МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тульский государственный университет»

Политехнический институт
Кафедра «Технологические системы пищевых, полиграфических
и упаковочных производств»

Утверждено на заседании кафедры
«Технологические системы пищевых,
полиграфических и упаковочных производств»
«26» января 2022 г., протокол № 6

Заведующий кафедрой
_____В.В. Прейс

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ЛАБОРАТОРНЫМ ЗАНЯТИЯМ ПО ДИСЦИПЛИНЕ (МОДУЛЮ)

«Компьютерные технологии»

основной профессиональной образовательной программы высшего образования – программы бакалавриата

по направлению подготовки
29.03.03 Технология полиграфического и упаковочного производства

с направленностью (профилем)
Технология полиграфического производства

Формы обучения: заочная

Идентификационный номер образовательной программы: 290303-01-22

Тула 2022 год

ЛИСТ СОГЛАСОВАНИЯ
методических указаний по выполнению лабораторных работ дисциплины (модуля)

Разработчик:

Проскуряков Н.Е., профессор, докт. техн. наук, профессор
(ФИО, должность, ученая степень, ученое звание)

(подпись)

СОДЕРЖАНИЕ

Лабораторная работа №1
СИСТЕМЫ СЧИСЛЕНИЯ

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель работы: усвоение вероятностного и объемного подхода к определению количества информации.

Задачи работы: усвоение способов использования формул Хартли и Шеннона, усвоение способа определения количества информации, приходящегося на один символ и на сообщение в целом.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

К вопросу об определении количества информации существуют два основных подхода.

1. Вероятностный подход.

Рассмотрим в качестве примера опыт, связанный с бросанием правильной игральной кости, имеющей N граней. Результаты данного опыта могут быть следующие: выпадение грани с одним из следующих знаков: 1, 2, ..., N.

Введем в рассмотрение численную величину $H(\alpha)$ измеряющую **энтропию (**неопределенность). Согласно формуле Шеннона:

$$H(\alpha) = -\sum_{i=1}^{N} P_i \log P_i,$$

где N - число возможных состояний системы, а P_i - вероятность того, что система находится в i-м состоянии.

В случае равновероятного выпадения каждой из граней (а в данном случае $P_i = \frac{1}{6}$) формула Шеннона переходит в формулу **Хартли**:

$$H(\alpha) = -\sum_{i=1}^{N} \frac{1}{N} \log \frac{1}{N}.$$

Так как информация кодируется числовыми кодами в той или иной системе счисления, одно и то же количество разрядов в разных системах счисления может передать разное число состояний отображаемого объекта, что можно представить в виде соотношения:

$$N = m^n,$$

где m- основание системы счисления (разнообразие символов, применяемых в алфавите); n - число разрядов (символов) в сообщении. Наиболее часто используются двоичные и десятичные логарифмы. Единицами измерения в этих случаях будут соответственно бит и дит. Очевидно, что $H(\alpha)$ будет равно единице при N = 2. Иначе говоря, в качестве единицы принимается количество информации, связанное с проведением опыта, состоящего в получении одного из двух равновероятных исходов (примером такого опыта может служить бросание монеты, при котором

возможны два исхода: «орел», «решка»). Такая единица количества информации называется «бит».Тогда для двоичной системы формула Хартли принимает вид:

$$H(\alpha) = \log_2 N$$

В случае опыта с бросанием кубика, количество информации, связанное с появлением каждой из цифр на гранях:

$$H(\alpha) = \log_2 6 \approx 2{,}58 \text{ бит.}$$

Пример. Определим количество информации, связанное с появлением каждого символа в сообщениях, записанных на русском языке. Будем считать, что русский алфавит состоит из 33 букв и знака «пробел» для разделения слов. По формуле Хартли

$$H(\alpha) = \log_2 34 \approx 5{,}09 \text{ бит.}$$

Однако в словах русского языка (равно как и в словах других языков) различные буквы встречаются неодинаково часто. Ниже приведена табл. 1.1 вероятностей частоты употребления различных знаков русского алфавита, полученная на основе анализа очень больших по объему текстов.

Таблица 1.1. Частотность букв русского языка (в процентах).

Буква алфавита	P_i	Буква алфавита	P_i	Буква алфавита	P_i
_	17.5	К	2.8	Г	1.2
О	9	М	2.6	Ч	1.2
Е,Ё	7.2	Д	2.5	Й	1
А	6.2	П	2.3	Х	0.9
И	6.2	У	2.1	Ж	0.7
Т	5.3	Я	1.8	Ю	0.6
Н	5.3	Ы	1.6	Ш	0.6
С	4.5	З	1.6	Ц	0.4
Р	4	Ь	1.4	Щ	0.3
В	3.8	Ъ	1.4	Э	0.3
Л	3.5	Б	1.4	Ф	0.2

Так как появление букв не равновероятно, то для подсчета следует использовать формулу Шеннона.

$$H(\alpha) = -\sum_{i=1}^{34} P_i \log_2 P_i \approx 4{,}72 \text{ бит.}$$

Полученное как и можно было предположить, меньше вычисленного ранее. Величина $H(\alpha)$ вычисляемая по формуле Хартли, является максимальным количеством информации, которое могло бы приходиться на один знак.

Очевидно, что для того, чтобы определить количество информации в сообщении следует умножить полученное число на количество знаков.

Аналогичным образом определяется количество информации в сообщениях на других языках.

Таблица 1.2. Частотность букв европейских языков (в процентах).

Буква алфавита	Французский язык	Немецкий язык	Английский язык	Испанский язык	Итальянский язык
_	19	14,3	18,4	17,3	15,2
A	7.68	5.52	7.96	12.90	11.12
B	0.80	1.56	1.60	1.03	1.07
C	3.32	2.94	2.84	4.42	4.11
D	3.60	4.91	4.01	4.67	3.54

E	17.76	19.18	12.86	14.15	11.63
F	1.06	1.96	2.62	0.70	1.15
G	1.10	3.60	1.99	1.00	1.73
H	0.64	5.02	5.39	0.91	0.83
I	7.23	8.21	7.77	7.01	12.04
J	0.19	0.16	0.16	0.24	0.52
K	1,5	1.33	0.41	0.07	0.83
L	5.89	3.48	3.51	5.52	5.95
M	2.72	1.69	2.43	2.55	2.65
N	7.61	10.20	7.51	6.20	7.68
O	5.34	2.14	6.62	8.84	8.92
P	3.24	0.54	1.81	3.26	2.66
Q	1.34	0.01	0.17	1.55	0.48
R	6.81	7.01	6.83	6.95	6.56
S	8.23	7.07	6.62	7.64	4.81
T	7.30	5.86	9.72	4.36	7.07
U	6.05	4.22	2.48	4.00	3.09
V	1.27	0.84	1.15	0.67	1.67
W	0,03	1.38	1.80	0.02	0,02
X	0.54	0.2	0.17	0.07	0.32
Y	0.21	1.7	1.52	1.05	0.18
Z	0.07	1.17	0.05	0.31	1.24

2. Объемный подход

В двоичной системе счисления знаки 0 и 1 называют **битами** *(bit* - от английского выражения *Binary digiTs* — двоичные цифры). В компьютере бит является наименьшей возможной единицей информации. Объем информации, записанной двоичными знаками в памяти компьютера или на внешнем носителе информации, подсчитывается просто по числу требуемых для такой записи двоичных символов. При этом, в частности, невозможно нецелое число битов (в отличие от вероятностного подхода).

Для удобства использования введены и более крупные, чем бит, единицы количества информации. Так, двоичное слово из восьми знаков содержит один **байт информации.**

1 байт (Б)=8 бит;
1 килобит (кбит) = 1000 бит
1 килобайт (КБ) = 1024 байта
1 мегабит (Мб) = 1000 килобит;
1 мегабайт (МБ) = 1024 килобайта;
1 гигабит (Гб) = 1000 мегабит;
1 гигабайт (ГБ) = 1024 мегабайта.

Между вероятностным и объемным количеством информации соотношение

неоднозначное. Далеко не всякий текст, записанный двоичными символами, допускает измерение объема информации в вероятностном (кибернетическом) смысле, но заведомо допускает его в объемном. Далее, если некоторое сообщение допускает измеримость количества информации в обоих смыслах, то это количество не обязательно совпадает, при этом кибернетическое количество информации не может быть больше объемного.

В прикладной информатике практически всегда количество информации понимается в объемном смысле.

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

1) Операционная система Windows XP;
2) Microsoft Калькулятор Плюс.

4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Определить количество информации, содержащееся в сообщении, соответствующем варианту задания, по формуле Хартли, по формуле Шеннона и с точки зрения объемного подхода. При расчете не учитывать знаки препинания.

Варианты заданий

№ варианта	Язык	Сообщение
1	Русский	В большинстве современных компьютеров проблема сначала описывается в понятном им виде (при этом вся необходимая информация как правило представляется в двоичной форме — в виде единиц и нулей, хотя существовали и компьютеры на троичной системе счисления), после чего действия по её обработке сводятся к применению простой алгебры логики.
2	Английский	Information science, in studying the collection, classification, manipulation, storage, retrieval and dissemination of information has origins in the common stock of human knowledge. Information analysis has been carried out by scholars at least as early as the time of the Abyssinian Empire with the emergence of cultural depositories, what is today known as libraries and archives.
3	Немецкий	„Informationswissenschaft untersucht das Auswerten/Selektieren, Erschlieben, Bereitstellen/Wiederverwerten, Suchen, Vermitteln und Finden von relevantem (vorwiegend digital vorliegendem) Wissen, durch Informations- und Kommunikationsprozesse.“
4	Французский	La science de l'information etudie l'application et l'usage des connaissances dans les organisations, et l'interaction entre les gens, les organisations et les systèmes d'information. Cette science est un domaine interdisciplinaire qui touche e la fois l'informatique, mais aussi les sciences de l'information et des bibliothèques, les technologies de l'information et de la communication, les sciences cognitives et les sciences sociales.
5	Испанский	En sentido restringido, la documentacion como ciencia documental se podria definir (a grandes rasgos) como la ciencia del procesamiento de la informacion. Integradora y globalizadora, se trata de una ciencia enriquecedora y generalista, de ambito multidisciplinar o interdisciplinar. La ciencias de la documentacion engloban, segun la mayoria de los

		autores: la biblioteconomia, la archivística, la documentacion y la museologia.
6	Итальянский	L'implementazione fisica di questo concetto e variata con il progredire della tecnologia, e sono esistiti computer profondamente diversi dal punto di vista del meccanismo di funzionamento (meccanici, elettromeccanici ed elettronici), della modalita di rappresentazione delle informazioni (analogica e digitale) e di altre loro peculiarita (architettura interna, programmabilita, ecc.).
7	Русский	Начинающие пользователи и особенно дети зачастую с трудом воспринимают идею того, что компьютер — просто машина и не может самостоятельно «думать» или «понимать» те слова, которые он показывает. Компьютер лишь механически отображает заданные программами точки, линии и цвета при помощи устройств ввода-вывода.
8	Английский	Documentalists emphasized the utilitarian integration of technology and technique toward specific social goals. According to Ronald Day, "As an organized system of techniques and technologies, documentation was understood as a player in the historical development of global organization in modernity – indeed, a major player inasmuch as that organization was dependent on the organization and transmission of information."
9	Немецкий	Bearbeitung der Dokumente, so dass der Informationsinhalt (Content) optimal strukturiert, leicht auffindbar und gut lesbar im Dokumentenspeicher (verwaltet) abgelegt ist. Dies geschieht durch thematische Informationsfilter wie die Inhaltserschließung (Indexing), die mit verschiedenen Dokumentationssmethoden wie Schlagwortmethode, Thesauri und Klassifikationen zur Einspeisung der Dokumente in die Dokumentspeicher arbeitet.
10	Французский	Abregee en RI ou IR (Information Retrieval en anglais), la recherche d'information est la science qui etudie la maniere de repondre pertinemment a une requete en retrouvant de l'information dans un corpus. Celui-ci est composes de documents d'une ou plusieurs bases de donnees, qui sont decrits par un contenu ou les métadonnées associees. Les bases de donnees peuvent etre relationnelles ou non structurees, telles celles mises en reseau par des liens hypertexte comme dans le World Wide Web, l'internet et les intranets. Le contenu des documents peut etre du texte, des sons, ses images ou des donnees.
11	Испанский	Cada ciencia documental tiene una larga historia, pero la mas antigua es sin duda la archivistica. Segun el pais, se trata de unos estudios universitarios con titulacion superior (existente en dos ciclos antes de la Convergencia Europea, por ejemplo en Japon), con un nombre u otro, pero que tambien se imparte en centros

		de enseñanza privados desde hace anos. En America Latina, la carrera profesional suele denominarse "Bibliotecología y Ciencias de la Informacion".
12	Итальянский	In questa forma e al pari della televisione, esso rappresenta il mezzo tecnologico simbolo che più ha modificato le abitudini umane dal secondo dopoguerra ad oggi: la sua invenzione ha contribuito alla nascita e allo sviluppo dell'informatica moderna, che ha segnato l'avvento della cosiddetta terza rivoluzione industriale e della societa dell'informazione.

5. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1) Ознакомиться с содержанием лабораторной работы.

2) Определить общее число значащих символов в сообщении.

3) Определить количество информации, связанное с появлением каждого символа в сообщении при условии их равновероятной частотности по формуле Хартли.

4) Определить общее количество информации в сообщении при условии равновероятной частотности символов.

5) Определить количество информации, связанное с появлением каждого символа в сообщении при условии их разновероятной частотности по формуле Шеннона.

6) Определить общее количество информации в сообщении при условии разновероятной частотности символов.

7) Исходя из того, что для кодирования одного знака требуется 1 байт определить количество информации в сообщении с точки зрения объемного подхода. Перевести полученное значение в килобиты и килобайты.

8) Сравнить полученные значения и сделать вывод.

9) Подготовиться к ответам на контрольные вопросы.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) В чем разница между формулами Шеннона и Хартли?

2) Какое количество информации содержится в однократном броске монеты? В трехкратном?

3) Какое количество информации в сообщении будет больше – вычисленное по формуле Хатрли или по формуле Шеннона?

4) В чем отличие вероятностного и объемного подходов?

5) Какие единицы измерения информации вы знаете?

Лабораторная работа №2
АЛГОРИТМЫ ЛИНЕЙНОЙ СТРУКТУРЫ

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель работы: получение навыков построения алгоритмов линейной структуры.

Задачи работы: изучение методов разработки и программирования алгоритмов с линейной структурой.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Операторы языка программирования Turbo Pascal, представленные в нужной последовательности, позволяют реализовать алгоритм решения задачи. При этом каждый из операторов выполняет некоторое действие над данными.

Операторы Turbo Pascal подразделяются на две группы: простые и структурированные. Простые операторы не содержат в себе других операторов; структурированные включают в себя другие операторы - как простые, так и структурированные. (В последнем случае в состав операторов могут входить и другие операторы, иными словами, возможно множество уровней вложенности операторов.)

К простым операторам относятся оператор присваивания, оператор перехода и пустой оператор, а к структурированным - составной оператор, условные операторы (`If` и `Case`) и операторы цикла (`While, Repeat` и `For`).

1. Структура программы Turbo Pascal.

Программа на языке Pascal состоит из разделов:

1) Раздел объявления меток;
2) Раздел объявления констант;
3) Раздел объявления типов;
4) Раздел объявления переменных;
5) Раздел объявления процедур и функций;
6) Раздел инструкций программы.

Структура программы в общем виде выглядит следующим образом:

```
Label
{ объявления меток }
Const
{ объявления констант )
Type
{ объявления типов }
Var
{ объявления переменных }
{ объявления процедур и функций программиста }
Begin
{ инструкции основной программы}
End.
```

2. Объявление переменных

Следует помнить, что каждая переменная программы должна быть объявлена. Инструкция объявления переменной выглядит так:

```
ИмяПеременной:Тип;
```

В имени переменной можно использовать буквы латинского алфавита и цифры (первым символом должна быть буква).

К основным типам данных языка Pascal относятся:

- целые числа (integer и др.);
- действительные числа (real и др.);
- символы (char);
- строки (string);
- логический (boolean).

3. Оператор присваивания.

Знак присваивания `:=` делит этот оператор на две части. В правой части представлено выражение, состоящее из идентификаторов констант, переменных, функций и знаков операций, которое необходимо вычислить. После вычисления полученное значение присваивается переменной, указанной в левой части оператора. При этом тип переменной должен быть совместим с типом вычисленного выражения.

Пример

```
a:=b+c;
a:='Ь';
new:=234;
x:=true;
```

4. Составной оператор.

Составной оператор представляет собой последовательность некоторых операторов, выполняющихся в том порядке, в каком они представлены в тексте программы. Зарезервированные слова `Begin` и `End` являются так называемыми операторными скобками, в которые заключены операторы, входящие в составной оператор. Когда, в зависимости от определенного условия, требуется обеспечить последовательное выполнение некоторого набора операторов, без составного оператора не обойтись.

Составной оператор имеет вид

```
begin s1; s2; … sn end;
```

Здесь S1-Sn - операторы, образующие составной оператор; зарезервированные слова `Begin` и `End` - операторные скобки. Частный случай составного оператора – раздел операторов (или тело) любой программы на Turbo Pascal. Составной оператор может включать другие составные операторы (т.е. составным может быть любой из операторов s1-sn), причем допускается любой уровень вложенности.

5. Вывод данных.

Операторы `Write` и `Writeln` предназначены для вывода на экран монитора сообщений и значений переменных. Одна инструкция `Write` (`Writeln`) может вывести на экран значения нескольких переменных и (или) несколько сообщений;

```
Write(a1,a2,...ak);
```

Реализует вывод значений переменных в строку экрана.

```
Write(a1:6:2,a2:6:2,...ak:6:2);
```

Реализует вывод значений переменных в строку экрана, причем для каждой из них выводится 6 знаков, 2 из них после запятой.

```
Write('список значений',a1,a2,...ak);
```

Реализует вывод на экран надписи «Список значений» и переменных в строку экрана.

```
WriteLn(a1,a2,...ak);
```

Реализует вывод значений переменных в строку экрана и переход к началу следующей строки.

```
WriteLn;
```

Оператор `Writeln` без параметров переводит курсор в начало следующей строки экрана.

Стандартная библиотека `Crt` (для ее использования в текст программы нужно включить директиву `Uses Crt`), содержит процедуры, используя которые, можно задать цвет фона и цвет символов, выводимых операторами write и writeln.

```
TextBackGround(Red);
```

Устанавливает красный цвет символов.

```
Textcolor(LightBlue);
```

Устанавливает светло-голубой цвет фона.

При использовании этих процедур цвет можно задать с помощью именованной или целой константы;

```
ClrScr;
```

Очищает экран.

При выводе каких либо данных следует в конце программы добавить оператор readln без параметров – для того чтобы информация не исчезла с экрана, либо добавить строку

```
Repeat Until KeyPressed;
```

И в том и в другом случае данные останутся на экране, пока не будет нажата какая-либо клавиша.

Пример

Приведем программу, которая выводит на экран фразу: Каждый охотник желает знать, где сидят фазаны, позволяющую запомнить порядок следования цветов радуги (первая буква слова кодирует цвет: каждый — красный, охотник — оранжевый, желает — желтый, знать — зеленый, где — голубой, сидят — синий, фазаны — фиолетовый). Каждое слово фразы должно быть выведено наиболее подходящим цветом.

```
Program RADUGA;
Uses Crt;
Begin
TextBackGround(Black);
ClrScr;
TextColor(Red);
write('Каждый');
TextColor(LightRed);
write('охотник ');
TextColor(Yellow);
write('желает ');
TextColor(Green);
write('знать ');
TextColor(LightBlue);
write('где ');
TextColor(Blue);
write('сидит ');
TextColor(Magenta);
write('фазан.');
Repeat Until KeyPressed;
End.
```

6. Ввод данных.

Для ввода исходных данных используются операторы процедур ввода.

`Read(a1,a2,...aK);`

Реализует чтение K значений вводимых данных и присваивает эти значения переменным a1,a2,...ak. Используя один оператор `Readln`, можно ввести значения нескольких переменных.

В случае несоответствия типа введенных данных типу переменной, значение которой вводится с клавиатуры, программа завершает работу и на экран выводится сообщение `Error: invalid numeric format` (если программа запущена из среды разработки, т. е. из Turbo Pascal) или Run time error (если программа запущена из операционной системы).

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

1) Операционная система Windows XP;
2) Microsoft Калькулятор Плюс;
3) Turbo Pascal 7.0 или ABC Pascal.

4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Заданием на лабораторную работу является написание программы, реализующей алгоритм решения поставленной задач на языке Turbo Pascal.

Варианты заданий

№ варианта	Задание
1	Дана длина ребра куба. Найти объем куба и площадь его боковой поверхности.
2	Даны два действительных числа. Найти среднее арифметическое и среднее геометрическое этих чисел.
3	Даны катеты прямоугольного треугольника. Найти его гипотенузу и площадь.
4	Найти площадь равнобочной трапеции с основаниями a и b и углом α при большем основании a.
5	Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.
6	Треугольник задан длинами сторон. Найти радиусы вписанной и описанной окружностей.
7	Вычислить расстояние между двумя точками с координатами x_1, y_1 и x_2, y_2.
8	Треугольник задан координатами своих вершин. Найти периметр треугольника.
9	Треугольник задан координатами своих вершин. Найти площадь треугольника.
10	Прямоугольник задан координатами своих вершин. Найти периметр прямоугольника.
11	Прямоугольник задан координатами своих вершин. Найти площадь прямоугольника.
12	Даны длины сторон параллелограмма a и b и углом α между ними. Найти площадь параллелограмма.

5. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1) Прочитать задание и выписать все переменные, которые следует использовать при его решении.
2) Продумать порядок решения задачи и составить алгоритм ее решения.
3) Составить список операторов, которые понадобятся при написании программы.
4) Описать в разделе описаний все используемые переменные и их типы.
5) Составить программу решения задачи.
6) Проверить правильность программы путем введения данных, результат решения для которых известен.
7) Если необходимо, отладить программу.
8) Оформить ввод и вывод данных удобочитаемым образом.
9) Сохранить результат, как «lr02.pas»

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) Расскажите об алгоритмах линейной структуры.
2) Чем отличается оператор присваивания от проверки равенства?
3) Какие операторы ввода вы знаете?
4) Какие операторы вывода вы знаете?
5) Что такое объявление переменных?

Лабораторная работа №3
АЛГОРИТМЫ РАЗВЕТВЛЯЮЩЕЙСЯ СТРУКТУРЫ

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель работы: усвоение принципов построения алгоритмов разветвленной структуры.

Задачи работы: ознакомиться с понятием алгоритм, его свойствами и способами представления. Изучить основные положения ГОСТ 19.701-90 (ИСО 5807-85), связанные с изображением схем программ (графическим представлением алгоритма). Изучить построение линейных и разветвляющихся алгоритмов.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Правила выполнения изображения схем алгоритмов (ГОСТ 19.701-90) (ИСО 5807-85).

Алгоритм - конечная последовательность точно определенных действий, приводящих к однозначному решению поставленной задачи.

Алгоритм должен обладать такими свойствами как:

- массовость (универсальность);
- определенность (детерминированность);
- правильность (адекватность);
- поэтапность (дискретность).

Алгоритмы могут быть заданы:

- словесно, с помощью слов и предложений естественного языка;
- таблично, в форме таблиц и расчетных формул;
- графически, с помощью специальных символов - блоков.

Описание алгоритмов с помощью блок-схем - наиболее наглядный и распространенный способ задания алгоритмов.

Условные обозначения и правила выполнения изображения схем алгоритмов изложены в ГОСТ 19.701-90 (ИСО 5807-85).

Стандарт не распространяется на форму записей и обозначений, помещаемых внутри символов или рядом с ними и служащих для уточнения выполняемых ими функций.

Требования стандарта являются обязательными. Схемы алгоритмов состоят из имеющих заданное значение символов, краткого пояснительного текста и соединяющих линий. Схемы могут использоваться на различных уровнях детализации, причем число уровней зависит от размеров и сложности задачи обработки данных. Уровень детализации должен быть таким, чтобы различные части и взаимосвязь между ними были понятны в целом.

В стандарте используются следующие понятия:

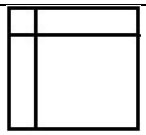
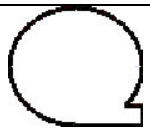
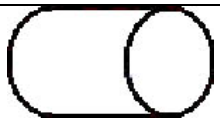
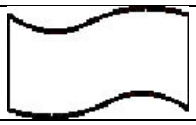
1) основной символ - символ, используемый в тех случаях, когда точный тип (вид) процесса или носителя данных неизвестен или отсутствует необходимость в описании фактического носителя данных;

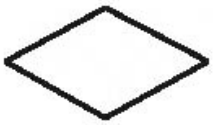
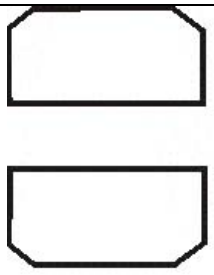
2) специфический символ - символ, используемый в тех случаях,

когда известен точный тип процесса или носителя данных или когда необходимо описать фактический носитель данных;

3) схема - графическое представление определения, анализа или метода решения задачи, в котором используются символы для отображения операций, данных, потока, оборудования и т. д.

2. Описание символов.

	1. Символы данных
	1.1. Основные символы данных
	1.1.1. Данные. Символ отображает данные, носитель данных не определен.
	1.1.2. Запоминаемые данные. Символ отображает хранимые данные в виде, пригодном для обработки, носитель данных не определен.
	1.2. Специфические символы данных
	1.2.1. Оперативное запоминающее устройство. Символ отображает данные, хранящиеся в оперативном запоминающем устройстве.
	1.2.2. Запоминающее устройство с последовательным доступом. Символ отображает данные, хранящиеся в запоминающем устройстве с последовательным доступом (магнитная лента, кассета с магнитной лентой, магнитофонная кассета).
	1.2.3. Запоминающее устройство с прямым доступом. Символ отображает данные, хранящиеся в запоминающем устройстве с прямым доступом (магнитный диск, магнитный барабан, гибкий магнитный диск).
	1.2.4. Документ. Символ отображает данные, представленные на носителе в удобочитаемой форме (машинограмма, документ для оптического или магнитного считывания, микрофильм, рулон ленты с итоговыми данными, бланки ввода данных).
	1.2.5. Ручной ввод. Символ отображает данные, вводимые вручную во время обработки с устройств любого типа (клавиатура, переключатели, кнопки, световое перо, полоски со штриховым кодом).
	1.2.6. Карта. Символ отображает данные, представленные на носителе в виде карты (перфокарты, магнитные карты, карты со считываемыми метками, карты со сканируемыми метками).
	1.2.7. Бумажная лента. Символ отображает данные, представленные на носителе в виде бумажной ленты.
	1.2.8. Дисплей. Символ отображает данные, представленные в человекочитаемой форме на носителе в виде отображающего

	устройства (экран для визуального наблюдения, индикаторы ввода информации).
	2. Символы процесса
	2.1. Основные символы процесса
	2.1.1. Процесс. Символ отображает функцию обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации или к определению, по которому из нескольких направлений потока следует двигаться).
	2.2. Специфические символы процесса
	2.2.1. Предопределенный процесс. Символ отображает предопределенный процесс, состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте (в подпрограмме, модуле).
	2.2.2. Ручная операция. Символ отображает любой процесс, выполняемый человеком.
	2.2.3. Подготовка. Символ отображает модификацию команды или группы команд с целью воздействия на некоторую последовательную функцию (установка переключателя, модификация индексного регистра или инициализация программы).
	2.2.4. Решение. Символ отображает решение или функцию переключаемого типа, имеющую один вход и ряд альтернативных выходов, один из которых может быть активизирован после вычисления условий, определенных внутри этого символа.
	2.2.5. Параллельные действия. Символ отображает синхронизацию двух или более параллельных операций.
	2.2.6. Граница цикла. Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т.д. помещаются внутри символа в начале или конце в зависимости от расположения операции, проверяющей условие.
	3. Символы линий
	3.1. Основной символ линий
	3.1.1. Линия. Символ отображает поток данных или управления.
	3.2. Специфические символы линий

	3.2.1. Передача управления. Символ отображает непосредственную передачу управления от одного процесса к другому, иногда с возможностью прямого возвращения к инициирующему процессу после того, как инициированный процесс завершит свои функции. Тип передачи управления должен быть назван внутри символа (например, запрос, вызов, событие).
	3.2.2. Канал связи. Символ отображает передачу данных по каналу связи.
	3.2.3. Пунктирная линия. Символ отображает альтернативную связь между двумя или более символами. Кроме того, символ используют для обведения аннотированного участка.
	4. Специальные символы
	4.1. Соединитель. Символ отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линий и продолжения ее в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение.
	4.2. Терминатор. Символ отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы, внешнее использование и источник или пункт назначения данных).
	4.3. Комментарий. Символ используют для добавления описательных комментариев или пояснительных записей в целях объяснения или примечаний.
	4.4. Пропуск. Символ (три точки) используют схемах для отображения пропуска символа или группы символов, в которых не определены ни тип, ни число символов. Символ используют только в символах линий или между ними. Он применим равным образом в схемах, изображающих общие решения с неизвестным числом повторений.

3. Правила применения символов.

1) Символ предназначен для графической идентификации функции, которую он отображает, независимо от текста внутри этого символа.

2) Символы в схеме должны быть расположены равномерно. Следует придерживаться разумной длины соединений и минимального числа длинных линий.

3) Формы символов, установленные настоящим стандартом, должны служить руководством для фактически используемых символов. Не должны изменяться углы и другие параметры, влияющие на соответствующую форму символов. Символы должны быть, по возможности, одного размера.

4) Символы могут быть вычерчены в любой ориентации, но, по возможности, предпочтительной является горизонтальная ориентация.

5) Минимальное количество текста, необходимо для понимания функции данного символа, следует помещать внутри данного символа. Текст для чтения должен записываться слева направо и сверху вниз независимо от направления потока.

6) Если объем текста, помещаемого внутрь символа, превышает его размеры, следует использовать символ комментария.

7) В схемах может использоваться идентификатор символов. Это связанный с данным символом идентификатор, который определяет символ для использования в справочных целях в других элементах документации (например, в листинге программы). Идентификатор символа должен располагаться слева над символом.

8) В качестве первого и последнего символа алгоритма должен быть использован символ указателя конца.

4. Правила выполнения соединений.

1) Потоки данных или потоки управления в схемах показываются линиями. Направление потока слева на право и сверху вниз считаются стандартным. В случаях, когда необходимо внести большую ясность в схему (например, при соединениях), на линиях используют стрелки. Если поток имеет направление, отличное от стандартного, стрелки должны указывать это направление.

2) В схемах следует избегать пересечение линий. Пересекающиеся линии не имеют логической связи между собой, поэтому изменения направления в точках пересечения не допускаются.

3) Две или более входящие линии могут объединяться одну исходящую линию. Если две или более линий объединяются в одну линию, место объединения должно быть смещено.

4) Линии в схемах должны подходить к символу либо слева, либо сверху, а исходить либо справа, либо снизу. Линии должны быть направлены к центру символа.

5) При необходимости линии в схемах следует разрывать для избежания излишних пересечений или слишком длинных линий, а также, если схема состоит из нескольких страниц. Соединитель в начале разрыва называется внешним соединителем, а соединитель в конце разрыва - внутренним соединителем.

6) Ссылки к страницам могут быть приведены совместно с символом комментария для их соединителей.

5. Применение символов.

Символ	Наименование символа	1	2	3	4	5
Символы данных						
Основные	Данные	+	+	+	+	+
	Запоминаемые данные	+	-	+	+	+
	Специфические ОЗУ	+	-	+	+	+
	ЗУ с послед. выборкой	+	-	+	+	+
	ЗУ с прямым доступом	+	-	+	+	+
	Документ	+	-	+	+	+
	Ручной ввод	+	-	+	+	+
	Карта	+	-	+	+	+
	Бумажная лента	+	-	+	+	+
	Дисплей	+	-	+	+	+

Символы процесса						
Основные	Процесс	+	+	+	+	+
Специфические	Предопределенный процесс	-	+	+	+	-
	Ручная операция	+	-	+	+	-
	Подготовка	+	+	+	+	-
	Решение	-	+	+	-	-
	Параллельные действия	-	+	+	+	-
	Граница цикла	-	+	+	-	-
Символы линий						
Основные	Линия	+	+	+	+	+
Специфические	Передача управления	-	-	-	+	-
	Канал связи	+	-	+	+	+
	Пунктирная линия	+	+	+	+	+
Специальные символы	Соединитель	+	+	+	+	+
	Терминатор	+	+	+	-	-
	Комментарий	+	+	+	+	+
	Пропуск	+	+	+	+	+

Примечание. Знак "+" указывает, что символ используют в данной схеме, знак "-" - не используют.

1 - Схема данных;
2 - Схема программы;
3 - Схема работа системы;
4 - Схема взаимодействия программ;
5 - Схема ресурсов системы;
ОЗУ - оперативное запоминающее устройство;
ЗУ - запоминающее устройство.

5. Примеры построения алгоритмов.

Алгоритмы бывают: линейные, разветвляющиеся, циклические. Линейный алгоритм не содержит логических условий, имеет одну ветвь обработки и изображается линейной последовательностью связанных друг с другом блоков. Разветвляющийся алгоритм содержит одно или несколько логических условий и имеет несколько ветвей обработки. Разветвляющиеся алгоритмы могут иметь несколько структур:

- неполная альтернатива, обработка производится при выполнении условия в противном случае обработка не производится;

- полная альтернатива, обработка производится при выполнении условия по ветви 1, в противном случае по ветви 2;

- конструкция выбора, обработка производится при выполнении одного из нескольких различных условий по соответствующей ему ветви.

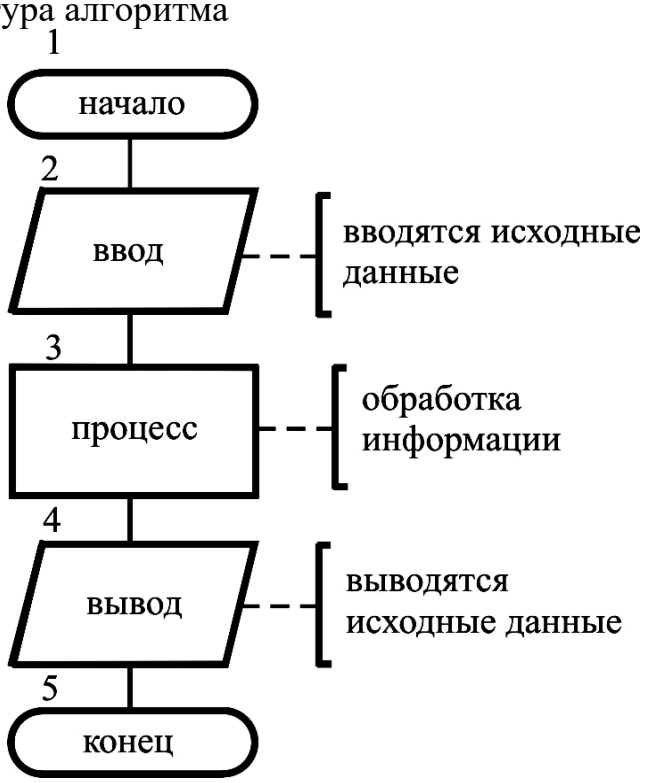
Блок - Решение имеет один вход и несколько выходов, которые следует показывать:

1) несколькими линиями от данного символа к другим символам;

2) одной линией от данного символа, которая затем разветвляется в соответствующее число линий.

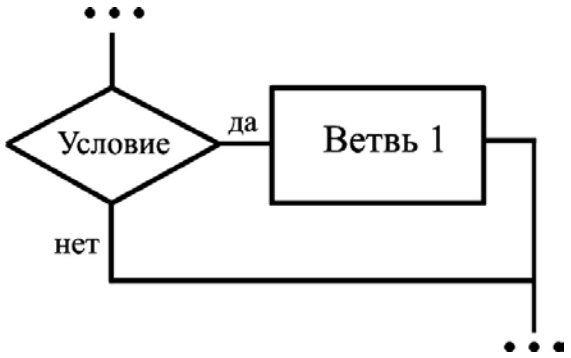
3) каждый выход из символа должен сопровождаться соответствующими значениями условий, чтобы показать логический путь, который он представляет, с тем, чтобы эти соответствующие ссылки были идентифицированы.

Линейная структура алгоритма



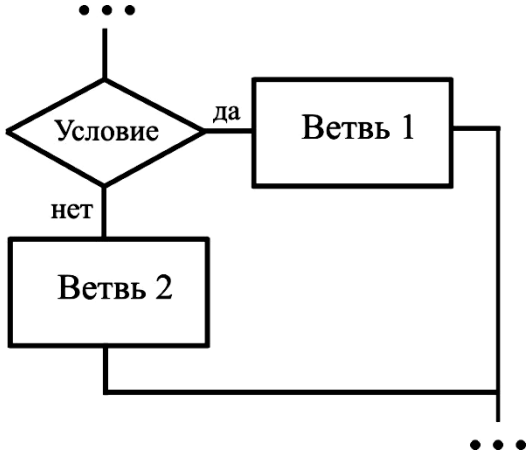
Разветвляющиеся структуры алгоритмов

а) неполная альтернатива



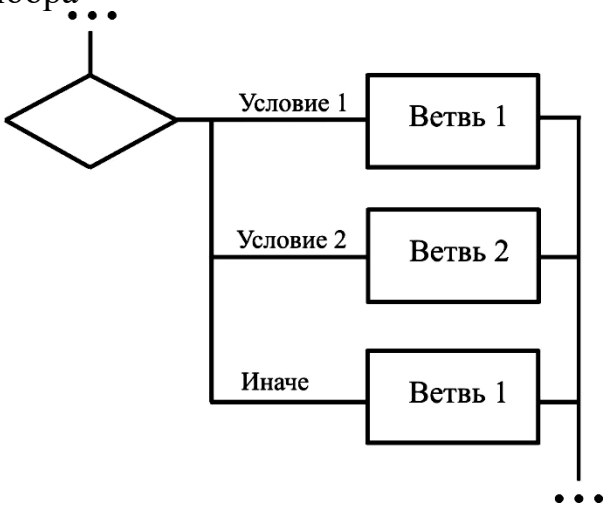
Если условие выполняется выполнить обработку информации по ветви 1.

б) полная альтернатива



Если условие выполняется выполнить обработку информации по ветви 1, иначе по ветви 2.

в) конструкция выбора



Если выполняется условие 1, то выполняется обработка по ветви 1, если выполняется условие 2, то выполняется обработка по ветви 2, если выполняется условие 3, то выполняется обработка по ветви 3, иначе выполняется обработка по ветви 4.

6. Символы, рекомендованных к использованию в данной работе.

Данные. Символ отображает данные, носитель данных не определен.

Процесс. Символ отображает функцию обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации или к определению, по которому из нескольких направлений потока следует двигаться).

Решение. Символ отображает решение или функцию переключаемого типа, имеющую один вход и ряд альтернативных выходов, один из которых может быть активизирован после вычисления условий, определенных внутри этого символа.

Линия. Символ отображает поток данных или управления.

Соединитель. Символ отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линий и продолжения ее в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение.

Терминатор. Символ отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы, внешнее использование и источник или пункт назначения данных).

Комментарий. Символ используют для добавления описательных комментариев или пояснительных записей в целях объяснения или примечаний.

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

1) Операционная система Windows XP;

2) Microsoft Калькулятор Плюс.

4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

В соответствии с номером варианта найти значение дискретной функции, заданной на нескольких интервалах. Разработать алгоритм с разветвляющейся структурой.

Варианты заданий

n	задание	n	задание
1	$y=\begin{cases}(x+1)^4+1, если\ x<-1\\1, если\ -1\le x<1\\1-(x-1)^4, если\ 1\le x\end{cases}$	2	$y=\begin{cases}11-7\sqrt{\lvert x-1\rvert}, если\ x<-3\\x, если\ -3\le x<3\\7\sqrt{x+1}+11, если\ x\ge 3\end{cases}$
3	$y=\begin{cases}e^{x-1}, если\ x\le 1\\lg(x+2), если\ x>1\end{cases}$	4	$y=\begin{cases}\ln x, если\ x\ge 3\\e^x, если\ 2\le x<3\\x, если\ x<2\end{cases}$
5	$y=\begin{cases}sin^2(x+0.1), если x<0.2\\1, если 0.2\le x\le 0.3\\cos^2(x+0.1), если x>0.3\end{cases}$	6	$y=\begin{cases}(x+1)^4, если\ x<-1\\1-cos(\pi x), если\ -1\le x<1\\-(x-1)^2, если\ x\ge 1\end{cases}$
7	$y=\begin{cases}1, если\ x<0\\2+cos(\pi x), если\ 0\le x<0.4\\1, если\ x\ge 0.4\end{cases}$	8	$y=\begin{cases}1+\cos x, если\ x<0.2\\1, если\ 0.2\le x<0.4\\-1, если\ x\ge 0.4\end{cases}$
9	$y=\begin{cases}0, если\ x\le 1\\x\ln x, если\ 1<x<3\\x^x, если\ x\ge 3\end{cases}$	10	$y=\begin{cases}4x^2+2x-8, если\ x<3\\1, если\ 3\le x<3.5\\x^3-x+10, если\ x\ge 3.5\end{cases}$
11	$y=\begin{cases}1-2x^2, если\ x\le 0\\1, если\ 0<x<0.5\\1+(x-0.5)^4, если\ x\ge 0.5\end{cases}$	12	$y=\begin{cases}0, если\ x\le 1\\1+\cos\pi x, если\ -1<x<0\\1.5, если\ x\ge 0\end{cases}$

5. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1) Прочитать задание и выписать все переменные, которые следует использовать при его решении.

2) Продумать порядок решения задачи и составить словесный алгоритм ее решения.

3) Составить список операторов, которые понадобятся при написании программы.

4) Составить блок-схему алгоритма решения задачи.

.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) Дайте определение алгоритма ?
2) Назовите свойства алгоритмов?
3) Каким образом можно описать алгоритм решения задачи ?
4) Как определяется разветвляющаяся структура алгоритма ?
5) Чем характеризуется полная и неполная альтернатива ?
6) В каких случаях используется конструкция выбора ?

7) Может ли разветвляющаяся структура иметь ветвь, направленную к началу программы ?

8) Если в алгоритме два условия (блока решения) стоят в одной ветви, где заканчивается первое и второе условия ?

9) Сколько условий можно записать в одном блоке решения ?

Лабораторная работа №4

АЛГОРИТМЫ ЦИКЛИЧЕСКОЙ СТРУКТУРЫ

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель работы: усвоение принципов построения алгоритмов циклической структуры.

Задачи работы: изучить основные положения ГОСТ 19.701-90 (ИСО 5807-85), связанные с изображением схем программ (графическим представлением алгоритма). Изучить построение циклических алгоритмов.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Приведены в методических указаниях по выполнению лабораторной работы №3.

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

1) Операционная система Windows XP;

2) Microsoft Калькулятор Плюс.

4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

В соответствии с номером варианта найти значение дискретной функции, заданной на нескольких интервалах. Разработать и описать алгоритм с циклической структурой.

Варианты заданий

n	задание	n	задание
1	$y=\begin{cases} x+0.6, & x\le -1 \\ x^3, & -1<x<1 \\ x-0.6, & x\ge 1 \end{cases}$	2	$y=\begin{cases} \sin^3(0.5+x), & x<0.5 \\ 1, & 0.5\le x\le 1 \\ x^2-1, & x>1 \end{cases}$
3	$y=\begin{cases} \sin^2(x+0.1), & x<0.2 \\ 1, & 0.2\le x\le 0.3 \\ \cos^2(x+0.1), & x>0.3 \end{cases}$	4	$y=\begin{cases} 4x^2+2x-8, & x\le 1 \\ 1, & -1<x\le 1 \\ x^3-x+10, & x>1 \end{cases}$
5	$y=\begin{cases} 1, & x<2 \\ 2-(x-2)^2, & 2\le x<3 \\ (x-4)^2, & x\ge 3 \end{cases}$	6	$y=\begin{cases} x+0.8, & x\le -1 \\ x^3, & -1<x\le 1 \\ 2\sqrt{x}-1, & x>1 \end{cases}$
7	$y=\begin{cases} \lvert x-1\rvert, & x<1 \\ x^2, & x\ge 1 \end{cases}$	8	$y=\begin{cases} e^{x-1}, & x\le 1 \\ \lg(x+2), & x>1 \end{cases}$
9	$y=\begin{cases} e^{x+1}, & x\le 2 \\ 5, & x>2 \end{cases}$	10	$y=\begin{cases} \sin^2 x, & x\le 0.5\pi \\ 1, & 0.5\pi<x<\pi \\ 0, & x\ge \pi \end{cases}$
11	$y=\begin{cases} \lvert x-1\rvert, & x<1 \\ x+e^{2x}, & x\ge 1 \end{cases}$	12	$y=\begin{cases} \sin 0.5(x+1), & x\ge 0 \\ 0, & x\le -2 \\ \sin \pi x, & x\ge 1 \end{cases}$

5. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

5) Прочитать задание и выписать все переменные, которые следует использовать при его решении.

6) Продумать порядок решения задачи и составить словесный алгоритм ее решения.

7) Составить список операторов, которые понадобятся при написании программы.

8) Составить блок-схему алгоритма решения задачи.

.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) Дайте определение алгоритма ?
2) Назовите свойства алгоритмов?
3) Каким образом можно описать алгоритм решения задачи ?
4) Как определяется циклическая структура алгоритма ?
5) Какие виды циклов вы знаете?
6) Может ли цикл быть бесконечным?

Лабораторная работа №5
ПРОГРАММЫ РАЗВЕТВЛЯЮЩЕЙСЯ СТРУКТУРЫ
1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель работы: усвоение принципов написания программ разветвляющейся структуры.

Задачи работы: составить алгоритм и написать программу разветвляющейся структуры.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Если в программе, в зависимости от некоторого условия, требуется выбрать тот или иной вариант действий, можно воспользоваться одним из условных операторов. В Turbo Pascal предусмотрены два условных оператора - `If` и `Case`.

1. Оператор IF.

Оператор `IF` имеет вид:

```
If  P  Then  s1   else   s2;
```

При выполнении этого оператора сначала вычисляется некоторое логическое выражение P (условие), в случае истинности которого выполняется оператор s1, а в случае ложности - оператор s2. Использованные здесь ключевые слова `If`, `Then` и `Else` имеют смысл; “если, то” и “иначе” соответственно.

Возможен сокращенный вариант оператора `IF`:

```
 If p Then s1;
```

Здесь, если условие P истинно, выполняется оператор s1. Если же условие P ложно, управление просто передается следующему (за оператором `If`) оператору в программе. Можно сказать, что первый вариант оператора `If` осуществляет выбор между двумя действиями, а второй - между действием и отсутствием действия.

В качестве условия может быть просто использовано логическое значение True или False, либо константа логического типа, имеющая одно из этих значений.

Сравнивать между собой числа и другие значения в Turbo Pascal можно с помощью шести операторов. Речь идет об операторах `>` (больше), `<` (меньше), `=` (равно), `<>` (не равно), `>=` (больше или равно) и `<=` (меньше или равно). Результат сравнения всегда представляет собой значение логического типа.

В условии также широко могут использоваться три логических оператора: `Not`, `And` и `Or`. Например, выражение

```
x1 and x2;
```

будет иметь значение `True`, если x1 и x2 принадлежат логическому типу и оба равны `True`.

Пример

```
If x>y   Then   z:=x/2 else  z:=y+2;
If   a and b>c  Then  d:=a+b  else  d:=c;
```

Во многих программах, в зависимости от некоторого условия, часто требуется выполнить не одно, а последовательность действий. Однако оператор `If` в Turbo Pascal обеспечивает выполнение единственного оператора, присутствующего после зарезервированного слова `Then` или `Else`. Выйти из положения можно, применив составные операторы. Оператор `IF` с составными операторами выглядит так:

```
If P Then Begin s1  s2; … sn End
Else Begin v1; v2;   … vn End;
```

Здесь s1-sn и v1-vn - некоторые операторы, образующие составные операторы.

Turbo Pascal допускает вложенность операторов IF.

2. Оператор CASE.

Рассмотренный выше оператор `If` позволяет осуществить выбор одной из двух альтернатив. В случае, если альтернатив больше, выйти из положения позволяет оператор `Case`.

Оператор `Case` имеет вид:

```
Case P of
        a: s1;
        b: s2;
        …
        n: sn;
  Else s(n+1)
End;
```

При выполнении этого оператора сначала вычисляется некоторое выражение P, называемое селектором выбора, а затем, в зависимости от полученного значения (если оно равно одной из констант a, b, ..., n, которые называются константами выбора), выполняется один из операторов: s1, s2,..., sn, - помеченный соответствующей константой. Причем каждый из этих операторов может быть составным. Операторы s1, s2, ... sn отделяются один от другого точками с запятой. Значение селектора выбора в операторе `Case` может повторяться, однако в этом случае будет выполнена только первая подходящая ветвь, а затем управление передается следующему (после `Case`) оператору в программе.

Если значение выражения P не совпадает ни с одной из констант выбора, выполняется оператор s(n+1), содержащийся после ключевого слова `Else`, причем ветвь `Else` в операторе `Case` необязательна. Использованные здесь зарезервированные слова `Case`, `Of`, `Else` и `End` имеют смысл: «вариант», «из», «иначе» и «конец» соответственно.

Выражение, играющее роль селектора выбора, должно принадлежать порядковому типу данных (т.е. типу, имеющему конечное число значений). К порядковым относятся, например, типы данных Integer, Boolean и Char. Однако тип Real порядковым не является.

Кроме одиночных констант, в вариантах оператора `Case` могут использоваться диапазоны значений и списки (представленные через запятую):

```
Case x of
   1..5:  s1
   2,  3,  8:s2
   4,  6,  9..13:s3
Else
s4
End;
```

Пример

В программе, которая выводит название дня недели, соответствующее его номеру, выбор происходит следующим образом:

```
Case x of
1: Write ('понедельник');
2: Write ('вторник');
3: Write ('среда');
4: Write ('четверг');
5: Write ('пятница');
```

```
6: Write ('суббота');
7: Write ('воскресенье');
End;
```

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

1) Операционная система Windows XP;
2) Microsoft Калькулятор Плюс;
3) Turbo Pascal 7.0 или ABC Pascal.

4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Заданием на лабораторную работу является написание программы, реализующей алгоритм решения поставленной задач на языке Turbo Pascal.

Варианты заданий для расчета

№ варианта	Задание
1	Написать программу, которая вычисляет частное от деления двух чисел. Программа должна проверять правильность введенных пользователем данных и, если они неверные (делитель равен нулю), выдавать сообщение об ошибке.
2	Написать программу вычисления площади кольца. Программа должна проверять правильность исходных данных. Радиус отверстия не может быть больше радиуса кольца.
3	Написать программу решения квадратного уравнения. Программа должна проверять правильность исходных данных и в случае, когда коэффициент при второй степени неизвестного равен нулю, выводить соответствующее сообщение.
4	Написать программу вычисления стоимости покупки с учетом скидки. Скидка в 10% предоставляется, если сумма покупки больше 1000 руб.
5	Написать программу вычисления стоимости покупки с учетом скидки. Скидка в 3% предоставляется в том случае, если сумма покупки больше 500 руб., в 5% — если сумма больше 1000 руб.
6	Написать программу, которая сравнивает два числа, введенных с клавиатуры. Программа должна указать, какое число больше, или, если числа равны, вывести соответствующее сообщение.
7	Написать программу, которая.выводит пример на умножение двух однозначных чисел, запрашивает ответ пользователя, проверяет его и выводит сообщение "Правильно!" или "Вы ошиблись" и правильный результат.
8	Написать программу, которая проверяет, является ли четным введенное пользователем целое число и выводит результата на экран.
9	Написать программу, которая вычисляет оптимальный вес пользователя, сравнивает его с реальным и выдает рекомендацию о необходимости поправиться или похудеть. Оптимальный вес вычисляется по формуле: рост (в сантиметрах)—100.

10	Написать программу, которая запрашивает у пользователя номер дня недели и выводит одно из сообщений: "Рабочий день", "Суббота" или "Воскресенье".
11	Написать программу, которая после введенного с клавиатуры числа (в диапазоне от 1 до 999), обозначающего денежную единицу, дописывает слово "рубль" в правильной форме. Например, 12 рублей, 21 рубль
12	Написать программу проверки знания даты начала второй мировой войны. В случае неверного ответа пользователя программа должна выводить правильный ответ. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

5. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1) Прочитать задание и выписать все переменные, которые следует использовать при его решении.
2) Продумать порядок решения задачи и составить алгоритм ее решения.
3) Составить список операторов, которые понадобятся при написании программы.
4) Описать в разделе описаний все используемые переменные и их типы.
5) Составить программу решения задачи.
6) Проверить правильность программы путем введения данных, результат решения для которых известен.
7) Если необходимо, отладить программу.
8) Оформить ввод и вывод данных удобочитаемым образом.
9) Сохранить результат, как «lr05.pas»

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) Какие варианты выбора вы знаете?
2) Как с помощью оператора If можно организовать выбор из 3-х и более альтернатив?
3) Что такое логическая переменная?
4) Расскажите о составных условных операторах.
5) Расскажите о логических операторах.

Лабораторная работа №6
ПРОГРАММЫ ЦИКЛИЧЕСКОЙ СТРУКТУРЫ

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель работы: усвоение принципов написания программ циклической структуры.

Задачи работы: составить алгоритм и написать программу циклической структуры.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Для многократного повторения одних и тех же действий в Turbo Pascal предусмотрены три оператора цикла. Если число повторений цикла (или итераций) заранее неизвестно, однако известно условие завершения цикла, в таких случаях применяются операторы `Repeat` и `While`. Если же число повторений известно, то применяется оператор `For`.

1. Оператор While.

Оператор цикла `While`, известный как оператор цикла с предусловием, имеет вид:

```
While P do s;
```

При выполнении этого оператора сначала вычисляется некоторое логическое выражение P (условие), принадлежащее типу Boolean, в случае истинности которого выполняется оператор s (являющийся, как правило, составным оператором). После этого вычисление условия, его проверка и выполнение оператора s повторяются до тех пор, пока выражение P не становится равным False. Затем управление передается следующему (после `While`) оператору в программе. Использованные здесь ключевые слова `While` и `Do` имеют смысл «пока» и «выполнять» соответственно.

Пример

Предположим, даны числа X и Y (X>1). Требуется получить все члены бесконечной последовательности X, X^2, X^3, ... которые меньше Y.

```
Program sequence;
   var x,y,z: real;
Begin
Read(x,y);
z:=x;
While z<y do
  Begin
  Writeln(z);
  z:=z*x;
  End;
End.
```

2. Оператор Repeat.

Оператор цикла `Repeat`, известный как оператор цикла с постусловием, имеет вид:

```
Repeat s until P;
```

При выполнении этого оператора сначала выполняется тело цикла (s), затем вычисляется некоторое логическое выражение P (условие), принадлежащее типу Boolean, в случае ложности которого вновь выполняется тело цикла. Затем выполнение тела цикла, вычисление условия P и его проверка повторяются до тех пор, пока выражение P не становится равным True. После этого управление передается следующему (за `Repeat`) оператору в программе. Использованные здесь зарезервированные слова `Repeat` и `Until` имеют смысл повторять и пока не соответственно.

Оператор цикла Repeat отличается от оператора While, во-первых, тем, что здесь условие проверяется после выполнения тела цикла. Иными словами, гарантируется хотя бы однократное его выполнение. Во-вторых, оператор Repeat выполняется до тех пор, пока условие равно False, и управление передается следующему (за Repeat) оператору, когда условие становится равным True (для оператора While имеет место обратная зависимость).

Пример

Приведем фрагменты программы, вычисляющей степень числа 3 в диапазоне между 1 и 300.

```
a:=1;
Repeat
Writeln(a);
a:=a*3
Until  a>=300
End;
```

Также следует обратить внимание, что тело цикла Repeat не требуется заключать в операторные скобки Begin..End. Если в операторе While после ключевого слова Do выполняется единственный оператор (и если требуется циклически выполнять несколько действий, приходится несколько операторов объединять в составной оператор), то в операторе Repeat между ключевыми словами Repeat и Until можно ввести любое количество операторов, без необходимости заключать их в операторные скобки.

Наконец, в операторе Repeat после последнего оператора в теле цикла нет точки с запятой.

3. Оператор For.

Оператор цикла For, известный как оператор цикла с параметром, имеет вид:

```
For i:=a to b do s;
```

При выполнении этого оператора сначала вычисляется некоторое начальное значение a, которое присваивается переменной i, называемой параметром цикла. Затем вычисляется конечное значение b и проверяется, имеет ли место равенство i=b. Если равенства нет, выполняется оператор s, который может быть составным, и переменная увеличивается на единицу. После этого проверка (не равен ли параметр конечному значению), выполнение оператора s и увеличение переменной на единицу выполняются циклически до тех пор, пока не наступает равенство i=b. Параметр цикла i, а также начальное и конечное значения (a и a) могут принадлежать любому порядковому типу (например, Integer или Char). (Но при этом все они должны быть одного типа.) Если начальное значение превышает или равно конечному значению с самого начала, оператор s не выполнится ни разу.

Использованные здесь зарезервированные слова For, To и Do имеют смысл: «от», «до» и «выполнять» соответственно.

Оператор цикла For имеет такие особенности:

1) В теле цикла запрещается явно изменять значение параметра цикла (с помощью оператора присваивания, например);

2) По завершении работы оператора цикла FOR значение параметра (I) считается неопределенным.

Возможна и другая форма оператора цикла с параметром:

```
For i:=a downto b do s;
```

Здесь, чтобы выполнялся оператор s, начальное значение a должно превышать конечное значение b. Кроме того, в этом случае параметр i с каждым циклом уменьшается на единицу, пока не становится равным конечному значению B.

Оператор цикла с параметром следует использовать тогда, когда заранее точно известно, сколько раз должно быть выполнено тело цикла.

Пример

```
For i:=l to 5 do x:=sqr(x);
For i:=z downto a do Write (i);
```

4. Вложенные циклы.

В программах на Turbo Pascal возможно использование вложенных циклов. Это подразумевает, что существует внешний цикл и один или несколько внутренних циклов. Каждое повторение внешнего цикла означает завершение всех внутренних циклов; при этом всем выражениям, которые управляют внутренними циклами, вновь присваиваются начальные значения.

Пример

```
For i:=1 to 10 do
    For k:=1 to 5 do
      A[i,j]:=0; {обнуление матрицы}
```

5. Оператор перехода и пустой оператор.

Оператор перехода имеет вид:

```
Goto P;
```

где P - метка, которой помечен некоторый иной оператор в программе. Использованное здесь зарезервированное слово `Goto` имеет смысл перейти. В данном случае речь идет о переходе на метку, указанную после оператора `Goto` (и которой помечен иной оператор в программе).

После оператора `Goto` должна быть указана единственная метка и такой же меткой обязательно должен быть помечен один (и только один) из операторов в программе. Однако один и тот же оператор может помечаться несколькими метками. Иными словами, каждый оператор перехода передает управление в одну (и только в одну) точку программы, однако возможна передача управления из разных точек программы в одну точку.

Все метки, используемые в программе, должны быть объявлены. Раздел описания меток находится между заголовком и телом программы.

Пример

Текст программы, в которой использован оператор перехода:

```
Program jump;
label 1;
var n:integer;
Begin
Read (n);
If n>1000 Then Goto 1
else n:=n+100;
write(n);
1: end.
```

Злоупотреблять использованием операторов `Goto` не рекомендуется, поскольку в результате получаются малопонятные программы, тем более, что без оператора `Goto`, как правило, можно обойтись другими средствами Turbo Pascal.

Если метка используется внутри процедуры или функции, она "видна" только в пределах этой процедуры или функции и должна быть описана в ней. Иными словами, передача управления снаружи внутрь процедуры или функции с помощью оператора перехода невозможна.

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

1) Операционная система Windows XP;
2) Microsoft Калькулятор Плюс;
3) Turbo Pascal 7.0 или ABC Pascal.

4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Заданием на лабораторную работу является написание программы, реализующей алгоритм решения поставленной задач на языке Turbo Pascal.

Варианты заданий для расчета

№ варианта	Задание
1	Написать программу, которая выводит таблицу квадратов первых десяти целых положительных чисел.
2	Написать программу, которая выводит таблицу квадратов первых пяти целых положительных нечетных чисел.
3	Написать программу, которая вычисляет сумму первых n целых положительных целых чисел. Количество суммируемых чисел должно вводиться во время работы программы.
4	Написать программу, которая вычисляет сумму первых n целых положительных четных чисел. Количество суммируемых чисел должно вводиться во время работы программы.
5	Написать программу, которая определяет максимальное число из введенной с клавиатуры последовательности положительных чисел (длина последовательности не ограничена).
6	Написать программу, которая вычисляет сумму первых n членов ряда 1, 3, 5, 7, ... Количество суммируемых членов ряда задается во время работы программы.
7	Написать программу, которая выводит таблицу степеней двойки (от нулевой до десятой).
8	Написать программу, которая вычисляет факториал введенного с клавиатуры числа. (Факториалом числа n называется произведение целых чисел от 1 до n. Например, факториал 1 равен 1, 8 — 40320).
9	Написать программу, которая вводит с клавиатуры 5 дробных чисел и вычисляет их среднее арифметическое.

10	Написать программу, которая вычисляет среднее арифметическое вводимой с клавиатуры последовательности дробных чисел. Количество чисел должно задаваться во время работы программы.
11	Написать программу, которая вводит с клавиатуры последовательность из пяти дробных чисел и после ввода каждого числа выводит среднее арифметическое полученной части последовательности.
12	Написать программу, которая "задумывает" число в диапазоне от 1 до 10 и предлагает пользователю угадать число за 5 попыток.

5. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1) Прочитать задание и выписать все переменные, которые следует использовать при его решении.
2) Продумать порядок решения задачи и составить алгоритм ее решения.
3) Составить список операторов, которые понадобятся при написании программы.
4) Описать в разделе описаний все используемые переменные и их типы.
5) Составить программу решения задачи.
6) Проверить правильность программы путем введения данных, результат решения для которых известен.
7) Если необходимо, отладить программу.
8) Оформить ввод и вывод данных удобочитаемым образом.
9) Сохранить результат, как «lr06.pas»

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) Расскажите о счетном операторе цикла.
2) Расскажите о циклах с предусловием.
3) Расскажите о циклах с постусловием.
4) Как выбрать, какой оператор цикла использовать в каждом конкретном случае?
5) Расскажите об операторах перехода.

Лабораторная работа №7
МАССИВЫ

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель работы: усвоение технологии работы с массивами.

Задачи работы: построить схему алгоритма обработки двумерного массива с использованием вложенных циклов и последующая разработка программы.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Массивы.

Одним из структурированных типов данных является регулярный тип или массив. Массив представляет собой совокупность связанных данных, состоящую из фиксированного числа элементов одного типа, который называется базовым. Для определения массива достаточно указать его базовый тип, а также число элементов в массиве и метод их нумерации. Возможен доступ к отдельным элементам массива – для этого достаточно указать имя массива и номер (индекс) нужного элемента. Описание регулярного типа имеет следующий вид:

```
Type  s=Array [i] Of a;
```

Здесь Array и Of – зарезервированные слова, имеющие смысл: «массив» и «из»; s – имя регулярного типа; i – тип индексов (указывается в квадратных скобках); a – тип элементов массива (базовый тип). Элементы массива могут представлять собой значения любого типа. Для индексирования массива в Turbo Pascal используется тип данных. Это может быть любой из целочисленных типов (за исключением Longint), и Boolean, а также перечислимый и интервальный типы (за исключением интервальных типов на основе Longint). Вещественные типы для индексирования массивов не годятся.

Также, массив можно описать непосредственно в разделе описания переменных.

Пример

```
var
  a1:Array [byte] of  boolean;
  a2:Array [char] of boolean;
  a3:Array [Red,Yellow,Green] of char;
  a4:Array [1..100] of integer;
```

Обращения к отдельным элементам массивов A1 и A2 выглядят так:

```
a1 [90]:=false;
a2 [z]:=true;
```

Следует понимать, что индексы массива могут представляться не только в виде явно заданных значений, но и переменных, и выражений.

Разумеется, массивы можно объявлять не только в качестве анонимных типов, но и как типы с собственным именем – в разделе описания типов:

```
Type
   LogScale=Array [byte] of boolean;
var
   a1:LogScale;
```

Массив – это последовательность однотипных элементов, так же, как и строка, однако если символы строки представляют собой только значения типа Char, то элементы массива могут принадлежать различным типам – как простым, так и структурированным.

Обращения к отдельным элементам строк и массивов выглядят одинаково – для этого достаточно указать имя строки или массива и индекс. Однако имеются и отличия: если текущей длиной строки можно манипулировать, то для массивов ничего подобного не предусмотрено. Длина строки также ограничена 255 символами, в то время как для массивов такого ограничения нет. Кроме того, можно ввести или вывести значение всей строковой переменной с помощью единственного оператора (например, `Read(X)` или `Write(X)`, где X – значение типа STRING), а для массивов это невозможно.

В целом же вместо массива символьных значений часто можно использовать строку, и наоборот. Какому типу данных следует отдать предпочтение в том или ином случае, зависит от особенностей применения.

2. Многомерные массивы.

В Turbo Pascal элементы массива могут представлять собой значение не только простых, но и структурированных типов. Допускается, например, существование массивов, элементами которых являются также массивы. Описание подобного массива выглядит так:

```
a1=array[1..5] of array [1..4] of integer;
```

Этот массив можно представить в качестве двумерной матрицы с 20 элементами (4 на 5). Возможна сокращенная запись приведенного выше описания массива:

```
a1=array [1..5, 1..4] of integer;
```

Эти описания эквивалентны. Обращение к одному из элементов данного массива выглядит так:

```
a1[3][4]:=13;
```

или

```
a1[3,4]:=13;
```

Возможны не только двумерные, но и многомерные массивы, причем число измерений не ограничивается. Однако при этом сохраняется ограничение в 64К (или 65520 байт) на размер переменных регулярного типа. Это связано с максимальным объемом памяти, выделяемой для данной переменной.

3. Применимые к массивам операции.

Поскольку элемент массива трактуется как переменная, он может фигурировать в выражениях. Набор операций над элементами массива соответствует операциям, допустимым для базового типа.

Таблица 1.

Примеры действий над элементами массива

Пример действия	Комментарий
`Write(6,a1[5])`	На экране отображается число 6, а затем значение 5-го элемента массива A1.
`a4[5]:=a4[3]+a4[2]`	Значения 3 и 2-го элементов массива A4 складываются, и полученная сумма в качестве значения присваивается элементу 5 того же массива.
`abc:=abc+a4[88]`	Значение 88-го элемента массива A4 суммируется со значением переменной abc, и полученная сумма присваивается этой же переменной.
`abc:=a4[55]+a4[56]`	Значения 55 и 56-го элементов массива A4 складываются, и полученная сумма присваивается переменной abc.
`a4[33]:=a4[33]+20`	Значение 33-го элемента массива A4 увеличивается на 20.

Что касается набора операций над массивами в целом, то скопировать все элементы из одного массива в другой можно единственным оператором присваивания. Например, если X и Y –массивы, принадлежащие одному типу, то правомерен оператор

```
X := Y;
```

Этот оператор копирует значения всех элементов массива X в массив Y. Однако нельзя использовать с массивами операции сравнения. Тем не менее, если такая необходимость существует, можно организовать поэлементное сравнение массивов.

Также абсолютно неприменимы к массивам арифметические и логические операции.

Кроме того, к массивам (в отличие от строк) нельзя применять стандартные процедуры Read и Write. Однако можно организовать считывание или вывод на экран каждого элемента массива в отдельности.

Массивы относятся к структурированным типам данных в Turbo Pascal. Массив состоит из фиксированного числа элементов (компонент) одного типа и характеризуется общим именем. Доступ к отдельным элементам массива осуществляется с помощью общего имени и порядкового номера (индекса или адреса) необходимого элемента.

Имя массива - это любое допустимое в Turbo Pascal имя, отличное от служебных слов, имен функций и процедур. Массив может быть описан в подразделе Var или в подразделах Var и Type, одновременно.

Массив с одним индексом называют **одномерным**, с двумя - **двумерным**, с тремя - **трехмерным** и т.д. Число индексов у массива в Turbo Pascal не ограничивается, но необходимо помнить, что размер массива не должен превышать 64 Кбайт.

Любой двумерный массив представляет собой матрицу: первому индексу можно поставить в соответствие строки, а второму - столбцы матрицы. Кроме того, двумерный массив можно интерпретировать как одномерный, элементами которого является другой одномерный массив.

Описание такого массива имеет вид:

```
Type  tstr=array[1..25] of real;
Var  masssiv:array[1..10] of tstr;
```

Это описание равносильно описанию в примере, приведенному выше для массивов с именами Matr1 и Matr2.

Оперативная память под элементы массива выделяется на этапе трансляции. В памяти компьютера элементы массива следуют друг за другом.

Если массив двумерный, то память под него выделяется так, что быстрее меняется самый правый индекс. В качестве примера рассмотрим порядок выделения оперативной памяти под массив, описанный следующим образом: Var M:array[1..2,1..4] of byte;

Например:

Этот массив будет располагаться в памяти в следующем порядке:

```
M[1,1]; M[1,2]; M[1,3]; M[1,4]; M[2,1]; M[2,2]; M[2,3]; M[2,4].
```

В Turbo Pascal можно одним оператором присваивания передать все элементы одного массива другому массиву того же типа.

Например:

```
Var   m1,m2:array[1..10] of word;
. . .
Begin
. . .
m1:=m2;  { перезапись из одного массива в другой}
. . .
End.
```

Для **сравнения** содержимого двух массивов необходимо использовать оператор цикла с параметром и указываться индексы.

У т**ипизированных констант–массивов** в качестве начального значения используется список констант, отделенных друг от друга запятыми. Список заключается в круглые скобки.

Например:

1)

```
Const Mas:array[1..10] of byte=(1,1,1,1,1,1,1,1,1,1);
```

Заполнение массива из 10 целых чисел значением, равным единице.

2)

```
Const        massim:  array[0..5]  of    char  =  ('a','b','c',
'd','e','f');
```

Заполнение массива из 6 элементов символами - буквами латинского алфавита.

3)

```
Const        Matr:    array[1..5,1..2]of    byte    =    ((0,0),
(0,0),(0,0),(0,0),(0,0));
```

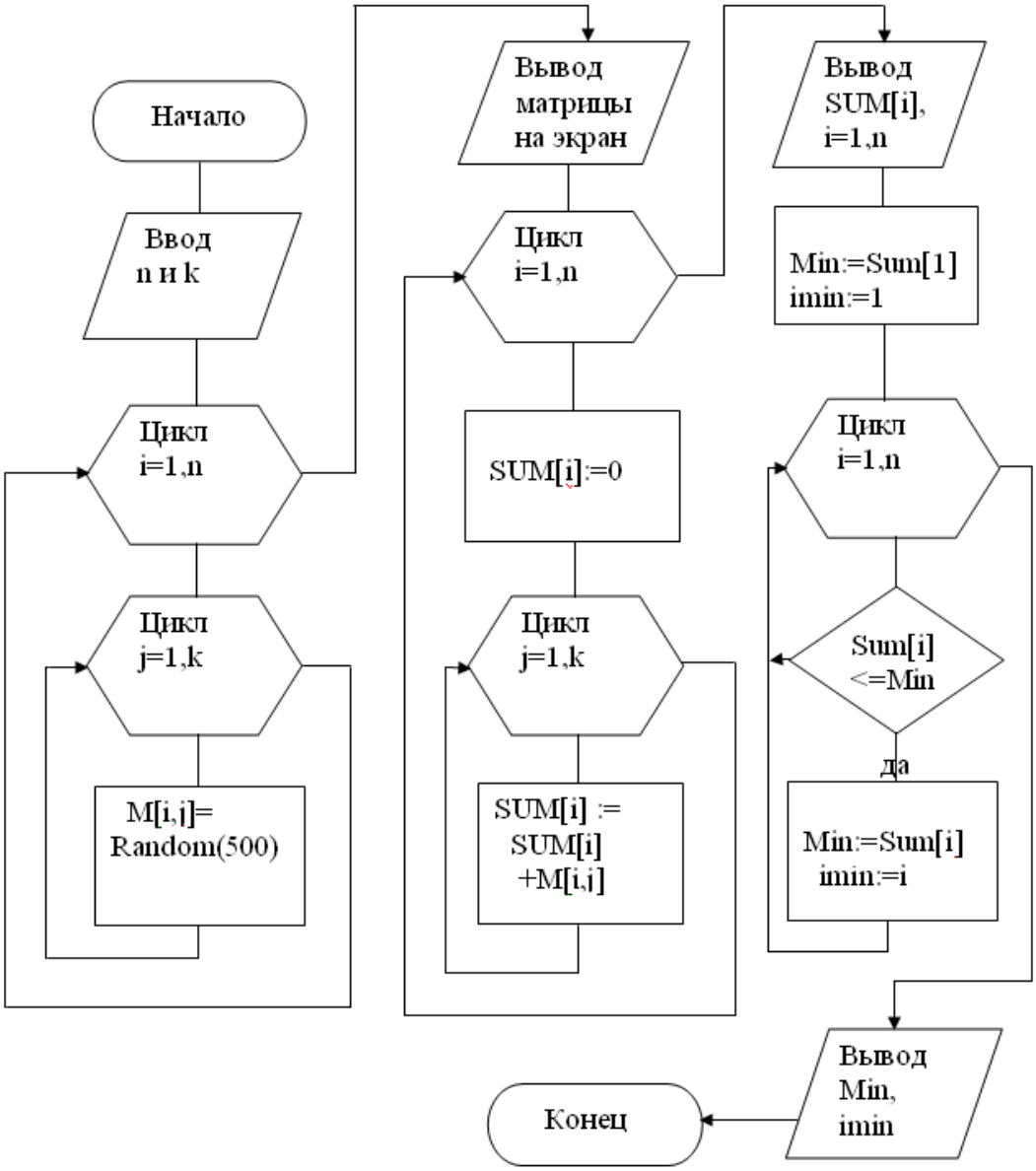
Обнуление матрицы из 10 целых чисел.

Замечание: количество переменных в списке констант должно строго соответствовать объявленной длине массива по каждому индексу!

3. Типовой пример

Составить схему алгоритма и программу определения сумм элементов в каждой строке матрицы $M=\{m_{ij}\}$, $i=1..n$, $j=1..k$, где n - число строк, k - число столбцов матрицы, m_{ij} - целые числа из диапазона: от 0 до 500. Записать полученные значения сумм в одномерный массив «sum», а затем вывести их на экран дисплея. Числа в массив M занести с помощью функции `Random`. Определить и вывести на экран номер строки с минимальным значением суммы.

Схема алгоритма



Текст программы

```
Uses crt;
{ Описание данных}
Var M:array[1..50,1..100]of integer;
    Sum:array[1..50] of longint;
    n,k,i,j,imin:integer;
    Min:longint;
BEGIN
clrscr;
{ Ввод данных}
writeln(' Введите число строк и столбцов');
readln(n,k);
Randomize;  { Стандартная процедура см. теорию}
Заполнение матрицы случайными числами}
  for i:=1 to n do
  for j:=1 to k do
       M[i,j]:=Random(500);
writeln('   Элементы заполненной матрицы');
```

```
  for i:=1 to n do
       begin
       for j:=1 to k do
       write(M[i,j]:4);
       writeln;
       end;
writeln('  Сумма элементов в каждой строке');
write('  Номера строк :     ');
  for i:=1 to n do
       write(i,'    ');writeln;
write('  Сумма в строке :  ');
  for i:=1 to n do
        begin
        Sum[i]:=0;
           for j:=1 to k do
               Sum[i]:=Sum[i]+M[i,j];
        write(Sum[i], ' ');
        end;
writeln;
{  Поиск минимального значения}
Min:=Sum[1]; imin:=i;
   for i:=1 to n do
       if   Sum[i] <= Min then
            begin
            Min:=Sum[i]; imin:=i;
            end;
writeln('Минимальная сумма =',Min,'в строке',imin);
readln; END.
```

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

1) Операционная система Windows XP;
2) Microsoft Калькулятор Плюс;
3) Turbo Pascal 7.0 или ABC Pascal.

4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Разработать схему алгоритма и программу решения задачи согласно варианту задания.

1. Составить схему алгоритма и программу согласно варианту задания. В прямоугольной матрице размером $T * M$, имеющей имя MATR содержатся целые числа. T - число строк, M - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму элементов в каждой строке. Определить строку с максимальным значением этой суммы и вывести ее номер на экран.

2. В прямоугольной матрице размером $M * T$, имеющей имя MAS содержатся целые числа. M - число строк, T - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму элементов в каждой строке. Определить строку с минимальным значением этой суммы и вывести ее номер на экран.

3. В прямоугольной матрице размером $K * M$, имеющей имя MATR содержатся целые числа. K - число строк, M - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму положительных элементов в каждой строке. Определить строку с максимальным значением этой суммы и вывести ее номер на экран.

4. В прямоугольной матрице размером $M * K$, имеющей имя MAT содержаться целые числа. M - число строк, K - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму отрицательных элементов в каждой строке. Определить строку с максимальным значением этой суммы и вывести ее номер на экран.

5. В прямоугольной матрице размером $L * M$, имеющей имя MATR содержаться целые числа. L - число строк, M - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму положительных элементов в каждой строке. Определить строку с минимальным значением этой суммы и вывести ее номер на экран.

6. В прямоугольной матрице размером $M * K$, имеющей имя MAT содержаться целые числа. M - число строк, K - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму отрицательных элементов в каждой строке. Определить строку с минимальным значением этой суммы и вывести ее номер на экран.

7. В прямоугольной матрице размером $M * K$, имеющей имя MA содержаться целые числа. M - число строк, K - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму элементов в каждом столбце. Определить столбец с максимальным значением этой суммы и вывести его номер на экран.

8. В прямоугольной матрице размером $M * K$, имеющей имя M содержаться целые числа. M - число строк, K - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму элементов в каждом столбце. Определить столбец с минимальным значением этой суммы и вывести его номер на экран.

9. В прямоугольной матрице размером $M * K$, , имеющей имя MM содержаться целые числа. M - число строк, K - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму положительных элементов в каждом столбце. Определить столбец с максимальным значением этой суммы и вывести его номер на экран.

10. В прямоугольной матрице размером $M * K$, , имеющей имя MAM содержаться целые числа. M - число строк, K - число столбцов. Ввести элементы матрицы с

клавиатуры. Определить и вывести на экран сумму отрицательных элементов в каждом столбце. Определить столбец с минимальным значением этой суммы и вывести его номер на экран.

11. В квадратной матрице размером $K * K$, имеющей имя МА содержаться целые числа. K - число строк и столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму положительных элементов в каждом столбце. Определить столбец с минимальным значением этой суммы и вывести его номер на экран.

12. В квадратной матрице размером $T * T$, имеющей имя МКА, содержаться целые числа. T - число строк и столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму отрицательных элементов в каждом столбце. Определить столбец с максимальным по модулю значением этой суммы и вывести его номер на экран.

5. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1) Прочитать задание и выписать все переменные, которые следует использовать при его решении.
2) Продумать порядок решения задачи и составить алгоритм ее решения.
3) Составить список операторов, которые понадобятся при написании программы.
4) Описать в разделе описаний все используемые переменные и их типы.
5) Составить программу решения задачи.
6) Проверить правильность программы путем введения данных, результат решения для которых известен.
7) Если необходимо, отладить программу.
8) Оформить ввод и вывод данных удобочитаемым образом.
9) Сохранить результат, как «lr7.pas»

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) В чем отличие одномерных и многомерных массивов?
2) Что такое базовый тип массива?
3) Расскажите о создании пользовательского типа массива?
4) Какие операции применимы к массивам в целом?
5) В каких случаях необходима поэлементная обработка массивов?

Лабораторная работа №8
ВНЕШНИЕ ФАЙЛЫ

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель работы: усвоение принципов использования внешних файлов.

Задачи работы: освоить использование внешних файлов для ввода и вывода данных.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Концепция файла.

Файловая переменная связывается с именем файла в результате обращения к стандартной процедуре `Assign`:

```
Assign (файловая переменная, имя файла);
```

Здесь файловая переменная - идентификатор, объявленный в программе как переменная файлового типа; имя файла - текстовое выражение, содержащее имя файла.

Имя файла - это любое выражение строкового типа, которое строится по правилам определения имен в MS-DOS (операционной системе ПК). Имя содержит до восьми разрешенных символов. За именем может следовать расширение - последовательность до трех разрешенных символов; расширение, если оно есть, отделяется от имени точкой. Перед именем может указываться так называемый путь к файлу: имя диска и/или имя текущего каталога и имена каталогов вышестоящих уровней. Весь путь к файлу отделяется от имени файла обратной косой чертой. Максимальная длина имени вместе с путем - 79 символов.

Пример

```
var  finp: text;
     fout: file of String;
const name = 'c:\dir\subdir\out.txt';
Begin
     Assign(finp,'123.dat') ;
     Assign(fout,name);
End.
```

Инициировать файл означает указать для этого файла направление передачи данных. В Турбо Паскале можно открыть файл для чтения, для записи информации, а также для чтения и записи одновременно.

Для чтения файл инициируется с помощью стандартной процедуры `Reset`:

```
Reset (файловая переменная);
```

Здесь файловая переменная, связанная ранее процедурой `Assign` с уже существующим файлом или логическим устройством-приемником информации. При выполнении этой процедуры дисковый файл или логическое устройство подготавливается к чтению информации. В результате специальная переменная-указатель, связанная с этим файлом, будет указывать на начало файла, т.е. на компонент с порядковым номером 0.

В Турбо Паскале разрешается обращаться к типизированным файлам, открытым процедурой `Reset` (т.е. для чтения информации), с помощью процедуры `Write` (т.е. для записи информации). Такая возможность позволяет легко обновлять ранее созданные типизированные файлы и при необходимости расширять их. Для текстовых файлов, открытых процедурой `Reset`, нельзя использовать процедуру `Write` или `Writeln`.

```
Rewrite (файловая переменная);
```

Инициирует запись информации в файл или в логическое устройство, связанное ранее с файловой переменной. Процедурой Rewrite нельзя инициировать запись информации в ранее существовавший дисковый файл: при выполнении этой процедуры старый файл уничтожается и никаких сообщений об этом в программу не передается. Новый файл подготавливается к приему информации и его указатель принимает значение 0.

Стандартная процедура:

```
Append (файловая переменная);
```

Инициирует запись в ранее существовавший текстовый файл для его расширения, при этом указатель файла устанавливается в его конец. Процедура Append применима только к текстовым файлам, т.е. их файловая переменная должна иметь тип Text (см. выше). Процедурой Append нельзя инициировать запись в типизированный или нетипизированный файл. Если текстовый файл ранее уже был открыт с помощью Reset или Rewrite, использование процедуры Append приведет к закрытию этого файла и открытию его вновь, но уже для добавления записей.

2. Общие файловые процедуры.

Ниже описываются процедуры и функции, которые можно использовать с файлами любого вида. Специфика работы с типизированными, текстовыми и нетипизированными файлами рассматривается в следующих разделах.

```
Close (файловая переменная);
```

Закрывает файл, однако связь файловой переменной с именем файла, установленная ранее процедурой Assign, сохраняется.

При создании нового или расширении старого файла процедура обеспечивает сохранение в файле всех новых записей и регистрацию файла в каталоге. Функции процедуры Close выполняются автоматически по отношению ко всем открытым файлам при нормальном завершении программы. Поскольку связь файла с файловой переменной сохраняется, файл можно повторно открыть без дополнительного использования процедуры Assign.

```
Rename (файловая переменная, новое имя);
```

Переименовывает файл. Здесь новое имя - строковое выражение, содержащее новое имя файла. Перед выполнением процедуры необходимо закрыть файл, если он ранее был открыт процедурами Reset, Rewrite или Append.

```
Erase (файловая переменная);
```

Уничтожает-файл. Перед выполнением процедуры необходимо закрыть файл, если он ранее был открыт процедурами Reset, Rewrite или Append.

```
 Flush(файловая переменная);
```

Очищает внутренний буфер файла и, таким образом, гарантирует сохранность всех последних изменений файла на диске. Формат обращения:

Любое обращение к файлу в Турбо Паскале осуществляется через некоторый буфер, что необходимо для согласования внутреннего представления файлового компонента (записи) с принятым в ДОС форматом хранения данных на диске. В ходе выполнения процедуры Flush все новые записи будут действительно записаны на диск. Процедура игнорируется, если файл был инициирован для чтения процедурой Reset.

```
EOF (файловая переменная) : Boolean;
```

Логическая функция, тестирующая конец файла. Возвращает True, если файловый указатель стоит в конце файла. При записи это означает, что очередной компонент будет добавлен в конец файла, при чтении - что файл исчерпан.

```
ChDir (путь);
```

Изменение текущего каталога. Здесь путь - строковое выражение, содержащее путь к устанавливаемому по умолчанию каталогу.

```
GetDir (устройство, каталог);
```

Позволяет определить имя текущего каталога (каталога по умолчанию). Здесь устройство - выражение типа Word , содержащее номер устройства: 0 - устройство по умолчанию, 1 - диск A, 2 - диск B и т.д. Каталог - переменная типа String, в которой возвращается путь к текущему каталогу на указанном диске.

```
MkDir (каталог);
```

Создает новый каталог на указанном диске. Здесь каталог - выражение типа String, задающее путь к каталогу. Последним именем в пути, т.е. именем вновь создаваемого каталога не может быть имя уже существующего каталога.^

```
RmDir (каталог);
```

Удаляет каталог. Удаляемый каталог должен быть пустым, т.е. не содержать файлов или имен каталогов нижнего уровня.

Ряд полезных файловых процедур и функций становится доступным при использовании библиотечного модуля `DOS.TPU`, входящего в стандартную библиотеку `TURBO.TPL` . Эти процедуры и функции указаны ниже. Доступ к ним возможен только после объявления `Uses Dos` в начале программы.

```
FindFirst (маска, атрибуты, имя);
```

Возвращает атрибуты первого из файлов, зарегистрированных в указанном каталоге. Здесь маска - строковое выражение, содержащее маску файла; атрибуты - выражение типа Byte, содержащее уточнение к маске (атрибуты); имя - переменная типа SearchRec, в которой будет возвращено имя файла.

При формировании маски файла используются следующие символы-заменители ДОС:

* означает, что на месте этого символа может стоять сколько угодно (в том числе ноль) разрешенных символов имени или расширения файла;

? означает, что на месте этого символа может стоять один из разрешенных символов.

3. Текстовые файлы.

Текстовые файлы связываются с файловыми переменными, принадлежащими стандартному типу Text. Текстовые файлы предназначены для хранения текстовой информации. Именно в такого типа файлах хранятся, например, исходные тексты программ. Компоненты (записи) текстового файла могут иметь переменную длину, что существенно влияет на характер работы с ними.

Для доступа к записям применяются процедуры `Read, Readln, Write, Writeln`. Они отличаются возможностью обращения к ним с переменным числом фактических параметров, в качестве которых могут использоваться символы, строки и числа. Первым параметром в любой из перечисленных процедур может стоять файловая переменная. В этом случае осуществляется обращение к дисковому файлу или логическому устройству, связанному с переменной процедурой `Assign`.

```
Read (файловая переменная,список ввода);
```

Обеспечивает ввод символов, строк и чисел. Здесь список ввода - последовательность из одной или более переменных типа Char, String, а также любого целого или вещественного типа.

Процедура Read прекрасно приспособлена к вводу чисел. При обращении к ней за вводом очередного целого или вещественного числа процедура «перескакивает» маркеры конца строк, т.е. фактически весь файл рассматривается ею как одна длинная строка, содержащая текстовое представление чисел. В сочетании с проверкой конца файла функцией EOF процедура Read позволяет организовать простой ввод массивов данных.

Пример

```
const
     N = 1000; {Максимальная длина ввода}
var
     f : text;
     m : array [1..N] of real;
     i : Integer;
Begin
Assign(f, 'prog.dat') ;
Reset(f); i := 1;
While not EOF(f) and (i <= N) do
Begin
Read(f ,m[i]);
Inc(i);
End;
Close(f);
     .......
End.
```

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

1) Операционная система Windows XP;
2) Microsoft Калькулятор Плюс;
3) Turbo Pascal 7.0 или ABC Pascal.

4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Заданием на лабораторную работу является написание программы, реализующей алгоритм решения поставленной задач на языке Turbo Pascal.

Варианты заданий для расчета

№ варианта	Задание
1	Написать программу, которая создает файл result.txt, запрашивает произвольное число и число его степеней, после чего записывает в файл result.txt требуемое число степеней исходного числа.
2	Написать программу, которая позволяет дописывать в файл phone.txt фамилии и номера телефонов. В файле каждый элемент данных (имя, фамилия, телефон) должен находиться в отдельной строке.

3	Написать программу, которая создает файл result.txt, запрашивает 7 пар чисел, после чего записывает в файл result.txt попарно эти числа, а также их среднее арифметическое.
4	Написать программу, которая позволяет дописывать в файл dates.txt фамилии и даты рождения. В файле каждый элемент данных (имя, фамилия, дата рождения) должен находиться в отдельной строке.
5	Написать программу, которая создает файл result.txt, запрашивает 5 пар чисел, после чего записывает в файл result.txt попарно эти числа, а также их среднее геометрическое.
6	Написать программу, которая позволяет найти нужные сведения в телефонном справочнике phone.txt. Программа должна запрашивать фамилию человека и выводить его телефон. Если в справочнике есть одинаковые фамилии, то программа должна вывести список всех людей, имеющих эти фамилии.
7	Написать программу, которая создает файл result.txt, запрашивает 6 произвольных чисел после чего записывает в файл result.txt исходные числа и их факториалы.
8	Написать программу, которая позволяет найти нужные сведения в списке дней рождения справочнике dates.txt. Программа должна запрашивать фамилию человека и выводить его дату рождения. Если в справочнике есть одинаковые фамилии, то программа должна вывести список всех людей, имеющих эти фамилии.
9	Написать программу, которая создает файл result.txt, запрашивает 17 произвольных чисел после чего записывает в файл result.txt исходные числа, их квадраты и квадратные корни.
10	Написать программу, которая позволяет дописывать в файл countries.txt фамилии и страну проживания. В файле каждый элемент данных (имя, фамилия, название страны) должен находиться в отдельной строке.
11	Написать программу, которая создает файл result.txt, запрашивает 12 произвольных чисел после чего записывает в файл result.txt исходные числа, их кубы и кубические корни.
12	Написать программу, которая позволяет найти нужные сведения в списке адресов в справочнике countries.txt. Программа должна запрашивать фамилию человека и выводить название страны в которой он живет. Если в справочнике есть одинаковые фамилии, то программа должна вывести список всех людей, имеющих эти фамилии.

5. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1) Прочитать задание и выписать все переменные, которые следует использовать при его решении.
2) Продумать порядок решения задачи и составить алгоритм ее решения.
3) Составить список операторов, которые понадобятся при написании программы.
4) Описать в разделе описаний все используемые переменные и их типы.
5) Составить программу решения задачи.
6) Проверить правильность программы путем введения данных, результат решения для которых известен.

7) Если необходимо, отладить программу.
8) Оформить ввод и вывод данных удобочитаемым образом.
9) Сохранить результат, как «lr08.pas»

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) Расскажите о внешних файлах.
2) Какие параметры файлов вы знаете?
3) Что понимается под файловой переменной?
4) Как инициализировать файл для записи?
5) Как инициализировать файл для чтения?

Лабораторная работа №9
ПОСТРОЕНИЕ ГРАФИЧЕСКИХ ПРИМИТИВОВ

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель работы: освоить создание элементов графического оформление при помощи модуля Graph.

Задачи работы: изучить принципы использования подключаемых модулей, изучить технологию использования модуля Graph, освоить создание графических примитивов.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Инициализация графического режима.

Для вывода графических изображений на экран турбо Паскаль предоставляет пользователю библиотеку Graph. Общий вид программы с использованием Graph имеет следующий вид:

```
Program ingr;
Uses Graph;
var
    grDriver, grMode, errCode: integer;
Begin
grDriver:= Detect;
InitGraph (grDriver, grMode,'');
If errCode= grOK Then
Begin
    {команды рисования}
End
Else
begin
writeln('Ошибка! Графический режим не удалось
открыть!');
readln;
end.
```

Чаще всего причиной возникновения ошибки при обращении к процедуре InitGraph является неправильное указание местоположения файла с драйвером графического адаптера. Настройка на местоположение драйвера осуществляется заданием маршрута поиска нужного файла в имени драйвера при вызове процедуры InitGraph. Если, например, драйвер зарегистрирован в подкаталоге DRIVERS каталога PASCAL на диске D, то нужно использовать вызов:

```
InitGraph(Driver, Mode, 'd:\Pascal\Drivers');
```

Процедура CloseGraph завершает работу адаптера в графическом режиме и восстанавливает текстовый режим работы экрана. Заголовок:

```
Procedure CloseGraph;
```

Процедура RestoreCRTMode служит для кратковременного возврата в текстовый режим. В отличие от процедуры CloseGraph не сбрасываются установленные параметры графического режима и не освобождается память, выделенная для размещения графического драйвера. Заголовок:

```
Procedure RestoreCRTMode;
```

2. Процедуры и функции библиотеки Graph.

Многие графические процедуры и функции используют указатель текущей позиции на экране, который в отличие от текстового курсора невидим. Положение этого указателя, как и вообще любая координата на графическом экране, задается относительно левого верхнего угла, который, в свою очередь, имеет координаты 0,0. Таким образом, горизонтальная координата экрана увеличивается слева направо, а вертикальная - сверху вниз.

Процедура `SetViewPort` устанавливает прямоугольное окно на графическом экране. Заголовок:

```
Procedure SetViewPort(XI,Y1,X2,Y2: Integer; ClipOn: Boolean);
```

Здесь `X1...Y2` - координаты левого верхнего и правого нижнего углов окна; `ClipOn` - выражение типа `Boolean`, определяющее «отсечку» не умещающихся в окне элементов изображения.

Координаты окна всегда задаются относительно левого верхнего угла экрана. Если параметр `ClipOn` имеет значение `True`, элементы изображения, не умещающиеся в пределах окна, отсекаются, в противном случае границы окна игнорируются.

Процедура `GetViewSettings` возвращает координаты и признак отсечки текущего графического окна. Заголовок:

```
Procedure GetViewSettings(var Viewlnfo: ViewPortType);
```

Здесь `Viewlnfo` - переменная типа `ViewPortType`.

Процедура `ClearDevice` Очищает графический экран. После обращения к процедуре указатель устанавливается в левый верхний угол экрана, а сам экран заполняется цветом фона, заданным процедурой `SetBkColor`. Заголовок:

```
Procedure ClearDevice;
```

Процедура `ClearViewPort` очищает графическое окно, а если окно не определено к этому моменту - весь экран. При очистке окно заполняется цветом с номером 0 из текущей палитры. Указатель перемещается в левый верхний угол окна. Заголовок:

```
Procedure ClearViewPort;
```

Ниже приведены наиболее распространенные процедуры графического модуля.

Процедура `Rectangle` вычерчивает прямоугольник с указанными координатами углов. Заголовок:

```
Procedure Rectangle(X1,Y1,X2,Y2: Integer);
```

Здесь `X1... Y2` - координаты левого верхнего и правого нижнего углов прямоугольника. Прямоугольник вычерчивается с использованием текущего цвета и текущего стиля линий.

Процедура `Circle` вычерчивает окружность. Заголовок:

```
Procedure Circle(X,Y: Integer; R: Word);
```

Здесь `X, Y`- координаты центра; `R` - радиус в пикселях.

Окружность выводится текущим цветом. Толщина линии устанавливается текущим стилем, вид линии всегда `SolidLn` (сплошная). Процедура вычерчивает правильную окружность с учетом изменения линейного размера радиуса в зависимости от его направления относительно сторон графического экрана, т.е. с учетом коэффициента `GetAspectRatio`. В связи с этим параметр `R` определяет количество пикселей в горизонтальном направлении.

Процедура `Ellipse` вычерчивает эллипсную дугу. Заголовок:

```
Procedure Ellipse(X,Y: Integer; BegA,EndA,RX,RY: Word);
```

Здесь X, Y - координаты центра; BegA, EndA - соответственно начальный и конечный углы дуги; RX, RY- горизонтальный и вертикальный радиусы эллипса в пикселях.

Процедура SetColor устанавливает текущий цвет для выводимых линий и символов. Заголовок:

```
Procedure SetColor(Color: Word);
```

Здесь Color - текущий цвет.

Функция GetColor возвращает значение типа Word, содержащее код текущего цвета. Заголовок:

```
Function GetColor: Word;
```

Процедура SetBkColor устанавливает цвет фона. Заголовок:

```
Procedure SetBkColor(Color: Word);
```

Здесь Color - цвет фона. В отличие от текстового режима, в котором цвет фона может быть только темного оттенка, в графическом режиме он может быть любым. Установка нового цвета фона немедленно изменяет цвет графического экрана.

Процедура SetFillStyle устанавливает стиль (тип и цвет) заполнения. Заголовок:

```
Procedure SetFillStyle(Fill,Color: Word);
```

Здесь Fill - тип заполнения; Color - цвет заполнения.

Пример

Программа демонстрирует все стандартные типы заполнения.

```
Uses Graph, CRT;
var
d,r,e,k,j,x,y: Integer;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
x := GetMaxX div 6;{Положение графика}
y := GetMaxY div 5;{на экране}
for j := 0 to 2 do{Два ряда}
for k := 0 to 3 do{По четыре квадрата}
begin
Rectangle((k+1)*x,(j+1)*y,(k+2)*x,(j+2)*y);
SetFillStyle(k+j*4,j+1);
Bar((k+1)*x+1,(j+1)*y+1,(k+2)*x-1,(j+2)*y-1)
end;
if ReadKey=#0 then k := ord(ReadKey);
CloseGraph
end
end.
```

Процедура FloodFill заполняет произвольную замкнутую фигуру, используя текущий стиль заполнения (узор и цвет). Заголовок:

```
Procedure FloodFill(X,Y: Integer; Border: Word);
```

Здесь X, Y- координаты любой точки внутри замкнутой фигуры; Border - цвет граничной линии.

Если фигура незамкнута, заполнение «разольется» по всему экрану. Следует учесть, что реализованный в процедуре алгоритм просмотра границ замкнутой фигуры не отличается совершенством. В частности, если выводятся подряд две пустые строки, заполнение прекращается.

Процедура Bar заполняет прямоугольную область экрана. Заголовок:

```
Procedure Bar(X1,Y1,X2,Y2: Integer);
```

Здесь X1...Y2 - координаты левого верхнего (и правого нижнего углов закрашиваемой области. Процедура закрашивает (но не обводит) прямоугольник текущим образцом узора и текущим цветом, которые устанавливаются процедурой SetFillStyle.

Процедура Bar3D вычерчивает трехмерное изображение параллелепипеда и закрашивает его переднюю грань . Заголовок:

```
Procedure Bar3D (X1,Y1,X2,Y2,Depth: Integer; Top: Boolean);
```

Здесь X1... Y2 - координаты левого верхнего и правого нижнего углов передней грани; Depth - третье измерение трехмерного изображения («глубина») в пикселях; Top - способ изображения верхней грани.

Процедура FillPoly обводит линией и закрашивает замкнутый многоугольник. Заголовок:

```
Procedure FillPoly(N: Word; var Coords);
```

Здесь N - количество вершин замкнутого многоугольника; Coords - переменная типа PointType, содержащая координаты вершин.

Иногда бывает целесообразным создав один элемент графической оболочки использовать его несколько раз (например однотипные кнопки). В этом случае могут быть полезными функции сохранения и выдачи изображений.

Функция ImageSize возвращает размер памяти в байтах, необходимый для размещения прямоугольного фрагмента изображения. Заголовок:

```
Function ImageSize(X1,Y1,X2,Y2: Integer): Word;
```

Здесь X1... Y2 - координаты левого верхнего и правого нижнего углов фрагмента изображения.

Процедура Getlmage помещает в память копию прямоугольного фрагмента изображения. Заголовок:

```
Procedure Getlmage(X1,Y1,X2,Y2: Integer; var Buf);
```

Здесь X1...Y2 - координаты углов фрагмента изображения; Buf - переменная или участок кучи, куда будет помещена копия видеопамяти с фрагментом изображения. Размер Buf должен быть не меньше значения, возвращаемого функцией ImageSize с теми же координатами X1....Y2.

Процедура PutImage выводит в заданное место экрана копию фрагмента изображения, ранее помещенную в память процедурой Getlmage. Заголовок:

```
Procedure Putlmage(X,Y: Integer; var Buf; Mode: Word);
```

Здесь X,Y- координаты левого верхнего угла того места на экране, куда будет скопирован фрагмент изображения; Buf - переменная или участок кучи, откуда берется изображение; Mode - способ копирования.

Как видно, координаты правого нижнего угла не указываются, так как они полностью определяются размерами вновь выводимой на экран копии изображения.

Координаты левого верхнего угла могут быть какими угодно, лишь бы только выводимая копия уместилась в пределах экрана (если копия не может разместиться на экране, она не выводится и экран остается без изменений). Параметр `Mode` определяет способ взаимодействия вновь размещаемой копии с уже имеющимся на экране изображением. Взаимодействие осуществляется путем применения кодируемых этим параметром логических операций к каждому биту копии и изображения.

При выполнении данной работы широко используются возможности текствого вывода. Описываемые ниже стандартные процедуры и функции поддерживают вывод текстовых сообщений в графическом режиме. Это не одно и то же, что использование процедур `Write` или `WriteLn`. Дело в том, что специально для графического режима разработаны процедуры, обеспечивающие вывод сообщений различными шрифтами в горизонтальном или вертикальном направлении, с изменением размеров и т.д. Однако в стандартных шрифтах, разработанных для этих целей фирмой Borland, отсутствует кириллица, что исключает вывод русскоязычных сообщений.

С другой стороны, процедуры `Write` и `WriteLn` после загрузки в память второй половины таблицы знакогенератора (а эта операция легко реализуется в адаптерах EGA и VGA) способны выводить сообщения с использованием национального алфавита, но не обладают мощными возможностями специальных процедур.

Ниже описываются стандартные средства модуля Graph для вывода текста.

Процедура `OutText` выводит текстовую строку, начиная с текущего положения указателя. Заголовок:

```
Procedure OutText(Txt: String);
```

Здесь `Txt` - выводимая строка.

Процедура `OutTextXY` выводит строку, начиная с заданного места. Заголовок:

```
Procedure OutTextXY (X,Y: Integer; Txt: String);
```

Здесь `X, Y` - координаты точки вывода; `Txt` - выводимая строка. Отличается от процедуры `OutText` только координатами вывода. Указатель не меняет своего положения.

Процедура `SetTextStyle` устанавливает стиль текстового вывода на графический экран. Заголовок:

```
Procedure SetTextStyle(Font,Direct,Size: Word);
```

Здесь `Font` - код (номер) шрифта; `Direct` - код направления; `Size` - код размера шрифта.

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

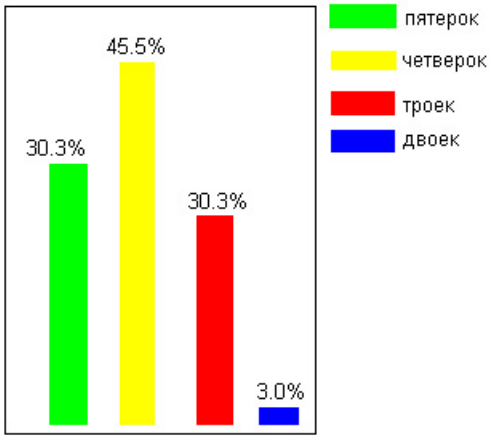
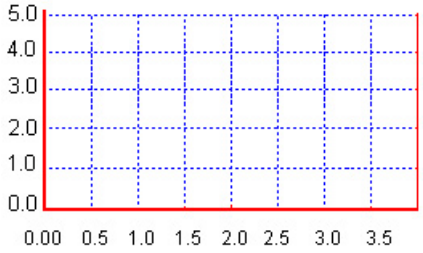
Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

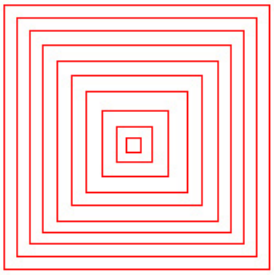
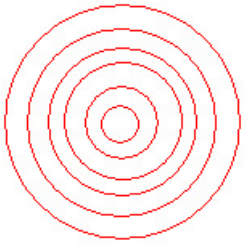
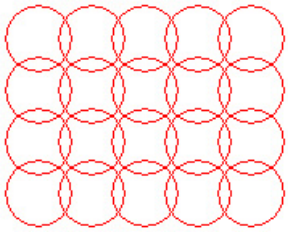
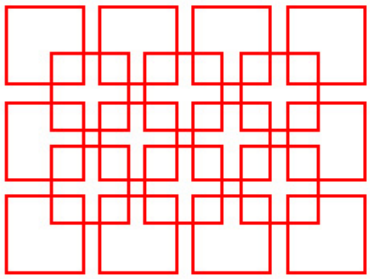
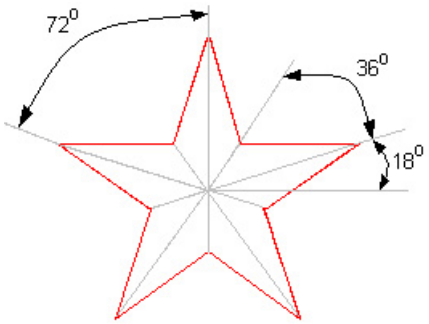
1) Операционная система Windows XP;
2) Microsoft Калькулятор Плюс;
3) Turbo Pascal 7.0 или ABC Pascal.

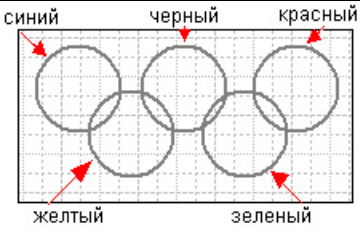
4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Заданием на лабораторную работу является написание программы, реализующей алгоритм решения поставленной задач на языке Turbo Pascal.

Варианты заданий

№ варианта	Задание
1	Написать программу, которая выводит круговую диаграмму, отражающую товарооборот (в процентах) книжного магазина. Исходные данные (объем продаж в рублях по категориям: книги, журналы, открытки и канцтовары) вводятся во время работы программы. Пример диаграммы приведен ниже. 
2	Написать программу, которая выводит на экран гистограмму успеваемости студентов группы, например, по итогам контрольной работы. Исходные данные следует ввести в алфавитно-цифровом режиме работы. 
3	Написать программу, которая рисует окружность, движущуюся по экрану.
4	Написать программу, которая выводит на экран точечный график функции y = 0,5x2 + 4x — 3. Диапазон изменения аргумента — от —15 до 5, шаг аргумента — 0,1. График вывести на фоне координатных осей, точка пересечения которых должна находиться в центре экрана.
5	Написать программу, которая выводит на экран оцифрованную координатную сетку. 
6	Написать программу, которая выводит на экран узор, изображенный ниже.

	
7	Написать программу, которая выводит на экран изображенный ниже узор. Окружности должны быть разного цвета (см. таблицу кодировки цветов). 
8	Написать программу, которая выводит изображенный ниже узор. 
9	Написать программу, которая выводит изображенный ниже узор. 
10	Написать программу, которая выводит на экран контур пятиконечной звезды. 
11	Написать программу, которая выводит на экран флаг Олимпийских игр. Изображение флага приведено ниже (одной клетке соответствует пять пикселов).

	
12	Написать программу, которая выводит на экран изображение шахматной доски.

5. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1) Прочитать задание и выписать все переменные, которые следует использовать при его решении.
2) Продумать порядок решения задачи и составить алгоритм ее решения.
3) Составить список операторов, которые понадобятся при написании программы.
4) Описать в разделе описаний все используемые переменные и их типы.
5) Составить программу решения задачи.
6) Проверить правильность программы путем введения данных, результат решения для которых известен.
7) Если необходимо, отладить программу.
8) Оформить ввод и вывод данных удобочитаемым образом.
9) Сохранить результат, как «lr09.pas»

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) Расскажите об инициализации графического модуля.
2) Расскажите об окнах и координатах.
3) Какие процедуры, регулирующие типы обводки заполнения вы знаете?
4) Какие процедуры, предназначенные для рисования геометрических фигур вы знаете?
5) Какие существуют процедуры для вывода текста в графическом режиме?

Лабораторная работа №10
ПРОЦЕДУРЫ И ФУНКЦИИ

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель работы: усвоение принципов модульного программирования.

Задачи работы: изучить основные положения модульного программирования, получить навыки создания процедур и функций.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Основные принципы структурного программирования.

При разработке сложных программ используют так называемый структурный подход к программированию и нисходящее проектирование программ, когда сложная программ разбивается на более (или менее) функционально-законченные части, каждая из которых проще исходной программы. Такие программы легче отлаживать и использовать. Отдельные части программы называют подпрограммами. Использование подпрограмм наиболее эффективно в тех случаях, когда одна и та же подпрограмма может использоваться в программе не один раз, возможно с различными параметрами. Это позволяет экономить память компьютера.

Подпрограммы, в свою очередь, могут разбиваться на более мелкие части, реализуемые также в виде подпрограмм более низкого уровня.

В языке Турбо Паскаль используют подпрограммы двух типов: процедуры (Procedure) и функции (Function). Подпрограммы по структуре сходны с программой, но они обязательно имеют оригинальное имя, которое указывается в заголовке. Подпрограммы описываются в разделе описаний, использующих (вызывающих) их программ (или подпрограмм).

2. Процедуры.

Описание процедур в Паскале имеет вид:

```
Procedure   Имя процедуры (формальные параметры);
          Раздел  описаний
     Begin
     Раздел операторов
     End;
```

Формальные параметры вместе с круглыми скобками могут отсутствовать. Формальные параметры представляют собой список переменных с указанием их типа. Все типы, используемые в заголовках процедур и функций, кроме простых, должны быть описаны в подразделе Type вызывающей эти процедуры или функции программной единицы. Те параметры, которые изменяются в процедуре, называют выходными и перед ними в заголовке процедуры обязательно ставится слово Var. Параметры, имеющие файловый тип, должны быть обязательно описаны как Var - параметры и в процедурах и в функциях.

Вызов процедуры в использующих ее программных единицах (основной программе или подпрограммах) имеет следующий вид:

```
Имя процедуры (фактические параметры);
```

Фактические параметры могут отсутствовать вместе со скобками, в том случае, если нет формальных параметров в описании указанной процедуры.

Если параметры все же необходимы, то между фактическими и формальными параметрами должно быть установлено соответствие по их количеству, порядку следования и типу данных.

Имена фактических и формальных параметров могут быть как одинаковыми, так и различными.

Пусть в программе две процедуры P1 и P2 вызываются из основной программы. В свою очередь в процедуре P1 используется процедура P11 и она должна быть описана в разделе описаний вызывающей ее процедуры P1.

```
Раздел описаний основной программы
               Procedure P1;
                    Раздел описаний процедуры P1
…
Procedure P11
     Раздел описаний процедуры P11
     Begin
     Раздел операторов процедуры P11
     End;
Procedure P1
     Begin
     Раздел операторов процедуры P1
     End;
Procedure P2;
     Раздел описаний процедуры P2
     Begin
     Раздел операторов процедуры P2
     End;
BEGIN
…
Раздел операторов основной программы
…
END.
```

Имена, объявленные в разделе описаний основной программы, действуют в разделе операторов основной программы и в любой подпрограмме. Эти имена называются **глобальными**. Имена, объявленные в какой-либо подпрограмме, действуют в этой подпрограмме и в любой, объявленной в ней процедуре или функции. Такие имена называются **локальными**. Они недоступны для операторов основной программы. Область действия меток переходов в пределах каждой программной единицы своя. Нельзя перейти по оператору GOTO из одной процедуру в другую.

Рассмотрим **пример** разработки программы, содержащей две процедуры, каждая из которых используется дважды с различными фактическими параметрами.

Даны два массива M1 и M2, содержащие K1 и K2 целых чисел, соответственно. Определить максимальные числа в каждом из этих массивов, сравнить найденные значения между собой и вывести большее из них на экран.

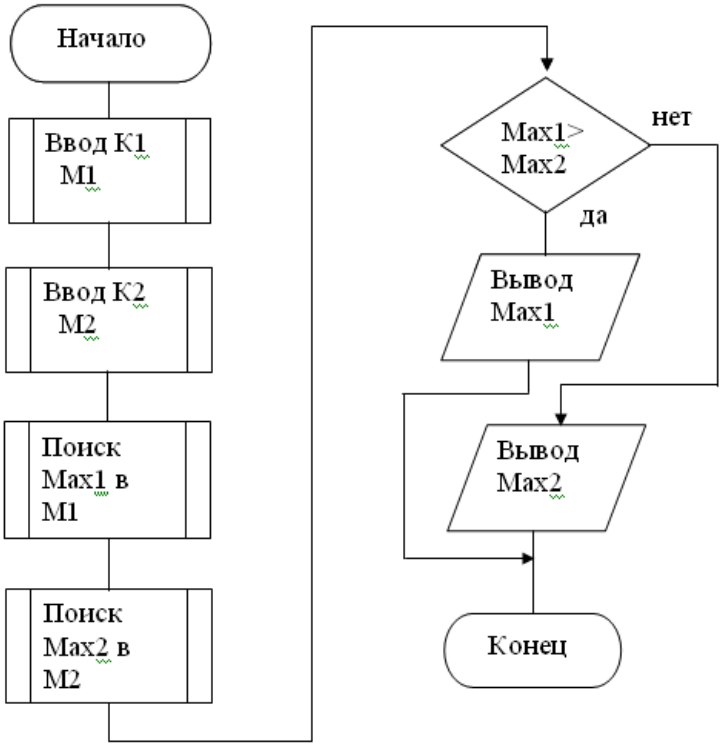


Рис. 1. Схема алгоритма.

Выделим глобальные переменные, которые используются в главной программе: M1, M2, K1, K2, Max1, Max2.

Текст программы:

```
Uses crt;
     Type Tmas=array[1..1000] of integer;
     Var M1, M2: Tmas;
         K1, K2, Max1, Max2 : integer;
{ Процедура ввода длины массива и самого массива }
Procedure Vvod(Var K:integer; Var M:Tmas);
     Var i:integer;
          Begin
               Write('  Введите длину массива');
               Readln(K);
          Writeln('  Введите элементы массива целых чисел, через пробел');
          For i:=1 to K do
          Read(M[i]); readln;
End; { конец процедуры ввода}
{ Процедура поиска максимального элемента в массиве}
Procedure Poisk_max(K:integer; M:Tmas; Var Max:integer);
     Var i:integer;
          Begin
          Max:=M[1];      { За максимум принимаем первый элемент}
          For i:=2  to K do
          If M[i]>Max then  Max:=M[i]; {Запоминаем новый максимум}
End;
{  Начало основной программы}
Begin
     Clrscr;
     Writeln('  Ввод первого массива');
     Vvod(K1, M1);
     Writeln('  Ввод второго массива');
     Vvod(K2,M2);
     Poisk_max(K1, M1, Max1);
```

```
     Poisk_max(K2, M2, Max2);
     If Max1>Max2 then writeln(' Max1 больше и оно =  ',Max1)
          Else writeln(' Max2 больше и оно =  ',Max2);
     Readkey; { Останов для просмотра результатов}
End.
```

3. Функции.

Подпрограммы - функции (Function) имеют следующие отличительные особенности (по сравнению с процедурами):

1) Функция имеет только один результат выполнения. Этот результат обозначается именем функции и передается в вызвавщую эту функцию программную единицу.
2) Для функции обязательно указывается ее тип. Тип может быть простым (скалярным) или строковым.

Описание функции в Паскале имеет вид:

```
Function   Имя функции (формальные параметры):Тип результата;
          Раздел  описаний
Begin
Раздел операторов
End;
```

Вызов функции производится по ее имени с указанием фактических параметров. Аналогично процедурам, фактические и формальные параметры в функции могут отсутствовать.

Правила использования фактических и формальных параметров, а также локальных и глобальных переменных, совпадают с использованием их в процедурах.

Если функция кроме выдачи своего значения меняет значения каких-либо глобальных переменных, то говорят, что она имеет побочный эффект.

Рассмотрим **пример** создания программ с использованием функций. Пусть необходимо произвести следующие вычисления:

$$S = \frac{\sum_{i=1}^{n} x_i + \sum_{j=1}^{m} y_j}{a^n + b^m},$$

где a, b, n, m - целые переменные;

$\{x_i\}$, $\{y_j\}$ - массивы, содержащие n и m вещественных чисел, соответственно.

В Паскале нет стандартных функций сумммирования элементов массива и возведения чисел в степень больше 2. Разработаем свои функции для решения этих задач и будем использовать их для решения поставленной задачи.

Текст программы

```
Uses crt;
     Type Tmas=array[1..100] of real;
     Var  a,b,n,m,i,j:byte;
          X,Y:Tmas;
          S:Real;
Function Summa(Dlmas:byte; Mas:Tmas):Real;
     Var Sum:real;  i:byte;
     Begin
     Sum:=0.0;
     For i:=1 to Dlmas do
     Sum:=Sum+Mas[i];
```

```
     Summa:=Sum;
End;
Function step(Pok:byte;Osn:byte):real;
     Var i:byte; St:real;
     Begin
     St:=Osn;
     For i:=2 to Pok do
     St:=St*Osn;
     step:=St;
End;
{ Главная программа}
BEGIN
     Write('  Введите длину первого массива - N  ');
     Readln(N);
     Write('  Введите длину второго массива - M  ');
     Readln(M);
     Writeln('   Введите элементы массива X');
               For i:=1 to N do Read(X[i]); readln;
     Writeln('   Введите элементы массива Y');
               For j:=1 to M do Read(Y[j]); readln;
     Write('   Введите a и b ');  Readln(a,b);
     S:=(Summa(N,X)+Summs(M,Y))/(step(N,a)+step(M,b));
     Writeln('  Полученный результат  S = ',S);readkey;
END.
```

4. Рекурсии.

Рекурсия - это такой способ организации вычислительного процесса, при котором подпрограмма обращается сама к себе. Такая рекурсия называется прямой. Рекурсии позволяют писать более короткие программы, но при выполнении они работают медленнее и могут вызвать переполнение стека, так как при каждом входе в подпрограмму ее локальные переменные размещаются в программном стеке, размер которого ограничен.

Типичным **примером** прямой рекурсии является вычисление n!.

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$$

Текст программы с использованием прямой рекурсии для вычисления n!.

```
Var n:integer;
{Рекурсивная функция}
Function Fact(n:integer):real;
     Begin
     If n=0 then Fact:=1
     Else Fact:=n*Fact(n-1);
End;
{  Главная программа}
Begin
     Repeat
     Writeln(' Введите положительное n<=33');
     Readln(n);
     Writeln(Fact(n));
     Until Eof;
  End.
```

Примечание: для выхода из этой программы можно задать значение n>33 или нажать Ctrl/z и Enter. При n>33 возникает переполнение при умножение чисел с плавающей запятой.

Рекурсивный вызов может быть также косвенным. В этом случае подпрограмма обращается к себе опосредованно, путем вызова другой подпрограммы, в которой, в свою очередь, содержится обращение к первой. Такой вызов иногда называют закольцованным.

Для реализации такой возможности в ТР используется опережающее описание процедур и функций и директива Forward.

Для этого в самом начале программы вставляют только заголовки процедуры или функции в виде:

```
Procedure  имя-процедуры(параметры);Forward;
```

или

```
Function имя_функции(параметры);Forward;
```

Позже, в необходимых местах, описываются сами процедуры или функции в обычном виде и в их заголовке параметры уже указывать не нужно.

Например: процедура с именем A вызывает процедуру с именем B, а процедура B, в свою очередь, вызывает процедуру A.

```
Procedure A(y:тип);Forward;  {Опережающее описание процедуры A}
Procedure B(x:тип);  { Заголовок процедуры B}
     . . .   {Раздел описаний процедуры B}
     begin
     . . .
     A(p);  {Вызов процедуры A из B}
     . . .
end;
Procedure A;  {Основное описание процедуры A}
     . . .                  {Раздел описаний процедуры A}
     Begin
     . . .
     B(g);  {Вызов процедуры B из A}
     . . .
End;
```

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

1) Операционная система Windows XP;
2) Microsoft Калькулятор Плюс;
3) Turbo Pascal 7.0 или ABC Pascal.

4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Заданием на лабораторную работу является написание программы, реализующей алгоритм решения поставленной задач на языке Turbo Pascal.

Варианты заданий

№ вар.	Функция

1	$$F = \sum_{i=1}^{t}(x_i + a)^n + \sum_{j=1}^{m}\frac{y_j}{c^m}$$
2	$$F = \frac{(a+b)^n}{\sum_{j=1}^{m}((a+b)^m - y_j)}$$
3	$$F = \sum_{i=1}^{n} x_i + a^n + \sum_{k=1}^{m}\frac{y_k}{ac^m}$$
4	$$F = \frac{(a^m - b^n)}{\sum_{j=1}^{m}((a-b)^m - y_j)}$$
5	$$F = \frac{\sum_{i=1}^{t}(x_i - c^t)}{c^{t+2}\cdot(t-2)!}$$
6	$$F = \frac{\sum_{i=1}^{k}(x_i - c)^k}{c!-(k-c)!}$$
7	$$F = \sqrt{(x^n + y^m)} + \frac{(n+m)!}{\sum_{k=1}^{t} z_k - \sum_{j=1}^{p} s_j}$$
8	$$F = \frac{\sum_{i=1}^{s}(x_i - y_i)^s + s!}{\sum_{k=1}^{n}(a_k - b_k)^n - n!}$$

9	$F = \dfrac{\sqrt{a^k - b^t}}{\sum\limits_{i=1}^{k}(x_i + b)} + \sum\limits_{i=1}^{t}(y_i + a)$
10	$F = \dfrac{\sum\limits_{i=1}^{n}(x_i - y_i)^n - (n+3)!}{\sum\limits_{k=1}^{n}(a_k - b_k)^n - n!}$
11	$F = \dfrac{\sqrt{a^{k+2} + b^{t-2}}}{\sum\limits_{i=1}^{t}(a^k x_i - b)} + \sum\limits_{i=1}^{k}(b^t y_i - a)$
12	$F = \dfrac{\sqrt{a^k + b^t}}{\sum\limits_{i=1}^{k} x_i} + \sum\limits_{i=1}^{t} y_i$

5. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1) Прочитать задание и выписать все переменные, которые следует использовать при его решении.
2) Продумать порядок решения задачи и составить алгоритм ее решения.
3) Составить список операторов, которые понадобятся при написании программы.
4) Описать в разделе описаний все используемые переменные и их типы.
5) Составить программу решения задачи.
6) Проверить правильность программы путем введения данных, результат решения для которых известен.
7) Если необходимо, отладить программу.
8) Оформить ввод и вывод данных удобочитаемым образом.
9) Сохранить результат, как «lr10.pas»

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) Расскажите о модульном программировании.
2) Что такое функция?
3) Что такое процедура?
4) В чем принципиальное отличие функции и процедуры?
5) Что такое рекурсия? Приведите пример использования рекурсии.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Основная литература

1. Мурат Е.П. Информатика III : учебное пособие / Мурат Е.П.. — Ростов-на-Дону, Таганрог : Издательство Южного федерального университета, 2018. — 150 c. — ISBN 978-5-9275-2689-5. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/87415.html

2. Старыгина С.Д. Информатика: технологии и офисное программирование : учебное по-собие / Старыгина С.Д., Нуриев Н.К., Нургалиева А.А.. — Казань : Казанский национальный исследовательский технологический университет, 2018. — 232 c. — ISBN 978-5-7882-2565-4. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/100670.html.

3. Оболонин И.А. Основы компьютерного проектирования в инфокоммуникационных технологиях : учебно-методическое пособие / Оболонин И.А.. — Новосибирск : Сибирский государственный университет телекоммуникаций и информатики, 2018. — 250 c. — ISBN 2227-8397. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: http://www.iprbookshop.ru/84070.html.

4. Кузнецов С.Д. Основы баз данных: курс лекций. Учеб. пособие. - М.: Интернет-Университет Информ. Технологий, 2005.- 488 с.

5 Джорджес Г. 50 эффективных приемов обработки цифровых фотографий с помощью Photoshop / Г. Джорджес; пер. с англ. С.Д. Панасюка; под ред. В.С. Иващенко. - М. [и др.] : Диалектика, 2006 .— 464 с.

4. Гурский, Ю. Компьютерная графика: Photoshop CS3, CorelDRAW X3, Illustrator CS3 / Ю. Гурский, И. Гурская, А. Жвалевский .— М.[и др.] : Питер, 2008 .— 992 с.

Дополнительная литература

1. Мельников В.П. Информационные технологии. Учебник для ВУЗов. М.: Академия, 2008. – 426 с.

2. Кузин А.В. Базы данных: учеб. пособие для вузов / А. В. Кузин, С. В. Левонисова .— 2-е изд., стер .— М.: Академия, 2008. – 316 с.

3. Грекул В.И. Проектирование информационных систем: курс лекций: учеб. пособие для вузов / В.И. Грекул, Г.Н. Денищенко, Н.Л. Коровкина. - М.: Интернет-Ун-т Информ. Технологий, 2005. — 304 с.

4. Волкова, Е.В. Художественная обработка фотографий в Photoshop / Е.В. Волкова. - М.[и др.] : Питер, 2005 .— 269 с.

Периодические издания

1) Компьютерра : компьютерный еженедельник. — М. : ООО Журнал"Компьютерра".

2) Мир ПК : журнал для пользователей персональных компьютеров. — М. : Открытые системы.

Интернет-ресурсы

1) http://cictemnik.ru/ - сайт, посвященный современным компьютерным системам;

2) http://www.computerra.ru/ - сайт «Компьютера онлайн»