БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Г. А. Расолько Е. В. Кремень Ю. А. Кремень

ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ

В трех частях

Часть 1
ОСНОВНЫЕ ПРИЕМЫ ПРОГРАММИРОВАНИЯ НА PASCAL

Рекомендовано
Учебно-методическим объединением
по естественно-научному образованию
в качестве учебно-методического пособия
для студентов учреждений высшего образования,
обучающихся по специальности «математика»

Учебное электронное издание

Минск, БГУ, 2025

ISBN 978-985-881-746-6 (ч. 1)
ISBN 978-985-881-745-9

УДК 004.42(075.8)
ББК 32.973-018я73-1

Р е ц е н з е н т ы:
кафедра инженерной психологии и эргономики
Белорусского государственного университета
информатики и радиоэлектроники
(заведующий кафедрой доктор психологических наук,
член-корреспондент Международной академии
психологических наук, профессор *Т. В. Казак*);
доктор физико-математических наук, доцент *А. С. Кравчук*

Расолько, Г. А. Практикум по программированию [Электронный ресурс] : учеб.-метод. пособие. В 3 ч. Ч. 1. Основные приемы программирования на Pascal / Г. А. Расолько, Е. В. Кремень, Ю. А. Кремень. – Минск : БГУ, 2025. – 1 электрон. опт. диск (CD-ROM). – ISBN 978-985-881-746-6.

Представлен учебный материал для проведения занятий по дисциплине «Практикум по программированию». Кратко изложена теория и рассмотрены задачи по темам: арифметика ЭВМ, структурная методология разработки программ, средства алгоритмических языков, простые и структурированные данные и работа с ними, базовые операторы языка и методы программирования, механизмы структурирования программ.

Предназначено для студентов учреждений высшего образования, обучающихся по специальности «математика».

Минимальные системные требования:
PC, Pentium 4 или выше;
RAM 1 Гб; Windows XP/7/10; Adobe Acrobat.

Оригинал-макет подготовлен в программе MicroSoft Word.

Ответственный за выпуск *Т. М. Турчиняк*.
Дизайн обложки *В. П. Явуз*. Технический редактор *В. П. Явуз*.
Компьютерная верстка *В. П. Явуз*.
Корректор *Е. О. Алёшина*.

Подписано к использованию 28.01.2025. Объем 1,75 МБ.

Белорусский государственный университет.
Управление редакционно-издательской работы.
Пр. Независимости, 4, 220030, Минск.
e-mail: urir@bsu.by
http://elib.bsu.by/

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящий практикум направлен на формирование у студентов навыков решения различных типов задач с использованием современных информационных технологий. Основная цель данной дисциплины – развитие алгоритмического мышления, изучение современных методов программирования и приобретение навыков работы с современными вычислительными средствами.

В процессе занятий студенты должны:

- развивать математическое, логико-алгоритмическое и программистское мышление;
- применять практические знания и умения, современные методы и системы программирования;
- осваивать приемы и основы методологии структурного и модульного программирования;
- формировать творческий подход к конструированию алгоритмов, что способствует развитию аналитических способностей.

Основное внимание уделяется не только кодированию программ, но и их проектированию, где особый акцент делается на современных технологиях: проектирование сверху вниз, модульное программирование с использованием подпрограмм и модулей, анализ эффективности и оптимизация программ, широкое использование рекурсии. Все это направлено на привитие определенного стиля программирования и дальнейшее развитие навыков посредством обучения объектно ориентированному программированию.

Первая часть учебно-методического пособия посвящена основам арифметики ЭВМ, структурной методологии разработки программ и средствам алгоритмических языков. Она направлена на формирование фундаментальных знаний, которые являются основой для успешного освоения более сложных концепций программирования.

В результате изучения первой части практикума обучающиеся должны знать:

- системы счисления, включая перевод чисел между различными системами счисления;
- формы представления данных в персональном компьютере и принципы их хранения;
- структурную методологию разработки программ, включая блок-схемы и структурограммы;

- средства алгоритмических языков программирования и методы разработки программ;
- простые и структурированные данные и работу с ними.

Эти знания помогут студентам не только понять, как работают компьютеры на уровне арифметики и логики, но и освоить базовые принципы проектирования и разработки программ.

Тема 1
АРИФМЕТИКА ЭВМ
(8 часов. Тестирование)

1.1. Системы счисления

Под *системой счисления* будем понимать способ представления всякого числа при помощи некоторого алфавита символов, которые называются *цифрами*.

Система счисления (с/с) называется *позиционной*, когда одна и та же цифра имеет разное значение, соответствующее позиции цифры в последовательности цифр, образующих число.

Количество S разных цифр, используемых в позиционной системе счисления, называется ее *основанием*. Цифры, используемые в системе счисления для записи чисел, называются *базисными числами*. Эти цифры обозначают S целых чисел, обычно такого ряда: 0, 1, 2, 3, ..., (S – 1). В системах счисления, где $S > 10$, цифры больше 9 обозначаются буквами.

Система счисления	Основание систем счисления	Базовые числа
Десятичная	$S = 10$	0, 1, 2, 3, …, 9
Двоичная	$S = 2$	0, 1
Шестнадцатеричная	$S = 16$	0, 1, …, 9, A, B, C, D, E, F

Задание. Распишите последовательность первых 30 натуральных чисел в разных системах счисления.

В общем случае в позиционной системе с основанием S любое число x может быть представлено в виде выражения, зависящего от основания S:

$$x = \varepsilon_r S^r + \varepsilon_{r-1} S^{r-1} + ... + \varepsilon_1 S^1 + \varepsilon_0 S^0 + \varepsilon_{-1} S^{-1} + ... + \varepsilon_{-t} S^{-t} + ..., \qquad (1)$$

где в качества коэффициентов ε_i могут быть любые из S цифр, используемых в данной системе счисления.

Вместо вида (1) принято писать короче:

$$x = \varepsilon_r \varepsilon_{r-1} ... \varepsilon_1 \varepsilon_0 . \varepsilon_{-1} ... \varepsilon_{-t} .... \qquad (2)$$

Иногда представление (2) записывают в виде

$$x = \overline{\varepsilon_r \varepsilon_{r-1} ... \varepsilon_1 \varepsilon_0 . \varepsilon_{-1} ... \varepsilon_{-t} ....}$$

Позиции цифр в числе, которые отсчитываются от разделителя целой и дробной части, называются *разрядами*. Разделителем целой части и дробной в математике служит запятая, а в программировании – точка.

В позиционной системе счисления значение каждого разряда больше, чем значение соседнего справа разряда, в число раз, равное основанию S системы.

В 2 с/с ($S = 2$) используют цифры 0, 1; тогда любое число, записанное в виде (2), можно записать в форме (1) так:

$$x = \alpha_m \cdot 2^m + \alpha_{m-1} \cdot 2^{m-1} + \ldots + \alpha_1 \cdot 2^1 + \alpha_0 \cdot 2^0 + \alpha_{-1} \cdot 2^{-1} + \alpha_{-2} \cdot 2^{-2} + \ldots,$$

где $\alpha_j = \{0,1\}$, $j = 0, \pm 1, \ldots$.

Таблицы сложения, вычитания и умножения в двоичной системе счисления

0 + 0 = 0	0 – 0 = 0	0 · 0 = 0
0 + 1 = 1	1 – 0 = 1	0 · 1 = 0
1 + 0 = 1	1 – 1 = 0	1 · 0 = 0
1 + 1 = 10	10 – 1 = 1	1 · 1 = 1

Задачи

1. Как по записи числа в двоичной системе счисления определить, четное оно или нет?

2. Выполните указанные арифметические действия над числами в двоичной системе счисления:

а) 11001 + 1101;
б) 11, 000011 + 11,111101;
в) 111, 11 + 1111,1;
г) 1000 – 11;
д) 100,001 – 1, 111;
е) 111,010111 – 101,1111;
ж) 1101 · 1100;
з) 1101,001 · 1011,01;
и) 0,111001 · 101,11.

3. Переведите следующие числа из двоичной системы счисления в десятичную (дробную часть получите с четырьмя знаками):

а) 110010110110011;
б) 1111,011010111;
в) 100,111111;
г) 101110,111000001;
д) 1000110,001111001111;
е) 110001,11110001111.

4. Переведите следующие числа из шестнадцатеричной системы счисления в десятичную:

а) 75047;
б) BC0,1979;
в) 35C,D5;
г) 0,0A5;
д) 0,4A;
е) 4680.

5. Докажите, что числа 144_q и 169_s являются полными квадратами, а число 1331_p – полным кубом в любых системах счисления для $q > 4$, $s > 9$, $p > 3$.

6. Сколько четырехзначных натуральных чисел можно записать, если основание системы счисления равно 5? Равно n?

7. Сколько пятизначных натуральных чисел, цифры которых размещены по возрастанию, можно записать, если основание системы счисления равно 8? Равно n?

8. Определите признак делимости на 15 числа, записанного в шестнадцатеричной системе счисления.

9. Даны числа, записанные в разных системах счисления. Найдите для каждого из них предыдущее и следующее числа:

2122_3, 777_8, 1001_2, 1101_2, 30133_4, 44444_5, 8012788_9.

10. Распределите числа в порядке возрастания их значений:

а) 1011010_2, 101101_3, 10110_4, 1011_5, 101_6, 10_7;

б) 1121_3, 717_8, 1001_2, 1101_2, 30123_4, 44444_5;

в) 1022021_3, 102202_4, 10220_5, 1022_6, 102_7, 10_8.

11. Расположите числа в порядке уменьшения их значений:

а) 3311_9, 1601_7, 2101_6;

б) 12312_4, 1019_{11}, 2435_8;

в) 5318_9, $1B91_{12}$, 5204_6.

12. Найдите результаты для следующих выражений:

а) $AA_{13} \cdot 1B2_{13} + 99_{11} \cdot 192_{11}$;

б) $BA_{12} \cdot 1C2_{13} + 99_{11} \cdot 179_{11}$.

13. Определите основание системы счисления исходя из равенства:

а) $202_x - 121_x = 22_{4x} - (x_{10} + 1)^2$;

б) $121_x + B_{16} = (x_{10} + 10_2)^2$;

в) $323_{2x} \cdot 4_{5x} - 135_{3x-2} = 513_{3x}$;

г) $20_x\,(x_{10} + 10_2 \cdot 57_x) - 510_x = 2430_x$;

д) $xx_{2x} = 13_{0,5x} \cdot 22_x + 22_{0,5x}$;

е) $20_x\,(x_{10} + 10_2 \cdot 57_x) - 510_x = 2430_x$.

1.2. Переводы чисел из двоичной системы счисления в шестнадцатеричную или восьмеричную системы счисления и обратно

Общий метод перевода небольших чисел из одной системы счисления в другую такой: *представить число в виде* (1) *и выполнить действия в новой системе счисления.*

Так как основание восьмеричной и шестнадцатеричной систем счисления является целой степенью числа 2: $8 = 2^3$, $16 = 2^4$, то правила достаточно простые.

Представление чисел разных систем счисления дается в таблице ниже.

Система счисления			Пример перевода из двоичной в шестнадцатеричную с/с	Пример перевода из двоичной в восьмеричную с/с
десятичная	шестнадцатеричная	двоичная		
0	0	0	0000	000
1	1	1	0001	001
2	2	10	0010	010
3	3	11	0011	011
4	4	100	0100	100
5	5	101	0101	101
6	6	110	0110	110
7	7	111	0111	111
8	8	1000	1000	
9	9	1001	1001	
10	A	1010	1010	
11	B	1011	1011	
12	C	1100	1100	
13	D	1101	1101	
14	E	1110	1110	
15	F	1111	1111	
16	10	10000		
<...>	<...>	<...>		

Правило 1. Чтобы перевести число из шестнадцатеричной системы счисления в двоичную, нужно любую цифру расписать в двоичной системе счисления по четыре разряда (из восьмеричной системы счисления в двоичную – по три разряда; из четверичной системы счисления в двоичную – по два разряда), а потом отбросить незначащие нули.

Правило 2. Чтобы перевести число из двоичной системы счисления в систему счисления с основанием 2^k, $k = 3,4$, нужно от запятой (точки) налево и направо выделить группы по k цифр (дополняя при необходимости незначащими нулями крайние группы) и каждой группе поставить в соответствие определенную цифру новой системы.

Задачи

1. Переведите следующие числа из двоичной системы счисления в шестнадцатеричную:

а) 10001100110101101;
б) 0,10011111001110001;
в) 1010,101011011111;
г) 11110001100001101;
д) 1110000,110001111101.

2. Переведите следующие числа из двоичной системы счисления в восьмеричную:

а) 10001100110101101;
б) 0,10011111001110001;
в) 1010,101011011111;
г) 11110001100001101;
д) 1110000,110001111101.

3. Переведите следующие числа из двоичной системы счисления в четверичную:

а) 10001100110101101;
б) 0,10011111001110001;
в) 1010,101011011111;
г) 11110001100001101;
д) 1110000,110001111101.

4. Переведите следующие числа из восьмеричной системы счисления в шестнадцатеричную:

а) 573217;
б) 0,27341;
в) 73526,0125.

5. Предложите способ перевода числа, записанного в троичной системе счисления, в девятеричную, минуя десятичную систему счисления.

6. Переведите в секунды в двоичной системе счисления:
а) 3 часа, 18 минут, 12 секунд;
б) 24 часа, 15 минут.

1.3. Перевод вещественных чисел из системы счисления с основанием *«p»* в систему счисления с основанием *«q»*

Перевод целых положительных чисел из системы счисления с основанием *«p»* в систему счисления с основанием *«q»*

Пусть x_p – целое положительное число в системе счисления с основанием «*p*». В соответствии с формулой (1) в системе счисления с основанием «*q*», число x_p может быть представлено в виде

$$x_p = a_n \cdot q^n + a_{n-1} \cdot q^{n-1} + ... + a_0 \equiv$$
$$\equiv (...(a_n \cdot q + a_{n-1}) \cdot q + ... + a_1) \cdot q + a_0. \qquad (3)$$

Отсюда для нахождения коэффициентов при степенях q в формуле (3) получаем следующее правило перевода.

Правило 3. Чтобы перевести целое число из системы счисления с основанием *«p»* в систему счисления с основанием *«q»*, нужно выполнить следующие действия. В системе счисления с исходным основанием *«p»* разделить x на q. Первый остаток – младшая цифра (a_0). Затем полученное частное необходимо разделить на q. Остаток есть следующая цифра (a_1). Новое частное делим на q. Этот процесс продолжаем до тех пор, пока не получим частное, меньшее чем q. Это частное есть старшая цифра (a_n). Все действия выполняются в системе с основанием *«p»*. Полученные остатки записываем в новой системе счисления с основанием «q».

Перевод правильных дробей из системы счисления с основанием *«p»* в систему счисления с основанием *«q»*

Пусть y_p – правильная дробь в системе счисления с основанием *«p»*. Аналогично предыдущему имеем

$$y_p = a_{-1} \cdot q^{-1} + a_{-2} \cdot q^{-2} + \ldots + a_{-m} \cdot q^{-m} \equiv$$
$$\equiv q^{-1}\left(a_{-1} + q^{-1}\left(a_{-2} + q^{-1}\ldots\right)\right)$$

Правило 4. Чтобы перевести правильную дробь из системы счисления с основанием *«p»* в систему счисления с основанием *«q»*, нужно исходное число умножить на q (в системе счисления с исходным основанием *«p»*). Целая часть полученного числа есть цифра a_{-1}. Полученную дробную часть умножить на q. Целая часть полученного числа есть цифра a_{-2} и т. д. Все действия выполняются в системе с основанием *«p»*. В конце перевода нужно записать полученные цифры $(a_{-1}, a_{-2}, \ldots)_p$ в соответствующие цифры в новой системе счисления с основанием *«q»*.

Процесс перевода дробной части необходимо остановить при достижении одного из следующих условий:

1) когда получили произведение, равное нулю (число перевелось точно);
2) когда получили произведение, которое уже встречалось раньше, значит, нашли период;
3) когда получили нужное количество цифр в числе, которое переводится.

Перевод смешанных дробей

Для перевода целой части числа применяется правило 3, для перевода дробной части числа – правило 4.

Задачи

1. Переведите следующие числа из десятичной системы счисления в двоичную, восьмеричную и шестнадцатеричную:

а) 149,0112;
б) 854,0328;
в) 351,051;
г) 0,0805;
д) 745,2702.

2. Переведите следующие числа из десятичной системы счисления в двоичную, троичную, четверичную, пятеричную, шестеричную, семеричную, восьмеричную и девятеричную:

а) 0,65;
б) 46,75;
в) 235,25;
г) 271,1;
д) 17,4;
е) 101,2.

1.4. Формы представления данных

Числа можно представлять в форме с фиксированной точкой (2) и в форме с плавающей точкой. Представление числа с плавающей точкой в общем случае имеет вид

$$x = q \cdot s^p, \tag{4}$$

где q – мантисса числа; s^p – характеристика числа x (s – основание характеристики; p – порядок).

Мантисса – дробь с фиксированной точкой со знаком, *порядок* – целое число со знаком. Порядок p определяет положение точки в числе x.

Чтобы получить число в форме с плавающей точкой, нужно перевести число в нужную систему счисления, а потом записать его по правилу 4:

$$\left(a_{n-1}...a_1a_0.a_{-1}a_{-1}...a_{-m}\right)_p = 0.a_{n-1}...a_{-m} \times \underbrace{p^n}_{=10_p^n} = 0.a_{n-1}...a_{-m} \cdot 10_p^n.$$

Формы представления чисел в персональном компьютере

Информация в памяти ЭВМ хранится и обрабатывается в двоичной системе счисления. Неделимой наименьшей единицей хранения информации является один *бит*, т. е. двоичный разряд, который может иметь значение 0 или 1. Группа из 8 бит называется *байтом*. Вся оперативная память состоит из байтов, которые нумеруются с нуля. Последовательность в 1024 байта называется 1 килобайтом (1 Кбайт = 2^{10} байтов); 1 мегабайт =

= 1024 Кбайтов (1 Мбайт = 2^{20} байтов); 1 гигабайт = 1024 Мбайтов (1 Гбайт = = 2^{30} байтов); 1 терабайт = 1024 Гбайтов (1 Тбайт = 2^{40} байтов); 1 петабайт = 1024 Тбайт (1 Пбайт = 2^{50} байтов); 1 эксабайт = 1024 Пбайт (1 Эбайт = = 2^{60} байтов) и др.

Адресом любого данного считается адрес (номер) самого первого байта поля памяти, выделенной для его хранения.

Форма записи данных в памяти ЭВМ называется *внутренним представлением данного*.

В ЭВМ применяют две формы представления чисел: с *фиксированной* и с *плавающей точкой*.

Хранение чисел с фиксированной точкой

Для хранения информации в оперативной памяти служит *ячейка*. Существуют ЭВМ, у которых ячейка имеет постоянную длину, в ПК – переменную, наименьшая длина – 1 байт.

Различают ячейки на 2, 4, 6, 8, 10 байт. Ячейка в 2 байта называется *словом*, в 4 байта – *двойным словам*, в 1 байт – *полусловом*. Будем считать, что байты размещаются так (k = 2, 4, 6, 8, 10):

Байт	1	2	3	4	5	6	<...>	k

При представлении чисел с фиксированной точкой местоположение точки условно фиксируется в определенном месте ячейки относительно разрядов числа. Каждый бит имеет свой вес в байте. Когда в байте точка зафиксирована справа и используется беззнаковая форма хранения целых положительных чисел, то биты имеют вес в соответствии со следующей схемой:

Бит	7	6	5	4	3	2	1	0
Вес бита	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Этим методом представляют целые числа.

Используют два варианта представления целых чисел: со знаком и без знака. В первом случае для кода знака выделяют «знаковый» разряд (обычно крайний слева) и хранят там 0 или 1 по схеме: $0 \to +$, $1 \to -$

Бит	7	6	5	4	3	2	1	0
	Знак							
Вес бита		2^6	2^5	2^4	2^3	2^2	2^1	2^0

Хранение целых чисел

Для представления целых чисел в форме с фиксированной точкой со знаком в ЭВМ применяют *прямой*, *обратный* и *дополнительный* коды.

В прямом коде хранится модуль числа, а в старшем разряде стоит знаковый бит: 0 – для положительных чисел, 1 – для отрицательных.

Получение обратного кода отрицательного числа сводится к получению инверсии (замены 0 на 1, а 1 на 0) n – разрядного кода модуля этого числа. Знаковый бит имеет 0 для положительного числа и 1 – для отрицательного.

Для представления отрицательного числа в дополнительном коде число нужно перевести в двоичную систему счисления, дополнить нулями до n разрядов, выполнить инверсию двоичных цифр и к коду добавить число 1.

В компьютере целые положительные числа хранятся в прямом коде, а целые отрицательные – в дополнительном.

Тип данных	Размер в байтах	Диапазон чисел	Способ хранения чисел
`Byte`	1	0..255	Хранятся только неотрицательные числа в прямом коде
`Word`	2	0..65535	
`Shortint`	1	–128..127	Неотрицательные числа хранятся в прямом коде, отрицательные – в дополнительном
`Integer`	2	–32768..32767	
`Longint`	4	–2147483648..2147483647	

Принципы хранения чисел с плавающей точкой

Так как арифметические действия над числами с плавающей точкой требуют, кроме выполнения операции над мантиссой, определенных операций над порядками (сравнения, сложения, вычитания), то для упрощения операций над порядками их сводят к действиям над целыми положительными числами без знака. К порядку p добавляют целое число – *смещение*.

Обычно смещение $A = 2^{k-1}$, где k – число двоичных разрядов, используемых для записи модуля порядка. Тогда $p_{\text{см}} = p + A > 0$ и $p_{\text{см}}$ называют *характеристикой числа*.

При фиксированном числе разрядов мантиссы любое число представляется в ЭВМ с наиболее возможной точностью нормализированным числом. Используют две формы нормализации.

Число $x = q \cdot s^p$ называется *нормализированным*, когда мантисса:

1) правильная дробь: $\frac{1}{s} \le |q| < 1$;

2) $1 \le |q| < s$.

При такой нормализации старшую цифру мантиссы можно не хранить.

В памяти нужно хранить два числа: мантиссу и порядок, а основание системы счисления $p = 2$ известно. Под число отводится ячейка – четное количество байт. Ячейку условно разбивают на две части. В одной размещают «сдвинутый» порядок числа, а во второй – мантиссу (обычно цифры после точки как условно целое). Под знак мантиссы также отводится 1 бит.

Использование «предполагаемого» (спрятанного) старшего разряда мантиссы приводит к необходимости представления чисел с нулевой мантиссой особым кодам, ведь нулевая мантисса не отличается от мантиссы, равной 1/2. Таким кодам взяли код, равный в компьютере всем нулям разрядной сети.

Замечание 1. Данные, хранящиеся в ЭВМ в форме с плавающей точкой, почти всегда представлены с погрешностью и только приблизительно равны исходному числу. Погрешность обусловлена ограничением на длину мантиссы (на количество разрядов для мантиссы).

Замечание 2. Под порядок также отводится ограниченное количество разрядов, значит диапазон чисел с плавающей точкой ограничен: $\min \le |x| \le \max$. Значения min и max также зависят от конкретной ЭВМ.

Форматы чисел с плавающей точкой

Размеры памяти, выделяемой под числа с плавающей точкой, существенно зависят от аппаратной реализации компьютера. Для IBM PC/AT числа с плавающей точкой занимают поле (ячейку) в 4 байта (`Single`), 6 байт (`Real`), 8 байт (`Double`), 10 байт (`Extended`).

Характеристика – это смещенный код порядка с отрицательным нулем. По стандарту смещение порядка равно $\left(2^{k-1}-1\right)$, где k – количество разрядов, отведенных для хранения кода порядка. Значение порядка лежит в интервале $\left[-\left(2^{k-1}-1\right),-0\right] \cup \left[1,\left(2^{k-1}-1\right)\right]$. Для типа `Real` смещение равно $\left(2^{k-1}+1\right)$.

Код $\underbrace{11\ldots1}_{k}$ не используется, так как зарезервирован для указания переполнения порядка или на потерю точности мантиссы.

При этом получаются следующие диапазоны представляемых чисел:

• при 4-байтовом хранении $k = 8$, $p_{см} \in [-127; 127]$, что соответствует диапазону числа $\pm 10^{-38} \div 10^{38}$, представляемого с погрешностью 2^{-23};

• при 6-байтовом хранении $k = 8$, $p_{см} \in [-127; 127]$, что соответствует диапазону числа $\pm 10^{-38} \div 10^{38}$, представляемого с погрешностью 2^{-39};

• при 8-байтовом хранении $k = 11$, $p_{см} \in [-1023; 1023]$, что соответствует диапазону числа $\pm 10^{-308} \div 10^{308}$, представляемого с погрешностью 2^{-52};

• при 10-байтовом хранении $k = 15$, $p_{см} \in [-16383; 16383]$, что соответствует диапазону числа $\pm 10^{-4932} \div 10^{4932}$, представляемого с погрешностью 2^{-64}.

Приведем схему хранения чисел в памяти.

Знак мантиссы

Single	±	Характеристика (*l*)			Нормализованная мантисса (*f*)		
номера битов	31	30	…	23	22	…	0

$\Rightarrow$ число расшифровывается так: $v = (-1)^{\pm} 2^{l-127} (1.f)$, если $0 < l \le 255$.

Знак мантиссы

Real	±	Нормализованная мантисса (*f*)			Характеристика (*l*)		
номера битов	47	46	…	8	7	…	0

$\Rightarrow$ число расшифровывается так: $v = (-1)^{\pm} 2^{l-129} (1.f)$, если $0 < l \le 255$.

Знак мантиссы

Double	±	Характеристика (*l*)			Нормализованная мантисса (*f*)		
номера битов	63	62	…	52	51	…	0

$\Rightarrow$ число расшифровывается так: $v = (-1)^{\pm} \cdot 2^{(l-1023)} \cdot (1.f)$, если $0 < l < 2047$.

Знак мантиссы

Extended	±	Характеристика (*l*)			Нормализованная мантисса (*f*)		
номера битов	79	78	…	64	63	…	0

$\Rightarrow$ число расшифровывается так: $v = (-1)^{\pm} \cdot 2^{(l-16383)} \cdot (0.f)$, если $0 < l < 32\,765$.

Во всех формах представления чисел $v = 0$, при $l = 0$.

Современные вычислительные машины оперируют лишь с конечными дробями. Но результат выполнения арифметической операции над конечными дробями не всегда является конечной дробью. Например, операции

деления, извлечения квадратного корня и т. д. могут приводить к бесконечным дробям. Но даже если результат и будет конечной дробью, то его чаще всего нельзя представить в том виде, который принят в компьютере. Например, результат операции умножения над конечными дробями всегда будет дробью, но число разрядов дроби может превышать отведенное для хранения число базисных элементов.

Задачи

1. Получите диапазоны для целочисленных данных.

2. Известно, что для целого числа, занимающего 8 бит, диапазон представляемых чисел составляет от –128 до 127. Распишите диапазон чисел, если это поле расширяется на: а) 1 бит; б) 2 бита; в) 3 бита; г) 4 бита.

3. Вычислите Byte (1000).

4. Вычислите Byte (–1000).

5. Вычислите Shortint (2023).

6. Вычислите Shortint (–2023).

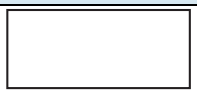
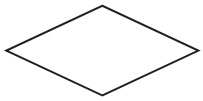
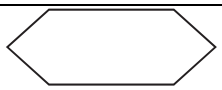
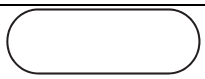
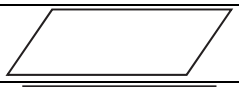
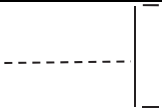
7. Распечатайте последовательностью нулей и единиц внутреннее хранение чисел: 1,5 и –1,5, представленных как: а) *Single*; б) *Real*; в) *Double*.

Тема 2
СТРУКТУРНАЯ МЕТОДОЛОГИЯ РАЗРАБОТКИ ПРОГРАММ
(8 часов)

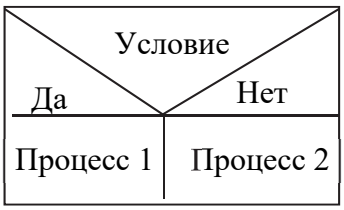
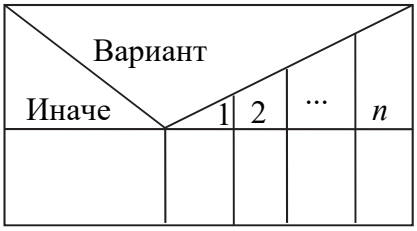
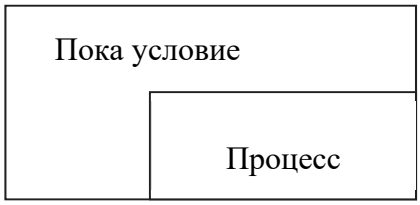
2.1. Блок-схемы, структурограммы

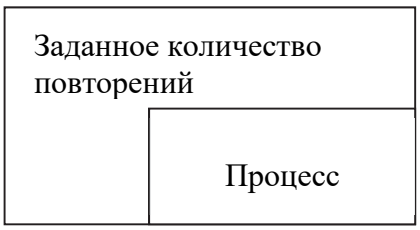
Блок-схема алгоритма отображает в графическом виде последовательность операций и переходные фазы. Каждому действию соответствует определенная фигура (ромб, квадрат, овал и т. д.), поэтому располагать их нужно в правильном порядке.

Приведем *условные графические символы схем алгоритмов и программ в виде блок-схем* и их функции.

Обозначение блока	Функция
	Выполнение операций, в результате которых изменяется значение данных
	Выбор направления выполнения алгоритма в зависимости от некоторых условий
	Выполнение операций, которые изменяют команды или группы команд
	Начало или конец схемы программы
	Использование ранее созданных и отдельно описанных алгоритмов
	Указание связи между прерванными линиями потока, связывающими символами
	Обмен данными между внешней и оперативной памятью
	Ввод-вывод данных, носителем которых служит бумага
	Связь между элементами схемы и толкованием
	Естественное – сверху вниз или слева направо (без стрелок) и очевидное со стрелками

Рассмотрим *основные конструкции структурограмм*.

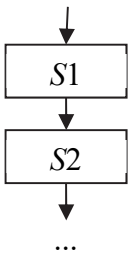
Обозначение символа	Функция	Операторы языка
	Любая группа действий, образующая блок	begin … end
	Выбор направления выполнения алгоритма в зависимости от некоторого условия	if … then … else
	Выбирается вариант дальнейшего действия	case … of … end
	Цикл с предусловием	while … do
	Цикл с постусловием	repeat … until

	Цикл на заданное количество повторений	for … to … do
		
		for … downto … do

Структурная блок-схема – это блок-схема, которая может быть выражена как композиция из шести элементарных схем, или структур управления.

Основные конструкции структур управления

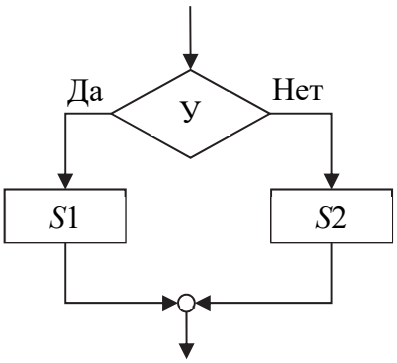
Следование



*S*1 – процесс 1

*S*2 – процесс 2

Альтернатива

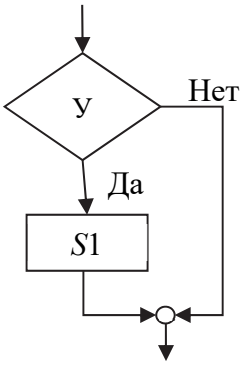


У – условие

*S*1 – процесс 1

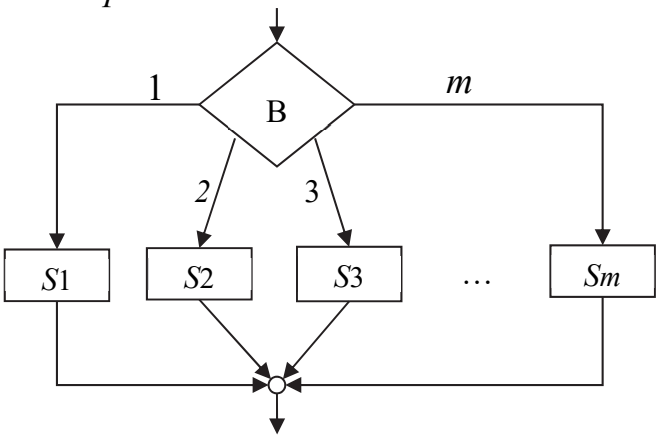
*S*2 – процесс 2

(оператор `if … then … else`)



(оператор `if … then`)

Выбор



B – вариант выбора

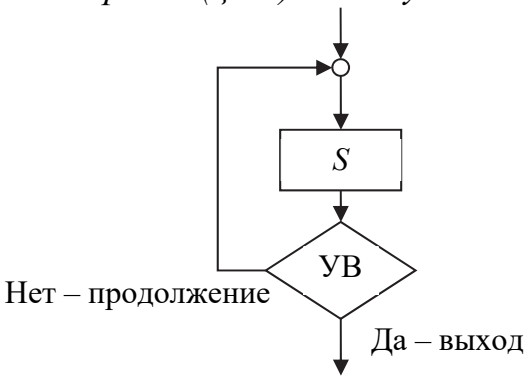
*S*1 – процесс 1

. . .

Sm – процесс *m*

(оператор `case … of … end`)

Повторение (цикл) с постусловием

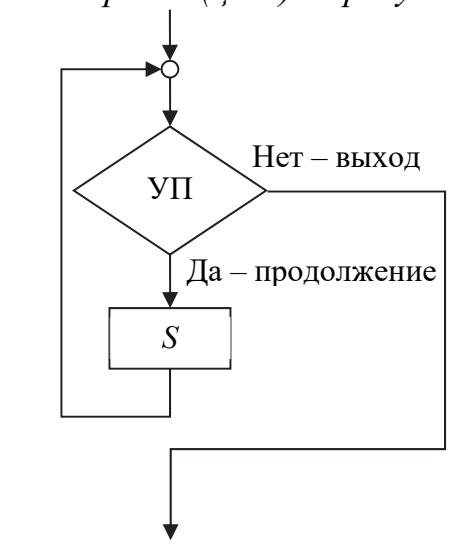


S – процесс

УВ – условие выхода из цикла

(оператор `repeat … until`)

Повторение (цикл) с предусловием

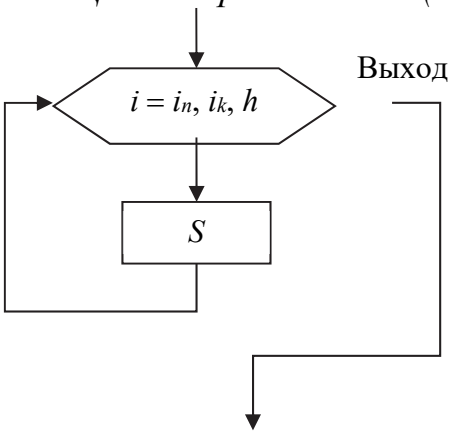


УП – условие продолжения цикла

S – процесс

(оператор `while … do`)

Цикл с перечислением (параметром)



i – параметр цикла
i_n – начальное значение параметра цикла
i_k – конечное значение параметра цикла
h – шаг изменения параметра цикла
S – процесс

(оператор `for … to(downto) … do`)

Метод пошаговой детализации

Пользуясь управляющими конструкциями уже при разработке алгоритма, легко следить за правильностью его частей, а этим и доказывать правильность всего алгоритма.

Современные компьютеры со своими системами программирования позволяют писать программу сразу на компьютере и довольно эффективно выявлять и исправлять ошибки. Это все хорошо для довольно простых и коротких программ. А когда нужно реализовать сложный алгоритм, тогда невозможность просмотра всего текста сразу может стать источником ошибок. Создание программы превращается в процесс беспрерывного внесения правок и вставок. В результате возникает запутанная и ненадежная программа.

Это неизбежно приводит к обязательному требованию – разрабатывать программу так же, как проектируется любое устройство. При этом программа получается как результат последовательных этапов разработки.

Наиболее общая тактика разработки программ состоит в *декомпозиции* всей задачи на отдельные более простые подзадачи, которые приводят к решению всей задачи. Затем *процесс декомпозиции* распространяется на подзадачи и т. д. до тех пор, пока не получатся подзадачи, легко реализуемые на языке программирования.

Декомпозиция задачи – это итерационный процесс, который не исключает обращение назад.

При разработке программы может оказаться, что на некотором шаге приняты неудовлетворительные решения или отдельные подзадачи почти не реализуемы на языке программирования. В этом случае, возможно, придется вернуться на несколько шагов назад и пересмотреть декомпозиции снова.

Задачи

Линейные и разветвляющиеся алгоритмы

1. Треугольник задан длинами сторон. Найдите:
а) длины высот;
б) длины медиан;
в) длины биссектрис;
г) площадь треугольника;
д) углы треугольника;
е) радиусы вписанной и описанной окружностей.

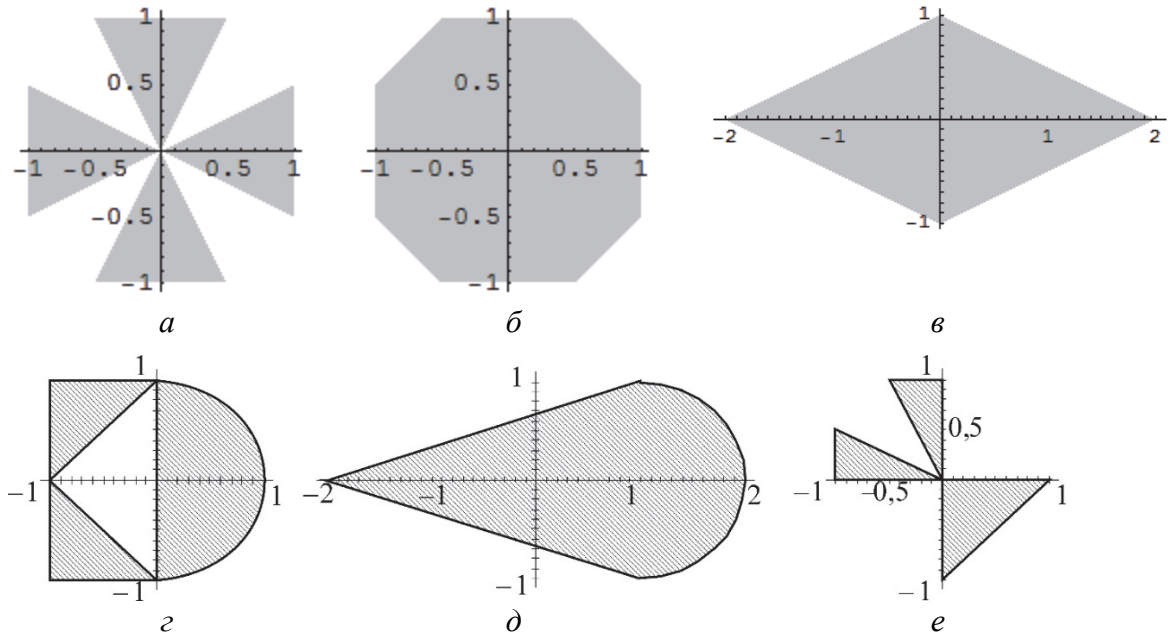
2. Треугольник задан координатами своих вершин. Найдите:
а) длины сторон;
б) длины медиан;
в) длины биссектрис;
г) периметр треугольника.

3. Найдите φ – угол (в градусах) между положением часовой стрелки в начале суток и ее положением в h часов, m минут и s секунд.

4. Даны вещественные числа a, b, c. Решите следующее уравнение: $ax^2 + bx + c = 0$, используя вспомогательный алгоритм решения приведенного квадратного уравнения.

5. Даны вещественные числа a, b, c. Решите следующее уравнение: $ax^4 + bx^2 + c = 0$.

6. По заданным координатам точки $M(x, y)$ выясните, находится ли эта точка в заштрихованной области:



7. Будем считать, что стандартные функции $\sin x$ и $\cos x$ применимы только тогда, когда аргумент x принадлежит отрезку $[0; \pi/2]$. Подсчитайте $y = \sin x$ и $z = \cos x$ для произвольного числа x.

8. Будем считать, что стандартные функции $\sin x$ и $\cos x$ применимы только тогда, когда аргумент x принадлежит отрезку $[0; \pi/4]$. Подсчитайте $y = \sin x$ и $z = \cos x$ для произвольного числа x.

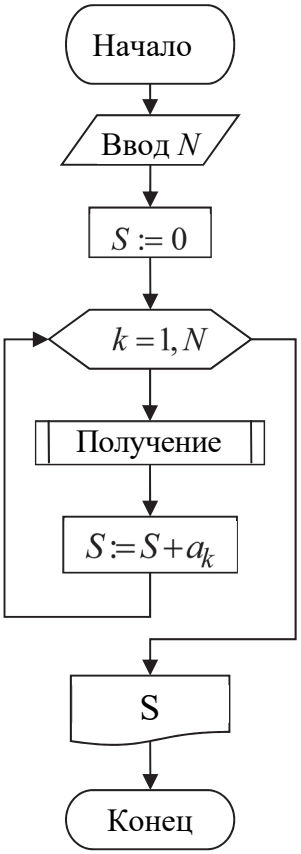
2.2. Структурные блок-схемы для базовых алгоритмов

Задача 1. Найти одну из следующих сумм:

1) N первых нечетных чисел;
2) N первых случайных чисел;
3) N первых простых чисел;
4) N чисел a_k, где $a_k = \dfrac{2^k}{k!}$, $k = 1, 2, 3, ..., N$.

Решение. Во всех случаях нужно подсчитать $S = \sum_{k=1}^{N} a_k$.

Блок-схема получения суммы чисел



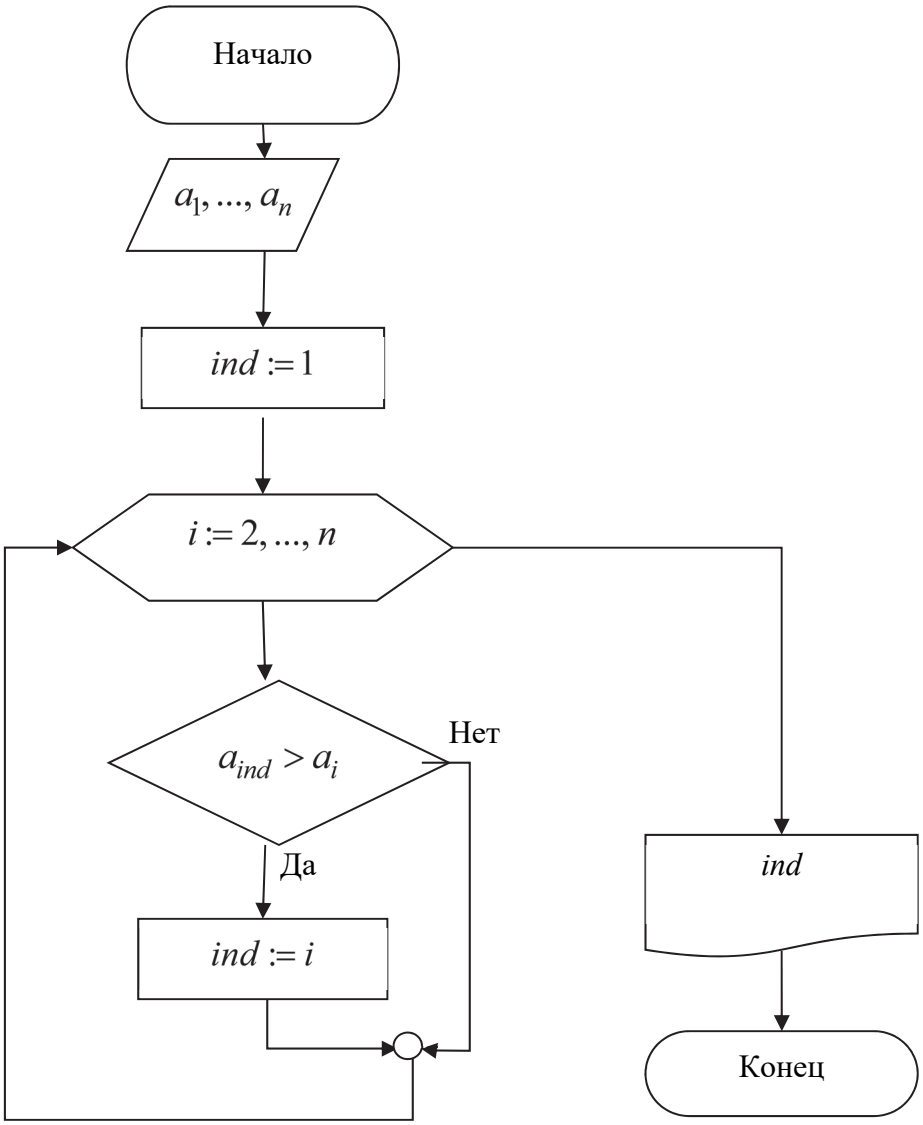
Задача 2. Найти индекс наименьшего элемента в одномерном массиве.

Алгоритм. Запоминаем индекс первого элемента *ind* = 1.

Организуем цикл на количество оставшихся элементов, в котором сравниваем все очередные элементы с элементом, имеющим индекс *ind*.

Когда находится меньший элемент, то его номер запоминаем в переменной *ind*.

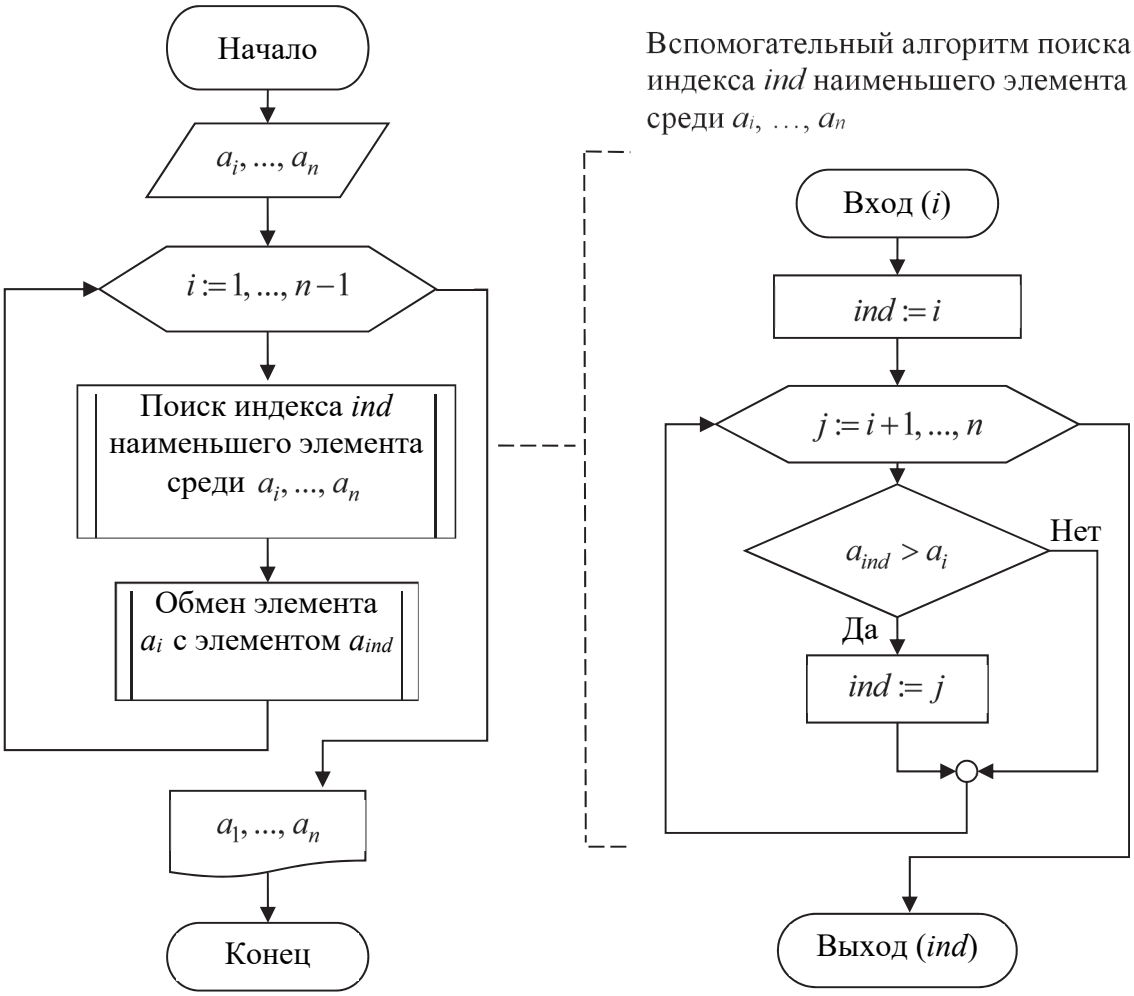
Блок-схема



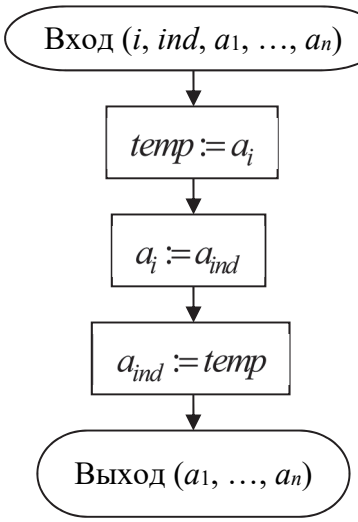
Задача 3. Упорядочить элементы массива по возрастанию.

Алгоритм. При разработке блок-схемы будем опираться на предыдущую задачу. Вызовем вспомогательный алгоритм поиска индекса наименьшего элемента среди a_i, …, a_n. Затем первый наименьший элемент обменяем с a_1, второй наименьший – с a_2 и т. д. Такой алгоритм упорядочения называется *сортировкой методом прямого выбора*.

Блок-схема



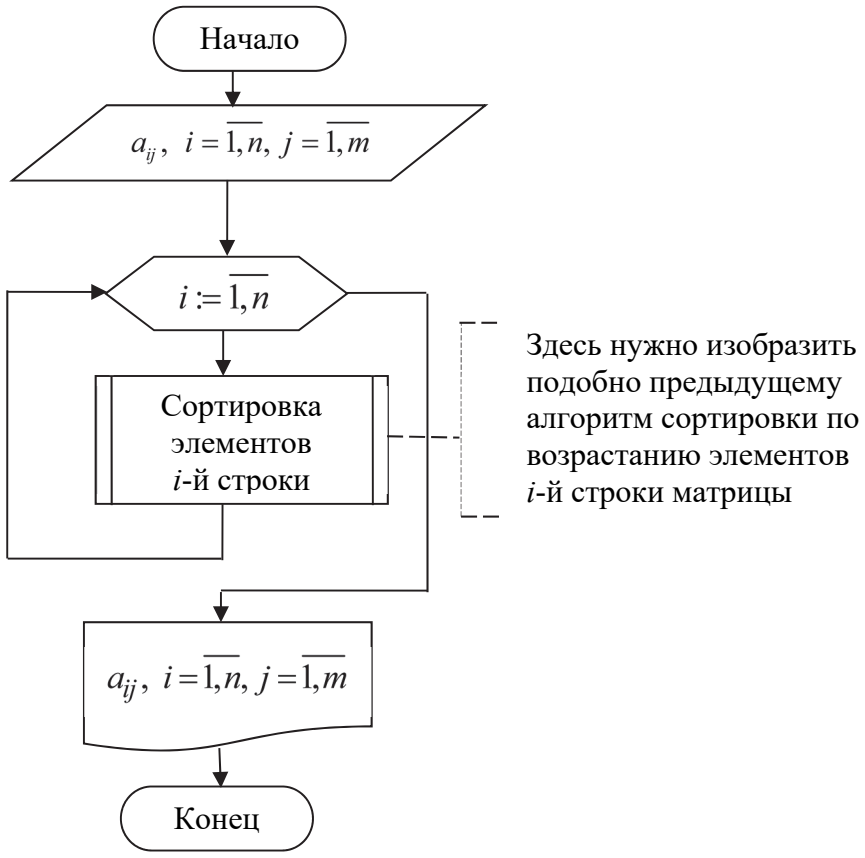
Вспомогательный алгоритм обмена элементов a_i и a_{ind}



Задача 4. Упорядочить элементы каждой строки матрицы $A_{n,m}$ по возрастанию.

Алгоритм. Используем метод пошаговой детализации.

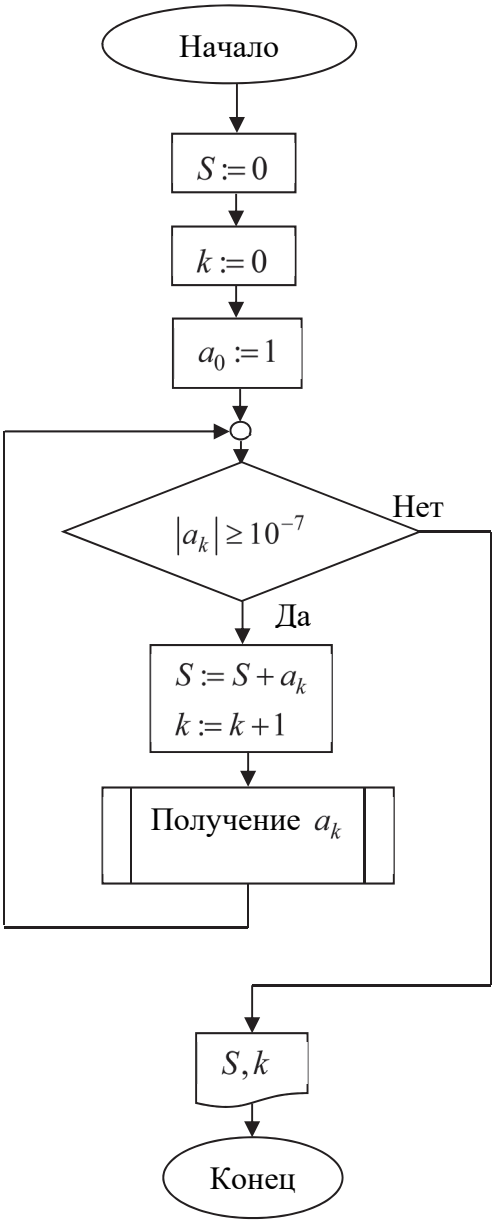
Так как каждая зафиксированная *i*-я строка матрицы представляет собой одномерный массив $a_{i1}, a_{i2}, ..., a_{im}$, то используем предыдущий алгоритм упорядочения одномерного массива, блок-схему которого нужно изобразить самостоятельно на месте комментария.



Задача 5. Найти сумму $S=\sum_{k=0}^{\infty}\frac{(-2)^k}{k!}=1-\frac{2}{1!}+\frac{4}{2!}-\ldots;$ процесс накопления слагаемых закончить, когда получится слагаемое по модулю меньше чем 10^{-7}.

Решение. Рассмотрим сумму $S=\sum_{k=0}^{\infty}a_k,$ где $a_k=\frac{(-2)^k}{k!}$. Вспомним сначала школьную задачу подсчета суммы бесконечной геометрической прогрессии. Возможность подсчитать сумму имела место только при условии, что каждое следующее слагаемое по модулю было меньше предыдущего.

Алгоритм. $S := 0$. Вычисляем первое слагаемое. Если оно по модулю не меньше 10^{-7}, добавляем к сумме S. Вычисляем следующее слагаемое. Если оно по модулю еще больше чем 10^{-7}, добавляем его в сумму S, и т. д., пока очередное вычисленное слагаемое по модулю не станет меньше чем 10^{-7}.



Получить a_k довольно просто. Так как $a_k = \frac{(-2)^{k-1}}{(k-1)!}\frac{(-2)}{k}$, то $a_k =$ $= -2 * a_{k-1} / k$, где $a_0 := 1$.

Замечание. a_k – элемент a с номером k. Если на номер k смотреть как на индекс, то под каждый элемент a_k ($k = 0, 1, 2, \ldots$) нужно место, а в такой

постановке задачи неизвестно количество a_k, которое требуется при объявлении массива. Очевидно, что все слагаемые хранить не нужно, так как после добавления слагаемого к сумме S можно его заменить следующим, вычисленным через предыдущее.

Задачи

Циклы на известное число повторений

1. Дано натуральное число n. Вычислите:

а) $\sqrt{3+\sqrt{6+...+\sqrt{3(n-1)+\sqrt{3n}}}}$;

б) $\frac{1}{\sin 1}+\frac{1}{\sin 1+\sin 2}+...+\frac{1}{\sin 1+\sin 2+...+\sin n}$.

2. Даны вещественные числа x, a, натуральное число n. Вычислите:

а) $\underbrace{(...((}_{n}x+a)^2+a)^2+...+a)^2+a$;

б) $\underbrace{(...((}_{n}x+a)x+a)x+...+a)x+a$.

3. Целое неотрицательное число M задано массивом своих двоичных цифр. Получите массив двоичных цифр числа $M + 1$.

4. Некоторое число записано в массиве последовательностью своих цифр в q-й системе счисления. Найдите значение этого числа в десятичной системе счисления, используя представление этого числа в виде многочлена по степеням q.

5. Подсчитайте:

а) $\sum_{i=1}^{50}\sum_{j=1}^{30}\sin(i^3+j^2)$;

б) $\sum_{i=1}^{100}\sum_{j=m}^{i}\frac{(-1)^j}{2i+j}$.

6. Утверждается, что разность любого натурального числа и суммы его цифр кратна 9. Проверьте этот факт для всех чисел, лежащих между заданными m и n.

7. Для заданного натурального n и вещественного x подсчитайте:

а) $\sum_{k=1}^{n}(-1)^k\frac{x^k}{(k+1)!}\sin\frac{k}{100+k^2}$;

б) $\sum_{k=1}^{n}\frac{(2x)^{2k}}{(k)!!}$.

8. Пусть

$$v_1 = v_2 = 0;\ v_3 = 1{,}5;\ v_i = \frac{i+1}{i^2+1} v_{i-1} - v_{i-2} v_{i-3}, \quad i = 4, 5, \ldots.$$

Дано натуральное число n ($n \geq 4$). Получите v_n.

9. Пусть

$$u_1 = u_2 = 0;$$
$$v_1 = v_2 = 1;$$
$$u_i = \frac{u_{i-1} - u_{i-2} v_{i-1} - v_{i-2}}{1 + u_{i-1}^{\ 2} + v_{i-1}^{\ 2}};$$
$$v_i = \frac{u_{i-1} - v_{i-1}}{|u_{i-2} + v_{i-1}| + 2},\ i = 3, 4, \ldots.$$

Дано натуральное число n ($n \geq 3$). Получите v_n.

10. Подсчитайте:

а) $\sum_{i=1}^{50} \sum_{j=1}^{40} \frac{1}{i + j^2}$;

б) $\sum_{i=1}^{n} \sum_{j=1}^{m} \frac{i - j + 1}{(i + j)^3}$;

в) $\sum_{i=1}^{50} \sum_{j=1}^{30} \sin(i^3 + j^2)$;

г) $\sum_{i=1}^{100} \sum_{j=m}^{i} \frac{(-1)^j}{2i + j}$.

11. Для заданного натурального n подсчитайте:

а) $\sum_{k=1}^{n} \frac{1}{(k^2)!}$;

б) $\sum_{k=1}^{n} \frac{(-1)^k}{k(2k)!}$;

в) $\sum_{k=1}^{n} k^k$;

г) $\sum_{i=1}^{n} \sum_{j=m}^{i} \frac{(-1)^j j^2}{2i + j}$.

12. Для заданного натурального n и вещественного x подсчитайте:

а) $\sum_{k=1}^{n} \frac{(2x)^{2k}}{(k)!!}$;

б) $\sum_{k=1}^{n} (-1)^k \frac{x^k}{(k+1)!} \sin \frac{x^k}{100 + k^2}$;

в) $\sum_{k=1}^{n} \frac{(2k-1)!!}{(2k)!!(2k+1)} x^{2k+1}$;

г) $\sum_{k=1}^{n} (-1)^k \frac{x^k \ln^k x}{k},\ x > 0.$

Итерационные циклы

1. Даны положительные вещественные числа a, ε. В последовательности $x_1, x_2, \ldots$, полученной по закону:

$$x_0 = \begin{cases} \min(2a,\ 0{,}95) & \text{при } a \le 1, \\ a/5 & \text{при } 1 < a < 25, \\ a/25 & \text{при } a \ge 25; \end{cases}$$

$$x_n = \frac{4}{5}x_{n-1} + \frac{a}{5x^4_{n-1}},\ n = 1, 2, \ldots,$$

найдите первый член x_n, для которого выполнено неравенство $\frac{5}{4}a\left|x_n - x_{n-1}\right| < \varepsilon$. Вычислите для найденного значения x_n разность $a - x_n^5$.

2. Пусть

$$v_1 = v_2 = 0;\ v_3 = 1{,}5;\ \ v_i = \frac{i+1}{i^2+1}v_{i-1} - v_{i-2}v_{i-3},\ \ i = 4, 5, \ldots.$$

Найдите первый член v_n, для которого выполнено неравенство $\left|v_n - v_{n-1}\right| < \varepsilon$.

3. Сравните скорость сходимости (число слагаемых для достижения заданной точности ε) следующих разложений чисел π:

1) $\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots\right)$;

2) $\pi = 3 + 4\left(\frac{1}{2\cdot3\cdot4} - \frac{1}{4\cdot5\cdot6} + \frac{1}{6\cdot7\cdot8} - \frac{1}{8\cdot9\cdot10} + \ldots\right)$;

3) $\pi = \sqrt{6\left(1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \ldots\right)}$.

4. Для заданного вещественного x подсчитайте:

а) $\sum_{k=1}^{\infty}(-1)^k \frac{x^k}{(k+1)!}\cos\frac{x^k}{100 + k^2}$; б) $\sum_{k=1}^{\infty}\frac{(2x)^{2k}}{(k)!!}$.

5. Даны положительные вещественные числа a, x, ε. В последовательности $y_1, y_2, \ldots$, полученной по закону $y_0 = a,\ y_i = \frac{1}{2}(y_{i-1} + \frac{x}{y_{i-1}}),\ i = 1, 2, \ldots$, найдите первый член y_n, для которого выполнено неравенство $\left|y^2_n - y^2_{n-1}\right| < \varepsilon$.

Метод пошаговой детализации

1. Задана вещественная матрица размера $n \times m$. Определите числа $b_1, ..., b_n$, равные:

а) суммам элементов строк;

б) произведениям элементов строк;

в) наименьшим значениям элементов строк;

г) значениям средних арифметических элементов строк;

д) разностям наибольших и наименьших значений элементов строк.

2. Задана вещественная матрица размера $n \times m$. Определите числа $b_1, ..., b_n$, где b_k – это:

а) наибольшее из значений *k*-й строки;

б) произведение квадратов тех элементов *k*-й строки, модули которых принадлежат отрезку [–1; 1,5];

в) число отрицательных элементов *k*-й строки;

г) значение первого по порядку положительного элемента *k*-й строки;

д) модуль суммы наибольшего и наименьшего значения элементов *k*-й строки.

3. Задана вещественная матрица размера *n*×*m*. Определите числа массива $b_1, ..., b_n$ из нулей и единиц, если $b_k = 1$, тогда, когда:

а) элементы *k*-й строки образуют возрастающую последовательность;

б) элементы *k*-й строки образуют либо возрастающую, либо убывающую последовательность;

в) число отрицательных элементов *k*-й строки меньше числа положительных элементов;

г) число ненулевых элементов *k*-й строки равно 1.

4. Задана вещественная квадратная матрица порядка *n*. Получите $x_1 x_n + x_2 x_{n-1} + ... + x_n x_1$, где x_k – наибольшее значение элементов *k*-й строки данной матрицы.

5. Все элементы с наименьшим значением в каждой строке данной целочисленной квадратной матрицы порядка *n* замените нулями.

Тема 3
СРЕДСТВА АЛГОРИТМИЧЕСКИХ ЯЗЫКОВ
(2 часа)

3.1. Алгоритмические языки программирования

Алгоритмический язык программирования – формальный язык, используемый для записи, реализации и изучения алгоритмов.

Алгоритмический язык образуют три его составляющие: алфавит, синтаксис и семантика.

Алфавит – фиксированный для данного языка набор символов (букв, цифр, специальных знаков и т. д.), которые могут быть использованы при написании программы.

Синтаксис – правила построения из символов алфавита специальных конструкций, с помощью которых составляется алгоритм.

Семантика – система правил толкования конструкций языка.

Таким образом, программа составляется с помощью соединения символов алфавита в соответствии с синтаксическими правилами и с учетом правил семантики.

Содержательно язык программирования – это средство общения между человеком (программистом) и компьютером (исполнителем).

Всякий язык программирования можно определить как множество *предложений*, т. е. некоторое множество цепочек или конечных последовательностей элементарных единиц из некоторого непустого конечного множества символов, называемого алфавитом языка.

У каждого языка свой уникальный синтаксис, который состоит из команд, операторов, функций и структур данных.

Синтаксису противопоставляется *семантика языка*, которая определяет значение написанного на языке программного кода. Она описывает, как код должен взаимодействовать с другими элементами программы (переменными, функциями, классами и модулями).

Изучение синтаксиса языка – ключевой элемент освоения языка программирования. Только так можно создавать работоспособные приложения, веб-сайты и другие программы.

Кроме того, знание синтаксиса помогает разработчикам писать чистый и понятный код, улучшать и оптимизировать его.

Использование *интервалов* улучшает читабельность вашего кода. Не используйте пробелы в конце строк и в пустых строках.

Строки обычно пишутся *не длиннее 80 символов*. Комментарии должны предшествовать коду, к которому относятся. Для длинных комментариев используйте многострочные комментарии.

Несмотря на то что у разных языков программирования свои уникальные синтаксические конструкции, *знание основных концепций одного из них может помочь в освоении любого другого*.

Популярность языков программирования

Популярность языков программирования меняется со временем. Полезно следить за показателями, отражающими их популярность с разных точек зрения. Тем, кто делает первые шаги в программировании, важно определиться с направлением, в котором вы будете работать. Тем, кто желает повысить квалификацию, важно узнавать о новых и набирающих популярность технологиях, чтобы развиваться.

Типизация языков программирования

Также языки бывают с динамической и статической типизацией. В первом случае тип переменной определяется непосредственно при выполнении программы, во втором – на этапе компиляции. Например, Python – это динамически типизированный язык; такие языки, как C, C#, Java, – статически типизированные.

Python – это язык с динамической типизацией, т. е. в ходе выполнения программы одна и та же переменная может хранить значения различных типов. Оператор присваивания просто создает связь между именем переменной и значением. Хотя каждое значение имеет собственный тип данных, например целое число или строка, сами переменные не имеют типа и в процессе выполнения программы могут ссылаться на значения любых типов. Этим Python отличается от языка C или Pascal, например, в которых каждая переменная имеет определенный тип, размер и местоположение в памяти, где сохраняется ее значение.

Неявная типизация означает, что при объявлении переменной вам не нужно указывать ее тип, при явной – это делать необходимо.

В качестве примера языков с явной типизацией можно привести Java, C++. Вот как будет выглядеть объявление целочисленной переменной в Java и Python.

Java:

```
int a = 1;
```

Python:

```
a = 1
```

Сильная типизация не позволяет производить операции в выражениях с данными различных типов, слабая – позволяет.

В языках с сильной типизацией вы не можете складывать, например, строки и числа, нужно все приводить к одному типу.

К первой группе можно отнести Python, Java, ко второй – C и C++.

Стиль программирования

Стиль программирования – это способ построения программ, основанный на определенных принципах программирования, и выбор подходящего языка, который делает понятными программы, написанные в этом стиле.

Каждый стиль программирования имеет свою концептуальную базу и требует своего умонастроения и способа восприятия решаемой задачи.

Стиль программирования – это выражение опыта общения людей, занимающихся разработкой и использованием программ, выработанного в результате многолетней практики такого общения.

Рассмотрим общие рекомендации к стилю программирования.

1. Программа должна быть спроектирована таким образом, чтобы существовала *возможность ее поддержки и ресурсы расширения* при возникновении сопутствующих задач.

2. Все имена следует записывать по-английски. Комментарии также рекомендуется писать на английском языке, тогда они будут легко переносимыми с компьютера на компьютер.

3. *Комментарии* нужны только для человека. Комментарии – важнейший элемент стиля и мощный инструмент, позволяющий сделать программу действительно понятной.

4. Каждое *описание переменной*, смысл которой нельзя угадать по ее названию, должно сопровождаться комментарием, поясняющим не программистскую, а содержательную суть переменной.

5. *Алгоритм* программы также нуждается в пояснениях. Комментируйте логику программы.

6. Следует учитывать стандартный размер экрана, за которым работает программист. *Длина строк* программы не должна превышать ширины экрана (80 символов).

7. Пока размер программы составляет несколько десятков строк, *модульность* достаточно легко обозревается и ее проще хранить в одном файле. С ростом объема программы просматривать ее становится тяжело. Если программа поделена на модули, то определить, в каком модуле ошибка, проще.

8. Необходимо использовать верхний, нижний и смешанный регистр *символов* для визуальной маркировки различных сложных идентификаторов:

1) *зарезервированные слова* языка Pascal должны быть записаны только маленькими буквами. Например:

`type`, `var`, `const`, `procedure`, `function`, `begin`, `end`, `if`, `then`, `else`, `repeat`, `until`.

2) *директивы* компилятора должны быть записаны буквами в верхнем регистре;

3) *имена*, *представляющие типы*, должны быть обязательно написаны в смешанном регистре, начиная с верхнего. К именам типов можно добавлять префикс **`T`**. Например:

```
type
      TMatrix = array [1..10, 1..10] of integer;
```

4) *имена переменных*. При решении математических задач желательно не отходить от математической нотации обозначения данных. Другие *имена переменных* должны быть записаны в смешанном регистре, начиная с нижнего. Например:

`line`, `savingsAccount`.

5) *имена функций и процедур* должны быть написаны начиная с большой буквы, т. к. вызов функции без параметров визуально не отличим от просто подстановки переменной. Чтобы избежать путаницы, переменные всегда называть с маленькой буквы, а процедуры и функции с большой;

6) именованные *константы* (включая значения перечислений) должны быть записаны в верхнем регистре с нижним подчеркиванием в качестве разделителя. Например:

`MAX_ITERATIONS`, `COLOR_RED` – правильно;
`COLOR_red` – неправильно.

9. Следует *избегать* использования *глобальных переменных*, область видимости которых – весь файл.

10. В программе все постоянные значения, которые имеют некоторый смысл, следует оформлять как константы.

11. В операторе `if` наиболее вероятный положительный сценарий следует располагать в части `then`, исключение – в части `else`.

12. Следует строго избегать исполнимых *выражений в условиях*.

13. *Избегайте* использования оператора `goto`. Также ограничьте использование подобных утверждений, таких как `exit`, `break`, `continue`. Для этого достаточно просто изменить условие выполнения цикла или использовать цикл другого типа.

14. Рекомендуется всегда писать символ-разделитель операторов «;» сразу же после оператора. Это позволит избежать ошибок при добавлении новых операторов.

15. В ходе оформления *процедур и функций* необходимо следовать следующим рекомендациям:

1) избегать *дублирования* кода. Код, используемый два и более раза, оформляется в виде функции или подпрограммы. Дублируемый код в классе необходимо выделять в отдельный метод;

2) каждая функция, процедура должна решать только *одну* задачу. В этом случае ее будет легко протестировать;

3) *избегать длинного списка* аргументов. Приближаясь к числу семь, список аргументов становится не воспринимаемым при чтении. Возможно, следует объединить группы аргументов в новый тип данных;

4) процедура *не должна быть большой*. Объем процедуры должен быть таким, чтобы ее было несложно охватить как единое целое, понять сразу всю логику ее работы, иначе есть смысл подумать о дальнейшем разбиении алгоритма.

16. Во время *форматирования кода* необходимо учитывать, что легкость восприятия программы зависит от того, насколько удачно размещен ее текст.

17. Программы следует писать «лесенкой». Она должна отражать структурную вложенность языковых конструкций. Например:

```
sum := 0;
for i := 0 to n do
  if not odd(i) then writeln('even')
  else
      begin
       writeln('odd');
       sum := sum + i
      end;
writeln ('Sum of odd numbers = ', sum);
```

18. В условных конструкциях и теле цикла, даже если оператор один, *предпочтительнее* ставить операторные скобки `begin` ... `end`, поскольку в дальнейшем код может измениться, и понадобится поставить несколько операторов. Необходимо следовать правилам написания конструкции `begin` ... `end`:

1) использовать пустые строки для *отделения логически обособленных блоков* друг от друга, функции разделять одной пустой строкой;

2) скобки не отделять пробелами с внутренней стороны, между функцией и ее аргументами пробел не ставить;

3) в списке аргументов перед запятой пробел не ставить, после запятой – ставить. Полезно ставить пробелы при знаках действий.

19. Необходимо следовать правилу «один оператор – одна строка». Нельзя размещать в строке несколько операторов – это затрудняет понимание.

20. Не располагать блок *из нескольких инструкций* на одной строке после `if, while` и т. д.

21. Все блоки описаний `var`, `const` и т. д. следует использовать не более одного раза.

3.2. Методы разработки программ

На стадии разработки алгоритмов и программ существенную помощь оказывают методы проектирования:

- проектирование сверху вниз (нисходящее программирование);
- модульное программирование;
- проектирование снизу-вверх (восходящее программирование);
- структурное кодирование.

Ниже приведен *план разработки алгоритмов и программ*.

1. Переформулировать условие задачи на более алгоритмический вариант.
2. Выяснить, что на входе и что на выходе.
3. Составить тестовые примеры, не забывая о «крайних случаях».
4. Решить тестовые примеры вручную.
5. Если задача решается как композиция известных задач, то перейти к пункту 9.
6. Разработать алгоритм программы.
7. Выполнить ручную отладку.
8. Если есть ошибки, то перейти к пункту 6.
9. Написать текст программы.
10. Решить программу на компьютере и проверить ее на всех тестовых примерах.
11. Документировать решение задачи.

Задачи

По плану разработки алгоритмов и программ оформить задачу.

Даны вещественные числа a, b, c. Решите следующее уравнение $ax^4 + bx^2 + c = 0$, используя вспомогательный алгоритм решения приведенного квадратного уравнения.

Тема 4
ПРОСТЫЕ И СТРУКТУРИРОВАННЫЕ ДАННЫЕ И РАБОТА С НИМИ
(4 часа. Тестирование)

Тип данных определяет множество значений, которые могут принимать данные программы, и множество операций, допустимых над этими данными.

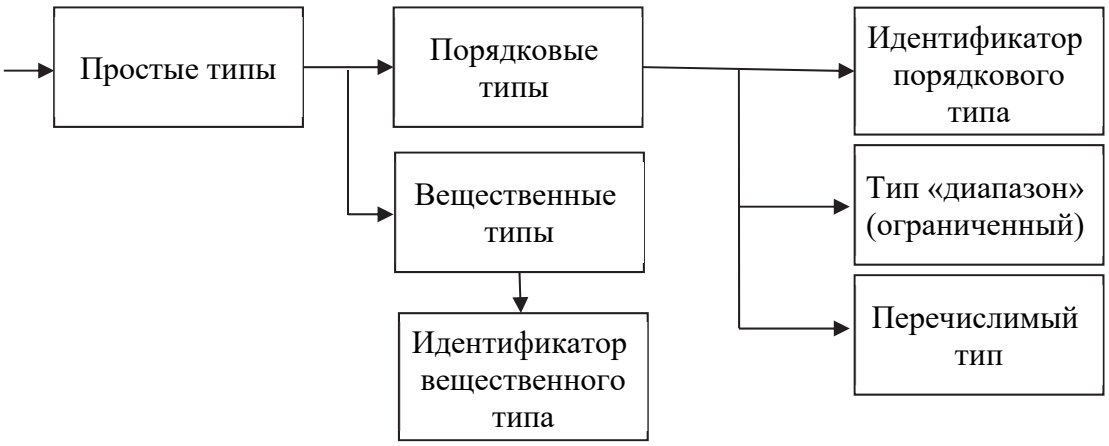
Pascal является типизированным языком, т. е. тип данных определяется при их описании и не может быть изменен. Данные могут участвовать только в операциях, допустимых их типом.

Базовыми в системе типов являются *простые (скалярные) типы*. *Стандартные* скалярные типы делятся на четыре группы*:*

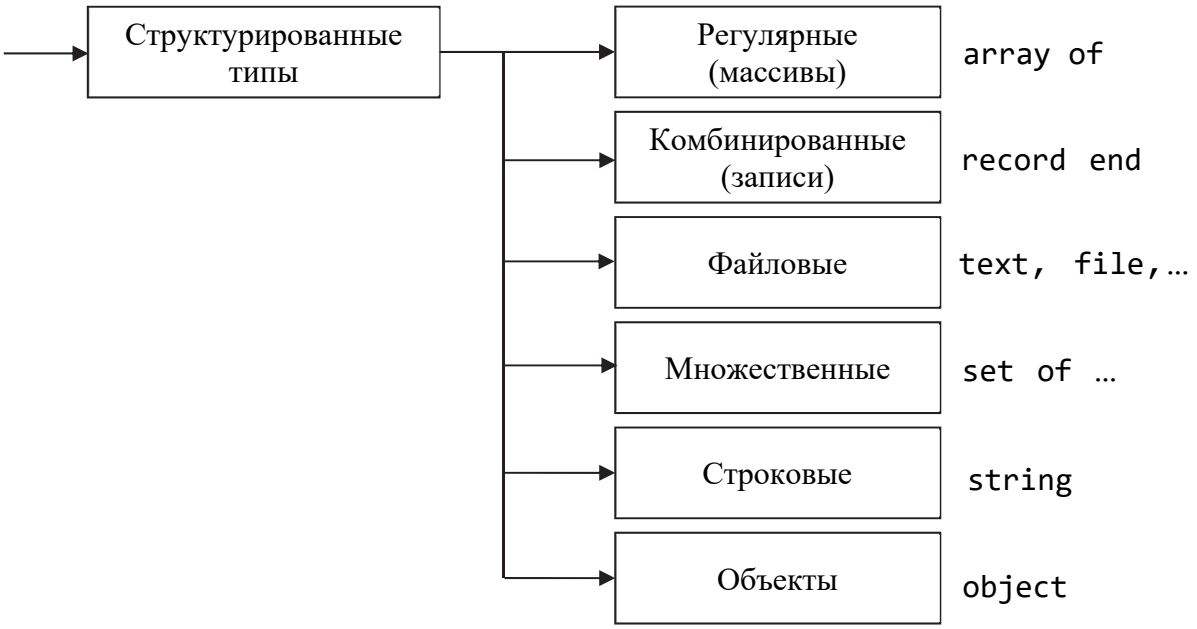
1) целочисленные (`Shortint, Integer, Longint, Byte, Word`);
2) вещественные (`Real, Single, Double, Extended, Comp`);
3) символьный (`Char`);
4) логический (`Boolean`).

Ниже приведена система типов языка Pascal.

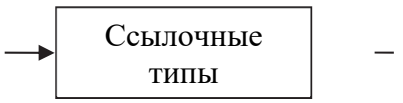
Первая группа



Вторая группа

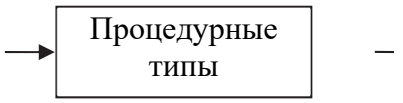


Третья группа



– образуются из любых других типов (как ^базовый тип)

Четвертая группа



– позволяют работать с подпрограммами как с переменными

Составные, или структурированные, типы по определенным правилам строятся из простых типов. Любой скалярный тип характеризуется множеством его различных значений со своим линейным порядком.

4.1. Простые типы

Целочисленные данные

Эта группа типов обозначает множество целых чисел в различных диапазонах. Существует пять целых типов, которые отличаются допустимым диапазоном значений и размером занимаемой оперативной памяти. Их характеристики и названия приведены в таблице.

Целый тип	Диапазон	Размер памяти	Особенность
Shortint	–128 .. 127	1 байт	Со знаком
Integer	–32768 .. 32767	2 байта	
Longint	–2147483648 .. 2147483647	4 байта	
Byte	0 .. 255	1 байт	Без знака
Word	0 .. 65535	2 байта	

Над целыми данными определены следующие операции, дающие целый результат: *, +, –, но / (деление) дает вещественный результат.

DIV – деление нацело (с усечением), откидывается дробная часть.

MOD – выделение остатка от деления двух целых операндов.

Кроме арифметических операций, к данным целого типа применяют операции отношений: <, >, =, <>, <=, >=. Результатом операций отношений является True (истина) или False (ложь).

Целый результат дают также следующие стандартные функции (аргумент от имени функции отделяется круглыми скобками):

abs(x), аргумент x – целого типа, результат – абсолютное значение x;

sqr(x), аргумент x – целого типа, результат – x^2;

trunc(x), аргумент x – вещественного типа, результат – целая часть x (дробная – отбрасывается) trunc(3.7) → 3, trunc(–3.7) → –3;

round(x), аргумент x – вещественного типа, результат – округленное целое.

Логический (булевский) результат дает функция odd(x), где x – целого типа. Если x нечетное, то результат TRUE, иначе – FALSE.

В следующих функциях аргумент может быть целочисленным, но результат будет вещественным:

sin(x), cos(x), arctan(x), ln(x), exp(x), sqrt(x).

Процедуры Dec(x[,n]) и Inc(x[,i]) переводят целочисленный аргумент в целочисленный.

Битовая арифметика

Битовая, или поразрядная, арифметика хорошо развита в языке Pascal. Необходимость в ней возникает, когда нужно работать не с десятичными значениями чисел, а с их двоичным представлением. Битовые операции позволяют сравнивать отдельные биты двух чисел, выделять отдельные фрагменты в числе, заменять их.

Битовые операции к вещественным числам не применяются. Они применяются только к данным целого типа: Byte, Shortint, Word, Integer, Longint.

Общая формула для значений данных беззнаковых типов `Byte` (1 байт) и `Word` (2 байта) имеет следующий вид:

`Byte`: значение $= \beta_7 \cdot 2^7 + \beta_6 \cdot 2^6 + \beta_5 \cdot 2^5 + ... + \beta_1 \cdot 2^1 + \beta_0;$

`Word`: значение $= \beta_{15} \cdot 2^{15} + \beta_{14} \cdot 2^{14} + \beta_{13} \cdot 2^{13} + ... + \beta_1 \cdot 2^1 + \beta_0.$

Значения разрядов β_0, β_1, ... равны или 0, или 1 и, соответственно, умножаются на вес разряда. Отсюда получаем диапазон представления:

`Byte`: $\underset{7\,6\,5\,4\,3\,2\,1\,0}{\boxed{11111111}} \Rightarrow 2^8 - 1 = 255$ (8 бит)

`Word`: $\boxed{11111111|11111111} \Rightarrow 2^{16} - 1 = 65535$ (16 бит)

Внутреннее отличие имеют представления целых типов со знаками: `Shortint, Integer, Longint`. Самый левый бит отводится под знак числа: для отрицательного числа он равен 1, для положительного – 0.

Можно запомнить формулу перевода из двоичной системы счисления в другую систему счисления для типов `Shortint` (1 байт), `Integer` (2 байта), `Longint` (4 байта):

`Shortint`: значение $= -\beta_7 \cdot 2^7 + \beta_6 \cdot 2^6 + \beta_5 \cdot 2^5 + ... + \beta_1 \cdot 2^1 + \beta_0;$

`Integer`: значение $= -\beta_{15} \cdot 2^{15} + \beta_{14} \cdot 2^{14} + \beta_{13} \cdot 2^{13} + ... + \beta_1 \cdot 2^1 + \beta_0;$

`Longint`: значение $= -\beta_{31} \cdot 2^{31} + \beta_{30} \cdot 2^{30} + \beta_{29} \cdot 2^{29} + ... + \beta_1 \cdot 2^1 + \beta_0.$

Отсюда диапазон представления чисел:

а) положительных `Integer`: $\boxed{01111111|11111111} \Rightarrow 2^{15} - 1 = 32767;$

б) отрицательных `Integer` (получим наименьшее, если ничего не будем добавлять): $\boxed{10000000|00000000} \Rightarrow -2^{15} = -32768;$

в) положительных `Shortint`: $\boxed{01111111} \Rightarrow 2^7 - 1 = 127;$

г) отрицательных `Shortint`: $\boxed{10000000} \Rightarrow -2^7 = -128.$

Рассмотрим *действия битовой арифметики*.

Первая группа поразрядной арифметики – это логические операции над битами.

Операция	Название	Форма записи	Приоритет	Тип
`NOT`	Поразрядное отрицание	`NOT A`	1 (высший)	Унарная
`AND`	Логическое умножение (и)	`A1 AND A2`	2	Бинарная
`OR`	Логическое сложение (или)	`A1 OR A2`	3	
`XOR`	Исключающее «или»	`A1 XOR A2`	4 (низший)	

`NOT` – поразрядное отрицание – значение каждого бита изменяет на противоположное.

`AND, OR, XOR` – логические операции, выполняемые поразрядно в соответствии с таблицей истинности.

A	B	A AND B	A OR B	A XOR B	A XOR B XOR B
0	0	0	0	0	0
0	1	0	1	1	0
1	0	0	1	1	1
1	1	1	1	0	1

Следующая группа поразрядных операций – *циклические сдвиги*.

Операция	Название	Форма записи
`shl`	Циклический сдвиг на *N* позиций влево (*l*)	`A shl N`
`shr`	Циклический сдвиг на *N* позиций вправо (*r*)	`A shr N`

Вещественные данные

Данные этого типа имеют своими значениями подмножество вещественных чисел, которые допустимы в компьютере. Они занимают в памяти от 4 до 10 байт и представляются в форме с плавающей точкой в двоичной системе счисления. Количество значащих цифр числа ограничено, поэтому значения данных вещественного типа могут быть представлены в машине неточно.

Над вещественными данными определены *арифметические операции*, *операции отношений* и *ряд функций*.

1. Арифметические операции: +, –, *, / (один из операндов может быть целого типа, но результат получится вещественного типа).

2. Операции отношений: =, >, <, <>, >=, <= (результат – `true` или `false`).

Замечание. К операндам вещественного типа не стоит применять операцию отношения равно «=», поскольку условие может не выполниться из-за представления вещественных чисел в памяти ПК и неизбежных ошибок округления при подсчете значений выражений этого типа.

Поэтому отношение $a1 = a2$ лучше заменить отношением $\text{abs}(a1 = a2) < E$, где E – некоторая достаточно малая величина – погрешность округления.

3. Математические функции: abs(x), sqr(x), sin(x), cos(x), arctan(x), ln(x), exp(x), sqrt(x).

В перечисленных функциях аргумент x может быть и данным целого типа, но результат, кроме abs(x) и sqr(x), – всегда вещественный.

В стандарте языка нет операции возведения в степень a^x (за исключением sqr(x)), поэтому для вещественного x используют тождество $a^x = e^{x\ln a}$,

но тут надо следить за знаком основания a, а для целочисленного x используйте соответствующий циклический алгоритм.

4. Другие функции:

• frac(x) – вычисление дробной части x (`fraction` – дробь);

• int(x) – вычисление целой части x, отбрасыванием дробной;

• pi – возвращает значение числа π: 3,141592653897932385 (19 цифр);

• random – генерирует значение вещественного случайного числа из диапазона [0..1) ;

• random (I) – генерирует значение целого неотрицательного случайного числа из диапазона [0; I – 1];

• randomize – изменение базы генерации случайных чисел. Применяется только один раз в программе.

Замечание. В языке нет некоторых встроенных функций, поэтому их можно вычислять, используя известные формулы:

$$\operatorname{tg} x = \frac{\sin x}{\cos x}; \qquad \operatorname{ctg} x = \frac{\cos x}{\sin x}; \qquad \arcsin x = \operatorname{arctg} \frac{x}{\sqrt{1-x^2}};$$

$$\arccos x = \frac{\pi}{2} - \arcsin x; \qquad \log(x) = \frac{\ln x}{\ln 2}; \qquad \log_{10}(x) = \lg(x) = \frac{\ln x}{\ln 10};$$

$$\operatorname{arcctg} x = \frac{\pi}{2} - \operatorname{arctg} x; \qquad \operatorname{sh} x = \frac{e^x - e^{-x}}{2}; \qquad \operatorname{ch} x = \frac{e^x + e^{-x}}{2};$$

$$\operatorname{th} x = \frac{\operatorname{sh} x}{\operatorname{ch} x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}; \qquad \operatorname{cth} x = \frac{\operatorname{ch} x}{\operatorname{sh} x} = \frac{e^x + e^{-x}}{e^x - e^{-x}}.$$

Символьные данные

Переменные этого типа имеют описание `Char`, а значениями данных являются символы из множества `ASCII` (`American Standard Code for Interchange Information` – американский стандартный код для обмена информацией). Это множество состоит из 256 различных символов, упорядоченных определенным образом, и содержит буквы, цифры, разделители и другие символы. Первые 128 символов стандартные, остальные могут меняться.

Функция `Ord(C)` (`Order` – порядок), где `C` – типа `Char`, дает порядковый номер символа в таблице `ASCII` кодов.

Функция `Chr(i)` по порядковому номеру `i` дает символьное значение – символ. Если $0 \le$ `i` ≤ 255, то этот символ можно найти в таблице кодов.

Две функции `Pred(c)` и `Succ(c)` дают, соответственно, предыдущий и последующий символы. Их можно использовать и для других простых дискретных типов. Не имеют смысла такие выражения: `Succ(#255)` и `Pred(#0).`

Функция UpCase переводит символы из нижнего регистра в верхний регистр для букв латинского алфавита: UpCase('a')⇒'A'.

Логические данные

Переменные этого типа имеют описание Boolean. Существуют два значения логического типа (правда и ложь), и эти значения в языке Pascal обозначаются стандартными идентификаторами true и false. Значения логических типов занимают 1 байт памяти. Внутреннее представление для false есть 0, для true – 1. Функция ord возвращает значение 1 для логического значения true и значение 0 для логического значения false.

Над значениями логического типа возможны операции сравнения, причем считается, что false < true. Кроме того, существуют четыре стандартные логические операции: AND, OR, XOR, NOT.

AND	true	false
true	true	false
false	false	false

OR	true	false
true	true	true
false	true	false

XOR	true	false
true	false	true
false	true	false

Данные перечислимого типа

Перечислимый тип – это тип данных, в котором количество всех возможных значений ограничено. Его можно расписать в ряд по значениям, перечисляя через запятую в круглых скобках названия (идентификаторы) элементов-значений типа (эти названия должны быть уникальными в пределах программы). Перечислимому типу можно присвоить имя, тогда он описывается в секции TYPE.

К переменным перечислимого типа применимы функции Ord(x), Succ(x) и Pred(x).

Поскольку значения перечислимого типа упорядочены, их можно сравнивать: кто имеет больший порядковый номер, тот и больше.

Данные интервального типа

Мы можем определить тип, который будет содержать значения только из ограниченного поддиапазона некоторого базового порядкового типа.

Базовым типом, из которого вычленяются диапазоны, может быть какой-то целочисленный тип, Char и любой из введенных программистом перечислимых типов.

С данными интервального типа работают функции Succ, Pred, Ord.

К ним можно применять все стандартные подпрограммы и операции,

которые определены для базового типа. Только если диапазон взят из базового типа, который может выводиться на экран или вводится с клавиатуры (или ввод-вывод в файл), тогда и ограниченные данные таким образом можно вводить-выводить.

Выражения языка

Выражение – это формальное правило для подсчета некоторого (нового) значения. Выражения строятся из операндов, знаков операций и круглых скобок.

Операнды представляют собой «элементарные» значения. Ими могут быть переменные, константы, элементы массивов, вызовы функций и т. д.

Операции определяют действия по вычислению новых значений, исходя из значений операндов. Большинство операций являются бинарными, и знак операции записывается между операндами. Для унарных операций, а это @ – взять адрес, `NOT` – логическое отрицание, унарные ±, знак операции ставится перед операндом. Никогда нельзя опускать знак операции «*». Операции продуманы так, чтобы выражение писали в одну строку.

Синтаксис выражения подразумевает определенное старшинство операций (приоритеты). Приоритеты задают очередность выполнения операций в выражении. *Круглые скобки* ставятся, когда нужно изменить порядок действий, который определен приоритетами операций. Количество открытых скобок должно равняться количеству закрытых. Часть выражения, заключенная в круглые скобки, при подсчетах рассматривается как отдельный операнд, т. е. все операции внутри подвыражения будут выполнены в первую очередь.

Операнд, находящийся между двух операций с разными приоритетами, связывается с операцией с бóльшим приоритетом (… + *a**…), а если приоритеты равны, то берется слева.

Наибольший приоритет имеет операция вычисления значения функции, остальные – в соответствии с таблицей.

Операция	Приоритет	Вид операции
`@, NOT, +, -`	Высокий, первый	Унарные
`*, /, DIV, MOD, AND, SHL, SHR`	Второй	Бинарные типа умножения
`+, -, OR, XOR`	Третий	Бинарные типа сложения
`=, <>, <, >, >=, <=, IN`	Низкий, четвертый	Отношения

Операции подразделяются на арифметические, логические, строковые, операции отношений и др. Так же называются и выражения. Фактически данные, участвующие в выражении, и дают тип выражения.

Задачи

1. Определите результат выполнения следующих операций.

`20 div 3`	`2 div 3`	`7 div 5`
`125 div 25`	`20 mod 3`	`2 mod 3`
`7 mod 5`	`125 mod 25`	`12 and 22`
`-4 and 7`	`42 and 2`	`-32 and 2`
`12 or 22`	`-4 or 7`	`42 or 2`
`-32 or -2`	`12 xor 22`	`-4 xor 7`
`42 xor 2`	`-32 xor -2`	`9 shr 3`
`-5 shr 7`	`11 shr 4`	`36 shr 11`
`9 shl 4`	`-5 shl 7`	`11 shl 4`
`-11 shl 11`	`not 0`	`not 80`

2. Запишите следующие числа в виде констант.

`7!`	`-1/6`	`1/10000`
`6.38`	2^{-15}	`e`
`-0.6(4)`	2^{5}	`LXIV`
`11/4`	`-1/2`	

3. Какие возможные формы записи могут иметь в программе следующие числовые константы?

`23`	$365*10^{8}$	$11*10^{-13}$
`-12`	`1000`	`0.314`
`4.25`	`3.171`	$-2.427*10^{9}$
`-0.01`	`-0.00000125`	`175`
10^{-7}	`0.00001`	$5*10^{-4}$

4. Запишите на языке программирования следующие математические выражения, если это возможно.

$\frac{-b+\sqrt{b^2-4ac}}{2a};$	$i+2\sin 4x;$	$x\in[a;b];$
$n!(2n+1);$	$\frac{e^{-x}+5}{2};$	$\pi/6;$
$\frac{(-1)^n x^2}{n!};$	$\sum_{i=1}^{20}\left(i^2+1\right);$	$\lim_{n\to\infty}\ \frac{n}{\sqrt{n^2+1}};$
$\frac{5\pi}{6}\le x\le 2\pi;$	$\int_0^1\sqrt{x^2+7}dx;$	$\lvert x/2\rvert-5;$

$(-1)^n \frac{\pi}{4} + k\pi;$ $\frac{\log_5 x}{\log_7 5} + y;$ $y \in [0;1] \cup [7;15);$

$1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \cdots + \frac{x^{20}}{20!};$ $pk(kx+b)^{p-1};$ $\sqrt{2 + x\sqrt{2 + x\sqrt{2 + x}}}.$

5. Напишите функцию перекодировки любой буквы (латинской или кириллицы) из прописной в строчную.

6. Напишите функцию перекодировки любой буквы (латинской или кириллицы) из строчной в прописную.

7. Дано: `m, n, k, l` – целые числа, `x=1654.127`. Что будет выведено в результате выполнения следующего кода?

```
m:=trunc(x*100) mod 10 + round(x*100) mod 10;
n:=pred(round(x*100)) mod 10+succ(round(x*100)) mod 10;
k:=Trunc(int(x));
l:=succ(trunc(frac(x)*100));
Writeln(m,n,k,l);
```

8. Пусть имеется некоторое случайное восьмизначное число *n*. Требуется:
а) переставить первую и последнюю цифры числа;
б) добавить число 1 к первой цифре;
в) добавить число 5 к первой цифре;
г) добавить число 1 ко второй цифре;
д) добавить число 5 ко второй цифре;
е) удвоить первую цифру.

9. Найдите ошибки в следующих описаниях:

```
const
      n=180;
      pi=3.14159;
type
      figure='0'..'9';
      letter=a..z;
      angle=-n..n;
      cut=0..n-1;
      period=-pi..pi;
      plus='+'..'+';
      week=(sun, mon, tue, wed, thu, fri, sat);
      week_day=mon..fri;
      week_off=sat..sun;
```

10. Найдите ошибки в программе и исправьте их. Что напечатается в результате ее выполнения?

```
type   sezon = (zima,vesna,leto,osen);
var
       x, y  : sezon;
       t     : (teplo,xolodno);
{1} procedure oxota(s:sezon;var p:(teplo,xolodno));
 begin
   if (ord(s)=0) or (ord(s)=1) then p:=xolodno
   else p:=teplo
 end;
begin
    x:=vesna;
{2} y:=x;
{3} t:=teplo;
{4} y:=t;
{5} t:=xolodok;
{6} read(x,t);
{7} write(vesna);
{8} writeln(t);
{9} if succ(x)=leto  then  write(true)
    else write(false);
{10} case y of
      zima..leto : write('это - не осень');
      osen       : write('а вот это осень!');
     end;
{11} case t of
      pred(xolodno) : write('не холодно');
      succ(teplo)   : write('не тепло');
end;
{12} for x:=zima to osen do write(x<y);
{13} for t:=xolodno to teplo do write(ord(t));
{14} while ord(zima)+ord(vesna)<ord(osen) do
            write('Большой ПРИВЕТ!');
{15} oxota(x,t);
end.
```

11. Объясните работу следующей программы:

```
{$R-}
  var    a, b : integer;
         c    : real;
  begin
```

```
 a:=33000;
 b:=33;
 c:=a/b;
 writeln('c=',c:5:2);
end.
```

12. Объясните работу следующей программы:

```
{$R+}
  var    a, b : word;
         c    : real;
  begin
   a:=66000;
   b:=66;
   c:=a/b;
   writeln('c=',c:5:2);
  end.
```

4.2. Структуры данных и работа с ними средствами языка Pascal

Множества

Тип «множество» образуется специальной конструкцией:

```
set of базовый_порядковый_тип
```

Множество в языке Pascal – это ограниченная (<= 256 элементов) совокупность попарно различных объектов одного скалярного порядкового типа, который называется базовым. В качестве базового может быть любой порядковый тип с элементами, для которых функция `Ord` возвращает значения в диапазоне от 0 до 255. Скалярные типы `Byte` и `Char` могут быть основой для построения множеств.

Попытка объявленным переменным присвоить значения, которые не входят в базовое множество, не вызывает программное прерывание, такие значения просто игнорируются. Например:

```
type
SetOfChar  = set of Char;    {множество из символов      }
SetOfByte  = set of Byte;    {множество из чисел         }
SetOfDigit = set of 0..9;    {множество-интервал чисел   }
SetOfDChar = set of '0'..'9';{множество-интервал символов}
var
     SChar  : SetOfChar;
```

```
    SByte  : SetOfBYTE;
    SDigit : SetOfDigit;
Begin
      SDigit := [0,2,4,6,8];
      SDigit := [10];              {ошибка не фатальная}
      SChar  := ['a'..'z','A'..'Z'];
      SChar  := ['2'];             {ошибка не фатальная}
      SChar  := [2];{Type mismatch – не совпадение типов}
         …
```

Порядок чередования элементов при их перечислении не имеет значения, также не имеет значения количество повторений элемента. Например:

```
SChar := ['a', 'a', 'a', 'a']
```

эквивалентно однократному упоминанию символа `'a'`.

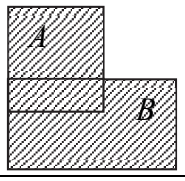
Например, после объявления переменных:

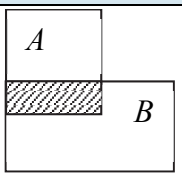
```
var
   x : byte;
   s : set of byte;
```

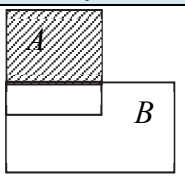
возможны следующие присваивания:

```
x := 3;
s := [1, 2, x];
s := s+[x+1];
```

Операции над множествами, которые определены в языке Pascal, представлены в следующих таблицах.

Объединение множеств (то, что входит в 1-е и 2-е множества)				Рисунок
В математике		В языке Pascal		
Обозначение	Пример	Обозначение	Пример	
$\cup$	$A \cup B$	+	$A + B$	

Пересечение множеств (то, что общее в 1-м и 2-м множествах)				Рисунок
В математике		В языке Pascal		
Обозначение	Пример	Обозначение	Пример	
$\cap$	$A \cap B$	*	$A * B$	

Разность множеств (то, что есть только в 1-м множестве, за исключением того, что есть в обоих)				Рисунок
В математике		В языке Pascal		
Обозначение	Пример	Обозначение	Пример	
\	$A \setminus B$	–	$A - B$	

Результат этих операций – всегда множество.

Вторая группа операций – это операции сравнения или отношений: =, <>, >=, <=. Результат всегда `true` или `false`.

Название		Форма	Пояснения: `false` – в противном случае
=	Проверка на равенство	`S1=S2`	`True`, когда `S1` и `S2` состоят из одинаковых элементов
<>	Проверка на неравенство	`S1<>S2`	`True`, когда `S1` и `S2` отличаются хоть одним элементом
<=	Проверка на подмножество ⊂	`S1<=S2`	`True`, когда все элементы `S1` входят в `S2`, независимо от порядка чередования
>=	Проверка на подмножество ⊃	`S1>=S2`	`True`, когда все элементы `S2` в `S1`
in	Проверка на вхождение элемента в множество	`E in S1` `E in […]`	`True`, когда значение элемента `E` принадлежит базовому типу множества и входит в множество `S1` или в множество-константу ([...])

Операцию `IN` хорошо использовать, когда нужно проверить попадание значения в диапазоны перечислимых типов:

```
if ch in ['a'.. 'x', 'A'.. 'X'] then …;
if j in [100..200] then …;
```

Здесь использованы константы типа «множество». Этим упрощаются логические операторы. Сравните следующее условие:

```
c in ['0'.. '9', 'A'.. 'Z']
```

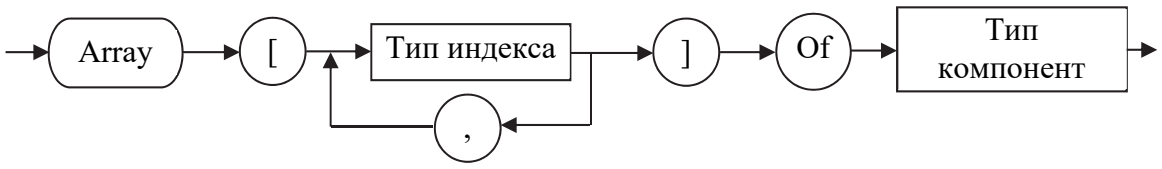
и такое:

```
(c>='0') and (c<='9') or (c>='A') and (c<='Z')
```

Массивы

Переменные типа массив, или *массив*, – это упорядоченная совокупность однотипных данных, которые хранятся последовательно.

Тип «массив» определяется синтаксической диаграммой.



Тип индекса – порядковый (кроме LongInt). Он показывает, в каких границах изменяется индекс массива. Тип компонента – любой известный тип данных.

Массив обязательно имеет фиксированные размеры, которые становятся известными по типу индексов и определяют, сколько элементов хранится в нем. Любой элемент в массиве можно найти по его индексу. *Индексы* – это выражения указанного порядкового типа.

Если в описании массива задан один индекс, массив называется *одномерным*, два – *двумерным*, n – *n-мерным*.

Размеры массива ограничены только объемом памяти.

В памяти компьютера массивы хранятся как последовательности компонентов, при этом, если в массиве несколько индексов, в первую очередь изменяется последний индекс, потом предпоследний и так далее до первого.

Отсюда следует, что матрицы хранятся по строкам:

```
a : array[1..5, 1..5] of byte;
{
a[1,1], a[1,2], a[1,3],…, a[1,5], a[2,1],…, a[5,5]
}
```

Адрес начала массива в памяти соответствует адресу его первого элемента, т. е. элемента с минимальными значениями индексов.

Строки символов

Объявление строковых переменных:

```
VAR идентификатор : STRING [max_длина];
```

или

```
VAR идентификатор : STRING;
```

Ограничения: 0≤ max_длина ≤255. Когда при определении опущена длина строки, тогда по умолчанию она составляет 255. Так как длина может быть разной, значит, STRING – это динамические строки.

Приведем различные примеры представления строк.

1. Первая форма записи:

```
'Номер п/п | Ф.И.О. | Должность     |'
```

где | – символ вертикальной черты имеет код #179 в альтернативной кодовой таблице.

2. Вторая форма записи:

```
'Номер п/п '#179' Ф.И.О. '#179' Должность     '#179
```

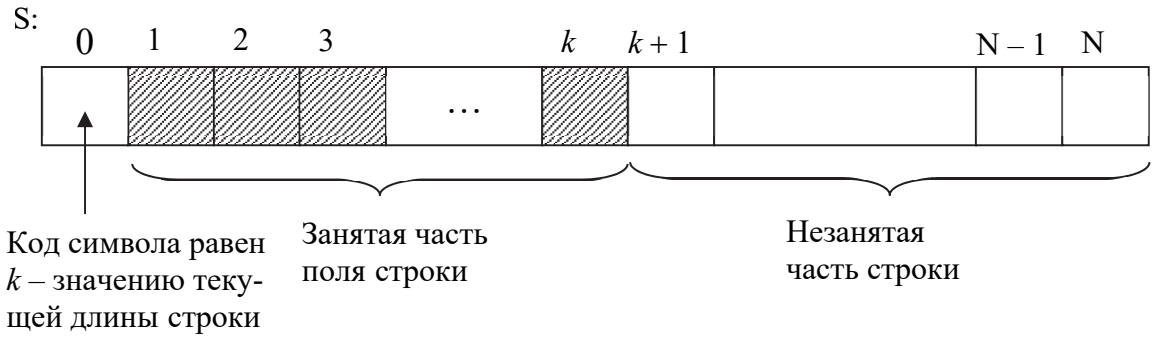
3. Строка #7#32#179#32#32#179 эквивалентна строке ^G' | |'

Тип STRING без длины является базовым строковым типом, и он совместим со всеми производными строковыми типами.

При попытке записать в переменную строку больше, чем объявлено в определении строки, лишняя справа часть отсекается.

Когда в переменную пишется строка, короче объявленной, запоминается ее текущая длина.

Для определенной строковой переменной S длиной N символов отводится N+1 байт памяти, из которых нулевой байт содержит длину строки, а остальные N байт отведены для символов строки.



Строки различных длин совместимы между собой в операторах присвоения значения и в операциях сравнения.

Редактирование строк

1. Функция Copy(S, Start, N) – выделяет из строки S подстроку длиной N символов, начиная с позиции Start (параметры Start, N – типа Integer).

Если значение Start превосходит длину строки S, результатом будет пустая строка. Если значение N больше, чем количество символов от Start до конца строки S, тогда вернется остаток строки S от позиции Start до конца строки.

2. Функция Pos(Subs, S) – возвращает номер символа в строке S, начиная с которого в строку S входит подстрока Subs (тип результата Pos – Byte). Если S не содержит Subs, тогда функция вернет 0.

3. Процедура Delete(S, Start, N) – удаляет N символов строки S, начиная с позиции Start. Когда Start=0 или превышает длину строки S, строка не изменится. Если N = 0, тогда строка не изменится. Если N превосходит остаток строки, будет удалена подстрока от Start и до конца строки S.

4. Процедура `Insert(Subs, S, Start)` – вставляет подстроку `Subs` в строку `S`, начиная с позиции `Start`. Если строка-результат после вставки имеет длину, превышающую длину строки `S`, то строка-результат автоматически укорачивается до объявленной длины `S` (отсекается правый конец).

5. Процедура заполнения `FillChar(V, Len, C)`, где `V` – переменная любого типа, `Len` (тип `Word`) – число байт переменной `V`, которые будут заполнены значением `C` (`C` – типа `Byte` или `Char`).

Преобразование строк

1. Процедура `Str(V, S)` – преобразует числовое значение `V` в строку `S`. После параметра `V` может быть указан формат в виде `V:m` или `V:m:n`, где `n < m`, `m`, `n` – данные целого типа; `m` – ширина поля для числа; для вещественного числа `n` – количество знаков после десятичной точки.

Для целых чисел задают только поле `m`. Если же для действительных чисел задают только `m`, тогда число представится в экспоненциальной форме.

2. Процедура `Val(S, V, ErrorCode)` – преобразует значение строки `S` в величину целочисленного или действительного типа и записывает в числовую переменную `V`. Переменная `ErrorCode` типа `Integer`. Если преобразование возможно, тогда переменная `ErrorCode` равна `0`, в противном случае переменная `ErrorCode` имеет номер позиции, в которой произошла ошибка при преобразовании.

Изменение (приведение) типов и значений

В языке Pascal существует мощное средство, позволяющее обойти все возможные ограничения на совместимость типов или значений (например, в операторе присваивания «:=»). Это *операция приведения типов*.

Операция приведения типов применяется только к переменным и значениям.

Суть этой операции в следующем: при определении типа мы определяем форму хранения информации в оперативной памяти, и переменная данного типа будет представлена в памяти наперед известной структурой.

Но если «посмотреть» на ее образ в памяти с точки зрения машинного представления другого типа, тогда можно будет трактовать то же самое значение как значение, принадлежащее к другому типу.

Формат операции приведения типов:

```
Имя_типа (переменная_или_значение);
```

Это имя_типа должно быть стандартным или указано программистом.

Приведение типов НЕ переопределяет типы переменных, а только дает возможность нарушить правила совместимости типов при условии, что соответствующие значения совместимы в машинном представлении.

Задачи

1. Подсчитайте значения выражений:

```
[2..13]*[3, 13..60]+[4..10]-[5..15]*[6];
[2..10]-[4, 6]-[2..12]*[8..15];
(['0'.. '7']+['2'.. '9'])*(['a']+['z']).
```

2. Объясните работу следующей программы:

```
Var
  p, q, r, s, y : Boolean;
 Begin
  Randomize;
  p := Boolean(random(2));
  q := Boolean(random(2));
  s := Boolean(random(2));
  r := Boolean(random(2));
  y := (p and (q and not q))  or
         not (r  or (s  or  not  s));
  writeln(y);
 end.
```

3. Объясните работу следующей программы:

```
{$R-}
Var
        x, y : integer;
        c    : real;
begin
  x:=word(66000);
  y:=66;
  c:=x/y;
  writeln('c=',c:5:2);
end.
```

4. Имеются описания:

```
Type
      Day    = (yesterday, today, tomorrow);
      Vector = array[1..30] of real;
var
      a: vector;
      b: array [-2..2] of (x, y, z);
      c: array ['0'..'9'] of vector;
      d: array [day] of 0..23;
```

Для каждого из массивов `a, b, c, d` определите:

- сколько в нем элементов;
- какие значения могут иметь его элементы;
- как найти его первый и последний элементы.

5. Объясните работу следующей программы:

```
program pract;
type
        arr4byte=array [1..4] of byte;
        arr2word=array [1..2] of word;
        rectype =
                record
                  word1, word2: word;
                end;
var
     l  : longint;
     s  : shortint;
     b  : byte;
     w  : word;
     a4 : arr4byte;
     a2 : arr2word;
     rec: rectype;
 begin
     l:= 123456;
     s:= -2;
```

```
     a2:= arr2word(l);
     a4:= arr4byte(l);
     w:= rectype(l).word1;
     w:= arr2word(l)[1];
     rectype(l).word1:= 0;
     b:= byte(s);
     b:= arr4byte(a2)[1];
end.
```

6. Объясните работу следующей программы:

```
Var
     a : longint;
     b : array[1..4] of byte absolute a;
 begin
   a:=$74657374;
   writeln(b[1]:3, b[2]:3, b[3]:3, b[4]:3);
 end.
```

7. Объясните работу следующей программы:

```
Var
     a : longint;
     b : array[1..4] of char absolute a;
 begin
   a:=$74657374;
   writeln(b[1]:3, b[2]:3, b[3]:3, b[4]:3);
 end.
```

8. Заданы три множества $A, B, C,$ содержащие целые числа из диапазонов $a_1..a_n,\ b_1..b_n,\ c_1..c_n$ соответственно. Образуйте новые множества:

а) $D = (A \bigcup B) \bigcap (A \setminus C)$;

б) $E = (A \setminus B) \bigcup (B \setminus C)$;

в) $F = (A \setminus B) \bigcap (B \bigcup C)$.

Напечатайте образованные множества и их мощности.

9. Имеется описание:

```
Var     x, y, z: set of 8..255;
```

Переменной *x* присвойте значение множества всех целых чисел от 8 до 255, переменной *y* – множества всех простых чисел из этого диапазона, а переменной *z* – множества всех составных чисел из этого диапазона. Напечатайте значения переменных *x*, *y*, *z*.

10. Имеются два предложения, слова в которых разделены запятыми и пробелами. Можно ли из букв первого предложения составить второе и наоборот? Если нельзя ни то, ни другое, то перечислите буквы, которых не хватает в первом предложении, чтобы составить второе. Решите эту задачу в двух вариантах: а) для букв латинского алфавита; б) отечественного алфавита.

11. Переведите целое десятичное число в двоичную систему счисления и напечатайте результат в виде строки символов из нулей и единиц.

12. Задан массив предложений. Определите частоту, с которой встречаются разные буквы в каждом из предложений.

13. Задан массив слов, описанный как типизированная константа, где каждое слово зашифровано (записано в обратном порядке). Расшифруйте и напечатайте эти слова.

14. Задана строка символов. Зашифруйте в ней все большие латинские буквы с помощью циклической подстановки:

$$A => B => C => \ldots => Y => Z => A.$$

15. Дана строка каких-то русских букв. Создайте еще одну строку русских букв, которых нет в заданной строке.

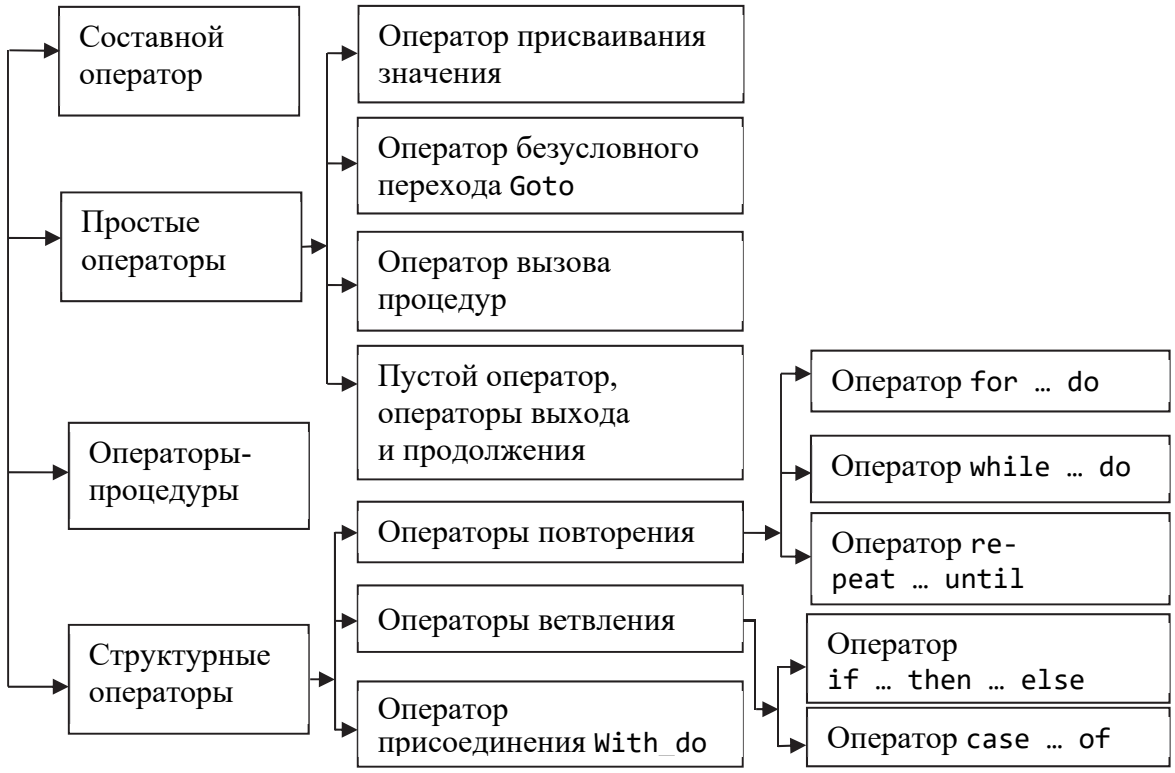
Тема 5
БАЗОВЫЕ ОПЕРАТОРЫ ЯЗЫКА И МЕТОДЫ ПРОГРАММИРОВАНИЯ
(6 часов. Тестирование)

Программа на алгоритмическом языке представляет собой совокупность инструкций, которые нужно выполнить с помощью автоматического устройства (ЭВМ).

Инструкции делятся на *операторы-описания структур данных* (описания) и *операторы-инструкции*, которые выполняют действие и в дальнейшем будут называться операторами.

Программа на языке Pascal состоит из описательной (I) и исполнительной (II) частей, записанных в строгой последовательности (I), (II). Первая часть может отсутствовать. Исполнительная часть представляет собой составной оператор.

Рассмотрим следующую схему основных операторов.

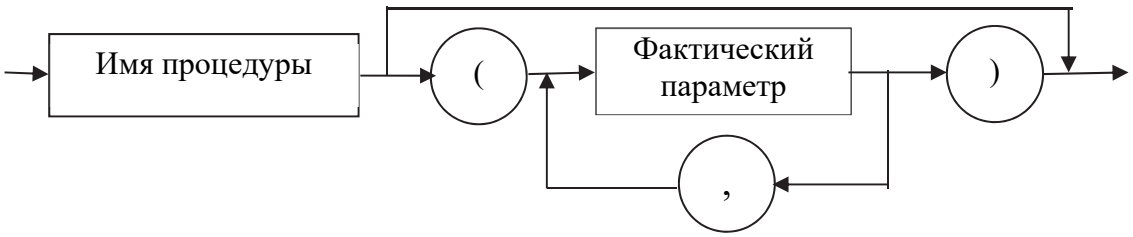


Замечание. В языке Pascal существуют процедуры, которым дан статус оператора. Однако имеются и операторы сложных конструкций.

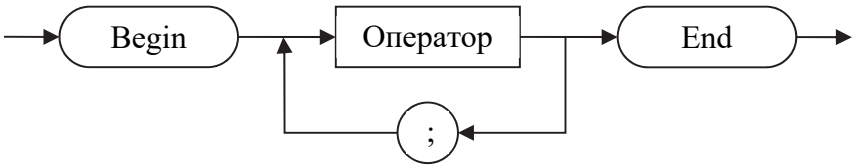
Напомним синтаксические диаграммы основных операторов.

5.1. Простые операторы

Оператор вызова процедуры



Составной оператор



Задачи

Простые операторы

1. Определите, есть ли в следующих выражениях пустые операторы:

```
if x>0 then x:=2 else; y:=x+1;
if odd(k) then else k:=0;
begin x:=2; y:=5; end;
begin f:=true; d:=d or a end;
begin if x=0 then goto 1; y:=x; 1:end;
```

2. Что напечатает следующая программа? Перепишите ее без операторов `goto`.

```
Var
    x : real;
Label
     10, 20, 30, 40, 50, 60, 70;
  Begin { ну как Вам эта программа? }
        x := 16;
        goto 50;
    10: x := sqrt(x);  goto 40;
    20: writeln(x);    goto 70;
    30: x := x-1;      goto 10;
    40: x := x*2+2;    goto 60;
    50: x := x + x;    goto 30;
    60: x := sqr(x);   goto 20;
    70: end.
```

Ввод и вывод данных

3. Определите, какие значения будут присвоены переменным после выполнения следующих операторов (`i, j, k` – переменные целого типа, `x, y` – вещественного, `s, t` – символьного или строкового типов, а символ _ обозначает пробел)?

Оператор	Варианты ввода с клавиатуры
`read(x, j, k);`	`23.5_100_27_<Enter>`
`read(x, j, k);`	`10_2_37.25<Enter>`
`read(s, i);`	`'Ввод текста'_25785<Enter>`
`read(s, i);`	`7_'ABAB'_<Enter>`
`read(i, t, y);`	`14_'12'_13<Enter>`
`read(i, t, y);`	`'i'_'j'_'k'_<Enter>`
`readln(i, j, k)`	`67_+43_123<Enter>`

4. Какие значения будут выведены на экран или напечатаны после выполнения следующих операторов (`I, K` – переменные целого типа, `X, Y, Z` – вещественного, `S, T` – символьного, `B` – логического)?

Значение переменных	Оператор
`i:=134; s:= 'арбуз'; y:=2.75;`	`write(i, s, y);`
`z:= 2e-12;`	`write('ответ=', z);`
`x:= 12.35;  y:= -113.005;`	`write(1.5*y, (x+y)/2);`
`i:= 175;  k:= -211;`	`writeln('i=',i, 'k=',k);`
`s:= 'отладка';  x:= 17;`	`writeln('s=', s, x);`
`i:= 587;`	`write(i, i, i);`
`s:= '? ';`	`writeln(s, s, s, s);`
`b:= false;`	`write(b, not b);`

5. Какие значения будут выведены на экран или напечатаны после выполнения следующих операторов (`I` – выражение целого типа, `R` – вещественного, `B` – логического, `C` – символьного, `S` – строкового)?

Значение переменной в выводимом выражении	Оператор
`234`	`write(i:8);`
`-1`	`write(i:6);`
`415`	`write(i+i:7);`
`-12.14`	`write(r:15);`

Значение переменной в выводимом выражении	Оператор
132.175	write(r:13);
-36.25	write(r:12);
412.02	write(r:8:3);
-16.25	write(r:7:2);
-16.25	write(r:9:4);
'a'	write(c:3);
'в'	write(c:6);
'аэропорт-2'	write(c:2, c:4);
'?? '	write(s:17);
true	write(b:5, not b:7);

6. Изобразите размещение чисел, напечатанных в результате работы следующей последовательности операторов:

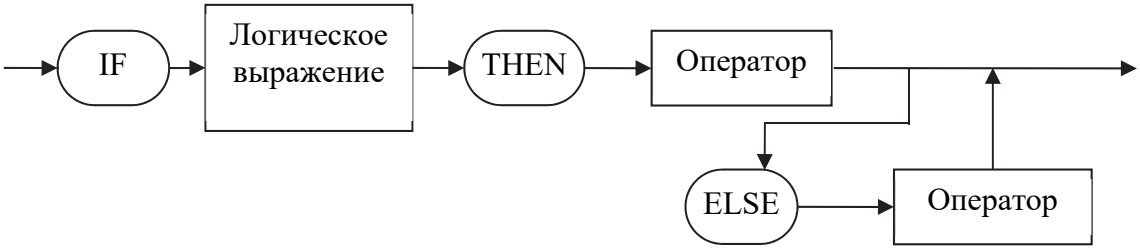
```
write(1);      write(2, 3);   writeln(4);
write(5, 6);   writeln;
writeln(7, 8);
```

7. Для заданного целого числа a напечатайте следующую таблицу, если известно k – количество цифр в этом числе:

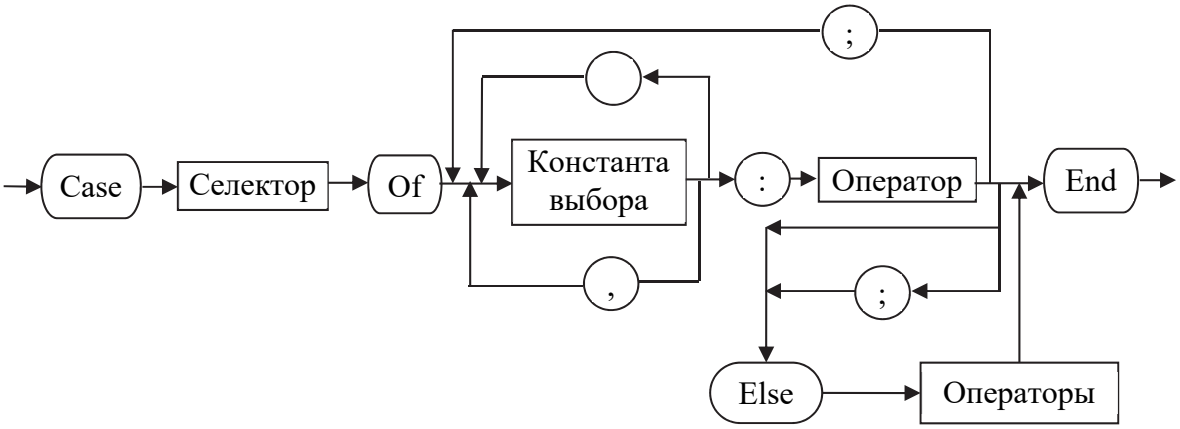
a
a^3 a^6
a^6 a^3 a

5.2. Структурированные операторы

Условный оператор



Оператор выбора



Задачи

Оператор условный и оператор варианта

1. Решите уравнения:

а) $x^2 - 4{,}6x + 5{,}29 = 0$ и $100x^2 - 460x + 529 = 0$;

б) $x^2 - 5{,}4x + 7{,}29 = 0$ и $100x^2 - 540x + 729 = 0$;

в) $x^2 - 5{,}2x + 6{,}76 = 0$ и $100x^2 - 520x + 676 = 0$

на компьютере и без него. Объясните возникшую ситуацию.

2. Запишите последовательность операторов для решения задач:

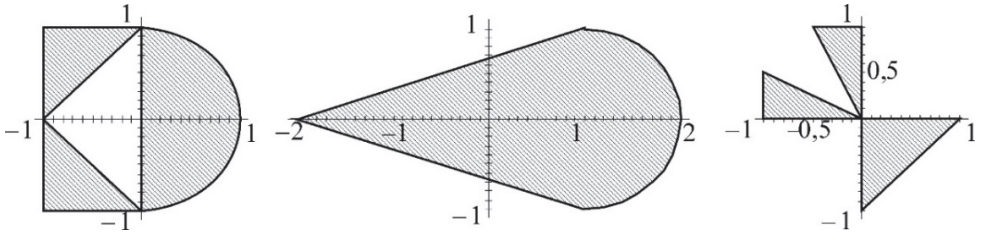
а) считая, что стандартные функции $\sin x$ и $\cos x$ можно применять только к аргументам из отрезка [0; π/2], подсчитайте $y = \sin(x)$ для произвольного x;

б) заданы числа a_1, b_1, c_1, a_2, b_2, c_2. Получите координаты точки пересечения прямых, которые описываются уравнениями $a_1x + b_1x = c_1$, $a_2x + b_{21}x = c_2$, или сообщите, что эти прямые совпадают, не пересекаются или вообще не существуют.

3. Задано целое число `k (1≤k≤180)`. Определите, какая цифра находится в `k`-й позиции последовательности **`10111213...9899`**, где выписаны подряд все двузначные числа.

4. Какая цифра находится на `k`-м месте в последовательности **`1101001000100001...`**, в которой выписаны подряд последовательные степени числа `10`: 0-я, 1-я, 2-я и т. д.?

5. По заданным координатам точки $M(x, y)$ выясните, находится ли эта точка в заштрихованной области.



а *б* *в*

6. Имеется описание:

```
type letter=(a, b, c, d);
var  x: letter;
```

Введите значение переменной x.

7. Напишите программу вычисления площади геометрических фигур:

$$S=\begin{cases} ab, & \text{если } n=1,\\ ah/2, & \text{если } n=2,\\ (a+b)h/2, & \text{если } n=3,\\ \pi R^2, & \text{если } n=4,\\ \pi R^2\varphi/360, & \text{если } n=5, \end{cases}$$

используя:

а) оператор `if`;

б) оператор `case`.

8. Укажите неправильные способы использования оператора `Case`:

а)
```
Var d : double;
    …
Case d of
  12.5 : write('12.5');
  18.0 : write('18.0');
  19   : write('19');
end.
```

б)
```
Var ch : Char;
…
Case ch of
  'a' : writeln('error'); exit;
  'b' : readln(a);
 else writeln('okey');
end.
```

в)
```
Var b : boolean;  a : byte;
 …
 Case b of
   true  : a:=a+1;
   false : a:=a-1;
 end.
```

г)
```
Var a, b : byte;
 …
 Case b of
   1 : begin writeln('first');
             a:=a+1; end;
   2 : writeln('tho');
   3 : exit;
 end.
```

9. Объясните работу следующего фрагмента программы:

```
var  k : word; …
begin
k := byte(1367);
 case k mod 13 of
   1, 3, 5, 7, 11 : d := k; p := true end;
   0..2           : begin d := 2; p := false end;
   9              : begin d := 1; p := false end
  else begin d := k mod 13; p := true end
 end;
 writeln(p, '  ', d);
end.
```

10. Программы определения значения наибольшего из трех чисел имеют вид (варианты ответов):

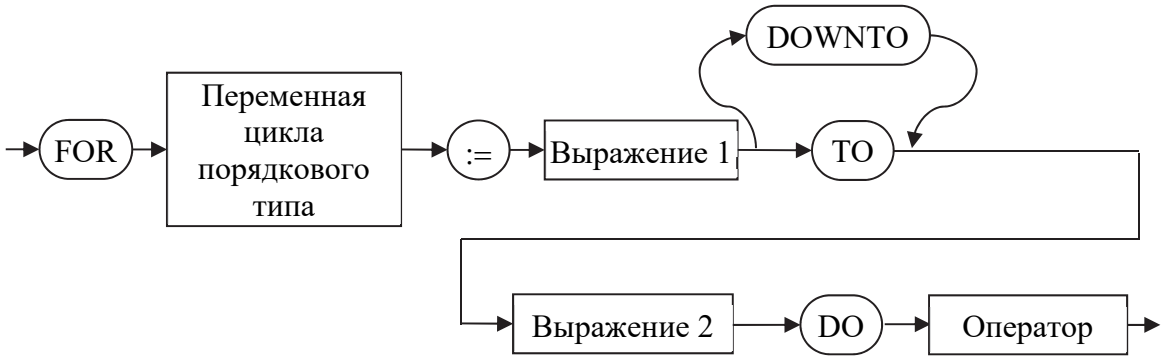
а)
```
Program My;
Var x, y, z : Integer;
Begin
 ReadLn (x, y, z);
 If (x>y) and (x>z)
  Then WriteLn(x)
  Else
    If (y>x) And (y>z)
        Then WriteLn (y)
        Else WriteLn (z);
End.
```

б)
```
Program My;
Var x, y, z : Integer;
Begin
 ReadLn (x, y, z);
 If x>y Then
   If x>z Then WriteLn(x)
          Else WriteLn(z)
 Else
   If y>z Then WriteLn(y)
          Else WriteLn(z);
End.
```

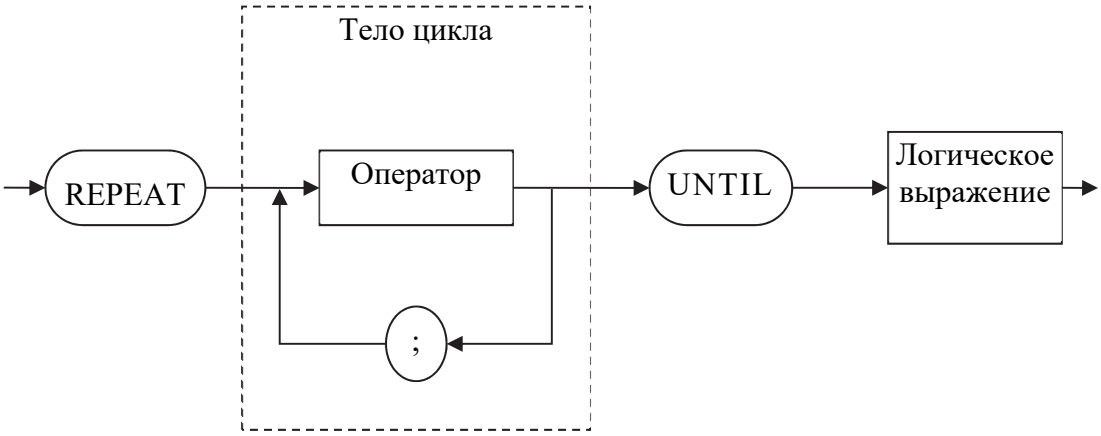
в)
```
Program My;
Var x, y, z : Integer;
Begin
 ReadLn(x, y, z);
 If (x>y) Or (x>z) Then WriteLn(x)
 Else
   If (y>x) Or (y>z) Then WriteLn(y)
   Else WriteLn(z);
End.
```

5.3. Циклы

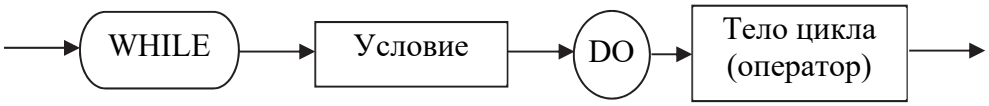
Оператор повторения FOR



Оператор повторения REPEAT



Оператор повторения WHILE



Задачи

1. Заданы 100 целых чисел, значения которых находятся в диапазоне от 1 до 50. Определите, сколько среди них чисел, первая значащая цифра в десятичной записи которых есть 1 или 2.

2. Задана некоторая целая константа из n цифр ($n \le 255$). Определите, сколько разных цифр входит в эту константу.

3. Найдите наибольшее и наименьшее значение функции $F(x) = x^3 - e^{-x} + \cos(x/5)$ при условии, что x изменяется от 0,3 до 1,8 с шагом 0,1.

4. Объясните работу следующей программы.

```
Var    i, j : integer;
  begin
   j:=0;
   for i:=1 to 10 do
     for i:=1 to 10 do
       for i:=1 to 10 do
        begin
         inc(j);
         writeln(j:8)
        end;
  end.
```

5. Подсчитайте для заданного вещественного x и натурального $n > 0$ следующие суммы:

а) $S = x + x^2 + \ldots + x^n$;

б) $S = \cos x + \cos x^2 + \ldots + \cos x^n$.

6. Подсчитайте для заданного натурального n и вещественного x суммы:

а) $S = \left(\frac{1}{1!} + \sqrt{|x|}\right) + \left(\frac{1}{2!} + \sqrt{|x|}\right) + \ldots + \left(\frac{1}{n!} + \sqrt{|x|}\right)$;

б) $S = \frac{(x-2)(x-4)\cdots(x-2n)}{(x-1)(x-3)\cdots(x-2n+1)}$.

7. По заданному значению x подсчитайте значения следующих многочленов по схеме Горнера:

а) $y = x^{10} + 2x^9 + 3x^8 + \ldots + 10x + 11$;

б) $y = 11x^{10} + 10x^9 + 9x^8 + \ldots + 2x + 1$.

8. Известно, что все `n` элементов целочисленного массива `A` не меньшие, чем –22 и не большие, чем 32. Подсчитайте, сколько раз встречается в массиве `A` каждый из элементов.

9. Вычислите:

`S:=1-x/1!+x²/2!-x³/3!+...+(-1)ⁿ xⁿ/n! (n>1).`

Варианты ответов:

а)
```
S:=x; i:=1;
repeat
 inc(i);
 S:=S+exp(i*ln(x))/i!;
until i>n;
```

б)
```
S:=1; a:=1; i:=1;
repeat
 inc(i);
 a:=-x*a/i; S:=S + a;
until i>n;
```

в)
```
S:=0; i:=1;
repeat
 S:=S+(-x)^i/i!;
 inc(i);
until i>n;
```

10. Чему будет равно значение переменной *f* в результате выполнения фрагмента алгоритма?

```
x := -24;     y := 60;
m := abs(x);  n := abs(y);
while (m > 0) and (n > 0) do
 begin
  if m > n then m :=2*(m mod n)
  else n :=2*(n mod m)
 end;
f:= m + n;
```

11. Выберите программы, в которых находится сумма цифр некоторого случайного числа.

Варианты ответов:

а)
```
Var
   s, n : comp;
begin
 s:=0;
 randomize;
 n:=random(Maxint);
 while n>0 do
  begin
   s:=s+n mod 10;
   n:=n div 10
  end;
end.
```

б)
```
Var
   s, n : comp;
begin
 s:=0;
 randomize;
 n:=random(Maxint);
 while n>0 do
  begin
   s:=s+n div 10;
   n:=n mod 10
  end;
end.
```

в)
```
Var
   s, n : integer;
begin
 s:=0; randomize;
 n:=random(Maxint);
 while n>0 do
  begin
   s:=s+n mod 10;
   n:=n div 10
  end;
end.
```

г)
```
Var
   s, n : longint;
begin
 s:=0;
 randomize;
 n:=random(Maxint);
 while n>0 do
  begin
   s:=s+n div 10;
   n:=n mod 10
  end;
end.
```

12. Какой алгоритм реализован в следующем фрагменте программы?

```
Type
    t = 2..9;
function Decimal_P(n: word; p: t): longint;
 var
    a,r: longint;
 begin
   a := 0;
   r := 1;
   while n >= 1 do
    begin
      a := a + (n mod p) * r;
      r := r * 10;
      n := n div p
    end;
   Decimal_P := a
  end;
```

13. Для некоторого заданного x подсчитайте приближенное значение бесконечной суммы с точностью $\varepsilon > 0$.

$$S = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+2}}{(2n)!}$$

14. Подсчитайте приближенное значение бесконечной суммы с точностью $\varepsilon > 0$.

$$S = \sum_{i=1}^{\infty} (-1)^i \frac{\sin \dfrac{i}{100 + i^2}}{(i+1)!}$$

15. Каким образом можно описать вычисление $S = 5!$ Выберите верные варианты:

а)
```
 S:=1;
i:=2;
while i<=5 do
 begin
   S:=S*i;
   i:=i+1
 end;
```

б)
```
S:=1;
i:=2;
repeat
  S:=S*i;
  i:=i+1
until i<5;
```

в)
```
S:=1;
i:=2;
repeat
  S:=S*i;
  i:=i+1
until i>5;
```

г)
```
S:=1;
i:=2;
while i<=5 do
  S:=S*i;
  i:=i+1;
```

д)
```
S:=1;
for i:=2 to 5 do
  S:=S*i;
```

е)
```
S:=1;
for i=2 to 5 do
  S:=S*i;
```

16. Задана вещественная матрица размера $n \times m$. Определите числа $b_1, \ldots, b_n$, равные:

а) суммам элементов строк;

б) произведениям элементов строк;

в) наименьшим значениям элементов строк;

г) значениям средних арифметических элементов строк;

д) разностям наибольших и наименьших значений элементов строк.

17. Задана вещественная матрица размера $n \times m$. Определите числа $b_1, \ldots, b_n$, где b_k – это:

а) наибольшее из значений k-й строки;

б) произведение квадратов тех элементов k-й строки, модули которых принадлежат отрезку $[-1; 1{,}5]$;

в) число отрицательных элементов k-й строки;

г) значение первого по порядку положительного элемента k-й строки;

д) модуль суммы наибольшего и наименьшего значения элементов k-й строки.

18. Задана вещественная матрица размера $n \times m$. Определите числа массива $b_1, \ldots, b_n$ из нулей и единиц, если $b_k = 1$, когда:

а) элементы k-й строки образуют возрастающую последовательность;

б) элементы k-й строки образуют либо возрастающую, либо убывающую последовательность;

в) число отрицательных элементов k-й строки меньше числа положительных элементов;

г) число ненулевых элементов k-й строки равно 1.

19. Задана вещественная квадратная матрица порядка n. Получите $x_1 x_n + x_2 x_{n-1} + \ldots + x_n x_1$, где x_k – наибольшее значение элементов k-й строки данной матрицы.

20. Все элементы с наименьшим значением в каждой строке данной целочисленной квадратной матрицы порядка n замените нулями.

Тема 6
МЕХАНИЗМЫ СТРУКТУРИРОВАНИЯ ПРОГРАММ
(6 часов. Тестирование)

План разработки алгоритмов и программ

1. Переформулируйте условие задачи на более алгоритмический вариант, если это необходимо.
2. Выясните, что на входе и что на выходе.
3. Четко определите, какие данные и сколько их необходимо для решения задачи.
4. Определите, какие результаты ожидаются после выполнения программы.
5. Составьте тестовые примеры, не забывая о «крайних случаях». Тесты должны покрывать абсолютно все возможные варианты развития событий в программе.
6. Решите тестовые примеры вручную. Опишите для каждого теста входные данные и ожидаемый результат. Составленный список впоследствии войдет в отчет.
7. Решите задачу на компьютере и проверьте ее на всех тестовых примерах.
8. Просмотрите код программы на предмет соответствия общих требований к нему (см. пункт «Стиль программирования»). При необходимости оптимизируйте и модернизируйте код.
9. Просмотрите код программы с точки зрения его эффективности (см. пункт «Эффективность программ»). При необходимости оптимизируйте и модернизируйте код.
10. После отладки программы внесите в ее текст комментарии таким образом, чтобы сторонний программист мог легко понять код вашей программы.
11. Документируйте решение задачи и оформите отчет.

Пример плана оформления отчета по заданию

1. Постановка задачи.
2. Алгоритм.
2. Графическая схема иерархии подзадач или словесное описание.
3. Система тестов.
4. Листинг программы с комментариями.
5. Итоги тестирования.

Задача 1. В текстовом файле имеются целые числа, количество которых кратно трем. Читая из данного текстового файла по три числа, проверьте, существует ли треугольник со сторонами, соответствующими полученным числам. Если ответ положительный – определите тип треугольника.

Решение. Используя сведения из геометрии, получим следующий алгоритм.

Алгоритм.

1. Организуем цикл на чтение данных из файла.
   - Заполняем массив из трех чисел, считанных из файла.
   - Вызываем подпрограмму `isTriangle` для проверки, выполняются ли для данных чисел неравенства треугольника (сумма любых двух сторон больше третьей стороны).
   - Если `isTriangle` выдала истину, определяем тип треугольника (теорема косинусов) при помощи подпрограмм `checkSides`, `checkAngles`.
   - В противном случае проверку не проводим.
2. Печатаем результат.
3. Конец цикла.

В программе используются следующие подпрограммы.

1. `function isTriangle (var s: TValues): boolean;`

Параметры: массив `s` трех величин. Возвращает `true`, если величины могут быть сторонами треугольника, и `false` – в противном случае. Здесь и далее

```
type TValues = array[1..3] of integer;
```

2. `function checkSides (var s: TValues): string;`

Параметры: массив `s` трех величин. Возвращает строку с описанием типа треугольника по отношению сторон.

3. `function checkAngles (var s: TValues): string;`

Параметры: массив `s` трех величин. Возвращает строку с описанием типа треугольника по наибольшему углу. Использует вспомогательную подпрограмму `sortSides`.

4. `procedure sortSides (var s: TValues);`

Параметры: массив `s` трех величин. Переставляет в третий элемент массива наибольшее значение. Использует вспомогательную подпрограмму `swap`.

5. `procedure swap (var a, b: integer);`

Меняет местами величины *a* и *b*.

Системы тестов

```
function istriangle (var s: TValues): boolean;
```

№	a	b	c	Результат	Пояснение
1	7	7	8	`true`	треугольник равнобедренный
2	5	6	5	`true`	треугольник равнобедренный
3	5	3	8	`false`	не существует треугольника с таким набором сторон
4	3	4	5	`true`	разносторонний прямоугольный треугольник
5	2	3	4	`true`	неравносторонний треугольник

```
procedure sortSides (var s: TValues);
```

№	a	b	c	Результат
1	5	7	5	`{5, 5, 7}`
2	8	3	5	`{5, 3, 8}`

```
function checkSides (var s: TValues): string;
```

№	a	b	c	Результат
1	7	7	7	равносторонний
2	5	5	6	равнобедренный
3	7	10	12	разносторонний

```
function checkAngles (var s: TValues): string;
```

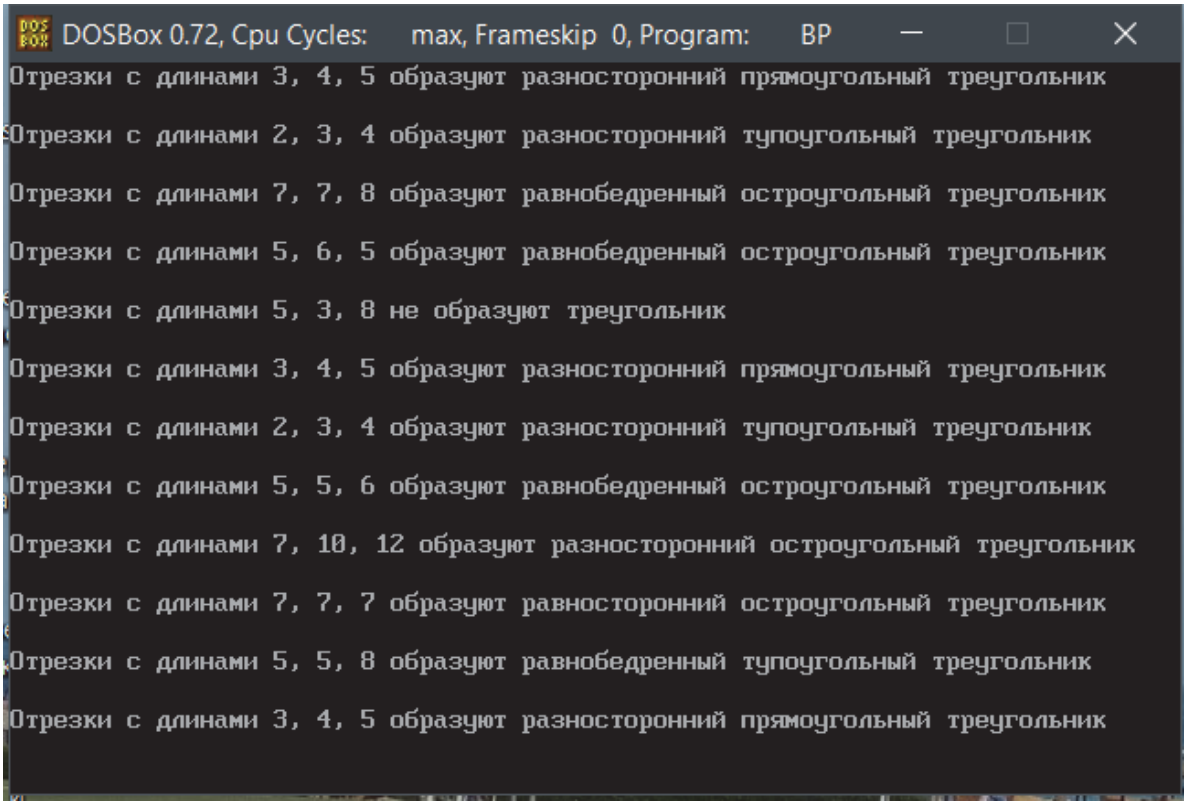
№	a	b	c	Результат
1	7	7	7	остроугольный треугольник
2	5	5	8	тупоугольный треугольник
3	3	4	5	прямоугольный треугольник

Листинг программы

```
program triangle;
uses crt;
type
     TValues = array [1..3] of integer;

function istriangle(var s: TValues): boolean;
 begin
  istriangle:=(s[1]+s[2]>s[3]) and (s[1]+s[3]>s[2]);
  istriangle istriangle and (s[3]+s[2]>s[1]);
 end;

procedure swap(var a, b: integer);
var
    tmp: integer;
 begin
     tmp := a;
     a   := b;
     b   := tmp;
 end;

procedure sortsides(var s: TValues);
begin
     if s[1]>s[3] then swap(s[1], s[3]);
     if s[2]>s[3] then swap(s[2], s[3]);
end;

function checksides(var s: TValues): string;
var
    aisb, aisc, bisc: boolean;
 begin
  aisb := s[1]=s[2];
  aisc := s[1]=s[3];
  bisc := s[2]=s[3];
  if aisb and bisc then checksides:='равносторонний '
  else
       if aisb or aisc or bisc then
               checksides :='равнобедренный '
       else
               checksides :='разносторонний ';
 end;
```

```
function checkangles(var s: TValues): string;
var
    expr: integer;
 begin
    sortsides(s);
    expr := s[3]*s[3] - s[1]*s[1] - s[2]*s[2];
    if expr>0  then
          checkangles := 'тупоугольный треугольник'
    else
         if expr<0 then
         checkangles := 'остроугольный треугольник'
        else
         checkangles := 'прямоугольный треугольник';
 end;

var
  s: TValues;
  filepath: string;
  t: text;
 begin
  clrscr;
  filepath:='testtriangle.txt';
  assign(t, filepath);
  reset(t);
  while not eof(t) do
   begin
    read (t, s[1], s[2], s[3]);
    if istriangle(s) then
      begin
       write('отрезки с длинами ',s[1],', ',
              s[2],', ',s[3],' образуют ');
       write (checksides(s));
       writeln (checkangles(s));
      end
    else
              writeln ('отрезки с длинами ',
                  s[1],', ',s[2],', ',s[3],
                 ' не образуют треугольник');
    writeln;
   end;
   readln;
 end.
```

Результаты тестирования



Задача 2. В текстовом файле имеются вещественные числа, количество которых кратно трем. Читая из текстового файла по три числа и считая их коэффициентами уравнения степени не выше третьей, решите полученное уравнение. Рассчитайте статистические сведения о характере корней уравнений и их количестве.

Решение. Имеем уравнение вида $a_2x^2 + a_1x + a_0 = 0$, у которого старший коэффициент может быть равен нулю. Используя известные сведения по решению уравнений второй и первой степеней, получим следующий алгоритм.

Алгоритм.

1. Организуем цикл на чтение данных из файла.

   - Заполняем массив из трех чисел типа
   `TValues = array [0..2] of real,`
   считанных из файла.
   - Вызываем подпрограмму `checkEq`, которая распознает вариант ответа.
   - Вызываем подпрограммы получения информации по уравнению: `getNumOfRoots`, `getDetails`, `printSign`.
   - Печатаем результат.

2. Конец цикла.

3. Вызываем подпрограмму подсчета статистики по всем решенным уравнениям: `viewStatistics`.

4. Конец.

В программе используются следующие подпрограммы.

1. `function checkEq (var a: TValues): byte;`

Параметры: массив коэффициентов уравнения `a`. Возвращает код завершения, соответствующий типу уравнения и количеству его корней.

2.
```
procedure getDetails (var a: TValues;
                var x1, x2: real; var code: byte);
```

Параметры: массив коэффициентов уравнения `a`; переменные `x1`, `x2` для хранения корней (корня) уравнения, если таковые имеются; переменная `code` для хранения типа уравнения. Использует функцию `checkEq` для получения сведений по уравнению с дальнейшим нахождением корней. Также использует вспомогательные функции `printSign`, `getNumOfRoots` для вывода информации на консоль.

3. `function printSign (var v: real): string;`

Параметры: величина `v`. Возвращает строку, состоящую из знака величины и самой величины для вывода на консоль.

4. `function getNumOfRoots (var hint: byte): string;`

Параметры: код завершения `hint`. Возвращает строку с указанием типа уравнения и количества его корней согласно значению `hint`.

5.
```
procedure viewStatistics (const codes: TCodes;
                          cases: byte);
```

Параметры: массив кодов `codes`, количество вариантов ответа при решении уравнения `cases`. Использует вспомогательную функцию `getNumOfRoots` для вывода на консоль типа уравнения, затем выводит количество уравнений соответствующего типа.

Программа тестируется на значениях, считываемых из текстового файла `testEquation.txt`.

Система тестов

1. `function checkEq (var a: TValues): byte;`

№	a_2	a_1	a_0	Результат (код завершения)
1	0	0	0	5
2	0	0	1	4
3	2	4	2	1
4	3	4	2	0
5	2	5	2	2
6	0	4	2	3
7	2	0	2	0
8	2	0	-2	2
9	2	4	0	2

2. `function getNumOfRoots (var hint: byte): string;`

№	Код	Результат (строка)
1	0	квадратное без корней
2	1	квадратное с равными корнями
3	2	квадратное с разными корнями
4	3	линейное с одним корнем
5	4	линейное без корней
6	5	с бесконечным числом корней

3. `procedure getDetails (var a: TValues;`
`var x1, x2: real; var code: byte);`

№	a_2	a_1	a_0	Результат (строка)
1	0	0	0	Уравнение 0x^2 + 0x + 0 = 0 – с бесконечным числом корней
2	0	0	1	Уравнение 0x^2 + 0x + 1 = 0 – линейное без корней
3	2	4	2	Уравнение 2 x^2 + 4 x + 2 = 0 – квадратное с 2 одинаковыми корнями x1 = -1, x2 = -1
4	3	4	2	Уравнение 3 x^2 + 4 x + 2 = 0 – квадратное без корней
5	2	5	2	Уравнение 2 x^2 + 5x + 2 = 0 – квадратное с 2 разными корнями x1 = -1, x2 = -2
6	0	4	2	Уравнение 0 x^2 + 4 x + 2 = 0 – линейное с одним корнем x = -1
7	2	0	2	Уравнение 2 x^2 + 0 x + 2 = 0 – квадратное без корней
8	2	0	-2	Уравнение 2 x^2 + 0 x – 2 = 0 – квадратное с разными корнями x1 = 1, x2 = -1
9	2	4	0	Уравнение 2 x^2 + 4 x + 0 = 0 – квадратное с разными корнями x1 = 0, x2 = -2

Примечание. Значения переменной `code` соответствуют результатам работы функции `getNumOfRoots`.

4. `procedure viewStatistics (const codes: TCodes;`
`cases: byte);`

№	Сообщение	Результат (число случаев)
1	квадратное без корней	2
2	квадратное с 2 одинаковыми корнями	1
3	квадратное с разными корнями	3
4	линейное с одним корнем	1
5	линейное без корней	1
6	с бесконечным числом корней	1

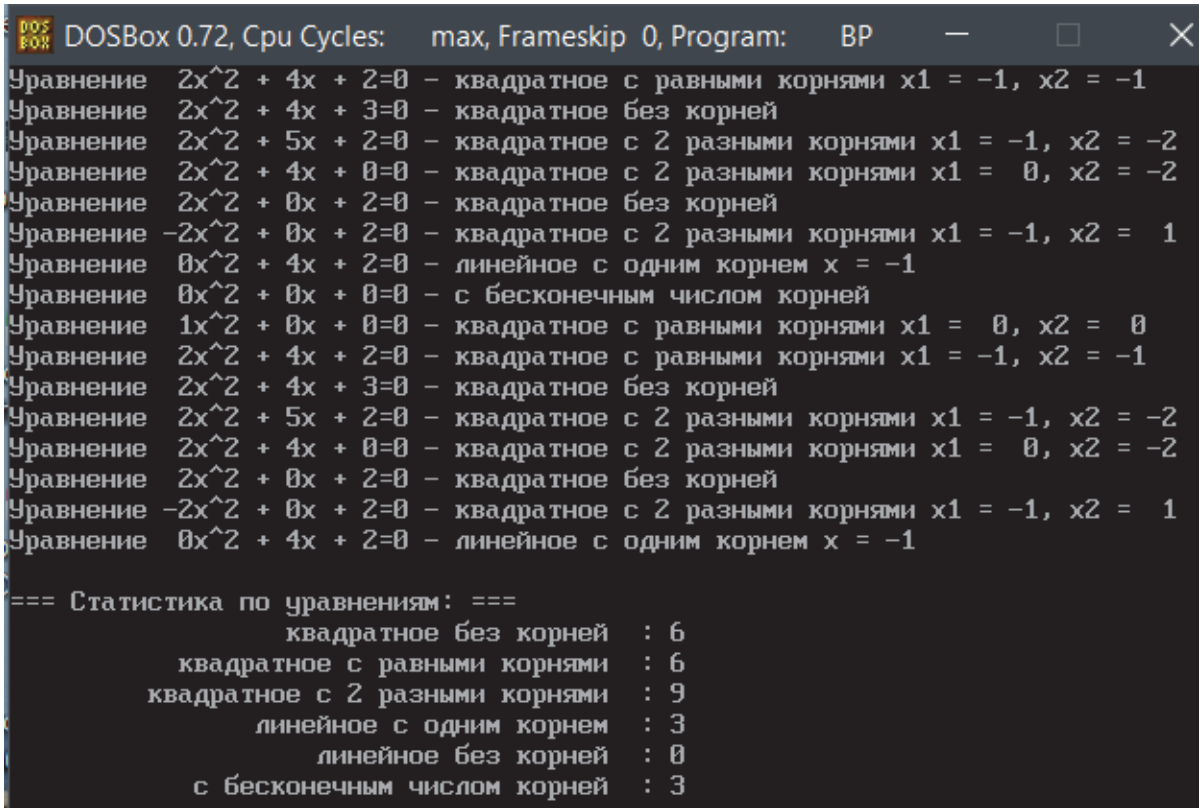
Листинг программы

```
program equation;
uses crt;
const
     cases = 6;
type
     TValues = array [0..2] of real;
     TCodes = array [0..cases-1] of word;
     {тип для хранения возможных вариантов ответов
     при решении уравнения}
function checkEq (var a: TValues): byte;
var
     D: real;
     code: byte;
begin
    if a[2] <> 0 then
      begin
       D := a[1]*a[1] - 4*a[2]*a[0];
       if D>0.0 then code := 2     // 2 разных корня
       else
         if D = 0 then code := 1   // 2 равных корня
                  else code := 0;  // нет вещ. корней
       end
    else // линейное уравнение
       if a[1]<>0 then code := 3    // один корень
       else
         if a[0]<>0 then code := 4   // нет корней
         else code := 5;         // множество корней
     checkEq := code;
end;

 function getNumOfRoots (hint: byte): string;
 begin
  case hint of
   0: getNumOfRoots:='квадратное без корней ';
   1: getNumOfRoots:='квадратное с равными корнями ';
   2: getNumOfRoots:='квадратное с разными корнями ';
   3: getNumOfRoots:='линейное с одним корнем ';
   4: getNumOfRoots:='линейное без корней ';
   5: getNumOfRoots:='с бесконечным числом корней ';
   end;
 end;
```

```
function printSign (var vel: real):string;
var
        s: string;
begin
     if vel<0 then
        begin
         str(abs(vel):2:0, s);
         printSign:= '-'+s;
        end
     else
        begin
         str(vel:2:0, s);
         printSign:= '+'+s;
        end;
 end;

procedure getDetails (var a: TValues;
           var x1, x2: real; var code: byte);
var
       roots : byte;
       info  : string;
       D     : real;
begin
    roots := checkEq(a);
    info  := getNumOfRoots(roots);
    code  := roots;
    write('Уравнение ',a[2]:2:0,'x^2 ',
      printSign(a[1]),'x ',printSign(a[0]),
      '=0 - ',info);
    if roots in [1, 2] then
      begin
       D := a[1]*a[1] - 4*a[2]*a[0];
       x1 := (-a[1]+sqrt(D))/(2.0*a[2]);
       x2 := (-a[1]-sqrt(D))/(2.0*a[2]);
       write ('x1 = ', x1:2:0, ', x2 = ', x2:2:0);
      end
     else
        if roots=3 then
         begin
           x1 := -a[0]/a[1];
           write ('x = ', x1:2:0)
         end
end;
```

```
procedure viewStatistics (const codes: TCodes;
                          cases: byte);
 var i: byte;
 begin
      writeln ('=== Статистика по уравнениям: ===');
 for i:=0 to cases-1 do
      begin
       write (getNumOfRoots(i));
       writeln (' : ', codes[i]);
      end;
  end;

var
  x1, x2   : real;
  coeffs   : TValues;
  code     : byte;
  filepath : string;
  t        : text;
  counters : TCodes;
  i        : byte;
begin
      //clrscr;
      filepath := 'testEquation.txt';
      assign(t, filepath);
      reset(t);
      while not eof(t) do
      begin
       read (t, coeffs[0], coeffs[1], coeffs[2]);
       getDetails(coeffs,x1,x2,code);
       inc(counters[code]);
       writeln;
      end;
      writeln;
      viewStatistics(counters, cases);
      readln;
end.
```

Результаты тестирования



Заключение. Все тестовые примеры подтвердились.

Задача 3. Известно, что латинский квадрат – это матрица `n×n`, в каждой строке и столбце которой уникальным образом расположены числа от `1` до `n`. Проверьте, является ли некоторая квадратная матрица при `n=9` латинским квадратом. Если нет – выведите номер строки, где было замечено первое несоответствие.

Решение. Выделим вспомогательные подпрограммы, проверяющие принадлежность элементов матрицы указанному диапазону, затем уникальности элементов в каждой строке и каждом столбце, и печати матрицы.

Алгоритм.

Способ 1. В программе используются следующие процедуры и функции.

1. `function isInRange(const a:TMatr; n:integer;`
   `var k:integer): boolean;`

Параметры: `a` – матрица, `n` – размерность, переменная `k`, в которую будет записан номер строки, если найдено первое несоответствие или значение 0. Под несоответствием понимается обнаружение элемента, лежа-

щего вне диапазона допустимых значений от 1 до n. Функция возвращает true, если ни одной такой величины не было найдено и false – в противном случае. Здесь и далее:

```
type tmatr = array [1..n, 1..n] of word;
```

2. `function isLatin(const a: TMatr; n: integer;`
`var k: integer): boolean;`

Параметры: a – матрица, n – размерность и переменная k, в которой будет храниться номер первой строки или столбца, где было замечено несоответствие уникальности элементов матрицы. Возвращает false, если критерий латинского квадрата не выполнен, true – в противном случае. Используется вспомогательная функция yeasNo.

3. `function yeasNo(i:integer): boolean;`

Параметр: индекс i, указывающий на проверяемые строку и столбец. Возвращает false, если хотя бы один элемент строки или столбца встретился более одного раза, true – в противном случае.

4. `procedure init_f(var a:Tmatr; n: integer; name: string);`

Параметры: матрица a – возвращаемое значение, количество ее строк (столбцов) n и строка name, содержащая имя исходного текстового файла.

5. `procedure showmatrix(const a: Tmatr; n: integer);`

Параметры: матрица a и количество ее строк (столбцов) n для последующей печати элементов.

Система тестов

Программа тестируется:

1) на матрице-константе:

```
const
      c:Tmatr=( (2,3,5,1,7,9,4,8,6),
                (6,4,8,2,5,3,1,9,7),
                (9,1,7,8,6,4,5,2,3),
                (7,9,2,4,3,8,6,5,1),
                (47,8,6,9,1,5,3,7,2),
                (3,5,1,7,22,6,8,4,9),
                (1,6,9,5,8,2,7,3,4),
                (5,2,3,6,4,7,9,1,8),
                (8,7,4,3,9,1,2,6,6) );
```

Ожидаемый результат: `Не латинский квадрат (выход за диапазон в строке 5)`

2) на матрице из текстового файла latin1.txt:

```
2  3  5  1  7  9  4  8  6
6  4  8  2  5  3  1  9  7
9  1  7  8  6  4  5  2  3
7  9  2  4  3  8  6  5  1
4  8  6  9  1  5  3  7  2
3  5  1  7  2  6  8  4  9
1  6  9  5  8  2  7  3  4
5  2  3  6  4  7  9  1  8
8  7  4  3  9  1  2  6  5
```

Ожидаемый результат: True

3) на матрице из текстового файла latin2.txt:

```
6  4  8  2  5  3  1  9  7
9  1  7  8  6  4  5  2  3
7  9  2  4  3  8  6  5  1
4  8  6  9  1  5  3  7  2
3  5  1  7  2  6  8  4  9
1  6  9  5  8  2  7  3  4
5  2  3  6  4  5  9  1  8
8  7  4  3  9  1  2  6  5
2  3  5  1  7  9  4  8  6
```

Ожидаемый результат: FALSE (повторение элемента «5» в строке 6)

Листинг программы

```
program latins1;
uses crt;
const n =9;
type
     TMatr = array [1..n, 1..n] of word;
     TMas  = array [1..n] of word;
var
     a, b : Tmatr;
     k    : integer;
const
     c:Tmatr=( (2,3,5,1,7,9,4,8,6),
               (6,4,8,2,5,3,1,9,7),
               (9,1,7,8,6,4,5,2,3),
               (7,9,2,4,3,8,6,5,1),
               (47,8,6,9,1,5,3,7,2),
               (3,5,1,7,22,6,8,4,9),
```

```
                (1,6,9,5,8,2,7,3,4),
                (5,2,3,6,4,7,9,1,8),
                (8,7,4,3,9,1,2,6,6) );

procedure showmatrix(const a: Tmatr; n: integer);
var
     i, j: integer;
 begin
   for i := 1 to n  do
     begin
       for j := 1 to n do
          write(a[i,j]:4);
       writeln;
     end;
 end;

procedure init_f(var a:Tmatr; n:integer;
                 name:string);
var
     t    : text;
     i, j : integer;
 begin
   assign(t, name);
   reset(t);
   for i := 1 to n do
     for  j := 1 to n do read(t, a[i,j]);
   close(t);
 end;

function isInRange(const a: TMatr; n: integer;
                   var k: integer): boolean;
var
    i, j : integer;
    flag : boolean;
 begin
   flag := true;
   i := 1;
   k := 0;
   while flag and (i <= n) do
     begin
         j := 1;
         while flag and (j <= n) do
           begin
```

```
            flag :=(a[i,j]>0) and (a[i,j] <= n);
            if not flag then k := i;
            j := j + 1;
           end;
         i := i + 1;
     end;
   Latin := flag;
 end;

function isLatin(const a: TMatr; n: integer;
                 var k: integer): boolean;
  function yeasNo(i:integer):boolean;
    var
         b1, b2 : Tmas;
         j      : integer;
         fl     : Boolean;
    begin
        k:=0;
        for j:=1 to n do
            begin
             b1[j]:=0;
             b2[j]:=0;
            end;
         j := 1;
         fl := true;
         while fl and (j <= n) do
          begin
               inc(b1[a[i,j]]);
               inc(b2[a[j,i]]);
               if (b1[a[i,j]]>1) or (b2[a[j,i]]>1) then
                    begin
                         k:=j;
                         fl:=false;
                    end;
               j := j + 1;
          end;
     yeasNo := fl;
end;
var
    i    : integer;
    flag : boolean;
begin
```

```
    i:=1;
    k:=0;
    flag := true;
    while flag and (i <= n) do
        begin
            flag := yeasno(i);
            if not flag then k:=i;
            inc(i);
        end;
   isLatin:=flag;
end;

begin
clrscr;
writeln('---------------------');
init_f(a,n,'latin1.txt');
writeln('==========');
showmatrix(a,n);
if isInRange(a, n, k)
    then  writeln (isLatin(a, n, k))
    else writeln(' не латинский квадрат');
if k>0 then
    writeln('Несоответствие в строке ', k);
readln;
init_f(b,n,'latin2.txt');
writeln('==========');
showmatrix(b,n);
if isInRange(b, n, k)
   then writeln (isLatin(b, n, k))
   else writeln(' не латинский квадрат');
if k>0 then
   writeln('Несоответствие в строке ', k);
readln;
writeln('==========');
showmatrix(c,n);
if isInRange(c, n, k)
   then writeln (isLatin(c, n, k))
   else writeln(' не латинский квадрат');
if k>0 then
   writeln('Несоответствие в строке ', k);
readln;
end.
```

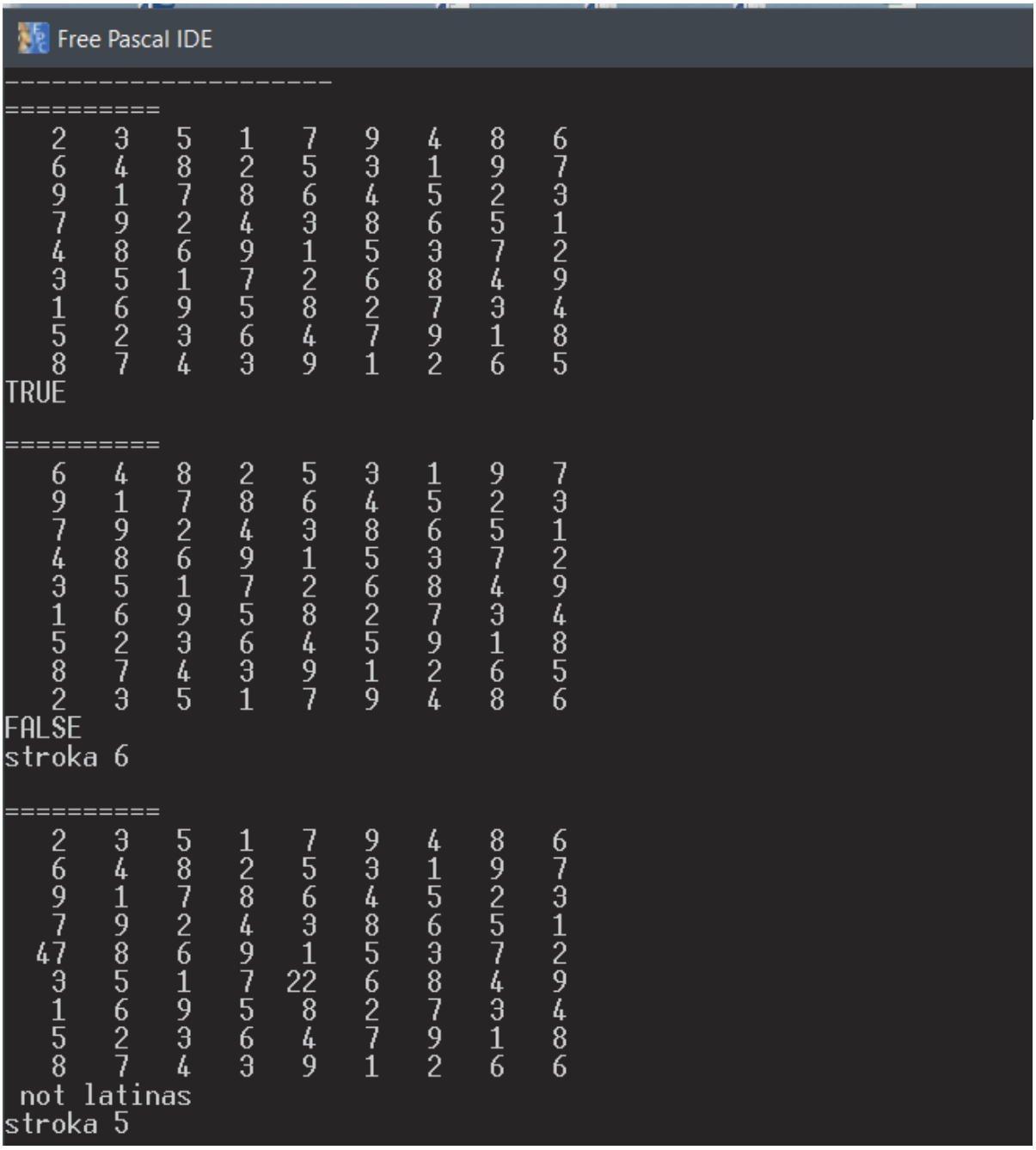
Замечание. Другим подходом к решению задачи является использование множеств.

Тип `Set` в Pascal хранит только различные элементы, поэтому изначально пустые множество-строка и множество-столбец в цикле заполняются элементами матрицы некоторой строки и столбца соответственно. Затем полученные множества сравниваются с эталонным множеством, заведомо содержащим все необходимые элементы. Ответом будет истина или ложь.

Способ 2. Листинг программы аналогичен листингу из способа 1 за исключением реализации подпрограммы `isLatin` и объявления типа `TSet`.

```
const n =9;
type
    TMat = array [1..n, 1..n] of integer;
    TSet = set of 1..n;
…
function isLatin(var a: TMat; n: integer): boolean;
var
    s, s1, s2 : TSet;
    i, j, k   : integer;
    flag      : boolean;
 begin
    s := [1..n];
    i := 1;
    flag := true;
    while flag and (i<= n) do
      begin
        s1:=[];  s2:=[];
        for j:=1 to n do
          begin
            s1 := s1 + [a[i,j]];
            s2 := s2 + [a[j,i]];
          end;
        flag := (s1=s) and (s2=s);
        inc(i);
      end;
   isLatin:=flag;
 end;
```

Результаты тестирования



Задачи

1. Имеется алгоритм перевода некоторого десятичного числа в другую систему счисления.

```
type
  t = 2..9;

function Decimal_P(n: word; p: t): longint;
var
  a,r: longint;
begin
  a := 0;  r := 1;
    while n >= 1 do begin
      a := a + (n mod p) * r;
      r := r * 10;
      n := n div p
    end;
  Decimal_P := a
end;
begin
 writeln(12,Decimal_P(12,2):12);
 readln;
end.
```

Переведите натуральное число n в систему счисления с основанием $p > 1$, сняв ограничения вида `type  t = 2..9;`

2. Пусть процедура `Сокр(a, b, p, q)` с целыми параметрами (`b<>0`) приводит дробь `a/b` к несокращаемому виду `p/q`.

- Какие из параметров этой процедуры являются исходными данными для нее, а какие – результатами?
- Какие параметры нужно объявить как параметры-значения, а какие как параметры-переменные?
- Допустимы ли обращения `Сокр(k+1, 14, n, 7)` и `Сокр(k, sqrt(36), k, n)`, где `k` и `n` – целые переменные?
- Напишите процедуру `Сокр;`
- Используйте процедуру `Сокр` для приведения выражения `1+1/2+1/3+…+1/20` к несократимому виду `c/d`.

3. Заданы два целочисленных массива разной длины.

Напечатайте числа, которые встречаются в первом массиве, но которых нет во втором.

4. Перемножьте два многочлена разных заданных степеней, если известно, что коэффициенты многочленов хранятся в массивах.

5. Задана числовая последовательность $a_1, a_2, \ldots, a_n$.

Подсчитайте сумму $S_{i,j} = a_i + a_{i+1} + \ldots + a_j$ для любых $1 \le i \le j \le n$. Определите среди этих сумм наибольшую.

6. Используя массивы $A, B, C,$ которые содержат по 50 вещественных чисел, подсчитайте:

$$t = \begin{cases} \dfrac{\min(b_i)}{\max(b_i)} + \dfrac{\max(c_i)}{\min(b_i + c_i)}, & \text{если } \min(a_i) < \max(b_i), \\ \max(b_i + c_i) + \min(c_i), & \text{иначе.} \end{cases}$$

7. Имеется описание:

```
Const
     n=15;  m=20;
type
      matr = array [1..n, 1..m] of real;
var
     a : matr;
```

Задайте функцию `sum(A)`, которая подсчитывает величину $x_1 x_n + x_2 x_{n-1} + \ldots + x_n x_1$, где x_i – максимальный элемент i-й строки матрицы A.

8. Заданы массивы $A, B, C,$ которые содержат соответственно 31, 21 и 16 вещественных чисел. Вводятся числа x, y. Подсчитайте величину:

$$\frac{(a_0 x^{30} + a_1 x^{29} + \ldots + a_{30})^2 - (b_0 y^{20} + b_1 y^{19} + \ldots + b_{20})}{c_0 (x+y)^{15} + c_1 (x+y)^{14} + \ldots + c_{15}}.$$

9. Напишите рекурсивные функции, которые находят:

а) скалярное произведение двух векторов, которые заданы своими координатами;

б) $\max(a_1, \ldots, a_n)$;

в) $S = \sum_{i=1}^{n} a_i$;

г) $P = \prod_{i=1}^{n} a_i$;

д) $n!!$;

е) x^n.

10. Задана функция:

```
Function F (n: integer): integer;
 begin
  if n>100 then F:= n-10 else F:= F(F(n+1))
 end;
```

Подсчитайте F(106), F(99) и F(85).

Докажите, что вообще $F(n) = \begin{cases} n-10, & \text{когда } n > 100, \\ 91, & \text{иначе.} \end{cases}$

11. Заданы вещественные числа a_1, …, a_{12}, s, t. Подсчитайте $P(1) - P(t) + P^2(s - t) - P^3(1)$, где

а) $P(x) = a_0 x^{12} + a_1 x^{11} + ... + a_{12}$;

б) $P(x) = a_{12} x^{12} + a_{11} x^{11} + ... + a_0$.

12. Найдите наибольший общий делитель и наименьшее общее кратное чисел a и b.

13. Известно, что астрологи делят год на 12 периодов и каждому из них ставят в соответствие один из знаков Зодиака. Найдите по заданной дате соответствующий знак Зодиака.

14. В старояпонском календаре был принят 60-годовой цикл, который состоит из пяти 12-годовых подциклов. Подциклы обозначались названиями цвета: зеленый, красный, желтый, белый, черный. Внутри каждого подцикла годы носили названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, курицы, собаки и свиньи (1984 год – год зеленой крысы – был началом очередного цикла). По заданному году подсчитайте его название по старояпонскому календарю.

СПИСОК ЛИТЕРАТУРЫ

Расолько, *Г. А*. Теория и практика программирования на Pascal / Г. А. Расолько, Ю. А. Кремень. – Минск : Выш. шк., 2015.

Расолько, *Г. А*. Теория и практика программирования на языке Pascal / Г. А. Расолько, Ю. А. Кремень. – Минск : Выш. шк., 2022.

Аляев, Ю. А. Практикум по алгоритмизации и программированию на языке Pascal : учеб. пособие / Ю. А. Аляев, В. П. Гладков, О. А. Козлов. – М. : Финансы и статистика, 2004.

Долинский, *М. С*. Алгоритмизация и программирование на Turbo Pascal: от простых до олимпиадных задач / М. С. Долинский. – СПб. : Питер, 2005.