает графическое окно указанным цветом.

Графические примитивы

1. Точка
2. Линия
3. Прямоугольник
4. Окружность
5. Эллипс
6. Сектор
7. Дуга

Точка

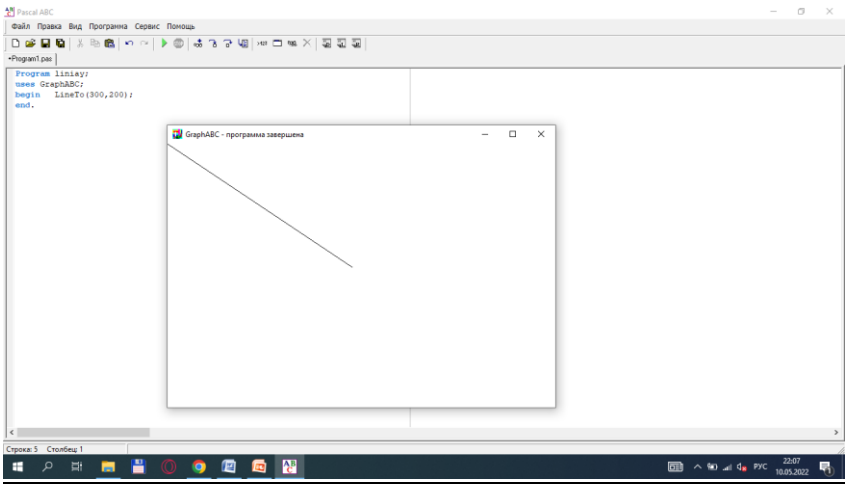
SetPixel(x,y,color) - закрашивает один пиксел с координатами (x,y) цветом color

```
program tochka;
uses GraphABC;
begin
  SetPixel(300,200,clred);
end.
```

Линии

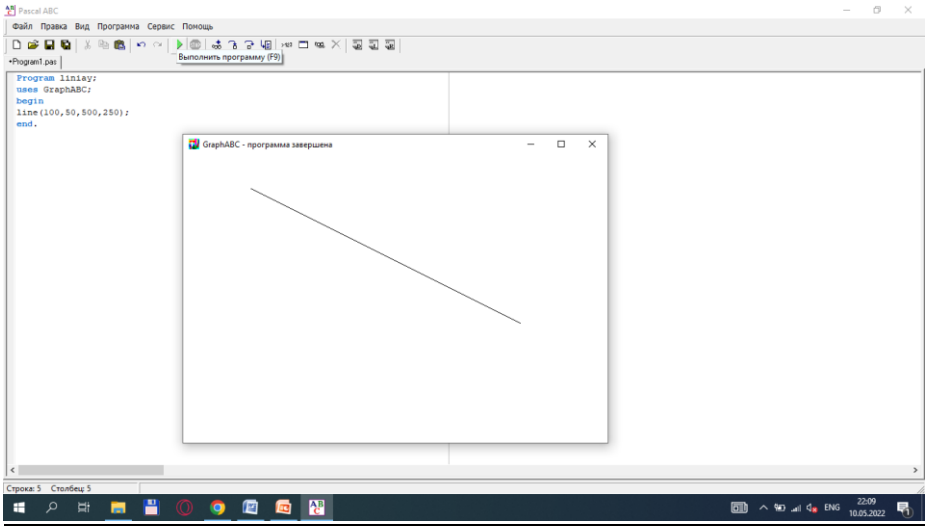
LineTo(x,y) - рисует отрезок от текущего положения пера до точки (x,y); координаты пера при этом также становятся равными (x,y).

```
Program liniay1;
uses GraphABC;
begin   LineTo(300,200);
end.
```



Line(x1,y1,x2,y2) - рисует отрезок с началом (x1,y1) и концом (x2,y2). MoveTo(x,y) - устанавливает текущую позицию рисования в точку (x,y).

```
Program liniay2;
uses GraphABC;
begin
line(110,65,450,290);
end.
```



Цвет линии

SetPenColor(color) - устанавливает цвет пера, задаваемый параметром color.

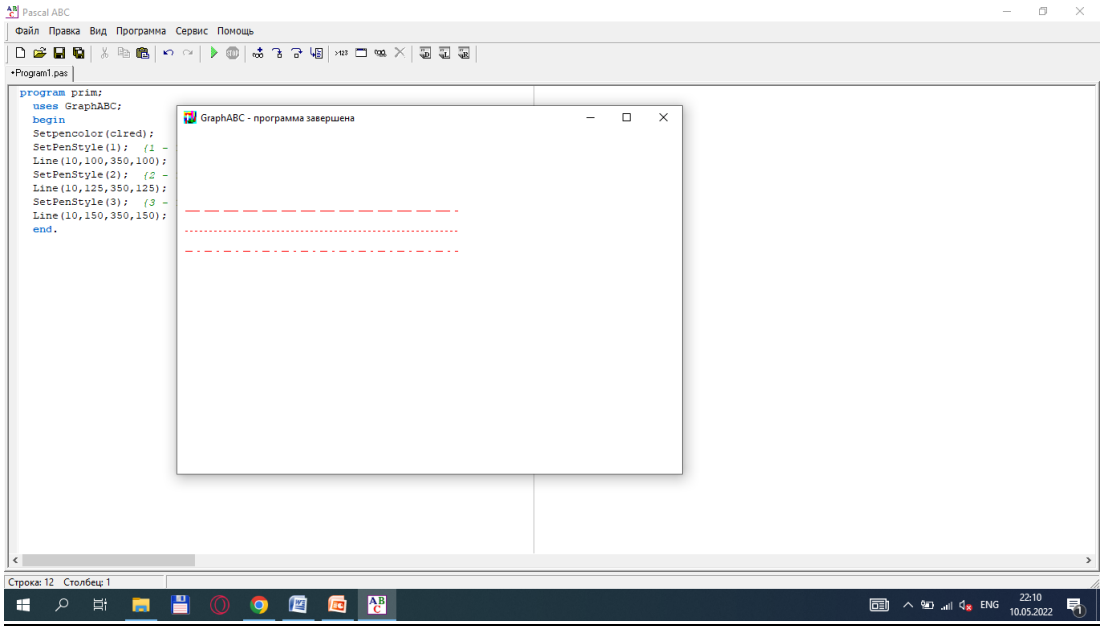
```
Program liniay3;
uses GraphABC;
```

```
begin
  setpencolor(clred);
  line(30,30,400,350);
end.
```

Пунктирная линия

SetPenStyle(<номер от 1 до 6>); - устанавливает стиль пера, задаваемый номером.

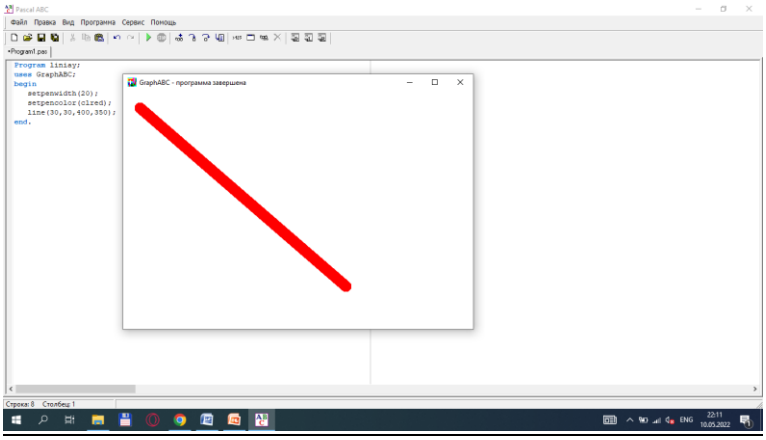
```
program punktir;
  uses GraphABC;
  begin
  Setpencolor(clred);
  SetPenStyle(1);  {1 - длинный штрих}
  Line(10,100,350,100);
  SetPenStyle(2);  {2 - короткий штрих}
  Line(10,125,350,125);
  SetPenStyle(3);  {3 - штрих-пунктир}
  Line(10,150,350,150);
  end.
```



Толщина линии

SetPenWidth(n) - устанавливает ширину (толщину) пера, равную n пикселям.

```
Program liniay;
uses GraphABC;
begin
  setpenwidth(20);
  setpencolor(clred);
  line(30,30,400,350);
end.
```



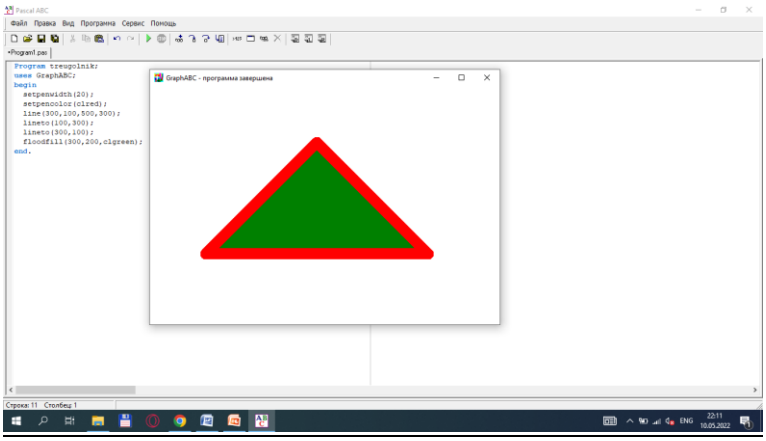
Треугольник

Треугольник рисуется процедурами **Line(x1,y1,x2,y2); LineTo(x,y);**

```
Program treugolnik;
uses GraphABC;
begin
  setpenwidth(20);
  setpencolor(clred);
  line(300,100,500,300);
  lineto(100,300);
  lineto(300,100);
```

floodfill(300,200,clgreen);

end.



Прямоугольник

Rectangle(x1,y1,x2,y2) - рисует прямоугольник, заданный координатами противоположных вершин (x1,y1) и (x2,y2).

Program pryamougolnik1;

uses GraphABC;

Begin

Rectangle(50,50,200,200);

end.

Заливка цветом

FloodFill(x,y,color) - заливает область одного цвета цветом color, начиная с точки (x,y).

Program pryamougolnik2;

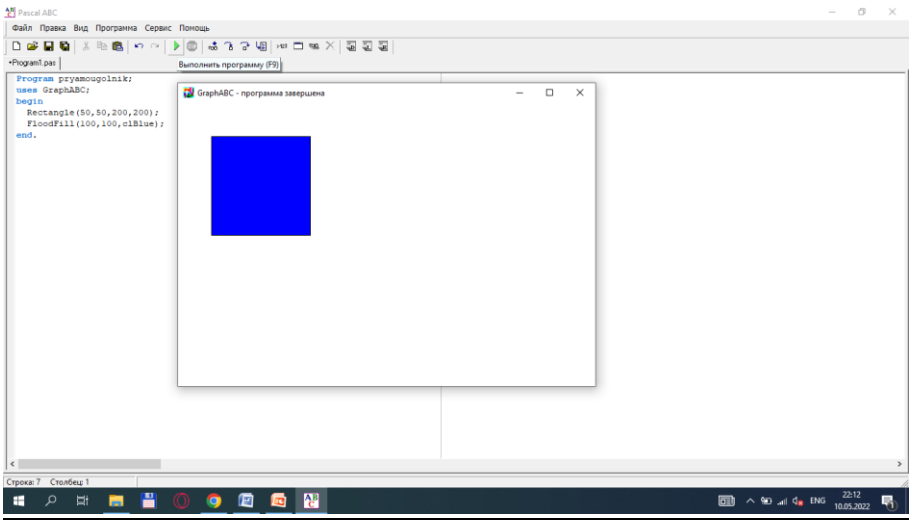
uses GraphABC;

begin

Rectangle(50,50,200,200);

FloodFill(100,100,clBlue);

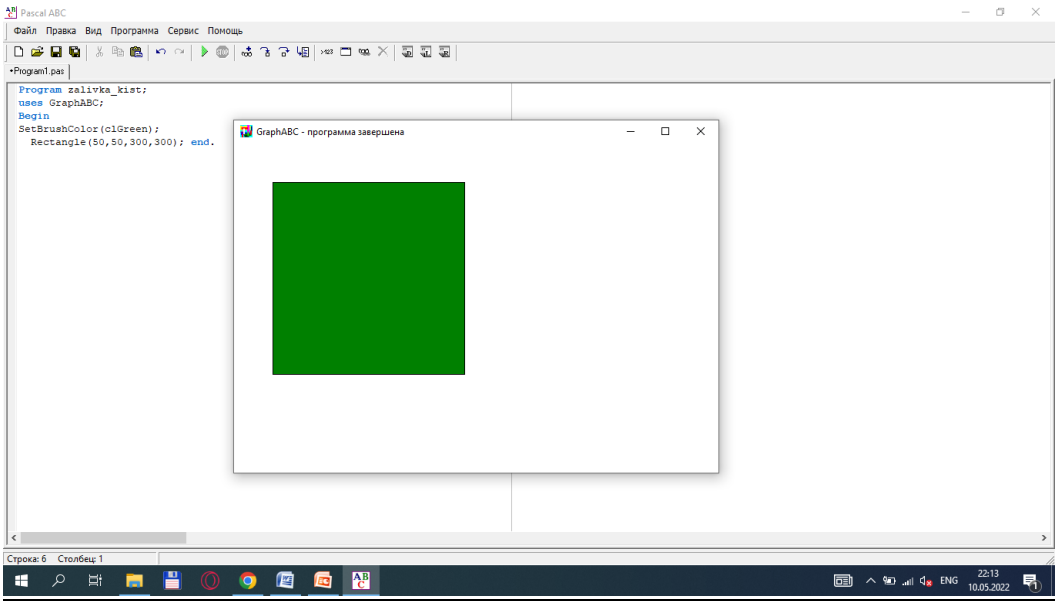
end.



Заливка кистью

SetBrushColor(color) - устанавливает цвет кисти. Заливка кистью распространяется на замкнутый контур, описание которого следует за процедурой установки цвета кисти.

```
Program zalivka_kist;
uses GraphABC;
Begin
SetBrushColor(clGreen);
  Rectangle(50,50,300,300); end.
```

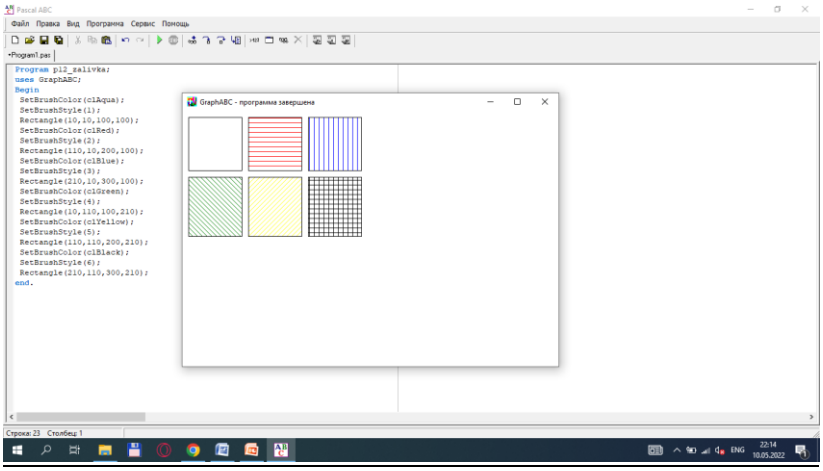


Установка стиля кисти

SetBrushStyle(номер от 0 до 7 или название) - устанавливает стиль кисти, задаваемый номером или символической константой.

По умолчанию задается стиль 0 – сплошная заливка цветом.

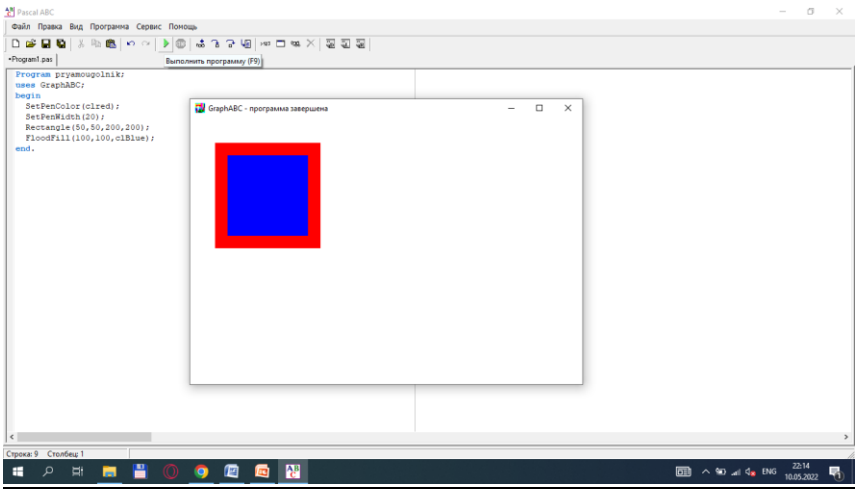
```
Program p12_zalivka;
uses GraphABC;
Begin
 SetBrushColor(clAqua);
 SetBrushStyle(1);
 Rectangle(10,10,100,100);
 SetBrushColor(clRed);
 SetBrushStyle(2);
 Rectangle(110,10,200,100);
 SetBrushColor(clBlue);
 SetBrushStyle(3);
 Rectangle(210,10,300,100);
 SetBrushColor(clGreen);
 SetBrushStyle(4);
 Rectangle(10,110,100,210);
 SetBrushColor(clYellow);
 SetBrushStyle(5);
 Rectangle(110,110,200,210);
 SetBrushColor(clBlack);
 SetBrushStyle(6);
 Rectangle(210,110,300,210);
end.
```



Цвет и толщина контура

Цвет и толщина контура задаются процедурами **SetPenWidth(w); SetPenColor(color);**

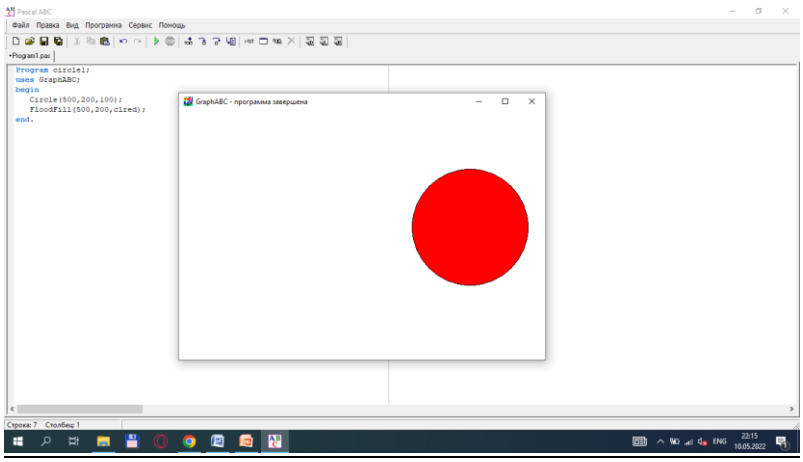
```
Program pryamougolnik;
uses GraphABC;
begin
  SetPenColor(clred);
  SetPenWidth(20);
  Rectangle(50,50,200,200);
  FloodFill(100,100,clBlue);
end.
```



Окружность

Circle(x,y,r) - рисует окружность с центром в точке (x,y) и радиусом r.

```
Program circle1;
uses GraphABC;
begin
  Circle(500,200,100);
  FloodFill(500,200,clred);
end.
```



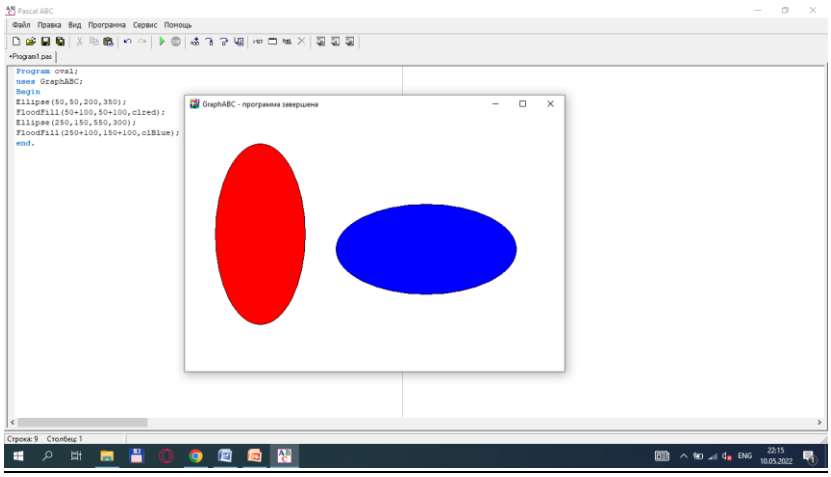
Эллипс

Ellipse(x1,y1,x2,y2) - рисует эллипс, заданный своим описанным прямоугольником с координатами противоположных вершин (x1,y1) и (x2,y2).

```
Program oval;
uses GraphABC;
Begin
Ellipse(50,50,200,350);
FloodFill(50+100,50+100,clred);
Ellipse(250,150,550,300);
```

FloodFill(250+100,150+100,clBlue);

end.



Дуга окружности

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

Program duga;

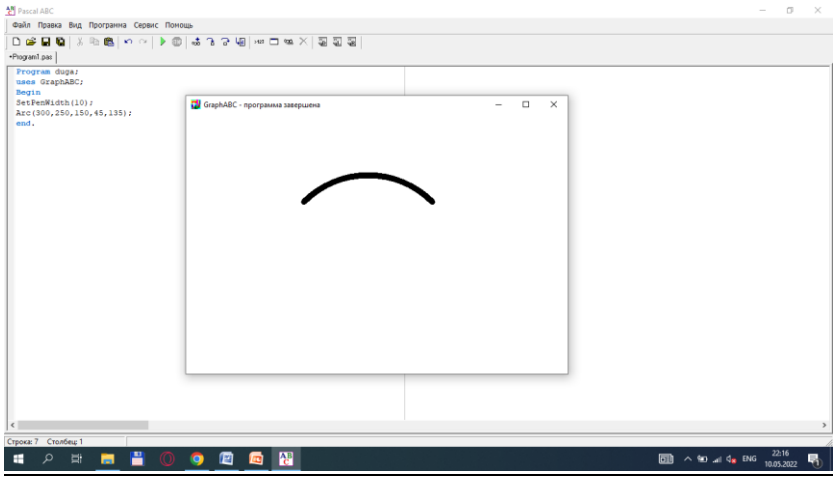
uses GraphABC;

Begin

SetPenWidth(10);

Arc(300,250,150,45,135);

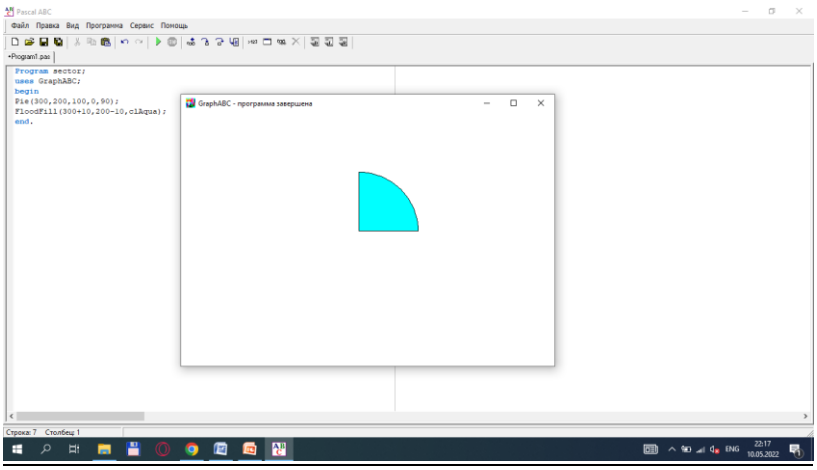
end.



Сектор

Pie(x,y,r,a1,a2) - рисует сектор окружности, ограниченный дугой (параметры процедуры имеют тот же смысл, что и в процедуре Arc).

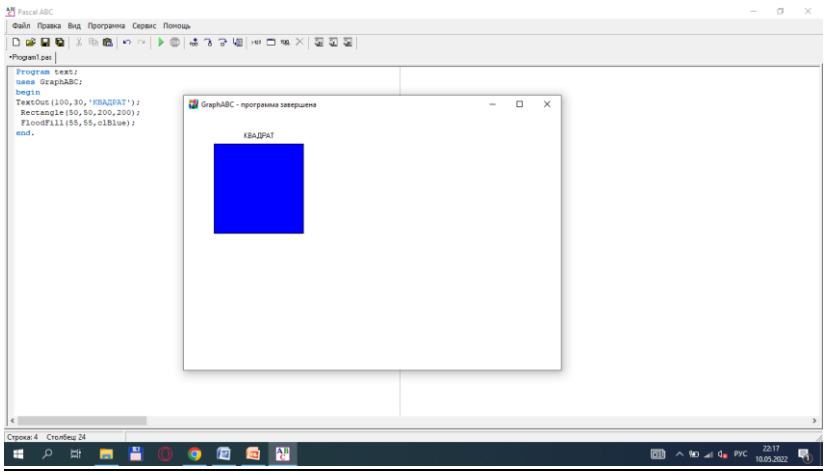
```
Program sector;
uses GraphABC;
begin
Pie(300,200,100,0,90);
FloodFill(300+10,200-10,clAqua);
end.
```



Вывод текста в графическое окно

TextOut(x,y,'строка'); - выводит строку текста в позицию (x,y) (точка (x,y) задает верхний левый угол прямоугольника, который будет содержать текст).

```
Program text;
uses GraphABC;
begin
TextOut(100,30,'Квадрат');
 Rectangle(50,50,200,200);
 FloodFill(55,55,clBlue);
end.
```



Действия со шрифтом

SetFontName(‘name’) – устанавливает наименование шрифта.

SetFontColor(color) - устанавливает цвет шрифта.

SetFontSize(sz) - устанавливает размер шрифта в пунктах.

SetFontStyle(fs) - устанавливает стиль шрифта.

По умолчанию установлен шрифт, имеющий наименование **MS Sans Serif.**

Наиболее распространенные шрифты – это **Times, Arial** и **Courier New.** Наименование шрифта можно набирать без учета регистра.

Пример:

SetFontName(‘Times’);

Стиль шрифта

Задается именованными константами:

fsNormal – обычный;

fsBold – жирный;

fsItalic – наклонный;

fsBoldItalic – жирный наклонный;

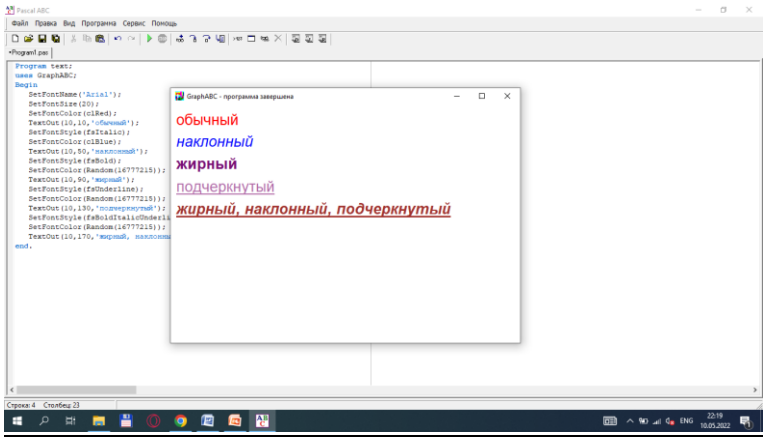
fsUnderline – подчеркнутый;

fsBoldUnderline – жирный подчеркнутый;

fsItalicUnderline – наклонный подчеркнутый;

fsBoldItalicUnderline – жирный наклонный подчеркнутый.

```
Program text;
uses GraphABC;
Begin
  SetFontName(‘Arial’);
  SetFontSize(20);
  SetFontColor(clRed);
  TextOut(10,10,‘обычный');
  SetFontStyle(fsItalic);
  SetFontColor(clBlue);
  TextOut(10,50,‘наклонный');
  SetFontStyle(fsBold);
  SetFontColor(Random(16777215));
  TextOut(10,90,‘жирный');
  SetFontStyle(fsUnderline);
  SetFontColor(Random(16777215));
  TextOut(10,130,‘подчеркнутый');
  SetFontStyle(fsBoldItalicUnderline);
  SetFontColor(Random(16777215));
  TextOut(10,170,’жирный, наклонный, подчеркнутый');
end.
```



Используемые цвета

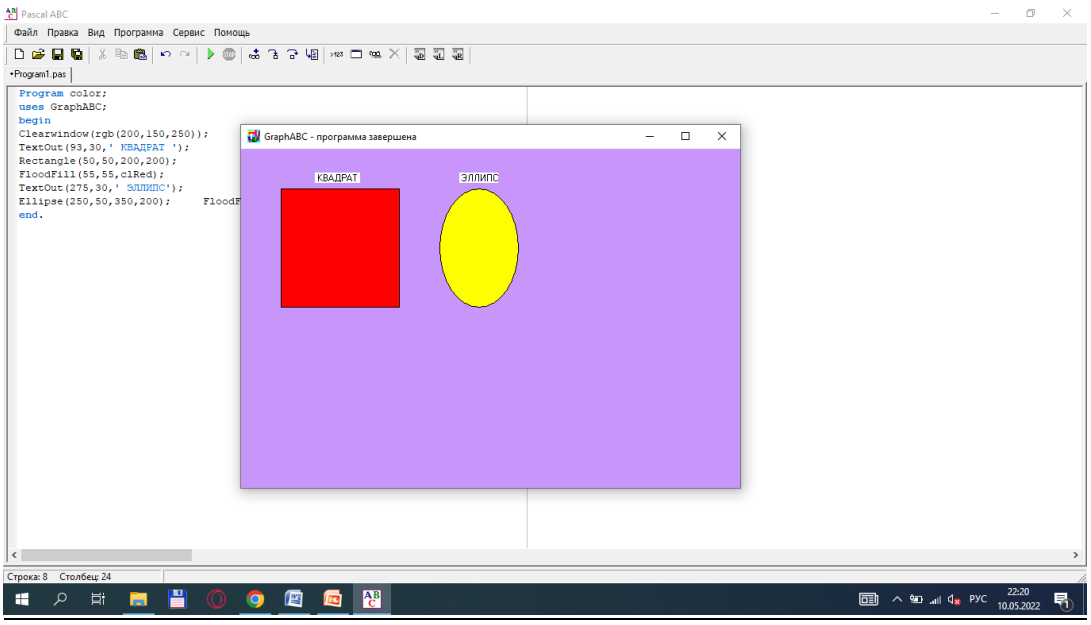
Цвет можно задавать и с помощью функции RGB(r,g,b) где r, g и b – целые числа в диапазоне от 0 до 255.

Функция возвращает целое значение, являющееся кодом цвета, который содержит красную, зеленую и синюю составляющие с интенсивностями r, g и b соответственно (0 соответствует минимальной интенсивности, 255 – максимальной).

RGB(255,255,255) – соответствует белому цвету.

RGB(0,0,0) – соответствует черному цвету.

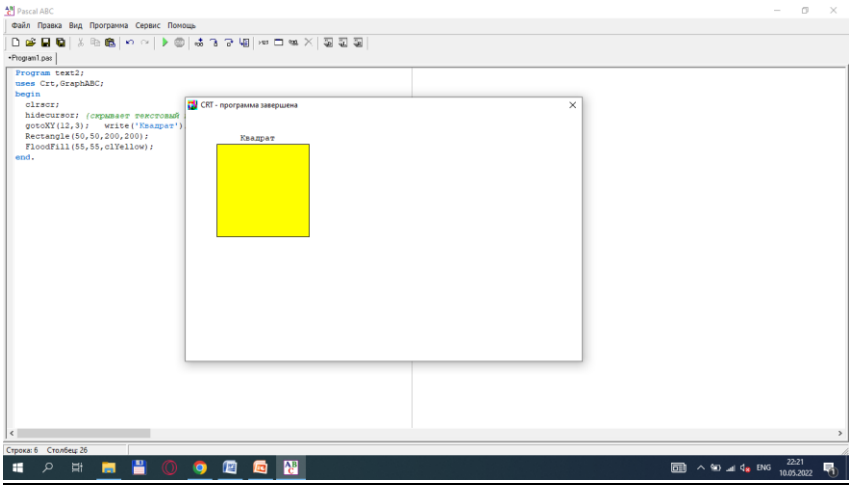
```
Program color;
uses GraphABC;
begin
Clearwindow(rgb(200,150,250));
TextOut(93,30,' Квадрат ');
Rectangle(50,50,200,200);
FloodFill(55,55,clRed);
TextOut(275,30,' Эллипс');
Ellipse(250,50,350,200);     FloodFill(250+50,50+50,clYellow);
end.
```



Вывод текста в графическое окно

Текст можно вывести с помощью операторов **Gotoxy(x,y) и Write(‘текст’),** подключив дополнительно модуль **Crt.**

```
Program text2;
uses Crt,GraphABC;
begin
  clrscr;
  hidecursor; {скрывает текстовый курсор}
  gotoXY(12,3); write(‘Квадрат');
  Rectangle(50,50,200,200);
  FloodFill(55,55,clYellow);
end.
```



Загрузка готового рисунка

n:=LoadPicture(fname) – загружает рисунок из файла с именем fname в оперативную память и возвращает описатель рисунка в целую переменную n; если файл не найден, то возникает ошибка времени выполнения.

Загружать можно рисунки в формате .bmp, .jpg или .gif.

Вывод рисунка в графическое окно

DrawPicture(n,x,y); – выводит рисунок с описателем n в позицию (x,y) графического окна.

```
uses GraphABC;
var pic: integer;
Begin
 pic:=LoadPicture('demo.bmp');
 DrawPicture(pic,10,10);
 DestroyPicture(pic);
end.
```

Сохранение созданного рисунка

SavePicture(n, ‘fname’) - сохраняет рисунок с описателем n в файл с именем fname. Рисунки можно сохранять в формате .bmp, .jpg или .gif.

Для сохранения рисунка в файл можно также использовать команду **SaveWindow**. У нее один параметр - имя файла.

```
pic:=LoadPicture('111.jpg');
DrawPicture(pic,55,110); // вывод рисунка на экран из файла
DestroyPicture(pic);
```

SaveWindow('demo.bmp');// сохранение созданного рисунка в файл.

Заливка кистью

SetBrushPicture(‘fname’) - устанавливает в качестве образца для закраски кистью образец, хранящийся в файле fname, при этом текущий цвет кисти при закраске игнорируется.

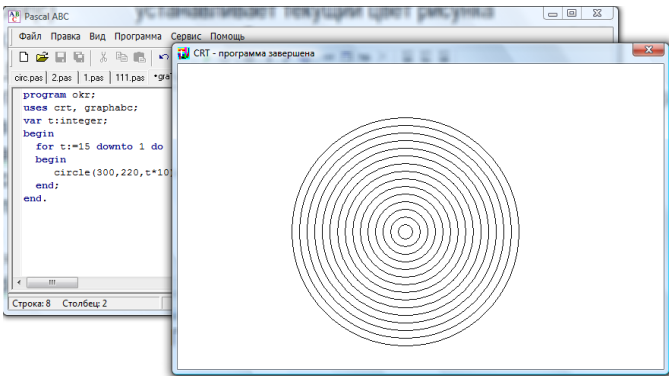
```
uses GraphABC;
begin
    SetBrushPicture('brush4.bmp');
    Ellipse(0,0,640,400);
end.
```

Пример программы

Начертить на экране 15 окружностей уменьшающегося радиуса, чтобы радиус был в 10 раз больше номера окружности.

```
program okr;
uses crt, graphabc;
var t:integer;
begin
  for t:=15 downto 1 do
    circle(300,220,t*10);
end.
```

Пример графической программы:



Простейшие задачи на графику:

1. В центре окна (640x400) нарисовать окружность радиусом 40 и закрасить ее красным цветом.

2. В центре окна (640x400) нарисовать прямоугольник 80x50 и закрасить его зеленым цветом.

3. По углам окна (640x400) нарисовать по одной окружности радиусом 15 и закрасить их разными цветами.

4. В центре окна (640x400) нарисовать квадрат 80x80, в его центре нарисовать окружность радиусом 40 и закрасить их разными цветами.

5. Нарисовать несложный домик и около него простую фигурку человечка.

6. Нарисовать в центре экрана простой самолетик на взлетной полосе и вдоль нее несколько деревьев.

Задания для самостоятельного решения

1. Расставить на экране 100 точек со случайными координатами X и Y ("звездное небо"). Координаты выбирать с помощью random.

2. Нарисовать в центре экрана Олимпийские кольца.

3. Расставить на экране 30 окружностей со случайными координатами X и Y, случайного радиуса и цвета ("мыльные пузыри").

4. Через точку в центре экрана провести 20 отрезков, вторая координата которых выбирается случайно ("разбитое стекло").

5. Написать графическую иллюстрацию к задаче: запросить с клавиатуры координаты точки (X, Y) и горизонтального отрезка прямой (Xn, Xk, Yn) и определить, лежит ли точка на прямой. Сообщение вывести на экран.

6. Начертить на экране 5 окружностей уменьшающегося радиуса, начиная с 100 точек с шагом 5, со смещением центра по горизонтали вправо на 20 точек.

7. Начертить 100 закрашенных эллипсов со случайными координатами, размером и цветом.

8. Начертить прямоугольник с координатами углов x1=130, y1=80, x2=220, y2=200 и в нем расставить 100 случайных точек ("звезды в окне").

9. Нарисовать на экране ряд окружностей уменьшающегося радиуса от 100 до 10, по горизонтали справа налево "воронка". По оси X использовать весь экран.

10. В центре экрана из отдельных элементов (линия, эллипс, окружность, дуга, прямоугольник) в графике нарисуйте свои инициалы.

11. Начертить в центре экрана 5 квадратов со стороной, увеличивающейся от 50 с шагом 20.

Анимация в языке Паскаль

Реализовать анимацию (движение объектов) в программировании можно следующим образом:

1) сначала рисуется фигура **цветным** инструментом;
2) затем с теми же координатами рисуется **та же** фигура **белым** цветом;
3) после чего происходит сдвиг фигуры и действия повторяются.

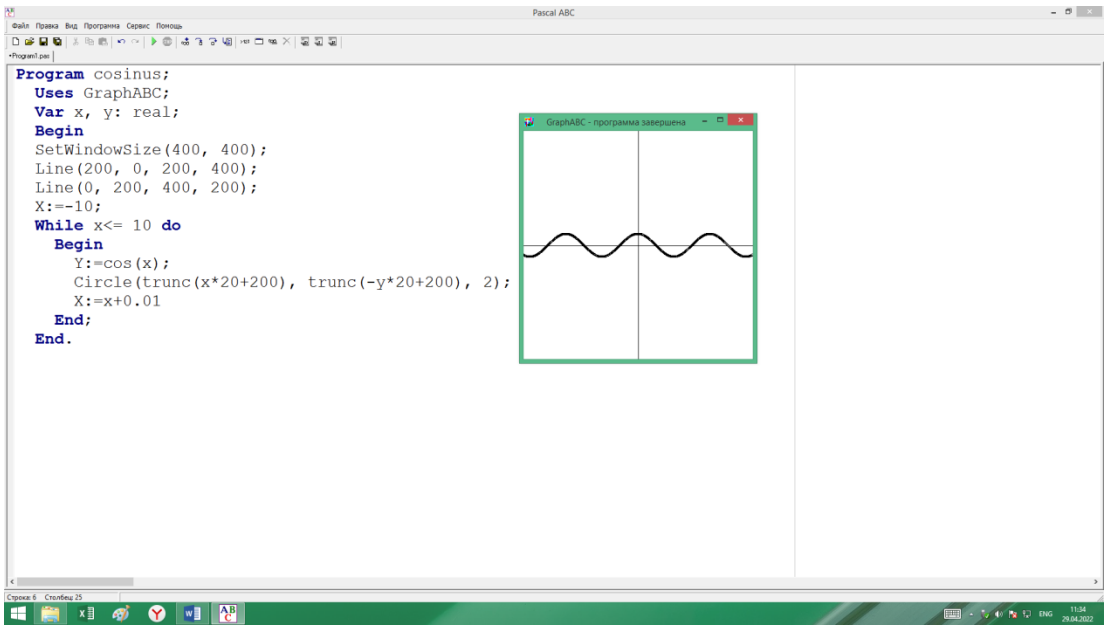
Пример. Движение круга по горизонтали.

```
Program dvizhenie_kruga;
    Uses GraphABC;
    Var x: integer;
    Begin
        X:=40;
        Repeat
            SetPenColor(clWhite);
            Circle(x, 100, 10); //  белая окружность
            SetPenColor(clBlack);
        X:=x+1;
        Circle(x, 100, 10); //  черная окружность
        Until x>600;
    End.
```

Задача. Построить график функции y=cos x.

```
Program cosinus;
    Uses GraphABC;
    Var x, y: real;
    Begin
    SetWindowSize(400, 400);
    Line(200, 0, 200, 400);
```

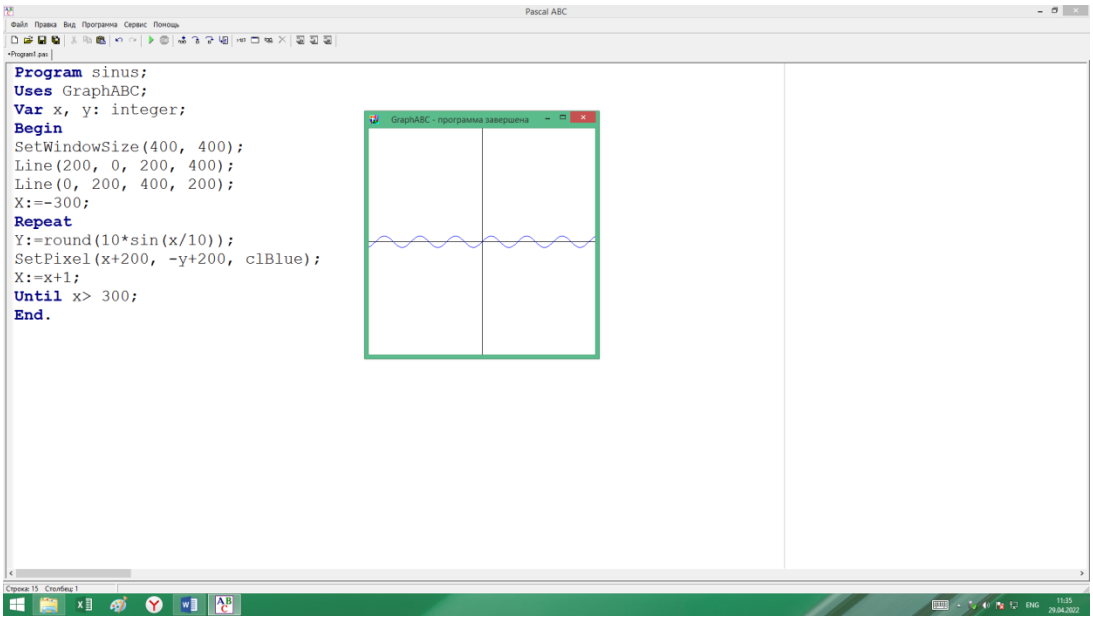
```
Line(0, 200, 400, 200);
X:=-10;
While x<= 10 do
        Begin
                Y:=cos(x);
                Circle(trunc(x*20+200), trunc(-y*20+200), 2);
                X:=x+0.01
        End;
End.
```



Задача. Построить график функции y=10sin(x/10).

```
Program sinus;
Uses GraphABC;
Var x, y: integer;
Begin
SetWindowSize(400, 400);
Line(200, 0, 200, 400);
Line(0, 200, 400, 200);
```

```
X:=-300;
Repeat
Y:=round(10*sin(x/10));
SetPixel(x+200, -y+200, clBlue);
X:=x+1;
Until x> 300;
End.
```



Задачи для решения

1. Написать программу, организующую движение круга по вертикали, сверху вниз.
2. Построить графики функций: $y=x^2$, $y=tg\ x$, $y= e^x$, $y= \ln x$.
3. Нарисовать красивую картинку с использованием всех графических возможностей Пакаля. Добавить анимацию (например, по нажатию пробела в окне домика включается свет, солнце перемещается по небу и т.д.).

Задания для самостоятельного решения

1. Шарик, двигающийся из левого верхнего угла экрана в правый верхний.

2. Шарик, двигающийся из левого верхнего угла экрана в правый нижний.

3. Шарик, двигающийся из левого верхнего угла экрана по периметру в ту же точку.

4. Шарик, падающий вниз из середины экрана.

5. Написать программу движения по экрану окружности радиусом 10 по диагонали из левого верхнего в правый нижний угол экрана.

6. Написать программу движения по экрану окружности радиусом 10 по диагонали из правого верхнего в левый нижний угол экрана.

7. Начертить прямоугольник с координатами углов x1=130, y1=80, x2=220, y2=200 и в нем расставить 100 случайных точек ("звезды в окне").

8. Написать программу движения по экрану прямоугольника со сторонами 15 и 10 точек по вертикали сверху вниз.

9. Написать программу движения по периметру экрана окружности радиусом 10.

10. Написать программу движения по экрану окружности радиусом 10 по вертикали в центре экрана снизу вверх.

11. Написать программу движения по экрану окружности радиусом 10 по горизонтали центре экрана справа налево.

12. Написать программу одновременного (по очереди) движения по вертикали на экране 5 точек с любыми координатами ("капли дождя").

13. Написать программу одновременного (по очереди) движения по экрану трех окружностей радиусом 5 по горизонтали x1=100, x2=130, x3=150 со случайным приращением координаты по оси X для каждой окружности ("бег наперегонки").

ЗАДАЧИ ПО ВСЕМУ КУРСУ ДЛЯ ПОВТОРЕНИЯ

1. Составить программу решения системы неравенств. Напечатать исходные данные и результаты в принятом в математике виде.

$a_1x + b_1 < 0$

$a_2x + b_2 < 0$

2. Написать программу для разложения натурального числа на простые множители.

3. Заданное натуральное число N представить в виде суммы различных чисел Фибоначчи. Сколько слагаемых будет входить в эту сумму.

4. Дано натуральное число N.

а) выяснить, входит ли цифра 3 в запись числа N.

в) переставить первую и последнюю цифры числа N.

определить сумму его цифр.

5. Напечатать таблицу, состоящую из чисел от 1 до N и их факториалов.

12. Найти все "Пифагоровы тройки чисел" в первой тысяче натурального ряда. "Пифагоровой тройкой чисел" называются такие натуральные A, B, C, что $A^2 + B^2 = C^2$.

13. Дружественными числами являются два натуральных числа, таких, что каждое из них равно сумме всех натуральных делителей другого, исключая само это другое число. Написать программу, которая ищет дружественные числа в диапазоне от 1 до 10000.

14. Дана последовательность из не менее, чем двух натуральных чисел. Вычислить сумму тех из них, порядковые номера которых – простые числа.

15. Напечатайте в порядке возрастания первые 1000 чисел, которые не имеют простых делителей, кроме 2, 3, 5. (2, 3, 4, 5, 6, 8, 9, 10, 12, …).

16. Перевести заданное число N из десятичной системы счисления в двоичную.

17. Дана непустая последовательность натуральных чисел. Вычислить сумму тех из них, порядковые номера которых – числа Фибоначчи.

18. Задать число К из интервала [20, 50]. Задать еще 10 чисел из этого интервала. Сколько раз число К встретится среди них.

19. Дан массив чисел. Найти, сколько в нем пар одинаковых соседних элементов.

20. Составить программу решения неравенства ax2 + bx + c < 0. Напечатать исходные данные и результаты в принятом в математике виде.

21. Заданное натуральное число N представить в виде суммы различных чисел Фибоначчи. Сколько слагаемых будет входить в эту сумму.

22. Даны отрезки a, b, c, d. Для каждой тройки этих отрезков, из которых можно построить треугольник, напечатать площадь данного треугольника.

23. Дана последовательность из не менее, чем двух натуральных чисел. Вычислить сумму тех из них, порядковые номера которых – простые числа.

24. Дана непустая последовательность натуральных чисел. Вычислить сумму тех из них, порядковые номера которых – числа Фибоначчи.

25. Дано целое число, лежащее в диапазоне 1-999. Вывести его строку-описание вида "четное двузначное число", "нечетное трехзначное число" и т.д.

26. Дана последовательность натуральных чисел. Найти НОД всех членов последовательности.

27. Дана последовательность натуральных чисел. Найти НОК всех членов последовательности.

28. Дано целое число в диапазоне 1 - 7. Вывести строку - название дня недели, соответствующее данному числу (1 - "понедельник", 2 - "вторник" и т. д.).

29. Дано целое число K. Вывести строку - описание оценки, соответствующей числу К (1 - "плохо", 2 - "неудовлетворительно", 3 - "удовлетворительно", 4 - "хорошо", 5 - "отлично"). Если К не лежит в диапазоне 1 - 5, то вывести строку "ошибка".

30. Дан номер месяца - целое число в диапазоне 1 - 12 (1 - январь, 2 - февраль и т. д.). Вывести название соответствующего времени года ("зима", "весна", "лето", "осень").

31. Арифметические действия над числами пронумерованы следующим образом: 1 - сложение, 2 - вычитание, 3 - умножение, 4 - деление. Дан номер действия N (целое число в диапазоне 1 - 4) и вещественные числа A и B (B не равно 0). Выполнить над числами указанное действие и вывести результат.

32. Единицы длины пронумерованы следующим образом: 1 - дециметр, 2 - километр, 3 - метр, 4 - миллиметр, 5 - сантиметр. Дан номер единицы длины (целое число в диапазоне 1 - 5) и длина отрезка в этих единицах (вещественное число). Найти длину отрезка в метрах.

33. Найти среднее арифметическое положительных элементов таблицы A(100), стоящих на четных местах.

34. Даны 3 целочисленные таблицы A[1:K], B[1:N], C[1:M]. Известно, что существуют целые числа, встречающиеся во всех 3-х таблицах. Найти одно из таких чисел.

35. Дана целочисленная таблица A[1:100]. Подсчитайте наибольшее число одинаковых идущих в ней подряд элементов.

36. Подсчитайте количество различных чисел, встречавшихся в целочисленной таблице A[1:100]. Повторяющиеся элементы учитывайте один раз.

37. Построить алгоритм решения задачи. Даны таблицы вещественных чисел, причем

A(1)>=A(2)>=…>=A(N),

B(1)>=B(2)>=…>=B(K).

Постройте таблицу вещественных чисел, содержащую все элементы таблицы A и B так, чтобы C(1)>=C(2)>=…>=C(N+K).

38. Задать число K из интервала [20, 50]. Задать еще 10 чисел из этого интервала. Сколько раз число K встретится среди них.

39. Дан массив чисел. Найти, сколько в нем пар одинаковых соседних элементов.

40. Дан упорядоченный по убыванию массив чисел. Вводим число N. Вставить это число в упорядоченный массив так, чтобы массив-результат тоже был упорядочен по убыванию.

41. Дана прямоугольная матрица C размером K*L. Получить новую матрицу путем умножения всех элементов данной матрицы на наименьший по абсолютной величине.

42. Упорядочить строки матрицы B размером K*L по возрастанию диагональных элементов.

43. Дана квадратная матрица K порядка B, найти произведение элементов матрицы, расположенных в столбце с максимальным элементом по главной диагонали.

44. Описать алгоритм определения наибольшей суммы модулей элементов по столбцам матрицы C размером K*I.

45. Дана матрица N*M, состоящая из натуральных чисел. Найти в ней наименьший элемент и определить его местоположение. Если таких элементов несколько, то вывести на экран положение каждого из них.

46. Написать программу, удаляющую пробелы из заданной фразы.

47. Найти позицию начала первого появления в s1 текста, содержащего буквы s2 в обратном порядке.

48. Найти позицию первого символа в фразе s1 не являющегося буквой.

49. Дана строка t, написать программу для получения новой строки, в которой каждую группу подряд идущих букв "к" заменить на одну букву "к".

50. Составить программу для определения является ли данная буква гласной или согласной или одной из букв й, ъ, ь.

51. Обращение строки. Введите строку символов и выведите на экран эту строку в обратной последовательности.

52. Шифровка 1. Введите с клавиатуры строку символов и в каждой паре соседних символов этой строки поменяйте символы местами. Выведите зашифрованную строку на экран. Примените этот же способ для восстановления строки.

53. Шифровка 2. Введите строку символов, преобразуйте символы в коды таблицы кодировки и выведите получившуюся строку на экран. Раскодируйте получившуюся строку.

54. Используя коды чисел, решите задачу о проверке «счастливого» билета. Номер билета вводится с клавиатуры и состоит из четного числа цифр (число цифр неизвестно), если сумма цифр первой половины номера равна сумме оставшихся цифр номера, то этот билет считается «счастливым».

55. Напишите программу, которая производит подсчет количества каждого символа, встречающегося во вновь созданной строке. Результат вывода на экран должен иметь следующий вид:

aasdfgfasgas

a – 4

s – 3

d – 1

f – 2

g – 2

Сначала выводится исходная строка, затем результат подсчета для каждого символа только один раз.

56. Дан типизированный файл f, элементы которого – записи о сотрудниках учреждения. Каждая запись содержит следующие сведения: фамилия, имя, номер телефона. Написать программу поиска номера телефона и вывода его на экран по введенным с клавиатуры фамилии и имени сотрудника.

57. Описать процедуру, которая построчно печатает содержимое непустого файла, вставляя в начало каждой строки ее порядковый номер и пробел.

58. В текстовом файле t1 записана непустая последовательность целых чисел, разделенных пробелами. Описать процедуру positive(t1, t2), записывающую в текстовый файл t2 все положительные числа из t1.

59. В текстовом файле t записана непустая последовательность вещественных чисел, разделенных пробелами. Описать функцию max(t) для нахождения наибольшего из этих чисел.

60. Сформируйте список из фамилий и размеров обуви, которую носит каждый студент вашей группы. Используя этот список, выведите на экран фамилии студентов, начинающихся с «К», у которых размер обуви не менее 37.

61. Составить список студентов учебной группы их М человек. Для каждого студента указать ФИО, год рождения, год поступления в ВУЗ, курс, группу, оценки в 2 сессии по 3 предметам. Информацию о каждом студенте оформить в программе в виде записи. Составить программу, которая обеспечивает ввод исходной информации, вывод на экран в виде таблицы.

62. Составить список студентов учебной группы их М человек. Для каждого студента указать ФИО, год рождения, год поступления в ВУЗ, курс, группу, оценки в 2 сессии по 3 предметам. Информацию о каждом студенте оформить в программе в виде записи. Совокупность записей объединить в массиве записей (в типизированный файл).

63. Составить список студентов учебной группы их М человек. Для каждого студента указать ФИО, год рождения, год поступления в ВУЗ, курс, группу, оценки в 2 сессии по 3 предметам. Информацию о каждом студенте оформить в программе в виде записи.

64. Дана непустая последовательность символов. Требуется построить и распечатать множество, элементами которого являются встречающиеся в последовательности буквы от «T» до «X» и знаки препинания.

65. Дана непустая последовательность символов. Требуется построить и распечатать множество, элементами которого являются встречающиеся в последовательности цифры от «5» до «9» и знаки арифметических операций.

66. Дана непустая последовательность символов. Требуется построить и распечатать множество, элементами которого являются встречающиеся в последовательности буквы от «G» до «N» и цифры от «0» до «9».

67. Дана непустая последовательность символов. Требуется построить и распечатать множество, элементами которого являются встречающиеся в последовательности буквы от «A» до «F» и от «X» до «Z».

68. Дана непустая последовательность символов. Требуется построить и распечатать множество, элементами которого являются встречающиеся в последовательности знаки препинания.

69. Дан типизированный файл f, элементы которого – записи о сотрудниках учреждения. Каждая запись содержит следующие сведения: фамилия, имя, номер телефона. Написать программу поиска номера телефона и вывода его на экран по введенным с клавиатуры фамилии и имени сотрудника.

70. Создать функцию f(n) для расчета факториала числа n. Вывести в файл значения факториала для чисел от 2 до 10.

71. Создать функцию $f(x) = x + 10x - 3x + 2x\cos(2x) + 1$. Вывести в бинарный (или текстовый) файл в два столбика значения x от 0 до 10 с шагом в 0.1 и соответствующие значения f(x). Считать эти данные из файла и вывести их в строчку на экран компьютера.

72. Сгенерировать массив случайных целых чисел, состоящий из 10 элементов. Записать его в строчку в текстовый файл. Считать эти данные из файла, упорядочить этот массив и дописать в файл новую строчку с уже упорядоченным массивом.

73. Составьте программу для нахождения целых решений линейного Диофантова уравнения ax+by=c в интервале значений x и y от 0 до 100 Ввод коэффициентов организуйте через текстовый файл in.txt, а все найденные решения – out.txt.

ЗАКЛЮЧЕНИЕ

Настоящее учебно-практическое пособие включает в себя достаточный теоретический материал, а также примеры и задачи для изучения языка PascalABC студентами 1 – 2 курсов. Программа данного курса предусмотрена подготовкой бакалавров, изучающих дисциплину «Программирование».

Пособие направлено на развитие алгоритмического стиля мышления, а также умений и навыков работы с системой программирования, что поможет студентам освоить современные методы обработки информации, и успешно решать новые профессиональные задачи.

С точки зрения полноты представленного материал издание не претендует на то, чтобы заменить учебники. Однако студентам, изучающим программирование, оно будет полезно тем, что в нем сконцентрирован основной теоретический материал, необходимый для овладения основами языка PascalABC, а также представлены многочисленные примеры решенных задач, что облегчит переход к самостоятельному написанию программ.

Использование данного пособия в учебном процессе способствует формированию у студентов, обучающихся по направлению 44.03.05 – Педагогическое образование, компетенции ПК-3 (способность реализовывать образовательные программы различных уровней в соответствии с современными методиками и технологиями, в том числе информационными, для обеспечения качества учебно-воспитательного процесса).

ПРИЛОЖЕНИЕ 1. РЕАЛИЗАЦИЯ МЕЖПРЕДМЕТНЫХ СВЯЗЕЙ ПРИ ОБУЧЕНИИ ПРОГРАММИРОВАНИЮ НА УРОКАХ ИНФОРМАТИКИ И ИКТ

Связь информатики с математикой или физикой легко проследить и реализовать. Связь информатики с другими учебными дисциплинами менее очевидна, но ее можно реализовать в процессе составления тестов для проверки знаний или создавая электронный учебный курс по предметам, не связанным с информатикой. Здесь приводится пример реализации связей между информатикой (тема «Алгоритмизация и программирование») и школьным предметом литература.

Пожалуй, никакой другой предмет не обладает таким огромным потенциалом для организации межпредметных связей, как информатика и ИКТ. Информатика инновационна по самой своей природе. Информатика предоставляет другим дисциплинам мощный инструмент, который может оптимизировать учебный процесс.

По теме Алгоритмизация и программирование урожай межпредметных связей особенно богат; для каждого школьного предмета можно написать тестирующую, обучающую программу или создать электронный учебник и т.д. Многие математические задачи имеют разные алгоритмы решения. Очень полезно решение одной и той же задачи разными методами. Это позволяет учащимся не только решить задачу, но и сравнить методы решения, выбрать наиболее короткий и понятный.

Студентам, изучающим программирование, как будущим учителям, – следует уделить внимание связям между содержательной линией дисциплины информатика и ИКТ «Алгоритмизация и программирование» и школьной дисциплиной – литература. Прежде всего, информатика использует естественный язык в качестве одного из способов описания алгоритмов – начального этапа процесса программирования, а также для создания вербальных (т.е. словесных, текстовых) языковых моделей. Эти модели используют последовательности предложений на формализованных диалектах естественного языка для описания той или иной области действительности (примерами такого рода моделей являются полицейский протокол, правила дорожного движения, всяческие учебники).

Словесность же использует системный анализ для описания событий и ситуаций, а также средства информационных технологий для усиления образности художественных текстов.

Привлечение метода проектов сделает процесс программирования еще более увлекательным для школьников и студентов. По определению, проект – это совокупность определенных действий, документов, предварительных текстов, замысел для создания реального объекта, предмета, создания разного рода теоретического продукта. Это всегда творческая деятельность.

В соответствии с программой по литературе, каждый ученик (или группа учеников) выбирает стихотворение, не известное остальным учащимся. В каждой строке делается пропуск на месте одного из слов и предоставляются варианты ответов. Пользователю следует выбрать правильный вариант, тогда программа пропустит его к следующей строчке.

В процессе написания программы учениками наверняка будет выучено выбранное стихотворение. А в результате взаимного обмена готовыми программами запас выученных стихотворений пополнится у всех участников данного проекта.

Приведем пример подобной программы, написанной на языке PascalABC, в результате работы с которой ученики знакомятся со стихотворение **А.С. Пушкина** «Буря» и запоминают его.

```
Program Burya_Pushkin;

uses crt; // подключение модуля CRT для вывода информации в отдельном окне

label 1; // описание метки

var i, n: integer; // описание переменных

begin

writeln('Здравствуйте! Вам предстоит вставить пропущенные слова в стихотворение А.С. Пушкина. Чтобы выбрать правильный вариант ответа, вводите его номер.');

writeln;

writeln('Ты видел ... на скале');
```

```
writeln('1. Птицу.  2. Кустик.  3. Деву.  4. Зебру.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>3 then  //номер правильного ответа 3
begin
writeln('Не верно. Придётся начать заново.');
goto 1; // в случае ошибки переход  на конец программы
end
else
writeln('Правильный выбор!');
writeln;
writeln('В одежде ... над волнами');
writeln('1. Новой.  2. Белой.  3. Серой.  4. Мокрой.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>2 then
begin
writeln('Пушкин А.С. думал иначе. Придётся начать заново.');
goto 1;
end
else
writeln('Ура! Верно');
writeln;
writeln('Когда, бушуя в ... мгле');
writeln('1. Тёмной.  2. Серой.  3. Бурной.  4. Бурой.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>3 then
begin
writeln('Увы, не верно. Придётся начать заново.');
goto 1;
```

```
end
else
writeln('Пушкин думал так же');
writeln;
writeln('Играло море с ...');
writeln('1. Парусами.  2. Берегами.  3. Островами.  4. Челноками.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>2 then
begin
writeln('Не верно. Придётся начать заново.');
goto 1;
end
else
writeln('ДА!');
writeln;
writeln('Когда луч ... озарял');
writeln('1. Молний.  2. Света.  3. Яркий.  4. Прожектора.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>1 then
begin
writeln('Не верно. Придётся начать заново.');
goto 1;
end
else
writeln('Пушкин А.С. так и написал! Верно!');
writeln;
writeln('Её всечасно блеском ...');
```

```
writeln('1. Алым.  2. Ярким.  3. Светлым.  4. Синим.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>1 then
begin
writeln('Не верно. Придётся начать сначала.');
goto 1;
end
else
writeln('Да!');
writeln;
writeln('И ветер бился и ...');
writeln('1. Метал.  2. Играл.  3. Стонал.  4. Летал.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>4 then
begin
writeln('Не верно. Конец игры');
goto 1;
end
else
writeln('Верно! Идём дальше');
writeln;
writeln('С её летучим ...');
writeln('1. Одеялом.  2. Покрывалом.  3. Одеяньем.  4. Сарафаном.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>2 then
begin
writeln('Нет, так не пойдёт!');
```

```
goto 1;
end
else
writeln('Угадали!');
writeln;
writeln('... море в бурной мгле');
writeln('1. Ужасно.  2. Чудесно.  3. Опасно.  4. Прекрасно.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>4 then
begin
writeln('Увы, нет...');
goto 1;
end
else
writeln('Александр Сергеевич думал так же!');
writeln;
writeln('И небо в ... без лазури');
writeln('1. Звёздах.  2. Блёстках.  3. Тучах.  4. Пятнах.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>2 then
begin
writeln('Не верно. Начните сначала((((((');
goto 1;
end
else
writeln('Да, именно так.');
writeln;
writeln('Но ... дева на скале');
```

```
writeln('1. Верь мне.  2. Только.  3. Знай же.  4. Всё же.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>1 then
begin
writeln('Не верно. Жаль...');
goto 1;
end
else
writeln('Да! Да! Да!');
writeln;
  writeln('Прекрасней волн, ... и бури.');
writeln('1. И скал.  2. Морей.  3. Небес.  4. Ветров.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>3 then
begin
writeln('Обидно. Это ведь была последняя строчка. Начните заново.');
goto 1;
end
else
begin
clrscr;
writeln('Поздравляю! Теперь вы знаете ещё одно стихотворение А.С. Пушкина');
 writeln;   // на экран будет выведено все стихотворение целиком
writeln('       БУРЯ');
writeln('Ты видел деву на скале');
writeln('В одежде белой над волнами,');
writeln('Когда, бушуя в бурной мгле,');
```

```
writeln('Играло море с берегами,');
writeln('Когда луч молний озарял');
writeln('Её всечасно блеском алым');
writeln('И ветер бился и летал');
writeln('С её летучим покрывалом?');
writeln('Прекрасно море в бурной мгле');
writeln('И небо в блёстках без лазури;');
writeln('Но верь мне: дева на скале');
writeln('Прекрасней волн, небес и бури.');
end;
1: end.
```

В результате работы со следующей программой ученики знакомятся со стихотворение **Александра Блока** «О доблестях, о подвигах, о славе…» и запоминают его.

```
Program A_Blok;
uses crt; // подключение модуля CRT для вывода информации в отдельном окне
label 1; // описание метки
var i, n: integer; // описание переменных
begin
writeln('Здравствуйте! Вам предстоит вставить пропущенные слова в стихотворение А.А. Блока. Для выбора варианта ответа вводите его номер.
Для начала вспомним, как начинается стихотворение.');
writeln;
writeln('1. О подвигах, о доблести, о славе');
writeln('2. О мудрости, о подвигах, о славе');
writeln('3. О нежности, о доблести, о славе');
writeln('4. О доблестях, о подвигах, о славе ');
```

```
writeln('Введите номер правильного ответа'); readln(n);
if n<>4 then  //номер правильного ответа 4
begin
writeln('Не верно. Придётся начать заново.');
goto 1; // в случае ошибки переход  на конец программы
end
else
writeln('Правильный выбор!');
writeln;
writeln('Я ... на горестной земле,');
writeln('1. Вспоминал.  2. Забывал.  3. Написал.  4. Прочитал.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>2 then
begin
writeln('Блок А.А. думал иначе. Придётся начать заново.');
goto 1;
end
else
writeln('Ура! Верно');
writeln;
writeln('Когда, твоё лицо в ... оправе');
writeln('1. Резной.  2. Большой.  3. Простой.  4. Витой.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>3 then
begin
writeln('Увы, не верно. Придётся начать заново.');
goto 1;
end
```

```
else
writeln('Автор думал так же');
writeln;
writeln('Передо мной ... на столе.');
writeln('1. Глядело.  2. Сияло.  3. Стояло.  4. Блистало.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>2 then
begin
writeln('Не верно. Придётся начать заново.');
goto 1;
end
else
writeln('ДА!');
writeln;
writeln('Но ... настал, и ты ушла из дому.');
writeln('1. Час.  2. Срок.  3. День.  4. Миг.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>1 then
begin
writeln('Не верно. Придётся начать заново.');
goto 1;
end
else
writeln('Блок А.А. так и написал! Верно!');
writeln;
writeln('Я бросил ... заветное кольцою');
writeln('1. В ночь.  2. В печь.  3. В бак.  4. В окно.');
writeln('Введите номер правильного ответа'); readln(n);
```

```
if n<>1 then
begin
writeln('Не верно. Придётся начать сначала.');
goto 1;
end
else
writeln('Да!');
writeln;
writeln('Ты отдала свою ... другому,');
writeln('1. Любовь.  2. Мечту.  3. Печаль.  4. Судьбу.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>4 then
begin
writeln('Не верно. Конец игры');
goto 1;
end
else
writeln('Верно! Идём дальше');
writeln;
writeln('И я забыл ... лицо.');
writeln('1. Надменное.  2. Прекрасное.  3. Красивое.  4. С вес-
нушками.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>2 then
begin
writeln('Нет, так не пойдёт!');
goto 1;
end
```

```
else
writeln('Угадали!');
writeln;
writeln('Летели дни, кружась … роем,');
writeln('1. Огромным.  2. Жужжащим.  3. Заклятым.  4. Проклятым.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>4 then
begin
writeln('Увы, нет...');
goto 1;
end
else
writeln('Александр Александрович думал так же!');
writeln;
writeln('Вино и страсть ... жизнь мою…');
writeln('1. Сжигали.  2. Терзали.  3. Губили.  4. Портили.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>2 then
begin
writeln('Не верно. Начните сначала((((((');
goto 1;
end
else
writeln('Да, именно так.');
writeln;
writeln('И вспомнил ... пред аналоем,');
writeln('1. Я тебя.  2. Я себя.  3. Нас двоих.  4. Священника.');
```

```
writeln('Введите номер правильного ответа'); readln(n);
if n<>1 then
begin
writeln('Не верно. Жаль...');
goto 1;
end
else
writeln('Да! Да! Да!');
writeln;
writeln('И ... тебя, как молодость свою.');
writeln('1. Клял.  2. Ждал.  3. Звал.  4. Призывал.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>3 then
begin
writeln('Не верно.');
goto 1;
end
else
writeln('Yes!!!!');
writeln;
writeln('Я звал тебя, но ты …');
writeln('1. Не улыбнулась.  2. Не откликалась.  3. Не оглянулась.
4. Лишь ухмыльнулась.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>3 then
begin
writeln('Нет...');
goto 1;
```

```
end
else
writeln('Правильно!');
writeln;
writeln('Я слёзы лил, но ты  ....');
writeln('1. Не подошла.  2. Не снизошла.  3. Не утешила.  4. Опять ушла.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>2 then
begin
writeln('Нет, нет и нет!.');
goto 1;
end
else
writeln('Yes! Да!');
writeln;
writeln('Ты в ... плащ печально завернулась,.');
writeln('1. Синий.  2. Тёплый.  3. Белый.  4. Серый.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>1 then
begin
writeln('Ошибочка вышла.');
goto 1;
end
else
writeln('Всё так и есть!');
writeln;
writeln('В ... ночь ты из дому ушла.');
```

```
writeln('1. Слепую.  2. Сырую.  3. Глухую.  4. Шальную.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>2 then
begin
writeln('Не верно.');
goto 1;
end
else
writeln('Двигаемся дальше! Осталось немного))))');
writeln;
writeln('Не знаю, где приют ... гордыне');
writeln('1. Такой.  2. Лихой.  3. Своей.  4. Слепой.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>3 then
begin
writeln('Увы... Конец был уже близок.');
goto 1;
end
else
writeln('Вы правы!');
writeln;
writeln('Ты, милая, ты, ..., нашла…');
writeln('1. Странная.  2. Злобная.  3. Смелая.  4. Нежная.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>4 then
begin
writeln('Неправильно. Еще одна попытка.');
goto 1;
```

```
end
else
writeln('Молодец!');
writeln;
writeln('Я ... сплю, мне снится плащ твой синий,');
writeln('1. Крепко.  2. Сладко.  3. Глубоко.  4. Беспокойно.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>1 then
begin
writeln('Не верно. Возвращаемся к началу.');
goto 1;
end
else
writeln('Yes!');
writeln;
writeln('В котором ты в ... ночь ушла.');
writeln('1. Глухую.  2. Сырую.  3. Слепую.  4. Шальную.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>2 then
begin
writeln('Ошибка. Придётся начать заново. Скоро выучите всё стихотворение!');
goto 1;
end
else
writeln('Да! Продолжаем!');
writeln;
writeln('Уж не мечтать ... , о славе,');
```

```
writeln('1. О доблестях.  2. О подвигах.  3. О нежности.  4. О мужестве.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>3 then
begin
writeln('Не верно. Начинаем сначала.');
goto 1;
end
else
writeln('Верно!');
writeln;
writeln('Всё ... , молодость прошла!');
writeln('1. Растворилось.  2. Не сложилось.  3. Пролетело.  4. Миновалось.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>4 then
begin
writeln('Не верно. Начинаем сначала.');
goto 1;
end
else
writeln('Верно!');
writeln;
writeln('Твоё лицо в его ... оправе');
writeln('1. Простой.  2. Витой.  3. Большой.  4. Резной.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>1 then
begin
```

```
writeln('Обидно. Это ведь была предпоследняя строчка. Начните заново. Нужно доводить начатое до конца))');
goto 1;
end
else
writeln('Ура! И последний шаг. ');
writeln;
 writeln('Своей рукой ... я со стола.');
writeln('1. Переставил.  2. Смахнул.  3. Сместил.  4. Убрал.');
writeln('Введите номер правильного ответа'); readln(n);
if n<>4 then
begin
writeln('Обидно. Это ведь была последняя строчка. Начните заново. Нужно доводить начатое до конца))');
goto 1;
end
else
begin
clrscr;
writeln('Поздравляю! Теперь вы знаете ещё одно стихотворение А.А. Блока');
 writeln;   // на экран будет выведено все стихотворение целиком
writeln;
writeln('О доблестях, о подвигах, о славе');
writeln('Я забывал на горестной земле,');
writeln('Когда твоё лицо в простой оправе');
writeln('Передо мной сияло на столе.');
writeln;
writeln('Но час настал, и ты ушла из дому.');
```

```
writeln('Я бросил в ночь заветное кольцо.');
writeln('Ты отдала свою судьбу другому,');
writeln('И я забыл прекрасное лицо.');
writeln;
writeln('Летели дни, кружась проклятым роем,');
writeln('Вино и страсть терзали жизнь мою…');
writeln('И вспомнил я тебя пред аналоем,');
writeln('И звал тебя, как молодость свою.');
writeln;
writeln('Я звал тебя, но ты не оглянулась,');
writeln('Я слёзы лил, но ты не снизошла.');
writeln('Ты в синий плащ печально завернулась,');
writeln('В сырую ночь ты из дому ушла.');
writeln;
writeln('Не знаю, где приют своей гордыне');
writeln('Ты, милая, ты, нежная, нашла…');
writeln('Я крепко сплю, мне снится плащ твой синий,');
writeln('В котором ты в сырую ночь ушла.');
writeln;
writeln('Уж не мечтать о нежности, о славе,');
writeln('Всё миновалось, молодость прошла!');
writeln('Твоё лицо в его простой оправе');
writeln('Своей рукой убрал я со стола.');
end;
1: end.
```

В процессе работы над созданием программы школьники и студенты усваивают следующие операторы языка программирования и алгоритмические конструкции: организация ввода и вывода информации; оператор безусловного перехода; условный оператор в полной

форме; работа с модулем CRT; можно использовать также циклы с постусловием, если в результате неправильного ответа пользователю будут даваться еще попытки дойти до конца стихотворения.

Программу можно улучшить следующим образом:

- Сделать так, чтобы каждый новый вопрос выводился после предварительной очистки экрана.
- Сделать так, чтобы следующий вопрос выводился на экран после нажатия клавиши. Это можно сделать, добавив оператор READLN после правильного ответа на очередной вопрос, который будет ожидать нажатия любой клавиши.
- Подсчитать количество попыток дойти до конца теста. То есть после очередного неверного ответа программа не заканчивается, а отправляет пользователя на начало, запоминая в специальной переменной, сколько раз это произошло, прежде чем на все вопросы был получен правильный ответ.
- Учитель может использовать программу для контроля учащихся, когда задаёт им выучить стихотворение наизусть.
- Создать подобного вида тест по другим дисциплинам школьного курса.

ПРИЛОЖЕНИЕ 2. ИГРА «ЖИЗНЬ»

Понятие клеточного автомата

На протяжении всей своей истории человечество постоянно генерировало какие-то идеи, обеспечивающие его развитие. Делались открытия и появлялись изобретения в самых разных отраслях жизни. Каменный топор, колесо, бумага, автомобиль – предметы, вещества и соответствующие им понятия, без которых развитие цивилизации представить невозможно. Истина, бог, время, функция – абстрактные понятия, возникшие в процессе развития человечества. Все они возникали тогда, когда знания и потребности человека достигали определённого уровня.

По всей видимости, в конце 60-х годов XX века понятие «клеточный автомат» было просто необходимо для дальнейшего прогресса физики, биологии, химии, математики, компьютерных наук и других областей знаний, так они изобретались многократно под разными названиями, в разных странах, людьми, работающими в разных областях знаний. «Итеративные массивы», «вычисляющие пространства», «однородные структуры» и «клеточные автоматы» являются синонимами.

Когда Станислав Улам начинал свои исследования он, не предполагал, что, основываясь на его идеях, выдающийся американский учёный Джон фон Нейман придумает концепцию клеточных автоматов. Хотя фон Нейман был математиком и физиком, эта идея пришла к нему при решении задач из области биологии. Он использовал клеточные автоматы для создания более правдоподобных моделей пространственно протяжённых систем.

Результаты его исследований приведены в фундаментальной работе, которая вышла в 1966 году, в то время как фон Нейман умер в 1957. Дописывал её Артур Бёркс, ставший известным специалистом по клеточным автоматам.

Однако вернёмся назад во времени. В конце Второй Мировой Войны, в то время как фон Нейман создавал один из первых электронных компьютеров, немецкий инженер Конрад Цузе прятался от нацистов в австрийских Альпах. Там, в уединении, у него возникло множество идей из области параллельных вычислений. Среди прочего

он придумал и «вычисляющие пространства», то есть клеточные автоматы. Особый интерес Цузе вызывало их применения к задачам численного моделирования в механике. К сожалению политическая обстановка помешала работам учёному стать известными в то время. За работами же фон Неймана следил весь научный мир.

Однако настоящий эффект разорвавшейся бомбы произвела статья ведущего рубрики математических игр и головоломок журнала «Scientific American» Мартина Гарднера. Он опубликовал описание клеточного автомата Джона Хортона Конвея (John Horton Conway) «Жизнь». Игра «Жизнь», как стали называть этот автомат, фактически, стала культовой и сделала понятия «клеточный автомат» чрезвычайно популярным особенно среди людей с техническим образованием.

Область применения клеточных автоматов чрезвычайно широка. Однако необходимо отметить, что в подавляющем большинстве случаев, они применяются либо в качестве параллельных вычислительных сред, либо в качестве сред моделирования пространственно распределённых систем (например, физических, биологических, химических, социологических и других).

- Многофункциональная среда моделирования клеточных автоматов позволила бы использовать вычислительную машину в качестве:

- подчас весьма дорогостоящей экспериментальной установки для физических, биологических, химических и прочих опытов;

- средства реализации и визуализации алгоритмов параллельных вычислений.

Клеточный автомат является дискретной динамической системой, поведение которой полностью определяется в терминах локальных зависимостей.

Назовём дискретным пространством пространство над дискретным множеством элементов. Экземпляр пространства этого класса будем называть решёткой клеточного автомата, а каждый его элемент – клеткой. Каждая клетка характеризуется определённым значением из некого множества.

О клетке говорят, что она содержит или имеет соответствующее значение, либо находится или пребывает в состоянии, кодируемом данным значением. Оно можем быть булевым, целым, числом с плавающей точкой, множеством или другим объектом, в зависимости от потребностей задачи.

Совокупность состояний всех клеток решётки называется состоянием решётки. Состояние решётки меняется в соответствии с некоторым законом, который называется правилами клеточного автомата. Каждое изменение состояния решётки называется итерацией. Время в рассматриваемой модели дискретно и каждая итерация соответствует некому моменту времени.

Правила определяют, какое значение должно содержаться в клетке в следующий момент времени, в зависимости от значений в некоторых других клетках в текущий момент, а также, возможно, от значения, содержащегося в ней самой в текущий момент. Если новое состояние клетки зависит от текущего её состояния, то о соответствующем клеточном автомате говорят, что он является автоматом с клетками с памятью, иначе – автоматом с клетками без памяти.

Множество клеток, влияющих на значение данной, за исключением её самой, называется окрестностью клетки. Окрестность клетки удобнее задавать, если на решётке ввести метрику, поэтому далее, для удобства, будем говорить о решётке, как о дискретном метрическом пространстве.

Приведём пример клеточного автомата, который можно реализовать без использования вычислительной машины. Для этого необходимо взять пачку листов в клетку таких, чтобы при наложении одного листа на другой нижний слегка просвечивал. Кроме того, сетка, делящая лист на клетки, должна быть на каждом листе напечатана на одном и том же месте.

Допустим, что каждая клетка может содержать один бит информации. Это означает, что клетка может, например, быть либо помеченной, либо не помеченной. Пометим какие-либо клетки на первом листе, сформировав тем самым начальное состояние решётки. Для совершения итерации необходимо на первый лист положить второй и слегка обозначить (так, чтобы пометки на нижнем листе все же были

видны) на верхнем листе те клетки, которые на предыдущем листе также были помеченными, если среди восьми их ближайших соседей найдутся две или три помеченные клетки. Также легонько обозначьте те клетки, которые на предыдущем листе были пустыми, если среди их соседей найдётся ровно три помеченные клетки.

Когда описанная процедура будет проделана для всех клеток верхнего листа, те из них, которые слегка выделены, можно пометить окончательно. Дальше эту процедуру можно повторять от листа к листу до тех пор, пока хватит терпения.

Таким образом, можно реализовать клеточный автомат «Жизнь», который обладает множеством интересных свойств и любопытнейшей историей.

Предложенная модель обладает всеми упомянутые выше необходимыми характеристиками клеточных автоматов. Во-первых, у неё есть решётка, составленная из квадратов. Во-вторых, на решётке определена окрестность. Для каждой клетки она представляет собой множество из восьми непосредственно примыкающих к ней соседей. В-третьих, определено множество состояний клетки. В данном случае это – множество эквивалентное множеству из двух элементов {«жива», «мертва»}. В-четвёртых, описаны правила, обладающие свойством локальности (на каждую клетку влияют только клетки окрестности). В-пятых, автомат работает итерационно. Результат каждой итерации находится на отдельном листе бумаги.

Отметим основные свойства классической модели клеточных автоматов.

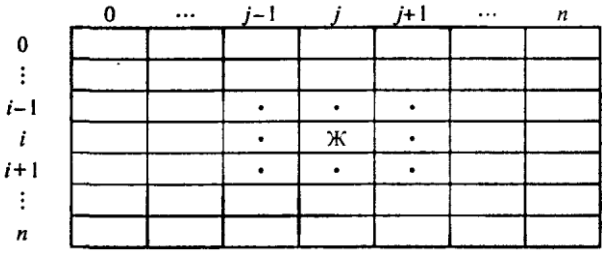
1. Локальность правил. На новое состояние клетки могут влиять только элементы её окрестности и, возможно, она сама;

2. Однородность системы. Ни одна область решётки не может быть отличена от другой по каким-либо особенностям ландшафта, правил и т.п. Однако на практике решётка оказывается конечным множеством клеток (ведь не возможно выделить не ограниченный объём данных). В результате могут иметь место краевые эффекты, клетки стоящие на границе решётки будут отличны от остальных по числу соседей. Во избежание этого можно ввести краевые условия, завернуть решётку в тор или, например, лист Мёбиуса.

3. Множество возможных состояний клетки – конечно. Это условие необходимо, чтобы для получения нового состояния клетки требовалось конечное число операций. Отметим, что оно не мешает использовать клетки для хранения чисел с плавающей точкой при решении прикладных задач. Например, переменная, имеющая тип данных double, может принимать, с точностью до знака, значения от 1.7·10–308 до 1.7·10+308 . Однако множество её значений конечно, а не континуально. Так как её размер – 8 байт, то множество её значений имеет мощность 264;

4. Значения во всех клетках меняются единовременно, в конце итерации, а не по мере вычисления. В противном случае порядок перебора клеток решётки, при совершении итерации, существенно влиял бы на результат.

Модель, которую опишем во всех деталях, предложена Д.Конвейем -имитационная модель роста, распада и различных изменений в популяции живых организмов, известная под названием «Жизнь».



0	1	0	0	0
0	0	1	0	0
0	1	1	1	0
0	0	0	0	0

Для построения алгоритма игры рассмотрим квадратное поле из n+1 столбцов и строк с обычной нумерацией от 0 до n. Крайние граничные столбцы и строки для удобства определим как «мертвую зону», они играют лишь вспомогательную роль.

Для любой внутренней клетки поля с координатами (i, j) можно определить 8 соседей. Если клетка «живая», ее закрашиваем, если клетка «мертвая», она пустая.

Зададим правила игры. Если клетка (i, j) «живая» и в окружении более трех «живых» клеток, она погибает (от перенаселения). «Живая» клетка также погибает, если в окружении менее двух «живых» клеток (от одиночества). «Мертвая» клетка оживает, если вокруг нее имеются три «живые» клетки.

Для удобства введем двумерный массив *A*[0..*n,* 0..*n*], элементы которого принимают значение 0, если соответствующая клетка пустая, и 1, если клетка «живая». Тогда алгоритм определения состояния клетки с координатой *(i,j)* можно определить следующим образом:

```
S = A[I-1, J-1] + A[I-1, J] + A[I-1, J+1] + A[I+1, J-1]
        + A[I+1, J] + A[I+1, J+l] + A[I, J+l] + A[1, J-1];
If (A[I, J] = 1) And ((S > 3) Or (S < 2); Then B(I, J] = 0;
If (A[I, J] = 0) And (S = 3) Then B[I, J] = 1;
```

Здесь массив *B* [0..*n,* 0..*n*] определяет координаты поля на следующем этапе. Для всех внутренних клеток от *i* = 1 до *n* - 1 и *j* = 1 до *n* - 1 справедливо сказанное выше. Отметим, что последующие поколения определяются аналогично, лишь стоит осуществить процедуру переприсваивания

```
For I = 1 To N-l
For J = 1 To N-l
A[I, J] = B[I, J]
Next J,I
```

На экране дисплея удобнее выводить состояние поля не в матричном, а в графическом виде. Это вполне можно осуществить. Осталось лишь определить процедуру задания начальной конфигурации игрового поля. При случайном определении начального состояния клеток подходит алгоритм

```
For I = 1 To K
Kl = RND*(N-l): K2 = RND*(N-l): A[K1, K2] = 1
Next i
```

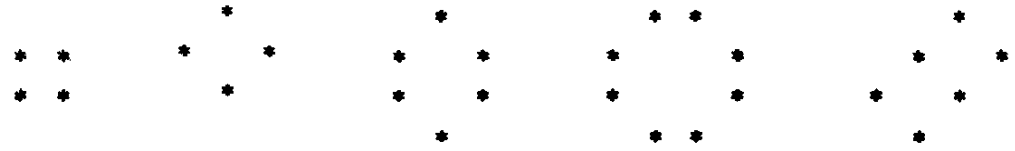
Интереснее случай, когда начальную конфигурацию мы задаем сами, и удобнее для пользователя делать это непосредственно в графическом виде.

Теперь приступим к моделированию. Поставим несколько вопросов:

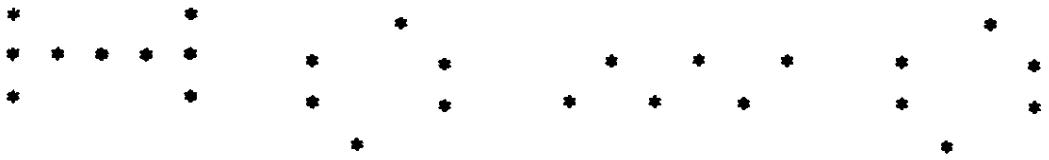
- какие конфигурации исчезают с течением времени и как быстро?
- какие структуры могут существовать бесконечно?
- каковы законы организации структур, их взаимодействия?

Перед попыткой ответить на эти вопросы читателю будет полезно поэкспериментировать самому. Предварительно отметим лишь некоторые структуры.

Стационарные структуры, не зависящие от времени



Существуют структуры, которые повторяют себя через 1, 2, ...,*n* временных шагов, их называют **циклами «-*n*»** .



Но наиболее интересны движущиеся структуры - «планеры», «корабли», «паровозы», их столкновения, «аннигиляции» и зарождение новых структур.

Задания: составить в тетради конспект по вопросам

1. Клеточные автоматы
2. История возникновения игры «Жизнь»
3. Правила игры
4. Примеры конфигураций

Ссылки:

1. https://life.written.ru/game_of_life_review_by_gardner
2. Игра «Жизнь» онлайн http://www.michurin.net/online-tools/life-game.html

3. https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%B0_%C2%AB%D0%96%D0%B8%D0%B7%D0%BD%D1%8C%C2%BB.

Алгоритм

1) Ввод данных с клавиатуры, чтобы задавать конфигурации, за которыми мы хотим проследить. Лучше использовать в качестве элементов матрицы пробелы и *.
2) Подсчитываем количество соседей: S:= a[i-1, j-1]+ a[i, j-1]+ a[i+1, j-1]+ a[i-1, j]+ a[i-1, j+1]+ a[i+1, j+1]+ a[i+1, j]+ a[i-1, j+1].
3) Если S<2 или S>3, то клетка погибает;
4) Если S=3, то в клетке появляется жизнь;
5) Если S=2, то в клетке жизнь продолжается.
6) Перерисовываем матрицу по нажатию клавиши.
7) Переприсваиваем матрицу для следующего шага.

1								…	22
				*					
				*					
				*					
				*					
22									

Чтобы не заботиться, есть ли у клетки соседи, можно описать массив с запасом, например, **для поля 10x10**:

```
a:array[0..11, 0..11] of integer;
b:array[0..11, 0..11] of string;
i, j, g, k:integer;
begin
for i:=1 to 10 do
```

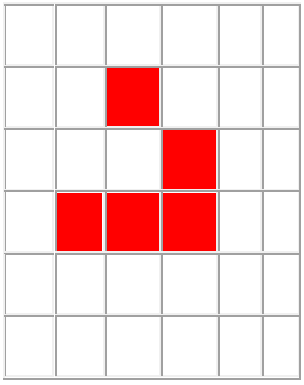
```
for j:=1 to 10 do
read(a[i,j]);
writeln;
```

1								…	22
				*					
				*					
				*					
				*					
22									

Задания.

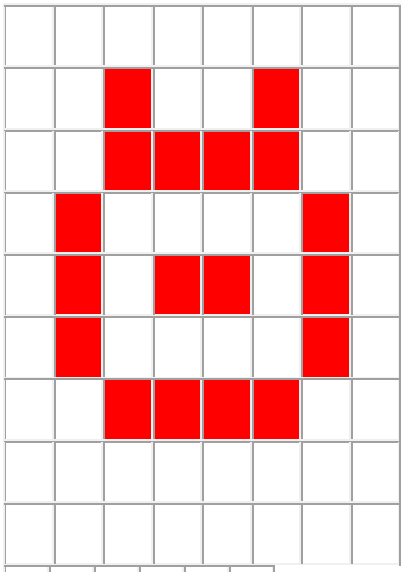
Проследить с помощью тренажера на сайте и с помощью написанной программы и зарисовать преобразования конфигураций:

1. Две горизонтальные клетки **
2. Три горизонтальные клетки ***
3. Четыре горизонтальные клетки ****
4. Пять горизонтальных клеток *****
5. Шесть горизонтальных клеток ******
6. Семь горизонтальных клеток *******
7. Исследуйте эволюцию "глайдера". Покажите, что он воспроизводится через каждые 4 хода (английское слово *glider* означает планер).

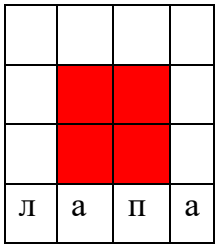


Чеширский кот

Проследите за судьбой «Чеширского кота». Убедитесь, что на шестом ходу от него остается только улыбка, которая затем пропадает, и остается лишь след лапы (блок из четырех фишек). (Напомним, что Чеширский кот — это персонаж сказки Л.Кэрролла "Алиса в стране чудес". Он обладает чудесной способностью исчезать, оставляя свою улыбку.)



И.	Т.	Д.			
у	л	ы	б	к	а



л	а	п	а

Пример программы «Игра Жизнь»

```
Program Game;
uses crt;
var
```

```
a:array[0..10,0..10] of integer;
b:array[0..10,0..10] of string;
i, j, g, k:integer;
begin
for i:=1 to 5 do
        for j:=1 to 5 do
        read(a[i,j]);
        writeln;
for i:=1 to 5 do begin
   for j:=1 to 5 do begin
      if a[i,j]=0 then b[i,j]:=' '  else b[i,j]:='*';
      write(b[i,j]:2);
   end;
writeln;
end;
writeln;
for g:=1 to 25 do begin readln;
clrscr;
for i:=1 to 5 do begin
   for j:=1 to 5 do begin
      k:=0;
      k:= a[i-1, j-1]+ a[i, j-1]+ a[i+1, j-1]+ a[i-1, j]+ a[i-1, j+1]+
a[i+1, j+1]+ a[i+1, j]+ a[i, j+1];
      if (a[i,j]=0) and (k=3) then b[i,j]:='*';
      if (a[i,j]=1) and (k<2) then b[i,j]:=' ';
      if (a[i,j]=1) and (k>3) then b[i,j]:=' ';
       write(b[i,j]:2);
   end;
```

```
writeln;
end;
for i:=1 to 5 do
        for j:=1 to 5 do
        if (b[i,j]='*') then a[i,j]:=1 else a[i,j]:=0;
        writeln;
end;
end.
```

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Сергеева, В. П., Технология проектирования в образовательном процессе: учебно-методическое пособие. / В. П. Сергеева, И. С Сергеева – М.: УЦ «Перспектива», 2016. – 156 с.
2. Поляков К.Ю. Информатика. 10 класс. Учебник. Базовый и углубленный уровни. В 2 частях. ФГОС / К.Ю. Поляков. – М.: Издательство «Бином. Лаборатория знаний», 2019. – 352 с.
3. Наумова С.Б. Реализация межпредметных связей информатики и литературы при обучении программированию. – Электронный журнал "Столица науки" №2 (31), февраль, 2021. 14 с.
4. Наумова С.Б., Степанова Д.О. Реализация межпредметных связей при обучении программированию на уроках информатики и ИКТ в 10 классе. Сборник: Актуальные проблемы современной науки: взгляд молодых ученых. Материалы VI Международной научно-практической конференции. Материалы Круглого стола. г.Грозный, 2020. С. 683-688.
5. Тишин, В. И. Программирование на Паскале [Электронный ресурс]: практикум / В. И. Тишин. -Эл. изд. - М.: БИНОМ. Лаборатория знаний. 2020.– 364 с.: ил.
6. Грацианова, Т.Ю. Программирование в примерах и задачах [Электронный ресурс / Т.Ю). Грацианова. - М.: БИНОМ. 2015 - (ВМК МГУ - школе).
7. Комлев, Н.Ю. Самоучитель игры на Паскале. АВС и немного Турбо. [Электронный ресурс] / Комлев Н.Ю. - М.: СОЛОН-ПРЕСС. 2013. –256 с.
8. Гарднер М. Математические досуги.–М: Мир, 1972.
9. Канцедал, С.А. Алгоритмизация и программирование: Учебное пособие / С.А. Канцедал. - М.: ИД ФОРУМ: НИЦ ИНФРА-М. 2014. - 352 с.: ил.; - (Профессиональное образование).
10. Давыдова Н.А. Программирование [Электронный ресурс]: учебное пособие/ Давыдова Н.А., Боровская Е.В. - Электрон, текстовые данные. - М.: БИНОМ. Лаборатория знаний, 2014. –239 с.

11. Златопольский Д.М. Программирование. Типовые задачи, алгоритмы, методы [Электронный ресурс] / Златопольский Д.М.– Электрон. текстовые данные.– М.: БИНОМ. Лаборатория знаний, 2012. – 223 c. Режим доступа http://www.iprbookshop.ru/12264.– ЭБС «IPRbooks»

Интернет-ресурсы

1. http://pascalabc.net/
2. http://projecteuler.net/
3. Задачник PascalABC http://interacia.net/index.php/2011-02-15-18-33-42/begin40.html
4. Пособие для учащихся «Основы программирования на Паскаль ABC» http://gigabaza.ru/download/64205.html

Учебное электронное издание

НАУМОВА Светлана Борисовна

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PascalABC

Учебно-практическое пособие

Издается в авторской редакции

Системные требования: Intel от 1,3 ГГц; Windows XP/7/8/10; Adobe Reader; дисковод CD-ROM.

Тираж 25 экз.

Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых
Изд-во ВлГУ
rio.vlgu@yandex.ru

Педагогический институт, кафедра ФМОиИТ
svetl.naumova2012@yandex.ru