Министерство образования и науки Российской Федерации
ФГБОУ ВПО «Удмуртский государственный университет»
Факультет информационных технологий и вычислительной техники
Кафедра теоретических основ информатики

А.Е. Анисимов

ПРАКТИКУМ ПО ОСНОВАМ ПРОГРАММИРОВАНИЯ

Учебно-методическое пособие

Ижевск

2014

УДК 004.424
ББК 32.973.26-018
А67

Рекомендовано к изданию Учебно-методическим советом УдГУ

Рецензент:
С.Ю. Купчинаус, кандидат технических наук, кандидат педагогических наук, доцент, заведующий кафедрой мультимедиа и интернет-технологий ФИТиВТ

Анисимов А.Е.

А67 Практикум по основам программирования: учебно-методическое пособие/А.Е. Анисимов.– Ижевск: Изд-во «Удмуртский университет», 2014. – 95 с.

Предлагаемое учебно-методическое пособие предназначено для использования в рамках дисциплин начального обучения программированию программ среднего профессионального образования по специальности «Информационные системы» и направлений подготовки высшего образования по программам бакалавриата «Прикладная информатика», «Информационные системы», «Фундаментальная информатика и информационные технологии» и ряда других.

Освоение практикума предполагает выполнение обучающимся ряда лабораторных работ для формирования начальных практических навыков программирования и закрепления теоретического материала дисциплин.

УДК 004.424
ББК 32.973.26-018

ISBN 978-5-4312-XXXX-X

Содержание

Введение

Данное учебно-методическое пособие предназначено для использования в рамках таких курсов программ среднего профессионального образования по специальности "Информационные системы" как «Основы алгоритмизации и программирования», и дисциплин высшего образования по программам бакалавриата направлений "Прикладная информатика", "Информационные системы", "Фундаментальная информатика и информационные технологии", таких как «Информатика и программирование», «Практикум на ЭВМ», «Технологии программирования» и ряда других.

Результатом прохождения практикума предполагается формирование у обучающихся ряда профессиональных компетенций, в том числе способность применять в профессиональной деятельности языки и технологии программирования, разработку алгоритмических и программных решений профессиональных задач. Практикум предназначен для закрепления студентами начальных сведений и получения первых навыков программирования, освоения базовых конструкций языков программирования.

Практикум состоит из серии лабораторных работ. В начале каждой из работ дается краткий теоретический и справочный материал по рассматриваемой теме, примеры решения задач. С этой частью материала студенту необходимо ознакомиться самостоятельно. Далее следуют предлагаемые для решения задачи: общая задача (для которой дается готовое решение в виде текста программы), самостоятельная задача (аналогичная общей, но для которой требуется составить программу самостоятельно) и индивидуальная задача, вариант которой определяет преподаватель. Студенту необходимо в компьютерном классе решить все три задачи и представить соответствующие программы для сдачи преподавателю. Тексты программ должны быть надлежащим образом оформлены и документированы. В конце каждой работы предлагаются контрольные вопросы для самопроверки или опроса студентов преподавателем.

Особенностью практикума является во многом самостоятельная деятельность студента при наличии необходимого контроля со стороны преподавателя и дифференцированный подход к распределению задач.

Лабораторная работа № 01. Типы, выражения, ввод и вывод

Тема

Типы, выражения, ввод и вывод

Цель

Познакомиться с базовыми (скалярными) типами языка программирования Pascal, построением выражений, операторами присваивания, ввода и вывода.

Теория

Типы

Любая компьютерная программа предназначена для обработки информации, или – данных. Данные могут быть разной природы (числовые, текстовые, графические и др.). В зависимости от этого операции по их обработке различаются. Например, два числа можно сложить (12 + 25 = 37), из двух символов можно получить новую строку ('a' + 'b' = 'ab'). Характеристики этих значений и применимых к ним операций описываются в программе *типом данных*.

Тип данных *определяет множество значений и набор операций над этими значениями*. Любые данные в программе принадлежат определенному типу. К простейшим (базовым) типам данных языка Pascal относят четыре семейства типов: целые, вещественные, символьный и логический типы. Для объекта каждого из типов выделяется память определенного размера в байтах. В следующей таблице приведено краткое описание каждого из этих типов[1].

[1] Описание множества значений и размера памяти типов дано для систем программирования, ориентированных на шестнадцатиразрядную архитектуру, таких как TurboPascal или FreePascal. Более точные данные можно узнать в документации конкретной системы или языка.

Тип или семейство типов	Имя типа	Множество значений	Размер памяти, байт	Операции	Пример выражения (в скобках – его значение)
Целые	byte	0…255	1	**+** сложение **–** вычитание ***** умножение **div** частное от деления **mod** остаток от деления	12 + 25 (37) 37–125 (-88) -24*4 (-96) 17 div 6 (2) 17 mod 6 (5)
	shortint	-128…127	1		
	word	0…65535	2		
	integer	-32768…32767	2		
	longint	-2147483648 … 2147483647	4		
Вещественные	single	[7-8 цифр] [10^{-45}…10^{+38}]	4	**+** сложение **–** вычитание ***** умножение **/** деление	2.5 + 0.01 (2.51) 3.71E02 – 6.1 (364.9) 3.14 * 2 (6.28) 50.50 / 25.25 (2.0)
	real	[11-12 цифр] [10^{-39}…10^{+38}]	6		
	double	[15-16 цифр] [10^{-324}…10^{+308}]	8		
	comp	[19-20 цифр] [2^{-63}…2^{+63}]	8		
	extended	[19-20 цифр] [10^{-4951}..10^{+4932}]	10		
Символьный	**char**	символы таблицы кодировки ASCII; у каждого из символов есть код (например, у символа 'a' код 97)	1	**ord(x)** – код символа x **chr(i)** – символ с кодом i	ord(‘a’) (97) chr(97) (‘a’)
Логический	**boolean**	FALSE, TRUE (означают ложь, истина)	1	**and** – «И» **or** – «ИЛИ» **xor** – «Искл. ИЛИ» **not** – «НЕ»	FALSE and TRUE (FALSE) FALSE or TRUE (TRUE) FALSE xor TRUE (TRUE) not FALSE (TRUE)

Таблица 1. Базовые типы языка Pascal

Кроме указанных операций к скалярным типам могут быть применены операции отношения (сравнения): < (меньше), > (больше), <= (меньше или равно), >= (больше или равно), = (равно), <> (не равно). Они всегда возвращают логическое значение (истина или ложь).

Выражения

Кроме операций для базовых типов в языке Pascal определены ряд стандартных функций, вычисляющих различные значения. Некоторые из них приведены в следующей таблице.

Математическое обозначение	Стандартная функция Pascal	Тип аргумента	Тип значения	Пример
|x|	abs(x)	вещественный	вещественный	abs(-5)
sin x	sin(x)	вещественный (радианы)	вещественный	sin(2*pi)
cos x	cos(x)	вещественный (радианы)	вещественный	cos(0.0)
arctg x	arctan(x)	вещественный	вещественный	arctan(pi/4)
x^2	sqr(x)	вещественный	вещественный	sqr(10.0)
$\sqrt{x}$	sqrt(x)	вещественный	вещественный	sqrt(1.96)
a^x	power(a, x)	вещественный, вещественный	вещественный	power(2, 5)
e^x	exp(x)	вещественный	вещественный	exp(4.5)
ln x	ln(x)	вещественный	вещественный	ln(1.0)
Дробная часть	frac(x)	вещественный	вещественный	frac(5.623)
Целая часть	int(x)	вещественный	целый	int(5.623)
Ближайшее целое	round(x)	вещественный	целый	round(5.62)
Случайное вещественное из полуинтервала	random	-	вещественный	random

[0; 1)				
Случайное целое из полуинтервала [0; x)	random(x)	целый	целый	random(100)

Таблица 2. Некоторые стандартные функции Pascal

С использованием операций и стандартных функций можно в программе построить различные сложные выражения. Например, математическое выражение

$$\frac{\left|\sin 2x^2 + cos\sqrt{x^2 + e^{2\pi}}\right|}{\ln(x+y)^3}$$

будет записано в программе на Pascal следующим образом: abs(sin(2 * sqr(x)) + cos(sqrt(sqr(x)+exp(2 * pi)))) / ln(power(x + y, 3))

В выражениях могут использоваться именованные объекты следующих видов.

Переменная – именованный объект программы определенного типа, способный хранить во время её работы определенное значение. Значение переменной можно изменить оператором присваивания.

Константа - именованный объект программы определенного типа, значение которого определено при описании и измениться не может.

Имена переменных и констант программы должны быть определены в соответствующих разделах описаний. Использовать имя в программе, не определенное ни в одном разделе описаний нельзя. Каждое имя в программе (но не в подпрограммах!) можно использовать один раз.

Оператор присваивания присваивает переменной значение выражения. Записывается в программе так:

```
ИмяПеременной := Выражение;
```

Тип переменной (слева) должен соответствовать типу выражения (справа). Разрешается вещественным переменным присваивать выражения целого типа. Нельзя использовать в выражениях переменную, которой ранее в программе не было присвоено значения.

Ввод, вывод

Для **ввода данных** с клавиатуры используется операторы

```
read(СписокПеременных);
readln(СписокПеременных);
```

При исполнении оператора ввода необходимо последовательно ввести с клавиатуры значения соответствующих базовых типов. Разница между этими операторами заключается в том, что после выполнения `read` следующее значение считывается в этой же строчке, а после `readln` – с новой строки.

Для **вывода данных** на экран используется операторы

```
write(СписокВыражений);
writeln(СписокВыражений);
```

Оператор последовательно выводит вычисленные значения выражений на экран. Разница между этими операторами заключается в том, что после выполнения `write` следующее значение выводится в этой же строчке, а после `writeln` – с новой строки.

Для **форматированного вывода** значения выражения оператором `write` или `writeln` указывают после двоеточия форматирующий параметр (целое число). Этот параметр указывает количество знакомест на экране, которые выделяются для вывода значения. Например, оператор `writeln(123:6)` выведет на экран число 123, а недостающие до шести три знака будут заменены пробелами. Для вывода *вещественного* значения указывают второй форматирующий параметр, означающий количество цифр дробной части числа. Например оператор `writeln(123.1:6:2)` выведет в дробной части числа два знака, то есть `123.10`.

Пример 1.1.

```
program Example1_1;
var
  x, y, s : integer;
begin
  writeln('Введите два целых числа:');
```

```
  readln(x, y);
  s := x + y;
  writeln('Сумма чисел равна ', s:6);
end.
```

Задача 1.1 (общая)

Написать и отладить программу PROG1_1, решающую следующую задачу: *Даны три положительных вещественных числа a, b и c. Считая, что a, b и c означают длины сторон треугольника, найти и вывести его периметр и площадь.*

Напомним, что периметр треугольника вычисляется по формуле
$p = a + b + c$,
а площадь – по формуле
$s = \sqrt{pp * (pp - a) * (pp - b) * (pp - c)}$, где $pp = \frac{p}{2}$ - полупериметр треугольника.

Ход работы.

1. Создайте новый файл, сохраните его в вашей персональной папке под именем prog11.pas.
2. Наберите в окне файла следующий текст, сохраняя все отступы и комментарии:

```
{Файл: prog11.pas
Задача:
  Даны три положительных вещественных числа a, b и c. Считая, что a,
  b и c означают длины сторон треугольника, найти и вывести его
  периметр и площадь.
Автор:
  Иванов В.В., гр. 39-11
Дата:
  10.09.2014 }

program PROG1_1;
var
  a, b, c, p, pp, s : real;
begin
  {Ввод}
  writeln('Введите стороны треугольника');
  readln(a, b, c);
  {Вычисление}
  p := a + b + c;    {Периметр}
  pp:= p / 2;        {Полупериметр}
  s :=sqrt (pp * (pp - a) * (pp - b) * (pp - c)); {Площадь}
  {Вывод}
  writeln('Периметр =',p:8:4);
  writeln('Площадь  =',s:8:4);
end.
```

3. Сохраните программу. Отладьте программу на следующей системе тестов (значения вводятся через пробел, в конце – нажать на клавишу ввода):

№ теста	Вход			Выход
	a	b	c	
1.	3	4	5	Периметр =12.0000 Площадь = 6.0000
2.	10	6	8	Периметр =24.0000 Площадь =24.0000
3.	1.5	1.5	2.5	Периметр =5.5000 Площадь =1.0364
4.	1	3	1	*{Аварийное завершение программы}*

4. Выясните, почему в последнем тесте программа аварийно завершилась?

Задача 1.2 (самостоятельная)

Написать и отладить программу PROG1_2, решающую следующую задачу: *Дано положительное вещественное число R. Найти площадь круга и длину окружности радиуса R.*

Напомним, что длина окружности радиуса R вычисляется по формуле $L = 2\pi R$, а площадь – по формуле $S = \pi R^2$. Значение числа π в Pascal имеет предопределенная константа `pi`, которую можно без описания использовать в выражениях. Для отладки программы заранее составьте (на бумаге) систему тестов.

Задача 1.3 (индивидуальная)

Написать и отладить программу PROG1_3, решающую задачу из следующего списка. Вариант задачи определяет преподаватель.

Вариант 01. Даны вещественные положительные числа a и b. Считая эти числа длинами сторон прямоугольника, найти его периметр, площадь и длину диагонали.

Вариант 02. Даны вещественные положительные числа a, b. Найти диагональ, площадь и периметр прямоугольного треугольника с катетами a и b.

Вариант 03. Даны вещественные положительные числа a, b и c. Считая эти числа длинами ребер прямоугольного параллелепипеда, найти его объём, площадь поверхности и длины диагоналей граней.

Вариант 04. Дано вещественное положительное число V. Считая это число объёмом куба, найти его ребро и площадь поверхности.

Вариант 05. Даны вещественные числа r1 и r2. Найти площадь кольца, образованного двумя окружностями с радиусами r1 и r2.

Вариант 06. Дано натуральное трехзначное число N. Найти сумму цифр этого числа. Указание: операция целочисленного деления `div` на 10 дает число без последней цифры, а остаток от деления `mod` на 10 - последнюю цифру числа.

Вариант 07. Даны вещественные числа a, b, c. Найти площадь треугольника со сторонами a и b и углу между ними с градусов. Указание: для пере-

вода градусов в радианы можно использовать формулу $\text{Радианы} = \text{Градусы} \cdot \frac{\pi}{180}$

Вариант 08. Даны вещественные числа a, d, n. Найти n-ый член и сумму n первых членов арифметической прогрессии с первым членом равным a и разностью d. Указание: $a_n = a_1 + (n-1)d$, $s_n = \frac{a_1+a_n}{2} n$.

Вариант 09. Даны вещественные числа x1, y1, x2, y2, x3, y3. Найти периметр и площадь треугольника, вершины которого имеют координаты (x1, y1), (x2, y2) и (x3, y3). Указание: расстояние между точками (x1, y1) и (x2, y2) можно найти по формуле $L = \sqrt{(x2-x1)^2 + (y2-y1)^2}$

Вариант 10. Даны целые неотрицательные числа h, m и s. Найти углы поворота в градусах часовой и минутной стрелок часов, когда они показывают время h часов m минут и s секунд

Вариант 11. Дано целое неотрицательное число N. Считая, что N обозначает длину, выраженную в верстах, выразить эту длину в километрах, метрах и сантиметрах. В одной версте 1066 м 80 см. Пример: N = 12 верст = 12 км 801 м 60 см.

Вариант 12. Дано целое неотрицательное число N. Считая, что N - это время в секундах, выразить это время в сутках, часах, минутах и секундах. Пример: N = 100000 секунд = 1 сутки 3 часа 46 минут 40 секунд.

Вариант 13. Даны значения трёх целочисленных переменных a, b, c. Переместить их значения так, чтобы переменная a получила бы исходное значение b, b получила бы значения c , а переменная c - значение a.

Вариант 14. Дано натуральное число N, не превосходящее 9999. Получить число M перестановкой цифр числа N в обратном порядке.

Вариант 15. *Даны два вещественных значения a и b. Найти среди них максимальное и минимальное без использования ветвления.

Контрольные вопросы

1. Что такое тип данных?
2. Какие базовые (скалярные) типы данных имеются в языке Pascal?

3. В чем сходны и чем отличаются друг от друга целые типы данных?
4. Что такое переменная? В чем сходство и различие переменной и константы?
5. Можно ли вещественной переменной присвоить целочисленное выражение?
6. Можно ли переменной целого типа присвоить вещественное выражение?
7. Для чего предназначены операторы ввода и вывода?
8. В чем различие между операторами `read` и `readln`? А между операторами `write` и `writeln`?
9. Как в программе отмечаются комментарии?
10. Можно ли использовать в программе переменную, которая не была объявлена в разделе переменных?
11. Можно ли использовать для двух разных переменных в программе одно и то же имя?
12. Для чего используются форматирующие параметры при выводе на экран?
13. Что означает каждый из форматирующих параметров?

Лабораторная работа № 02. Ветвление

Тема

Ветвление

Цель

Ознакомиться с программированием разветвляющихся алгоритмов, операторами условного перехода и выбора

Теория

Разветвляющийся алгоритм

Ветвлением в программе называют исполнение тех или иных операторов в зависимости от истинности некоторого условия. Блок-схема такого алгоритма приведена на рисунке.

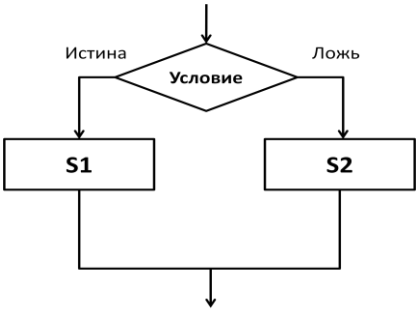


Рисунок 1. Разветвляющийся алгоритм

Если условие имеет значение «истина», то будет исполнен оператор S1, в противном случае («ложь») – оператор S2. Таким образом, ветвление может иметь две ветви, которые обязательно сходятся.

В частном случае одна из ветвей может отсутствовать, тогда ветвление называют неполным.

Условный оператор

Разветвляющийся алгоритм в программе реализует **условный оператор**. Его синтаксис:

```
if Условие then
  Оператор1
else
  Оператор2;
```

Ветвь **else** со вторым оператором может отсутствовать. Перед словом **else** точка с запятой никогда не ставится.

Если в одной из ветвей необходимо исполнить не один, а несколько операторов, тогда используют **операторные скобки begin** и **end**:

```
if Условие then
begin
  Оператор1_1;
  Оператор1_2;
  Оператор1_3;
end
else
  Оператор2;
```

Обратите внимание, что вложенные операторы набираются с отступом, обычно в два пробела. Тем самым структура программы становится более читабельной. Такой принцип набора текста программы называется «лесенка», он является стандартным для программирования практически на всех языках.

Пример 2.1.

Программа решает следующую задачу: *Даны два вещественных числа. Найти и вывести наибольшее из них.*

```
program Example2_1;
var
  a, b, max : real;
begin
  {Ввод}
  writeln('Введите два вещественных числа:');
  readln(a, b);
  {Вычисление}
  if a > b then
    max := a
  else
    max := b;
  {Вывод}
```

```
  writeln('Наибольшее значение равно ', max:6:2);
end.
```

Пример 2.2.

Программа решает следующую задачу: *Даны два вещественных числа b и c. Решить уравнение* $bx + c = 0$.

Ошибкой будет считать, что корень такого уравнение всегда равен $x = -\frac{c}{b}$, так как в условии задачи не сказано, что $b \neq 0$. Поэтому необходимо проверить это условие, чтобы программа во всех случаях выдавала правильный ответ.

```
program Example2_2;
var
  b, c, x : real;
begin
  {Ввод}
  writeln('Введите коэффициенты уравнения bx+c=0');
  readln(b, c);
  {Вычисление}
  if b <> 0 then
  begin
    x := - c / b;
    writeln('Корень уравнения x = ', x:6:2)
  end
  else
    if c <> 0 then
      writeln('Уравнение не имеет корней')
    else
      writeln('Любое число является корнем уравнения');
end.
```

Оператор выбора

Оператор выбора **case** также реализует ветвление в программе, но в особом случае, когда условием является последовательная проверка на равенство некоторого выражения каждому значению из некоторого набора. Синтаксис оператора следующий:

```
case Выражение of
  СписокЗначений1 : Оператор1;
  СписокЗначений2 : Оператор2;
  . . .
  else
    ОператорN;
end;
```

Значение выражения (которое после слова **case**) последовательно сверяется со значениями из списков. При совпадении с одним из значений будет исполнен соответствующий оператор. Если ни одно значение не равно значению выражения, то будет исполнен оператор после **else**. Как и в случае условного оператора, ветвь **else** может отсутствовать.

Пример 2.3.

Программа решает следующую задачу: *Дан номер месяца. Определить время года.*

```
program Example2_3;
var
  M : integer;
begin
  {Ввод}
  writeln('Введите номер месяца:');
  readln(M);
  {Вычисление}
  case M of
    12, 1, 2 : writeln('Зима');
    3, 4, 5  : writeln('Весна');
    6, 7, 8  : writeln('Лето');
    9, 10, 11: writeln('Осень');
    else
      writeln('Неверный номер месяца');
  end;
end.
```

Оператор безусловного перехода

Оператором безусловного перехода **`goto`** пользоваться не рекомендуется.

Задача 2.1 (общая)

Написать и отладить программу PROG2_1, решающую следующую задачу: *Даны три вещественных числа a, b и c, причем* $a \neq 0$. *Решить уравнение* $ax^2 + bx + c = 0$.

Уравнение задачи называется квадратным. Алгоритм его решения известен из школьного курса математики: начинается решение с нахождения дискриминанта $D = b^2 - 4ac$, и в зависимости от его знака уравнение может иметь два, один или ноль корней.

Ход работы.

1. Создайте новый файл, сохраните его в вашей персональной папке под именем prog21.pas.
2. Наберите в окне файла следующий текст, сохраняя все отступы и комментарии:

```
{Файл: prog21.pas
Задача:
  Даны три вещественных числа a, b и c, причем a<>0. Решить
  уравнение ax^2+bx+c=0.

Автор:
  Иванов В.В., гр. 39-11
Дата:
  11.09.2014 }

program PROG2_1;
var
  a, b, c, D, x1, x2: real;
begin
  {Ввод}
  writeln('Введите три числа a, b и c, причем a<>0');
  readln(a, b, c);
  {Вычисление}
  D := sqr(b) - 4 * a * c;
  if D > 0 then
  begin
    x1 := (- b - sqrt(D)) / (2 * a);
    x2 := (- b + sqrt(D)) / (2 * a);
    writeln('Корни уравнения x1=', x1:6:2, 'x2=', x2:6:2);
  end
  else
    if D = 0 then
    begin
      x1 := - b / (2 * a);
      writeln('Корень уравнения x1=', x1:6:2);
    end
    else
        writeln('Уравнение не имеет корней');
end.
```

3. Сохраните программу. Отладьте программу на следующей системе тестов (значения вводятся через пробел, в конце – нажать на клавишу ввода):

№ теста	Вход			Выход
	a	b	c	
1.	1	2	-3	Корни уравнения x1= -3.00

				x2= 1.00
2.	4	4	1	Корень уравнения x1= -0.50
3.	2	3	1.5	Уравнение не имеет корней

4. Придумайте еще три различных теста и протестируйте на них программу.

5. А что произойдет с программой, если ввести первый коэффициент a равным нулю? Почему?

Задача 2.2 (самостоятельная)

Написать и отладить программу PROG2_2, решающую следующую задачу: *Даны три вещественных числа a, b и c. Решить уравнение* $ax^2 + bx + c = 0$.

Указание. Казалось бы, условие задачи то же самое, что и в задаче 2.1. Однако в нем нет ограничения, что $a \neq 0$. Это значит, что в случае равенства нулю этого коэффициента уравнение превращается в линейное. Используйте решение примера 2.2 и задачи 2.1.

Задача 2.3 (индивидуальная)

Написать и отладить программу PROG2_3, решающую задачу из следующего списка. Вариант задачи определяет преподаватель.

Вариант 01. Даны три вещественных числа a, b и c. Найти среди них наибольшее значение.

Вариант 02. Дан номер года N. Определить, является ли он високосным. Примечание: в действующем григорианском календаре високосным считается года, если он делится на 4 ,за исключением годов, которые делятся на 100, но не делятся на 400.

Вариант 03. Даны значения трех вещественных переменных a, b и c. Перераспределить их значения в порядке не убывания, то есть чтобы $a \leq b \leq c$.

Вариант 04. Даны значения трех вещественных переменных a, b и c, отличные друг от друга. Обменять значения переменной, имеющей наибольшее значение с переменной, имеющей наименьше значение.

Вариант 05. Даны два вещественных числа x и y. Определить область координатной плоскости, которой принадлежит точка с координатами (x, y). Это может быть ось Ox или Oy, одна из четырех четвертей или начало координат.

Вариант 06. . Даны три вещественных числа a, b и c. Если треугольник с такими сторонами не существует. то выдать на экран число 0. Если треугольник равносторонний – то число 3; если равнобедренный, то число 2; для всех других треугольников число 1.

Вариант 07. . Даны четыре вещественных числа a, b, c и d. Определить, можно ли прямоугольник со сторонами a и b разместить в прямоугольнике со сторонами c и d или наоборот?

Вариант 08. Даны вещественные числа $x_1, y_1, x_2, y_2, x_3, y_3$. Определить, лежит ли точка с координатами (x_1, y_1) внутри или вне прямоугольника, стороны которого параллельны осям координат, а противолежащие вершины имеют координаты (x_2, y_2) и (x_3, y_3).

Вариант 09. Даны координаты точки (x, y), радиус r и координаты центра окружности (x_c, y_c). Определить, лежит ли точка внутри или вне круга, ограниченного указанной окружностью, или на самой окружности.

Вариант 10. Дано четыре числа a, b, c и d. Если они все равны друг другу, то вывести на экран число 4; если равны только три числа - то вывести число 3; если равны два числа, то вывести 2. Во всех остальных случаях вывести 1.

Вариант 11. Даны четыре точки на числовой оси A, B, C и D (известно, что A<B, а C<D). Найти длину пересечения отрезков AB и CD.

Вариант 12. Даны вещественные числа x_1, y_1, x_2, y_2. Указать, каким четвертям координатной плоскости принадлежат точки с координатами (x_1, y_1) и (x_2, y_2). Считать, что оси координат не включаются в четверти.

Вариант 13. Каждая клетка шахматной доски 8×8 определяется двумя целыми числами - номером горизонтали m и номером вертикали n. Даны четыре натуральных числа m_1, n_1, m_2, n_2. Проверить, бьет ли конь, находящийся на m_1, n_1клетку m_2, n_2?

Вариант 14. . Даны вещественные положительные числа x, y, a, b и c. Определить, пройдет ли кирпич с ребрами a, b и c в прямоугольное отверстие в стене со сторонами x и y?

Вариант 15. *Дано натуральное число N, меньшее 1000. Вывести на экран это число в виде числительного русского языка, то есть прописью. Например, число N=719 должно быть выведено так: «Семьсот девятнадцать».

Контрольные вопросы

1. Что содержит и как исполняется разветвляющийся алгоритм?
2. Может ли быть, чтобы ни одна из ветвей ветвления не была исполнена?
3. Может ли отсутствовать одна из ветвей ветвления?
4. Как записывается условный оператор?
5. Для чего используются слова **`begin`** и **`end`** в программе?
6. Для чего используются отступы слева при наборе текста программы?
7. Как записывается оператор выбора?
8. Могут ли быть в списках значений оператора выбора повторяющиеся значения?
9. Можно ли заменить оператор выбора другими операторами? Как это сделать?

Лабораторная работа № 03. Циклы. Цикл for

Тема

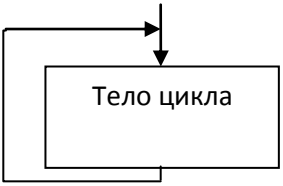
Циклы. Цикл for

Цель

Изучить понятие и назначение цикла. Научиться использовать цикл for

Теория

Цикл



Циклическим (циклом) называется такой алгоритм, в котором задана некоторая последовательность действий для многократного исполнения. Эта последовательность действий называется ***телом цикла***. Заметим, что тело цикла указано в алгоритме один раз, но исполняться оно может многократно. Однократное исполнение тела цикла называется ***итерацией.***

В языках программирования, как правило, используют три вида циклов:

- цикл со счетчиком (for),
- цикл с предусловием (while),
- цикл с постусловием (repeat).

Он отличаются друг от друга тем, как организовано прекращение исполнения цикла (выход). Кроме этого, в цикле for заранее известно количество повторений; напротив, для циклов с условиями это, как правило, неизвестно.

Цикл for

Эта разновидность цикла содержит специальную переменную (***счетчик***, параметр цикла), которая последовательно изменяет свое значение от заданного ***начального*** до ***конечного значения*** с некоторым ***шагом,*** при этом для каждого значения счетчика тело цикла исполняется один раз (то есть

происходит итерация). Обычно принято, что сам счетчик во время отдельной итерации не изменяется. В качестве параметра цикла можно использовать переменную только перечислимого типа (обычно – целого типа); вещественные переменные не могут быть параметрами цикла.

Так как начальное, конечное значения и шаг цикла явно заданы, то можно непосредственно до исполнения цикла заранее подсчитать количество итераций. Например, если известно, что студент вуза учится с 1 по 5 курс, причем на каждый курс он затрачивает один год, то всего он будет учиться 5-1+1=5 лет. Так как количество повторов исполнения тела цикла вычислимо еще до самого процесса, то цикл со счетчиком часто называют ***циклом с заранее известным числом итераций***.

Синтаксис цикла for с *увеличением* параметра от начального (НЗ) до конечного (КЗ) значения:

```
for Параметр := НЗ to КЗ do
  Оператор;
```

Синтаксис цикла for с *уменьшением* параметра от начального (НЗ) до конечного (КЗ) значения:

```
for Параметр := НЗ downto КЗ do
  Оператор;
```

Исполнение цикла for…to происходит в следующей последовательности:

1. вычисляются НЗ и КЗ;
2. параметру цикла присваивается НЗ;
3. если параметр не превзошел КЗ, то цикл продолжается, иначе – прекращается;
4. исполняется Оператор – тело цикла;
5. параметр увеличивается (или уменьшается) на одно значение;
6. повторяются шаги 3, 4 и 5.

Примеры

Пример 3.1.

Следующий фрагмент программы выведет на экран числа 1, 2, 3, 4, 5:

```
for i := 1 to 5 do
  writeln(i);
```

```
1
2
3
4
5
```

Заметим, что тот же результат даст программа:

```
  writeln(1);
  writeln(2);
  writeln(3);
  writeln(4);
  writeln(5);
```

но с циклом текст программы (код) получается короче и проще.

Пример 3.2.

Программа решает следующую задачу: *Дано целое положительное число N. Найти сумму и произведение целых чисел от 1 до N.*

В задаче по данному введенному натуральному числу N требуется найти сумму чисел

$S = 1 + 2 + 3 + \ldots + N$

и произведение чисел

$P = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N$.

Используемый в программе приём называется «цикл с накоплением суммы/произведения». Например, для N=5 сумма S=15, произведение P=120. Изучите программу.

```
program Example3_2;
var
  N, i, s, p : integer;
begin
  writeln('Введите число N');
```

```
 readln(N);

  {Нахождение суммы чисел от 1 до N}
  s := 0;
  for i:=1 to N do
    s := s + i;

  {Нахождение произведения чисел от 1 до N}
  p := 1;
  for i:=1 to N do
    p := p * i;

  {Вывод ответа}
  writeln('Сумма чисел от 1 до ', N, ' равна ', s);
  writeln('Произведение чисел от 1 до ',N,' равно ', p);

end.
```

Примечание. Произведение натуральных чисел от 1 и до N называется **факториалом числа** N, записывается

$N! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N$

Например, $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$.

Пример 3.3.

Программа решает следующую задачу: *Дано целое положительное число n. Найти n-ое число Фибоначчи.*

Бесконечная **последовательность чисел Фибоначчи** a_1, a_2, a_3, a_4, … определяется следующим образом:

$$a_1 = a_2 = 1;\ \text{для}\ i > 2:\ \ a_i = a_{i-1} + a_{i-2}$$

То есть два первых числа последовательности Фибоначчи – это 1 и 1, а каждое следующее равно сумме двух предыдущих: 1 1 2 3 5 8 13 21 34 55 89 144 и т.д.

Изучите программу.

```
program Example3_3;
var
  a1, a2, a3, n, i : integer;
begin
  writeln('Введите n');
  readln(n);

  if ((n = 1) or (n = 2)) then
    writeln(n,'-ое число Фибоначчи равно 1')
  else
  begin
    a1 := 1;
    a2 := 1;
    for i:=3 to n do
    begin
      a3 := a1 + a2;
      a1 := a2;
      a2 := a3;
    end;
    writeln(n,'-ое число Фибоначчи равно ', a3)
  end;
end.
```

Задача 3.1 (общая)

Написать и отладить программу PROG3_1, решающую следующую задачу: *Дано целое положительное число N. Найти сумму чисел*

$$\frac{1}{2^1}+\frac{1+2}{2^2}+\frac{1+2+3}{2^3}+\cdots+\frac{1+2+\cdots+N}{2^N}$$

Заметим, что в числителях дробей стоит сумма натуральных чисел 1+2+3+…, как в примере 3.2. В знаменателях – натуральная степень числа 2; нужно помнить, что такая степень – это произведение состоящее из n множителей, равных 2.

Также заметим, что указанное в задаче выражение можно записать короче, используя математическое обозначение с буквой «сигма»:

$$\sum_{i=1}^{N} \frac{1+2+\cdots+i}{2^i}$$

Ход работы.

1. Создайте новый файл, сохраните его в вашей персональной папке под именем prog31.pas.
2. Наберите в окне файла следующий текст, сохраняя все отступы и комментарии:

```
{Файл: prog31.pas
Задача:
Дано целое положительное число N. Найти сумму чисел
  N
СУММА  (1+2+…+i)  / 2^i
 i=1
Автор:
  Иванов В.В., гр. 39-11
Дата:
  11.09.2014  }
program PROG3_1;
var
  N, i, S1, P : integer;
  S : real;
begin
  writeln('Введите N');
  readln(N);

  S1 := 0;  {Сумма в числителе 1+2+…+i}
  P := 1;   {Степень 2^i}
  S := 0.0; {Общая сумма дробей}
  for i:=1 to N do
  begin
    S1 := S1 + i;
    P  := P * 2;
    S := S + S1 / P;
  end;
  writeln('Искомая сумма = ', S:6:2);
end.
```

3. Сохраните программу. Отладьте программу на следующей системе тестов:

№ теста	Вход	Выход
	N	
1.	1	Искомая сумма = 0.500
2.	3	Искомая сумма = 2.000
3.	5	Искомая сумма = 3.093
4.	10	Искомая сумма = 3.922

4. Своими словами опишите роль каждой из переменных **i, S1, P, S** в программе (для чего используется? какие значения принимает?).

Задача 3.2 (самостоятельная)

Написать и отладить программу PROG3_2, решающую следующую задачу: *Даны целое положительное число N и вещественное число a. Найти*

$$\prod_{i=1}^{N}\left(\frac{a^i}{1+2+\cdots+i}+1\right)$$

В задаче знак П означает произведение (по аналогии со знаком «сигма», означающем сумму). Для решения задачи предварительно составьте на бумаге систему тестов для различных (небольших) значений N и a.

Задача 3.3 (индивидуальная)

Вариант 01. Дано целое положительное число N. Найти

$$\sum_{i=1}^{N}\frac{i}{1+2+\cdots+i}$$

Вариант 02. Дано целое положительное число N. Найти

$$\sum_{i=1}^{N}\frac{3^i}{i!}$$

Вариант 03. Дано целое положительное число N. Найти

$$\sum_{i=1}^{N} \frac{2^i}{\frac{1}{1}+\frac{1}{2}+\cdots+\frac{1}{i}}$$

Вариант 04. *Дано целое положительное число N. Найти*

$$\sum_{i=1}^{N} \frac{i!}{1+2+\cdots+i}$$

Вариант 05. Дано целое положительное число N. Найти

$$\prod_{\mathrm{i}=1}^{\mathrm{N}} \frac{1+2+\cdots+\mathrm{i}}{\mathrm{i}!}$$

Вариант 06. Дано целое положительное число N. Найти

$$\prod_{\mathrm{i}=1}^{\mathrm{N}} \left(2^{\mathrm{i}} + \frac{1+2+\cdots+\mathrm{i}}{\mathrm{i}!}\right)$$

Вариант 07. Дано целое положительное число N. Найти

$$\sqrt{1+\sqrt{1+\sqrt{1+\cdots\sqrt{1}}}}$$ (всего N корней)

Вариант 08. Дано целое положительное число n. Найти

$$\cfrac{1}{2^{\mathrm{n}}+\cfrac{1}{2^{\mathrm{n}-1}+\cfrac{1}{\vdots}}}$$
$$2^2+\frac{1}{2^1}$$

Вариант 09. Дано целое положительное число n. Найти

$$\sqrt{\mathrm{n}+\sqrt{(\mathrm{n}-1)+\sqrt{(\mathrm{n}-2)+\cdots\sqrt{1}}}}$$

Вариант 10. Дано целое положительное число n. Найти

$$\sqrt{3n+\sqrt{3(n-1)+\sqrt{3(n-2)+\cdots\sqrt{3}}}}$$

Вариант 11. Дано целое положительное число n. Найти

$$\cfrac{1}{n+\cfrac{1}{(n-1)+\cfrac{1}{\vdots}}}$$
$$2+\frac{1}{1}$$

Вариант 12. Дано целое положительное число n, вещественное число x. Среди чисел $\sin x, \sin x^2, \sin x^3, \cdots, \sin x^n$ найти наибольшее, наименьшее и указать их номера.

Вариант 13. Дано целое положительное число n, вещественное число x. Найти значение многочлена вида $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x^1 + a_0$, если коэффициенты многочлена вводятся последовательно с клавиатуры в следующем порядке: $a_n, a_{n-1}, \ldots, a_1, a_0$. Указание: массивы не использовать.

Вариант 14. Дано натуральное число n. Среди натуральных чисел от 1 до n найти число с наибольшим количеством делителей.

Вариант 15. *Дано натуральное число n. Среди чисел Фибоначчи $a_1, a_2, a_3, \ldots, a_n$ найти все простые числа. Напоминание: простым называется целое число (большее 1), которое делится только на себя и на 1.

Контрольные вопросы

1. Что такое «цикл»?
2. Что такое «тело цикла» и «итерация»?
3. Какие виды циклов бывают?
4. Для чего используется параметр цикла в операторе for ?
5. Могут ли измениться начальное и конечное значения параметра цикла во время исполнения тела цикла?

6. Какого типа переменные допускается использовать в качестве параметра цикла for?
7. Почему цикл for называют циклом с заранее известным числом итераций?
8. Сколько раз будет исполнено тело цикла for...do, если начальное значение параметра больше конечного?
9. Опишите своими словами алгоритм накопления суммы. Чем отличается от него алгоритм накопления произведения?
10. Как организовать цикл, чтобы получить ряд значений с шагом отличным от 1? Например, с шагом 2 или с шагом -3 ?

Лабораторная работа № 04. Циклы с условием

Тема

Циклы с условием

Цель

Научиться использовать циклы с предусловием и постусловием

Теория

В отличие от цикла с параметром for, у которого ещё до начала исполнения заранее известно количество итераций, циклы с условиями применяют в тех случаях, когда количество повторов тела цикла *заранее неизвестно* и определяется только по ходу исполнения цикла.

Цикл с предусловием while

Цикл с предусловием состоит из условия цикла и его тела. Пока условие (логическое выражение) имеет значение «Истина», будет исполняться тело цикла.

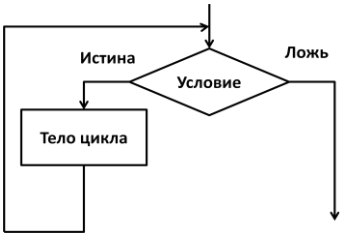


Рисунок 2. Цикл с предусловием

Как видно из блок-схемы, если условие цикла с самого начала имеет значение «Ложь», то тело цикла ни разу не будет исполнено. Если в процессе исполнения цикла условие всегда принимает значение «Истина», то цикл начинает исполняться бесконечно – происходит ***зацикливание.*** Это означает, что в алгоритме допущена ошибка.

Синтаксис цикла с предусловием:

```
while Условие do
  Оператор;
```

Цикл с постусловием repeat-until

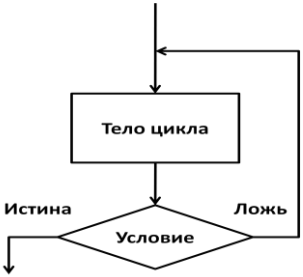


Рисунок 3. Цикл с постусловием

В цикле с постусловием вначале выполняется тело цикла, а затем проверяется условие (логическое выражение). Если условие ложно, то происходит возврат к повторному исполнению тела цикла и так далее.

Таким образом, хотя бы один раз тело цикла будет исполнено. Зацикливание происходит в том случае, если условие всегда имеет значение «Ложь».

Примеры

Пример 4.1.

Программа решает следующую задачу: *Дано целое число N. Вывести все натуральные степени числа 2 (2, 4, 8, 16 и т.д.), не превосходящие N.*

Для поиска всех степеней числа 2, не превосходящих данного N будем их последовательно накапливать в переменной p до тех пор, пока очередная степень не превзойдет N.

```
program Example4_1;
var
  N, p : integer;
begin
  writeln('Введите целое число N');
  readln(N);

  {Нахождение натуральных степеней числа 2, не превосходящих N}
  writeln('Степени числа 2, не превосходящие ', N);
  p := 2; {Первая степень числа 2}

  while p <= N do
  begin
    write(p,' ');
    p := p * 2;
  end;
end.
```

Заметим, что если ввести число N меньше 2, то программа вообще не выдаст результата, так как натуральных степеней 2, меньших 2, нет.

Пример 4.2.

Программа решает следующую задачу: *Дано вещественное положительное число x. Найти* $\sqrt{x}$ *с помощью итерационной формулы Герона с точностью 0.000001.*

Итерационная формула Герона определяет последовательность чисел, первое число в которой равно

$a_1 = x,$

а каждое последующее, начиная со второго, находится по формуле

$a_i = \frac{1}{2}\left(a_{i-1} + \frac{x}{a_{i-1}}\right)$, для $i = 2, 3, 4, \ldots$

Последовательность таких чисел быстро сходится к величине $\sqrt{x}$.

Например, для x=2 начало последовательности имеет вид: 2; 1.5; 1.417; 1.414 и т.д. Чем больше номер члена последовательности, тем он ближе к $\sqrt{2} \approx 1{,}41421356$

В приведенной программе друг за другом получают числа последовательности (в переменной `a`) до тех пор, пока квадрат очередного числа не будет отличаться от x менее, чем на 0.0000001, то есть пока выполняется условие $|a_i^2 - x| \geq 0.000001$.

```
program Example4_2;

var
  a, x : real;
begin
  writeln('Введите положительное x');
  readln(x);
  a:=x;
  while abs(sqr(a)-x) >= 0.000001 do
    a:= 1/2 * (a + x / a);

  writeln('Квадратный корень из ', x, ' примерно равен
',a:12:8);
end.
```

Пример 4.3.

Программа решает следующую задачу: *Дано вещественное число a. Определить, на каком номере слагаемого сумма*

$$\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots$$

превзойдет a.

Из курса математического анализа известно, что указанная бесконечная сумма является гармоническим рядом, который расходится (то есть достигает любого наперед заданного числа).

В приведенной ниже программе в сумму s последовательно добавляются слагаемые вида $\frac{1}{i}$ до тех пор, пока сумма не превзойдет *a*. Номер последнего слагаемого i и будет ответом задачи.

```
program Example4_3;
var
  s, a : real;
  i : integer;
begin
  writeln('Введите число a');
  readln(a);
  i:=0; {Номер слагаемого, знаменатель}
  s:=0; {Сумма}
  repeat
    i := i + 1;
    s := s + 1 / i
  until s > a;

  writeln('Номер слагаемого равен ', i);
end.
```

Задача 4.1 (общая)

Написать и отладить программу PROG4_1, решающую следующую задачу: *Дано целое число N > 1. Указать все простые множители, в произведение которых раскладывается N.*

Любое натуральное число разлагается в произведение простых чисел, причем единственным образом (с точностью до порядка перемножения). Например, $60 = 2 \cdot 2 \cdot 3 \cdot 5$.

В предлагаемой программе число N будем делить на число p, пока оно будет делиться; затем p увеличим на 1 и повторим деление. Начальное значение p = 2 (первое простое число).

Ход работы.

1. Создайте новый файл, сохраните его в вашей персональной папке под именем prog41.pas.
2. Наберите в окне файла следующий текст, сохраняя все отступы и комментарии:

```
{Файл: prog41.pas
Задача:
  Дано целое число N > 1. Указать все простые множите-
ли, в
  произведение которых раскладывается N.
Автор:
  Иванов В.В., гр. 39-11
Дата:
  10.09.2014
}

program PROG4_1;
var
  N, p : integer;
begin
  {Ввод}
  writeln('Введите число N > 1');
  readln(N);

  {Разложение}
  writeln(N,' разлагается в произведение простых чисел:');
  p:=2;                  {Первое простое число}
  while N > 1 do         {Пока N раскладывается}
    if N mod p = 0 then {Если N делится на p нацело}
    begin
      write(p, ' ');     {Выводим множитель p}
```

```
      N := N div p       {Делим N на множитель p}
    end
    else
      inc(p);{Увеличиваем на 1 – переходим к следующему
числу p}
end.
```

3. Сохраните программу. Отладьте программу на следующей системе тестов:

№ теста	Вход	Выход
	N	
1.	60	2 2 3 5
2.	3465	3 3 5 7 11
3.	17	17

4. *Ответьте на вопрос: Переменная p «пробегает» все натуральные значения, начиная с 2, но почему выводятся только *простые* числа?

Задача 4.2 (самостоятельная)

Написать и отладить программу PROG4_2, решающую следующую задачу: *Дано целое число N. Вывести на экран в порядке возрастания квадраты всех натуральных чисел, не превосходящие N.*

Использовать следующую таблицу тестов

№ теста	Вход	Выход
	N	
1.	10	1 4 9
2.	100	1 4 9 16 25 36 49 64 81 100
3.	-5	-

Задача 4.3 (индивидуальная)

Вариант 01. Дано натуральное число a. Найти номер слагаемого в сумме $1 + 2 + 3 + 4 + \cdots$, на котором эта сумма превзойдет a.

Вариант 02. Дано натуральное число a. Найти номер слагаемого в сумме $1! + 2! + 3! + 4! + \cdots$, на котором эта сумма превзойдет a.

Вариант 03. Дано натуральное число a. Найти номер слагаемого в сумме $2^1 + 2^2 + 2^3 + 2^4 + \cdots$, на котором эта сумма превзойдет a.

Вариант 04. Дано натуральное число a. Найти номер слагаемого в сумме $1 + (1 + 2) + (1 + 2 + 3) + \cdots$, на котором эта сумма превзойдет a.

Вариант 05. Дано вещественное число M. Среди чисел последовательности $1\sin 1, 2\sin 2, 3\sin 3, 4\sin 4, \ldots$ найти первое число, большее M, и указать его номер.

Вариант 06. Дано натуральное число N. Найти наибольшее число, факториал которого не превосходит N.

Вариант 07. Дано натуральное число N. Вывести в порядке возрастания все числа Фибоначчи, не превосходящие N.

Указание. Числа Фибоначчи были определены в примере 3.3 предыдущей лабораторной работы. Бесконечная последовательность чисел Фибоначчи a_1, a_2, a_3, a_4, … определяется следующим образом: $a_1 = a_2 = 1$: $a_i = a_{i-1} + a_{i-2}$ (для $i > 2$).

Вариант 08. Дано вещественное число a. Среди элементов последовательности

$\frac{1!}{1}, \frac{2!}{1+2}, \frac{3!}{1+2+3}, \ldots$ указать первый из элементов, больший a.

Вариант 09. Дано вещественное положительное число b. Среди элементов последовательности

$\frac{2^1}{1!}, \frac{2^2}{2!}, \frac{2^3}{3!}, \ldots$ указать первый из элементов, меньший b. Рекомендация: при тестировании вводите положительные числа b, близкие к нулю.

Вариант 10. Дано натуральное число N. Найти количество единиц в двоичной записи этого числа.

Вариант 11. Дано натуральное числоN. Найти сумму цифр этого числа.

Указание. Для выделения последней цифры целого числа N можно использовать операцию N mod 10, для её отбрасывания из числа – операцию N div 10. Таким образом можно последовательно выделить все цифры исходного числа.

Вариант 12. Дано натуральное число N. Найти наибольшую цифру этого числа. Указание - см. предыдущий вариант.

Вариант 13. Дано натуральное число N. Построить новое число удалением из числа N нечетных цифр. Указание - см. предыдущий вариант.

Вариант 14. Дано натуральное число N. Определить, является ли это число палиндромом? Указание: палиндром - это последовательность символов, одинаково читаемая слева направо и справа налево. Также см. указание к предыдущему варианту.

Вариант 15. *Дано натуральное число N. Разложить это число в произведение степеней с простыми основаниями и натуральными показателями. Пример: 60 = 2^2 × 3^1 × 5^1.

Контрольные вопросы

1. До каких пор будет выполняться цикл с предусловием?
2. До каких пор будет выполняться цикл с постусловием?
3. В каком случае произойдет зацикливание при исполнении цикла с предусловием?
4. Какое минимальное количество итераций возможно в цикле с предусловием? А в цикле с постусловием?
5. В каких случаях используют цикл с параметром, а в каких – цикл с условием?

6. В каких случаях наиболее подходящим будет использование цикла с предусловием, а в каких – с постусловием? Приведите примеры.
7. *Можно ли заменить в программе цикл с предусловием кодом с использованием цикла с постусловием? Если можно, то как это сделать?
8. *Можно ли заменить в программе цикл с параметром кодом с использованием цикла с предусловием? Если можно, то как это сделать?
9. *Можно ли организовать в программе цикл без использования любого из операторов цикла?

Лабораторная работа № 05. Линейные массивы

Тема

Линейные массивы

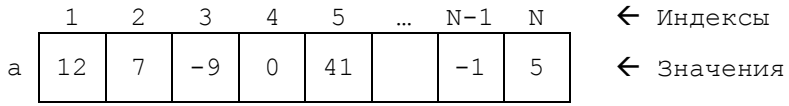
Цель

Изучить понятие линейного массива и основные операции с ним.

Теория

Линейный массив

Линейный массив – это структура, состоящая из индексированных элементов одного типа. Каждый элемент массива имеет свой индекс (номер) и некоторое значение.



Определить массив в программе можно, например, так:

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

Здесь объявлен массив из `N` (в данном случае – десяти) элементов целого типа.

Доступ к каждому элементу массива осуществляется по его индексу. Например, присвоить значение 41 пятому элементу массива можно так:

```
a[5] := 41;
```

Запись `a[i]` означает значение элемента, а выражение в квадратных скобках `i` – его индекс.

Операции с линейным массивом

Обычно, для последовательного перебора элементов массива используют цикл `for`:

```
for i:=1 to N do
  ...
```

– здесь в месте многоточия обычно находятся операторы, обрабатывающие очередной элемент массива `a[i]`.

Например, для нахождения суммы всех элементов массива можно использовать такой цикл:

```
S := 0;
for i:=1 to N do
  S := S + a[i];
```

Для нахождения индекса элемента с наибольшим значением – такой цикл:

```
imax := 1;
for i:=2 to N do
  if a[i] > a[imax] then
    imax := i;
```

В Pascal определена операция присваивания одному массиву значения другого массива, например:

```
a := b;
```

в этом случае произойдет поэлементное копирование значений из `b` в `a`. Такое присваивание допустимо, если оба массива имеют один и тот же тип.

Задача 5.1 (первая общая)

Написать и отладить программу PROG5_1, решающую следующую задачу: *Дан линейный массив из N целых чисел (N=10). Найти количество,сумму и среднее арифметическое всех четных чисел.*

В представленной ниже программе линейный массив заполняется случайными целями числами. Программа ничего не вводит, ответ сразу выдается на экран.

Ход работы.

1. Создайте новый файл, сохраните его в вашей персональной папке под именем prog51.pas.
2. Наберите в окне файла следующий текст, сохраняя все отступы и комментарии:

```
{Файл: prog51.pas
Задача:
  Дан линейный массив из N целых чисел (N=10). Найти
  количество,сумму и среднее арифметическое всех четных чисел.
Автор:
  Иванов В.В., гр. 39-11
Дата:
  11.09.2014 }
program PROG5_1;
const
  N  = 10;

var
  a     : array[1..N] of integer;
  i     : integer;
  s, c  : integer;

begin

  {Заполнение массива a случайными числами}
   randomize;
   for i:=1 to N do
     a[i]:=random(100);
```

```
  {Вывод массива a на экран}
  writeln('Массив:');
  for i:=1 to N do
    write(a[i]:6);
  writeln;

  {Подсчет количества и суммы четных чисел}
  c:=0;       {Количество}
  s:=0;       {Сумма}
  for i:=1 to N do
    if a[i] mod 2 = 0 then {Проверка четности значения}
    begin
      c:=c+1;
      s:=s+a[i];
    end;

  {Вывод ответа}
  if c <> 0 then
  begin
    writeln('Количество четных чисел = ', c :6);
    writeln('Сумма четных чисел = ', s :6);
    writeln('Среднее арифметическое четных чисел = ', s/c :6:2);
  end
  else
    writeln('В массиве нет четных чисел.');
end.
```

3. Сохраните программу. Отладьте программу. Проанализируйте, правильный ли ответ она выдает?

Задача 5.2 (вторая общая)

Написать и отладить программу PROG5_2, решающую следующую задачу: *Дан линейный массив из N целых чисел (N=10). Найти наибольший, и наименьший элементы массива, обменять их местами.*

Ход работы.

1. Создайте новый файл, сохраните его в вашей персональной папке под именем prog52.pas.

2. Наберите в окне файла следующий текст, сохраняя все отступы и комментарии:

```
{Файл: prog52.pas
Задача:
  Дан линейный массив из N целых чисел (N=10). Найти наибольший,
  и наименьший элементы массива, обменять их местами.
Автор:
  Иванов В.В., гр. 39-11
Дата:
  11.09.2014 }
program PROG5_2;
const
  N  = 10;

var
  a     : array[1..N] of integer;
  i     : integer;
  imin,imax  : integer;
  d     : integer;

begin

  {Заполнение массива a случайными числами}
   randomize;
   for i:=1 to N do
     a[i]:=random(100);

  {Вывод массива a на экран}
  writeln('Исходный массив:');
  for i:=1 to N do
    write(a[i]:6);
  writeln;

  {Нахождение индексов наибольшего и наименьшего элементов
массива}
  imin:=1;      {Индекс наименьшего элемента}
  imax:=1;      {Индекс наибольшего элемента}
  for i:=2 to N do
  begin
    if a[i] < a[imin] then
      imin := i;

    if a[i] > a[imax] then
      imax := i;
  end;
```

```
  {Вывод ответа}
  writeln('Наименьший элемент a[',imin,']=',a[imin]);
  writeln('Наибольший элемент a[',imax,']=',a[imax]);

  {Обмен наименьшего и наибольшего элементов}
  d:=a[imin];
  a[imin]:=a[imax];
  a[imax]:=d;

  {Вывод массива a на экран}
  writeln('Полученный массив:');
  for i:=1 to N do
    write(a[i]:6);
  writeln;
end.
```

3. Сохраните программу. Отладьте программу. Проанализируйте, правильный ли ответ она выдает?

Задача 5.3 (самостоятельная)

Написать и отладить программу PROG5_3, решающую следующую задачу: *Дан линейный массив из N целых чисел (N=10). Найти наибольшее значение среди нечетных элементов или сообщить, что таковых нет.*

Задача 5.4 (индивидуальная)

Написать и отладить программу PROG5_4, решающую задачу из следующего списка. Вариант задачи определяет преподаватель. Количество элементов массива можно взять произвольным, например, N = 20 или N=50.

Вариант 01. Дан линейный массив из N целых чисел (N=10). Найти наибольший элемент массива и обменять его местами с первым элементом.

Вариант 02. Дан линейный массив из N целых чисел (N=10). Найти наименьший элемент массива и обменять его местами с последним элементом.

Вариант 03. Дан линейный массив из N целых чисел (N=10). Найти наибольший элемент массива и определить, является он четным или нечетным?

Вариант 04. Дан линейный массив из N целых чисел (N=10). Найти наименьший элемент массива и определить, в какой половине массива он находится – в первой или второй?

Вариант 05. Дан линейный массив из N целых чисел (N=10). Найти индекс наибольшего элемента массива, затем – обнулить все элементы с большими, чем у него, индексами.

Вариант 06. Дан линейный массив из N целых чисел (N=10). Найти индекс наименьшего элемента массива, затем – обнулить все элементы с меньшими, чем у него, индексами.

Вариант 07. Дан линейный массив из N целых чисел (N=10). Найти индексы наибольшего и наименьшего элементов массива, затем – найти сумму элементов, расположенных в массиве между ними.

Вариант 08. Дан линейный массив из N целых чисел (N=10). Циклически сдвинуть элементы этого массива на две позиции вправо. Указание: циклический сдвиг на одну позицию вправо - это присваивание каждого значения элементу справа, а последнего значения - первому элементу.

Вариант 09. Дан линейный массив из N целых чисел (N=10). Найти все локальные максимумы в массиве и их обнулить. Указание: локальным максимумом в массиве называется элемент, значение которого больше его соседей - слева и справа (если они есть).

Вариант 10. Дан линейный массив из N целых чисел (N=10). Найти наибольший элемент в первой половине массива и наименьший элемент во второй половине массива, обменять их местами.

Вариант 11. Даны два линейных массива x и y по N целых чисел в каждом (N=10). Найти такие значения элементов, которые имеются и в x, и в y.

Вариант 12. Даны два линейных массива x и y по N целых чисел в каждом (N=10). Найти такие значения элементов, которые имеются в x, но отсутствуют в y.

Вариант 13. Дан линейный массив из N целых чисел (N=10). Переместить все отрицательные элементы массива в его начало, сохраняя взаимное расположение.

Вариант 14. Даны два линейных массива x и y по N вещественных чисел в каждом (N=10). Рассматривая пары значений $(x_1, y_1), (x_2, y_2), \ldots$ как координаты точек плоскости, указать номера двух наиболее удаленных точек.

Вариант 15. *Дан линейный массив из N целых чисел (N=10). Найти и вывести самую длинную неубывающую подпоследовательность подряд идущих элементов массива и ее длину. Если таковых несколько, то вывести одну из них. Пример: исходный массив 1 2 3 2 3 3 4 3 4 5, ответ 2 3 3 4, длина = 4.

Контрольные вопросы

1. Что такое линейный массив?
2. Что такое значение элемента массива?
3. Что такое индекс элемента?
4. Как производится доступ к элементу?
5. Как можно последовательно перебрать элементы с начала массива? С конца массива?
6. Можно ли в Pascal одному массиву присвоить значение другого массива целиком? При каком условии это возможно?
7. Что произойдет, если при выполнении программы будет обращение к несуществующему элементу массива? Как можно исправить эту ошибку?

Лабораторная работа № 06. Матрицы

Тема

Матрицы

Цель

Изучить понятие матрицы и научиться применять основные операции с матрицей.

Теория

Матрица

Матрица – это двумерный массив (таблица), состоящий из элементов одного типа. У каждого элемента матрицы два индекса – номер строки и номер столбца. Первый индекс означает номер строки, второй – номер столбца.

Пример матрицы 4 строки × 5 столбцов:

	1	2	3	4	5
1	a[1,1]	a[1,2]	a[1,3]	a[1,4]	a[1,5]
2	a[2,1]	a[2,2]	a[2,3]	a[2,4]	a[2,5]
3	a[3,1]	a[3,2]	a[3,3]	a[3,4]	a[3,5]
4	a[4,1]	a[4,2]	a[4,3]	a[4,4]	a[4,5]

Определить тип матрицы в программе можно так:

```
array [диапазон1, диапазон2] of тип;
```

Например:

```
const
  m = 4;
  n = 5;
```

```
var
  a : array [1..m, 1..n] of integer;
  b : array [0..1, byte] of real;
```

В этой программе определены целочисленная матрица a, состоящая из 4 строк и 5 столбцов, и вещественная матрица b, состоящая из 2 строк и 256 столбцов.

Доступ к элементам матрицы производится по индексам. Например, `a[2,3]` – элемент 2 строки и 3 столбца; `a[i, j]` – элемент в строке с индексом `i` и столбце с индексом `j`. При обращении к элементу матрицы индексы не должны выходить за пределы своих диапазонов, иначе возникнет ошибка времени исполнения. Например, в описанной выше матрице обращение `a[5, 4]` будет некорректным.

Операции с матрицами

Обычно для последовательного перебора элементов матрицы используют два цикла `for`, первый из которых (внешний) перебирает индексы строк, а второй (вложенный) – индексы столбцов.

```
for i:=1 to m do
  for j:=1 to n do
    ...
```

– здесь в месте многоточия обычно находятся операторы, обрабатывающие очередной элемент матрицы `a[i, j]`.

Например, для нахождения суммы всех элементов матрицы можно использовать такой код:

```
S := 0;
for i:=1 to m do
  for j:=1 to n do
    S := S + a[i, j];
```

Для нахождения индексов элемента с наибольшим значением – такой код:

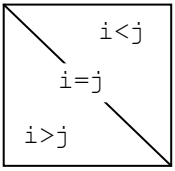
```
imax := 1;
jmax := 1;
```

```
for i:=1 to m do
  for j:=1 to n do
    if a[i, j] > a[imax, jmax] then
    begin
      imax := i;
      jmax := j;
    end;
```

Пример 6.1.

Программа решает следующую задачу: *Дано целочисленная квадратная матрица размером n×n. Найти сумму элементов, расположенных выше главной диагонали, и сумму элементов, расположенных ниже главной диагонали.*

В квадратных матрицах (то есть таких, у которых количество строк и столбцов совпадает), главной диагональю называется последовательность элементов, начиная от левого верхнего и до правого нижнего угла. Индекс строки и индекс столбца у каждого элемента главной диагонали совпадают – `a[1, 1]`, `a[2, 2]`, `a[3, 3]` и т.д. Для элементов `a[i, j]`, расположенных *выше* главной диагонали, выполняется условие `i < j`; соответственно, для элементов *ниже* главной диагонали – условие `i > j`:



Решение:

```
program Example6_1;
const
  n = 4;
var
  a : array [1..n, 1..n] of integer;
  i, j : integer;
  SumAbove, SumUnder : integer;
begin
  {Ввод матрицы с клавиатуры}
```

```
  writeln('Введите элементы матрицы:');
  for i:=1 to n do
    for j :=1 to n do
      read(a[i, j]);

  {Вывод матрицы на экран}
  writeln('Введенная матрица:');
  for i:=1 to n do
  begin
    for j :=1 to n do
      write(a[i, j]:4);
    writeln;
  end;

  {Вычисление}
  SumAbove := 0; {Сумма элементов выше главной диагонали}
  SumUnder := 0; {Сумма элементов ниже главной диагонали}
  for i:=1 to n do
    for j :=1 to n do
      if i < j then
        SumAbove:= SumAbove + a[i, j]
      else
        if i > j then
          SumUnder := SumUnder + a[i, j];

  {Вывод ответа}
  writeln('Сумма элементов выше главной диагонали ', SumAbove);
  writeln('Сумма элементов ниже главной диагонали ', SumUnder);
end.
```

Задача 6.1 (общая)

Написать и отладить программу PROG6_1, решающую следующую задачу: *Дана целочисленная прямоугольная матрица размера m×n, заполненная случайными числами. Найти сумму максимальных элементов строк матрицы.*

Ход работы.

1. Создайте новый файл, сохраните его в вашей персональной папке под именем prog61.pas.
2. Наберите в окне файла следующий текст, сохраняя все отступы и комментарии:

```
{Файл: prog61.pas
Задача:
  Дана целочисленная прямоугольная матрица размера m×n,
  заполненная случайными числами. Найти сумму максимальных
  элементов строк матрицы.
Автор:
  Иванов В.В., гр. 39-11
Дата:
  10.09.2014}

program PROG6_1;
const
  m = 4;
  n = 5;
var
  a : array [1..m, 1..n] of integer;
  i, j : integer;
  Max, SumMax : integer;
begin
  {Заполнение матрицы случайными числами}
  randomize;
  for i:=1 to m do
    for j:=1 to n do
      a[i, j] := random(100);

  {Вывод матрицы на экран}
  writeln('Исходная матрица:');
  for i:=1 to m do
  begin
    for j :=1 to n do
      write(a[i, j]:4);
    writeln;
  end;

  {Нахождение суммы максимальных элементов строк}
```

```
  SumMax := 0;      {Сумма максимумов строк}
  for i:=1 to m do {Перебор строк матрицы}
  begin
    {Нахождение максимума в строке с индексом i}
    Max := a[i,1];
    for j:=2 to n do
      if a[i, j] > Max then
        Max := a[i, j];
    SumMax := SumMax + Max; {Прибавление максимума в сумму}
  end;

  {Вывод ответа}
  writeln('Сумма максимальных элементов строк равна ', SumMax);
end.
```

3. Сохраните программу. Отладьте программу. Проанализируйте, правильный ли ответ она выдает?

4. Измените значения констант m и n на другие (например, 10 и 10, 2 и 15 и др). Проанализируйте, как изменяется при этом работа программы.

Задача 6.2 (самостоятельная)

Написать и отладить программу PROG6_2, решающую следующую задачу *Дана целочисленная прямоугольная матрица размера m×n, заполненная случайными числами. Среди сумм элементов столбцов матрицы найти наименьшую.*

Задача 6.3 (индивидуальная)

Написать и отладить программу PROG6_3, решающую задачу из следующего списка. Вариант задачи определяет преподаватель.

Вариант 01. Дана целочисленная квадратная матрица размера m×n, заполненная случайными числами. Найти максимальный элемент среди элементов, расположенных выше главной диагонали матрицы.

Вариант 02. Дана целочисленная квадратная матрица размера m×n, заполненная случайными числами. Найти произведение элементов, расположенных на главной диагонали матрицы.

Вариант 03. Дана целочисленная квадратная матрица размера m×n, заполненная случайными числами. Найти сумму элементов, расположенных выше побочной диагонали матрицы, и наименьшее значение среди элементов, расположенных ниже побочной диагонали матрицы. Указание. Побочная диагональ матрицы – это последовательность элементов, начиная от верхнего правого и заканчивая нижним левым элементом. Выясните вначале соотношение между индексами побочной диагонали.

Вариант 04. Дана целочисленная прямоугольная матрица размера m×n, заполненная случайными числами. Найти в каждой строке матрицы среднее арифметическое четных элементов и указать номер строки, где эта величина максимальна.

Вариант 05. Дана целочисленная прямоугольная матрица размера m×n, заполненная случайными числами. Найти в каждом столбце матрицы наименьшее значение и указать наибольшее среди них.

Вариант 06. Дана целочисленная прямоугольная матрица размера m×n, заполненная случайными числами. Найти наибольший элемент матрицы, и в той строке, в которой он находится найти наименьшее значение.

Вариант 07. Дана целочисленная прямоугольная матрица размера m×n, заполненная случайными числами. Найти и обменять местами наибольший и наименьший элементы матрицы.

Вариант 08. Дана целочисленная прямоугольная матрица размера m×n, заполненная случайными числами. Для каждого внутреннего элемента матрицы (то есть кроме крайних строк и столбцов) найти сумму значений его соседей, вывести эти суммы в виде матрицы. Сосед – это элемент матрицы, индексы которого отличаются от данного не более чем на 1.

Вариант 09. Дана целочисленная прямоугольная матрица размера m×n, заполненная случайными числами. Осуществить циклический сдвиг на одну позицию по часовой стрелке значений элементов, расположенных на внешнем слое матрицы (крайних строках и столбцах).

Вариант 10. Дана целочисленная прямоугольная матрица A размера m×n, заполненная случайными числами. Построить новую матрицу B такого

же размера, где каждый элемент равен сумме элементов соответствующей строки и столбца исходной матрицы.

Вариант 11. Дана целочисленная прямоугольная матрица размера m×n, заполненная случайными числами. Найти наибольший и наименьший элементы матрицы, затем обменять значениями строки, где они находятся.

Вариант 12. Дана целочисленная прямоугольная матрица размера m×n, заполняемая с клавиатуры. Указать номера тех строк матрицы, которые являются палиндромами. Указание: палиндром - это последовательность, одинаково читаемая слева направо и справа налево.

Вариант 13. Дана вещественная прямоугольная матрица размера m×n, заполненная случайными числами. Найти и указать индексы всех седловых точек матрицы. Указание: элемент матрицы называется седловой точкой, если он является наименьшим в строке и наибольшим в столбце, либо, наоборот – наибольшим в строке и наименьшим в столбце. Указание 2: используйте вспомогательные линейные массивы для хранения наибольших и наименьших значений строк и столбцов.

Вариант 14. Заполнить целочисленную квадратную матрицу размера n×n последовательными целыми числами от 1 до n^2 по строкам, причем в строках с нечетными номерами элементы последовательно располагаются слева направо, а с четными номерами - справа налево.

Вариант 15. *Заполнить целочисленную квадратную матрицу размера n×n последовательными целыми числами от 1 до n^2, двигаясь по закручивающейся внутрь спирали, начиная от верхнего левого угла матрицы.

Контрольные вопросы

1. Что такое матрица?
2. Как производится доступ к отдельному элементу матрицы?
3. Можно ли обратиться к элементу матрицы, используя *один* индекс?

4. В каком случае и когда может произойти ошибка «выход индекса за пределы своего диапазона»?
5. Как организовать перебор элементов матрицы по столбцам (сверху вниз)?
6. Как организовать перебор элементов главной диагонали квадратной матрицы? Элементов побочной диагонали?
7. Как можно обменять местами значения двух строк матрицы?
8. *Можно ли задать или изменить размер матрицы во время выполнения программы?
9. Сколько необходимо вложенных циклов, чтобы организовать перебор всех различных *пар элементов* матрицы?

Лабораторная работа № 07. Строки

Тема

Строки

Цель

Изучить строковый тип даных, операции и некоторые алгоритмы со строками

Теория

Для обработки символьной информации в программах используют строковый тип данных, позволяющий хранить и обрабатывать строки, слова, тексты.

Строка

Строка (строковый тип) – тип данных, значениями которого является последовательность символов.

Элементами строки являются символы, то есть значения типа char. Текущее количество символов в строке называется ее **длиной**. Длина строки может изменяться. Длина пустой строки, то есть не содержащей ни одного символа, равна нулю.

Строковый тип в программе обозначается так:

```
string
```
или
```
string[МаксимальнаяДлина]
```

В первом случае максимальная длина строки составляет 255 символов, во втором – указанное в скобках количество. Например:

```
var
  s    : string;
  name : string[100];
```

Значения строк записываются в виде последовательности символов, обрамляемой одиночными кавычками (апострофами). Например, `'Это`

строка символов!'. Пустая строка обозначается двумя апострофами - ''. Обращаться к кажому символу строки можно по его индексу, как в линейном массиве. Первый символ имеет индекс 1. Например,

```
s := 'МАМА';
s[1] := 'П';
s[3] := s[1]; {Значением строки s станет 'ПАПА'}
```

Операции со строками

Для обработки строк в языке Pascal реализованы различные стандартные функции и процедуры:

Операция	Назначение	Пример

s1 + s2	Конкатенация (сцепление) строк	`s1 := 'МАША';`
		`s2 := 'ПЕТЯ';`
		`s := s1 + '+' + s2;`

function **Length** (s: string): integer;	Возвращает длину строки s	`s := 'МАША+ПЕТЯ';`
		`L := length(s);`
		`{L=9}`

function **Pos** (s1, s: string): integer;	Возвращает позицию первого вхождения подстроки s1 в строку s	`s := 'ИВАНОВА';`
		`i := POS('ВА', s);`
		`{i=2}`

function **Copy** (s: string; index, count: integer): string;	Возвращает подстроку строки s длины count с позиции index	`s := 'БАКЕНБАРДЫ';`
		`s1 := Copy (s, 6, 4);`
		`{s1 = 'БАРД'}`

procedure **Insert** (s1: string; s: string; index: integer);	Вставляет подстроку s1 в строку s с позиции index	`s := 'abcd';`
		`insert('FF',s,3);`
		`{s = 'abFFcd'}`

procedure **Delete** (s: string; index,count: integer);	Удаляет из строки s count символов с позиции index	`s := 'ПАСПОРТ';`
		`delete(s,2,3);`
		`{s = 'ПОРТ'}`

Таблица 3. Некоторые стандартные операции со строками

Строки сравниваются посимвольно слева направо до первого несовпадающего символа. Например, будет истинно:

```
'ABCDE' < 'ABCF'
```

```
'2' >'12'
```

Недостающие символы более короткой строки считаются символами с кодом 0.

```
'PASCAL' < 'PASCALABC'
```

Пример 7.1.

Программа решает следующую задачу: *Дана строка символов. Найти в ней количество русских букв, латинских букв и цифр.*

Примечание. В данной программе также используются *множества* символов, обозначаемые в квадратных скобках (например, `['а'..'я']`) и операция проверки принадлежности множеству `in` (в математике - ∈).

```
program Example7_1;
var
  s : string;
  rus, lat, digit, i : integer;
begin
  writeln('Введите строку:');
  readln(s);

  rus := 0;   {Счетчик русских букв}
  lat := 0;   {Счетчик латинских букв}
  digit := 0; {Счетчик цифр}
  for i:= 1 to length(s) do
    if s[i] in ['а'..'я','А'..'Я'] then
      inc(rus)
    else
      if s[i] in ['a'..'z','A'..'Z'] then
        inc(lat)
      else
        if s[i] in ['0'..'9'] then
          inc(digit);

  writeln('Во введенной строке: ', s);
  writeln('  Русских букв - ', rus);
```

```
  writeln('  Латинских букв - ', lat);
  writeln('  Цифр - ', digit);
end.
```

Пример 7.2.

Программа решает следующую задачу: *Дана строка символов. Проверить баланс круглых скобок в строке.*

В этой программе последовательно просматриваются символы строки. Если очередной символ – открывающая скобка, то счетчик открытых скобок k увеличивается на 1, если закрывающая – то уменьшается на 1. При этом k не может быть отрицательным. Баланс будет сооблюден, если после такого прохода k снова станет равен 0.

```
program Example7_2;
var
  s       : string;
  L, i, k : integer;
begin
  {Ввод строки}
  writeln('Введите строку:');
  readln(s);

  {Проверка баланса}
  k := 0;          {Счетчик открытых скобок}
  L := length(s);  {Длина строки}
  i := 1;          {Индекс}
  while (i<=L) and (k>=0) do
  begin
    if s[i]='(' then
      inc(k);
    if s[i]=')' then
      dec(k);
    inc(i);
  end;

  {Вывод ответа}
```

```
  if k = 0 then
    writeln('Баланс скобок соблюден')
  else
    writeln('Баланс скобок не соблюден');
end.
```

Пример 7.3.

Программа решает следующую задачу: *Даны три строки символов s1, s2 и s3.* Заменить каждое вхождение строки s2 в строку s1 на строку s3. Пример: вход s1='БАКЕНБАРДЫ', s2='БА', s3='+8+'; выход: s1='+8+КЕН+8+РДЫ'.

```
program Example7_3;
var
  s1, s2, s3 : string;
  position : integer;
begin
  {Ввод данных}
  write('Введите строку s1=');
  readln(s1);
  write(' Введите строку s2=');
  readln(s2);
  write(' Введите строку s3=');
  readln(s3);

  {Замена в строке s1 всех вхождений s2 на s3}
  position := Pos(s2, s1); {Позиция первого вхождения s2 в s1}
  while position <> 0 do  {Пока вхождения s2 в s1 есть}
  begin
    Delete(s1, position, length(s2)); {Удаляем s2 из s1}
    Insert(s3, s1, position);         {Вставляем s3 в s1}
    position := Pos(s2, s1);   {Позиция след. вхождения s2 в s1}
  end;

  {Вывод ответа}
  writeln('s=',s1);
end.
```

Задача 7.1 (общая)

Написать и отладить программу PROG7_1, решающую следующую задачу: *Словом в строке называется последовательность букв, не прерываемая другими символами. Дана строка символов. Найти в строке самое длинное слово.*

Ход работы.

1. Создайте новый файл, сохраните его в вашей персональной папке под именем prog71.pas.
2. Наберите в окне файла следующий текст, сохраняя все отступы и комментарии:

```
{Файл: prog71.pas
Задача:
  Словом в строке называется последовательность букв, не
  прерываемая другими символами. Дана строка символов.
  Найти в строке самое длинное слово.
Автор:
  Иванов В.В., гр. 39-11
Дата:
  10.09.2014}

program PROG7_1;
const
  Letter : set of char = ['a'..'z','a'..'я'];{Множество букв}
var
  s : string;       {Входная строка}
  Wrd, WrdMax : string; {Слово, максимальное слово}
  i, L : integer;
begin
  {Ввод строки}
  writeln('Введите строку');
  readln(s);

  {Поиск в строке самого длинного слова}
  L:=length(s); {Длина входной сроки}
  WrdMax := ''; {Самое длинное слово}
  i := 1;      {Счетчик символов для прохода по строке}

  while i<=L do   {Пока не закончилась строка}
  begin
    {Пропуск символов, не являющихся буквами}
    while (i<=L) and not (s[i] in Letter) do
      inc(i);

    {Выделение из строки очередного слова в Wrd}
    Wrd:='';
    while (i<=L) and (s[i] in Letter) do
    begin
      Wrd:=Wrd+s[i];
      inc(i)
    end;

    {Проверка выделенного слова Wrd на максимальную
длину}
```

```
    if length(Wrd) > length(WrdMax) then
      WrdMax := Wrd;
  end;

  {Вывод ответа}
  if WrdMax <> '' then
    writeln('Самое длинное слово - ', WrdMax)
  else
    writeln('В строке нет слов...')
end.
```

3. Сохраните программу. Отладьте программу на системе тестов:

№ теста	Вход	Выход
	s	
1.	машина москвич - лучшая	Самое длинное слово - москвич
2.	мама мыла раму	Самое длинное слово - мама
3.	2*2=4	В строке нет слов…

4. Как изменить программу, чтобы она также учитывала заглавные буквы A…Z и А…Я?

Задача 7.2 (самостоятельная)

Написать и отладить программу PROG7_2, решающую следующую задачу. *Словом в строке называется последовательность букв, не прерываемая другими символами. Дана строка символов. Найти, сколько слов в строке такой же длины, как и первое слово.*

Рекомендация. Возьмите за основу программу задачи 7.1. При выделении слов из строки используйте общий счетчик слов, чтобы определить, первое это слово или не первое; и, дополнительно, счетчик слов, длина которых равна длине первого слова.

Задача 7.3 (индивидуальная)

Написать и отладить программу PROG7_3, решающую задачу из следующего списка. Вариант задачи определяет преподаватель.

Вариант 01. Дана строка символов s, символ c. Найти количество вхождений символа c в строку s. Пример: вход s='мама мыла раму', c='а'; выход: 4 вхождения.

Вариант 02. Дана строка символов s, символы c1 и c2. В строке s заменить каждое вхождение символа c1 на символ c2. Пример: вход s='мама мыла раму', c1='а', c2='0'; выход: s='м0м0 мыл0 р0му'.

Вариант 03. Дана строка символов s. Определить, является ли она палиндромом? Указание: палиндром - это последовательность символов, одинаково читаемая слева направо и справа налево.

Вариант 04. Даны две строки символов s1, s2. Найти количество вхождений строки s2 в строку s1. Пример: вход s1='БАКЕНБАРДЫ', s2='БА'; выход: 2 вхождения. Указание: используйте функции Pos и Delete.

Вариант 05. Даны две строки символов s1, s2. Удалить каждое вхождение строки s2 в строку s1. Пример: вход s1='БАКЕНБАРДЫ', s2='БА'; выход: s1='КЕНРДЫ'. Указание: используйте функции Pos, Delete.

Вариант 06. Даны две строки символов s1, s2. Удалить каждое вхождение строки s2 в строку s1, начиная со второго. Пример: вход s1='БАРАБАН', s2='БА'; выход: s1='БАРАН'. Указание: используйте функции Pos, Delete.

Вариант 07. Дана строка символов s, содержащая арифметическое выражение – сумму двух натуральных чисел. Найти значение этой суммы. Пример: вход '123+5558', выход 5681.

Вариант 08. Словом в строке называется последовательность букв, не прерываемая другими символами. Дана строка символов. Найти самое короткое слово строки. Подсказка: пустая строка не является словом!

Вариант 09. Словом в строке называется последовательность букв, не прерываемая другими символами. Дана строка символов. Найти, сколько раз в строке повторяется первое слово. Пример: вход s = 'мы не рабы, рабы не мы'; выход: 2 раза.

Вариант 10. Словом в строке называется последовательность букв, не прерываемая другими символами. Дана строка символов. Найти, сколько раз

в строке повторяется последнее слово. Пример: вход s = 'мы не рабы, рабы не мы'; выход: 2 раза. Подсказка: можно дважды пройти по строке.

Вариант 11. Словом в строке называется последовательность букв, не прерываемая другими символами. Дана строка символов. Найти количество слов, состоящих только из русских букв.

Вариант 12. Дана строка символов s, в которой могут быть скобки двух типов – круглые ( ) и квадратные [ ]. Проверить баланс скобок. Пример. вход: '1(2[34)5]', выход: баланс нарушен.

Вариант 13. Словом в строке называется последовательность букв, не прерываемая другими символами. Найти в строке все слова-палиндромы. Указание: палиндром – это последовательность символов, одинаково читаемая слева направо и справа налево.

Вариант 14. Словом в строке называется последовательность букв, не прерываемая другими символами. Дана строка символов, содержащая русские буквы и пробелы. Построить из неё новую строку, в которой первую букву каждого слова сделать прописной. Пример: вход 'мама мыла Раму', выход 'Мама Мыла Раму'.

Вариант 15. *Словом в строке называется последовательность букв, не прерываемая другими символами. Дана строка символов. Построить из неё новую строку, исключив повторные вхождения слов. Указание: можно построить "словарь" – массив найденных *различных* слов.

Контрольные вопросы

1. Что такое строковый тип данных?
2. Что такое длина строки?
3. Может ли изменяться длина строки во время исполнения программы?
4. Как обратиться к отдельному символу строки?
5. Как организовать последовательный перебор символов строки?
6. Что такое операция конкатенации строк?

7. Для чего нужна функция Copy при обработке строк?
8. Как происходит сравнение строк?
9. Сравните друг с другом следующие строки: '123' и '32'; 'АБВГДейка' и 'АБВГД'; 'ОПРСТ' и 'ОПРСЯ'.
10. *Можно ли определить длину сроки без использования функции Length?
11. Различаются ли символы алфавита верхнего и нижнего регистра при сравнении строк?

Лабораторная работа № 08. Подпрограммы

Тема

Подпрограммы

Цель

Изучить базовые механизмы процедурного программирования - понятие подпрограммы, вызов подпрограмм, передача параметров, локальные данные. Структурная организация программы

Теория

С развитием технологий программирования стало ясно, что разработка программ большого размера становится весьма трудоемким процессом, а преодоление трудностей, возникающих при этом, отнимает значительное количество времени программиста. Очевидным решением такого рода проблем является возможность разделения большой программы на относительно независимые и взаимодействующие составные части - подпрограммы. Каждая отдельная подпрограмма структурно похожа на программу и призвана решать логически самостоятельную часть общей задачи. Такой подход называется процедурным программированием.

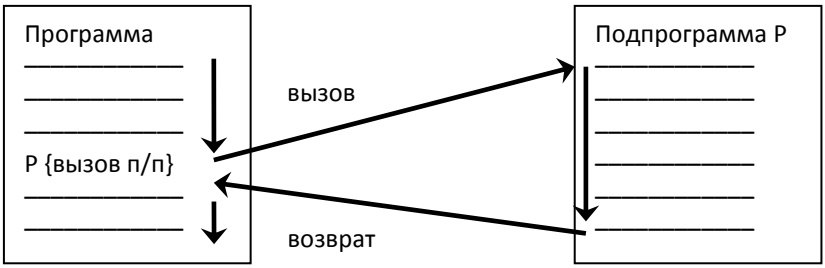
Для реализации его концепции в языки программирования были введены такие механизмы как определение и вызов подпрограммы, локальные данные и параметры. Кроме этого, программистским сообществом были выработаны соглашения по организации структуры и оформления программы для более эффективного и качественного процесса разработки программного обеспечения.

Подпрограмма определяется в программе только один раз, а использоваться может многократно. Это уменьшает размер текста программы, исключает повторную разработку уже созданного кода.

Основные понятия процедурного программирования

Подпрограмма – это относительно самостоятельная именованная часть программы, которая снабжена механизмами вызова и передачи параметров. Любая подпрограмма имеет имя.

Вызов подпрограммы – обращение к подпрограмме для её исполнения; при вызове управление передается операторам подпрограммы, а после их исполнения возвращается в точку вызова.



Возвращаемое значение – значение, которое подпрограмма после своего исполнения возвращает в точку вызова. Обычно это некоторый результат действия подпрограммы.

Виды подпрограмм в Pascal – процедуры и функции. **Процедура** (**`procedure`**) - это подпрограмма, не возвращающая значения в точку вызова. Процедура вызывается оператором вызова (указание имени подпрограммы). **Функция** (**`function`**) - подпрограмма, возвращающая в точку вызова возвращаемое значение. Обычно функция вызывается из некоторого выражения, в котором используется возвращаемое этой функцией значение.

Локальные данные – данные, определенные внутри какой-либо подпрограммы. Область видимости (доступность) локальных данных ограничена этой подпрограммой. Локальным данным противопоставляют **глобальные данные**, которые определены в самой программе и доступны отовсюду.

Параметры – специальные локальные переменные подпрограммы, используемые для передачи данных в подпрограмму и возврата результата из

неё. Таким образом, параметры используют для обмена данными с вызываемой подпрограммой. Различают **формальные параметры**, которые указывают в заголовке подпрограммы *при её описании*, и **фактические параметры**, которые подставляют на место формальных *при вызове* подпрограммы.

Описание процедур и функций

Описание каждой процедуры и функции состоит из заголовка и тела. В заголовке указываются тип подпрограммы (`procedure` или `function`), имя, формальные параметры. У функции в заголовке еще указывается тип возвращаемого значения. Тело подпрограммы состоит из локальных описаний (данных) и последовательности операторов между словами `begin` и `end`.

Синтаксис описания процедуры:

```
procedure ИмяПроцедуры(формальные_параметры);
  локальные_описания;
begin
  оператор;
  оператор;
  . . .
end;
```

Синтаксис описания функции:

```
function ИмяФункции(формальные_параметры):тип;
  локальные_описания;
begin
  оператор;
  оператор;
  . . .
end;
```

Формальные параметры и локальные описания в подпрограммах могут отсутствовать.

Среди операторов тела функции должен быть оператор вида

```
ИмяФункции := выражение;
```

Этот оператор определяет возвращаемое функцией значение.

Например, здесь определена процедура `TableSquares`, выводящая на экран таблицу квадратов натуральных чисел от 1 до n, где `n` передается процедуре как параметр:

```
procedure TableSquares (n : integer);
var
  i : integer;
begin {TableSquares}
  writeln('i  sqr(i)');
  for i:= 1 to n do
    writeln (i:4, sqr(i):4);
end; {TableSquares}
```

В следующем примере функция `Min` возвращает минимальное из двух переданных ей значений:

```
function Min(x, y : real) : real;
begin {Min}
  if x < y then
    Min := x
  else
    Min := y;
end; {Min}
```

Напомним, что параметры, указанные при определении подпрограммы называются формальными. В этих примерах параметры `n`, `x` и `y` – формальные.

Вызов процедур и функций

Вызов процедуры производится оператором вызова, то есть указанием имени этой процедуры со списком фактических параметров.

Например, вызов процедуры `TablesSquares` из предыдущего примера может выглядеть так:

```
TableSquares (10)
```

или

```
m :=7;
TableSquares (2*m);
```

Вызов функции производится, как правило, из некоторого выражения, в котором затем используется возвращаемое функцией значение.

Например, вызов функции `Min` из предыдущего примера может выглядеть так:

```
m := Min(7.1, -10);
```

или

```
a := -1.5;
b := 2.1;
c := Min (a, b) + Min (a+b, a-b);
```

Количество и типы фактических параметров должны соответствовать количеству и типам параметров формальных. При нарушении этого правила компилятор сообщит об ошибке. Например, указанную выше функцию `Min` нельзя вызвать так

```
m := Min(1, 2, 3);
```

так как фактических параметров должно быть столько, сколько формальных - два, а не три. А процедуру `TablesSquares` нельзя вызвать так:

```
TableSquares (1.5);
```

так как фактический параметр должен быть целого типа.

Стандартные процедуры и функции

Помимо определения и использования собственных подпрограмм в программе, программисту доступны большое количество готовых процедур и функций для выполнения самой разной работы.

Ряд стандартных функций и процедур являются **встроенными** в язык Pascal, это означает, что их можно непосредственно использовать в программе. Например, такими подпрограммами являются `sin(x)`, `random(n)`,`copy(s,i,c)`.

Большое количество стандартных функций и процедур находятся **в стандартных модулях**, для подключения которых к программе используют директиву `uses`. К таким модулям относятся Crt, System, Graph и другие. Подробную информацию о стандартных подпрограммах и модулях можно получить в справочной системе используемой системы программирования.

Локальные данные

Областью видимости локальных данных является та подпрограмма, в которой они определены. Время их жизни (то есть период, когда для них выделена память и они доступны для использования) совпадает с временем выполнения подпрограммы.

Подпрограммы могут быть вложенными, то есть внутри одной подпрограммы может быть определена другая. Если имя локальный переменной совпадает с именем глобальной переменной (или локальной переменной внешней подпрограммы), то использоваться будет именно та переменная, которая определена ближе всего к месту использования.

Например, в программе

```
program P1;
var
  t : integer; {глобальная переменная t}
  procedure P;
  var
    t : integer; {локальная переменная t}
  begin {P}
    t := 100; {присваивание локальной переменной}
  end;  {P}
begin
  t := 200; {присваивание глобальной переменной}
end.
```

определение локальной переменной с именем `t` делает невозможным обращение внутри процедуры `P` к одноименной глобальной переменной `t`. Это называется "перекрытие имени".

Обычно в подпрограммах определяют те переменные, которые понадобятся для реализации их логики и, наоборот, стараются не использовать переменные, определенные в другом месте (глобальные или локальные других подпрограмм). Такой подход называется "**локализация имён**", он делает логику всей программы более понятной. Сформулировать локализацию имен можно так: "определяйте данные как можно ближе к месту их использования".

Напротив, неявное изменение подпрограммой "чужих" переменных затрудняет понимание структуры программы и её алгоритма. Такое использование называется "**побочным эффектом**".

Например, в программе

```
program P2_1;
var
  t : integer; {глобальная переменная t}
  procedure P;
  begin {P}
    t := t + 1; {изменение глобальной переменной; по-
бочный эффект!}
  end;  {P}
begin
  t := 1;
  P;
  P;
  P;
  writeln(t);
end.
```

после трех вызовов процедуры P глобальная переменная `t` изменит свое значение с 1 на 4, причем это изменения будет скрыто внутри процедуры. Понять, почему "внезапно" изменилась `t` в теле самой программы может быть непросто - нужно будет исследовать все вызовы подпрограмм. Вместо этого тот же алгоритм можно реализовать и без побочного эффекта:

```
program P2_2;
var
  t : integer; {глобальная переменная t}
```

```
  procedure P (var x : integer);
  begin {P}
    x := x + 1; {изменение переданной процедуре пере-
менной через
                  параметр x}
  end;  {P}
begin
  t := 1;
  P(t);
  P(t);
  P(t);
  writeln(t);
end.
```

Здесь изменение `t` происходит после *явной передачи* этой переменной процедуре P через параметр. Побочный эффект устранен, так как изменение внешней переменной происходит явно.

Побочного эффекта следует избегать, к локализации имен стремиться.

Параметры

Параметры нужны для передачи информации подпрограмме при её вызове (входные), а также для передачи результатов после окончания работы подпрограммы (выходные).

При определении подпрограммы формальные параметры указывается в её заголовке в скобках, после каждого параметра (или после списка параметров) указывается после двоеточия их тип. Например:

```
procedure P (x : integer; y : real; z : string);
function F (m, n : integer; var p : byte) : integer;
```

При передаче параметра используются два различных механизма.

При **передаче параметра по значению** вычисляется значение фактического параметра-выражения, и его копия передается формальному параметру. Изменить при этом изнутри подпрограммы через параметр внешние переменные нельзя. Этот способ используется для *входных* параметров.

При **передаче параметра по ссылке** (параметры-переменные) подпрограмма получает не значение .а адрес фактической переменной, через который эта внешняя переменная может быть изменена. Этот способ используется для *выходных* параметров, хотя так можно передать и входную информацию. Для указания того, что параметр передается по ссылке, перед его именем ставится слово `var`.

Следующий небольшой пример демонстрирует разницу между этими двумя способами передачи параметров:

```
program P3;
var
  a, b : integer;
  procedure P (x : integer; var y : integer);
  begin {P}
    inc(x);
    inc(y);
  end;  {P}
begin
  a := 1;
  b := 1;
  writeln(a, ' ', b); {на экране: 1 1}
  P(a, b);
  writeln(a, ' ', b); {на экране: 1 2}
end.
```

Первый оператор `writeln` выведет на экран значения 1 и 1, второй - значения 1 и 2, так как переменная `a` передавалась процедуре `P` по значению (её значение было скопировано в `x`, но обратной связи нет), а вторая переменная `b` передавалась по ссылке (значит, все изменения `y` в процедуре фактически были произведены с `b`).

Таким образом, для определения выходных параметров нужно указывать слово `var`.

Структурная организации программы

В соответствии с *правилами процедурного программирования* общую структуру программы следует формировать так, чтобы каждая логически за-

вершенная часть общей задачи решалась в виде отдельной подпрограммы. Таким образом, программа будет представлять из себя совокупность отдельных подпрограмм, решающих части задачи. Следует также продумать, как подпрограммы будут обмениваться данными (параметры), следовать принципам локализации данных и исключения побочного эффекта.

Особо следует указать на необходимость *документирования* программы. Современные подходы в программировании предполагают, что к разработке программы будет иметь отношение не один человек-программист, а целый коллектив. Таким образом необходимо давать исчерпывающие комментарии для улучшения читабельности программы и быстрого понимания её логики.

Для этого перед определением каждой подпрограммы помещают её **спецификацию** - комментарий с описанием того, для чего предназначена эта подпрограмма, как её вызывать, какие входные и выходные параметры используются. Такая спецификация позволяет другим программистам, глубоко не вникая в особенности реализации, использовать подпрограммы в своих целях.

Одной из классических тактик разработки программы является "**нисходящее программирование**" ("сверху – вниз"). Оно предполагает последовательную разработку частей и объектов программы от общих к более детальным. Программа создается не как линейный текст, а как структура, причем элементы этой структуры разрабатываются по правилу "от общего к частному".

Пример 8.1.

Программа решает следующую задачу: *Даны три целых числа a, b и c. Найти среди них минимальное значение.*

В приведенной программе используется две функции: функция `Min(x, y)` возвращает минимальное значение среди своих двух аргументов `x` и `y`; функция `Min3(x, y, z)` возвращает минимальное значение среди своих трёх аргументов `x`, `y` и `z`. При этом для нахождения минимума из трёх функция `Min3` дважды вызывает функцию `Min`.

```
program Example8_1;

{-------------------------------------------------
Функция Min возвращает наименьшее целое значение
из своих двух аргументов x и y
-------------------------------------------------}
function Min(x, y : integer) : integer;
begin {Min}
  if x < y then
    Min := x
  else
    Min := y;
end; {Min}

{-------------------------------------------------
Функция Min3 возвращает наименьшее целое значение
из своих трёх аргументов x, y и z
-------------------------------------------------}
function Min3(x, y, z : integer) : integer;
begin {Min3}
  Min3 := Min(Min(x , y), z);
end; {Min3}

var
  a, b, c : integer;
begin
  writeln ('Введите три целых числа');
  readln(a, b, c);

  writeln('Минимальное из чисел ', a, ',', b, ',', c, '
равно');
  writeln(Min3(a, b, c));
end.
```

Пример 8.2.

Программа решает следующую задачу: *Даны три действительных числа a, b и c. Если треугольник со сторонами равными a, b и c существует, то найти и вывести его углы в градусах.*

Для нахождения косинуса угла α треугольника со сторонами a, b и c, лежащего напротив стороны a можно воспользоваться теоремой косинусов:

$$a^2 = b^2 + c^2 - 2bc\cos\alpha$$

откуда

$$\cos\alpha = \frac{b^2 + c^2 - a^2}{2bc}$$

Для перевода значения угла из радианов в градусы можно воспользоваться формулой:

$$\alpha_{\text{градусы}} = \frac{\alpha_{\text{радианы}}}{\pi} \cdot 180$$

```
program Example8_2;
var
  a, b, c : real;
  alpha, beta, gamma : real;
{-------------------------------------------------
Функция CorrectData возвращает значение true, если
существует треугольник со сторонами a, b и c, и
false - в противном случае.
-------------------------------------------------}
function CorrectData (a, b, c : real) : boolean;
begin {CorrectData}
  CorrectData := (a < b + c) and (b < a + c) and (c < a
+ b);
end;  {CorrectData}

{-------------------------------------------------
Функция Angle возвращает значение угла (в градусах),
лежащего напротив стороны a в треугольнике со
сторонами  a, b и c.
```

```
function Angle (a, b, c : real) : real;
var
  CosAngle, AngleRad : real;
begin {Angle}
  CosAngle := (sqr(b)+sqr(c)-sqr(a)) / (2 * b * c);
  AngleRad := arccos(CosAngle);
  Angle := AngleRad * 180 / pi;
end; {Angle}

begin
  writeln('Введите три стороны треульника');
  readln(a, b, c);
  if CorrectData(a, b, c) then
  begin
    alpha := Angle(a, b, c);
    writeln('Угол напротив стороны ', a:6:2, ' равен ',
alpha:6:2, ' градусов');

    beta  := Angle(b, a, c);
    writeln('Угол напротив стороны ', b:6:2, ' равен ',
beta:6:2, ' градусов');

    gamma := Angle(c, a, b);
    writeln('Угол напротив стороны ', c:6:2, ' равен ',
gamma:6:2, ' градусов');
  end
  else
    writeln('Треугольника с такими сторонами не
существует');
end.
```

Примечание. Если в используемой системе программирования Pascal отсутствует функция `arccos(x)`, то можно воспользоваться стандартной функцией `arctan(x)` и соотношением для перевода

$$\arccos x = 2arctan\sqrt{\frac{1-x}{1+x}}$$

Пример 8.3.

Программа решает следующую задачу: *Даны три трехмерных вектора $\vec{a}$, $\vec{b}$ и $\vec{c}$, заданные своими координатами. Найти сумму их скалярных произведений:*

$$SP = \vec{a} \cdot \vec{b} + \vec{b} \cdot \vec{c} + \vec{a} \cdot \vec{c}$$

```
program Example8_3;
type
  Vector = array [1..3] of real; {Тип "трехмерный вектор"}
var
  a, b, c : Vector;
  SP : real;

{-----------------------------------------
Процедура ReadVector вводит с клавиатуры
координаты трехмерного вектора x
-----------------------------------------}
procedure  ReadVector(var x : Vector);
begin {ReadVector}
  readln(x[1],x[2],x[3]);
end; {ReadVector}

{-----------------------------------------
Функция ScalarProduct возвращает
скалярное произведение векторов x и y
-----------------------------------------}
function ScalarProduct(x, y : Vector) : real;
var
  i :integer;
  s :real;
begin {ScalarProduct}
  s:=0;
  for i:=1 to 3 do
    s := s + x[i]*y[i];
  ScalarProduct := s;
```

```
end; {ScalarProduct}

begin
  {Ввод}
  writeln('Введите координаты вектора a');
  ReadVector(a);
  writeln('Введите координаты вектора b');
  ReadVector(b);
  writeln('Введите координаты вектора c');
  ReadVector(c);

  {Вычисление}
  SP := ScalarProduct(a, b) + ScalarProduct(b, c) +
ScalarProduct(a, c);

  {Вывод}
  writeln('Сумма скалярных произведений векторов равна
', SP:6:2);
end.
```

Задача 8.1 (общая)

Написать и отладить программу PROG8_1, решающую следующую задачу: *Даны две обыкновенные дроби* $\frac{a_1}{a_2}$ *и* $\frac{b_1}{b_2}$ *(*a_1, a_2, b_1, b_2*– целые числа, при этом* a_2, b_2 *не равны нулю). Найти обыкновенную несократимую дробь* $\frac{c_1}{c_2}$*, равную сумме двух данных.*

Например, $\frac{3}{8} + \frac{3}{12} = \frac{15}{24} = \frac{5}{8}$.

Напомним, что знаменатель суммы равен наименьшему общему кратному знаменателей слагаемых

$$c_2 = \text{НОК}(a_2, b_2)$$

а числитель суммы равен

$$c_1 = \frac{a_1 \cdot c_2}{a_2} + \frac{b_1 \cdot c_2}{b_2}$$

после чего дробь необходимо сократить, поделив числитель и знаменатель на их наибольший общий делитель $\text{НОД}(c_1, c_2)$.

НОД можно найти с помощью алгоритма Евклида. Для нахождения наименьшего общего кратного НОК можно воспользоваться соотношением

$$m \cdot n = \text{НОД}(m, n) \cdot \text{НОК}(m, n)$$

Программу будем разрабатывать "сверху – вниз" (методом нисходящего программирования).

Ход работы.

1. Создайте новый файл, сохраните его в вашей персональной папке под именем prog81.pas.
2. Внимание! В отличие от предыдущих лабораторных работ эта программа будет разрабатываться не как линейный текст, а последовательно, от одной подпрограммы к другой. Поэтому ниже будут представлены фрагменты программы в том порядке, в котором они *разрабатывались в соответствии с принципами нисходящего программирования*. Место каждого фрагмента в окончательном тексте программы определяется по правилу "вначале - определение, потом - вызов".
3. Наберите в окне файла *начало* программы. Здесь определен тип `Fraction`, представляющий собой обыкновенную дробь. Он реализован в виде записи (`record`), содержащей два целочисленных поля - `num` (числитель) и `den` (знаменатель).

```
program PROG8_1;
type
  Fraction = record
               num, den : integer
             end;
var
  a, b, c : Fraction;
```

4. Наберите *концовку* программы (тело). В этой части будут произведены вызовы процедур, решающих части соответствующие части

задачи: ввод двух данных дробей, нахождение их суммы, сокращение суммы, вывод результата. Определения этих процедур отложены на следующие этапы разработки программы.

```
begin
  writeln('Введите первую дробь:');
  ReadF(a);
  writeln('Введите вторую дробь:');
  ReadF(b);

  writeln('Исходные дроби:');
  WriteF(a);
  WriteF(b);

  SumF(a, b, c);
  Reduction(c);

  writeln('Сумма дробей =');
  WriteF(c);
end.
```

5. Реализуем процедуру ввода дроби с клавиатуры `ReadF`.

```
{Процедура ReadF вводит дробь x с клавиатуры}
procedure ReadF(var x : Fraction);
begin {ReadF}
  readln(x.num, x.den);
end;  {ReadF}
```

6. Реализуем процедуру вывода дроби на экран `WriteF`

```
{Процедура WriteF  выводит дробь x на экран}
procedure WriteF(x : Fraction);
begin {WriteF}
  writeln(x.num, '/',x.den);
end;  {WriteF}
```

7. Реализуем процедуру `SumF`, которая находит сумму дробей `x` и `y`, возвращая результат через параметр `z`. В процессе написания

этой процедуры появилась необходимость найти НОК двух чисел – это будет делать функция NOK, определение которой отложено на потом.

```
{Процедура SumF суммирует дроби x и y, результат - в z}
procedure SumF(x, y : Fraction; var z : Fraction);
begin {SumF}
  z.den := NOK(x.den, y.den);
  z.num := x.num * z.den div x.den +
           y.num * z.den div y.den;
end;  {SumF}
```

8. Реализуем функцию NOK, возвращающую наименьшее общее кратное двух целых чисел. Формула для него выводится из соотношения, приведенного выше. При определении функции возникла необходимость в нахождении НОД - это будет делать функция NOD.

```
{Функция NOK возвращает НОК двух целых чисел t1 и t2}
function NOK(t1, t2 : integer):integer;
begin {NOK}
  NOK := t1 * t2 div NOD (t1, t2);
end;  {NOK}
```

9. Реализуем функцию NOD, возвращающую наибольший общий делитель двух целых чисел. Используем алгоритм Евклида.

```
{Функция NOD возвращает НОД двух целых чисел}
function NOD(t1, t2 : integer):integer;
begin {NOD}
  while t1 <> t2 do
    if t1 > t2 then
      t1 := t1 - t2
    else
      t2 := t2 - t1;
  NOD := t1;
end;  {NOD}
```

10. Осталось реализовать процедуру `Reduction`, которая сокращает обыкновенную дробь. В процессе сокращения понадобилось найти НОД двух чисел, для которого ранее уже была написана функция `NOD`.

```
{Процедура Reduction сокращает дробь x}
procedure Reduction(var x : Fraction);
var
  t : integer;
begin {Reduction}
  t := NOD(x.num, x.den);
  x.num := x.num div t;
  x.den := x.den div t;
end;  {Reduction}
```

11. Теперь все вызываемые подпрограммы реализованы. Нужно правильно расставить их в тексте программы, учитывая, что определение должно предшествовать вызову. В этом примере подпрограммы можно расставить в следующей последовательности: `ReadF`, `WriteF`, `NOD`, `NOK`, `SumF`, `Reduction`, тело программы.

12. Отладьте программу на системе тестов:

№ теста	Вход		Выход
	первая дробь	вторая дробь	
1.	3 8	3 12	5/8
2.	50 100	210 35	13/2
3.	2 4	0 1	1/2

Задача 8.2 (самостоятельная)

Написать и отладить программу PROG8_2, решающую следующую задачу. *Даны три целых положительных числа a, b и c. Выбрать из них то число, сумма делителей которого максимальна.*

Рекомендация. Написать функцию `SumOfDevisors(x : integer):integer`, возвращающую сумму делителей целого числа `x`.

Задача 8.3 (индивидуальная)

Написать и отладить программу PROG8_3 в соответствии с вариантом. В программе необходимо использовать процедурный подход. По возможности, разработку программы производить методом нисходящего программирования.

Вариант 01. Даны три целых положительных числа a, b и c. Выбрать из них то число, сумма цифр которого максимальна.

Вариант 02. Даны три целых положительных числа a, b и c. Для каждого числа определить максимальную цифру и выбрать то число из трех, в котором она минимальна.

Вариант 03. Даны три целых положительных числа a, b и c. Выбрать такие два числа из трех, чтобы НОК этой пары был бы наибольшим.

Вариант 04. Даны три целых положительных числа a, b и c. Определить, какие из этих чисел являются полными квадратами.

Вариант 05. Даны три целых положительных числа a, b и c. Найти в каждом числе произведение минимальной и максимальной цифр.

Вариант 06. Даны три трехмерных вектора $\vec{a}$, $\vec{b}$ и $\vec{c}$, заданные своими координатами, вещественное число k. Найти вектор, равный $k\vec{a} + (1-k)\vec{b} + k^2\vec{c}$. Указание: реализовать функции умножения скаляра на вектор, суммы векторов.

Вариант 07. Даны три трехмерных вектора $\vec{a}$, $\vec{b}$ и $\vec{c}$, заданные своими координатами, вещественное число k. Найти вектор, равный $k(\vec{a} + \vec{b}) + (1-k)(\vec{a} + \vec{c})$. Указание: реализовать функции умножения скаляра на вектор, суммы и разности векторов.

Вариант 08. Даны три трехмерных вектора $\vec{a}$, $\vec{b}$ и $\vec{c}$, заданные своими координатами. Найти вектор, равный $(\vec{a} + \vec{b}) + (2\vec{a} + 3\vec{c})$. Указание: реализовать функции умножения скаляра на вектор, суммы векторов.

Вариант 09. Даны три трехмерных вектора $\vec{a}$, $\vec{b}$ и $\vec{c}$, заданные своими координатами. Найти число, равное $|\vec{a}+\vec{b}|+|\vec{b}+\vec{c}|$. Указание: реализовать функции суммы векторов, длины вектора. Длина вектора $\vec{a}$ равна $|\vec{a}| = \sqrt{a_1^2+a_2^2+a_3^2}$.

Вариант 10. Даны три трехмерных вектора $\vec{a}$, $\vec{b}$ и $\vec{c}$, заданные своими координатами. Найти вектор с наибольшей длиной. Указание: реализовать функцию, возвращающую длину вектора. Длина вектора $\vec{a}$ равна $|\vec{a}| = \sqrt{a_1^2+a_2^2+a_3^2}$.

Вариант 11. Даны два комплексных числа a и b ($a = a_{re} + a_{im}\,i$; $b = b_{re} + b_{im}\,i$). Найти сумму и разность данных чисел $a+b = (a_{re}+b_{re}) + (a_{im}+b_{im})i$, $a-b = (a_{re}-b_{re}) + (a_{im}-b_{im})i$. Указание: реализовать подпрограммы ввода, вывода комплексного числа, суммы, разности комплексных чисел.

Вариант 12. Даны два комплексных числа a и b ($a = a_{re} + a_{im}\,i$; $b = b_{re} + b_{im}\,i$). Найти произведение данных чисел $a \cdot b = (a_{re}\,b_{re} - a_{im}\,b_{im}) + (a_{re}\,b_{im} + a_{im}\,b_{re})i$. Указание: реализовать подпрограммы ввода, вывода комплексного числа, произведения комплексных чисел.

Вариант 13. Даны два комплексных числа a и b ($a = a_{re} + a_{im}\,i$; $b = b_{re} + b_{im}\,i$). Найти из них наибольшее и наименьшее по модулю комплексное число. Модуль комплексного числа равен $|a| = \sqrt{a_{re}^2 + a_{im}^2}$. Указание: реализовать подпрограммы ввода, вывода комплексного числа, модуля комплексного числа.

Вариант 14. Даны вещественные квадратные матрицы A и B порядка n. Найти матрицу, равную AB+BA. Указание: реализовать подпрограммы ввода, вывода матрицы, суммы и произведения матриц.

Вариант 15. Даны три строки символов a, b и c, состоящие только из цифр . Считая, что каждая строка представляет собой целое число, определить, какое из трех чисел: a·b, b·c или a·c меньше двух других. Указание: длины строк могут быть таковы, что значение не войдет ни в один целочисленный тип данных. Реализовать подпрограммы ввода, выводы строки-числа, произведения двух строк-чисел, сравнения двух строк-чисел.

Контрольные вопросы

1. Что такое подпрограмма?
2. Что такое вызов подпрограммы?
3. В чем особенности и преимущества процедурного программирования?
4. Что "делает" вызвавшая программа, пока исполняется вызванная ею подпрограмма?
5. В чем отличие процедуры от функции?
6. Что такое локальные данные? В чем отличие их от глобальных?
7. Какова область видимости локальных данных?
8. Если в подпрограмме (внешней) определена другая (вложенная) подпрограмма, то распространяется ли на вложенную подпрограмму область видимости локальных данных внешней подпрограммы?
9. Как разрешается конфликт одноименных данных?
10. Что такое параметры подпрограммы? Для чего они используются? В чем разница между формальными и фактическими параметрами?
11. Какова разница между механизмами передачи параметров по значению и по ссылке?
12. Что такое "побочный эффект" и почему его следует избегать?
13. Для чего нужно документировать (комментировать) программу и подпрограммы?
14. Что такое нисходящее программирование? В чем его преимущества?
15. *Как можно успешно компилировать программу, если в соответствии с тактикой нисходящего программирования вызов процедуры уже есть, а реализация еще не написана?

Список литературы

1. Павловская Т.А. Паскаль. Программирование на языке высокого уровня: Учебник для вузов.– Спб.: Питер, 2010. – 400с .
2. Анисимов А.Е. Сборник задач по началам программирования. – Ижевск, Из-во УдГУ, 2004. – 149 с.
3. Анисимов А.Е., Пупышев В. В.. Сборник заданий по основаниям программирования. – М., ИНТУИТ, 2005. – 352 с.

Учебное издание

Анисимов Андрей Евгеньевич

ПРАКТИКУМ ПО ОСНОВАМ ПРОГРАММИРОВАНИЯ

Учебно-методическое пособие

Авторская редакция

Подписано в печать 00.00.00. Формат 60x84 1/16.
Усл. печ. л. 0,00. Уч.-изд. л. 0,00.
Тираж 00 экз. Заказ № 0000.

Издательство «Удмуртский университет»
426034, Ижевск, Университетская, д. 1, корп. 4, каб. 207
Тел./факс: + 7 (3412) 500-295 E-mail: editorial@udsu.ru