Министерство образования Российской Федерации
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

В.А. БОЛЬШАКОВ, Г.И.ВОРОНОВ, Л.А. САВВАТЕЕВА

Информатика

Лабораторный практикум по программированию на Турбо-Паскале

Санкт-Петербург
2002

УДК 681.3.06

Большаков В.А., Воронов Г.И. Савватеева Л.А. Информатика. Лабораторный практикум по программированию на Турбо-Паскале. СПб.: Изд. РГГМУ, 2002. – 190 с.

Рецензент: д.т.н., проф. ГЭТУ(ЛЭТИ) Яшин А.И.

В лабораторный практикум по программированию на языке Паскаль в среде Турбо-Паскаль включены лабораторные работы, выполняемые студентами первого курса гидрометеорологического университета при изучении дисциплины "Информатика" в разделе программирование на ПЭВМ.

Практикум рассчитан на обучение основам "классического" алгоритмического программирования, и не рассматривает методы объектного программирования. Полный курс включает девять лабораторных работ, последовательно рассматривающих основные операторы, средства и методы программирования. Каждой лабораторной работе предшествует описание изучаемых аспектов языка и рассматриваемых алгоритмов, так что данный материал может использоваться и для самостоятельного обучения программированию на языке Паскаль. Для каждой работы приведено 30 вариантов заданий и один дополнительный разобранный вариант, по которому составлена программа.

Предисловие

Сегодня невозможно представить себе современного специалиста, не владеющего персональным компьютером. Особенно ценно при этом умение подчинить персональный компьютер своим требованиям, грамотно поставить, спроектировать и решить профессиональную задачу. Для этих целей в курсе информатики студентов первых курсов знакомят с основами программирования, чтобы с самого начала обучения студент освоил инструмент, который можно было бы в дальнейшем применять для решения как учебных, так и любых других практических задач.

В качестве базового языка программирования выбран один из самых распространенных алгоритмических языков – язык Паскаль, который, благодаря своей простоте и стройности, с нашей точки зрения, наиболее подходит для обучения студентов.

Лабораторный практикум предназначен в первую очередь для обучения основам и принципам программирования, во вторую – для реализации программирования на конкретном алгоритмическом языке Паскаль и в третью – для освоения средства программирования – среды Турбо-Паскаль. Лабораторный практикум ориентирован на освоение идей структурного программирования, начиная от простейших линейных алгоритмов, и до использования внешних подпрограмм и функций.

Задания к лабораторным работам проходили "обкатку" на студентах первого курса всех факультетов гидрометеорологического института. Никаких специальных вопросов, относящихся к профильным дисциплинам института, в практикуме не предусмотрено.

Это также связано с тем фактом, что обучение программированию проводится на первом курсе, когда студенты еще не обладают достаточными знаниями в высшей математике, не проходили разделов теории вероятностей и математической статистики, не начали изучать специальные дисциплины (приемы и методы которых студентам и приходится впоследствии реализовывать на ЭВМ). Поэтому основное внимание уделяется обучению и закреплению навыков по использованию типовых алгоритмов, рекомендуемых приемов и правил программирования, и, главное, умению построить алгоритм решения задачи.

Как уже упоминалось ранее, задания к лабораторным работам не ориентированы на какую-либо конкретную версию Турбо-Паскаля (за исключением лабораторных, посвященных изучению библиотеки Турбо-Паскаля), и могут быть использованы и при изучении других языков. Однако пояснения и примеры программ выполнены на языке Паскаль, а все программы проверялись в среде Турбо-Паскаль 6-ой версии.

Полный курс лабораторных работ включает девять заданий (по одному варианту каждой лабораторной). В каждой лабораторной изучается несколько новых синтаксических конструкций языка и структурных компонентов программных модулей и закрепляются ранее рассмотренные вопросы.

В лабораторных работах изучаются следующие темы:

- Линейные алгоритмы
- Алгоритмы с ветвлениями
- Алгоритмы простейших циклов
- Работа с одномерным массивом
- Обработка двумерных массивов
- Итеративные циклы
- Процедуры. Функции пользователя
- Графика в Турбо-Паскале
- Динамические переменные (списки)

Дополнительно рассмотрены некоторые вычислительные алгоритмы и соответствующие им программы. В данном курсе не рассматриваются вопросы объектного программирования из-за ограниченных возможностей по выделяемым на обучение временным и техническим ресурсам.

Все варианты подобраны так, чтобы в серии работ с одним номером встретились различные типы алгоритмов (итеративные и арифметические циклы, вычисления сумм и поиски экстремумов, сортировки массивов и обработки двумерных массивов) и обрабатывались данные разных типов – числовые (целочисленные и вещественные), символьные и логические. В конце каждого набора заданий приведен 31-й вариант средней сложности, который подробно разбирается и по нему составляется программа.

В начале каждой лабораторной работы рассматриваются основные моменты, связанные с изучаемыми в данной работе особенностями программирования. Поэтому лабораторный практикум может быть использован и для самостоятельного изучения как принципов программирования, так и их конкретной реализации на языке Паскаль.

Авторы выражают благодарность старшему преподавателю кафедры МИТ Волгину Д.И. за помощь, оказанную при проверке лабораторных работ и оформлении настоящего практикума.

Общая схема выполнения лабораторной работы

Выполнение каждой лабораторной обязательно включает несколько этапов, ни один из которых не должен пропускаться. Начинаться работа должна с разбора задания и составления списка объектов, которые будут встречаться в программе (в изображении алгоритма). Этот список объектов в виде таблицы идентификаторов (таблицы распределения памяти) особенно важен в первых работах. Как показал опыт, неумение выделить и описать используемые в программе данные в 90% случаев приводит к ошибкам в программах, например, порядковый номер путают со значением элемента в последовательности, общее количество объектов – с количеством объектов, удовлетворяющих какому-либо критерию и так далее.

Изображение алгоритмов следует выполнять максимально подробно, особенно в первых работах. Нельзя опускать изображения блоков алгоритма, если они "само собой разумеются". Можно позволить объединять в блоке несколько однотипных действий (например, очистку нескольких переменных), но запрос на ввод какого-либо значения и собственно ввод этого значения в переменную с клавиатуры необходимо изображать двумя отдельными блоками, пока обучающиеся не привыкнут, что всякому вводу данных пользователем программы должен предшествовать запрос со стороны программы (ПЭВМ).

Очевидно, что каждая лабораторная должна быть выполнена на компьютере, и только после этого может быть оформлена и представлена к защите. Отчет должен содержать титульный лист, задание, таблицу идентификаторов, блок-схему алгоритма, листинг программы и протокол работы программы (в виде распечатки текстового файла с исходными данными и результатами работы программы). Рекомендуемое оформление каждого раздела приведено ниже. Важно отметить, что с первой же лабораторной работы следует придерживаться порядка и аккуратности в оформлении лабораторной работы. Это важный фактор обучения программированию, так как такой стиль работы в дальнейшем существенно уменьшает количество ошибок и ускоряет процесс написания и отладки программ.

При наличии достаточного количества технических ресурсов (принтера и расходных материалов к нему) все части лабораторной работы (кроме блок-схемы алгоритма) могут распечатываться на принтере. Как минимум, в отчете должна присутствовать распечатка текста программы и результатов ее работы, остальные разделы могут быть выполнены вручную.

При написании текста программы следует придерживаться некоторых рекомендаций. В частности, текст программы должен содержать:

- Комментарий с указанием фамилии программиста и номера группы.
- Описание всех используемых в программе идентификаторов (имен).

- Ввод требуемого количества данных из указанного в задании файла или иной способ задания исходных данных, как сказано в задании.
- Вывод в выводной текстовый файл (до четвертой работы – на экран) введенных исходных числовых значений под заголовком "Исходные данные".
- Проведение обработки числового материала в соответствии с заданием.
- Вывод результатов обработки в выводной текстовый файл (до четвертой работы – на экран) под заголовком "Результаты расчета", с пояснениями, если результат не может быть выведен.

Текст программы (листинг) рекомендуется писать с выделением вложенных циклов, составных операторов и структур смещением на две – три позиции, что облегчает чтение программы и упрощает отладку (поиск пропущенных программных скобок и несоответствий алгоритму).

Листинг и вывод результатов следует размещать так, чтобы в дальнейшем был возможен вывод на печать. Для этого следует учитывать, что ширина стандартного листа бумаги реально позволяет печатать текст шириной до 75 символов в строке, листа из тетради – до 60 символов в строке, двойного листа из тетради – до 76 символов в строке (как правило, это стандартный шрифт матричного принтера).

Наконец, несколько слов о рекомендуемой литературе. Для работы желательно иметь описание языка и инструментальной среды используемой системы программирования. В частности, удобно пользоваться книгами [1,2,4], где описано и то, и другое.

Лабораторная работа № 1
Алгоритмы линейной структуры

Задачи лабораторной работы

Вопросы, изучаемые в работе

- Построение простейшей программы линейной структуры.
- Использование операторов присваивания и простейшего вывода для данных вещественного типа.
- Использование оператора описания переменных для данных вещественного типа.
- Изучение правил написания и вычисления арифметических выражений.
- Применение в выражениях встроенных математических функций языка Паскаль.

Задание (общее ко всем вариантам).

Запрограммировать вычисление заданной функции, вычислить и вывести на экран результат при указанных значениях аргументов. Проверить программу по приведенному в задании ответу. Оформить отчет по лабораторной работе в соответствии с образцом, приведенным для варианта № 31. Таблицу идентификаторов, блок-схему алгоритма и текст программы аккуратно выполнить от руки, с обозначениями пробелов (в программе), где они необходимы.

Требования к программе

- Программа должна содержать комментарий с указанием названия работы, номера варианта, фамилии студента и номера группы
- Аргументы задавать операторами присваивания значений.
- При программировании выражений, части выражений встречающиеся два и более раз вычислять один раз с запоминанием в промежуточных (рабочих) переменных.
- Вывод результата выполнять на экран по формату :12
- Отладить программу, чтобы результат совпадал по всем цифрам с приведенным ответом.

Содержание программы

- Заголовок программы с комментарием;
- Описание переменных;
- Задание значений аргументам;
- Вычисление (если необходимо) промежуточных значений;
- Расчет результата;
- Вывод результата на экран.

Общие пояснения

Данная работа, как и все последующие, предполагает написание программы на языке Паскаль. Поскольку это первая работа по составлению программы, ниже приводится общая структура и правила написания программ на языке Паскаль.

Текст должен быть написан латинскими символами (прописные и строчные символы не различаются) в файле с именем, удовлетворяющем правилам файловой системы DOS и с расширением .PAS. Рекомендуется использовать имена, состоящие не более чем из 6 символов, за которыми следует номер лабораторной работы.

Операторы могут начинаться с любой позиции строки, переноситься на последующие строки (переход на другую строку разрешен в любом месте, где можно вставить пробел) и записываться по несколько операторов в одной строке (что, однако, не рекомендуется). Операторы разделяются между собой символом ";".

Комментарии представляют собой любой текст (в том числе – кириллицей), заключенный в фигурные скобки. Он может занимать несколько строк и может вставляться внутрь любого оператора языка. Внутри комментария нельзя иметь фигурные скобки (вложенный комментарий).

Операторы не рекомендуется записывать дальше 75-ой колонки для обеспечения возможности распечатки текста программы на стандартных листах бумаги (или 60-ой колонок – для распечатки на странице из тетради).

Программа может начинаться с необязательного оператора

PROGRAM <имя программы>;

за которым должен идти блок описаний, состоящий из одного или нескольких разделов. Затем идет выполняемый блок, заканчивающийся символом ".". Как правило, блок описаний содержит раздел описания переменных, начинающийся с ключевого слова **VAR**. Остальные разделы могут отсутствовать. Выполняемый блок должен быть заключен в операторные скобки **BEGIN ... END**, причем рекомендуется любую соответствующую пару скобок записывать начиная с одной и той же колонки. Открывающую скобку **Begin** лучше всегда начинать с новой строки.

Пример простейшей программы будет приведен перед таблицей с данными вариантов заданий.

Уточним задачи лабораторной работы.

1. Программа линейной структуры – это программа, в которой каждое действие выполняется последовательно один и только один раз, т.е. в алгоритме присутствуют только структуры следования. Программы такого типа в практике встречаются редко – только для расчетов по каким-либо формулам. При этом в программе должны встречаться блоки ввода исходных данных, блоки вычислений выражений и блоки вывода результатов расчетов.

2. В данной работе, в целях упрощения программы, предлагается задать исходные данные с помощью оператора присваивания. Так как расчеты по формуле выполняются также с помощью операторов присваивания, в данной работе в выполняемом блоке будут только операторы присваивания и один оператор вывода результата на экран.

Запись оператора присваивания во всех случаях выполняется в виде:
<какой переменной> <присваиваем> <значение выражения>;.

Следует помнить, что присваивание выполняется справа налево. Примеры:

```
Summa := 0;      {Обнуление переменной Summa}
A := B;      {Значение переменной B копируется в ячейку
               (переменную) A}
I := I+1;                    {Увеличение значения
                              счетчика I на единицу}
Gip:=Sqrt(Sqr(X)+Sqr(Y)); {расчет гипотенузы по
                             величинам катетов}
```

Оператор вывода на экран в простейшем случае выглядит так:

```
Writeln(<список объектов вывода>);
```

или

```
Write(<список объектов вывода>);
```

Отличие первого варианта от второго в том, что после вывода первым оператором, курсор переводится на новую строку, и следующий вывод будет выполняться в другой строке экрана.

Список объектов вывода представляет собой перечень имен переменных, строковых констант и выражений, разделенных запятыми. За каждым элементом списка может следовать формат вывода в виде одного или двух целых чисел, отделенных от элемента двоеточием. Первое число указывает, сколько позиций выделяется для выводимого значения. Если при этом значение содержит меньше символов, оно дополняется слева пробелами, если значение не помещается в отведенное место, то предлагаемый формат вывода игнорируется.

Второе число используется только при выводе вещественных чисел и указывает, сколько дробных цифр выводить после десятичной точки. При

этом число выводится в форме с фиксированной точкой (без десятичного порядка). Если второго числа в формате нет, вещественное значение выводится в экспоненциальной форме.

Примеры операторов вывода:

```
Writeln('Сколько будет чисел?'); {запрос перед вводом количества чисел}
Write(X, Sin(X)+1.5);            {вывод значения переменной X и значения зависящего от него выражения}
Writeln('Минимальное - ',K,' по порядку число');
Writeln(A,A:5,A:12,A:8:2,A:10:4);   {при A=12.345, будет выведена следующая строка: }
```

```
 1.23450000000012E+01 1.2E+01 1.23E+01   12.35   12.3450
```

Видно, что при выводе вещественные числа отделяются друг от друга пробелом (или знаком "-") и округляются, если не помещаются в отводимое для них поле.

Следует помнить, что целые числа при выводе без формата пишутся подряд, без пробелов, например, если K=12, L=34, а M=-5:

```
Writeln(K,L,M);      { получим результат в виде: }
1234-5
```

3. Описание переменных вещественного типа производится с помощью стандартного описателя **real**. Следует помнить, что во всех задачах, если не оговорено особо, любые переменные, кроме счетчиков (переменных, хранящих порядковые номера) и граничных значений для счетчиков (например, размеров массивов, количество обрабатываемых чисел), должны описываться как вещественные.

Примеры операторов описаний:

```
VAR
  A,B,C :real;
  X1,X2 :real;
VAR R,D:  real;    {раздел описаний переменных может
                    встречаться несколько раз}
```

При записи выражений на языке Паскаль нужно помнить, что написанное выражение будет выполняться слева направо, если позволяет приоритет соседних операций и отсутствуют скобки. Знаки операций для числовых выражений и их приоритеты приведены в табл. 1.

Таблица 1. Арифметические операции Турбо-Паскаля

№	Название операции	Знак	Тип		Приоритет
			Операндов	Результата	
1	Умножение	*	Числовые	Как операнды	2
2	Деление	/	Числовые	Вещественный	2
3	Целочисленное деление	div	Целочисленные	Целочисленные	2
4	Остаток целочисленного деления	mod	Целочисленные	Целочисленные	2
5	Сложение	+	Числовые	Как операнды	3
6	Вычитание	-	Числовые	Как операнды	3

Обращение к функциям имеет более высокий приоритет (1-й), а скобки определяются как имеющие наивысший приоритет (0-й). Если в выражении соседние операции имеют разный приоритет, сначала выполняется операция с более высоким приоритетом, например:

Таблица 2. Запись математических выражений на Паскале.

Вид математического выражения	Запись на Паскале	Порядок вычислений
$3{,}5\,10^{-3} - 2A$	3.5e-3 - 2.0*A	*, -
$\frac{A + B}{C - D}$	(A+B)/(C-D)	+, -, /
$\frac{A \cdot B}{C \cdot D}$	A*B/C/D	*, /, /
$\sin X^2 + \sin^2 X$	sin(X*X)+sqr(sin(X))	*, sin, sin, sqr, +

В программе на Паскале можно пользоваться стандартной константой, соответствующей числу ¶ (3.1415925...). Ее обозначение в программе – Pi, и при ее использовании нельзя описывать и применять другую переменную с таким же именем.

При работе в Турбо-Паскале (5, 6 или 7 версии), можно пользоваться стандартными математическими функциями, имена которых приведены в табл. 3.

Таблица 3. Математические функции в Турбо-Паскале

Назначение функции	Имя функции	Тип	
		Аргументов	Результата
Абсолютное значение аргумента (модуль)	abs(X)	числовой	как у аргумента
Арктангенс аргумента (в радианах)	arctan(X)	Вещественный	Вещественный
Косинус (аргумент в радианах)	cos(X)	Вещественный	Вещественный
Экспонента X (e в степени X)	exp(X)	Вещественный	Вещественный

Назначение функции	Имя функции	Тип	
		Аргументов	Результата
Дробная часть вещественного аргумента	frac(X)	Вещественный	Вещественный
Целая часть вещественного аргумента	int(X)	Вещественный	Вещественный
Натуральный логарифм веществен.аргумента	ln(X)	Вещественный	Вещественный
Синус (аргумент в радианах)	sin(X)	Вещественный	Вещественный
Квадрат аргумента	sqr(X)	Числовой	Как у аргумента
Квадратный корень веществ. аргумента	sqrt(X)	Вещественный	Вещественный

Для применения других математических функций необходимо выражать их через приведенные в табл. 2, учитывая, что:

$$\arcsin(X) = \arctan\frac{X}{\sqrt{1 - X \cdot X}} \qquad \sqrt[3]{X} = X^{1/3} = \exp(1.0/3.0 \cdot \ln(X))$$

$\log10(X) = \ln(X)/Ln(10.0)$, $X^Y = \exp(Y*\ln(X))$, и так далее.

В качестве аргумента может выступать константа, имя переменной или выражение. Во всех случаях аргумент должен быть заключен в круглые скобки.

Разбор контрольного варианта

Титульный лист

Российский Государственный Гидрометеорологический университет
Кафедра Морских информационных технологий
Дисциплина "Информатика"

Лабораторная работа N 1
Алгоритмы линейной структуры

Вариант N 31

Выполнила ст. гр. Я-007
А.Я.Умненькая

Санкт-Петербург
2002

Рисунок 1. Форма титульного листа отчета

Задание

С помощью операторов присваивания задать значения всем аргументам, входящим в выражение, вычислить выражение и, присвоив полученное значение переменной *X*, вывести результат на экран

$$\sqrt{C^{D/3} - 0.5 \cdot A^{3/2} + \exp\left(A^{3/2} \cdot \frac{C+D}{2A}\right)}$$

$$A=10^{-2};\ C=10^{2};\ D=-2.5;$$

Кроме имен переменных, входящих в состав выражения, придется использовать имена встроенных функций: квадратного корня, экспоненты и натурального логарифма (для возведения в степень).

Таблица 4 Таблица идентификаторов

Имя	Тип	Р-р, б	Назначение
X	Вещественное.	6	Результат (выражения)
A	Вещественное	6	Аргумент
C	Вещественное	6	Аргумент
D	Вещественное	6	Аргумент
R	Вещественное	6	Рабочая переменная
sqrt	Вещ.функция	–	Вычисление квадратного корня
exp	Вещ.функция	–	Вычисление экспоненты
ln	Вещ.функция	–	Вычисление натурального логарифма

Блок-схема алгоритма

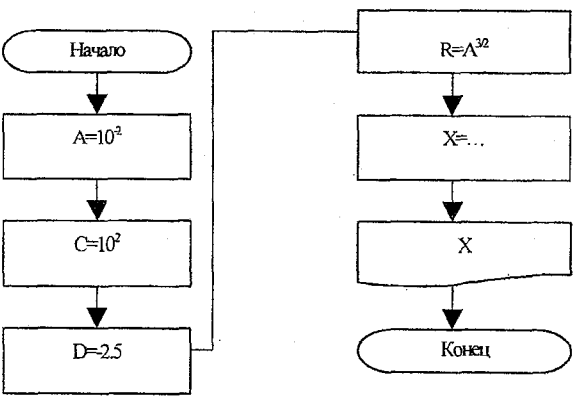


Рисунок 2 Алгоритм програмы 31 варианта

Текст программы

```
PROGRAM Lab_1;
{
   Лабораторная работа N 1
      Вариант N 31
   А.Я.Умненькая, ст. гр. Я-007
}
VAR
    A,C,D,X,R    :real;
BEGIN
   A:=1e-2;
   C:=1e2;
   D:=-2.5;
   R:=Exp(1.5*Ln(A));
   X:=Sqrt( Exp(D/3.0*Ln(C)) - 0.5*R +
            Exp(R*(C+D)/2.0/A) );
   Writeln(' X= ',X:12);
END.
```

Результат:

```
 X= 1.14453E+01
```

Варианты заданий

Таблица 5. Исходные данные к лабораторной работе №1

№ вар.	Программируемая формула	A	B	C	D	Результат
1	$\sqrt{\frac{A}{2\pi\left(B-\sqrt{B^2-C^2}\right)}}+\sin D$	10^5	5	2	2.5	1.95862E+2
2	$\sqrt{\frac{A}{(2D*\ln\frac{C}{B})}}-\frac{3.5}{C}+\ln\lvert D\rvert$	10^4	10	0.1	-3	-1.48774E+1
3	$\sqrt{\frac{A}{\left(2\pi D\left(\ln\frac{88D}{B}-1.75\right)\right)}}-3.5$	10^4	10	0.2	3	1.79615E+1
4	$\sqrt{A\cdot(3D+9B+10C)\Big/(25\pi D^2)}$	10^{-2}	-1.5	4.1	-3	1.61778E-2
5	$\sqrt{\frac{\ln A\cdot(B+C)}{4D\cdot A(B-C)}}-\exp\left(\frac{\sin B}{\cos C}\right)$	10^1	-1.7	3.9	-3	-3.83304E+0
6	$\left(\frac{A}{B(C+D)\ln\left(A\frac{C+D}{C-D}\right)}\right)^{3/5}$	10^3	3.5	4.1	-3	1.06442E+1
7	$\sqrt{A\cdot B\frac{\left(1+C\cdot\exp\left(D\Big/(A\cdot B)\right)\right)}{4\pi D}}$	10^1	-0.5	1.1	-1	9.65643E-1
8	$1.25\left(\frac{5.98A}{\exp\frac{B(C+1.41)^{0.5}}{87}}-D\right)$	10^2	-20.5	5.1	-1.5	1.36556E+3
9	$0.25\left(\frac{4D+A}{\exp\frac{B\sqrt{C}}{60}}-2.1D\right)$	10^{-1}	2.5	5.1	-1.5	-5.55037E-1
10	$0.25\left(\frac{\sin A}{5.2\,\text{le}{-6}\cdot\exp B}-4\sqrt{C\ln C}\right)-\pi D^2$	10^{-1}	1.2	5.1	2.05	1.42678E+3

№ вар.	Программируемая формула	A	B	C	D	Результат
11	$6.28-\frac{\sin(\ln A)+\cos(\lg B)}{34.2*10^{-3}\exp\left(\sqrt[4]{C}\right)}$	10^3	12	7.21	-	2.79759E-1
12	$\sqrt{\frac{10A+2\pi\cdot D}{1.1+\sqrt{\frac{13B}{0.5+\sqrt{\cos C}}}}}$	10	1.3	0.1	-.05	4.66048E+0
13	$7.7\cdot10^{-5}\cdot\sqrt{5.5^A-\frac{2\sin(A+B)}{1-\cos(\ln C)}+\frac{\pi}{D}}$	10^{-2}	1.39	3.1	0.55	1.39860E-4
14	$12A+\sqrt{7.41\left(B+\sin\left(\frac{C}{4}\right)\right)}-0.803\left(B\cdot\cos\frac{D}{3}+\sqrt{A}\right)$	10^{-3}	21.39	23.1	-0.12	-4.73017E+0
15	$\frac{A^B+B^A\ln C-C\cdot\lg\sqrt{A}}{2B+D}$	10^{-1}	2.1	0.1	-3.12	-2.24257E+0
16	$\frac{4\pi}{3}A^3-\frac{2.1B\cdot10^{C+1}}{C+1-D\cdot\exp A}+\sqrt{C+1}$	10^{-3}	-2.1	1.1	-3.12	1.07743E+2
17	$\sqrt{B^{1/3}+2.4A^{3/13}}+\sin\left(A^{3/13}\cdot\frac{C-D}{2\pi}\right)$	10^4	122.2	1.1	-3.12	4.39587E+0
18	$\frac{A}{B}\sqrt{\frac{(C-1)^2}{5.4B}+\frac{0.015(C-1)}{5.4A}}-1+C$	10^3	33.3	2.1	-	3.15920E+1
19	$\sqrt{\frac{A\ln D}{\sqrt{2-\pi+\ln D\cdot\left(\cos\left(\ln D-31*10^{-3}\right)\right)}}}$	-10^3	-	-	10	2.96095E+1
20	$\sqrt{A\cdot(3\sin D-9\cos\lvert B\rvert+10\operatorname{tg}C)}\Big/25\pi\cdot D^2$	-10^4	0.2	-0.5	3	5.26688E-1
21	$\sqrt{\frac{A\left(\left(B+\cos C\right)\right)-0.3C}{4D\cdot B\cdot\exp\left(B+\cos C\right)}}$	10^4	7.7	-0.9	0.77	9.38646E-1
22	$\left(\frac{\pi A(C-D)}{B(C+D)\cdot\ln\left(B\cdot\frac{C+D}{C-D}\right)}\right)^{3/8}\cdot\sin(C-D)$	10^3	-0.88	0.9	1.77	-1.08136E+1
23	$10^{-7}\left(\frac{5.98\cdot\exp(C+1.4)}{0.5+B}-\ln(C+1.4)\right)\cdot\cos\frac{A(C+14)}{1.81}$	10^{-1}	-0.33	2.2	-	1.28586E-4
24	$1.5\left(\frac{A/3}{\sin\frac{B\sqrt{\lvert C\rvert}}{A/3}}-\frac{3D}{A}\right)$	10^2	-0.33	-3.3	10	-2.78081E+3

№ вар.	Программируемая формула	A	B	C	D	Результат
25	$10^6 \frac{tgA}{3.1\cdot 10^6} - \sqrt[3]{\lvert C \ln \lvert C \rvert \rvert} - \pi \cdot B \cdot D^3$	10^{-1}	-0.83	-4.4	1.4	5.31933E+0
26	$1.2\cdot 10^{-3} + \frac{\cos\left(\ln\left(A/B\right)\right) + \cos\left(\lg\left(A/B\right)\right)}{4.2\cdot 10^{-3} \exp\left(\sqrt[4]{\lvert \ln\left(A/B\right) \rvert}\right)}$	10^3	5	-	-	-1.40486E+1
27	$-11\cdot 10^{-5} \sqrt{5B^A + \frac{2\sin(A+B)}{1-\cos(\ln C)} - \frac{\pi}{(A+B)B^A}}$	2.5	10	0.5	-	-4.37319E-3
28	$2\cdot 10^4 + \sqrt{4.7\lvert D + \sin\frac{C}{2} \rvert + 0.803 B \cdot \exp\frac{D}{3}}$	1.9	10^3	-2.1	13.5	4.27833E+2
29	$\frac{A^5 + B^5 \ln 5 - C \cdot \lg\sqrt{5B}}{5B + D/500}$	1.09	10^{-2}	-2.4	10^3	-1.10303E-2
30	$\frac{2\sqrt{B+1}}{3\cdot \lg B} \cdot \frac{\pi}{C} \cdot A^3 - \frac{10^{B+1}}{\exp(B+1)} + \sqrt{B+1}$	10.3	0.2	-10^4	-	-3.31949E+0
31	$\sqrt{C^{D/3} - 0.5A^{3/2} + \exp\left(A^{3/2} \cdot \frac{C+D}{2A}\right)}$	10^{-2}	-	10^2	-2.5	1.14453E+1

3802947

Лабораторная работа № 2
Программирование алгоритмов с ветвлениями

Задачи лабораторной работы

Вопросы, изучаемые в работе

- Построение простейшей программы с ветвлениями.
- Изучение условных операторов.
- Использование именованных констант.
- Использование операторов ввода для исходных данных.

Задание (общее ко всем вариантам).

Написать программу вычисления и вывода на экран значения функции F по значениям одного или нескольких аргументов, величины которых вводятся с клавиатуры операторами ввода. Результат вывести на экран. Проверить программу, задавая значения аргументов, указанные в задании, и сверяя результаты с приведенными ответами.

Оформить отчет по лабораторной работе по образцу первой работы, но текст программы распечатать из оболочки Турбо-Паскаля.

Требования к программе

- Программа должна содержать комментарий по форме, указанной в работе № 1.
- Константа, встречающаяся в задании два или более раз, должна быть использована в программе в форме именованной константы.
- Значения аргументов вводятся с клавиатуры, перед вводом должен стоять оператор запроса аргументов.
- Проверку программы выполнить для всех ветвей алгоритма.
- При выводе результата, одновременно выводить значения аргументов.
- Вывод результата выполнять по формату :8:4.

Общие пояснения

1.Алгоритмы с ветвлениями подразумевают, что в них существует больше одного пути, по которому можно пройти от начала к концу. Наличие параллельных ветвей алгоритма осложняет тестирование программ, так как требуется задать несколько вариантов исходных данных, чтобы отработали все ветви алгоритма.

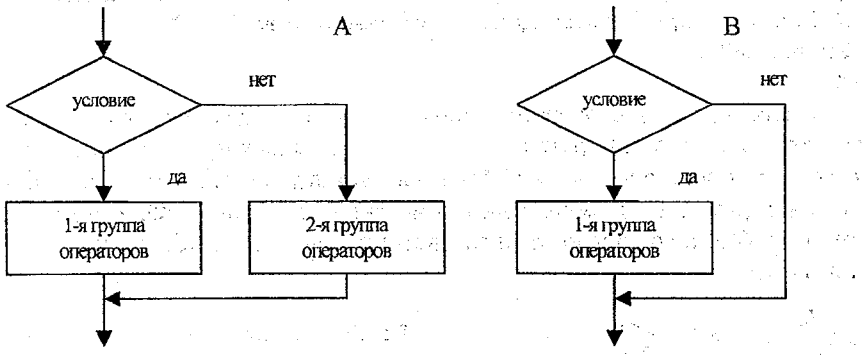


Рисунок 3. Структуры А – полного и В – неполного ветвлений

Ветвящиеся алгоритмы могут быть построены как из стандартных структур ветвления (А), так и из неполных (В).

В данной работе (как и вообще при программировании на языке Паскаль) предпочтительней использовать конструкцию типа (А), так как она отвечает требованиям структурного программирования.

2. Реализация структур ветвления на Паскале осуществляется с помощью условных операторов "if".

Таблица 6. Запись "if" операторов на Паскале

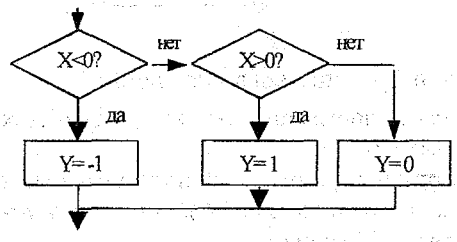
Форма (А) запишется:	Форма (В) с операторами по "да":

`if <услов.истинно> then`
`begin`
`  <операторы ветви 1>`
`end`
`else`
`begin`
`  <операторы ветви 2>`
`end;`
	`if <условие истинно> then`
	`begin`
	`  <операторы ветви 1>`
	`end;`
	Форма (Б) с операторами по "нет" :
	`if <условие истинно> then`
	`  goto Met;`
	`<операторы ветви 1>;`
	`Met: . . .`

Примеры написания таких операторов:

```
if A>0 then Y:=sin(X) else Y:=cos(X);
if (A+B > C) and (B < 0) then   { если требуется проверка}
  Writeln('Ветвь 1')            { нескольких условий,     }
else                            { каждое отношение следует}
  Writeln('Ветвь 2');           { заключать в скобки      }
if Pr then        {здесь Pr - логическая переменная,}
begin             {если Pr равно TRUE выполнится этот блок}
  Writeln(' При таких данных решения нет');
  Pr:=FALSE;
end;
```

Если в качестве оператора одной из ветвей используется условный оператор, то можно выбирать один из трех возможных путей. Вообще, количество "if"-операторов должно быть на единицу меньше, чем возможных ветвей алгоритма. Например, если нужно задать Y=-1, при X<0, Y=0 при X=0 и Y=1 при X>0, такой алгоритм и соответствующий ему текст на Паскале будет выглядеть:




```
If X<0 then Y=-1
else
  if Y>0 then Y=1
  else  Y=0;
```

Рисунок 4. Вложенный условный оператор

Условиями, определяющими какую ветвь алгоритма выполнять, являются логические выражения, принимающие значение TRUE или FALSE. В качестве таких выражений часто используются отношения между двумя однотипными данными. Турбо-Паскаль разрешает сравнивать вещественные и целочисленные данные, строчные и символьные данные. Подробнее см. учебное пособие с описанием языка.

Таблица 7. Операции отношений

№	Операция	Знак операции
1	Равно	=
2	Не равно	<>
3	Больше	>
4	Больше или равно (не меньше)	
5	Меньше	<
6	Меньше или рано (не больше)	<=

В качестве операций отношений для упорядоченных типов данных можно использовать шесть видов операций, приведенных в таблице N 5; для неупорядоченных типов разрешены только первые две операции. У всех одинаковый приоритет, причем он ниже, чем приоритеты любых других операций (арифметических и прочих), а результат всегда имеет логическое значение.

3. В программе на Паскале можно пользоваться константами, которые имеют не только значение, но и имя. Такие константы должны быть описаны в блоке описаний, в специальном разделе описаний констант. Использование таких именованных констант позволяет "вытащить" задание значений констант в начало программы, где их можно, при необходимости, изменить в одном месте. Пример раздела описаний констант:

```
CONST
  MAXBALL = 5;
  MINBALL = 2;
  ERR = ' Ошибка в программе';
  ABSNUL = -273.16;
```

В дальнейшем можно всюду вместо числового значения `-273.16` использовать имя **ABSNUL** и т.д.

Существует ряд констант, которыми можно пользоваться без их описания. Некоторые из них приведены в таблице 6.

Таблица 8. Стандартные константы Турбо-Паскаля

Имя	Тип	Значение	Назначение
TRUE	boolean	True	"истина"
FALSE	boolean	False	"ложь"
MAXINT	integer	+32767	Максимальное целое
MAXLONGINT	longint	+2147483647	Максимальное длинное целое
PI	double	3.14159265358...	Число π

4. В процессе работы программа пользуется данными, которые берет из ячеек памяти. Каким же образом эти значения попадают в эти ячейки? Существует всего три возможности. Во-первых, значения могут быть занесены в некоторые переменные в самый начальный момент при загрузке программы в память. Такие переменные называются типизированными константами (хотя по сути их правильнее называть инициализированными переменными). Во-вторых, переменная (ячейка памяти) может получить значение при выполнении оператора присваивания. Наконец, в переменную можно ввести значение с помощью процедуры ввода данных с внешнего устройства.

Только последний способ позволяет одной и той же программе обрабатывать различные наборы исходных данных. Если в программе нет операторов ввода, она при всяком запуске будет выполнять один и тот же расчет.

Оператор ввода (а точнее, процедура ввода) может вводить данные в оперативную память или из файла или с клавиатуры. При вводе с клавиатуры процедура имеет вид:

```
Read(<список переменных>);    или
Readln(<список переменных>);
```

где список переменных представляет собой перечень имен переменных через запятую, в которые заносятся вводимые значения. Очевидно, что список значений и список имен должны соответствовать друг другу по типам и порядку следования элементов списков. Отличие в этих процедурах проявляется только при вводе данных из файла. Оно заключается в том, что при втором варианте после ввода выполняется переход на новую запись файла, даже если в текущей записи данные не кончились.

Примеры:

Read(N); {программа ждет, пока не будет набрано число на клавиатуре и не нажата клавиша <Enter>, после чего переменная с именем N получит набранное значение}

Read(A,B,C); {необходимо набрать через пробел три числа и нажать <Enter>, первое попадет в ячейку с именем A и т.д.}

Нельзя в списке имен писать константы или выражения.

Если в программе требуется выполнить ввод данных с клавиатуры, предварительно следует предусмотреть команды вывода на экран запроса, какие параметры и в каком порядке пользователь должен вводить, например:

```
Writeln('задай коэффициенты уравнения: A,B,C');
Readln(A,B,C);
```

или

```
Writeln('Сколько вариантов будем считать?');
Readln(N);
```

Разбор контрольного варианта

Задание

Написать программу вычисления и вывода на экран (по формату :8:4) значения функции по значениям аргументов A и B, величины которых вводятся с клавиатуры операторами ввода. Проверить ее работу для каждой ветви алгоритма заданием соответствующих исходных данных.

Таблица 9. Данные задания 31 варианта

Вид функции	При условии	Данные для проверки		
$F = \sin(A+B)+1/(A+B)$	$A+B > 2.13$;	3.2	0.68	-0.4154
$F = \cos A - \ln(-A-B)$	$A+B < 2.13$;	0.34	-3.58	-0.2328
$F = \exp(A+B)/2.13$	$A+B = 2.13$	2	0.13	3.9506
		A	B	F

Таблица 10. Таблица распределения памяти

Имя	Тип	Р-р,байт	Назначение
F	Веществ.	6	Результат
A	Веществ.	6	Аргумент
B	Веществ.	6	-"-
C	Веществ.константа		2.13
R	Веществ.	6	Рабочая переменная
sin	Веществ.функция		Вычисление синуса
cos	Веществ.функция		Вычисление косинуса
exp	Веществ.функция		Вычисление экспоненты
ln	Веществ.функция		Вычисление натурального логарифма
Lab_2	Имя программы		Вычисление заданной функции

Блок-схема алгоритма

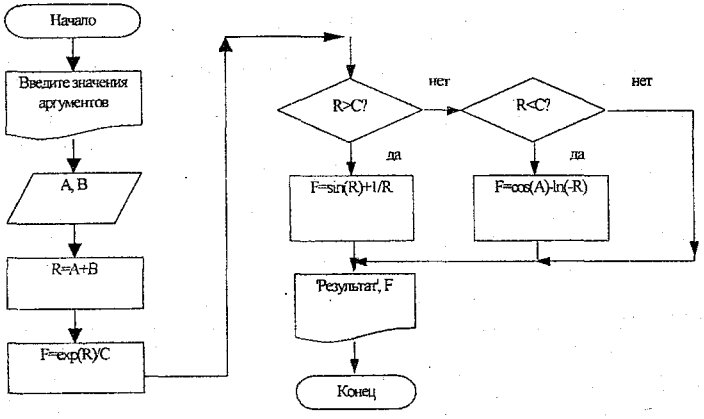


Рисунок 5. Алгоритм 31-го варианта

Текст программы

```
PROGRAM Lab_2;
{      Лабораторная работа N 2 Вариант N 31
   А.Я.Умненькая, ст. гр. Я-007  }
CONST
  C=2.13;
VAR
  A,B,F,R   :real;
BEGIN
  Writeln('Значения аргументов A и B ?');
  Readln(A,B);
  R:=A+B;
  F:= exp(R)/C;
  if R>C then
    F:= Sin(R)+1.0/R
  else
    if R<C then
      F:= cos(A)-ln(-R);
  Writeln(' Рез-т: ',F:8:4);
END.
```

Получены результаты по заданным наборам данных:

```
 Рез-т:  -0.4154
 Рез-т:   0.2328
 Рез-т:   3.9506
```

Варианты заданий

Таблица 11. Варианты заданий лабораторной работы № 2

№ вар.	Вид функции	При условии	Данные для проверки		
			A	B	Результат
1	$F = \exp(A/2) + \sqrt{-A}$	$A < -1.25;$	-4.0	0	2.1353
	$F = \lvert 1.25 - A \rvert + \lg B$	$A \geq -1.25 и B > 1;$	-1.0	100	4.2500
	$F = Sin(A \cdot B)$	*в_остальных_случаях*	4.52	0.25	-0.9983
2	$F = B\sin A + 7.04 \cdot A$	$B < 7.04;$	0.77	1.99	6.8061
	$F = A\sin B + A\cos B$	$B > 7.04;$	2.88	10	-3.9833
	$F = A\arcsin(B/10)$	*в_остальных_случаях*	1.01	7.04	0.7888

№ вар.	Вид функции	При условии	Данные для проверки		
			A	B	Результат
3	$F = tg(A-1.5)-1.5/B$ $F = \exp(B/(A-1.5))$ $F = \sin(A-1.5)+1.5B$	$\|A-1.5\|<1.5 и \|B\|>0.1$; $\|A-1.5\|>1.5 и \|B\|<0.1$; *в_остальных_случаях*	2.10 10.2 3.0	-1.2 20.1 1.0	1.9341 20.8130 2.497
4	$F = \|1.5\cdot 10^{-2} * A + B\|$ $F = \ln B - \exp(1.5e-2 * A)$ $F = B/(A\cdot \text{arctg}A)$	$\|A\|<1.5\cdot 10^{-2} и B<1$; $\|A\|<1.5\cdot 10^{-2} и B \geq 1$; *в_остальных_случаях*	0.01 0.01 1.11	0.5 2.0 -2.22	0.5001 -0.3070 -2.3881
5	$F = \sqrt{0.13456A+B-0.13456}$ $F = \exp(A)-0.13456B$ $F = \|A-B\|/(0.13456+\|A\|)$	$A>0.13456$; $\|A\|\leq 0.13456$; *в_остальных_случаях*	65.43 0.11 -2.13	1.33 10 -3.13	4.1626 -0.2293 0.4416
6	$F = 2.07\sqrt{A\cdot B}+\|A\cdot B\|$ $F = 1/2.07A\cdot B$ $F = \frac{\pi A-2.07\sin(A-B)+1}{2.07+\|A\|}$	$A\cdot B>2.07$; $A\cdot B<-2.07$; *в_остальных_случаях*	-2 -1.0 1	-2 3 1.0	-0.14 -0.1610 1.3491
7	$F = \|A\|^B - B/A$ $F = e^{-B}\cdot 0.333A$ $F = \cos(A\cdot B)-0.333A$	$\|A\|>0.333и \|B\|<2$; $\|A\|>0.333и \|B\|\geq 2$; *в_остальных_случаях*	1 2 0.11	0.55 5.45 10	0.4500 0.0211 0.4170
8	$F = tgB+\sin A$ $F = \sin(A\cdot B)-\cos(A\cdot B)$ $F = \pi\sqrt{\|B\cdot A\|+0.7788}$	$\|B\|<0.7788$; $A<0.7788, но \|B\|\geq 0.7788$; *в_остальных_случаях*	2.22 0.5 1	0.5 2.22 0.1	1.3429 0.451 0.9418
9	$F = (A+B)/(A-B-0.3456)$ $F = \sin(\pi/5-A+B)$ $F = \cos(B-A-0.3456)$	$A-B>0.3456$; $A-B<-0.3456$; *в_остальных_случаях*	1.5 0.5 0.2	0.5 1.5 0.01	3.0562 0.9983 0.9374
10	$F = \sqrt{\|462\cdot 10^{-3}-A\|}$ $F = \sin(A*B)-4.62e-1$ $F = A-\sqrt{0.462-\|B\|}$	$\|B\|\geq 462\cdot 10^{-3} или A>0$; $\|B\|<46.2*10^{-2} и A<-2$; *в_остальных_случаях*	-1 -3 -1.0	1.0 0 0.0	1.2091 -0.4620 -1.6797
11	$F = 4.777\cdot A^{B/4.777}$ $F = \sin(A-4.777\cdot B)-4.777\cdot B$ $F = B-\|A+4.777\cdot B\|$	$A>0 и B>0$; $A>0 и B\leq 0$; *в_остальных_случаях*	1.1 3.3 0	2.2 -1.2 -1	4.9914 6.1148 -5.7770

№ вар.	Вид функции	При условии	Данные для проверки		
			A	B	Результат
12	$F=\lvert A\rvert^{0.234}+B/0.234$ $F=\cos(A+0.234)*\sin(A+0.234)$ $F=\dfrac{\lvert A+B\rvert}{0.7(A+0.234)}$	$A\le -0.234 и B<0;$ $B\ge 0;$ $A>-0.234 и B<0$	-1 0.0 0	-2 1.0 -0.23	-7.547 0.2256 1.4286
13	$F=1.123\cdot \text{arctg}A+B/1.123$ $F=\exp(1.123A)/B$ $F=1.123*\sin A-\exp(-B)$	$\lvert B\rvert<1.123;$ $\lvert B\rvert\ge 1.123 и A<3;$ *в_остальных_случаях*	1 2.5 3.14	-1 2.1 2	-0.0085 7.8897 -0.1335
14	$F=A^{0.5555}-0.5555B$ $F=\sqrt{A}/B-0.5555*B$ $F=0.5555\exp A*\sin B$	$A>0 и \lvert B\rvert\le 0.5555;$ $A>0 и \lvert B\rvert<0.5555;$ $A\le 0$	1 4 -1	-0.1 2 0	1.0556 -0.1110 0.0000
15	$F=\text{arctg}(0.3456B)-A\cdot B$ $F=\lg\lvert A\cdot B\rvert+\sqrt{A\cdot B}$ $F=\sin B-\cos(0.3456A\cdot B)$	$\lvert A\cdot B\rvert<0.3456;$ $\lvert A\cdot B\rvert\ge 0.3456 и A\cdot B>0;$ *в_остальных_случаях*	1 2.5 1.5	-0.2 1.5 -1.5	0.131 2.5105 -1.7101
16	$F=\lg A-\sin B+0.444$ $F=\exp(0.444A)+\cos B$ $F=0.444\ln\lvert B\rvert$	$A>0.444;$ $A<-0.444;$ *в_остальных_случаях*	1 -1 0	0.5 2.3 9.5	-0.0354 -0.0248 0.9996
17	$F=A-(A-B)^{0.1357}$ $F=0.1357\cos^2(A-B)$ $F=\sin^2(A-B-0.1357)$	$A>B;$ $A\le B, но B<0;$ *в_остальных_случаях*	3.5 -3.5 1	2.5 -2.5 2	2.5 0,0396 0.8223
18	$F=\sqrt{A+B}-1.4973B$ $F=(A^{1.4973}-B)(A+B)$ $F=\lvert A+B\rvert/(1.4973-A)$	$A+B>0;$ $A+B\le 0 и A>0;$ *в_остальных_случаях*	2 1 -1.5	-1 -2 0.5	2.4973 -3.0000 0.3336
19	$F=2.8765*\ln(A\cdot B)$ $F=\cos(2.8765A)+B$ $F=2.8765/B$	$A+B>0;$ $A+B\le 0 и B>-2.8765;$ *в_остальных_случаях*	1.5 -1.5 0	2 -2 -5.8	3.1602 -2.3872 -0.4959
20	$F=A^{3.1212}+3.1212B^{0.777}$ $F=0.777/3.1212*\sin(A*B)$ $F=0.777+A\cdot B\cdot 3.1212$	$A>0.777 и B>0.777;$ $A<-0.777, или B<-0.777;$ *в_остальных_случаях*	1 -1 0.5	1.0 2 2	4.1212 -0.2264 3.8982
21	$F=\lg(A-B)$ $F=\text{arctg}\lvert A-B\rvert+0.001$ $F=A+0.001(A-B)$	$A-10^{-3}>B;$ $A<B и A>0;$ *в_остальных_случаях*	1.8 0.5 -0.2	0.8 1.7 100	-0.0000 0.8771 -0.3002

№ вар.	Вид функции	При условии	Данные для проверки		
			A	B	Результат
22	$F = 4.6241\sqrt{A+B}$	$A+B > 4.6241$;	200	-89.9	48.52
	$F = 1/(4.6241 - A - B)$	$A+B < -4.6241$;	-12.3	4.5	0.0805
	$F = (A+B)^5$	*в остальных случаях*	2.5	-0.5	32.0000
23	$F = A^{0.333} - B^{0.333}$	$A > 0 и B > 0$;	10	20	-0.5589
	$F = \ln(A+B)$	$A \le 0$, *или* $B \le 0$, *но* $A > -B$;	-10	15	1.6094
	$F = 0.333(A+B)^8$	*в остальных случаях*	3.5	-5.5	85.248
24	$F = \sin^4(1.121 \cdot A \cdot B)$	$A < 0 и B < 0$;	-1.88	-0.66	0.9370
	$F = \cos(1/A/B)$	$A > 0 и B > 0$;	1.25	0.13	0.9916
	$F = 1.121 \cdot A \cdot B$	*в остальных случаях*	5.5	-0.02	-0.1233
25	$F = \ln\frac{A}{B} - \frac{A}{8.338B}$	$\frac{A}{B} > 8.338$ *и* $\lvert B \rvert > 0.1$;	11.1	0.87	1.016
	$F = \exp\frac{A}{B} + 8.338B$	$\frac{A}{B} \le 8.338$ *и* $\lvert B \rvert > 0.1$;	3.9	1.55	25.3045
	$F = 8.338B$	*в остальных случаях*	0	0.06	0.5003
26	$F = \ln(\sin A) + \sqrt{\sin A} + 0.1149B$	$\sin A > 0.1149$;	1	3.55	1.1526
	$F = \mathrm{ctg}A + \cos B$	$\sin A \le 0.1149$ *и* $\lvert B \rvert < \pi/2$;	-0.5	3.1	-0.0551
	$F = 0.1149 \sin A$	$\sin A \le 0.1149$ *и* $\lvert B \rvert \ge \pi/2$;	1.7	0	1.1299
27	$F = 0.666 + A + 2\ln\lvert B \rvert$	$\lvert B \rvert < 1$ *и* $\lvert B \rvert > 10^{-5}$;	-0.08	0.28	-1.9599
	$F = 0.666\sqrt{2\ln\lvert B \rvert + A}$	$\lvert B \rvert \ge 1$ *и* $A > 0$;	1.89	4.92	1.5006
	$F = \cos(\ln\lvert B \rvert / 0.5 + A) - 0.666A$	*в остальных случаях*	2.40	-2.4	1.3569
28	$F = \exp\frac{1733}{A+B} + \ln(A+B) - 1733$	$A + B > 17.33$;	10.3	13.8	-12.0952
	$F = \cos A + \cos B + \cos(A+B) - \sin\frac{1733}{A+B}$	$A + B < 17.33$;	-20.6	-0.6	-1.2233
	$F = (A+B)/1733$	*в остальных случаях*	2.48	-1.75	0.0421
29	$F = A^{0.38} + B \cdot \sin(A^{0.38}) + 0.38$	$A > 0 и B < 0$;	5.77	-1.85	0.6055
	$F = B^{0.38} + A\cos(B^{0.38}) + \ln(-A)$	$A < 0 и B > 0$;	-3.96	2.04	1.6708
	$F = 1/(0.38 + A * B)$	*в остальных случаях*	1.0	0.1	2.0833
30	$F = (A+B)^{0.71} + A^{0.71} + B$	$A > 0 и B > 0$;	2.77	0.88	5.4488
	$F = \exp(-A-B) - A - B$	$A < 0$, *или* $B < 0$, *но* $A > B$;	0.84	-3.58	18.227
	$F = -\frac{A+B}{0.71 + \lvert -A-B \rvert}$	*в остальных случаях*	-1.39	-0.42	7.9204
31	$F = \sin(A+B) + 1/(A+B)$	$A + B > 2.13$;	3.2	0.68	-0.4154
	$F = \cos A - \ln(-A-B)$	$A + B < 2.13$;	0.34	-3.58	-0.2328
	$F = \exp(A+B)/2.13$	$A + B = 2.13$	2	0.13	3.9506

Лабораторная работа № 3
Работа с последовательностями чисел

Задачи лабораторной работы

Вопросы, изучаемые в работе

- Построение программы циклической структуры с использованием операторов арифметических и итеративных циклов, реализация последних с помощью условных переходов.
- Освоение типовых алгоритмов: вычисления суммы, произведения, поиск максимума, минимума во вводимой последовательности и их порядковых номеров.
- Использование операторов описания переменных для данных различных типов.
- Использование в программе контроля за входными данными и результатами расчета.

Задание (общее ко всем вариантам).

Составить программу обработки последовательно вводимых с клавиатуры чисел (пока не будет обработано заданное количество чисел, или не сработает условие окончания ввода). Полученный результат обработки вывести на экран.

Оформить отчет по работе аналогично оформлению отчета по работе № 2. Текст программы должен быть распечатан, результаты – переписаны от руки.

Требования к программе.

- Программа должна обрабатывать данные указанного типа в количестве до 100 чисел, если в программе не указано иное ограничение. Конкретное количество чисел вводить в программу с клавиатуры. Массивы не использовать.
- Все значения, на которые по смыслу накладываются ограничения, должны при вводе проверяться.
- Если в результате обработки данных результат не получен, при выводе программа должна сообщать об этом в понятной форме.
- При выводе на экран использовать длину выводимой строки не более 76 символов.
- Остальные требования – как и в предыдущих лабораторных работах.

Общие пояснения

1. Программы циклической структуры используются, когда необходимо несколько раз выполнить однотипные действия с различными данными. Если количество повторений тела цикла известно перед началом цикла, он называется арифметическим, если нет – итеративным. Для организации арифметического цикла в блок-схеме алгоритма используется блок "модификатор", а в программе – соответствующий ему оператор "`for...`".

Итеративный цикл строится с использованием блока "решение", в котором один из альтернативных путей представляет выход из тела цикла. В программе такой цикл может выполняться с помощью специальных операторов итеративных циклов или оператора условного перехода. В данной работе будет использоваться только "`if...`" оператор.

Рассмотрим сначала *арифметический цикл*. Оператор состоит из заголовка и тела цикла. Заголовок имеет вид:

```
for <имя параметра цикла> := <начальное значение> to
<конечное значение> do
```

Далее идет тело цикла: простой или составной оператор (до символа ";"). Если оператор составной, он заключается в операторные скобки `begin ... end;`

В качестве параметра цикла можно использовать любую целочисленную переменную, в которой в это время не хранится нужное в дальнейшем значение. Эта переменная изменяется в цикле автоматически. Параметр цикла – это переменная, обычно играющая роль не только счетчика количества выполненных повторений цикла, но одновременно служащая порядковым номером обрабатываемого числа или элемента массива.

Начальное и конечное значения параметра цикла могут быть выражениями, но они вычисляются только один раз – при первом входе в цикл. В теле цикла они не должны меняться. При каждом возврате к заголовку в цикле, счетчик автоматически увеличивается на единицу.

Тело цикла выполняется, пока счетчик не станет больше конечного значения (если счетчик равен конечному значению – цикл выполняется). Если требуется организовать цикл, в котором параметр уменьшается на единицу с каждым возвратом, то используют ключевое слово "`downto`", вместо "`to`".

Примеры написания оператора:

```
for i:=1 to 25 do  write('*'); {вывод строки из 25 *}
for k:=L+1 to N-1 do      { значения L и N должны    }
begin                     { быть определены до цикла }
  . . .
end;
for i:=N downto 2 do S:=S+i; {Сумма целых чисел 2..N}
```

Если нужно менять счетчик с другим шагом (не один), используют второй счетчик, изменяющийся в теле цикла по рекуррентной формуле, например `j:=j+3;` или вычисляемый через первый счетчик, например: `j:= 2+(i-1)*3;`

Итеративные циклы для завершения требуют проверки такого условия, результат которого может измениться в процессе выполнения тела цикла. По выполнении этого условия осуществляется переход на оператор, следующий за телом цикла. Возврат на начало тела цикла в этом случае выполняется оператором безусловного перехода. В программе при такой организации цикла требуется иметь две метки:

```
Start:
 . . .
 if A=Priznak then goto Finish;
 . . .
 goto Start;
Finish: . . .
```

Часто проверяют условие продолжения цикла, и по его выполнению возвращаются на начало тела цикла, иначе цикл заканчивается. В этом случае достаточно обычно одной метки:

```
Start:
  . . .
  if A<>Priznak then goto Start;
  . . .
```

Тело итеративного цикла при таких способах построения в скобки `begin ... end` заключать не обязательно.

2. Циклические процессы почти всегда требуют некоторых подготовительных действий, выполняемых до начала цикла. Это связано с использованием в теле цикла так называемых "рекуррентных соотношений", в которых некоторая переменная вычисляется с использованием своего старого значения, например:

```
N:=N-1; P:=-2*P/i; S:=S+A и т.д.
```

Все такие переменные перед циклом должны получить определенные значения, чтобы правильно вычисляться внутри цикла. Например, при вычислении суммы последовательно вводимых слагаемых, ячейка (переменная) для суммы должна обнуляться; переменная для накопления произведения делается равной единице (или первому сомножителю).

Поиск максимумов (минимумов) или их порядковых номеров в рядах значений также выполняется по рекуррентным зависимостям (если новая величина больше максимума, сделаем максимум равным...) – сперва используется старое значение максимума, чтобы создать новое значение.

При поисках максимума можно использовать два варианта начального задания: или в качестве максимума берется первое рассматриваемое значение (и цикл начинает выполняться начиная со второго числа), или в качестве

начального задается фиктивное значение – гарантированно меньшее, чем любое число сравниваемого ряда (т.е. теоретически минимальное возможное число). При этом циклическая обработка одинаково выполняется для всех чисел, начиная с первого.

3. Если в программе требуется обрабатывать нечисловые данные, для них можно заводить переменные нечисловых типов, которые также необходимо включить в блок описаний. Для символьных переменных используется описатель char, а для логических – описатель boolean. Например:

```
VAR
  i,j :integer;
  A,S :real;
  C,Sim :char;
  Q,Priz:boolean;
. . .
```

Символьные переменные можно сравнивать с символьными константами и между собой с помощью всех операций отношений (=,<,>, и т.д.), например:

```
if Sim > 'A' then ... или
if C = Sim then ...
```

Логические переменные можно использовать в условных операторах, причем не нужно записывать `if Q=TRUE then...`, а просто `if Q then...`, так как само значение Q может быть `TRUE` или `FALSE`.

Наконец несколько замечаний по контролю за вводимыми величинами. Поскольку в программе предполагается ввод исходных данных пользователем программы путем набора значений на клавиатуре, в программе должны быть предусмотрены выводы на экран запросов на ввод того или иного параметра, нужного программе. Эти запросы представляют собой операторы вывода некоторого текста, например:

```
Writeln(' Сколько чисел будет обрабатываться?');   или
Writeln(' Вводи очередной сомножитель');
```

Такие запросы должны быть перед каждым оператором ввода данных с клавиатуры. Другой момент, который следует отметить, это необходимость контроля за вводимыми величинами. Недопустимое значение может не только привести к неправильному ответу, но и просто "подвесить" программу или привести к ее аварийному снятию операционной системой. Поэтому все данные, на которые по их смыслу (или по возможностям реализованного метода решения) накладываются ограничения, должны проверяться после их ввода на попадание в допустимый диапазон.

Обычно считается, что можно найти сумму нуля и большего количества слагаемых, произведение одного или более сомножителей, выбрать наибольшее или найти среднее значение из одного или нескольких чисел, вычислить факториал нуля или большего целого числа и т.д. Иногда ограничения накладываются не только снизу, но и сверху.

Если, например, допустимый диапазон для значений переменной A ограничен числами –0.5 и 11.0, то проверку на ошибочное значение можно осуществить оператором:

```
if (A > 11.0) or (A < -0.5) then ...
```

Если такая проверка даст значение TRUE (т.е. A имеет недопустимое значение), тогда нужно выдать сообщение об этом и либо закончить выполнение программы, либо (что более разумно) вернуться к оператору запроса ввода этой величины еще раз. Так как оба эти действия следует выполнять по срабатыванию одного и того же условия, операторы, задающие эти действия следует поместить в скобки begin ... end, например:

```
if (A > 11.0) or (A < -0.5) then
begin
  Writeln(' Вы задали недопустимое значение');
  goto Vvod_A;
end; { здесь Vvod_A – метка перед оператором запроса на
ввод параметра, соответствующего переменной A }
```

Наконец, следует помнить, что не при всяких наборах исходных данных задача может иметь решение. Если по каким-либо причинам ответ выдать невозможно, следует об этом сообщить в понятной форме, а не выводить неправильное или невозможное для ответа значение. Например, если в задаче требуется делить что-то на сумму нескольких чисел с разными знаками, то сумма может случайно оказаться равной нулю, после чего деление невозможно и решения у задачи не окажется.

Тогда выдача результата может выглядеть так:

```
. . .
if Sum = 0 then
Writeln(' Нет решения, так как сумма равна 0')
else
begin
 R:=.../Sum;
 Writeln(' Отношение =', R:...);
end;
```

Разбор контрольного варианта

Задание

Таблица 12. Данные к заданию 31 варианта

№ вар.	Задание	Остановить обработку при…	Тип обрабат. данных
31	Нахождение номера последнего числа, превышающего значение 10.5 в последовательности вводимых произвольных чисел.	…вводе заказанного количества чисел	Вещественые

Решение задачи следует начинать с разбора задания и выделения объектов, упоминающихся или подразумевающихся в задании. Для каждого найденного объекта определяется его тип и придумывается имя (идентификатор) для последующего программирования. Имена, типы и назначения сводятся в таблицу идентификаторов.

Для данной задачи можно выделить: номер последнего числа, превышающего значение 10.5; количество чисел (*N*); вводимое число. Для запоминания искомого номера нужно знать порядковый номер вводимого числа. В задании оговорено, что вводимые числа – произвольные, т.е. могут быть как целыми, так и дробными, для их хранения в ЭВМ нужно иметь переменную (ячейку) вещественного типа. Текущий номер и искомый номер – величины целые и, вообще говоря, положительные. Для них можно завести или целые или беззнаковые переменные. Общее количество чисел также должно быть целым и больше нуля, иначе задача не имеет смысла. Таким образом, имеем таблицу:

Таблица 13. Идентификаторы программы 31-го варианта

Имя	Тип	Размер, байт	Назначение
N	Целый	2	Количество чисел
I	Целый	2	Текущий номер
Num	Целый	2	Номер последнего числа, которое > 10.5
A	Вещественый	6	Текущее число

Исходными данными в задаче являются, во-первых, количество чисел, а во-вторых, сами числа, последовательно вводимые в переменную A. Результатом будет порядковый номер последнего из чисел, которое превышает 10.5. Возможно, что среди вводимых чисел не найдется ни одного такого числа. В этом случае в конце необходимо выдать об этом сообщение, а не номер числа.

Контроль допустимости вводимых данных необходим только для количества чисел: оно должно быть не меньше единицы.

Алгоритм задачи состоит из трех последовательных обобщенных шагов: ввода данных, определения искомого номера и вывода результата.

В части ввода данных программа должна получить от пользователя значение количества чисел (*N*). Так как ввод данных выполняет человек с помощью клавиатуры, программа должна сообщить, что от него требуется.

Всякому вводимому с клавиатуры числу должен предшествовать запрос на дисплее: что вводить и в какой форме. Как и любое значение, вводимое пользователем с клавиатуры, количество чисел (*N*) должно контролироваться на допустимость введенной величины.

Если введено недопустимое значение, требуется выполнить два действия: сообщить об этом и вернуться на запрос нового значения N. Возврат назад возможен двумя способами: командой перехода на метку, поставленную перед оператором запроса или оператором итеративного цикла, причем цикла с "постусловием", так как один раз цикл должен выполниться обязательно.

В данном случае используем первый способ, и потому в программе появляется еще один объект – метка. Дадим ей имя `vvod`.

Кроме того, дадим имя нашей программе, например `laborator_N_3`. В результате в таблице имен добавится две строки:

Таблица 14. Окончание таблицы идентификаторов

Имя	Тип	Р-р, байт	Назначение
laborator_N_3	Имя программы	-	Поиск номера определенного числа
vvod	Метка	-	Возврат по ошибке ввода

Алгоритм

Раздел поиска номера обычно включает подготовку и цикл поиска. Так как количество чисел известно, используется арифметический цикл. В тело цикла входит получение очередного числа и, если нужно, запоминание его номера. Если встретится хотя бы одно число, удовлетворяющее условию превышения значения 10.5, номер будет найден, иначе нужно иметь признак, что такого числа не встретилось. В качестве такого признака можно использовать переменную *Num*, в которую перед циклом поиска заносится значение, невозможное для номера числа: например, –1. Если после цикла *Num* останется равным –1, следовательно, искомых чисел не встретилось.

Таким образом, раздел поиска включает присваивание начального значения переменной *Num* и цикл, тело которого содержит запрос ввода очередного числа, прием и занесение его в переменную для текущего числа `A`; проверку, что `A>10.5`, и если так, запоминание текущего номера в перемен-

ной ***Num***. Цикл поиска нужно вести до конца, так как нас интересует последнее число, удовлетворяющее условию выбора.

По окончании раздела поиска, в разделе вывода результатов, печатается найденный номер ***Num***, если он не равен –1, иначе выдается сообщение об отсутствии чисел, удовлетворяющих условию задачи. Построение алгоритма здесь приведено очень подробно, чтобы пояснить весь ход рассуждений. В последующих работах примеры построения алгоритмов будут даваться в укрупненном виде – т.е. только основные идеи. Для разработки алгоритма стоит обратиться к материалам лекции по структурному программированию.

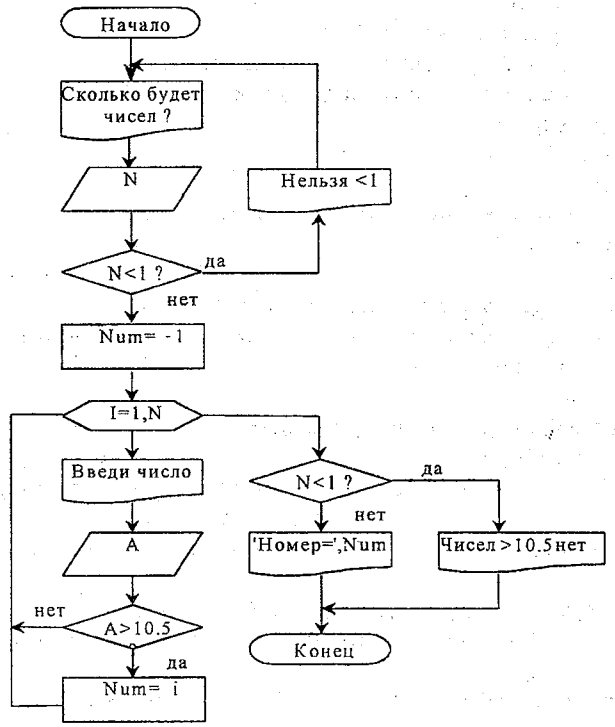


Рисунок 6. Алгоритм 31-го варианта

На основании построенного алгоритма можно написать текст программы, причем таблица идентификаторов используется для создания раздела описаний, а алгоритм – для выполняемого блока. При написании учтем требования к оформлению текста программы.

Получим следующую программу на Паскале.

Текст программы

```
PROGRAM laborator_N_3;
{  Программа Лабораторной работы N 3
       Вариант N 31.
  гр. Я-007, ст. Умненькая И.Я.  }
VAR   N,Num,i :integer;
          A    :real;
LABEL    vvod;
BEGIN  { Выполняемый блок.
        Первый раздел - ввод данных }
  vvod: writeln('Сколько будет чисел?');
  readln(N);
  if N < 1 then
  begin
    writeln('Нельзя рассматривать меньше 1 числа');
    goto vvod;
  end;
     { Основной раздел - поиск номера }
  Num:= -1; {задание признака отсутствия подходящих чисел}
  for i:=1 to N do
  begin
    writeln('Введите очередное число');
    readln(A);
    if A > 10.5 then  { если число удовлетворяет
                        условию задачи }
      Num:=i;         { запоминаем его номер }
  end;
    { Раздел вывода результатов поиска }
  if Num=-1 then writeln('Подходящих чисел не было')
  else
    writeln('Номер последнего числа,>10.5 равен ',
            Num);
END.
```

Варианты заданий

Таблица 15. Варианты заданий лабораторной работы № 3

№ вар.	Задание	Остановить обработку при…	Тип обраб. данных
1	Вычисление произведения последовательно вводимых чисел	…превышении абсолютной величины произведения 1000.0, или при вводе числа 0	Веществ.
2	Вычисление суммы только четных чисел из последовательно вводимых чисел	…вводе заказанного количества чисел	Целые
3	Нахождение номера минимального значения в последовательности вводимых чисел	…вводе признака конца – предварительно введенного пользователем числа	Целые
4	Вычисление суммы только положительных чисел из последовательно вводимых чисел	…превышении суммы 100.0	Веществ.
5	Нахождение максимального значения в последовательности вводимых чисел. При поиске пропускать числа из диапазона от 10.1 до 50.1.	…вводе заказанного количества чисел	Веществ.
6	Вычисление суммы последовательно вводимых чисел	…вводе признака конца – значения из заранее выбранного пользователем диапазона	Веществ.
7	Нахождение минимального значения среди последовательности вводимых чисел. Сам признак конца при определении минимума не учитывать.	…вводе признака конца – предварительно введенного пользователем числа из диапазона -10..5	Целые
8	Вычисление произведения последовательно вводимых чисел заданного количества	…вводе заказанного количества чисел – не меньше 5 но и не более 10	Веществ.
9	Нахождение минимального значения из чисел, кратных трем, среди последовательности вводимых чисел.	…вводе заказанного количества чисел	Целые
10	Вычисление суммы последовательно вводимых чисел	…превышении абсолютной величины суммы 100.1	Веществ.

№ вар.	Задание	Остановить обработку при…	Тип обраб. данных
11	Нахождение максимального из последовательно вводимых чисел	…вводе признака конца – предварительно выбранного пользователем числа	Целые беззнаковые
12	Вычисление произведения последовательно вводимых чисел	…вводе признака конца – числа из диапазона от -5 до 5 (включительно)	Целые
13	Нахождение номера максимального значения, кратного пяти, в последовательности вводимых чисел	…вводе признака конца – предварительно выбранного пользователем числа	Целые
14	Вычисление факториала вводимого числа. Если результат не помещается в 4 байта, программа должна сообщать об этом	Факториалом целого числа N (обозначается N!) называется произведение всех целых чисел от 1 до N. По определению, 0!=1	Длинные целые (4 байта)
15	Вычисление среднеарифметического тех из последовательно вводимых чисел, которые попадают в диапазон от 2.0 до 5.0 включительно.	…вводе заказанного количества чисел	Веществ.
16	Нахождение номера максимального отрицательного значения в последовательности произвольных вводимых чисел	…вводе признака конца – предварительно выбранного пользователем числа	Веществ.
17	Вычисление произведения только отрицательных из последовательно вводимых произвольных чисел	…вводе заказанного количества чисел	Веществ.
18	Нахождение номера наименьшего положительного числа в последовательности произвольных чисел, вводимых с клавиатуры	…вводе заказанного количества чисел	Веществ.
19	Вычисление среднеарифметического последовательно вводимых чисел	…вводе третьего отрицательного значения, которое рассматривать как признака конца, а не число.	Веществ.
20	Нахождение максимального отрицательного значения в последовательности вводимых произвольных (как положительных, так и отрицательных) чисел	…вводе заказанного количества чисел	Целые

№ вар.	Задание	Остановить обработку при...	Тип обраб. данных
21	Вычисление среднего значения только положительных элементов среди вводимых произвольных чисел	...вводе заказанного количества чисел	Веществ.
22	Нахождение разности номеров первого и последнего отрицательных чисел в последовательности водимых чисел	...вводе заказанного количества чисел	Целые
23	Вычисление среднеарифметического только отрицательных чисел среди произвольных вводимых чисел	...вводе признака конца (выбранного пользователем произвольного числа).	Веществ.
24	Нахождение разности максимального и минимального значений в последовательности вводимых чисел	...вводе заказанного количества чисел	Целые
25	Вычисление суммы только отрицательных чисел, абсолютная величина которых превышает 8.5, из последовательно вводимых произвольных чисел	...вводе заказанного количества чисел	Веществ.
26	Нахождение минимального значения среди последовательности вводимых чисел, из которых рассматриваются только четные	...вводе заказанного количества чисел	Целые
27	Вычисление отношения суммы всех положительных чисел к сумме всех отрицательных в последовательности вводимых чисел	...вводе заказанного количества чисел	Веществ.
28	Нахождение разности номеров максимального и минимального значений в последовательности вводимых чисел	...вводе заказанного количества чисел	Веществ.
29	Вычисление среднеарифметического значения только отрицательных элементов после первого 0 среди вводимых произвольных вещественных чисел	...вводе заказанного количества чисел	Веществ
30	Вычисление отношения суммы всех положительных чисел к сумме всех чисел в последовательности вводимых чисел	...вводе признака конца – числа 0	Веществ.
31	Нахождение номера последнего числа, превышающего значение 10.5 в последовательности вводимых произвольных чисел	...вводе заказанного количества чисел	Веществ.

Лабораторная работа № 4
Работа с одномерными массивами
(подсчет, поиск элементов, перестановки в массиве)

Задачи лабораторной работы

Вопросы, изучаемые в работе

- Изучение простейших составных данных – одномерных массивов.
- Освоение форматного вывода одномерных массивов разных типов.
- Создание и использование в программах входных текстовых файлов с данными.
- Вывод результатов работы в выводной текстовый файл (протокол работы).
- Дальнейшее изучение основных элементарных алгоритмов.

Задание (общее ко всем вариантам)

Составить программу обработки одномерного массива заданного типа произвольной длины. Количество используемых данных должно вводиться с клавиатуры (в пределах отведенного под массив места). Значения элементов массива вводятся или с клавиатуры, или из предварительно созданного текстового файла (в соответствии с заданием). Исходный массив и результаты его обработки выводить в выводной текстовый файл.

Оформить отчет по работе, аналогично отчету по работе № 3, но в печатном виде должны быть представлены текст программы и результаты работы.

Требования к программе

• Если специально не указано, что данные в массив вводятся с клавиатуры, перед запуском программы необходимо подготовить данные требуемого типа в отдельном текстовом файле, которому следует присвоить имя, совпадающее с именем файла Паскаль-программы, но с расширением ".DAT". Данные набивать, отделяя друг от друга пробелом, по 10 – 20 чисел в строке. Для символьных данных – все набивать в одной строке.

• Программа должна обрабатывать данные указанного типа в количестве до 50 чисел и до 250 символов, если количество не указано в задании. Конкретный размер массива при запуске программы вводить с клавиатуры.

• Все значения, на которые по смыслу накладываются ограничения, должны при вводе контролироваться.

• Вывод исходных данных и результатов производить в выходной текстовый файл. При выводе использовать длину выводимой строки не более 76 символов.

Содержание программы

- Заголовок программы с комментарием;
- Описание переменных и массива;
- Описание меток;
- Ввод исходных данных с клавиатуры;
- Открытие входного и выводного файлов;
- Заполнение массива из файла;
- Вывод на печать введенного массива в выводной файл под заголовком "Исходные данные";
- Проведение обработки массива в соответствии с заданием;
- Вывод результатов обработки в выводной файл под заголовком "Результаты расчета";
- Закрытие всех файлов.

Общие пояснения

Рассмотрим некоторые моменты, связанные с обработкой одномерных массивов. Следует помнить, что выделение памяти под массив при его описании может производиться только с помощью констант (в явном виде, или с предварительным описанием константы), например:

```
Const MIN=5;
      MAX=100;
Var   Massiv1 : array[1..110] of real;
      Massiv2 : array[MIN..MAX] of integer;
```

Кроме того, рекомендуется сначала завести свои описатели типов массивов, а затем с их помощью выделять место под конкретные переменные этих "массивных" типов, например:

```
Const  MAXI=200;
Type   IntMasMAX = array[1..MAX] of integer;  {описатель
для целочисленных массивов из MAX элементов}
       RealMass = array[1..110] of real; { описатель
       для вещественных массивов из 110 элементов }
Var Massiv1,Nmbs:IntMasMAX; {завели 2 целочисл. массива}
    Massiv2,WindV:RealMass; {Завели 2 веществ. массива}
```

Программы, использующие массивы, позволяют сначала ввести все необходимые данные, а только потом их использовать. Поэтому в таких программах следует предусматривать следующие шаги (в указанном порядке):

1. **Ввод исходных данных**. Обычно включает ввод количества элементов массива (с проверкой допустимости введенного значения) и ввод самих элементов массива в указанном количестве. Если ввод значений предусмотрен с клавиатуры, перед каждым оператором чтения должен быть запрос на ввод. Если используется ввод из файла, запросы не делаются, но перед вводом данных в массив из файла, последний следует открыть для чтения стандартными процедурами `Assign` и `Reset`, а в разделе описаний переменных для файла должна быть введена переменная с помощью описателя `text`:

```
VAR
  Finput : text; {завели файловую переменную для набора данных
текстового типа}
     . . .
BEGIN
     . . .
  Writeln(' Введи длину массива');
metka: Read(N);
  if(N<MIN) or (N>MAX) then
  Begin
    Writeln('Недопустимое количество, введите снова');
    goto metka;
  End;
  Assign(Finput,'UMNIK4.DAT');       { связали файловую
                переменную с набором данных UMNIK4.DAT }
  Reset(Finput);              { открыли файл для чтения }
  for i:=1 to N do
    Read(Finput,Massiv2[i]);
    . . .
```

Если в задании требуется вводить данные в массив, пока не встретится определенный признак, то можно поступить двумя способами. Организовать арифметический цикл со счетчиком от одного до предельно-допустимой длины массива и дополнительным выходом из цикла, если встретился признак; или использовать итеративный цикл с условием продолжения, если введенное значение – не признак, и его порядковый номер меньше длины массива.

2. **Вывод исходных данных**. Включает обычно вывод заголовка массива, возможно с указанием его длины, и затем вывод заполненных элементов массива в удобной для просмотра на экране (и при распечатке) форме. При выводе на дисплей можно ориентироваться на ширину экрана (80 позиций), при печати – на 60 (вдоль тетрадного листа), 76 (поперек тетрадного листа или при выводе на стандартный лист).

Обычно для распечатки результатов работы используют вывод в текстовый файл, который затем выводят на принтер, как и текст программы.

Чтобы вывод выполнять в файл, необходимо предусмотреть в программе следующие шаги:

• в разделе описаний переменных завести файловую переменную типа `text` для вывода;

• в выполняемом блоке связать файловую переменную с набором данных и открыть для вывода (используя стандартные процедуры `Assign` и `ReWrite`);

• в операторах `Write` и `WriteLn` в качестве первого (или единственного) параметра указывать имя файловой переменной;

• перед концом программы закрыть выводной файл стандартной процедурой `Close`.

Оформление вывода следует выполнять с использованием формата, размер которого определяется максимальными размерами выводимых значений.

Например, если числа целые и находятся в диапазоне от –999 до +999, формат должен быть не менее :5 (с учетом разделяющих пробелов). Если диапазон целых чисел неизвестен, следует рассчитывать на максимум. Для самых длинных целых чисел (–32768) он составит :7 и выводить удобно по 10 значений в строке.

Для вещественных чисел, если использовать экспоненциальную форму записи, достаточно оставлять три значащие цифры, что с учетом знака, точки и порядка числа составит :11, например, _–0.836E-02, и выводить имеет смысл по пять чисел в строке.

Конечно, удобнее числа выводить в форме с фиксированной точкой (что можно делать, если порядки чисел известны и они не сильно отличаются от нулевого). Например, по формату :8:2 вывод идет с точностью до сотых.

При выводе в конце каждой строки следует давать команду `Writeln` для перехода на новую строку. Определить, что пора менять строку, можно по остатку от деления текущего номера элемента на количество значений в строке. Ниже приведен пример вывода на печать в выводной текстовый файл `Fout` одномерного вещественного массива по "k" значений в строке:

```
Assign(Fout,'UMNIK4.RES'); {связали Fout с Н.Д. UMNIK4.RES }
ReWrite(Fout);                { открыли файл для записи }
Writeln(Fout,' Исходный массив из ',N,' элементов');
for i:=1 to N do
Begin
  Write(Fout,Massiv2[i]:8:2); { печать в текущей строке}
  if i mod k = 0 then Writeln(Fout); {если номер элемента
                    кратен "к", переходим на новую строку}
End;
. . .
Close(Fout);      {закрытие файла }
```

3. **Обработка массива**. Здесь могут встретиться различные ситуации, на которые следует обратить внимание.

а) Если в задаче предлагается использовать признак делимости на некоторое число значения переменной (например, требуется что-то делать с каждым элементом массива, который нацело делится на 5), то следует в цикле брать каждый элемент, и проверять на равенство нулю остатка от деления значения элемента массива на 5:

```
for i:=1 to N do
begin
  if K[i] mod 5 = 0 then
  { делать что надо, т.к. элемент массива делится на 5} ;
. . .
end;
```

б) Если же предлагается использовать признак делимости номера элемента, то разумнее сначала вычислить количество выбираемых номеров и организовать цикл по вычисленному количеству, каждый раз определяя, какой элемент массива надо обрабатывать.

Например, если надо что-то делать с каждым пятым элементом массива ***K*** длиной ***N*** элементов, то цикл обработки будет иметь вид:

```
M:=N div 5;  { определение числа повторений цикла }
for i:=1 to M do
begin
  j:=i*5; { определение номера обрабатываемого элемента}
  {  обработка K[j]-го элемента массива }
  .  .  .
end;
```

Если нужно обрабатывать каждый пятый элемент, начиная с третьего, программа будет выглядеть по-другому:

```
j:=3;
M:=(N-(3-1)) div 5;{определение числа повторений цикла}
for i:=1 to M do
begin     {  обработка K[j]-го элемента массива }
  .  .  .
  j:=j+5; {определение номера следующего обраб.элемента}
end;
```

в) Если требуется обрабатывать символьный массив, например, подсчитать, сколько раз встречается символ '©', нужно уметь, во-первых, присваивать символьным переменным нужные значения, во-вторых, вводить символы в массив с клавиатуры, и, в-третьих, уметь сравнивать символьные элементы массива.

Присваивание значения переменной можно сделать либо перенося это значение из другой переменной (если она содержит в данный момент нужный символ), либо задавая присваивание константы, либо используя функ-

цию Chr(i), преобразования целого числа в символ (с кодом, равным этому числу). Таблица кодов символов приведена в ***Приложении Г***. Примеры присваивания:

```
Const
      SimA ='A';
      Paragraf = #21;
      Kod = 21;
      SixKod = $15;
Var S1,S2,S,S4,S5,S6 : char;
  . . .
Begin
  . . .
  S1:= SimA;
  S2:=S1;
  S3:='A';
  S3:=Paragraf;
  S4:=#21;
  S5:=#$15;
  S6:=Chr(21);
  S7:=Chr($15);
```

Пример используемых операторов:

```
. . .
Var Sim : char;
  TextM:array[1..100] of char;{завели символьный массив}
  i,N : shortint;
    . . .
Begin
  . . .
{Если заполнение символьного массива TextM с клавиатуры}
  for i:=1 to N do
  begin
    writeln('Вводи очередной символ');
    readln(TextM[i]);
  end;
  . . .
{Если заполнение символьного массива TextM из файла}
  assign(fin,'UMNIK4.DAT');
  reset(fin);
  for i:=1 to N do
    read(fin,TextM[i]);
  . . .
  close(fin); { не забыть в конце закрыть все открытые файлы }
```

4. **Вывод результатов.** Если в процессе выполнения программы исходный массив изменяется (в нем меняются сами значения элементов, их ко-

личество или они переставляются), исправленный массив должен выводиться в конце программы под заголовком 'Измененный массив'.

Разбор контрольного варианта

Задание.

Таблица 16. Данные к заданию 31 варианта

№ вар.	Задание	Печатать элементы массива по		Тип обрабат. данных
		штук:	формату:	
31	Формирование символьного массива длиной не более ста элементов, заполнение его с клавиатуры (вводя по одному произвольному символу пока не встретится символ "."), подсчет и вывод на экран символа, который встретился чаще других и число его повторений.	30	:2	Символьный

При решении этой задачи нужно выделить место на символьный массив из ста элементов, заполнить его (в итеративном цикле – пока не встретится символ '.' или не введется сто символов), распечатать его (по 30 символов в строке) и провести подсчеты частоты вхождения каждого символа. Последняя часть представляет наибольшую сложность, поэтому остановимся на ней подробнее.

Так как нужно найти самый частый символ и количество его повторений, необходимо в программе завести переменные символьного типа для рассматриваемого символа (`SimI`) и для самого частого символа (`SimMax`), а также счетчики для рассматриваемого (`Ni`) и самого частого (`Nmax`) символов. В качестве `SimI` будем брать по очереди каждый символ из массива (во внешнем цикле по `i` – номеру в массиве длиной `N` элементов) и для него считать, сколько раз этот символ встречается в массиве (внутренний цикл по `j` – для всех элементов массива). Если после подсчета `Ni` окажется больше `Nmax`, значение `Ni` переносится в `Nmax`, а `SimI` – в `SimMax`.

Очевидно, перед внешним циклом `Ni` следует обнулить, а в `SimMax` занести, например, первый элемент символьного массива.

Длина массива `N` подсчитывается один раз при заполнении массива (и не может превышать 250).

Понятно, что данный алгоритм не самый эффективный, так как любой символ, встречающийся несколько раз, мы будем обсчитывать столько раз, сколько он встречается. Но пренебрежение этим обстоятельством позволит существенно упростить алгоритм, так как не нужно запоминать и каждый раз в начале тела внешнего цикла проверять, какие символы мы уже обсчитали.

Для разбираемого варианта не приводятся таблица идентификаторов и блок-схема алгоритма, хотя при оформлении лабораторной работы их следует включить в отчет.

С учетом выполненного выше разбора задачи, программа может иметь вид:

Текст программы

```
PROGRAM  Simbol_Array;
{   Программа Лабораторной работы N 4
            Вариант N 31.
    гр. Я-007, ст. А.Я. Умненькая }
VAR
  N,Ni,Nmax,i,j :integer;
  SimI,SimMax   :char;
  Simbol        :array[1..100] of char;
  Fout          :text;
BEGIN   {Выполняемый блок. Первый раздел - ввод данных }
  N:=0;
  writeln('Вводить по одному не более 100 символов,');
  writeln(' Для завершения - ввести символ "."');
  repeat
    writeln('Введи очередной символ');
    N := N + 1;
    readln(Simbol[N]);
  until (N = 100) or (Simbol[N] = '.');
  if Simbol[N]= '.' then N := N - 1; { Последний символ,
если он - точка, рассматриваться не будет }
  if N < 1 then
    writeln('Нельзя рассматривать меньше 1 символа')
  else
  begin
    Assign(Fout,'UMNIK4.RES'); {связали Fout с UMNIK.DAT}
    ReWrite(Fout);              {открыли файл для записи }
    writeln(Fout,'Исходный массив из ',N,' элементов');
    for i:=1 to N do
    begin
      write(Fout,Simbol[i]:2); { печать в текущей строке}
      if i mod 30 = 0 then writeln(Fout); { если номер
            элемента кратен 30 переходим на новую строку}
    end;
    writeln(Fout);
        { Основной раздел - поиск номера }
    Nmax := 0;    { количество наиболее частого символа }
```

```
    SimMax := Simbol[1];
    for i:=1 to N do { внешний цикл перебора символов }
    begin
      SimI := Simbol[i];
      Ni := 0;
      for j:=i to N do { внутренний цикл перебора символов }
        if Simbol[j] = SimI then Ni:=Ni+1;
      if Ni>Nmax then {если этот символ встретился чаще,}
      begin           {запомним его и число его вхождений}
        Nmax := Ni;
        SimMax := SimI;
      end;
    end;
       { Раздел вывода результатов поиска }
    if Nmax = 1 then
      writeln(Fout,'Все символы входят по 1 разу')
    else
      write(Fout,'Символ "',SimMax,'" встретился ',Nmax,
                 ' раз');
       { далее определим, если Nmax кончается на 2,3,4 и не во
втором десятке, изменим окончание, например, "23 раза" }
      if (Nmax mod 10 < 5) and (Nmax mod 10 > 1) and
         (Nmax div 10 <> 1)
        then writeln(Fout,'а');
  end;
  Close(Fout);
END.
```

Результаты работы

(содержимое файла UMNIK.RES)

```
Исходный массив из 95 элементов
 Н е г о в о р и т е м н е , о н у м е р - о н ж и в е т ! П
 у с т ь ж е р т в е н н и к р а з б и т - о г о н ь е щ е п
 ы л а е т , п у с т ь р о з а с о р в а н а - о н а е щ е ц
 в е т е т
Символ "е" встретился 14 раз
```

Варианты заданий

Таблица 17. Варианты заданий лабораторной работы №4

№ вар.	Задание	Печатать элементы массива по		Тип обрабатыв. данных
		штук:	формату:	
1	Формирование массива длиной не более ста элементов, заполнения его с клавиатуры (вводя по одному произвольному символу пока не встретится символ '!'), исключение из массива всех символов – цифр и пробелов со сдвигом остающихся элементов массива на освободившиеся места.	30	:1	Символьн.
2	Нахождение номера наименьшего элемента в массиве заданной длины среди всех положительных элементов. Длина массива вводится с клавиатуры.	5	:11	Веществ.
3	Вычисление суммы "***K***" слагаемых – элементов массива, начиная с третьего по порядку, и суммируя только элементы с нечетными номерами. Длина массива и количество суммируемых элементов вводится с клавиатуры.	6	:9:2	Веществ.
4	Перестановка максимального и минимального по значению элементов массива. Значение длины массива вводится с клавиатуры.	8	:7	Целые
5	Вычисление суммы всех четных (по значению) элементов массива, расположенных на нечетных по порядку местах. Длина массива вводится с клавиатуры.	10	:6	Целые
6	Нахождение максимального значения в массиве среди всех четных чисел. Длина массива вводится с клавиатуры.	9	:6	Целые
7	Вычисление суммы элементов массива, начиная с "***K***"-го по порядку и до элемента, равного нулю (если такой элемент встретится; иначе – до конца массива). "***K***", длину массива и значения элементов вводить с клавиатуры.	5	:12	Веществ.
8	Перестановка "i"-го и "j"-го по порядку элементов массива, при условии, что они с разными знаками. Если они с одинаковыми знаками, все элементы между ними обнулить. Длина массива и номера переставляемых элементов вводятся с клавиатуры.	10	:6	Целые

№ вар.	Задание	Печатать элементы массива по		Тип обрабатыв. данных
		штук:	формату:	
9	Вычисление суммы элементов массива, расположенных в конце массива, причем складывать нужно, начиная от конца массива, столько элементов, пока сумма не превысит значения 20.5 (или не будут сложены все элементы). Длина массива вводится с клавиатуры. Печатать сумму и количество сложенных элементов	6	:10:3	Веществ.
10	Нахождение максимального значения в массиве среди всех элементов после первого отрицательного. Длина и значения элементов массива вводятся с клавиатуры.	10	:5	Целые
11	Вычисление суммы элементов массива начиная с первого элемента со значением больше 0.9, и пока сумма по модулю не превысит заданного значения. Длина массива, значения элементов и предельное значение (признак для окончания суммирования) вводятся с клавиатуры.	5	:9:3	Веществ.
12	Перестановка максимального и минимального по коду символов массива длиной N элементов. Значение длины массива и сами элементы массива вводятся с клавиатуры. Массив печатать до и после перестановки.	30	:2	Символьн.
13	Формирование массива длиной N элементов, заполняя его с клавиатуры (вводя сначала количество символов, и затем по одному произвольному символу пока не введется указанное количество), подсчет и вывод на экран количества символов из диапазона от 'А' до 'я' (кириллицы).	30	:2	Символьн.
14	Вычисление среднего значения в одномерном массиве для всех элементов между первым и вторым нулем в массиве (или от первого нулевого значения до конца массива). Длина и значения элементов массива вводятся с клавиатуры.	5	:12	Веществ.
15	Перестановка одномерного массива в обратном порядке. Значение длины массива вводится с клавиатуры. Массив печатать до и после перестановки.	9	:8	Целые
16	Вычисление в одномерном массиве целой части среднего значения всех положительных четных по величине чисел. Длина массива вводится с клавиатуры.	8	:8	Целые
17	Исключение из массива всех пробелов и запятых со сдвигом остающихся элементов массива на освободившиеся места. Исходный массив длиной N символов (где N не больше 100) вводится из текстового файла.	30	:1	Символьн.

№ вар.	Задание	Печатать элементы массива по		Тип обрабатыв. данных
		штук:	формату:	
18	Вычисление целого среднеарифметического значения среди всех отрицательных элементов массива (содержащего и положительные значения), расположенных начиная с "***K***"-го по порядку элемента. Длина массива и значение "***K***" вводятся с клавиатуры.	10	:6	Целые
19	Определение длины самой длинной последовательности из расположенных подряд в одномерном логическом массиве значений **TRUE** и вывод найденной длины на экран. Длина массива и значения **TRUE** и **FALSE** вводятся с клавиатуры в форме ***F*** (для **FALSE**) и ***T*** (**TRUE**).	8	:7	Логич.
20	Вычисление среднеарифметического всех положительных элементов массива (содержащего и отрицательные значения), длина которого вводится с клавиатуры.	6	:10:2	Веществ.
21	Поиск места (номера элемента) в массиве, где первый раз подряд встречаются два четных числа. Длина массива вводится с клавиатуры.	10	:6	Целые
22	Формирование логического массива длиной N элементов, заполнения его с клавиатуры (вводя 1 – вместо **TRUE** и 0 – вместо **FALSE**), подсчет и вывод на экран количества значений **TRUE** и **FALSE** и сообщение, чего было больше.	10	:6	Логич.
23	Поиск номеров двух последних расположенных подряд отрицательных элементов в массиве. Длина массива вводится с клавиатуры.	6	:10	Веществ.
24	Формирование логического массива, заполняя его с клавиатуры (вводя вместо **TRUE** четные числа, а вместо **FALSE** – нечетные и заканчивая, когда встретится число 0); подсчет и вывод на экран количества значений **TRUE** и **FALSE** и сообщение, каких значений было больше.	12	:5	Логич.
25	Нахождение номера максимального значения в массиве после первого отрицательного и не далее второго отрицательного. Длина массива вводится с клавиатуры.	5	:10:3	Веществ.
26	Формирование символьного массива длиной N элементов, содержащего только цифры. (При заполнении его с клавиатуры пропускать не цифровые символы.) Подсчет и вывод на экран цифры, встретившейся чаще других (и сколько раз она встретилась.)	25	:3	Символьн.

№ вар.	Задание	Печатать элементы массива по		Тип обрабатыв. данных
		штук:	формату:	
27	Исключение из массива всех отрицательных по значению элементов со сдвигом элементов массива на освободившиеся места. Длина массива вводится с клавиатуры.	6	:9:2	Веществ.
28	Переписывание из исходного в выходной массив всех нечетных значений в начало, а всех четных – в конец. Длина исходного массива вводится с клавиатуры.	10	:6	Целые
29	Определение номера элемента, с которого начинается самая длинная последовательность четных чисел, расположенных в массиве подряд. Длина массива вводится с клавиатуры	9	:6	Целые
30	Определение номера элемента, с которого начинается самая длинная последовательность из отрицательных чисел, расположенных в массиве подряд. Длина массива вводится с клавиатуры.	6	:10:2	Веществ.
31	Формирование символьного массива длиной не более ста элементов, заполнение его с клавиатуры (вводя по одному произвольному символу пока не встретится символ "."), подсчет и вывод на экран символа, который встретился чаще других и число его повторений	30	:2	Символьн.

Лабораторная работа № 5
Работа с двумерными массивами
(сортировки и перестановки в массиве)

Задачи лабораторной работы

Вопросы, изучаемые в работе

- Изучение двумерных массивов (матриц).
- Использование операторов описаний типов пользователя.
- Освоение форматного вывода двумерных массивов в виде таблиц.
- Работа с нечисловыми типами данных.
- Изучение основных методов (алгоритмов) сортировки массивов данных.

Задание (общее ко всем вариантам)

Составить программу работы с двумерным массивом заданного типа, но с произвольными размерами (по количеству строк и столбцов) в пределах отведенного под массив места. Заполнить массив данными из имеющегося текстового файла, распечатать исходный массив в выводной текстовый файл. Провести обработку массива в соответствии с заданием, результаты работы вывести в тот же выводной файл.

Оформить отчет по лабораторной работе аналогично оформлению работы № 4.

Требования к программе

- Все значения, на которые по смыслу накладываются ограничения, должны при вводе проверяться.
- При выводе использовать длину выводимой строки не более 80 символов.
- Если требуется вводить вещественные данные, их следует читать из файла `D:\LAB1\DATF.TXT`, если целые – то из `D:\LAB1\DATI.TXT`, если символьные данные, то из файла `D:\LAB1\SIMBOL.TXT`, если логические – из `D:\LAB1\LOGICAL.TXT`.

Содержание программы.

- Заголовок программы с комментарием;
- Описание типов для массивов;
- Описание переменных, меток, констант (если надо);
- Ввод с клавиатуры размеров массива и других данных, необходимых программе;
- Ввод требуемого количества данных из указанного текстового файла с числами. Файл находится на диске **D:** в каталоге **\LAB1;**
- Вывод в выходной текстовый файл исходного массива под заголовком "Исходный массив";
- Проведение обработки числового материала в соответствии с заданием, указанным в выбранном варианте;
- Вывод результатов обработки в выходной текстовый файл под заголовком "Результаты расчета";
- Закрытие всех открытых файлов.

Общие пояснения

1. **Описание и использование двумерных массивов.** Как и одномерные, двумерные массивы представляют совокупность однотипных переменных, имеющих одинаковое имя и размещенных в памяти ЭВМ подряд. Однако, для обращения к отдельному элементу (компоненту) массива, требуется указывать уже два индекса. Для наглядности, двумерные массивы изображают на бумаге в форме прямоугольной таблицы, поэтому часто такие массивы с числовыми значениями элементов называют матрицами.

В Turbo Pascal компоненты массива могут быть любого (в том числе – составного) типа, индексы могут быть любого порядкового типа (т.е. не вещественного), но не `Longint`. В данной лабораторной работе будут рассматриваться компоненты только простого типа.

Описание массива можно производить двумя способами: через задание типа массива и непосредственно.

Например:

```
TYPE  {Создание массивов с использованием описателей типов}
      mas1 = array[1..100] of integer; { описатель для целочисленных одномерных массивов длиной не более 100 элементов }
      vector = array [1..30] of real;   { описатель для вещественных одномерных массивов длиной не более 20 элементов }
      mas2 = array[1..8, 1..10] of Char;  { описатель для символьных двумерных массивов размерами не более 8 строк и 10 столбцов }
```

```
    matrix = array[1..12] of vector;  { описатель для вещественных двумерных массивов размерами не более 10 строк и 20 столбцов}
```

Последний описатель можно было бы задать без использования описателя типа vector:

```
    matrix = array[1..12, 1..30] of real;  }
VAR          {здесь выделяется место под все массивы }
  Names                :mas2;
  Numbers, Ages        :mas1;
  Day_Tempr,Day_Wind:vector;
  Tempr1996            :matrix;
```

Создание массивов без использования специальных описателей типов:

```
VAR  Ball_Groop_1,B_M170 : array[1..30] of real;
{ выделение места под два одномерных вещественных массива}
     Prizn : array [1..10,1..10] of boolean;
{ квадратный массив логических переменных}
    B_M175 : array[1..30] of real;
    Ball_Kurs : array[1:12, 1..30] of real;
{ выделение места под двумерный вещественный массив}
```

Массивы, созданные одним оператором описания или разными операторами, но использующими общий описатель, являются совместимыми; для них возможны операции присваивания, операции отношения (только равно и не равно). Для массивов описанных выше можно написать операторы:

```
  . . .
  Ball_Groop1 := B_M170;
  Numbers:= Ages;
  If Day_Tempr=Day_Wind then ...
```

Но недопустим ни один из операторов:

```
  Ball_Kurs    := Tempr1996;
  Ball_Groop1  := B_M175;
  B_M175       := B_M170;
  While  B_M175<> B_M170 do ...
```

2.**Алгоритмы основных методов сортировок** (для всех случаев как пример рассмотрена сортировка чисел по возрастанию)

Алгоритм сортировки пересчетом. Этот метод относится к сортировкам в двух массивах. Берется первый элемент исходной последовательности и подсчитывается, сколько элементов имеют величину, меньшую чем взятый. Проверяемый элемент помещается в выходной массив на место, равное подсчитанному числу плюс один (но если в этом месте уже записано число – то на следующее место). Это проделывается для следующего элемента исходной последовательности, и так до конца. В выходном массиве элементы упорядочены. Чтобы определить, занято ли место в выходном массиве, его предварительно следует заполнить "признаком

свободного места" – таким числом, которого нет в сортируемом массиве. В качестве такого числа можно взять значение, например, на единицу больше максимального числа в сортируемом массиве.

Алгоритм сортировки выбором. Это метод сортировки на месте (в одном массиве). В массиве ищется минимальный элемент и переставляется на место первого элемента (а первый элемент – на место найденного минимального). Потом этот процесс повторяется, но массив рассматривается со второго элемента (первый уже на месте). Найденный минимальный элемент меняется местами со вторым. Это повторяется до (n–1)-го элемента (n-ый элемент автоматически окажется на месте). Примечание: при поиске минимального элемента нужно находить и запоминать его место – для перестановки элементов.

Алгоритм сортировки обменом. Это также метод сортировки на месте (в одном массиве); некоторые алгоритмы этого метода носят название метода пузырька. Сравниваются два соседних элемента (первый и второй), если они в правильном порядке – ничего не делается, если порядок нарушен – они меняются местами. Потом этот процесс повторяется для второго и третьего элементов, и так до конца (n–1 и n-ый элементы). В результате самый большой элемент оказывается в самом конце (на n-ом месте). Затем весь этот процесс повторяется, но уже до n–1-го элемента (так как n-ый уже на месте); затем – до n–2-го и так всего n–1 раз. Тем самым n–1 элементов снизу окажутся упорядоченными, а следовательно и первый элемент тоже.

Алгоритм сортировки вставками. Этот метод относится к сортировкам в двух массивах. Имеется два массива. Один заполнен неупорядоченными числами, второй – пустой. Из первого берется первое число и переписывается на первое место во второй массив. Берется второе число и сравнивается с переписанным во второй массив. Если уже записанное число больше второго, оно переписывается на одну позицию ниже (на второе место), а последнее взятое число – на место первого. Затем из исходного массива берется третье число и сравнивается со вторым из выходного массива. Если оно должно вставляться выше второго, второе переписывается на позицию ниже. Далее сравнение проводится с первым числом и, если вставляемое число больше или равно первому, оно записывается на место, освободившееся от второго числа, иначе первое переписывается на позицию ниже, а вставляемое – на его место. Процесс повторяется, пока не будут вставлены на нужные места во второй массив все числа из первого.

Сортировка слиянием. Этот метод предполагает, что имеется два или несколько наборов данных, каждый из которых уже упорядочен. Нужно объединить их в один упорядоченный набор данных. Как входные, так и выходной наборы могут размещаться во внешней памяти (в файлах). Алгоритм заключается в том, что заводится столько указателей, сколько

входных наборов данных объединяется одновременно; каждый указатель сначала указывает на первый элемент в каждом наборе. Выбирается самый маленький элемент из тех, на которые указывают эти указатели, и он переписывается в выходной массив, после чего указатель набора, из которого был переписан элемент, увеличивается на единицу (и теперь указывает на второй элемент этого набора). Далее снова производится выбор наименьшего элемента из тех, на которые указывают указатели, и так пока не будут выбраны все элементы из всех наборов. Если кончается один из наборов, а их всего два, то оставшиеся элементы второго набора дописываются в конец выводного файла. Если сливаются одновременно более двух наборов, вычерпанный набор перестает рассматриваться при выборе минимального элемента, или в этом наборе искусственно создается дополнительный элемент с очень большим ключом (каких не может быть у реальных элементов) и при выборе он никогда не будет взят. При этом не меняется схема выбора. Если первоначально было N упорядоченных коротких наборов данных, то объединяя их попарно этим методом, после первого прохода станет N/2 упорядоченных наборов, но каждый из них будет уже в два раза длиннее. Повторяя этот процесс и беря каждый раз выходные наборы в качестве входных для слияния, можно в итоге получить один упорядоченный набор. Первоначальные N наборов можно получить из исходного, например, беря по паре элементов, поменяв их местами, если они не в правильном порядке.

Разбор контрольного варианта

Задание

Таблица 18. Данные к заданию 31 варианта

№ вар.	Задание	Предельные значения числа...		Тип обраб. данных
		строк	столбцов	
31	Массив размером M строк N столбцов заполнить числами из входного текстового файла. Исходный массив распечатать. Столбец, где расположен минимальный элемент массива, упорядочить по возрастанию методом пересчета в одномерный массив. Созданный массив распечатать.	7	7	Веществ.

Формирование таблицы идентификаторов

В задании упоминаются: двумерный массив вещественного типа, количество строк (M<=7), количество столбцов (N<=7), входной текстовый файл, номер столбца с минимальным элементом, одномерный массив веще-

ственного типа. Кроме того, так как придется перебирать элементы двумерного массива, потребуется использовать текущий номер строки (`i`) и текущий номер столбца (`j`) элемента массива, а при поиске минимального элемента – использовать переменную для найденного минимума.

Сортировка пересчетом потребует заводить счетчик (`ki`) числа элементов, меньше рассматриваемого "`i`-го" элемента сортируемого столбца, признак незанятого места в выходном массиве (`Pr`), дополнительные счетчики в циклах перебора элементов при подсчете и записи (`i1`, `i2`). Еще можно упомянуть имя программы и выходного файла для результатов.

На основании перечисленных объектов таблица будет содержать:

Таблица 19. Идентификаторы программы 31 варианта

Имя	Тип	Р-р,байт	Назначение
SortNum	Имя программы	-	Сортировка столбца пересчетом
A	Веществ.массив	294	Исходный массив
B	Веществ.массив	42	Упорядоченный (выходной) массив
M	Целое	2	Количество строк (длина столбца)
N	Целое	2	Количество столбцов (длина строки)
Jmin	Целое	2	Номер строки с минимальным элементом
Fin	Текстовый файл	128	Файл с исходными числами
Fout	Текстовый файл	128	Файл с результатами работы
i	Целое	2	Текущий номер строки
j	Целое	2	Текущий номер столбца
Amin	Вещественное	6	Копия минимального элемента массива
ki	Целое	2	Число элементов массива < текущего
Pr	Вещественное	6	Признак незаполненного элемента
i1	Целое	2	Вспомогательный счетчик перебора
i2	Целое	2	Вспомогательный счетчик перебора
Vvod_M	Метка	-	Блок ввода числа строк
Vvod_N	Метка	-	Блок ввода числа столбцов

Алгоритм

Должен содержать следующие шаги:

1. Открытие входного и выходного файлов. Текстовый входной файл связывается с набором данных с вещественными числами `'D:\LAB1\DATF.TXT'` и открывается для чтения. Выходной файл связывается с новым набором данных `'UMNIK5.RES'` (т.к. без указания пути – значит в текущем каталоге) и открывается для записи.

2. Ввод количества строк массива. Контроль на попадание M в диапазон 2..7. Потребуется метка возврата (Vvod_M).
3. Ввод количества столбцов массива. Контроль на попадание N в диапазон 2..7. Потребуется метка возврата (Vvod_N).
4. Двойной цикл заполнения массива числами из файла. Запрос на ввод не требуется, так как ввод из файла. Контроль не требуется.
5. Распечатка исходного массива. Сначала печать заголовка ("Исходный массив"), затем печать по строкам с форматом, выделяя на число 10 позиций. После печати каждой строки массива – переход на новую запись в выходном файле.
6. Поиск столбца с минимальным элементом, запоминание его номера. Двойной цикл (по всем элементам массива), перед которым первый элемент запоминается в переменной Amin, а в переменной для номера Jmin запоминается 1.
7. Определение признака незанятого элемента выходного массива. В качестве признака незанятого элемента выходного массива следует использовать число, которое наверняка не встречается в исходном массиве. Таким числом будет любое, меньшее чем минимальное в массиве, в частности Amin –1.
8. Цикл заполнения выходного массива признаком незанятого места. Простой цикл присваивания M элементам массива A значения Pr.
9. Цикл работы с "i-м" элементом сортируемого столбца:
    a) Цикл расчета числа элементов, меньших "i-го". Перед циклом ki делаем равным не нулю, а единице, т.к. после подсчета числа элементов, меньших данного, его выходной номер должен стать на единицу больше этого числа.
    b) Поиск места в выходном массиве, для рассматриваемого элемента. Если найденное место уже занято, следует двигаться дальше по массиву, пока не встретится незанятое место. Этот процесс необходим, если в сортируемом ряду несколько одинаковых чисел, претендующих, на одно и тоже место.
10. Печать номера сортированного столбца.
11. Печать отсортированного массива (начинается с печати заголовка "Упорядоченный массив"), и печать одномерного выходного массива с тем же форматом, что и для входного массива.
12. Закрытие файлов.

При оформлении лабораторной работы, алгоритм следует изобразить в форме блок-схемы.

Текст программы.

```
PROGRAM  SortNum;
{    Программа Лабораторной работы N 5
            Вариант N 31.
      А.Я.Умненькая, ст. гр. Я-007 }
VAR
  M,N,Jmin,i,j,i1,ki : integer;
  Amin,Pr            : real;
  A : array [1..7,1..7] of real;
  B : array [1..7] of real;
  Fin,Fout : text;
LABEL
  Vvod_M, Vvod_N;
BEGIN
{ Открытие  входного  и выходного файлов }
  assign(Fout,'UMNIK5.RES');
  rewrite(Fout);
  assign(Fin,'D:\LAB1\DATF.TXT');
  reset(Fin);
{ Ввод  количества строк массива }
  Vvod_M:
    writeln ('Введите число строк массива');
  readln(M);
  if (M < 2) or (M > 7) then
  begin
    writeln('Недопустимое значение!');
    goto Vvod_M;
  end;
{ Ввод  количества столбцов массива }
  Vvod_N:
    writeln ('Введите число столбцов массива');
  readln(N);
  if (N < 2) or (N > 7) then
  begin
    writeln('Недопустимое значение!');
    goto Vvod_N;
  end;
{ заполнение массива числами из файла }
  for i:=1 to M do
    for j:=1 to N do
      read(Fin,A[i,j]);
{ Закрытие входного файла }
  close(Fin);
{ Распечатка исходного массива }
```

```
  Writeln(Fout,' Исходный массив из ',M,'x',N,
                ' элементов');
  for i:=1 to M do
  begin
    for j:=1 to N do
      Write(Fout,A[i,j]:8:2); {печать текущей строки }
    Writeln(Fout);
  end;
  Writeln(Fout);
{ Поиск  столбца с минимальным элементом }
  Amin := A[1,1];
  Jmin := 1;
  for i:=1 to M do
    for j:=1 to N do
      if A[i,j] < Amin then
      begin
        Amin := A[i,j];
        Jmin := j;
      end;
{ Определение признака незанятого места }
  Pr := Amin - 1.0;
{ Очистка выходного  массива }
  for i:=1 to M do B[i] := Pr;
{ сортировка "Jmin-го" столбца в выходной файл пересчетом }
  for i:= 1 to M do
  begin
    ki := 1;
    for i1:=1 to M do
      if A[i1,Jmin] < A[i,Jmin] then ki := ki+1;
{ сдвиг дальше, если элемент уже занят }
    while B[ki] <> Pr do ki := ki+1;
    B[ki] := A[i,Jmin];
  end;
{ Печать номера столбца с минимальным элементом }
  Writeln(Fout,'Результаты работы':30);
  Writeln(Fout);
  Writeln(Fout,' Минимальный элемент находится в ',
                 Jmin,' столбце');
{ Печать отсортированного массива }
  Writeln(Fout,'Сортированный массив':30);
  for i:=1 to M do
    Write(Fout,B[i]:8:2);
  Writeln(Fout);
{ Закрытие выходного файла }
```

```
  close(Fout);
END.
```

Содержимое файла результатов UMNIK5.RES

```
Исходный массив из 7x7 элементов
  -2.20   -6.93    0.20    8.97    8.09    5.38    7.82
   5.43   15.33   13.60    9.32   17.38   17.70   16.26
  13.13   13.78   20.59   17.91   15.16   19.02   21.66
  18.71   17.73   20.59   21.77   22.49   21.99   19.42
  15.68   17.73   21.90   13.50   17.68   18.50   15.51
  11.54   19.04   17.86   13.20   10.25   10.44    8.46
  11.92    5.60    5.75    2.69    8.51    0.52    3.61

            Результаты работы
Минимальный элемент находится в 2 столбце
          Сортированный массив
  -6.93    5.60   13.78   15.33   17.73   17.73   19.04
```

Варианты заданий

Таблица 20 Варианты заданий лабораторной работы №5

№ вар.	Задание	Предельные значения числа		Тип обраб. данных
		строк	столбцов	
1	Массив размером **M**, строк **N** столбцов заполнить числами из входного текстового файла. Исходный массив распечатать. Каждую строку массива упорядочить по убыванию обменом. Переделанный массив распечатать.	10	10	Целые
2	Массив размером **M** строк **N** столбцов заполняется символами из входного текстового файла. Исходный массив распечатывается. Каждый столбец массива упорядочить по возрастанию методом выбора. Переделанный массив распечатать.	20	20	Символьн.
3	Заполнить квадратный массив размерами **NxN** элементов числами из входного файла, распечатать, поменять местами столбец с номером "**K**" и строку с номером "**M**" (оба номера и длина массива вводятся с клавиатуры). Полученную матрицу распечатать.	10		Целые
4	Заполнить двумерный массив **A(MxN)** элементов символами из текстового файла. Первый столбец массива **A** упорядочить по возрастанию (по алфавиту) методом вставок, используя рабочий одномерный массив **B(M)**. Массив **A** распечатать до и после сортировки.	16	16	Символьн.
5	Массив размером **M** строк **N** столбцов заполняется числами из входного текстового файла. Исходный массив распечатывается. Столбец с максимальным последним элементом упорядочить по возрастанию методом обмена. Переделанный массив распечатать.	10	10	Целые
6	Заполнить двумерный массив **A(MxN)** элементов символами из одномерного массива **B** (длиной не более 256 элементов) "змейкой" от конца к началу – сперва **N**-ю строку справа налево, затем **N**–1-ю слева направо и т.д. Массив **B** предварительно заполняется из входного текстового файла. Оба массива распечатать.	16	16	Символьн.

№ вар.	Задание	Предельные значения числа		Тип обраб. данных
		строк	столбцов	
7	Квадратный массив размером **MxM** заполняется числами из входного текстового файла. Исходный массив распечатывается. Главную диагональ массива упорядочить по убыванию методом выбора. Переделанный массив распечатать.	10		Целые
8	Массив размером **M** строк **N** столбцов заполняется числами из входного текстового файла. Исходный массив распечатывается. Столбец с минимальным 1-ым элементом упорядочить по убыванию методом обмена. Переделанный массив распечатать.	6	6	Веществ.
9	Массив размером **M** строк **N** столбцов заполняется символами из входного текстового файла. Исходный массив распечатывается. Заказанную строку массива (номер которой вводится с клавиатуры) упорядочить по убыванию методом выбора. Переделанный массив распечатать.	19	19	Символьн.
10	Массив размером **M,** строк **N** столбцов заполняется числами из входного текстового файла. Исходный массив распечатывается. Заказанный столбец массива (номер которого вводится с клавиатуры) упорядочить по возрастанию методом обмена. Переделанный массив распечатать.	7	7	Веществ.
11	Квадратный массив размером **MxM** заполняется числами из входного текстового файла. Исходный массив распечатывается. Обратную диагональ массива упорядочить по возрастанию методом выбора. Переделанный массив распечатать.	7		Веществ.
12	Массив размером **M** строк, **N** столбцов заполняется числами из входного текстового файла. Исходный массив распечатывается. Столбец с минимальным вторым элементом упорядочить по возрастанию методом выбора. Переделанный массив распечатать.	10	10	Целые
13	Заполнить квадратный массив **A (NxN)** элементов (**N** не более 8) числами из входного файла, распечатать, построить новую матрицу **B (NxN)**, в которой каждый элемент **B (i,j)** равен минимальному из всех элементов массива **A**, у которых номер строки <=i, а номер столбца >=j. Полученную матрицу распечатать.	8		Веществ.

№ вар.	Задание	Предельные значения числа		Тип обраб. данных
		строк	столбцов	
14	Массив размером **M** строк, **N** столбцов заполняется числами из входного текстового файла. Исходный массив распечатывается. Строку, где расположен максимальный элемент массива, упорядочить по убыванию методом обмена. Переделанный массив распечатать.	10	10	Целые
15	Массив размером **M** строк, **N** столбцов заполняется числами из входного текстового файла. Исходный массив распечатывается. Строку с минимальным первым элементом упорядочить по убыванию методом выбора. Переделанный массив распечатать.	6	6	Веществ.
16	Массив размером **M** строк, **N** столбцов заполняется числами из входного текстового файла. Исходный массив распечатывается. Строку, где расположен минимальный элемент массива, упорядочить по убыванию методом пересчета в одномерный массив, который распечатать.	10	10	Целые
17	Массив размером **M** строк, **N** столбцов заполняется числами из входного текстового файла. Исходный массив распечатывается. Заказанный столбец массива (номер которого вводится с клавиатуры) упорядочить по убыванию методом вставок, используя рабочий одномерный массив, который затем переписать на место сортируемого столбца. Переделанный двумерный массив распечатать.	12	12	Целые
18	Заполнить двумерный массив логическими значениями из входного текстового файла, распечатать, перестроить матрицу: первый столбец поменять местами с последним, второй с предпоследним, и т.д. Затем так же переставить строки. Переделанный массив распечатать.	9	9	Логич.
19	Массив размером **M** строк **N**, столбцов заполняется числами из входного текстового файла. Исходный массив распечатывается. Второй столбец массива упорядочить по возрастанию методом пересчета в одномерный массив. Созданный массив распечатать.	12	12	Целые
20	Заполнить двумерный массив размерами 20x20 элементов символами из входного текстового файла, распечатать, перестроить матрицу: исключить из нее столбец с номером "**k**" и строку с номером "**m**" (оба номера вводятся с клавиатуры). Полученную матрицу размерами 19x19 распечатать.			Символьн.

№ вар.	Задание	Предельные значения числа		Тип обраб. данных
		строк	столбцов	
21	Массив размером **M** строк **N**, столбцов заполняется символами из входного текстового файла. Исходный массив распечатывается. Столбец, где расположен максимальный элемент массива, упорядочить по возрастанию методом обмена. Переделанный массив распечатать.	17	17	Символьн.
22	Заполнить квадратный массив **A (NxN)** элементов (**N** вводится с клавиатуры) числами из одномерного массива **B** (длиной не более 64 элементов) "змейкой" – сперва первую колонку сверху вниз, затем вторую снизу вверх и т.д. Массив **B** предварительно заполняется из входного текстового файла. Оба массива распечатать.	8		Веществ.
23	Заполнить двумерный массив **A (MxN)** элементов числами из текстового файла. Каждую строку массива **A** упорядочить по убыванию пересчетом в соответствующую строку массива **B (MxN)**. Оба массива распечатать.	9	9	Целые
24	Заполнить квадратный массив **A (NxN)** элементов числами из входного файла, распечатать, построить новую матрицу в **B (NxN)**, в которой каждый элемент **B (i,j)** равен сумме всех элементов, у которых номер строки >=**i**, а номер столбца <=**j**. Полученную матрицу распечатать.	8	8	Веществ.
25	Прочитать из текстового файла с данными первые десять чисел в одномерный массив a последующие числа – в двумерный массив (матрицу) 7x6 элементов. Распечатать оба массива. Заменить нулями в матрице те элементы с четной суммой индексов, для которых имеются равные значения в одномерном массиве. Измененную матрицу распечатать.			Целые
26	Заполнить квадратный массив **A (NxN)** элементов числами из текстового файла. Каждый столбец массива **A** упорядочить по возрастанию вставками в соответствующий столбец массива **B (NxN)**. Оба массива распечатать.	9		Целые
27	Заполнить квадратный массив **A(NxN)** элементов логическими из одномерного массива **B** (длиной не более 81 элементов) "по спирали" – сперва первую строку слева направо, затем последнюю колонку сверху вниз, затем последнюю строку справа налево, первую колонку снизу вверх и т.д. Массив **B** предварительно заполняется из входного файла. Оба массива распечатать.	10		Логич.

№ вар.	Задание	Предельные значения числа		Тип обраб. данных
		строк	столбцов	
28	Заполнить двумерный массив **A (2xN)** элементов символами из текстового файла. Распечатать массив **A**. Далее каждую строку массива **A** упорядочить по алфавиту обменом, затем заполнить по алфавиту одномерный массив **B** (длиной **2N** элементов) методом слияния из обеих строк массива **A**. Оба массива распечатать после сортировок.		15	Символьн.
29	Заполнить двумерный массив **A (Mx2)** элементов числами из текстового файла. Распечатать массив **A**. Далее каждый столбец массива **A** упорядочить по возрастанию обменом, затем заполнить по возрастанию одномерный массив **B** (длиной **2M** элементов) методом слияния из обоих столбцов массива **A**. Оба массива распечатать после сортировок.	11		Целые
30	Заполнить квадратный массив размерами **MxM**, где **M** – четное и равно 8, 10 или 12, логическими значениями из входного файла, распечатать. Поменять местами первый октант с третьим, а второй с четвертым. Октантом будем называть четверть массива, причем левая верхняя четверть – это 1-й октант, правая верхняя четверть – 2-ой и т.д. по часовой стрелке. Полученный массив распечатать.			Логич.
31	Массив размером M строк N столбцов заполняется числами из входного текстового файла. Исходный массив распечатывается. Столбец, где расположен минимальный элемент массива, упорядочить по возрастанию методом пересчета в одномерный массив. Созданный массив распечатать.	7	7	Веществ.

Лабораторная работа N 6

Программирование итерационных циклов (использование рекуррентных формул для расчета функций)

Задачи лабораторной работы

Вопросы, изучаемые в работе

- Изучение понятия рекуррентных соотношений.
- Применение алгоритмов вычисления элементарных функций как сумм бесконечного числа слагаемых.
- Продолжение изучения организации итеративных циклов с помощью операторов условных переходов, операторов с пост- и предусловием.
- Вывод результатов в табличной форме (с использованием форматов).

Задание (общее ко всем вариантам)

Для всех значений `X`, задаваемых параметрами из набора:

начальное значение `A`, конечное значение `B`, шаг `Dx`, количество шагов `Nx`, ограничение на число итераций `Nmax`, вычислить функцию `F(x)` как сумму `S` бесконечного числа слагаемых, заканчивая суммирование для каждого значения *x* при достижении заданной точности `EPS`.

Вывести в выводной текстовый файл исходные данные `A,B,Dx,EPS` и результаты расчета: значения `x,F(x),S` и количество слагаемых `K`, обеспечивающих требуемую точность. Результаты выводить в форме таблицы с рамками.

Оформить отчет по лабораторной работе.

Требования к программе

- Программа должна содержать комментарий с указанием названия работы, № варианта, фамилии студента и № группы.
- Значения, отмеченные в таблице вариантов символом "*" в программе вводятся с клавиатуры. Остальные исходные данные задаются типизированными константами.
- Количество слагаемых при вычислении суммы – не более ста.
- Если вариант задания содержит `Nmax` в качестве исходных данных, внешний цикл по `X` будет тоже итеративным и значение `Nx` в этом случае не рассчитывается. Цикл можно делать как с пред- так и с по-

стусловием, с окончанием по выполнению условия `(K > Nmax) or (i > 30)` { второе условие – ограничение размера таблицы }.

- В варианте №3 `Arcsin` выразить через другие функции (см. в лаб. 1) `Arcsin(x)=Arctan(x/sqrt(1-x*x))` .
- В варианте №15 учесть, что `sin(X)/X` при `X=0` равен 1.
- В варианте №21 учесть, что `cos(X)/X` при `X=0` не существует, поэтому такое значение X в таблице пропускать.

Содержание программы

- Заголовок программы с комментарием;
- Описание переменных;
- Описание инициализированных переменных;
- Ввод с клавиатуры исходных данных;
- Вычисление (если необходимо) вспомогательных значений;
- Вывод исходных данных в выводной файл;
- Формирование шапки таблицы;
- Задание начальных значений для величин, вычисляемых рекуррентными формулами;
- Цикл расчета и вывода в текстовый файл строк таблицы;
- Закрытие выводного файла.

Общие пояснения

Для приближенных вычислений значений некоторых функций используется метод представления этих функций в виде ряда бесконечного числа слагаемых.

Если неизвестная функция разложена в ряд

$F(x) = C_1x_1 + C_2x_2 + C_3x_3 + \dots + C_nx_n + \dots,$

то приближенно можно положить

$F(x) = F_n(x) = C_1x_1 + C_2x_2 + C_3x_3 + \dots + C_nx_n$,

и поправка на отбрасывание остальных членов ряда выразится остатком

$C_nx_n = C_{n+1}x_{n+1} + C_{n+2}x_{n+2} + \dots$

При достаточно большом n эта погрешность станет сколь угодно малой, так что $F_n(x)$ воспроизведет $F(x)$ с любой наперед заданной точностью.

Вопрос оценки остатка C_nx_n для получения требуемой точности вычисления требует особенного внимания.

Если рассматриваемый ряд оказывается знакопеременным, и притом с монотонно убывающими по абсолютной величине членами, то остаток имеет знак своего первого члена и по абсолютной величине меньше его.

В случае положительного ряда необходимо выполнить такую оценку при помощи методов математического анализа, кроме того, следует отметить, что далеко не всякий ряд, имеющий суммой интересующую нас функцию **F(x)**, пригоден для фактического вычисления этой функции (даже если его члены просты и оценка остатка производится легко). Вопрос – в быстроте сходимости, т.е. в быстроте приближения частичной суммы к функции **F(x)**. Для исследования сходимости ряда также рекомендуется применять методы математического анализа.

В вариантах задания к лабораторной работе диапазоны вычислений приближенных значений и задаваемая точность оценок выбраны из вышеприведенных соображений.

Разбор контрольного варианта

Задание

Рассчитать и построить таблицу точных и приближенных значений функции

$$F(X)=(1+X)^{-1/2}=1-\frac{1}{2}X+\frac{1*3}{2*4}X^2-\frac{1*3*5}{2*4*6}X^3+\frac{1*3*5*7}{2*4*6*8}X^4\ldots$$

начальное значение **X**: **A=-0.05**, конечное – **B=0.04**, шаг **X** – **Dx=0.01**(вводится с клавиатуры), требуемая точность **EPS = 1e-6** (вводится с клавиатуры), тип циклического оператора – **repeat** (постусловие).

Рассмотрение метода решения

Расчет суммы будем проводить по рекуррентной формуле: **S=S+C**, т.е. новое значение суммы **S** есть старое значение суммы **S** + очередное слагаемое **C**.

Чтобы получить рекуррентное соотношение для расчета слагаемых, поделим последующее слагаемое на предыдущее. Получим:

$$\frac{C2}{C1}=-\frac{1}{2}X,\quad \frac{C3}{C2}=-\frac{3}{4}X,\quad \frac{C4}{C3}=-\frac{5}{6}X,\quad \frac{C5}{C4}=-\frac{7}{8}X \quad \text{и т.д.}$$

Отсюда можно получить для расчета очередного слагаемого по предыдущему формулу:

```
C = - C*X*J/(J+1),
```

в которой **J** каждый раз увеличивается на два. Последняя рекуррентная формула (для **J**) : **J=J+2** .

Очевидно, что все эти операторы должны выполняться в цикле с заранее неизвестным числом шагов. Для начала расчета необходимо задать исходные значения для **J**, **C** и **S**. Вполне естественно начальными значениями **C** и **S** взять первое слагаемое, т.е. единицу, а величину **J** подобрать такой,

чтобы на первом шаге вычислить второе слагаемое по первому, т.е. J перед входом в цикл тоже сделать равным единице.

Окончание цикла осуществим, когда очередное слагаемое станет по модулю меньше EPS, но не более чем после сотого слагаемого. Последнее ограничение вводим для предотвращения возможности бесконечного цикла в случае возможных ошибок программирования.

В задании для организации итеративного цикла предложено использовать оператор repeat, поэтому условие выхода запишется:

```
...
until (abs(C) < EPS) or (K >100);
```

Полученные значения S, K, а также вычисленное выражение F(X) и величину аргумента X будем печатать с учетом размеров возможных значений в следующем порядке: X,F(x),S,K, разделяя их вертикальной чертой:

```
writeln('|',X:6:2,'|',1.0/Sqr(1.0+X):10:7,'|',
              S:10:7,'|',K:3,'|');
```

Значения не подписываем, так как расчет придется повторить несколько раз и насчитанные значения образуют таблицу, которую и озаглавим один раз до начала расчета по всем значениям X, например:

```
writeln('|  X  |  F(x)  |   S    | K |');
```

Так как в данном варианте нам даны начальное и конечное значения X и шаг его изменения, необходимо сначала вычислить, сколько будет значений X (и строк в таблице):

```
Nx := Trunc((B - A)/Dx) +1; {функция Trunc отбрасывает
                             дробную часть аргумента }
```

Добавляется единица, так как число точек всегда на один больше числа интервалов между ними. Это, кстати, следует помнить при вычислении Dx по Nx:

```
Dx := (B - A)/(Nx - 1);
```

Таблица идентификаторов

Таблица 21. Идентификаторы программы 31 варианта

Имя	Тип	Р-р (байт)	Назначение
Tabl_Of_Fx	Имя программы	-	Расчет таблицы значений функции
A	Веществ.константа	6	Начальное значение 0.05
B	"	6	Конечное значение 0.04
X	Вещественное	6	Значение аргумента на j-том шаге
Dx	"	"	Шаг аргумента
S	"	"	Сумма
C	"	"	Очередное слагаемое
EPS	"	"	Заданная точность

Имя	Тип	Р-р (байт)	Назначение
Nx	Целое	2	Количество шагов
J	"	"	Номер строки таблицы
K	"	"	Количество слагаемых
i	"	"	Счетчик цикла по X
fout	Послед. симв. файл	128	Для выходного файла
Vvod_Dx	Метка	–	Начало блока ввода шага
Vvod_EPS	Метка	–	Начало блока ввода точности

Алгоритм

Должен содержать следующие шаги:

1. Задание исходных данных в разделе констант (для **A** и **B**);
2. Ввод исходных данных (**Dx** и **EPS**);
3. Определение длины внешнего (по **X**) цикла;
4. Печать заголовка таблицы;
5. Внешний цикл (**for**) по **X**;
   1) вычисление **X**;
   2) задание начальных значений **J**, **C** и **S**;
   3) внутренний цикл вычисления **S**;
      а) расчет **C**;
      б) расчет **S**;
      в) расчет **J**;
      г) проверка условия окончания цикла;
   4) печать строки таблицы с результатами;
6. Завершение программы (печать нижней рамки таблицы, задержка).

Текст программы.

```
program Tabl_Of_Fx;
{
  Программа Лабораторной работы N 6  Вариант N 31.
 Использование рекуррентных формул в итеративных циклах.
   А.Я.Умненькая, ст. гр. Я-007
}
CONST {при описании переменным зададим исходные значения }
  A : real = -0.05;
  B : real = 0.04;
VAR
  X,Dx,S,C,EPS : real;
  Nx,J,K,i : integer; {I - для счетчика цикла по X }
  fout   : text;  { для выходного файла }
```

```
LABEL
  Vvod_Dx, Vvod_EPS;
BEGIN
  assign(fout,'Umnik6.res');
  rewrite(fout);
   { Ввод исходных данных (Dx и EPS) }
  Vvod_Dx:
  writeln ('Для X на отрезке[-0.05, 0.04]введите шаг счета');
  readln(Dx);
  if (Dx <0.002) or (Dx > 0.04) then
  begin
    writeln('Недопустимое значение!');
    goto Vvod_Dx;
  end;
 Vvod_EPS:writeln('введите минимальный размер слагаемого);
  readln(EPS);
  if (EPS < 1e-9) or (EPS > 1e-2) then
  begin
    writeln('Недопустимое значение!');
    goto Vvod_EPS;
  end;
{ Определение длины внешнего (по X) цикла }
  Nx := Trunc((B-A)/Dx) + 1;
{ печать исходных данных }
  writeln(fout,' Исходные данные');
  writeln(fout,' Интервал X: [',A:5:2,B:5:2,']);
  writeln(fout, Шаг X:',Dx:5:3, ', шагов:',Nx:3,',
          точность:',EPS:1);
{ Печать заголовка таблицы }
  writeln(fout,' Результаты расчетов':30);
  writeln(fout,'|   X   |     F(x)    |      S ',
                '     |Слагаемых|');
  writeln(fout,'|-------|-----------|-----------|',
                '---------|');
   { Внешний цикл (for) по X }
  for i:=1 to Nx do
  begin
{ вычисление X }
    X := A + Dx*(i-1);
    if X > B then X := B; { уточнение на конце интервала}
{ задание начальных значений J, C, S и числа слагаемых K}
    J := 1;
    C := 1;
    S := C;
```

```
    K := 1;
    Repeat  { внутренний цикл вычисления S }
      C := -C * X*J/(J+1);
      S := S + C;
      K := K + 1;
      J := J + 2;
    until (abs(C) < EPS) or (K >100);
{ печать строки таблицы с результатами }
  write(fout,'| ',X:5:2,' |', 1.0/Sqr(1.0+X):10:7,
            ' |');
  writeln(fout,S:10:7,' |',K:6,'     |');
 end;
{ Завершение программы (печать нижней рамки таблицы,
 закрытие файлов, задержка) }
 writeln(fout,'|--------------------------------',
                  '--------|');
 close(fout);
 readln;
END.
```

Результаты расчета

Файл **UMNIK6.RES** будет в этом случае содержать:

```
 Исходные данные
 Интервал X: [-0.05 0.04],
Шаг X:0.010, шагов: 9, точность: 1.0E-0006
            Результаты расчетов
|   X   |     F(x)    |      S      |Слагаемых|
|-------|-------------|-------------|---------|
| -0.05 | 1.1080332 | 1.0259783 |     6     |
| -0.04 | 1.0850694 | 1.0206207 |     5     |
| -0.03 | 1.0628122 | 1.0153462 |     5     |
| -0.02 | 1.0412328 | 1.0101525 |     5     |
| -0.01 | 1.0203041 | 1.0050378 |     4     |
|  0.00 | 1.0000000 | 1.0000000 |     2     |
|  0.01 | 0.9802960 | 0.9950372 |     4     |
|  0.02 | 0.9611688 | 0.9901475 |     5     |
|  0.03 | 0.9425959 | 0.9853293 |     5     |
|-----------------------------------------|
```

Варианты заданий

Таблица 22. Варианты заданий лабораторной работы N6

№ вар.	F(x) (вид разложения в сумму см.табл.23)	начальное значение A	конечное значение B	шаг Dx	число		точность вычисления EPS *	Тип цикла
					шагов Nx	итераций Nmax		
1	$\ln\left(\frac{1}{0.8-x}\right)$	-0.05	0.05	0.01*		-	1e-4	**if**
2	$\frac{1}{(1-x)^2}$	-0.09	0.09		10*		1e-6	**repeat**
3	$\arcsin(x)$	0		0.05*	11		1e-4	**while**
4	$\ln\left(\frac{1+x}{1-x}\right)$		0.08	0.01*	10		1e-5	**if**
5	e^{-x^2}	-1		0.5	12*		1e-6	**repeat**
6	$\ln(x)$	-2	0.5		9*		1e-5	**while**
7	$\ln(x+\sqrt{x^2+1})$	-0.5		0.1*	11		1e-5	**if**
8	$\frac{1}{1+x}$	0	0.08	0.01*			1e-4	**repeat**
9	$\frac{1}{(1+x)^2}$	-0.2	0.1	0.01*			1e-5	**while**
10	$\operatorname{arctg}(x)$	-0.5		0.1	10*		1e-6	**if**
11	$\frac{e^x+e^{-x}}{2}$	-0.5	0.5		21*		1e-6	**repeat**
12	$\ln(1-x)$	-0.1	0.2	0.03*			1e-4	**while**
13	$\ln(1+0.1x^2)$	-0.5	0.5		21*		1e-5	**if**
14	$\frac{1}{1-x^2}$	0		0.02	11*		1e-3	**repeat**

№ вар.	F(x) (вид разложения в сумму см.табл.23)	начальное значение A	конечное значение B	шаг Dx	число		точность вычисления EPS *	Тип цикла
					шагов Nx	итераций Nmax		
15	$\frac{\sin(x)}{x}$	-5	5	1*			1e-4	**while**
16	$(1+x)^{-3}$	-0.1	-0.2		13*		1e-5	**if**
17	$\sqrt{(1+x}$	0.1	0.3	0.02*			1e-5	**repeat**
18	$\frac{e^{x}-e^{-x}}{2}$	1	10*	1*			1e-6	**while**
19	$\ln(1+x)$	0	0.5		11*		1e-4	**if**
20	3^{x}	-0.3	0.1	0.02*			1e-5	**repeat**
21	$\frac{\cos(x)}{x}$	-3		1*	10*		1e-5	**while**
22	$\frac{2x}{1+x^{2}}$	-0.4		0.04	17*		1e-6	**if**
23	$\frac{1}{\sqrt{1+x^{2}}}$	-0.1	0.2	0.02*			1e-5	**repeat**
24	$x^{1/3}$	0.1	0.55	0.05*			1e-5	**while**
25	$\frac{1}{2}\ln\left(\frac{1+x}{1-x}\right)$	0.2	0.4		11*		1e-6	**if**
26	$\sin(x)$	0		0.5		20*	1e-5	**repeat**
27	e^{2x}	0.5		0.1		20*	1e-5	**while**
28	$\cos(x)$	1		1*		25*	1e-4	**if**
29	e^{x}	0		0.3		15*	1e-4	**repeat**
30	e^{-x}	0		0.4*		20*	1e-5	**while**

№ вар.	F(x) (вид разложения в сумму см.табл.23)	начальное значение A	конечное значение B	шаг Dx	число		точность вычисления EPS *	Тип цикла
					шагов Nx	итераций Nmax		
31	$\frac{1}{\sqrt{1+x}}$	-0.05	0.04	0.1*			1e-6	**repeat**

Таблица 23. Виды представления функций

№ вар.	Вид представления функции как суммы ряда слагаемых
1	$\ln\left(\frac{1}{0.8-x}\right)=\frac{x+0.2}{1}+\frac{(x+0.2)^2}{2}+\frac{(x+0.2)^3}{3}+\frac{(x+0.2)^4}{4}+\ldots$
2	$\frac{1}{(1-x)^2}=1+2x+3x^2+4x^3+5x^4+\ldots$
3	$\arcsin(x)=x+\frac{1}{2}\frac{x^3}{3}+\frac{1*3}{2*4}\frac{x^5}{5}+\frac{1*3*5}{2*4*6}\frac{x^7}{7}+\frac{1*3*5*7}{2*4*6*8}\frac{x^9}{9}+\ldots$
4	$\ln\left(\frac{1+x}{1-x}\right)=2x\left(1+\frac{1}{3}x^2+\frac{1}{5}x^4+\frac{1}{7}x^6+\ldots\right)$
5	$e^{-x^2}=1-\frac{x^2}{1!}+\frac{x^4}{2!}-\frac{x^6}{3!}+\ldots$
6	$\ln(x)=\frac{x-1}{1}-\frac{(x-1)^2}{2}+\frac{(x-1)^3}{3}-\frac{(x-1)^4}{4}+\ldots$
7	$\operatorname{arsh}(x)=x-\frac{1}{2}\frac{x^3}{3}+\frac{1*3}{2*4}\frac{x^5}{5}-\frac{1*3*5}{2*4*6}\frac{x^7}{7}+\frac{1*3*5*7}{2*4*6*8}\frac{x^9}{9}\ldots$
8	$\frac{1}{1+x}=1-x+x^2-x^3+x^4-\ldots$
9	$\frac{1}{(1+x)^2}=1-2x+3x^2-4x^3+5x^4-\ldots$
10	$\operatorname{arctg}(x)=x-\frac{x^3}{3}+\frac{x^5}{5}-\frac{x^7}{7}+\frac{x^9}{9}-\ldots$
11	$\operatorname{ch}(x)=\frac{e^x+e^{-x}}{2}=1+\frac{x^2}{2!}+\frac{x^4}{4!}+\frac{x^6}{6}+\ldots$
12	$\ln(1-x)=-x-\frac{x^2}{2}-\frac{x^3}{3}-\frac{x^4}{4}-\frac{x^5}{5}-\ldots$

№ вар.	Вид представления функции как суммы ряда слагаемых
13	$\ln(1+0.1x^2)=\frac{0.1*x^2}{1}-\frac{0.01*x^4}{2}+\frac{0.001*x^6}{3}-\frac{0.0001*x^8}{4}+\ldots$
14	$\frac{1}{1-x^2}=1+x^2+x^4+x^6+x^8+\ldots$
15	$\frac{\sin(x)}{x}=1-\frac{x^2}{3!}+\frac{x^4}{5!}-\frac{x^6}{7!}+\ldots$
16	$(1+x)^{-3}=1-\frac{2*3}{2}x+\frac{3*4}{2}x^2-\frac{4*5}{2}x^3+\frac{5*6}{2}x^4-\ldots$
17	$\sqrt{1+x}=1+\frac{1}{2}x-\frac{1}{2*4}x^2+\frac{1*3}{2*4*6}x^3-\frac{1*3*5}{2*4*6*8}x^4+\ldots$
18	$\mathrm{sh}(x)=\frac{e^x-e^{-x}}{2}=\frac{x}{1!}+\frac{x^3}{3!}+\frac{x^5}{5!}+\frac{x^7}{7!}+\ldots$
19	$\ln(1+x)=x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\frac{x^5}{5}-\ldots$
20	$3^x=1+\frac{x*\ln 3}{1!}+\frac{(x*\ln 3)^2}{2!}+\frac{(x*\ln 3)^3}{3!}+\ldots$
21	$\frac{\cos(x)}{x}=\frac{1}{x}-\frac{x}{2!}+\frac{x^3}{4!}-\frac{x^5}{6!}+\ldots$
22	$\frac{2x}{1+x^2}=2x\left(1-x^2+x^4-x^6+x^8-\ldots.\right)$
23	$\frac{1}{\sqrt{1+x^2}}=1-\frac{1}{2}x^2+\frac{1*3}{2*4}x^4-\frac{1*3*5}{2*4*6}x^6+\frac{1*3*5*7}{2*4*6*8}x^8-\ldots$
24	$x^{1/3}=1+\frac{1}{1!}\frac{1}{3}(x-1)+\frac{1}{2!}\frac{2}{3^2}(x-1)^2+\frac{1}{3!}\frac{2*5}{3^3}(x-1)^3+\frac{1}{4!}\frac{2*5*8}{3^4}(x-1)^4+\ldots$
25	$\mathrm{arth}(x)=\frac{1}{2}\ln\left(\frac{1+x}{1-x}\right)=x+\frac{x^3}{3}+\frac{x^5}{5}+\frac{x^7}{7}+\ldots$
26	$\sin(x)=\frac{x}{1!}-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots$
27	$e^{2x}=1+\frac{2x}{1!}+\frac{4x^2}{2!}+\frac{8x^3}{3!}+\frac{16x^4}{4!}+\ldots$
28	$\cos(x)=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots$

№ вар.	Вид представления функции как суммы ряда слагаемых
29	$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots$
30	$e^{-x} = 1 - \frac{x}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \ldots$
31	$\frac{1}{\sqrt{1+x}} = 1 - \frac{1}{2}x + \frac{1*3}{2*4}x^2 - \frac{1*3*5}{2*4*6}x^3 + \frac{1*3*5*7}{2*4*6*8}x^4 - \ldots$

Лабораторная работа N 7
Программирование процедур на Паскале

Задачи лабораторной работы

Вопросы, изучаемые в работе

- Изучение строения и использования процедур, механизма параметров процедур.
- Изучение строения и использования функций пользователя.
- Продолжение изучения основных алгоритмов сортировки.
- Освоение работы с файлами различных типов.

Задание (общее ко всем вариантам).

Написать программу работы с массивом с использованием процедур и функций распечатки и частичной обработки массива.

При написании подпрограмм не использовать глобальные переменные, кроме имен файлов. Все обмены данными между подпрограммами и вызывающей программой выполнять через параметры подпрограмм.

Исходные данные читать из существующего текстового или двоичного файла. Результаты расчета выводить в форматном виде в выходной текстовый файл.

Оформить отчет по лабораторной работе аналогично оформлению предыдущих работ.

Требования к программе и отчету по работе

- В таблице распределения памяти привести имена, используемые как в основной программе, так и в подпрограммах, в том числе и формальные параметры процедур и функций.
- Алгоритмы каждой подпрограммы и основной программы выполнить отдельно.
- Все значения, на которые по смыслу накладываются ограничения, должны при вводе проверяться.
- Все выводимые данные должны подписываться.

Содержание программы.

- Заголовок программы с комментарием;
- Описание типов массивов;

- Описание переменных основной программы;
- Описание процедур и функций;
- Ввод необходимых исходных данных с клавиатуры;
- Открытие входных и выводного файлов;
- Ввод необходимых исходных данных из файла;
- Вывод исходных данных, в том числе массива в выводной файл. При выводе массива использовать созданную процедуру;
- Обработка массива в соответствии с заданием с применением разработанной процедуры или функции;
- Вывод результатов обработки в выводной файл, возможно с применением процедуры;
- Закрытие всех файлов.

Пояснения к лабораторной работе

Общие пояснения к использованию процедур и функций.

Процедуры и функции пользователя являются двумя видами подпрограмм, которые необходимо описать в разделе описаний основной программы (или предварительно поместить в личную библиотеку) и вызывать по имени в нужных местах вызывающей программы.

И процедура, и функция состоят из заголовка, раздела описаний и выполняемого блока. Их описания должны находиться после других операторов описаний основной (вызывающей) программы, перед ее выполняемым блоком. В отличие от основной программы, заголовки процедуры и функции обязательны, и имеют вид:

`PROCEDURE` <имя процедуры>(<список параметров>); – для процедуры и

`FUNCTION` <имя функции>(<список аргументов>):<тип результата>; – для функции.

Список аргументов это, по существу, список входных параметров.

Раздел описаний процедуры или функции строится как раздел описаний основной программы, с учетом того, что все описанные переменные существуют только во время выполнения процедуры (функции) в виде локальных переменных и никак не связаны с переменными вызывающей программы.

Следует иметь в виду, что если во внешней программе переменная описана, а в процедуре – нет, ее использование в теле процедуры означает работу с переменной внешней программы (так называемые глобальные переменные).

Выполняемый блок (который должен быть заключен в скобки `Begin .. End`) заканчивается символом ";", а не точкой. В функции, в отличие от

процедуры, в выполняемом блоке имени функции обязательно должно быть присвоено значение, которое и будет являться ее значением. Такое присваивание может встречаться несколько раз, конечным значением будет результат последнего присваивания. Внутри выражений тела функции ее имя встречаться не должно, если это не специальная рекурсивная функция.

Обращение (вызов) процедуры выполняется написанием имени процедуры в форме отдельного оператора. Обращение к функции выполняется только внутри выражения, входящего в состав оператора, аналогично обращению к стандартным (встроенным) функциям Турбо-Паскаля.

Механизм параметров

В список параметров включаются исходные данные для работы процедуры/функции (входные параметры) и, если надо, указания, куда поместить результаты работы процедуры (выходные параметры).

Существует два способа передачи данных через список параметров из вызывающей программы в процедуру. Первый способ заключается в том, что процедуре передается некоторое данное в виде его значения, например, если данное числового типа, передается само число. Второй способ заключается в том, что передается не значение данного, а адрес ячейки, где это данное находится (говорят, что передается имя переменной). Таким образом, параметры могут передаваться по значению и по имени. В каких случаях какой способ следует использовать? Обычно входные параметры передаются в процедуру по значению, а выходные – всегда по имени.

Есть некоторые особенности при передаче массивов (и других составных данных большого размера) в качестве фактических параметров. Даже если массив представляет собой входные данные и не меняется в процессе выполнения процедуры, его обычно передают по имени, так как это требует передачи только адреса начала массива, а не копии всего массива, как это потребовалось бы при передаче массива по значению. Кроме того, массив основной программы (фактический параметр) и массив в списке формальных параметров должны иметь один и тот же тип – т.е. должен использоваться один и тот же явный описатель.

Отметим, что когда параметры передаются по имени, они занимают в памяти по 4 байта. Подробнее об этом смотри лабораторную работу № 9.

Примеры написания списков формальных и фактических параметров:

```
..(A,B,C:real;VAR X1,X2:real;VAR N:integer); – формальные,
..(0.762,Alfa,C[3]-1.2,X,Y,Num); соответствующие фактические.
```

Для передачи массива в процедуру:

```
TYPE
  Vect = array[1..10] of real;
  Mas5x8 = array[1..5,1..8] of integer;
. . .
VAR
  C1,C2: Vect;
  A:  Mas5x8;
  Metod: Integer;
. . .
PROCEDURE GetMatr(M:integer; Var X,Y:Vect;
                  Var Z:Mas5x8;...);
VAR A1,C1:integer;
. . .
Begin
  <Операторы тела процедуры >
End;
FUNCTION MinValueMatr( Var Z:Mas5x8; M,N:integer)
                       :Integer;
VAR MinZ, I, j :integer;
Begin
  <Операторы тела функции >
  MinValueMatr:= MinZ;
End;
BEGIN
  <Операторы основной программы >
. . .
  GetMatr(0, C2, C1, A,...); {обращение к процедуре}
. . .
  If MinValueMatr(A,5,7) div 2 < 3 then {обращение к
функции}
. . .
END.
```

Разбор контрольного варианта

Задание

Таблица 24. Данные к заданию 31 варианта

№ вар.	Программа	Процедуры (Функции)	M<=	N<=	Файл с данными
31	Прямоугольный массив заполнить числами из файла, начиная с 33 числа. С помощью функции в исходном массиве сделать элементы последнего столбца равными сумме всех четных элементов соответствующей строки.	1)Исходный и полученный массивы печатать процедурой. 2)Построить функцию, которая возвращает сумму четных по значению элементов заказанной строки прямоугольного массива.	13	10	DATI.BIN

Таблица идентификаторов

Составляется как для основной программы, так и для каждой подпрограммы пользователя.

Таблица 25. Идентификаторы программы 31 варианта

Имя	Тип	Р-р (байт)	Назначение
Основная программа			
KontMass	Имя программы	-	Обработка массива
Massiv	Описатель типа	-	Описатель целочисл. массивов
Stroka	Описатель типа	-	Описатель строки до 30 символов
M	Целое	2	Количество строк массива
N	Целое	2	Количество столбцов массива
i	Целое	2	Номер текущей строки массива
j	Целое	2	Номер текущего столбца массива
Fin	Двоичный файл прямого доступа	2	Файл с исходн. цельми числами
Fout	Последовательный символьный файл	128	Файл с результатами работы
Ouest1	Инициализированная строка	31	Запрос числа строк массива
Ouest2	Инициализированная строка	31	Запрос числа столбцов массива

Имя	Тип	Р-р (байт)	Назначение
A	Целочисленный массив	260	Обрабатываемый массив
InpMN	Имя процедуры	-	Ввод размеров массива
PrintMas	Имя процедуры	-	Вывод массива в файл протокола
DATI.BIN	Строка – константа	9	Имя набора данных с числами
Umnik_7.res	Строка – константа	12	Имя набора данных с протоколом
Sum	Имя функции	-	Суммирование четных значений
InpMN — Процедура ввода размеров массива			
Txt	Строка	31	Формальный параметр – строка запроса
Kol	Адрес целочисленной переменной	4	Формальный параметр – имя переменной для результата ввода
MaxK	Целое	2	Формальный параметр – предельное возможное значение
PrintMas — Процедура вывода массива в файл протокола (распечатки массива)			
Txt	Строка	256	Формальный параметр – текст заголовка
Mas	Адрес массива	4	Формальный параметр – имя выводимого массива
NStr	Целое	2	Формальный параметр – число строк массива
NKol	Целое	2	Формальный параметр – число столбцов массива
i	Целое	2	Номер строки
j	Целое	2	Номер столбца
Sum – Функция вычисления суммы четных элементов заданной строки массива			
Mas	Адрес массива	4	Формальный параметр – имя исходного массива
N	Целое	2	Формальный параметр – число столбцов массива
St	Целое	2	Формальный параметр – номер обрабатываемой строки
J	Целое	2	Номер элемента
S	Целое	2	Сумма
Odd	Стандартная логическая функция	-	Проверка нечетности аргумента

Блок-схема алгоритма

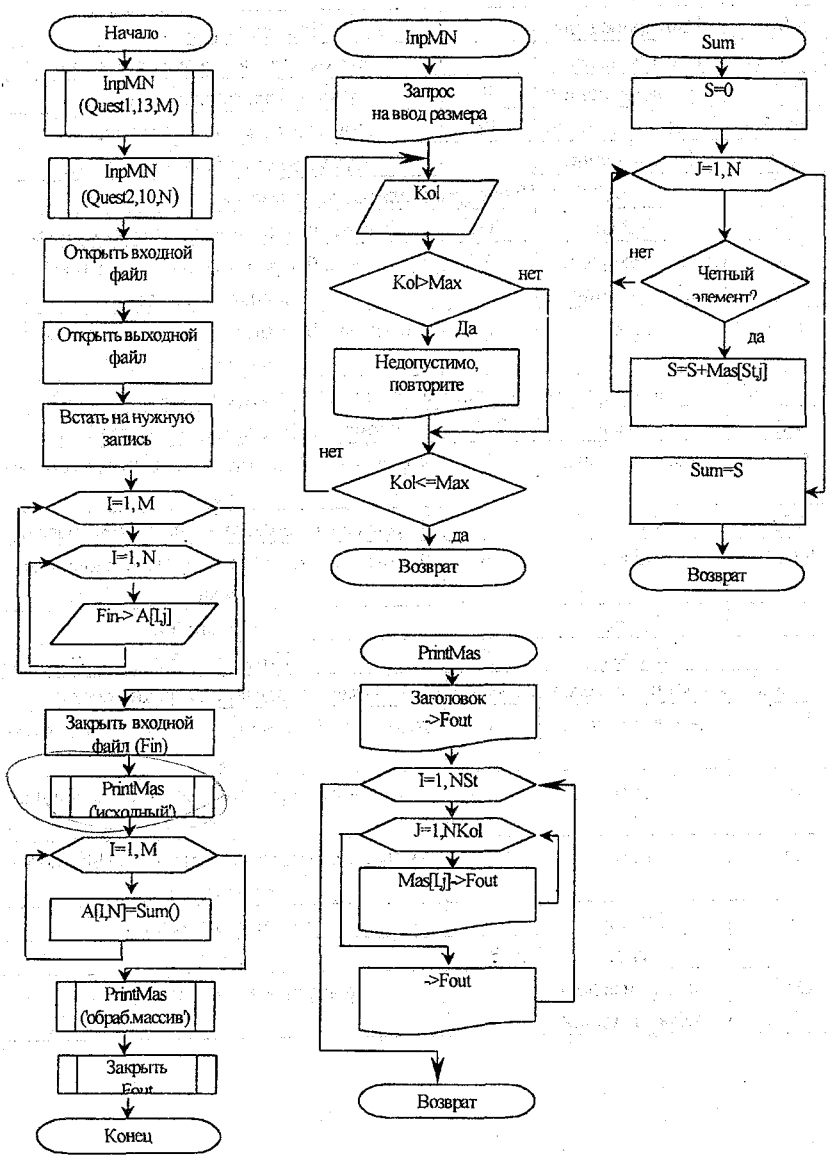


Рисунок 7. Алгоритм 31-го варианта

Текст программы.

```
PROGRAM KorrMas;
{   Лабораторная работа N 7     Вариант N 31
          Обработка массива
  А.Я.Умненькая, ст. гр. Я-007 }
TYPE
  Massiv = array[1..13,1..10] of integer;
  Stroka = string[30];
CONST Quest1: Stroka='Задайте число строк массива';
   Quest2: Stroka='Задайте число столбцов массива';
VAR A :Massiv;
  M,N,I,j : integer;
  Fin : file of integer;
  Fout : file of text;
{ Процедура ввода размеров массива }
PROCEDURE InpMN(Txt:Stroka; MaxK:integer;
                Var Kol:integer);
Begin
  WriteLn; WriteLn(Txt);
  Repeat
    Readln(Kol);
    If Kol>MaxKol then
     Writeln('Можно не более ',MaxK,'задайте снова');
  until Kol<=MaxK;
End;
{ Процедура вывода массива в файл протокола }
PROCEDURE PrintMas(Txt: string; Var Mas: Massiv;
                   NStr,NKol: integer);
VAR i,j : integer;
Begin
  Writeln(Fout,Txt);
  For i:= 1 to NStr do
```

```
  begin
    For j:= 1 to Nkol do Write(Fout,Mas[I,j]:7);
    WriteLn(Fout);
  end;
End;
{ Функция вычисления суммы четных элементов заданной
строки массива }
FUNCTION Sum(Var Mas: Massiv; N,St: integer)
              :integer;
VAR j,S : integer;
Begin
  S:=0;
   For j:=1 to N do
    If not Odd(Mas[St,j]) then S:=S+Mas[St,j];
  Sum:=S;
End;
BEGIN    { Основная программа }
  InpMn(Quest1,13,M);
  InpMn(Quest2,10,N);
  Assign(Fout,'Umnik7.res');
  ReWrite(Fout);
  WriteLn(Fout,'Сумма четных элем. в последний столбец');
  Assign(Fin,'D:\LAB1\DATI.BIN');
  ReSet(Fin);
  Seek(Fin,32);
  for i:=1 to M do
    for j:=1 to N do Read(Fin,A[I,j]);
  Close(Fin);
  PrintMas (' Исходный массив', A,M,N);
  for I:=1 to M do A[I,N]:=Sum(A,N,I);
  PrintMas (' Обработанный массив', A,M,N);
  Close(Fout);
END.
```

Содержимое набора данных UMNIK7.RES:

```
Сумма четных элем. в последний столбец
 Исходный массив
    9     8     0    -7     0    -5     7    -5     5
    6     9     9     2    -2    -8     3    -8    -9
    8    -4    -4     1     3     6     4     0    -5
    4     1    -2     4     2     1    -2    -6    -5
   -1     5     0     9     4    -3    -6  -691   885
 -709   800   382  -396  -140  -841   923   368  -686
  737   624   169  -610   458  -188  -423   126   355
 Обработанный массив
    9     8     0    -7     0    -5     7    -5     8
    6     9     9     2    -2    -8     3    -8   -10
    8    -4    -4     1     3     6     4     0    10
    4     1    -2     4     2     1    -2    -6     0
   -1     5     0     9     4    -3    -6  -691    -2
 -709   800   382  -396  -140  -841   923   368   328
  737   624   169  -610   458  -188  -423   126   410
```

Варианты заданий

Таблица 26. Варианты заданий лабораторной работы №7

№ вар.	Программа	Процедуры (Функции)	M<=	N<=	Файл с данными
1	В прямоугольном массиве с помощью функции найти элемент, у которого самая большая сумма "соседей". Найденный элемент и его индексы напечатать. Исходные числа из файла читать, начиная с 27 числа.	1) Распечатать исходный массив с помощью процедуры. 2) Построить функцию, которая для произвольного внутреннего элемента передаваемого ей массива находит сумму всех соседних (8 штук).	10	7	DATF.TXT
2	В прямоугольном массиве с помощью процедуры убрать все строки, у которых оба конца – с четными значениями. Исходные данные из файла читать, начиная с 44 числа.	1) Печатать двумерный массив до и после корректировки с помощью процедуры. 2) Построить процедуру, которая удаляет (со сдвигом вверх всех нижележащих строк) заказанную строку передаваемого ей прямоугольного массива	10	12	DATI.TXT
3	В прямоугольном массиве с помощью процедуры сделать все строки упорядоченными по убыванию. Исходные числа из файла читать, начиная с "**K**"-го числа, где **K** вводить с клавиатуры.	1) Печатать массив до и после сортировки с помощью процедуры. 2) Построить процедуру, которая в заказанной строке передаваемого ей прямоугольного массива упорядочивает элементы по возрастанию методом обмена.	11	9	DATI.BIN
4	Квадратный массив заполнить данными из файла, начиная с 21 числа. С помощью функции в исходном массиве поменять элементы главной диагонали с максимальными значениями в строке.	1) Исходный и полученный массивы печатать процедурой. 2) Построить функцию, которая возвращает номер максимального элемента заказанной строки прямоугольного массива.	7	7	DATF.TXT

5	Одномерный символьный массив длиной 300 элементов заполнить символами из файла, начиная с начала. С помощью процедуры заполнить новый одномерный символьный массив символами из исходного массива без повторения символов.	1) Построить процедуру, которая в исходном массиве заменяет пробелом все вхождения первого встретившегося символа (не пробела) и возвращает этот стертый символ в вызывающую программу. 2) Печатать каждый массив с помощью процедуры.	300		SIMBOL.TXT
6	В квадратном массиве, используя процедуру сортировки, сделать все строки упорядоченными по возрастанию. Исходные числа из файла читать, начиная с 20 числа.	1) Печатать массив до и после сортировки с помощью процедуры. 2) Построить процедуру, которая в заказанной строке передаваемого ей квадратного массива упорядочивает элементы по возрастанию методом выбора.	7	7	DATF.TXT
7	Прочитать в одномерный строковый массив (строки по 80 символов, количество строк не более 50), текст составленной Паскаль-программы. Используя функцию, подсчитать количество операторов в тексте программы. Перед каждым "end" должен стоять символ ';'.	1) Составить процедуру подсчета и печати количества групп операторов (заключенных в скобки "begin ..end") в исходном массиве строк. 2) Составить функцию, возвращающую количество символов ';' в строке.	50		*.PAS
8	В прямоугольном массиве, найдя с помощью процедуры место минимального элемента, удалить строку и столбец, проходящие через него. Исходные числа из файла читать, начиная с 99 числа.	1) Печатать массив до и после корректировки с помощью процедуры. 2) Построить процедуру, которая находит в передаваемом массиве номер строки и номер столбца самого маленького элемента.	14	11	DATI.BIN

9	Заполнить одномерный массив из **M** элементов. Затем с помощью процедуры заполнить одномерный целочисленный массив длиной 10 элементов, в котором i-й элемент указывает, сколько чисел исходного массива лежит в диапазоне от (i–1)*10 до i*10. Исходные числа из файла читать, начиная с 6 числа.	1) Исходный и полученный массивы печатать процедурой. 2) Построить процедуру, которая подсчитывает, сколько чисел в передаваемом ей одномерном целочисленном массиве попадает в заказанный диапазон.	100..200		DATI.BIN
10	Одномерный массив строк длиной по 80 символов заполнить из файла. Используя функцию, найти, какая русская буква (не различая прописные и строчные) встречается чаще. **К** функции обращаться, передавая ей строчную и прописную буквы и символьную строку массива.	1) Распечатать исходный массив процедурой. 2) Составить функцию, возвращающую суммарное количество вхождений двух заданных символов в заданной строке длиной 80 символов.	10		RUS.TXT
11	Прямоугольный символьный массив размером 10x30 заполнить из файла начиная с 29 символа. С помощью процедуры заполнить два одномерных массива длиной 10 элементов каждый. В первом массиве – наиболее частые символы, а во втором – их количество в каждой строке исходного массива. Полученные массивы также распечатать.	1) Исходный массив печатать процедурой. 2) Построить процедуру, которая в заказанной строке произвольного символьного массива (размерами до 10x30) находит, какой символ встречается чаще всего и сколько раз.			SIMBOL.TXT
12	Заполнить исходную квадратную матрицу **A** числами из файла, начиная с 39-го числа. Используя процедуры, вычислить матрицу **C=A*B**, где **B** – транспонированная матрица **A**.	1) Печатать все массивы с помощью процедуры. 2) Построить процедуру транспонирования квадратной матрицы (размерами до 7x7). 3) Построить процедуру, перемножения двух квадратных матриц размерами до 7x7 (см. пояснения к 5 лабораторной).	7	7	DATF.TXT

13	Прочитать в одномерный строковый массив (строки по 80 символов), количество строк не более 50, текст составленной Паскаль-программы. Используя функцию, подсчитать количество управляющих операторов в этой программе. Распечатать название и количество операторов каждого вида. Перед подсчетом перевести все латинские символы в прописные.	1) Составить процедуру печати двух массивов в форме: название оператора – количество вхождений. В процедуру передать строчный массив названий и числовой – количеств. 2) Составить функцию, возвращающую количество вхождений заданной строки в массиве из 50 строк по 80 символов. (Функцию использовать для строк: 'if', 'goto', 'for', 'case', 'while', 'repeat'.)	50		*.PAS
14	Используя процедуру заполнить одномерный целочисленный массив длиной 15 элементов, в котором каждый элемент указывает, сколько в исходном одномерном символьном массиве символов "а", "б" и т.д. до "п". Исходный массив заполнять из файла, пропустив 10 символов	1) Исходный массив печатать процедурой. 2) Построить процедуру, которая подсчитывает, сколько раз в передаваемом ей одномерном символьном массиве длиной N элементов встречается заказанный символ.	200		SIMBOL. TXT
15	В прямоугольном массиве с помощью функции найти элемент, у которого самая маленькая сумма соседних вместе с ним. Полученный элемент напечатать. Исходные числа из файла читать, начиная с 29 числа.	1) Печатать исходный массив с помощью процедуры. 2) Построить функцию, которая для произвольного внутреннего элемента передаваемого ей массива (размерами до 11x7), находит сумму элемента и всех соседних с ним (всего 9 штук).	11	7	DATF.BIN
16	Прямоугольный символьный массив размером 20x30 заполнить символами из файла, пропустив 49 символов. С помощью процедуры в массиве найти и заменить на '*' элемент с самой маленькой суммой кодов соседних символов.	1) Построить процедуру, которая для заказанного внутреннего элемента массива подсчитывает сумму кодов восьми соседних элементов. 2) Печатать каждый массив с помощью процедуры.	20	30	SIMBOL. TXT

17	Прямоугольный массив заполнить из файла каждым третьим числом. Используя логическую функцию распечатать для каждой строки исходного массива фразу "четных чисел больше" или "четных чисел не больше".	1) Распечатать исходный массив процедурой. 2) Составить функцию, возвращающую значение TRUE, если в заданной строке массива больше четных чисел и FALSE в противном случае.			DATI.BIN
18	В прямоугольном массиве с помощью процедуры сделать все строки упорядоченными по возрастанию. Исходные числа из файла читать, начиная с "**K**"-го числа, где **K**<10 вводить с клавиатуры.	1) Печатать массив до и после сортировки с помощью процедуры. 2) Построить процедуру, которая в заказанной строке передаваемого ей прямоугольного массива упорядочивает элементы по убыванию методом выбора.	7	8	DATF.TXT
19	Массив из M строк и пяти столбцов частично (первые три столбца) заполнить числами из файла, начиная с 33-го числа файла. Рассматривая первые три числа в каждой строке матрицы в качестве коэффициентов при нулевой, первой и второй степени X, заполнить четвертый и пятый элементы значениями корней уравнения с помощью процедуры.	1) Построить процедуру, вычис-ляющую вещественные корни квадратного уравнения по трем передаваемым ей коэффициентам (если отсутствует один или оба вещественных корня, процедура должна возвращать значения корней, равные нулю). 2) Заполненный массив распечатать процедурой.	10	-	DATF.BIN
20	С помощью процедуры в исходном массиве сделать все строки упорядоченными по возрастанию. Перед обращением к процедуре копировать строку в одномерный массив, после – копировать обратно. Исходный массив заполнить числами из файла, начиная с 18-го.	1) Печатать массив до и после сортировки с помощью процедуры. 2) Построить процедуру, которая в одномерном целочисленном массиве упорядочивает элементы по возрастанию методом выбора.	9	10	DATI.BIN

21	Прямоугольный массив заполнить числами из файла, начиная с четвертого числа. С помощью функции в исходном массиве сделать элементы первой строки равными максимальному элементу соответствующего столбца.	1) Исходный и полученный массивы печатать процедурой. 2) Построить функцию, которая возвращает максимальное значение в заказанном столбце прямоугольного массива.	12	10	DATI.TXT
22	В квадратном массиве, используя процедуру сортировки, сделать все столбцы упорядоченными по возрастанию (перед обращением к процедуре копировать столбец в одномерный массив, после – копировать обратно). Исходные числа из файла читать, начиная с 18-го.	1) Печатать двумерный массив до и после сортировки с помощью процедуры. 2) Построить процедуру, которая в одномерном вещественном массиве упорядочивает элементы по убыванию методом выбора.	7	7	DATF.BIN
23	Одномерный символьный массив длиной 300 элементов заполнить из файла, начиная с начала, и распечатать по 60 символов в строке. С помощью процедур сосчитать и распечатать, сколько трех-, шести- и девятисимвольных слов в массиве.	1) Построить процедуру, которая в передаваемом ей исходном массиве заменяет все разделители (символы: ' ', '.',',','-','(',')','"') на единый – точку. 2) Построить процедуру, которая в передаваемом ей массиве определяет количество слов заданной длины (слово – символы между двумя точками.)	300		SIMBOL.TXT
24	В прямоугольном массиве, используя процедуру, сделать все столбцы упорядоченными по возрастанию. Исходные числа из файла читать, начиная с 12-го.	1) Печатать массив до и после сортировки с помощью процедуры. 2) Построить процедуру, которая в заказанном столбце произвольного прямоугольного массива упорядочивает элементы по возрастанию методом обмена.	10	10	DATI.BIN

25	Одномерный символьный массив длиной 300 элементов заполнить из файла и распечатать по 60 символов в строке. С помощью функции заполнить два массива из 26 элементов, соответственно буквой и числом ее повторений (перед этим все символы массива сделать прописными). Упорядочить массив букв по убыванию повторений методом пересчета. Распечатать оба массива.	1) Распечатать исходный массив процедурой. 2) Составить функцию, возвращающую количество вхождений заданного символа в строке.	300		LAT.TXT
26	Используя процедуры, по исходным квадратным массивам (матрицам) A и B вычислить матрицу C=A*B+A. Исходные числа из файла читать, начиная с 18-го.	1) Печатать все массивы с помощью процедуры. 2) Построить две процедуры, которые для двух квадратных матриц (размерами до 7x7) вычисляют третью матрицу той же размерности: первая – сумму, вторая – произведение.	7	7	DATF.TXT
27	Одномерный символьный массив заполнить из числового файла, рассматривая его как символьный (пропуская концы строк). Используя функцию, заполнить два массива: цифр и числа их повторений. Пересортировать массив цифр по возрастанию числа повторений методом выбора. Распечатать оба массива до и после сортировки.	1) Распечатать исходный массив процедурой. 2) Составить процедуру печати массивов цифр и их повторений. 3) Составить функцию, возвращающую количество вхождений заданного символа в символьном массиве.	200		DATF.TXT

28	В прямоугольном вещественном массиве размером 9x7 заполнить первые пять столбцов числами, прочитанными из файла (читать начиная с 28 числа). С помощью этой процедуры в исходном массиве заполнить последний столбец	1) Построить процедуру, которая в заказанной строке прямоугольного массива 9x7 элементов находит самую большую по модулю разность двух элементов (не обязательно соседних) среди первых шести элементов и заносит найденное значение в седьмой элемент строки массива. 2) Заполненный массив распечатать процедурой			DATF.BIN
29	Прочитать в одномерный массив строк (длиной по 74 символа) первые 73 символа каждой строки файла DATF.TXT. Используя функцию, заполнить выходной одномерный строчный массив (элементы – строки длиной восемь символов) отдельными "словами" – изображениями чисел.	1) Составить строковую функцию, выбирающую из строки первое "слово" – первую подстроку между пробелами, заменяя при выборе в исходной строке выбранные символы пробелами. 2) Печать выходного массива по пять "слов" в строке выполнить с помощью процедуры.	10		DATF.TXT
30	Одномерный массив длиной M элементов из файла, начиная с 106 числа. Используя процедуру, заполнить целый одномерный массив длиной десять элементов, в котором каждый элемент будет равен суммарному количеству цифр каждого значения (от нуля до девяти) во всех числах прочитанного массива. Распечатать этот массив.	1) Исходный массив печатать процедурой. 2) Построить процедуру, которая подсчитывает, сколько цифр каждого значения (от нуля до девяти) встречается в передаваемом ей целом числе. Использовать операцию вычисления остатка от целочисленного деления на десять (целые числа содержат меньше шести цифр).	100..200		DATI.TXT
31	Прямоугольный массив заполнить числами из файла, начиная с 33-го. С помощью функции в исходном массиве сделать элементы последнего столбца, равными сумме всех элементов соответствующей строки.	1) Исходный и полученный массивы печатать процедурой. 2) Построить функцию, которая возвращает сумму четных по значению элементов заказанной строки прямоугольного массива.	13	10	DATI.BIN

Лабораторная работа N 8
Графика на Паскале

Задачи лабораторной работы

Вопросы, изучаемые в работе

- Изучение строения и использование процедур, функций, раздела `Graph`, используемого при выводе на экран информации в графической форме.
- Работа с цветом.
- Проектирование размещения графических объектов на экране.

Задание (общее ко всем вариантам)

- Построить в подходящей системе координат график заданной функции. В нижней строке экрана вывести уравнение, по которому строится график и диапазон изменения аргумента. Координатные оси графика должны быть размечены делениями, у которых должны стоять числовые значения.
- Исходные данные задаются константами в тексте программы.
- Результаты расчета выводятся в графической форме на экран.

Общие пояснения

Графический режим экрана

Чтобы использовать графику в программах на Турбо-Паскале необходимо научиться планировать размещение на экране выводимой информации (т.е. разобраться в алгоритме вывода графических объектов на экран) и уметь подключать и использовать стандартные графические процедуры и функции Турбо-Паскаля. Рассмотрим последовательно обе эти проблемы.

Для начала следует твердо усвоить, что экран дисплея может использоваться или в текстовом или в графическом режиме, но только не в обоих одновременно. Стандартное состояние экрана при работе в Турбо-Паскале (рассчитанном на операционную систему DOS) – в текстовый режим. При этом никакой графический вывод на экран невозможен, попытки вызова процедур графики приведут к ошибке и останову программы. После инициализации графики, бесполезными будут обращения к процедурам вывода на экран текстовой информации (с помощью Write и WriteLn), правда, без всяких сообщений и прерываний выполнения программы.

Перевод дисплея в графический режим (инициализация графики) позволяет получить доступ к любой точке экрана. При этом система координат экрана начинается в левом верхнем углу с точки, имеющей координаты 0,0. Ось X направлена слева направо, ось Y – сверху вниз. Каждая точка экрана имеет две координаты X и Y и какой-то цвет из допустимой палитры. Такая точка называется пикселем. В зависимости от состава аппаратных средств ПЭВМ, на экране максимально может помещаться или 350 (для EGA-адаптера), или 480 (для VGA-адаптера) строк по 640 пикселов в каждой строке. Палитра обычно содержит 16 цветов.

Алгоритмы построения графиков на экране

Графический режим может использоваться как для произвольного рисования (когда с помощью графических процедур проводятся отрезки прямых линий или дуги эллипсов через выбираемые точки экрана), так и для отображения рассчитываемых кривых (функций), задаваемых или уравнениями, или табличными значениями. Ниже рассматриваются только варианты построения графиков функций и аналитически заданных кривых.

Под функцией мы далее будем понимать зависимость вида `Y=f(x)`, а под аналитически заданной кривой – кривую, координаты каждой точки которой выражаются через независимый параметр (угол, время и т.д.): `X=f1(t)`, `Y=f2(t)`.

В принципе, оба вида рассчитываемых графиков можно строить по одному и тому же универсальному алгоритму, хотя график функции часто строят по упрощенной схеме. Рассмотрим сначала общий подход к построению графика.

Первым делом нужно определить, в каком виде график будет размещаться на экране. При этом решают, одинаковыми ли будут масштабы по обеим осям, какую часть экрана должен занять сам график и где будут подписи, будут ли изображаться оси координат и будут ли они отображать нулевые значения величин X и Y и т.д. По некоторым из этих вопросов решение можно принять только после того, как будут найдены диапазоны изменения значений X и Y участка строящейся кривой.

Для удобства последующих описаний обозначим именами X и Y (или именами, начинающимися с этих символов) величины, относящиеся к уравнениям кривой. Все экранные координаты и зависящие от них величины будем именовать, начиная с символа J для горизонтальных направлений и I – для вертикальных направлений.

В общем случае алгоритм построения графика включает следующие блоки:

1. Для заданных требований к строящейся кривой определить диапазон изменения X (`Xmin` и `Xmax`). При построении графика функции этот диапазон обычно задается.

2. Исходя из тех же требований, определить диапазон изменения Y (Ymin и Ymax). Для функции иногда предельные значения очевидны, как, например, для Sin(x) или Cos(x) или легко находятся для монотонных функций при граничных значениях аргументов (Exp(x), Ln(x) и т.д.). Если строится параметрически заданная кривая или сложная функция, поиск диапазонов проще выполнить после вычисления массива координат точек кривой (Xi[1..N] и Yi[1..N], которые в дальнейшем будут использоваться при построении самого графика). Из массивов максимумы и минимумы находятся простым перебором значений.
3. Принимается решение, будут ли изображаться оси координат (X=0 и Y=0) на рисунке кривой, и если нулевые значения не входят в найденные диапазоны, последние корректируются с учетом этих нулевых значений.
4. Если заранее известно, какой видеоадаптер будет использоваться, EGA или VGA, то максимальные размеры экрана в пикселях являются константами. Для EGA это 640 точек по X и 350 точек по Y, для VGA размер по Y равен 480 точкам. Тогда можно провести все дальнейшие расчеты экранных координат графика и только после этого переходить в графический режим для проведения самих построений на экране. Мы будем исходить из универсального подхода, строя алгоритм для размеров экрана, определяемых после открытия графического режима
5. Инициализация графики производится процедурой InitGraph. В качестве первого параметра задается целочисленная переменная, имеющая нулевое значение, тем самым выбирается наилучший возможный графический режим.
6. С помощью функций GetMaxX и GetMaxY определяются размеры экрана в пикселях.
7. Принимается решение о размерах и положении графика на экране. Если планируется поместить один рисунок (график), то на поля по краям экрана можно оставить до 20% размера экрана. Таким образом, можно задать:

```
Jmin=(0..0.2)GetMaxX   Jmax=(0.8..1.0)GetMaxX
Imin=(0.8..1.0)GetMaxY  Imax=(0..0.2)GetMaxY
```

8. Вычисляются коэффициенты перехода к экранным координатам. Из рис. 8 видно, что

```
Mx=(Jmax-Jmin)/(Xmax-Xmin); My=(Imax-Imin)/(Ymax-Ymin),
```

Отметим, что (My<0).

9. Все дальнейшие построения на экране выполняются с использованием формул перехода от декартовых координат X и Y к экранным координатам J и I:

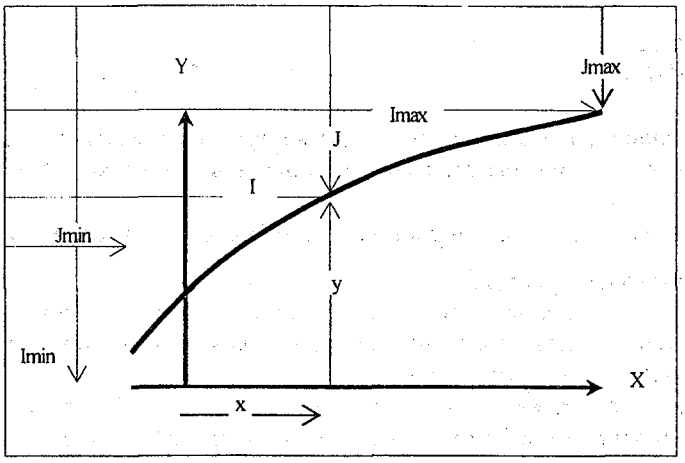


Рисунок 8. Схема размещения графических параметров

`J=Jmin+(X-Xmin)*Mx,  I=Imin+(Y-Ymin)*My`

Используя приведенные формулы, рассчитывается положение осей координат на экране. Перед их построением задаются параметры линий процедурами `SetColor` и `SetLineStyle` (см. список процедур ниже).

10. На координатных осях проводятся риски длиной 5–10 пикселей или строится координатная сетка (те же риски, но длиной от `Imin` до `Imax` и от `Jmin` до `Jmax`). Как и для осей, предварительно устанавливается стиль линий (линии более тонкие). Следует иметь ввиду, что строительство сетки или рисок необходимо выполнять от начала координат во всех четырех направлениях. Если начинать от края, риски, как правило, не пройдут через начало координат.
11. После задания характеристик кривой графика, указатель переводится в начальную точку кривой (не обязательно совпадающей с `Xmin,Ymin`).
12. В цикле по массиву точек `(Xi,Yi)` пересчитываются координаты `(J,I)` и проводятся отрезки процедурой `LineTo`.
13. После настройки параметров надписи процедурами `SetTextStyle` и `SetTextJustify`, указатель переводится в точку, относительно которой ориентируется текст и процедурой `OutTextXY` выводится надпись.
14. В конце программы, после всех задержек и, возможно, вывода графика на принтер, необходимо закрыть графический режим процедурой `CloseGraph`.

Примечание: работа в графическом режиме может выполняться, если в разделе подключенных модулей будет приведено имя модуля `Graph`:

```
PROGRAM ...
USES Graph;
  . . .
```

В данной работе рассмотрено два контрольных варианта – для функции (вариант №31) и параметрически заданной кривой (вариант №32).

Стандартный модуль работы с графическим экраном Graph

1) назначение модуля

Подключаются процедуры и функции по работе с экраном в графическом режиме, т.е. когда доступной становится любая точка (пиксель) экрана. Для их использования необходимо:

- подключить раздел графических подпрограмм Турбо-Паскаля, что выполняется в самом начале раздела описаний оператором `USES Graph;`
- инициализировать графику в выполняемом блоке, для чего загрузить в память драйвер управления монитором в графическом режиме. Это делается процедурой `InitGraph(...)`. Далее выполняется выбор цветов и характеристик графических примитивов (точек, линий, стандартных фигур и полигонов, символов текста) и изображаются нужные объекты, с помощью стандартных процедур и функций. По окончании работы графический режим монитора выключается (закрытием графики процедурой `CloseGraph`).

2) координаты экрана

Определяются возможностями видеосистемы ПЭВМ. Обычно стараются использовать наилучший возможный режим экрана

{ *** функции, связанные с координатами *** }

function GetX : integer; – получить текущую координату X;

function GetY : integer; – получить текущую координату Y;

function GetMaxX : integer; – получить максимально возможную координату экрана по X;

function GetMaxY : integer; – получить максимально возможную координату экрана по Y;

3) управление графическим режимом

{ *** определение, инициализация и восстановление текстового режима *** }

procedure DetectGraph (var GraphDriver, GraphMode : integer); – получение возможного типа драйвера и графического режима по установленным аппаратным средствам;

procedure InitGraph (var GraphDriver : integer;
var GraphMode : integer;
PathToDriver : String); – инициализировать графический режим экрана;

function GetMaxMode : integer; – получение наилучшего графического режима для данной ПЭВМ;
procedure SetGraphMode (Mode : integer); – задание графического режима;
function GetGraphMode : integer;– получение текущего графического режима;
procedure GraphDefaults; – установление графических параметров по-умолчанию (стандартных)
procedure RestoreCrtMode; – возвращение экрана в состояние, которое было до установления графики
procedure CloseGraph; – закрытие графического режима.

{ Коды завершений графических операций: }
grOk = 0; – без ошибок.
grNoInitGraph = -1; – не загружен драйвер графического режима.
grNotDetected = -2; – не определен тип видеокарты.
grFileNotFound = -3; – не найден файл с драйвером.
grInvalidDriver = -4; – ошибка работы драйвера.
grNoLoadMem = -5; – не хватает места в ОП для загрузки драйвера.
grNoScanMem = -6; – выход за пределы памяти при сканирующем заполнении
grNoFloodMem = -7; – выход за пределы памяти при заливке.
grFontNotFound = -8; – не найден заказанный шрифт.
grNoFontMem = -9; – не хватает места в ОП для загрузки шрифта.
grInvalidMode = -10; – неверный режим графики.
grError = -11; – ошибка графической операции.
grIOerror = -12; – ошибка графического ввода/вывода
grInvalidFont = -13; – ошибка в файле шрифта.
grInvalidFontNum = -14; – недопустимый номер шрифта.

{ *** Функции, возвращающие сведения об ошибках *** }
function GraphErrorMsg (ErrorCode : integer) : String; – название ошибки по коду;
function GraphResult : integer; – код ошибки последней графической операции.

4) управление экраном и окном
{ *** экран, окна, сохранение и восстановление окон *** }
procedure ClearDevice; – очиска грфического экрана цветом фона. Текущий указатель в левом верхнем углу;
procedure SetViewPort (x1, yI, x2, y2 : integer; Clip : boolean); – задание границ окна и типа отсечения;

procedure GetViewSettings (var ViewPort : ViewPortType); – получить характеристики окна;

procedure ClearViewPort; – очистить окно цветом фона;

procedure SetVisualPage (Page : word); – задает номер отображаемой графической страницы;

procedure SetActivePage (Page : word); – устанавливает для графического вывода активную страницу.

{ *** сохранение/восстановление части экрана *** }

function ImageSize (x1, y1, x2, y2 : integer) : word; – определить размер ОП для прямоугольника;

procedure GetImage (x1, y1, x2, y2 : integer; var BitMap); – сохранить в ОП образ прямоугольника;

procedure PutImage (X, Y : integer; var BitMap; BitBlt : word); – восстановить прямоугольник из ОП.

5) управление цветом

{ константы изображения цветов: }

Таблица 27. Кодировка цветов

Код	Имя константы	Цвет
0	Black	Черный (прозрачный)
1	Blue	синий
2	Green	зеленый
3	Cyan	голубой
4	Red	красный
5	Magenta	фиолетовый
6	Brown	коричневый
7	LightGray	светло-серый
8	DarkGray	темно-серый
9	LightBlue	светло-голубой
10	LightGreen	светло-зеленый
11	LightCyan	светло-синий
12	LightRed	светло-красный
13	LightMagenta	светло-фиолетовый (розовый)
14	Yellow	светло-коричневый(желтый)
15	White	белый

{ *** процедуры работы с цветом *** }

procedure SetBkColor (ColorNum : word); – установить цвет фона;

procedure SetColor(Color : word); – установить цвет рисования

function GetBkColor : word; – получить цвет фона;
function GetColor : word; – получить текущий цвет рисования;
function GetMaxColor : word; – получить максимально-возможное число цветов.

6) вывод точек
{ *** процедуры работы с точкой *** }
procedure PutPixel (X, Y : integer; Pixel : word); – поставить на экране точку (X,Y) заданным цветом (pixel);
function GetPixel (X, Y : integer) : word; – получить цвет точки с координатами (X,Y).

7) вывод линий
{ типы и толщины линий для процедур Get/SetLineStyle: }
SolidLn = 0; { сплошная };
DottedLn = 1; { пунктирная };
CenterLn = 2; { штрих-пунктирная (осевая) };
DashedLn = 3; { штриховая };
UserBitLn = 4; { задаваемая пользователем };
NormWidth = 1; { нормальная (тонкая) } ;
ThickWidth = 3; { толстая }.

{ *** процедуры перемещений и проведения отрезков линий *** }
procedure LineTo (X, Y : integer); – линия из текущей точки в (X, Y);
procedure LineRel (Dx, Dy : integer); – линия из текущей точки в точку, смещенную на (Dx, Dy);
procedure MoveTo (X, Y : integer); – переход в точку (X, Y);
procedure MoveRel (Dx, Dy : integer); – переход из текущей точки в точку, смещенную на (Dx,Dy);
procedure Line (x1, y1, x2, y2 : integer); – линия из точки (x1,y1) в точку (x2,y2);
procedure GetLineSettings (var LineInfo : LineSettingsType); – получить текущие настройки рисования линий;
procedure SetLineStyle (LineStyle : word;
Pattern : word;
Thickness : word); – задать текущие настройки рисования линий.

8) вывод и закраска контуров
{ *** многоугольники, их закраска и текстуры *** }
procedure Rectangle (x1, y1, x2, y2 : integer); – построить незакрашенный прямоугольник;
procedure Bar (x1, y1, x2, y2 : integer); – построить закрашенный прямоугольник;

procedure Bar3D (x1, y1, x2, y2 : integer; Depth : word; Top : boolean); – построить параллелепипед;
procedure DrawPoly (NumPoints : word; var PolyPoints); – построить контур многоугольника из NumPoints точек;
procedure FillPoly (NumPoints : word; var PolyPoints); – построить закрашенный многоугольник из NumPoints точек;
procedure GetFillSettings (var FillInfo : FillSettingsType); – получить текущие характеристики закраски;
procedure GetFillPattern (var FillPattern : FillPatternType); – получить текущие характеристики текстуры;
procedure SetFillStyle (Pattern : word; Color : word); – задать характеристики закраски;
procedure SetFillPattern (Pattern : FillPatternType; Color : word);– задать характеристики текстуры;
procedure FloodFill (X, Y : integer; Border : word); – залить область текущей закраской от заданной точки (X,Y) до границы, заданной цветом (Border).

9) окружности, эллипсы, дуги
{ *** построение окружностей, эллипсов и их частей *** }
procedure Arc (X, Y : integer; StAngle, EndAngle, Radius : word); – построение дуги окружности;
procedure GetArcCoords (var ArcCoords : ArcCoordsType); – получение параметров дуги окружности;
procedure Circle (X, Y : integer; Radius : word); – построение окружности заданного радиуса и центра;
procedure Ellipse (X, Y : integer;
    StAngle, EndAngle : word;
    XRadius, YRadius : word); – построение дуги эллипса;
procedure FillEllipse (X, Y : integer;
    XRadius, YRadius : word); – построение закрашенного эллипса.

procedure GetAspectRatio (var Xasp, Yasp : word); – получение относительного разрешения по X и Y;
procedure SetAspectRatio (Xasp, Yasp : word); – задание относительного разрешения по X и Y;
procedure PieSlice (X, Y : integer; StAngle, EndAngle, Radius : word); – построение закрашенного сектора круга;
procedure Sector (X, Y : Integer;
  StAngle, EndAngle,
  XRadius, YRadius : word); – построение закрашенного сектора эллипса.

10) вывод текста
{ константы для процедур Set/GetTextStyle }
DefaultFont = 0; { шрифт по-умолчанию };
TriplexFont = 1; { "оттененный" шрифт };
SmallFont = 2; { мелкий шрифт };
SansSerifFont = 3; { шрифт "сан-сериф"};
GothicFont = 4; { готический шрифт };
HorizDir = 0; { текст слева–направо };
VertDir = 1; { текст снизу–вверх };

{ размещение текста относительно заданной точки }
LeftText = 0; { текст влево от точки };
CenterText = 1; { точка в центре текста };
RightText = 2; { текст вправо от точки };
BottomText = 0; { текст под точкой };

{ CenterText = 1; уже определено выше }
TopText = 2; { текст над точкой };
UserCharSize = 0; { размер шрифта задается программистом }.

{ признаки отсечения фигур при выходе за пределы окна }
ClipOn = true; – за границами окна фигура не видна;
ClipOff = false; – за границами окна фигура видна;

{ *** процедуры вывода текста *** }
procedure GetTextSettings (var TextInfo : TextSettingsType); – получить текущие настройки вывода текста;
procedure OutText (TextString : string); – вывести текст относительно текущей точки;
procedure OutTextXY (X, Y : integer; TextString : string); – вывести текст относительно точки (X,Y);
procedure SetTextJustify (Horiz, Vert : word); – установить параметры центровки (размещения) текста;
procedure SetTextStyle (Font, Direction : word; CharSize : word); – выбрать шрифт, его размер и направление вывода текста;
procedure SetUserCharSize (MultX, DivX, MultY, DivY : word); – задать размер шрифта программиста;
function TextHeight (TextString : string) : word; – получить текущую высоту строки текста;
function TextWidth (TextString : string) : word; – получить текущую ширину строки текста.

Разбор контрольного варианта № 31

Задание

Построить график функции Y=exp(X) для интервала X от –1 до 2, нанести на график размеченные оси координат и сделать подпись (название функции).

Дополнительные требования: график изобразить толстой штриховой линией красного цвета на белом фоне, координатные оси (толстая линия) и разметка (тонкая линия) – черного (темно-серого) цвета, подпись – синего цвета под графиком посередине.

Программу составить для EGA и VGA адаптеров (универсальную). Драйвер графического адаптера (с именем EGAVGA.BGI) находится в каталоге D:\TP6\BGI.

Условия, принятые из соображений дизайна:

График будет занимать 60% ширины и высоты экрана. Разметку проводим через 0.5 по X и через 1.0 по Y. График рисуем процедурой LineTo, с шагом 0.2 по оси X.

Таблица идентификаторов

Таблица 28. Идентификаторы задачи 31-го варианта

Имя	Тип	Размер, (байт)	Назначение
Graph_work	Имя программы	-	Построение графика функции
Graph	Имя модуля	-	Стандартные графические подпрограммы
Crt	"	-	Стандартные подпрограммы работы с консолью
Print	"	-	Стандартные подпрограммы работы с принтером
X	Веществен.	6	Текущее значение аргумента
Y	"	"	Текущее значение функции
Xmin	"	"	Минимальное значение аргумента
Xmax	"	"	Максимальное значение аргумента
Ymin	"	"	Минимальное значение функции
Ymax	"	"	Максимальное значение функции
DeltaX	"	"	Диапазон изменения аргумента
DeltaY	"	"	Диапазон изменения функции
Dx	"	"	Шаг аргумента для расчета графика функции
Dxs	"	"	Шаг разметки оси X
Dys	"	"	Шаг разметки оси Y
Part	"	"	Доля экрана, занимаемая графиком

Имя	Тип	Размер, (байт)	Назначение
Pole	"	"	Размер полей вокруг графика в долях экрана
Mx	"	"	Коэффициент пересчета X в J
My	"	"	Коэффициент пересчета Y в I
I	Целое	2	Горизонтальная координата экрана
J	"	"	Вертикальная координата экрана
Imin	"	"	Нижняя граница графика на экране
Imax	"	"	Верхняя граница графика на экране
Jmin	"	"	Левая граница графика на экране
Jmax	"	"	Правая граница графика на экране
JAll	"	"	Размер экрана по X в пикселах
Iall	"	"	Размер экрана по Y в пикселях
J0	"	"	Координата оси Y на экране
I0	"	"	Координата оси X на экране
DeltaJ	"	"	Размер рисунка по X
DeltaI	"	"	Размер рисунка по Y
Nx	"	"	Шаг между рисками оси X в пикселях
Ny	"	"	Шаг между рисками оси Y в пикселях
Riska	Строка	6	Подпись текущей риски оси
GraphDrv	Целое	2	Тип графического драйвера
GraphMode	"	"	Номер графического режима
Code	беззнаковое	"	Код завершения процедуры инициализ. граф.
InitGraph	Имя процедуры	-	Инициализация графического режима
GraphResult	Имя функции	-	Возвращает код завершения граф. процедуры
ClearDevice	Имя процедуры	-	Очистка экрана заданным цветом фона
Halt	Имя процедуры	-	Останов (завершение) программы
SetBkColor	Имя процедуры	-	Установка цвета фона
SetColor	Имя процедуры	-	Установка цвета
SetLineStyle	Имя процедуры	-	Установка типа линии
SetTextStyle	Имя процедуры	-	Установка стиля текста
SetTextJustify	Имя процедуры	-	Установка способа размещения текста
GetMaxX	Имя функции	-	Возвращает размер экрана по X
GetMaxY	Имя функции	-	Возвращает размер экрана по Y
Round	Имя функции	-	Округляет вещественный аргумент в целое
MoveTo	Имя процедуры	-	Переход в заданную точку экрана
LineTo	Имя процедуры	-	Проведение отрезка в заданную точку
Line	Имя процедуры	-	Проведение отрезка
OutTextXY	Имя процедуры	-	Вывод текста
Str	Имя процедуры	-	Преобразование числа в строку с его изображен.

Имя	Тип	Размер, (байт)	Назначение
KeyPressed	Имя функции	-	Возвращает TRUE, если нажата клавиша
CloseGraph	Имя процедуры	-	Закрытие графического режима
Pr	Имя процедуры	-	Копирование графического экрана на принтер

Алгоритм

```
1.Задание констант, стандартных значений;
2.Ввод исходных данных;
3.Печать исходных данных;
4.Расчет характеристик функций;
5.Открытие графики с проверкой правильности срабатывания;
6.Настройка фона;
7.Расчет параметров графика на экране;
8.Расчет масштабных коэффициентов перехода от X к J и от
Y к I;
9.Построение графика функции:
  9.1.Задание характеристик линии;
  9.2.Начальная точка графика;
  9.3.Цикл расчета экранных координат графика (X->J, X-
>Y->I)  и проведения отрезков;
10.Построение осей координат:
  10.1.Задание характеристик линии;
  10.2.Построение осей;
11.Разметка осей:
  11.1.Задание характеристик линии;
  11.2.Характеристики шрифта для подписи значений;
  11.3.Цикл проведения разметки оси X, риски вверх от
оси, по 10 пикселов:
    11.3.1. от начала координат - вправо;
    11.3.1. от начала координат - влево;
  11.4. Цикл проведения разметки оси Y, риски вправо от
оси, по 10 пикселов:
      11.4.1.от начала координат - вверх;
      11.4.2.от начала координат - вниз (Для данной функ-
ции ниже оси X разметка не нужна);
12.Подпись графика:
  12.1.Характеристики шрифта, цвет;
  12.2.Вывод подписи;
13.Задержка графика на экране;
14.Вывод графика на печать - только если подключен прин-
тер;
15.Закрытие графического режима.
```

Блок-схема алгоритма

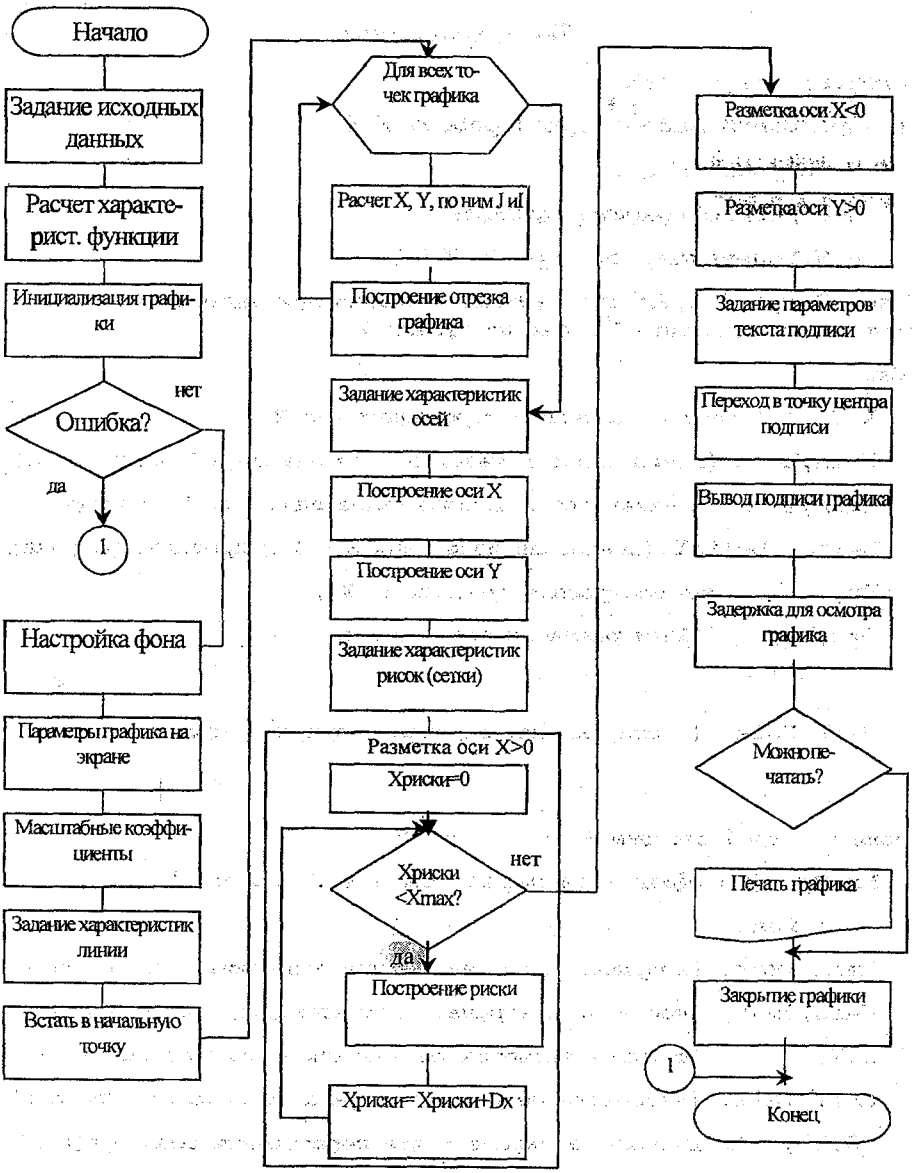


Рисунок 9. Алгоритм программы 31-го варианта

Текст программы

```
Program Graph_work;
{  Программа Лабораторной работы N 8.
     Вариант N 31.
   Построение графика функции.
   А.Я.Умненькая, ст. гр. Я-007 }
Uses Graph,Crt,Print;  {Print - только при наличии принтера
для печати графика с экрана на бумагу }
Var
  X,Y,    { текущие значения переменных X и Y }
  Xmin,Xmax,{минимальное и максимальное значения X на графике}
  Ymin,Ymax,{минимальное и максимальное значения Y на графике}
  DeltaX,DeltaY,{диапазоны изменения X и Y в пределах рисунка}
  Dx,       { шаг построения графика по X }
  Dxs,Dys,  { Шаги разметки осей по X и по Y }
                                                    нет
  Part,Pole, { Доля экрана, занятая рисунком и доля чистого
                                                    да
поля с каждой стороны }
  Mx,My  {Коэффициенты пересчета X в J и Y в I}
     : real;
  Imin,Imax, {координаты экрана, соответствующие Ymin и Ymax}
  Jmin,Jmax, {координаты экрана, соответствующие Xmin и Xmax}
  J,I,       {текущие значения переменных координат экрана }
  JAll,IAll, {Максимальные размеры экрана (в пикселях) по X,Y}
  J0,I0, {Координаты на экране точки пересечения осей графика}
  DeltaJ,DeltaI,{Экранные размеры рисунка по X и Y в пикселях}
  Nx,Ny          {Шаги разметки осей X и Y в пикселях }
     : integer;
```

```
  Riska : String[5]; {Строка для вывода чисел у рисок осей}
  GraphDrv,      {Тип графического драйвера }
  GraphMode: integer;  {Устанавливаемый графический режим}
  Code: word;     {Код завершения инициализации графики }
BEGIN { Начало программы }
{1.Задание констант, стандартных значений }
 Xmin:=-1.0; Xmax:=2.0;
 Dxs:=0.5;  Dys:=1.0;
 Part:=0.6; Dx:=0.2;
{2.Ввод исходных данных - в примере не используем}
{3.Печать исходных данных - в примере не используем}
{4.Расчет характеристик функций }
 DeltaX:=Xmax-Xmin;
 Ymin:=exp(Xmin);
 Ymax:=exp(Xmax);
 if Ymin>0 then Ymin:=0;
 if Ymax<0 then Ymax:=0;
 DeltaX:=Xmax-Xmin;
 DeltaY:=Ymax-Ymin;
{5.Открытие графики с проверкой правильности срабатыва-
ния}
 GraphDrv:=0;    {пусть определяет режим сам }
 InitGraph(GraphDrv,GraphMode,'D:\TP6\BGI');
 Code:=GraphResult;
 if Code<>0 then {если завершение с кодом не 0 - закон-
чить программу}
 Begin
 writeln('Ошибка открытия графики с кодом: ',Code);
  Halt; { Останов программы }
 End;
{6.Настройка фона}
```

```
 SetBkColor(15); {Фон белый }
 ClearDevice;  {Очистка экрана}
{7.Расчет параметров графика на экране}
 JAll:=GetMaxX;
 IAll:=GetMaxY;
 Pole:=(1.0-Part)/2.0;
 Jmin:=Round(Pole*JAll);
 Jmax:=Round((1.0-Pole)*JAll);
 Imin:=Round((1.0-Pole)*IAll);
 Imax:=Round(Pole*IAll);     { Imin > Imax !!! }
 DeltaJ:=Jmax-Jmin;
 DeltaI:=Imax-Imin;
{8.Расчет масштабных коэффициентов перехода от X к J и от
Y к I}
 Mx:=DeltaJ/DeltaX;
 My:=DeltaI/DeltaY;
{9.Построение графика функции}
 {9.1.Задание характеристик линии}
 SetColor(4);  { цвет красный }
 SetLineStyle(3,0,3); { штриховая толстая }
{9.2.Начальная точка графика}
 Nx:=Round(Dxs*Mx);
 Ny:=Round(Dys*My);  { Шаг разметки по Y - отрицательный
}
 J:=Jmin;
 I:= Imin+Round((exp(Xmin)-Ymin)*My);
 MoveTo (J,I);
{9.3.Цикл расчета экранных координат графика (X->J, X->Y-
>I) и проведения отрезков}
 X:=0;
 While X<=Xmax do
 Begin
```

```
  X:=X+Dx;
  Y:=exp(X);
  J:=Jmin+Round((X-Xmin)*Mx);
  I:=Imin+Round((Y-Ymin)*My);
  LineTo(J,I);
 End;
{10.Построение осей 10.1.Задание характеристик линии }
 SetColor(8);    { цвет темно-серый }
 SetLineStyle(0,0,3); { сплошная толстая }
{10.2.Построение осей }
 J0:=Jmin+Round((0-Xmin)*Mx);
 I0:=Imin+Round((0-Ymin)*My);
 Line(J0,Imin,J0,Imax); {Построение оси Y (где X=0)}
 Line(Jmin,I0,Jmax,I0); {Построение оси X (где Y=0)}
{11.Разметка осей 11.1.Задание характеристик линии }
 SetLineStyle(0,0,0); { сплошная тонкая }
{11.2.Характеристики шрифта для подписи значений }
 SetTextStyle(0,0,0); { шрифт стандартный, подпись гори-
зонтальна }
 SetTextJustify(1,2);  {Размещение текста симметрично,
относительно заданной точки по горизонтали и ниже точки
по вертикали}
{11.3.Цикл проведения разметки оси X, риски вверх от оси,
по 10 пикселов }
 J:=J0;
 X:=0;
 while J<=Jmax+1 do {от начала координат - вправо }
 begin
  Line(J,I0,J,I0-10);
  Str(X:3:1,Riska);
  OutTextXY(J,I0+5,Riska);
  X:=X+Dxs;
```

```
  J:=Jmin+Round((X-Xmin)*Mx);
 end;
 J:=J0;
 X:=0;
 while J>=Jmin do  {от начала координат - влево }
 begin
  Line(J,I0,J,I0-10);
  Str(X:3:1,Riska);
  OutTextXY(J,I0+5,Riska);
  X:=X-Dxs;
  J:=J-Nx;
 end;
{11.4.Цикл проведения разметки оси Y, риски вправо от
оси, по 10 пикселов }
 SetTextJustify(2,1);  {Размещение текста симметрично,
относительно заданной точки по вертикали и левее точки по
горизонтали }
 I:=I0;
 Y:=0;
 while I>=Imax do {от начала координат - вверх }
 begin
  Line(J0,I,J0+10,I);
  Str(Y:3:1,Riska);
  OutTextXY(J0-10,I,Riska);
  Y:=Y+Dxs;
  I:=I+Ny;
 end;
 { Для данной функции ниже оси X разметка не нужна}
 I:=I0;
 while I<=Imin do
 begin
```

```
  Line(J0,I,J0+10,I);
  I:=I-Ny;
 end;
{12.Подпись графика}
{12.1.Характеристики шрифта, цвет }
 SetColor(1);  { цвет синий }
 SetTextJustify(1,1);  {Размещение текста симметрично, относительно заданной точки по горизонтали и по вертикали}
{12.2.Вывод подписи }
 I:=Round(IAll*(1-Pole/2)); {середина нижнего поля}
OutTextXY(JAll div 2,I,'График функции Y = exp(X)');
{13.Задержка графика на экране }
 while Not KeyPressed do;
{14. Вывод графика на печать (на принтер) - только если он подключен }
 Pr;
 CloseGraph; {14.Закрытие графического режима }
END.
```

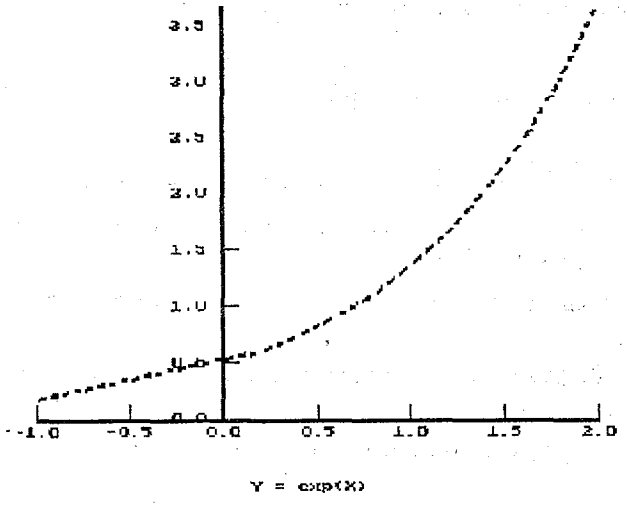


Рисунок 10. Результат работы программы 31-го варианта

Разбор контрольного варианта № 32

Задание

Построить график кривой, заданной параметрически: X=(2+0.5cos(8t))cos(t), Y=(2+0.5cos(8t))cos(t) для интервала t от 0 до 2π, нанести на график размеченные оси координат и сделать подпись.

Дополнительные требования: график изобразить тонкой сплошной линией красного цвета на белом фоне, координатные оси (толстая линия) и разметка (тонкая линия) – черного (темно-серого) цвета, подпись – синего цвета под графиком посередине.

Программу составить для EGA и VGA адаптеров (универсальную). Драйвер графического адаптера (с именем EGAVGA.BGI) находится в каталоге D:\TP6\BGI.

Условия, принятые из соображений дизайна:

График будет занимать 60% ширины и высоты экрана. Разметку проводим через 0.5 по X и Y. График рисуем процедурой LineTo, для 400 точек.

```
Program Graph_work2;
{  Программа Лабораторной работы N 8
     Вариант N 32.
   Построение кривой, заданной параметрически.
   А.Я.Умненькая, ст. гр. Я-007 }
Uses Graph,Crt;  { Print - только при наличии принтера
для печати графика с экрана на бумагу }
TYPE
  mas=array[0..400] of real;
Var
  X,Y:mas;    { текущие значения переменных X и Y }
  Xmin,Xmax,t,tmin,tmax, { минимальное и максимальное
значения X на графике }
  Ymin,Ymax, { минимальное и максимальное значения Y на
графике }
  DeltaX,DeltaY, { диапазоны изменения X и Y в пределах
рисунка }
  Dt,R,      { шаг построения графика по t }
  Xj,Yi,
  Dxs,Dys,  { Шаги разметки осей по X и по Y }
```

```
  Part,Pole, { Доля экрана, занятая рисунком и доля чис-
того поля с каждой стороны  }
  Mx,My  {Коэффициенты пересчета X в J и Y в I}
     : real;
  Imin,Imax,  { координаты экрана, соответствующие Ymin и
Ymax }
  Jmin,Jmax,  { координаты экрана, соответствующие Xmin и
Xmax }
  J,I,N,k,    { текущие значения переменных координат эк-
рана }
  JAll,IAll, {Максимальные размеры экрана (в пикселях) по
X и Y}
  J0,I0,  {Координаты на экране точки пересечения осей
графика }
  DeltaJ,DeltaI { Экранные размеры рисунка по X и Y в
пикселях}
     : integer;
  Riska : String[5]; { Строка для вывода чисел у рисок
осей }
  GraphDrv,      { Тип графического драйвера      }
  GraphMode: integer;{ Устанавливаемый графический режим
}
  Code: word;    { Код завершения инициализации графики }
BEGIN { Начало программы }
{1.Задание констант, стандартных значений }
 tmin:=0; tmax:=2.0*Pi;
 Dxs:=0.5;  Dys:=0.5;
 Part:=0.6; N:=401;
{4.Расчет характеристик функций }
 Dt:=(tmax-tmin)/(N-1);
 Xmax:=-100;
 Xmin:=100;
 Ymax:=-100;
```

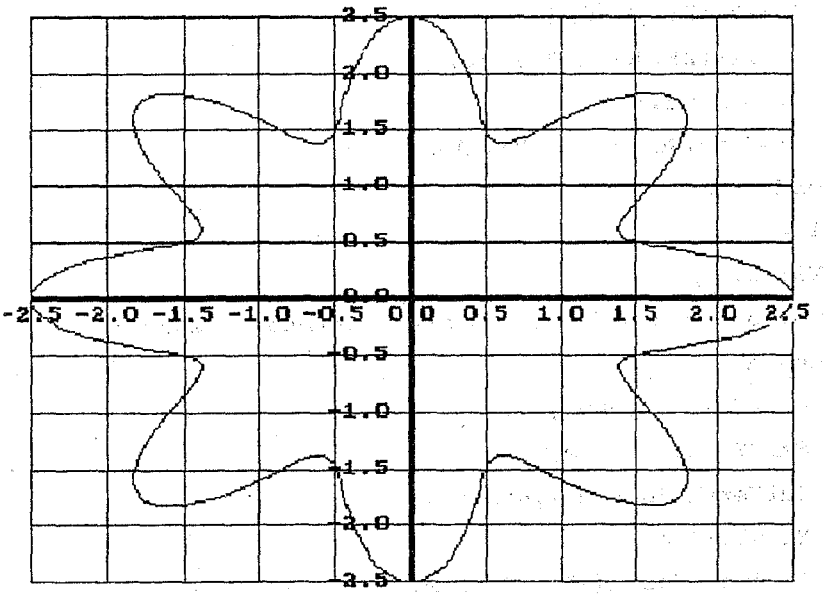
```
 Ymin:=100;
 For k:=0 to N-1 do
 begin
   t:=tmin+Dt*k;
   R:=2.0+0.5*cos(8.0*t);
   X[k]:=R*cos(t);
   Y[k]:=R*sin(t);
   if X[k]>Xmax then Xmax:=X[k];
   if X[k]<Xmin then Xmin:=X[k];
   if Y[k]>Ymax then Ymax:=Y[k];
   if Y[k]<Ymin then Ymin:=Y[k];
 end;
 DeltaX:=Xmax-Xmin;
 DeltaY:=Ymax-Ymin;
{5.Открытие графики с проверкой правильности срабатыва-
ния}
 GraphDrv:=0;    {пусть определяет режим сам }
 InitGraph(GraphDrv,GraphMode,'C:\voronov\STUD\TP\BGI');
 Code:=GraphResult;
 if Code<>0 then {если завершение с кодом не 0 - закон-
чить программу}
 Begin
 writeln('Ошибка открытия графики с кодом: ',Code);
  Halt; { Останов программы }
 End;
{6.Настройка фона}
 SetBkColor(15); {Фон белый }
 ClearDevice;  {Очистка экрана}
{7.Расчет параметров графика на экране}
 JAll:=GetMaxX;
 IAll:=GetMaxY;
```

```
 Pole:=(1.0-Part)/2.0;
 Jmin:=Round(Pole*JAll);
 Jmax:=Round((1.0-Pole)*JAll);
 Imin:=Round((1.0-Pole)*IAll);
 Imax:=Round(Pole*IAll);     { Imin > Imax !!! }
 DeltaJ:=Jmax-Jmin;
 DeltaI:=Imax-Imin;
{8.Расчет масштабных коэффициентов перехода от X к J и от
Y к I}
 Mx:=DeltaJ/DeltaX;
 My:=DeltaI/DeltaY;
{9.Построение графика функции}
 {9.1.Задание характеристик линии}
 SetColor(4);  { цвет красный }
 SetLineStyle(0,0,2);
{9.2.Начальная точка графика}
 J:=Jmin+Round((X[0]-Xmin)*Mx);
 I:=Imin+Round((Y[0]-Ymin)*My);
 MoveTo (J,I);
{9.3.Цикл расчета экранных координат графика (X->J, Y->I)
и проведения отрезков}
 for k:=1 to N-1 do
 Begin
   J:=Jmin+Round((X[k]-Xmin)*Mx);
   I:=Imin+Round((Y[k]-Ymin)*My);
   LineTo(J,I);
 End;
{10.Построение осей координат }
 SetColor(8);   { цвет темно-серый }
 SetLineStyle(0,0,3); { сплошная толстая }
 J0:=Jmin+Round((0-Xmin)*Mx);
```

```
 I0:=Imin+Round((0-Ymin)*My);
 Line(J0,Imin,J0,Imax); {Построение оси Y (где X=0)}
 Line(Jmin,I0,Jmax,I0); {Построение оси X (где Y=0)}
{11.Разметка осей}
 SetLineStyle(0,0,0); { сплошная тонкая }
 SetTextStyle(0,0,0); { шрифт стандартный, подпись гори-
зонтальна }
 SetTextJustify(1,2);  {Размещение текста симметрично,
относительно заданной
          точки по горизонтали и ниже точки по вертикали}
 J:=J0;
 Xj:=0;
 while J<=Jmax+1 do {от начала координат - вправо }
 begin
  Line(J,Imin,J,Imax);
  Str(Xj:3:1,Riska);
  OutTextXY(J,I0+5,Riska);
  Xj:=Xj+Dxs;
  J:=Jmin+Round((Xj-Xmin)*Mx);
 end;
 J:=J0;
 Xj:=0;
 while J>=Jmin do  {от начала координат - влево }
 begin
  Line(J,Imin,J,Imax);
  Str(Xj:3:1,Riska);
  OutTextXY(J,I0+5,Riska);
  Xj:=Xj-Dxs;
  J:=Jmin+Round((Xj-Xmin)*Mx);
 end;
```

```
 SetTextJustify(2,1);  {Размещение текста симметрично,
относительно заданной
точки по вертикали и левее точки по горизонтали }
 I:=I0;
 Yi:=0;
 while I>=Imax-1 do {от начала координат - вверх }
 begin
  Line(Jmin,I,Jmax,I);
  Str(Yi:3:1,Riska);
  OutTextXY(J0-10,I,Riska);
  Yi:=Yi+Dxs;
  I:=Imin+Round((Yi-Ymin)*My);
 end;
 I:=I0;
 Yi:=0;
 while I<=Imin do {от начала координат - вниз }
 begin
  Line(Jmin,I,Jmax,I);
  Str(Yi:3:1,Riska);
  OutTextXY(J0-10,I,Riska);
  Yi:=Yi-Dxs;
  I:=Imin+Round((Yi-Ymin)*My);
 end;
{12.Подпись графика}
 SetColor(1);  { цвет синий }
 SetTextJustify(1,1);  {Размещение текста симметрично,
относительно заданной
точки по горизонтали и по вертикали}
 I:=Round(IAll*(1-Pole/2)); {середина нижнего поля}
OutTextXY(JAll div 2,I,'X=(2+sin(6t))cos(t),
Y=(2+sin(6t))cos(t)');
```

```
{13.Задержка графика на экране }
 while Not KeyPressed do;
{14. Вывод графика на печать (на принтер) - только если
он подключен }
 {Pr;}
{14.Закрытие графического режима }
 CloseGraph;
END.
```



X=(2+sin(6t))cos(t), Y=(2+sin(6t))cos(t)

Рисунок 11. Результат работы программы 32-го варианта

Варианты заданий

Таблица 29. Варианты заданий лабораторной работы N9

N вар	Функция или параметрическое уравнение	Аргумент и его диапазон	Цвет	
			текста/ фона	графика/ осей координат
1	Парабола $Y=1.3*X^2-1.8$	X [-1.2, 1.2]	голубой белый	зеленый голубой
2	Окружность $X=0.5+2*\cos(t)$ $Y=0.2+2*\sin(t)$	t [0, 2π]	красный темно-серый	светло-красный коричневый
3	Степенная функция $Y=X^3-2*X^2+X$	X [-1, 3]	розовый темно-серый	светло-зеленый светло-серый
4	Эллипс $x=3*\cos(t)$, $y=15*\sin(t)$	t [0, 2π]	синий светло-серый	светло-синий темно-серый
31	Конхоида Никомеда $X=A+B*\cos(f)$ $Y=A*tg(f)+B*\sin(f)$	f [1.5, 4.5] A=1, B=2	светло-красный темно-серый	светло-зеленый красный
6	Кардиоида $X=4*\cos(t)*(1+\cos(t))$ $Y=4*\sin(t)*(1+\cos(t))$	t [0, 2π]	розовый красный	синий розовый
7	Дробно-рациональная функция $Y=(1.5*X+3)/(X-2)$	X [-4.2,1.9]	светло-серый розовый	голубой желтый
8	Декартов лист $X=3*A*t/(1+t^3)$ $Y=3*A*t2/(1+t^3)$	t [-0.5,10] A=2	синий белый	светло-синий светло-зеленый
9	Функция синус $Y=2.5*\sin(X)+0.5$	X [-2π, 2π]	белый зеленый	желтый коричневый
10	Циссоида $X=5*t^2/(1+t^2)$, $Y=5*t^2/(1+t^2)$, $t=tg(f)$	f [-π/4, π/4]	розовый черный	зеленый белый
11	Тригонометрическая функция $Y=\cos(X^2)$	X [-2π, 2π]	светло-зеленый светло-синий	белый розовый

N вар	Функция или параметрическое уравнение	Аргумент и его диапазон	Цвет	
			текста/ фона	графика/ осей координат
12	Строфоида $X=4*(t^2-1)/(t^2+1)$, $Y=4*t*(t^2-1)/(t^2+1)$ $t=tg(f)$	f $[-\pi/2.5, \pi/2.5]$	темно-серый голубой	белый желтый
13	Тригонометрическая функция $Y=tg(X)-2*X$	X $[-\pi/2.5, \pi/2.5]$	зеленый синий	фиолетовый черный
14	Астроида $X=3.5*\cos^3(t)$, $Y=3.5*\sin^3(t)$	t $[0, 2\pi]$	голубой белый	светло-зеленый зеленый
15	Арксинус $Y=\arcsin(0.5*X)$	X [-2, 2]	темно-серый голубой	светло-зеленый зеленый
16	Эпициклоида $X=(a+b)\cos(t)-a*\cos((a+b)*t/a)$, $Y=(a+b)\sin(t)-a*\sin((a+b)*t/a)$	t $[0, 2\pi]$ a=6, b=9	розовый фиолетовый	голубой синий
17	Логарифм $Y=\ln(X+2)$	X [-1.5, 5]	зеленый красный	желтый светло-зеленый
18	Гипоциклоида $X=2a*\cos(f)+a*\cos(2f)$ $Y=2a*\sin(f)-a*\sin(2f)$	f $[-\pi, \pi]$ a=1	белый розовый	темно-серый голубой
19	Арктангенс $Y=3*\text{arctg}(X)$	X [-5, 5]	светло-зеленый голубой	синий светло-синий
20	Эвольвента окружности $X=a*\cos(f)+a*f*\sin(f)$ $Y=a*\sin(f)-a*f*\cos(f)$	f $[-9\pi, 9\pi]$ a=1.5	зеленый синий	голубой черный
21	Дробно-рациональная нелинейная функция $Y = A + B/X + C/X^2$	X [0.18, 3] A=1, B=2, C=-0.5	темно-серый белый	фиолетовый красный
22	Лемниската $X=r*\cos(f)$ $Y=r*\sin(f)$ $r=a*\text{sqrt}(2*\cos(2f))$	f $[-\pi, \pi]$	светло-зеленый фиолетовый	синий белый
23	Локон Аньези $Y=A^3/(X^2 + A^2)$	X [-5, 5] A=2	зеленый белый	темно-серый светло-зеленый

N вар	Функция или параметрическое уравнение	Аргумент и его диапазон	Цвет	
			текста/ фона	графика/ осей координат
24	Архимедова спираль X=r*cos(f) Y=r*sin(f) r=A*f	f [-6π, 6π] A=1.5	розовый желтый	синий светло-красный
25	Трохоида(удлинненая циклоида) X=A*(1+B*sin(f)) Y=A*(1-B*cos(f))	f [-2π, 4π] A=1.5, B=1.3	темно-серый белый	светло-красный фиолетовый
26	Гиперболическая спираль X=(A*cos(f))/f Y=(A*sin(f))/f	f [0.1, 10] A=3	фиолетовый зеленый	светло-зеленый розовый
27	Удлиненная эпициклоида X=5*cos(f)-2*cos(5f) Y=5*sin(f)-2*sin(5f)	f [-π, π]	светло-серый синий	светло-синий красный
28	Логарифмическая спираль X=r*cos(f) Y=r*sin(f) r=A*exp(B*f)	f [0, 4] A=1.3, B=0.5	светло-зеленый синий	белый зеленый
29	Удлиненная гипоциклоида X=4*cos(f)+2*cos(4f) Y=4*sin(f)-2*sin(4f)	f [0, 2π]	темно-серый белый	синий зеленый
30	Улитка Паскаля X=2cos^2(t)+3cos(t), Y=2*cos(t)sin(t)+3sin t	t [0, 2π]	коричневый желтый	зеленый синий
31	Показательная функция Y=exp(X^2)	X [-1,2]	белый красный	красный темно-серый
32	X=(2+0.5cos(8t))cos(t), Y=(2+0.5cos(8t))cos(t)	t [0 до 2π]	синий белый	красный темно-серый

Лабораторная работа № 9
Динамические переменные. Списки

Задачи лабораторной работы

Вопросы, изучаемые в работе

- Разработка программы с динамическим выделением памяти.
- Работа с переменными комбинированного типа – записями.
- Работа с переменными ссылочного типа – указателями.
- Программирование списков записей.

Задание (общее ко всем вариантам)

В лабораторной работе требуется сформировать заданный тип списка, заполнить его в соответствии с указаниями варианта задания данными из входного файла (типизированного или текстового) и вывести содержимое списка в виде таблицы в выводной текстовый файл по одной записи в строку.

Файл данных `Dan.dat` находится в каталоге `D:\LAB1\` и состоит из записей. Первое поле каждой записи файла данных содержит фамилию и инициалы студента, второе и третье поля – оценки по дисциплинам, четвертое поле – среднюю оценку. Файл `Dan.txt` расположен там же и содержит ту же информацию, но в форме символьных строк.

В таблице вариантов указаны условия, которым должны отвечать записи данных, выбираемые из файла, а также типы списка и файла данных. Поля заглавного (в нульсвязных списках – первого обслуживаемого) звена должны содержать сведения о типе списка и количестве звеньев в нем.

В задании для типов списков используются следующие обозначения:

Таблица 30. Обозначения типов списков

Тип списка	Обозначение
Односвязный линейный	S1L
Односвязный кольцевой, заголовок внутри	S1KI
Односвязный кольцевой, заголовок вне	S1KO
Двусвязный линейный	S2L
Двусвязный кольцевой, заголовок внутри	S2KI
Двусвязный кольцевой, заголовок вне	S2KO
Стек	S0S
Очередь	S0O
Дек	S0D

Требования к программе

- Программа должна содержать комментарий с указанием названия работы, № варианта, фамилии студента и № группы.
- Все созданные в программе динамические переменные в конце должны быть удалены с освобождением памяти.
- В подпрограммах не использовать глобальные переменные, кроме имен файлов.
- Выводимая таблица должна быть озаглавлена в соответствии с заданием (какую выборку из исходного набора данных она содержит).
- В заглавном элементе списка должен быть записан тип списка в форме:

`Тип списка:<обозначение>` и количество записей с данными (в первом целочисленном поле)

Содержание программы

- формирование заглавного звена списка;
- цикл чтения записей из файла данных и занесения их в список;
- заполнение полей записи заглавного звена списка;
- вывод записей данных из списка в выводной файл;
- удаление списка.

Общие пояснения

Переменные, которые описываются в разделе описаний (VAR), называются статическими. Память для них выделяется перед началом выполнения программы, и во время выполнения программы не может быть изменена. По окончании программы, эта выделенная память автоматически освобождается.

Однако, статические переменные в Паскале не могут в сумме превышать 64 килобайта оперативной памяти. Кроме того, уже при составлении программы необходимо предусмотреть выделение памяти на максимально возможное количество данных, так как заранее требуемый объем данных может быть не известен.

Для устранения этих недостатков можно использовать динамическое выделение памяти под данные в процессе выполнения программы. Такая динамически выделяемая память размещается уже за пределами статического сегмента, и по объему ограничена только размерами свободной памяти ЭВМ.

Переменные, которые размещаются в этой памяти, называются динамическими. У них нет имен и для обращения к ним используются указатели – статические переменные адресного типа (или, что то же самое, ссылочного типа).

Значением указателя является физический адрес переменной базового типа, задаваемого идентификатором типа. Синтаксическое выражение для описания ссылочного типа имеет вид

<тип–указатель>::=^<идентификатор базового типа>

где символ ^ – признак ссылочного типа;

<идентификатор базового типа> – описанное ранее или стандартное имя типа.

Пример описания **<тип–указатель>**:

```
Type
  mas=array[1..100] of real;  {тип – массив вещественных чисел}
  dinmas=^mas; {тип указатель для значений переменных типа mas}
```

Переменные ссылочного типа вводятся, как и другие переменные, путем перечисления их имен в разделе описания переменных с указанием типа.

```
Var
  Pj:^integer; {указатель целого числа}
  Pc:^char; {указатель символа}
  Pd:^dinmas; {указатель массива}
```

Для приведенного примера описания динамических переменных Pj, Pc, и Pd значениями этих переменных будут, соответственно адреса данных каких-нибудь переменных базовых типов: соответственно целых, символьных и массива вещественных чисел.

Динамические переменные базового типа не имеют имен. В качестве имени для них используются конструкции вида:

^<имя указателя>

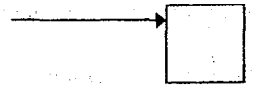
Например: **Pj^, Pc^, Pd^.**

Присваивать значения указателям можно тремя способами:

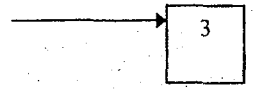
1. Записав в него адрес существующей статической переменной, получив его операцией @: **Pj:=@N;** конечно, тип переменной должен соответствовать базовому типу, использованному для описания Pj.
2. Присваиванием значения другого указателя того же типа или стандартного значения **nil.**
3. Присваиванием указателю адреса вновь выделенного места в оперативной памяти – то есть адреса начала динамической переменной.

В данной работе будут использоваться второй и третий способы присваивания значений указателям.

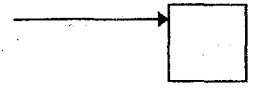
Динамическое выделение памяти можно осуществить с помощью процедуры **New(P)**, где P – переменная типа указатель (ссылка). Эта процедура выделяет память для размещения значений динамической переменной базового типа и присваивает указателю значение адреса выделенной области памяти.

```
New(Pj);
```


Чтобы обратиться к динамической переменной, необходимо применить операцию "разыменования" к указателю на эту переменную, используя знак операции "^" после имени указателя. Например, после выделения области памяти для хранения значений динамической переменной, ей можно присвоить значение оператором:

```
Pj^:=3;
```


Чтобы освободить выделенную область памяти для использования другими динамическими переменными нужно воспользоваться процедурой `Dispose(Pj)`, которая является обратной, по отношению к процедуре `New(Pj)`:

```
Dispose(Pj);
```


После освобождения выделенной памяти необходимо очистить указатель, ссылавшийся на удаленную динамическую переменную. Это выполняется присваиванием указателю "несуществующего" нулевого адреса `nil`:

```
Pj:=nil;
```


При работе с динамическими переменными важно помнить, что любая память, выделенная процедурой new, должна быть освобождена в программе процедурой *DISPOSE* !

Динамическая память чаще всего используется для хранения табличных данных. При этом строки таблиц называются записями и описываются как переменные комбинированного типа (типа `record`), а столбцы описываются как поля соответствующих типов, принадлежащие этим записям. В записях также предусматриваются поля указателей (адресов в памяти) последующих и предыдущих записей для организации связей с ними.

В зависимости от количества ссылочных полей в каждой записи (элементе списка) списки делятся на нуль-, одно-, двух- и многосвязные. Если у списка имеются концевые элементы, он называется линейным, если последний элемент списка связан с первым (или с заголовком) – список называется кольцевым.

В языке Паскаль для списков, в отличие от массивов и структур, нет специального ключевого слова для описания переменных типа списка. Их

создают с помощью комбинированных записей, содержащих ссылочные поля.

Ниже приведены схемы различных видов списков записей. На них символами "*" отмечены поля ссылок. Стрелками показаны связи между записями. Поля данных заглавных звеньев обычно используются для хранения общей информации о списках.

Односвязные списки

Пример организации односвязного списка приведен ниже.

```
Type
  Z=Record      {комбинированный тип для данных}
    a: String;      {строковое поле}
    b, c: Integer; {поле целых чисел}
    d: Real         {поле вещественных чисел}
  end;
  P=^S;         {тип указатель записи базового типа S}
  S=Record      {базовый тип для указателей типа P}
    ls:P;       {поле типа P ссылки на следующую запись}
    Dt:Z            {поле типа Z записи данных}
  end;
```

В этом примере типы Z и S введены для описания переменных – записей, содержащих в своих полях a, b, c, d данные, соответствующие описанным типам полей. P – тип указатель для динамических переменных базового типа S, т.е. значения переменных типа P будут адресами переменных типа S.

Наличие у комбинированной переменной типа S адресного поля ls типа P позволяет включать в состав записи ссылку на последующую запись и хранить таблицы в памяти машины в виде динамических списков связанных записей.

```
Var
  Dt:Z; {запись данных}
  Uz,U:P {указатели заглавного и текущего звеньев списка}
```

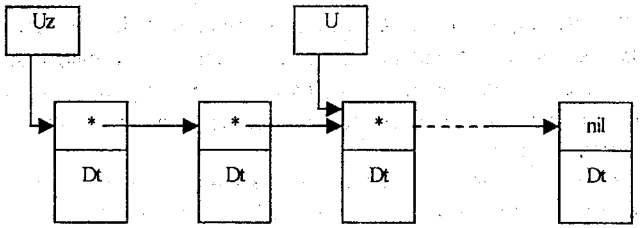


Рисунок 12. Односвязный линейный список

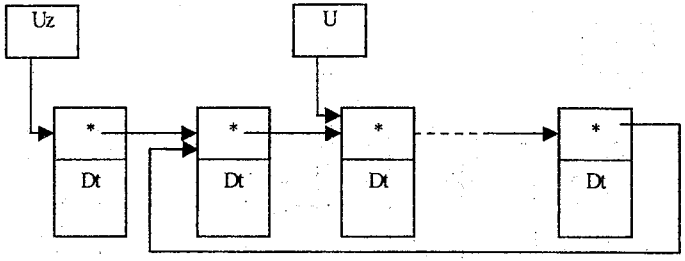


Рисунок 13. Односвязный кольцевой список с заголовком вне кольца

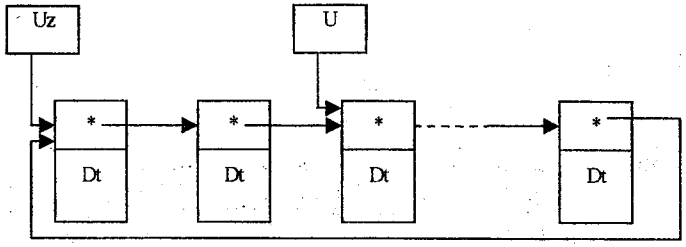


Рисунок 14. Односвязный кольцевой список с заголовком в кольце

В двусвязных списках базовый комбинированный тип `S` для указателей типа `P` будет иметь два адресных поля: поле `ls` ссылки на следующую запись списка и поле `lp` ссылки на предыдущую запись списка. Описание двусвязных списков аналогично приведенному выше (для односвязных списков) с отличием в структуре S:

```
P=^S;          {тип указатель записи базового типа S}
S=Record  {базовый тип для указателей типа P}
  ls: P;    {поле типа P ссылки на следующую запись}
  lp: P ;   {поле типа P ссылки на предыдущую запись}
  Dt: Z     {поле типа Z записи данных}
end;
```

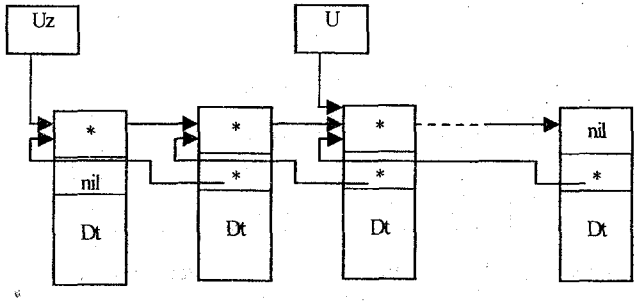


Рисунок 15. Линейный двусвязный список

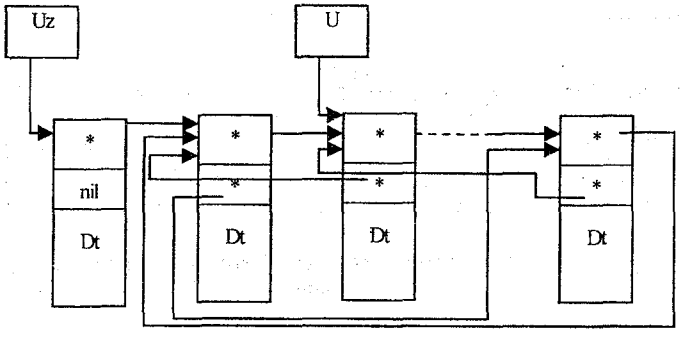


Рисунок 16. Кольцевой двусвязный список с заголовком вне кольца

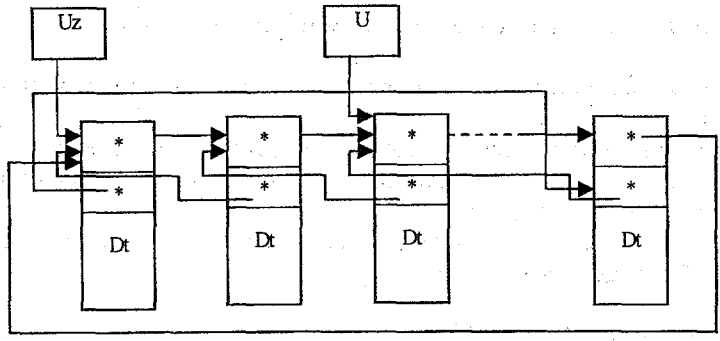


Рисунок 17. Кольцевой двусвязный список с заголовком в кольце

При работе с одно- и двусвязными списками нужно уметь добавлять элементы в любое место списка, удалять заказанный элемент из списка и перемещаться по списку с целью поиска или выбора информации из списка. Необходимо определять, пуст ли список в данный момент.

Обычно считается, что список пуст, если в нем имеется только звено заголовка и нет ни одного звена с данными. При работе с такими списками в начале необходимо создать заголовок (процедурой New), а в конце программы этот заголовок должен быть удален процедурой Dispose. Добавление звена в список и удаление из списка обычно оформляется в виде процедур. Перемещение по списку чаще всего выполняется с помощью итеративного цикла ("...Пока не достигнем звена с нужными данными или не просмотрим весь список..."). Ниже приведены некоторые примеры процедур для разных видов списков. Проверка "пуст ли список" обычно нужна для того, чтобы упростить процедуры работы со списками, так как с пустыми списками не все операции возможны.

Для линейных и кольцевых списков с заголовком вне кольца, проверяется ссылка из заголовка на следующее звено. Если `Uz^.ls=nil`, то список пуст. Для кольцевых с заголовком в кольце – если `Uz^.ls=Uz` , то список пуст.

Пример процедуры добавления элемента в линейный односвязный список (определяемый указателем `Uz`) после звена, заданного текущим указателем (U).

```
procedure PutS1(Var U:P;E:Z);
var
  q:P1;    { Завели рабочий указатель }
begin
  new(q); { Создаем новое звено, пока вне списка }
  q^.Dt:=E; { заполняем его информацией }
```

```
  q^.ls:=U^.ls; {подцепляем к новому звену конец списка за U
}
  U^.ls:=q {подцепляем к началу списка (до U) новое звено}
    end;
```

Если добавлять элемент нужно в начало списка (на которое ссылается заголовок списка), при вызове процедуры в качестве первого параметра приводится указатель заголовка списка.

Пример процедуры удаления элемента, заданного текущим указателем (U) из линейного двусвязного списка.

```
procedure DelS2(Var U:P);
var
  q:P;    { Завели рабочий указатель }
begin
  q:=U^.lp; {q указывает звено перед тем, на которое указывает
U }
  q^.ls:=U^.ls; {в звене q^ меняем ссылку на следующее звено -
за U^}
  if U^.ls<>nil then {если удаляемое звено - не последнее, то
}
  begin
    q:=U^.ls; { в звене за удаляемым }
    q^.lp:=U^.lp { меняем ссылку на предыдущее звено }
  end;
  dispose(U); { собственно удаление звена }
  U:=nil       { очистка указателя на удаленное звено }
end;
```

После срабатывания этой процедуры текущий указатель перестает указывать на что-либо (равен nil).

Пример процедуры удаления элемента, заданного текущим указателем (U) из кольцевого двусвязного списка.

```
procedure DelS2K(Var U:P);
    var
      q:P;
    begin
      q:=U^.lp;
      q^.ls:=U^.ls;
      q:=U^.ls;
      q^.lp:=U^.lp;
      dispose(U);
      U:=nil;
    end;
```

Выбор данных из списка производится без изменения списка, путем ссылки на информационную часть нужного звена: `U^.Dt`. Чтобы добраться

до нужного звена списка, по нему приходится последовательно перемещаться (в цикле, начиная от начала или от текущего звена). Так же как при работе с массивом, для перехода от элемента `A[i]` к следующему элементу необходимо выполнить `i:=i+1`.

При работе со списком следует использовать оператор `U:=U^ls`.

Создав список (в цикле, используя процедуру добавления звена) и поработав с ним, необходимо в конце программы удалить его (также, в цикле, используя процедуру удаления элемента списка, пока он не станет пустым). После этого не забыть удалить заголовок списка.

Нульсвязные списки

К таким спискам относятся ***стек, очередь*** и ***дек***. В отличие от прочих типов списков, по которым можно перемещаться, используя находящиеся в звеньях указатели, в нульсвязных списках доступны только элементы на одном или обоих концах. Только после удаления из такого списка доступного элемента, можно получить доступ к следующему элементу, который теперь становится крайним в списке.

В нульсвязных списках нет заглавных звеньев. У них есть только начало и конец. Началом (или начальным звеном) называется звено, которое можно взять из списка, концом – звено, после которого можно добавить в список новый элемент (новое звено).

В языке Паскаль нет средств, позволяющих описывать переменные типа нульсвязных списков. Их приходится моделировать односвязными списками с запретом перемещения по указателям звеньев. Работа с нульсвязными списками выполняется, только используя специальные процедуры и функции. Обычно достаточно использовать функцию проверки не пуст ли список и две процедуры: добавления элемента к списку и выбора элемента из списка (с удалением выбираемого звена).

Рассмотрим варианты нульсвязных списков.

Стек – это нульсвязный список, иногда называемый очередью, с правилом обслуживания `LIFO` (Last In – First Out – последним пришел – первым ушел). У стека начало и конец – это одно и то же звено, обычно называемое вершиной стека.

Создание пустого стека – это, по сути, создание указателя на вершину стека (`N`), и присвоение ему значения `nil`. Этот указатель в дальнейшем играет роль адреса начала и конца стека текущего звена.

Для добавления элемента данных в стек необходимо составить процедуру, получающую в качестве параметров указатель на вершину стека `N` и собственно данные `Dt`. Процедура должна создать динамическую переменную типа звена стека, в поле указателя перенести значение из указателя вершины стека, в поле данных записать значения `Dt` и в указатель на вершину

стека записать адрес созданной динамической переменной. Процедура должна вернуть эту новую вершину стека.

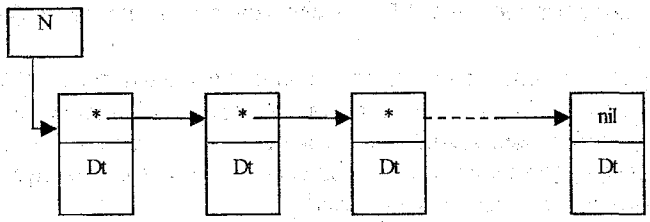


Рисунок 18. Стек

Процедура выбора из стека последнего введенного звена выполняется наоборот: в переменную для выбираемых данных переносятся значения поля `Dt`, новое значение указателя на вершину стека берется из поля ссылки на следующее звено, а старая вершина стека удаляется процедурой `Dispose`. В обоих процедурах используется рабочий указатель на звено стека. Описатели типов для стека полностью совпадают с описателями односвязного списка.

Очередь – нульсвязный список с правилом обслуживания `FIFO` (First In – First Out – первым пришел – первым ушел).

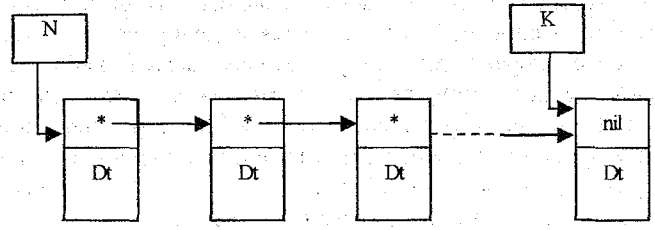


Рисунок 19. Очередь

Для работы с очередью необходимы такие же процедуры, как и для стека. При создании пустой очереди указателям `N` и `K` присваивается значение `nil`. Процедурам добавления в очередь и выбора из очереди приходится передавать указатели на оба конца, так как при создании первого звена, его адрес необходимо занести в оба указателя, а при удалении последнего звена очереди, обоим указателям надо присвоить значение nil.

Дек – это нульсвязный список, в котором добавление и выбор элементов возможен с любого конца. Его можно рассматривать как двусторонний стек или двустороннюю очередь. Название (deque) расшифровывается как double-ended queue – очередь с двумя концами.

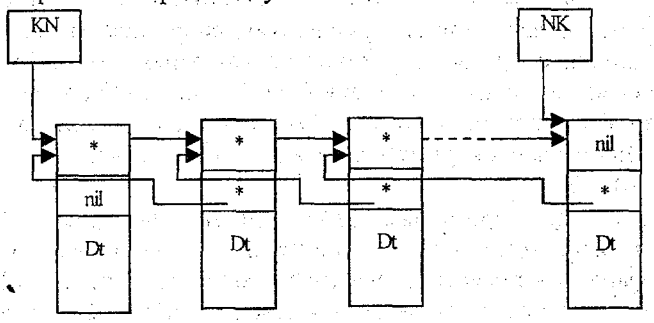


Рисунок 20. Дек

Проще всего дек моделировать двусвязным линейным списком, по которому запрещено перемещаться. Формально для него приходится составлять две процедуры добавления и две процедуры удаления элементов, находящихся на первом и втором концах дека. На практике процедуры добавления можно объединить в одну, используя дополнительный признак – к какому концу добавляется элемент. Хотя оба конца дека равноправны, для определенности одну сторону будем называть началом, обозначая указателем NK, а вторую –концом, с обозначением KN.

Обе процедуры удаления элемента также можно объединить в одну. В процедуру добавления (удаления) элемента таким образом придется передавать указатели на оба конца списка, данные (или адрес, куда их поместить при удалении) и признак, в начало ли идет добавление. В зависимости от того, какая сторона будет указана в списке параметров первой, с той и будет производиться работа. Вторая сторона необходима при занесении первого или удалении последнего звена из дека. Примеры этих процедур разобраны в контрольном варианте № 31.

Примеры процедур по добавлению и удалению элементов нульсвязных списков приведены в разобранном ниже контрольном варианте.

Описание файлов с данными

В данной работе предлагается использовать два типа файлов с исходными данными. Оба файла содержат одну и ту же информацию, но хранят ее в разной форме. Файл с именем `Dan.dat` представляет собой типизированный файл, каждая запись которого представляет структуру фиксированного типа и размера. Структура каждой записи представлена ниже:

```
1-ое поле - string[15];    { поле для Фамилии И.О.}
```

```
2-ое поле - integer; {поле балла за 1 семестр}
2-ое поле - integer; {поле балла за 2 семестр}
2-ое поле - real;    {поле для среднего балла }
```

Файл с именем `Dan.txt` является последовательным символьным файлом, в котором записи имеют различную длину. Каждая запись содержит фамилию, инициалы (в форме символ–точка–символ–точка без пробелов), две целочисленные оценки и средний балл, в виде изображения вещественного числа с фиксированной точкой. Каждое данное отделяется от соседних пробелом. Пример записи файла `Dan.txt` приведен ниже:

```
Петров П.П. 5 3 4.0
```

Работа с типизированным файлом достаточно проста. Описывается структура записи, содержащая указанные выше поля, Описывается типизированный файл, с записями такого типа (например с именем F), заводится переменная того же типа (например, с именем STUD). После открытия файла для чтения, оператором Read(F,STUD) в структуру STUD считывается информация об очередном студенте.

Работа с текстовым файлом осложнена тем, что в одной записи файла содержится как строка с именем и инициалами студента, так и числовая информация. Кроме того, длина фамилий в разных записях различна (но не больше 15 символов с инициалами). Организовать процедуру чтения одной записи можно по разному. Например, можно прочесть всю запись в рабочую строку, затем найти в ней, где начинается первое число (символ из диапазона '0'..'9'), все до этого символа скопировать в строку – фамилию. Из остатка взять один символ и превратить в число стандартной процедурой **`Val(...)`**, пропустить символ (пробел) и также получить вторую целочисленную оценку. Наконец, еще раз пропустив пробел, вырезать изображение среднего балла и процедурой **`Val(...)`** получить вещественное значение среднего балла. Все вырезаемые данные присваивать полям структуры данных, описывающих информацию о студенте.

В приведенном разборе контрольного варианта рассмотрен другой алгоритм чтения. Его суть заключается в следующем. Подготавливается рабочая строка длиной не менее 15 символов, для чего она заполняется каким-нибудь символом, в количестве 15 штук. Таким образом, в нулевом байте этой строки будет число 15 – текущая длина строки. Затем из записи файла читается по одному символу и заносится на соответствующее место в рабочую строку. Цикл прекращается, когда будет прочитан второй пробел (отделяющий инициалы от первой оценки). Так как при чтении ведется подсчет количества символов, в конце цикла известна длина фамилии. Из рабочей строки вырезается подстрока найденной длинны начиная от первого символа и копируется стандартной строковой функцией **`Copy(...)`** в поле структуры для фамилии студента.

Остаток входной записи читается обычным оператором **`Readln`** в две целочисленные и одну вещественную переменные для оценок (в поля струк-

туры). Отметим, что в качестве рабочей строки можно использовать само строчное поле структуры.

В этом случае операторы чтения одной записи можно записать, например, так:

```
  i:=0; { счетчик символов }
  P:=0; { счетчик пробелов }
  Stud.Name:='---------------'; {поле фамилии заполняем
строкой из прочерков на всю длину }
  while P<2 do   {пока не прочтем второй пробел }
  begin
    inc(i); { стандартная процедура увеличения на 1 }
    read(F,Stud.Name[i]);  {читаем из файла по символу в
поле имени }
    if Stud.Name[i]=' ' then P:=P+1; { подсчет пробелов}
  end;
  Stud.Name:=Copy(Stud.Name,1,i);  { Из полной строки по-
ля фамилии вырезаем подстроку по второй пробел (включи-
тельно) и заносим обратно в поле фамилии }
  readln(F,Stud.Bal1,Stud.Bal2,Stud.SrBal);  { остаток
записи файла читаем в поля оценок }
```

Разбор контрольного варианта

Задание

Таблица 31. Данные к заданию 31 варианта

№ вар.	Требования к записям, выбираемым из файла и помещаемым в список	Тип списка	Файл данных
31	В начало – только с пятерками, в конец – только с тройками	S0D	dan.txt

На основе общего задания и данных таблицы вариантов конкретное задание формулируется следующим образом:

Заполнить нульсвязный список *дек* данными из файла *DAN.TXT*, заносяя в начало дека записи о студентах, у которых все оценки – 5 баллов, а в конец дека – сведения о студентах, имеющих все оценки – 3 балла. Добавить в начало дека запись, в которой вместо фамилии указан тип списка, а вместо первой оценки – число записей с информацией в деке. Вывести в выводной текстовый файл таблицу записей из дека. Созданный дек удалить из памяти.

Содержание программы

Задание включает в себя следующие действия, подлежащие программированию:

- Формирование пустого дека;
- Открытие входного и выходного файлов;
- Чтение данных из файла с занесением нужных записей в дек с подсчетом их количества;
- Добавление информационной записи в начало дека;
- Вывод в выходной файл информационной записи из дека с удалением этой записи;
- Вывод "шапки" таблицы в выходной файл;
- Удаление из дека записей с выводом информации в табличном виде в выходной файл;
- Закрытие выходного файла.

При составлении программы, формирование дека и его распечатку в выходной файл оформим в виде процедур. Кроме того, отдельными процедурами оформим процессы добавления записи в дек и удаления (выбора) записи из дека.

Учитывая, что файл текстовый, воспользуемся процедурой чтения строки файла с распределением информации по полям структуры, как это рассмотрено в пояснениях выше.

Алгоритм

Укрупненные шаги алгоритма основной программы приведены под заголовком "содержание программы" (при оформлении отчета, алгоритмы следует включить в блок-схему программы). Рассмотрим строение процедур.

Добавление записи в дек. Исходя из схемы дека, приведенной на рис. 20, будем считать его пустым, если в нем нет ни одной записи, то есть указатели начала и конца дека указывают на `nil`. Если запись добавляется в пустой дек, то меняются оба указателя, если в непустой – меняется только один указатель (начала или конца). Проверку, что дек пуст, можно выполнять по содержимому любого указателя (`nil` или нет). Добавление записей к обоим концам выполняется почти одинаково: создается динамическая переменная типа звена дека, заполняется ее информационная часть, обе ссылки на соседние звенья делаются равными `nil`. Затем, в зависимости от того, к какому концу добавляется запись, меняется первая ссылка в новой записи (прицепляясь к деку) и вторая ссылка в концевой записи дека (связываясь с новой записью) или наоборот – вторая ссылка новой записи и первая ссылка конца дека. Наконец меняется указатель конца дека к которому добавлена запись.

Удаление записи из дека. В задаче требуется выбирать данные из дека с одной стороны, но для универсальности, составим процедуру выбора и удаления с заказанного конца дека. Как и в предыдущей процедуре, будем использовать логический параметр, равный истине, при работе с началом и лжи, при работе с концом дека. Если начало и конец ссылаются на одну и ту же область памяти (равны между собой), то при удалении записи дек становится пустым. Если нет, процесс выполняется в обратном порядке, по сравнению с добавлением записи.

Заполнение дека из текстового файла. Представляет собой цикл (пока не достигнут конец файла) чтения очередной строки текстового файла с заполнением информационной структуры, которую либо добавляют к началу дека (обе оценки – 5), либо к концу (обе оценки – 3), либо просто пропускают. Внутри цикла используются процедуры ввода одной строки и добавления записи в дек. При заполнении дека ведется подсчет количества записей дека.

Распечатка дека в выходной текстовый файл. Так как количество записей дека известно, вывод выполняется в форме арифметического цикла. Тело цикла содержит обращение к процедуре выбора записи из дека (из начала) и форматного вывода данных в выводной файл в виде строки таблицы.

При составлении блок-схем алгоритмов в отчете по лабораторной работе можно пользоваться именами переменных, описания которых предварительно заданы в таблице идентификаторов основной программы и подпрограмм.

Примечание: формальные параметры, передаваемые по имени (которым предшествует ключевое слово `var`), являются, по сути, указателями на данные, поэтому занимают в памяти по 4 байта, независимо от типа данных, на которые указывают.

Таблица идентификаторов

Таблица 32. Идентификаторы программы 31 варианта

Имя	Тип	Р-р, байт	Назначение
Основная программа			
Lab_9	имя программы	-	Работа с динамическими списками
data	описатель	-	Описатель типа для структуры данных
.Name	строка	16	Поле структуры данных для фамилии
.Bal1	целое	2	Поле структуры данных для первой оценки
.Bal2	целое	2	Поле структуры данных для второй оценки
.SrBal	вещественное	6	Поле структуры данных для средней оценки.
Pd	описатель	-	Описатель типа для указателей на звено дека

Имя	Тип	Р-р, байт	Назначение
Dek	описатель	-	Описатель типа для структуры данных записи дека
P1	указатель	4	Ссылка на следующий элемент дека
P2	указатель	4	Ссылка на предыдущий элемент дека
Student	структура	26	Структура – составное поле данных записи дека
Docum	структура	26	Данные одной записи
DN	указатель	4	Указатель первого конца дека (начала)
DK	указатель	4	Указатель второго конца дека (конца)
Fin	последовательн. символьн. файл	128	Входной текстовый файл с данными
Fout	последовательн. символьн. файл	128	Выходной текстовый файл с таблицей результатов
k	целое	2	Количество элементов дека
GetStud – процедура выбора строки из входного файла			
St	указатель на структуру	4	Формальный параметр – адрес структуры с данными по студенту
F	указатель на файл	4	Формальный параметр – имя входного файла
P	целое	2	Счетчик пробелов во входной записи
i	целое	2	Порядковый № символа во входной записи
PutDek Процедура формирования нового звена дека			
NK	указатель	4	Формальный параметр – адрес конца дека, к которому добавляют запись
KN	указатель	4	Формальный параметр – адрес второго конца дека
Inf	структура	24	Формальный параметр – добавляемые данные
Beg	логическое	1	Признак, что добавляется к началу
U	указатель	4	Вспомогательный указатель на добавляемое звено дека
DelDek Процедура удаления крайнего звена дека			
NK	указатель	4	Формальный параметр – адрес конца дека, у которого удаляют запись
KN	указатель	4	Формальный параметр – адрес второго конца дека
Inf	указатель на структуру	4	Формальный параметр – адрес структуры, куда поместить выбираемые данные
Beg	логическое	1	Признак, что удаляется из начала
U	указатель	4	Вспомогательный указатель на удаляемое звено дека
ReadFile Процедура чтения записи из файла и заполнения дека			
F	указатель на файл	4	Формальный параметр – указатель на входной файл для входных данных

Имя	Тип	Р-р, байт	Назначение
DekN	указатель	4	Формальный параметр – адрес указателя на начало дека
DekK	указатель	4	Формальный параметр – адрес указателя на конец дека
N	указатель	4	Формальный параметр – адрес переменной для числа элементов дека
Stud	структура	26	Рабочая структура для данных о студенте

Разработанный алгоритм с использованием перечисленных идентификаторов реализуется на языке Турбо-Паскаль приведенной ниже программой.

Текст программы

```
Program Lab_9;
{   Программа Лабораторной работы N 9
    Динамические переменные. Списки.
          Вариант N 31.
    А.Я.Умненькая, ст. гр. Я-007}
TYPE data = record {описатель структуры данных студента}
       Name : string[15];    { поле для Фамилии И.О.}
       Bal1,Bal2 : integer; {поля баллов за 2 семестра}
       SrBal : real;         {поле для среднего балла }
     end;
     Pd=^Dek;    {описатель указателей на вершины дека}
     Dek= record   { описатель звена дека}
       P1:Pd;  {поле указателя следующего звена от начала}
       P2:Pd;  {поле указателя следующего звена от конца }
       Student: data; { поле данных студента}
     end;
VAR
  Docum: data;  {рабочая структура данных о студенте }
  DN,DK : Pd;     { указатели на начало и конец дека }
  Fin,Fout:text; {файлы входных данных и результатов работы}
  k:integer;      { количество элементов дека}
Procedure GetStud(Var F:text; var St:data);
   {процедура чтения одной записи файла и формирования
данных студента}
Var
  P,i:integer;
Begin
  i:=0;
  P:=0;
```

```
  St.Name:='---------------'; поле фамилии заполняем
строкой из прочерков максимальной длины
  while P<2 do   {пока не прочтем второй пробел }
  begin
    inc(i);
    read(F,St.Name[i]);  {читаем из файла по символу в
поле имени }
    if St.Name[i]=' ' then P:=P+1; { подсчет пробелов}
  end;
  St.Name:=Copy(St.Name,1,i);  { Из полной строки поля
фамилии вырезаем подстроку по второй пробел (включитель-
но) и заносим обратно в поле фамилии }
  readln(F,St.Bal1,St.Bal2,St.SrBal);  { остаток записи
файла читаем в поля оценок }
End;
Procedure PutDek(Var NK,KN:Pd; Inf:data; Beg:boolean);
 {процедура добавления элемента в дек с заказанного конца}
Var U:Pd;
Begin
  New(U);
  U^.Student:=Inf;
  U^.P1:=nil;
  U^.P2:=nil;
  if NK=nil then  { если дек перед этим был пуст }
    KN:=U
  else
    if Beg then  { если добавляем в начало }
    begin
      U^.P1:=NK;
      NK^.P2:=U;
    end
    else   { если добавляем в конец }
    begin
      U^.P2:=NK;
      NK^.P1:=U;
    end;
  NK:=U;
End;
Procedure DelDek(Var NK,KN:Pd; var Inf:data;
Beg:boolean);
{процедура выбора элемента из заказанного конца дека }
Var U:Pd;
Begin
  U:=NK;
```

```
  Inf:=U^.Student;
  if NK=KN then    { если в деке был всего один элемент}
  begin      { делаем дек пустым}
    KN:=nil;
    NK:=nil;
  end
  else
    if Beg then  { если удаление из начала }
    begin
      NK:=U^.P1;
      NK^.P2:=nil;
    end
    else  { если удаление из конца }
    begin
      NK:=U^.P2;
      NK^.P1:=nil;
    end;
  Dispose(U);  { собственно освобождение памяти от эле-
мента }
End;
Procedure ReadFile(Var F:text; Var DekN,DekK:Pd; var
N:integer);
  { чтение файла с заполнением дека }
Var
  Stud:data;
Begin
  N:=0;
  While Not Eof(F) do
  begin
    GetStud(F,Stud);
    N:=N+1;
    if (Stud.Bal1=5) and (Stud.Bal2=5) then
      PutDek(DekN,DekK,Stud,TRUE)
    else
      if (Stud.Bal1=3) and (Stud.Bal2=3) then
        PutDek(DekK,DekN,Stud,FALSE)
      else  { если данные не заносим в дек }
        N:=N-1;
  end;
End;
Procedure WriteFile(Var F:text; Var NK,KN:Pd; N:integer);
  { процедура распечатки дека в выводной файл с удалением
дека}
Var
```

```
  i:integer;
  Stud:data;
Begin
  Writeln(F,'|-------------------------------------',
          '--------------|');
  Writeln(F,'| N |  Фамилия И.О.  | 1-й балл | 2-й ',
          'балл | Ср.балл |');
  For i:=1 to N do
  begin
    Writeln(F,'|---|----------------|----------|----',
            '------|---------|');
    DelDek(NK,KN,Stud,TRUE);
    WriteLn(F,'|',i:2,' | ',Stud.Name:15,'|',Stud.Bal1:6,
          '    |',Stud.Bal2:6,'    |', Stud.SrBal:6:1,'   |');
  end;
  Writeln(F,'|-------------------------------------',
          '--------------|');
End;
BEGIN    { ОСНОВНАЯ ПРОГРАММА }
  Assign(Fin,'Dan.txt');
  Reset(Fin);
  Assign(Fout,'Umnik9.res');
  ReWrite(Fout);
  DN:=nil;
  DK:=nil;
  k:=0;
  ReadFile(Fin,DN,DK,k);
  Close(Fin);
  With Docum do
  begin
    Bal1:=k;
    Bal2:=0;
    SrBal:=0.0;
    Name:='Список типа SOD';
  end;
  PutDek(DN,DK,Docum,TRUE); { добавление в начало дека записи со
сводной информацией. }
  DelDek(DN,DK,Docum,TRUE); {Выбор из дека сводной информации
для печати }
  Writeln(Fout, Docum.Name:20,' из ',Docum.Bal1,' строк');
  WriteFile(Fout,DN,DK,k); {печать дека в файл с удалением эле-
ментов дека}
  close(Fout);
End.
```

Варианты заданий

Таблица 33. Варианты заданий лабораторной работы № 9

№ вар.	Требования к записям, выбираемым из файла и помещаемым в список	Тип Списка	Файл данных
1	Средний балл выше 3	S2KI	dan.dat
2	Отличные оценки во всех полях	S2KO	dan.txt
3	Первые буквы фамилий А, Б, В	S0S	dan.dat
4	Оценки во всех полях ниже 4 баллов	S0O	dan.txt
5	Первая оценка выше 3	S1L	dan.dat
6	Первая или вторая оценка выше 3	S1KI	dan.txt
7	Первая оценкой ниже 5, а вторая выше 3	S1KO	dan.dat
8	Первые буквы фамилий А, В, Е и средние оценки выше 3	S2L	dan.txt
9	Оценки 3 во всех полях оценок	S2KI	dan.dat
10	Средние оценки выше 3 баллов	S2KO	dan.txt
11	Вторая оценка выше 4, а первая выше 3 баллов	S0S	dan.dat
12	Первая и вторая оценки выше 4 баллов	S0O	dan.txt
13	В начало – со средней оценкой <4 балла, в конец – прочие	S0D	dan.dat
14	Первые буквы фамилий А, Б, В и средние оценки выше 4	S1L	dan.txt
15	Вторые и средние оценки выше 4 баллов	S1KI	dan.dat
16	Первая оценка выше 3 баллов, а средняя оценка выше 3,5	S1KO	dan.txt
17	Все оценки не ниже 4 баллов	S2L	dan.dat
18	Вторая и средняя оценки выше 4 баллов	S2KI	dan.txt
19	Средняя оценка выше 3,5 баллов	S2KO	dan.dat
20	Средняя оценка ниже 4,5 и фамилии начинаются с букв А и В	S0S	dan.txt
21	Средняя оценка выше 4 баллов и фамилии, начинаются с букв от А до К	S0O	dan.dat
22	В начало – если первая оценка 3, в конец – если первая – 5	S0D	dan.txt
23	Первая оценка выше 3 балов, а средняя оценка 4 балла	S1L	dan.dat
24	Все оценки выше 4 и фамилии начинаются с букв А – Е	S1KI	dan.txt
25	Первая и средняя оценки выше 4 баллов	S1KO	dan.dat
26	Все оценки ниже 5 баллов	S2L	dan.txt
27	Первая или вторая оценка выше 4 баллов	S2KI	dan.dat
28	Первая или средняя оценки ниже 4 баллов	S2KO	dan.txt
29	Средняя оценка выше 4 баллов	S0S	dan.dat
30	Все оценки выше 4 или фамилии начинаются с букв А – Е	S0O	dan.txt
31	В начало – только с пятерками, в конец – только с тройками	S0D	dan.txt

Литература

1. *Бородич Ю.С., Вальвачев А.Н., Кузьмич А.И.* Паскаль для персональных компьютеров: Справочне пособие. Минск: "Вышейшая школа", 1991.
2. *Епанешников А.М., Епанешников В.А.* Программирование в среде Turbo Pascal 7.0 (третье издание). М.: "Диалог-МИФИ", 1996.
3. *Немнюгин С.А.* Turbo Pascal: практикум. СПб.: Питер, 2001.
4. *Кнут Д.* Искусство программирования для ЭВМ. /Пер. с англ.: В 3-х томах. т.3. Сортировки и поиск. М.: Мир, 1976.
5. *Бронштейн И.Н., Семендяев К.А.* Справочник по математике для инженеров и учащихся ВТУЗов (Изд. 13-е). М.: "Наука", 1986.

Приложение А.
Система меню и команды Турбо-Паскаля

Меню системы программирования Турбо-Паскаль

Для взаимодействия с системой программирования Турбо-Паскаль можно использовать специальную интегрированную среду этой системы. ***Интегрированная среда – это совокупность взаимосвязанных сервисных программных средств, обеспечивающих всестороннюю поддержку процесса разработки программ.*** Умелое использование возможностей среды Турбо-Паскаля позволяет значительно повысить эффективность этапов проектирования, тестирования и отладки программ.

Главное меню

При входе в интегрированную среду системы программирования Турбо-Паскаль (для этого достаточно вызвать модуль turbo.exe), сразу становится доступным главное меню, которое расположено в самой верхней строке экрана. Это единственное меню системы программирования Турбо-Паскаль, в котором пункты меню (опции, команды) расположены горизонтально. Опции – это содержащиеся в меню альтернативные варианты действий. В главном меню их десять: Ё, File, Edit, Search, Run, Compile, Debug, Options, Windows и Help. Каждая из опций имеет свое собственное меню. В отличие от главного эти дополнительные меню выпадающие, т.е. они расположены вертикально. Для перемещения курсора по опциям главного и дополнительных меню используются клавиши–стрелки. Для выполнения команды меню, выделенной подсветкой, необходимо нажать клавишу "Enter". Чтобы вернуться в главное меню, достаточно нажать клавишу F10 или Esc.

Выполнить некоторые часто используемые команды дополнительного меню можно, минуя главное меню, с помощью так называемых "горячих клавиш". За некоторыми клавишами или комбинациями клавиш в среде Турбо-Паскаль закреплены постоянные функции; таким образом "горячие" клавиши доступны практически всегда. Список всех горячих клавиш и соответствующих им команд приведен в таблице 35 настоящего приложения. В нижней строке экрана постоянно присутствует подсказка о назначении часто используемых горячих клавиш.

Таблица 34 Пункты главного меню Турбо-Паскаля

Ё	File	Edit	Search	Run	Compile	Debug	Options	Window	Help
About	Open... *F3*	Restore line	Find...	Run *Ctrl–F9*	Compile *Alt–F9*	Evaluate/modify... *Ctrl–F4*	Compiler...	Size/Move *Ctrl–F5*	Contents
Show version	New	Cut *Shift–Del*	Replace...	Program reset *Ctrl–F2*	Make *F9*	Watches ⇒	Memory sizes...	Zoom *F5*	Index *Shift–F1*
Clear desktop	Save as...	Paste *Shift–Ins*	Go to line number.	Trace into *F7*	Destination Disk	Breakpoints...	Debugger...	Cascade	Previous topic *Alt–F1*
Refresh display	Save *F2*	Copy *Ctrl–Ins*	Search again	Go to cursor *F4*	Build	Toggle breakpoint *Ctrl–F8*	Linker...	Tile	Topic search *Ctrl–F1*
	Save all	Copy example	Find procedure...	Step over *F8*	Primary file...		Directories...	Next *F6*	Help on help
	Change dir...	Show clipboard	Find error...	Parameters...			Environment ⇒	Previous *Shift–F6*	
	Print	Clear *Ctrl–Del*					Save options ...	Close *Alt–F3*	
	Get info...						Retrieve options..	Watch	
	DOS shell							Register	
	Exit *Alt–X*							Output	
								Call stack *Ctrl–F3*	
								User screen *Alt–F5*	
								List... *Alt–0*	

Команды опции Ё

About позволяет получить информацию об используемой версии системы программирования Турбо-Паскаль.

Refresh display – обновляет экран.

Clear desktop – закрывает все активные программы и очищает все списки.

Команды опции File.

Open: выбор и открытие файла с исходным текстом для редактирования.

После активизации опции Open на экране появляется диалоговое окно, в котором находится список файлов текущей директории и справа – ряд кнопок. Главными являются кнопки **[Open]** и **[Replace]**; первая загружает выбранный файл во вновь открываемое окно, вторая – в активное в данный момент окно редактирования. Если при использовании **[Replace]** новый выбранный файл загружается на место старого находящегося в окне файла, имя старого файла помещается в нижнюю часть меню опции **File**. В диалоговом окне опции Open открывается еще одно дополнительное окно – окно ввода. В это окно можно ввести имя необходимого Вам файла или группы файлов и работать именно с ним (с ними).

New – открытие нового окна редактирования и нового файла с именем NONAME< цифра > < цифра >.PAS.

Save – запись файла, находящегося в активном окне редактирования, на диск. Если файл имеет системное имя, начинающееся с NONAME, то система перед записью попросит переименовать файл.

Save As – переименование файлов, находящихся в активном окне редактирования, при записи на диск.

Save All – запись на диск всех файлов, находящиеся не только в активном окне, но и во всех остальных открытых к данному моменту окнах редактирования.

Change Dir – изменение текущего устройства и (или) директории. Существует два способа смены директории: первый заключается в указании в окне ввода полного адреса новой директории; второй – в выборе нужной директории из появляющегося в диалоговом окне дерева директорий.

Print – распечатка содержимого активного окна редактирования.

Printer Setup – указание имени специальной программы–фильтра, обрабатывающей данные перед выводом их на печать. Например, можно предусмотреть выделение различных элементов синтаксиса при печати. Сама печать производится с помощью опции Print.

Get info – вывод на экран информации о состоянии системы – доступной оперативной памяти и запущенных программах.

DOS Shell – временный выход из интегрированной среды системы программирования Турбо-Паскаль без выгрузки ее из оперативной памяти.

После этого становятся доступными все стандартные средства операционной системы. Отметим, что доступная оперативная память сокращается приблизительно на 240 К. Для того чтобы вернуться в среду Турбо-Паскаль, достаточно набрать на клавиатуре команду EXIT.

Exit – выход из среды Турбо-Паскаля и удаление ее из оперативной памяти компьютера. Если перед выбором этой опции не все редактируемые файлы были сохранены, система предложит их сохранить.

Команды опции Edit.

Команды этого режима предназначены для различных операций с редактируемыми текстами – выделение фрагментов текста, удаления, копирования и перемещения их в любое нужное место. Выделение фрагментов текста производится как с помощью клавиатуры, так и с помощью мыши. Кроме того, для перемещения текстов из одного окна в другое можно использовать дополнительное окно Clipboard, называемое карманом. Тексты, помещенные в карман, так же могут редактироваться. Меню опции Edit состоит из семи команд.

Restore line – отмена всех изменений, внесенных последней операцией редактирования.

Cut – удаление выделенного фрагмента текста из окна редактирования и помещение его в карман.

Copy – помещение копии выделенного фрагмента текста в карман.

Paste – помещение текста из кармана (или выделенного фрагмента текста из окна Clipboard) в то место активного окна, которое указано курсором.

Copy example – помещение копии фрагмента текста из окна помощи в карман.

Show Clipboard – открытие окна кармана. Окно кармана похоже на окно редактирования. Единственное отличие заключается в том, что любой фрагмент, вырезаемый (*Cut*) или копируемый (*Copy*) из окна кармана, автоматически помещается в конец текста, в данный момент находящегося в кармане. Отметим, что опция Paste берет из кармана не весь текст, а только выделенный фрагмент.

Clear – удаление выделенного фрагмента без помещения его в карман. Эту опцию можно использовать для очистки кармана.

Команды опции Search.

Команды этого режима предназначены для поиска любой последовательности символов в редактируемых текстах.

Find – (поиск) – при выборе этой опции на экране появляется диалоговое окно, а пользователю предлагается ввести искомую последовательность символов и определить условия и область поиска. Условия поиска задаются с помощью нескольких кнопок – флажков. Флажки позволяют установить, следует ли при поиске:

– различать прописные и строчные буквы;

– анализировать только слова или знаки пунктуации то же;

– распознавать включаемые в искомую строку спецификаторы формата.

Кроме того, с помощью кнопок-переключателей определяется область и направление поиска: областью поиска может быть весь текст или выделенный фрагмент; начало или конец области может также помечаться курсором; направление поиска может быть либо прямым, т. е. от начала области к концу, либо обратным.

Replace – замена в тексте одних последовательностей символов на другие. Диалоговое окно, которое появляется при выборе этой опции, очень похоже на соответствующее окно опции *Find*; исключением является дополнительное поле, в которое необходимо поместить строку замены. Если искомая строка найдена, система спрашивает, следует ли заменить только ее первое вхождение либо все вхождения сразу. Как и в случае опции *Find*, текст для поиска может быть взят из окна редактирования.

Search Again – установка тех же условий поиска, которые были сформированы в результате самого последнего вызова либо опции *Find*, либо опции *Replace* и проведение поиска.

Goto line number – поиск участка текста, содержащего строку с заданным номером. Найденный участок появляется в окне редактирования.

Find Error – определение места нахождения ошибки, возникающей во время выполнения программы. Если флаг *Debugging* установлен (см. меню опции *Option*), то при возникновении ошибки курсор автоматически будет помещен на строку, содержащую ошибочный оператор. Однако если это не так или если программа запущена не из среды Турбо-Паскаля, то ошибка локализоваться не будет (будет указан только адрес ошибочного оператора). В этом случае и следует использовать опцию *Find Error*. При выборе этой опции надо задать адрес ошибочного оператора в виде < сегмент >:< смещение>, а система определит его местонахождение.

Find Procedure – поиск в программе нужной процедуры или функции. Для инициализации поиска необходимо ввести имя подпрограммы.

Команды опции Run.

Команды опции *Run* позволяют проводить компиляцию, компоновку и выполнение программы, а также осуществлять прогоны программы в различных отладочных режимах.

Run – компиляция, компоновка и выполнение программы, находящейся в активном окне редактирования. Причем компилируются также модули, определяемые по правилам режима *Make* опции *Compile*. Если со времени последней компиляции исходный текст не претерпел изменений, программа сразу же будет выполнена.

Нажатие клавши *Ctrl–Break* приводит к приостановке процесса выполнения программы; курсор в этом случае будет установлен на строке, которая непосредственно следует за уже выполненной. Выполнение програм-

мы после этого можно продолжить. Повторное нажатие клавиш *Ctrl–Break* завершит работу программы.

Program Reset – освобождение памяти, отведенной для программы, закрытие всех файлов, используемых программой и прекращение текущего сеанса отладки.

Goto Cursor – вначале осуществляются все необходимые действия, связанные с компиляцией и компоновкой программы. После этого программа выполняется до строки, помеченной курсором (строка выделяется подсветкой). В этом режиме можно пользоваться всеми средствами встроенного отладчика.

Trace info – выполнение очередного оператора программы и приостановка выполнения программы. Следующая готовая к выполнению строка программы выделяется подсветкой. Если очередным оператором является обращение к подпрограмме, управление передается внутрь подпрограммы.

Step over – аналогично предыдущей опции, но имеется одно отличие: пооператорное сканирование подпрограмм с ее помощью не проводится.

Parameters – задается строка символов, которая будет интерпретироваться программой как набор ее входных параметров, аналогичных параметрам, задаваемых с помощью командной строки.

Команды опции Compile.

Выполняют компиляцию и компоновку Вашей программы, сформируют загрузочные файлы, а в случае необходимости сохраняют их на диске.

Compile – компиляция программы или модуля, находящегося в активном окне редактирования. После окончания компиляции на экране появляется окно, содержащее информацию о результатах компиляции. Если в тексте допущена синтаксическая ошибка, процесс компиляции прекращается, в окне редактирования появляется сообщение об ошибке, а курсор показывает место ошибки в тексте программы.

В том случае, если в тексте компилируемой программы (или модуля) имеется обращение к другим модулям, последние должны быть к этому моменту оттранслированы и храниться на диске в виде файлов с расширением .TPU.

Make – с помощью этой опции компилируются:

а) программа (модуль) либо находящийся в файле, имя которого указано в опции *Primary File*, либо присутствующий в активном окне редактирования;

б) все модули, которые удовлетворяют двум условиям: во-первых, вызываются откомпилированной программой (модулем) из пункта (а) и, во-вторых, изменялись с момента их последней компиляции;

в) модули, которые обращаются к определенным в пунктах (а) и (б) модулям, секции связи которых претерпели изменения. Для того, чтобы проверить, изменился ли исходный текст модуля с момента последней компиляции, дата создания файла с расширением .PAS сравнивается с датой создания

файла с тем же именем, но с расширением .TPU. Если файл с расширением .PAS не будет найден, то система воспользуется существующим файлом с расширением .TPU.

Все эти, на первый взгляд сложные правила направлены на то, чтобы упростить разработку больших программных систем, поскольку предусматривают перекомпиляцию только тех модулей, которые изменялись с момента последних испытаний системы.

Build –опция подобная опции *Make*, но предусматривающая перекомпиляцию всех модулей, определенных в пунктах (а), (б) и (в) вне зависимости от того, подвергались они изменениям с момента последней компиляции или нет.

Destination: позволяет определить, где будет размещаться загрузочный код программы или модуля в оперативной памяти или на диске. Эта опция может принимать одно из двух возможных значений MEMORY (память) или DISK (диск). Отметим, что даже если значением опции является MEMORY, все дополнительные TPU-файлы, сгенерированные в режимах RUN, MAKE и BUILD, будут записаны на диск. Если же значением опции является DISK, то на диске создается либо файл с расширением .EXE (для главной программы), либо файл с расширением .TPU (для модуля).

Primary File – указание имени файла с расширением .PAS, который будет обрабатываться в режимах RUN, MAKE и BUILD. Если поле *Primary File* пусто, то обрабатывается файл, содержащийся в активном окне редактирования. Чаще всего, хотя и не всегда, в этом поле следует задавать имя файла, содержащего текст главной программы.

Команды опции Debug.

В режиме *Debug* можно пользоваться большими возможностями отладчика, встроенного в среду программирования Турбо-Паскаль 6.0. К ним в первую очередь относятся средства работы с окнами наблюдений и средства временной остановки выполнения программы.

Evaluate/Modify – вывод на экран значения произвольного выражения в процессе отладки, а также просмотр и в случае необходимости изменение значения любой переменной. Выбор этой опции приводит к открытию диалогового окна, содержащего три поля: *Expression* (Выражение), *Result* (Результаты) и *New Value* (Новое значение). Поле *New Value* можно использовать только тогда, когда в окне *Expression* находится имя переменной. Для манипулирования полями диалогового окна предназначены две кнопки [*Evaluate*] (Вычислить) и [*Modify*] (Изменить). Заметим, что данные в окне *Result* выводятся в специальном жестком формате. Для изменения формы вывода можно использовать спецификаторы формата.

Watches – открытие и активизация окна наблюдений. Эта опция позволяет добавлять, редактировать и удалять окна наблюдений. Активизация *Add Watch* приводит к открытию диалогового окна, в котором необходимо поместить выражение, изменения значения которого отслеживаются в про-

цессе выполнения программы. Это выражение будет помещено в окно наблюдений. Окно наблюдений появляется на экране после нажатия клавиши F6.

Toggle breakpoint – установка/снятие точки останова в строке, где находится курсор.

Breakpoints – расширение возможностей использования точек останова при отладке программ. Активизация опции *Breakpoints* приводит к появлению на экране диалогового окна, содержащего информацию обо всех действующих в данный момент точках останова. О каждой из них хранятся следующие данные: номер строки, условие срабатывания и регулярность срабатывания. Последний элемент данных является числом, которое показывает, сколько раз надо пройти через точку останова перед тем, как она сработает.

Для управления диалоговыми окнами предназначены три кнопки: [*Edit*], [*Delete*] и [*View*]. Они позволяют производить редактирование данных, относящихся к выбранной точке останова; удаление и просмотр текста, связанного с выбранной точкой останова.

Активизация режима редактирования (кнопка [*Edit*]) приводит к появлению нового диалогового окна. Для работы в нем предназначены две кнопки: [*Modify*] и [*New*]. Первая позволяет менять условия уже существующей точки останова, а вторая – сгенерировать новую точку останова.

Отметим, что опция *Breakpoint* дает возможность работать сразу с несколькими исходными файлами.

Call Stack – используется только в том случае, когда выполнение программы временно приостановлено.

Выбор этой опции приводит к открытию диалогового окна, содержащего список имен активных в данный момент блоков. Список организуется в форме стека, на дне которого находится имя главной программы, а в вершине имя подпрограммы, которая последней получила управление.

С помощью клавиш-стрелок можно выделить подсветкой имя любой подпрограммы в списке. Если затем нажать клавишу *Enter*, то на экране появятся значения параметров этой подпрограммы. Очень удобна эта опция для отладки сложных программных комплексов.

Register – активизация окна *Register*, содержащего данные, характеризующие состояние регистров центрального процессора. Это окно целесообразно использовать тогда, когда в программе на Турбо-Паскале присутствуют операторы языка Ассемблер.

Output – активизация окна вывода. В окно помещается информация, во-первых, передаваемая программой на экран дисплея, и, во-вторых, запрашиваемая у пользователя.

User Screen – просмотр результатов работы программы не в отдельном окне, а на всем экране, т. е. можно сказать, что опция *User Screen* расширяет границы окна вывода до размеров экрана.

Команды опции Options.

Меню *Options* предназначено для управления режимами компиляции и компоновки программ, написанных на Турбо-Паскале, для определения параметров интегрированной среды и, в частности, встроенного отладчика. Специальные средства этого меню позволяют проводить настройку текстового редактора, управлять мышкой, выбирать необходимую цветовую гамму и т.п.

Compiler: выбор этой опции приводит к развертыванию на экране диалогового окна, в котором присутствуют кнопки-флаги, служащие для управления режимами компиляции, и поле ввода. Кнопки-флаги объединены во вспомогательные групповые окна (их также называют пультами)

1. С помощью окна *Code Generation* можно определять значения флагов, управляющих генерацией кода.

- Флаг *Force Far Calls* дает возможность установить для всех процедур и функций дальнюю или ближнюю модель вызова. Напомним, что то же самое можно сделать с помощью директивы компилятора {$F} или атрибутов *Near и Far*.
- Флаг *Overlays Allowed* управляет генерацией оверлейного кода модулей. Установка флага *Overlays Allowed* эквивалентна директиве компилятора {$0+}.
- Флаг *Word Align Data* определяет способ размещения переменных и констант в памяти. Если флаг установлен, то данные символьных типов выравниваются на начало слова. Это позволяет увеличить скорость выполнения программы, но ведет к нежелательному расходованию памяти. Аналогичная директива компилятора – {$A}.
- Флаг 286 *Instructions* позволяет генерировать код в расчете на процессор типа Intel 80286 или не делать этого. Того же эффекта можно добиться с помощью директивы компилятора {$G}.

2. Окно *Runtime errors* дает возможность установить флаги, обеспечивающие контроль ошибок, возникающих в процессе выполнения программы.

- Флаг *Range Checking* включает режим генерации дополнительного кода, служащего для проверки выхода значений переменных и индексов за пределы отведенных для них диапазонов. Аналогичная директива компилятора – {$R}.
- Флаг *Stack Checking* позволяет перед вызовом подпрограмм проверять, имеется ли достаточное для размещения локальных переменных количество свободного пространства в сегменте стека. Аналогичная директива компилятора – {$S}.
- Флаг *I/O Checking* используется для подключения и отключения средств, контролирующих правильность выполнения операций ввода-вывода. Если средства контроля ввода-вывода отключены (флаг *I/O Checking* сброшен), анализировать правильность выполнения операции ввода-

вывода можно с помощью системной функции *IOResult*. Аналогичная директива компилятора – {$I}.

- Флаг *Overflow checking* позволяет контролировать ситуации переполнения, возникающие при работе с данными. Если флаг установлен, то переполнение приводит к завершению выполнения программы и выдаче диагностического сообщения. Аналогичная директива компилятора – {$O}.

3. Окно *Debugging* содержит два флага – *Debug Information* и *Local Symbols*. Еще один дополнительный флаг *Information Symbol* добавляется при работе в защищенном режиме.

- Флаг *Debug Information* используется для включения и отключения режима генерации отладочной информации в процессе компиляции. Отладочная информация в основном сосредоточена в таблице, связывающей операторы исходного текста с фрагментами кода. Только в том случае, когда установлен флаг *Debug Information*, можно использовать возможности интегрированного отладчика, к числу которых относятся пошаговое выполнение программы, локализация ошибок, точки останова и т.п. Отладочная информация записывается в .EXE-файл программы или в .TPU-файл модуля. Аналогичная директива компилятора – {$D}.
- Флаг *Local Symbols* управляет доступом к локальным переменным и типизированным константам в режиме отладки. Следовательно, нельзя использовать опции *Watch* и *Call Stack*, если не установлен флаг *Local Symbols*. Устанавливать флаг *Local Symbols* можно только тогда, когда флаг *Debug Information* уже установлен. Аналогичная директива компилятора – {$L}.
- Флаг *Information Symbol* следует устанавливать для того, чтобы информация о глобальных символах, полученная в результате компиляции, использовалась отладчиком. Аналогичная директива компилятора – {$Y}.

4. Окно *Syntax Options* опции *Compile* позволяет установить ряд флагов, оказывающих влияние на генерируемый код.

- флаг *Strict Var-String* служит для проверки, соответствует ли длина строкового параметра с атрибутом *Var* длине аргумента. Аналогичная директива компилятора – {$V}.
- Если флаг *Complete Boolean Evaluation* установлен, то выполняются все операции, связанные с вычислением значения логического выражения даже тогда, когда результат уже известен. В противном случае вычисления прекращаются, когда становится ясен результат. Аналогичная директива компилятора – {$B}.
- флаг *Extended syntax* дает возможность вызывать функции как процедуры и использовать ASCII-строки. Аналогичная директива компилятора – {$X}.

- Установка флага *Typed @ operator* делает результат операции @ типизированным указателем. Тип результата определяется типом аргумента. Если же флаг сброшен, результат операции @ имеет тип *Pointer* вне зависимости от типа аргумента. Аналогичная директива компилятора – {$T}.
- Флаг *Open parameters* позволяет использовать в подпрограммах открытые параметры типа *String*. Аналогичная директива компилятора – {$P}.

5. Окно *Numeric Processing* позволяет установить флаги, определяющие, как будут обрабатываться числа с плавающей точкой, относящиеся к типам *SINGLE, DOUBLE, EXTENDED* и *COMP*.

- флаг *Numeric Processing* ориентирует компилятор на работу с числовым сопроцессором (или программами его эмуляции), что дает возможность использовать расширения действительного типа данных. Аналогичная директива компилятора – {$N}.
- флаг *Emulation* позволяет либо использовать все возможности сопроцессора (если он есть), либо эмулировать его работу программно. Значение флага *Emulation* анализируется только тогда, когда установлен флаг *Numeric Processing*. Аналогичная директива компилятора – {$E}.

6. Поле ввода *Conditional Defines* служит для определения ключевых слов, управляющих работой операторов условной компиляции.

Memory size – служит для определения размеров сегмента стека и минимального и максимального размеров доступной оперативной памяти. Аналогичная директива компилятора – {$M}.

Linker – с помощью этой опции может управлять режимом работы компоновщика. Выбор опции *Linker* приводит к развертыванию на экране диалогового окна, позволяющего с помощью кнопок-переключателей задать значения параметров *Map File* и *Link Buffer*.

- Параметр *Map File* служит для определения условий формирования *Map*-файла, в который помещается карта распределения памяти компонуемой программы. Этот параметр может принимать одно из четырех значений, указанных в окне. Выбор *Off* эквивалентен отказу от создания *Map*-файла, остальные значения позволяют определить, какую информацию следует помещать в *Map*-файл. *Map*-файлы используются для отладки программ с помощью внешних по отношению к системе программирования Турбо-Паскаль средств отладки, например Турбо-Debugger.
- Параметр *Link Buffer* позволяет указать, должен ли компоновщик размещать необходимые для его работы таблицы и саму программу в период компоновки в оперативной памяти или ему следует хранить их на диске. Если значением параметра является *MEMORY*, компоновщик будет работать быстро, но ему может не хватить оперативной памяти, если *DISK*, то работа компоновщика замедлится, но памяти потребуется меньше.

Debugger приводит к открытию диалогового окна, содержащего, в свою очередь, два групповых окна: *Debugging* и *Display Swapping*.

- окно *Debugging* содержит флаг *Integrated*, установка которого необходима для того, чтобы отладочная информация помещалась в .EXE-файл и можно было использовать средства встроенного отладчика. Второй флаг этого окна *Standalone* показывает, можно ли для отладки программы использовать внешний отладчик – Turbo-Debugger или нет. Отметим, что устанавливать флаги *Integrated* и *Standalone* имеет смысл тогда, когда программа или модуль компоновались с установленными флагами *Debug Information* и *Local Symbols*.
- окно *Display Swapping* служит для определения параметра *Display Swapping*, который может принимать одно из трех значений: *Smart*, *Always* и *None*. С помощью этих значений определяется, в каких случаях следует переключать экран с воспроизведения окна редактирования на окно вывода. В первом случае (значение *None*) переключения не будет происходить никогда, т.е. выводимые данные будут накладываться на текст программы; во втором случае (значение *Smart*) переключения будут связаны с обращениями к экрану для ввода и вывода, а также с вызовом подпрограмм; в третьем случае (значение *Always*) мгновенные переключения будут происходить перед выполнением каждого оператора программы. Естественно, что чаще остальных для *Display Swapping* выбирается значение *Smart*.

Directories: в этом режиме имеется возможность указать системе программирования Турбо-Паскаль, в каких директориях следует искать файлы и куда следует помещать файлы, необходимые для работы его программ. Выбор опции *Directories* приводит к открытию диалогового окна, содержащего четыре поля, предназначенных для ввода адресов директорий. Если необходимо ввести несколько адресов, то их следует разделять точкой с запятой.

- в поле *EXE&TPU Directory* помещается адрес директории, в которую будут записываться файлы с расширениями .EXE и .TPU;
- в поле *Include Directories* – адреса директорий, в которых будет осуществляться поиск файлов, включаемых в программу с помощью директивы компилятора {$I<имя файла >};
- в поле *Unit Directories* – адреса директорий, в которых расположены файлы системы программирования Турбо-Паскаль с расширением .TPU (например, GRAPH.TPU);
- в поле *Object Directories* – адреса директорий, содержащих файлы с расширением .OBJ (эти файлы чаще всего используются для хранения объектных кодов внешних подпрограмм, первоначально написанных на языке Ассемблер).

Environment – изменение некоторых параметров, связанных с работой интегрированной среды системы программирования Турбо-Паскаль 6.0. Выбор опции *Environment* приводит к появлению на экране дополнительного меню с позициями: *Preferences, Editor, Mouse, Startup, Colors, Save Options* и *Retrieve options*. Рассмотрим эти позиции.

1. Preferences: на экране появляется диалоговое окно, приглашающее определить значения нескольких параметров и установить флаги в групповых окнах:

- Параметр *Screen Size* определяет количество строк: либо 25, либо 43(50) строк для адаптеров EGA(VGA).
- Параметр *Source Tracking* дает возможность определить, следует ли открывать новое окно редактирования, если в процессе отладки участвует модуль, который еще не был загружен в одно из окон редактирования.
- Групповое окно *Auto Save* содержит значения трех флагов:
  - флаг *Editor Files* – автоматическое сохранение на диске копии файла из окна редактирования перед выходом из среды Турбо-Паскаля или перед прогоном программы.
  - флаг *Environment* – задание режима автоматического сохранения текущего состояния среды системы программирования Турбо-Паскаль в файле TURBO.TP в случае выхода из системы. Это очень полезно, поскольку новый сеанс работы можно начать в той среде, которая сформирована ранее
  - флаг *Desktop* устанавливается для сохранения информации о текущей конфигурации окон редактирования. Отметим, что флаг *Desktop* следует устанавливать только в том случае, когда установлен флаг *Environment*.
- В групповом окне *Options* размещаются три флага: *Auto track source, Close on go to source* и *Change dir on open*.
  - флаг *Auto track source* определяет, выделяется подсветкой или нет в окне редактирования строка, соответствующая текущему сообщению в окне *Messages*.
  - установка флага *Close on go to source* приводит к автоматическому закрытию окна *Messages* при переходе в окно редактирования.
  - флаг *Change dir on open* меняет текущую директорию на ту, в которой находится редактируемый файл.
- В групповом окне параметра *Desktop file* указывается, в какой директории находится файл *Turbo.dsk* (или *Tpx.dsk*). С помощью параметра *Desktop file options*, который появляется в меню только при работе в защищенном режиме, определяется, необходимо или нет сохранять после выхода из системы информацию о глобальных символах программы. Эта информация записывается в файл *Tpx.psm* и может быть использована в следующих сеансах работы с системой программирования. Напомним, что информация о глобальных символах нужна для работы браузера.

2. Editor приводит к появлению на экране диалогового окна с набором флагов и полей ввода, позволяющих управлять режимом работы текстового редактора системы программирования Турбо-Паскаль.

- флаг *Create backup files* – автоматическое переименование перед выполнением команды *Save* (см. опцию *File*) текстового файла, связанного с активным окном редактирования, в файл с расширением .BAK. Таким образом, на диске всегда будет сохраняться предыдущая версия программы или модуля.
- флаг *Syntax highlight* определяет, выделяются или нет в исходном тексте различные синтаксические элементы разными цветами. Предусмотрено выделение следующих синтаксических элементов: ключевых слов, идентификаторов, комментариев, разделителей, строк символов, чисел и фрагментов текста на Ассемблере. Кроме того, имеется возможность задавать групповые имена файлов, на которые распространяется выделение синтаксиса цветом.

Остальные флаги и поля ввода позволяют управлять режимами вставки/замены текста, положением курсора, работой с блоками и средствами табуляции.

3. Позиция ***Mouse*** опции *Environment* используется для управления режимами работы мыши. Активной клавишей мыши как известно является та, что расположена слева (не важно, имеет мышка две или три клавиши). Основные действия выполняются именно с помощью этой клавиши. Флаг *Revers House Buttons* позволяет сделать активной клавишей мыши правую. Параметр *Right Mouse Button* может принимать окно из шести приводимых в диалоговом окне опции *Mouse* значений.

Каждое значение (кроме *Nothing*) определяет ту функцию, которая будет выполняться после нажатия правой клавиши (или левой, если установлен флаг *Reverse Mouse Buttons*) мыши. Функции в основном дублируют некоторые режимы работы системы программирования Турбо-Паскаль. Параметр *Mouse Double Click* позволяет регулировать максимально допустимый интервал времени между двумя последовательными нажатиями клавиши мыши.

4. Позиция Sturtup позволяет выбрать глобальные характеристики режима работы интегрированной среды с помощью установки флагов и задания в специально выделенных полях ввода значений ряда параметров.

- Флаг *Dual Monitor Support* управляет переходом в режим двойного монитора. Этот режим возможен только в том случае, если параметр снабжен двумя видеоадаптерами. В режиме двойного монитора второй монитор обычно связан с окном вывода.
- Флаг *Graphics Screen Save* позволяет в процессе отладки сохранять образ графического экрана в памяти.
- Флаг *EGA/VGA palette save* дает возможность в случае необходимости восстановить стандартную 16-цветную EGA-палитру.
- Флаг *CGA Snow Checking* используется только для адаптеров CGA и связан со способом обновления экрана.
- Флаг *LCD Colour Set* нужно устанавливать только в том случае, если используется монитор с жидкокристаллическим экраном.

– Установка флага *Use expanded* memory разрешает интегрированной среде использовать для работы дополнительную память.
– Флаг *Load Turbo.tpl* устанавливается в том случае, когда желательно, чтобы файл *Turbo.tpl*, представляющий собой библиотеку системных модулей, загружался средой в оперативную память при запуске.

В диалоговом окне присутствуют еще четыре поля, содержащие: размеры областей динамической памяти, отводимые под различные элементы среды, и адрес директории (чаще всего на виртуальном диске), которая может использоваться для ускорения работы интегрированной среды в случае отсутствия дополнительной памяти.

Без особой необходимости заданную по умолчанию установку флагов менять не рекомендуется. Это, в первую очередь, связано с флагом *Load Turbo.tpl*.

5. Colors изменение основного и фонового цвета всех основных и вспомогательных окон интегрированной среды.

Save. – сохранение параметров сформированной той новой среды. Параметры компилятора, компоновщика и непосредственно самой среды обычно помещаются в файл *Turbo.tp*, данные о конфигурации окон редактирования и привязка их к файлам – в файл *Turbo.dsk* (или *Tpx.dsk*). Как уже было отмечено, сохранить параметры среды можно с помощью диалогового окна *Auto save* опции *Environment*.

Команды опции Window.

Это команды управления окнами. В системе программирования Турбо-Паскаль 6.0 могут быть открыты окна *редактирования, наблюдений, вывода, помощи и браузера*. Каждое из них может быть развернуто на весь экран или несколько окон могут одновременно присутствовать на экране. Есть средства, позволяющие осуществлять изменение размеров окна и прокрутку содержимого окна.

Size/Move – изменение размеров активного окна и места его расположения на экране. Операции можно осуществлять либо с помощью клавиш со стрелками, либо с помощью мыши. Если окно имеет специально предназначенный для изменения размеров уголок, удобнее пользоваться мышью.

Zoom – расширение окна до его максимальных размеров

Tile – размещение на экране дисплея всех открытых в данный момент окнон. Размеры всех окон одинаковы, друг друга они не перекрывают.

Cascade – расположение на экране всех открытых окон, но друг за другом; не перекрытым останется только одно активное окно. У всех остальных окон видны только их заголовки.

Close all – закрытие всех открытых окон.

Refresh Display – восстановление содержимого экрана для просмотра результатов работы программы.

Next – активизация следующего по порядку за активным в данный момент окно. Если окна размещены в соответствии с требованиями опции *Cascade*, то активизируемое окно становится, естественно, и самым верхним.

Previous – активизация окна, которое было открыто непосредственно перед текущим активным окном.

Close – закрытие активного окна. Для того чтобы закрыть окно, можно также подвести мышь к левому верхнему углу окна и два раза нажать левую клавишу мыши.

List – вывод на экран выводится список всех открытых окон. Можно вызвать из списка любое окно и активизировать его нажатием клавиши *Enter*.

Команды опции Help.

Служат для получения справочной информации по любым аспектам языка Турбо-Паскаль и его интегрированной среды. Эта информация, хранящаяся в файле TURBO.HLP, открывается в специальном окне, которое называется окном помощи или окном Help. Открыть окно помощи можно не только с помощью средств главного меню, но и другими способами:

– во-первых, можно получить конкретную справку о выбранной опции меню или об активном диалоговом окне, просто нажав клавишу F1 или кнопку [Help], если она присутствует на экране;

– во-вторых, справку о языке можно получить, поместив курсор в окне редактирования под интересующим Вас словом и нажав клавиши *Ctrl-Fl*.

Если окно *Help* активно, то присутствующие в нем подсвеченные ключевые слова или предложения могут быть выбраны для получения справочной информации. Удобно в этом случае пользоваться мышью.

Кроме того, тексты из окна *Help* могут копироваться в карман.

Contents – оглавление системы справочной информации.

Index – получение списка ключевых слов, с которыми связана имеющаяся в системе справочная информация. Для того чтобы подвести курсор к нужному ключевому слову, не обязательно пробегать весь список, достаточно набрать на клавиатуре начальные буквы искомого слова.

Topic Search – получение справки о языковой конструкции. Для этого необходимо, находясь в активном окне редактирования, подвести курсор к интересующему слову, а затем войти в меню и выбрать данную опцию (но проще использовать *Ctrl-F1*).

Previous Topic – восстановление содержимого предыдущего окна помощи. Всего в системе сохраняется двадцать запрошенных последними экранов со справочной информацией.

Using Help – пояснение, как пользоваться справочной информацией. Из окна помощи перейти в этот режим можно, просто нажав *F1*.

Локальные меню

Локальных меню всего пять (здесь учитывается и меню *Browse*, доступное только в защищенном режиме работы процессора) и каждое из них привязано к соответствующему окну: редактирования, помощи, наблюдений, сообщений и браузера. Активизация локального меню производится нажатием клавиш Alt-F10 или правой клавиши мыши.

Меню окна редактирования

В состав этого меню входят четыре опции меню ***Edit*** *– Cut, Copy, Paste и Clear;* опция меню ***Help*** *–Topic search;* опция меню ***Run*** *– Go to cursor;* две опции меню ***Debug*** *– Evaluate/modify и Add watch;* опция *Options*, которая дублирует опцию *Environment/Editor* меню ***Option***.

Меню окна помощи.

Меню включает четыре опции меню *Help – Contents, Index, Topic search и Previous topic*; а также опцию меню *Edit – Copy*.

Меню окна наблюдений

В состав этого меню входят шесть опций: *Add, Modify, Remove, Clear all, Enable и Disable.*

Add служит для добавления выражения в окно наблюдений. Любое выражение, присутствующее в окне наблюдений, можно сделать текущим. Для этого необходимо активизировать окно наблюдений и, с помощью клавиатуры, выделить подсветкой строку с искомым выражением.

Modify дает возможность редактировать текущее выражение, находящееся в окне наблюдений.

Remove позволяет удалить текущее выражение из окна наблюдений. Но в этом случае для удаления проще воспользоваться клавишей *Del.*

Clear all очищает окно наблюдений, т. е. удаляет из него все выражения.

Disable позволяет скрыть значение текущего выражения окна наблюдений: вместо значения будет появляться слово < disable >.

Enable отменяет действие опции *Disable.*

Меню окна сообщений.

В этом меню три опции: *Clear*, *Goto source* и *Track source.* Опция *Clear* позволяет очистить окно сообщений, опция *Goto source* – перейти в окно редактирования, в котором находится анализируемый файл и, наконец, опция *Track source* – перейти в окно редактирования и выделить подсветкой строку, соответствующую текущему сообщению.

Основные команды встроенного редактора текста

Таблица 35. Список горячих клавиш

Горячая клавиша	Функция	Опция меню
F1	Открытие окна с подсказками	HELP/TOPIC
F2	Сохранение файла, находящегося в активном окне редактирования	FILE/SAVE
F3	Загрузка файла в активное окно редактирования	FILE/OPEN
F4	Выполнение программы до строки, помеченной курсором	RUN/ GOTO CURSOR
F5	Увеличение/уменьшение размеров активного окна	WINDOW/ZOOM
F6	Переход в следующее открытое окно	WINDOW/NEXT
F7	Выполнение очередного оператора программы или подпрограммы	RUN/ TRACE INTO
F8	Выполнение очередного оператора программы	RUN/ TRACE OVER
F9	Компиляции программы/модуля и возможно связанных с ними модулей	COMPILE/ MAKE
F10	Возврат в главное меню	
CTRL+F1	Выдача справки о языковой конструкции	HELP/ TOPIC SEARCH
CTRL+F2	Прекращение отладки программы	RUN/PROGRAM RESET
CTRL+F3	Вывод на экран списка имен активных блоков	WINDOW/ CALL STACK
CTRL+F4	Просмотр значения выражения, изменение значения переменной	DEBUG/EVALUATE
CTRL+F5	Изменение размера и положения активного окна	WINDOW SIZE/MOVE
CTRL+F6	Добавление выражения в окно наблюдений	DEBUG/ ADD WATCH
CTRL+F8	Установка или отмена точки останова	DEBUG/TOGGLE BREAKPOINT
CTRL+F9	Запуск программы	RUN/RUN
CTrl+K, R	Вставить в текущей позиции курсора файл (с диска)	
Ctrl+K, W	Записать выделенный блок в отдельный файл	
Ctrl+K, T	Выделить слово	
Ctrl+K, B	Перейти в начало выделенного блока	

Горячая клавиша	Функция	Опция меню
Ctrl+Q, K	Перейти в конец выделенного блока	
Ctrl+Q, A	Найти и заменить	
Ctrl+Q, L	Восстановить строку	
Ctrl+Q, Y	Удалить текст до конца строки	
Ctrl+Y	Удалить строку	
Ctrl+Home	Перейти в начало экрана (окна)	
Ctrl+End	Перейти в конец экрана (окна)	
Ctrl+PgUp	Перейти к началу файла	
Ctrl+PgDn	Перейти к концу файла	
Shift+клавиша управления курсором	Выделение блока	
CTRL+Del	Удаление выделенного текста	EDIT/CLEAR
CTRL+Ins	Копирование выделенного текста в карман	EDIT/COPY
CTRL+L	Повторение последней операции Find (поиска) или Rèplace (замены)	
CTRL+P	Открытие последнего закрытого окна	SEARCH/PREVIOUS
SHIFT+DEL	Удаление выделенного текста из файла и помещение его в карман.	EDIT/CUT
SHIFT+INS	Помещение выделенного текста из кармана в активное окно, в место, определяемое курсором	EDIT/PASTE
SHIFT+F6	Переход в предыдущее открытое окно	WINDOW/ PREVIOUS
ALT+F1	Восстановление содержимого предыдущего окна подсказки	HELP/PREVIOUS TOPIC
ALT+F3	Закрытие активного окна	WINDOW/CLOSE
ALT+F5	Показ окна вывода (экрана операционной системы)	WINDOW/USER SCREEN
ALT+F6	Переход в окно, которое было открыто непосредственно перед текущим активным окном	WINDOW/PREVIOUS
ALT+F8	Определение места нахождения ошибки в исходном файле	SEARH/FIND ERROR
ALT+F9	Компиляция файла, находящегося в активном окне редактирования	COMPILE/ COMPILE
ALT+X	Выход из среды системы	FILE/EXIT
ALT+0	Вывод на экран списка всех открытых окон	WINDOW/LIST
ALT+цифра	Переход в открытое окно с номером, заданным цифрой 1..9	

Горячая клавиша	Функция	Опция меню
ALT+S, F	Поиск заданного текста в файле активного окна	SEARCH/FIND
ALT+S, R	Поиск заданного текста в файле активного окна и его замена	SEARCH/REPLACE
ALT+C	Активизация меню COMPILE	COMPILE
ALT+D	Активизация меню DEBUG	DEBUG
ALT+E	Активизация меню EDIT	EDIT
ALT+F	Активизация меню FIND	FIND
ALT+H	Активизация меню HELP	HELP
ALT+O	Активизация меню OPTIONS	OPTIONS
ALT+R	Активизация меню RUN	RUN
ALT+S	Активизация меню SEARCH	SEARCH
ALT+W	Активизация меню WINDOW	WINDOW

Приложение Б. Сообщения об ошибках

Таблица 36. Сообщения об ошибках на шаге компиляции

Ниже приводятся коды ошибок и сообщения об ошибках, генерируемые компилятором языка Турбо-Паскаль. Кроме перевода сообщений в некоторых случаях даются необходимые пояснения, а также рекомендации по устранению ошибок.

Код ошибки	Сообщение об ошибке	Перевод сообщения, возможная причина ошибки и рекомендации для ее устранения
1	Out of memory	«не хватает памяти» — Компилятор извещает, что доступной памяти недостаточно для размещения программы. Чтобы устранить ошибку, рекомендуется удалить из памяти ранее загруженные, но не используемые в данный момент программы, или указать, что объектный код должен выводиться на диск. Если это не дает результата, следует разделить программу или модуль на большее число модулей
2	Identifier expected	«ожидается идентификатор» — Возможно, в качестве идентификатора использовано зарезервированное слово
3	Unknown identifier	«неизвестный идентификатор» — Идентификатор не объявлен
4	Duplicate identifier	«дублируемый идентификатор» — Идентификатор объявлен дважды
5	Syntax error	«синтаксическая ошибка» — Обнаружен символ, отсутствующий в алфавите языка
6	Error in real constant	ошибка в записи константы вещественного типа
7	Error in integer constant	ошибка в записи константы целого типа
8	String constant exceeds line	«длина строковой константы превышает максимально допустимую длину строки» — Возможно, отсутствует закрывающий апостроф
9	Too many nested files	«слишком много вложенных файлов» — При включении исходных файлов (с помощью директив {$1 <имя файла >}) компилятор допускает не более 15 уровней вложенности

Код ошибки	Сообщение об ошибке	Перевод сообщения, возможная причина ошибки и рекомендации для ее устранения
10	Unexpected end of file	«несвоевременное появление признака конца файла» Возможно, не совпадает количество операторов begin и end или не закрыт комментарий
11	Line too long	«слишком длинная строка» — Длина строки превысила 126 символов
12	Type identifier expected	«ожидается идентификатор типа» — В объявлении отсутствует идентификатор типа
13	Too many open files	«слишком много открытых файлов» — Следует задать большее число файлов в CONFIG.SYS (в записи FILES = < число)
14	Invalid file name	неправильно задано имя файла
15	File not found	файл не найден
16	Disk full	на диске нет свободного места
17	Invalid compiler directive	неправильно записана директива компилятора
18	Too many files	«слишком много файлов» — С помощью директив {$1 <имя файла>} включается чрезмерное количество исходных файлов
19	Undefined type in pointer definition	«необъявленный тип в объявлении указателя» — Не объявлен тип данных, указанный в объявлении ссылочного типа
20	Variable identifier expected	ожидается идентификатор переменной
21	Error in type	ошибка в объявлении типа
22	Structure too large	«структура слишком большая» — Превышен допустимый размер (65520 байт) области памяти для данных структурированного типа
23	Set base type out of range	«число значений базового типа для множества превышает допустимое» — Базовый тип не должен содержать более 256 значений
24	File components may not be files or objects	компонентами файла не могут быть файлы или объекты
25	Invalid string length	«недопустимая длина строки» — Превышена максимально допустимая длина строки (255)
26	Type mismatch	«несоответствие типов» — Не соответствуют друг другу типы данных в выражении
27	Invalid subrange base type	недопустимый базовый тип для интервального типа
28	Lower bound greater than upper bound	нижняя граница больше верхней границы

Код ошибки	Сообщение об ошибке	Перевод сообщения, возможная причина ошибки и рекомендации для ее устранения
29	Ordinal type expected	ожидается ссылка на порядковый тип
30	Integer constant expected	ожидается целая константа
31	Constant expected	ожидается константа
32	Integer or real constant expected	ожидается целая или вещественная константа
33	Type identifier expected	ожидается идентификатор типа
34	Invalid function result type	недопустимый тип результата функции
35	Label identifier expected	ожидается идентификатор метки
36	Begin expected	ожидается зарезервированное слово begin
37	End expected	ожидается зарезервированное слово end
38	Integer expression expected	ожидается выражение целого типа)
39	Ordinal expression expected	ожидается выражение порядкового типа
40	Boolean expression expected	ожидается выражение булевого типа
41	Operand types do not match operator	типы операндов не соответствуют оператору
42	Error in expression	ошибка в выражении
43	Illegal assignment	неправильное присваивание
44	Field identifier expected	ожидается идентификатор поля записи
45	Object file too large	«объектный файл слишком большой» — OBJ-файл превысил максимально допустимый размер (65520 байт)
46	Undefined external	«не определена внешняя подпрограмма» — Вероятно, во внешней подпрограмме отсутствует соответствующее определение PUBLIC
47	Invalid object file record	«нераспознаваемая запись объектного файла» — Вероятно, объектный файл искажен
48	Code Segment too large	«кодовый сегмент слишком велик» — Превышен максимально допустимый размер кода программы или модуля (65520 байтов)
49	Data segment too large	«сегмент данных слишком велик» — Превышен максимально допустимый размер сегмента данных (65520 байтов)
50	Do expected	ожидается ключевое слово DO
51	Invalid PUBLIC definition	«неправильное определение public» — Несоответствие определения PUBLIC в программе на Ассемблере и директивы external в программе или модуле на Паскале

Код ошибки	Сообщение об ошибке	Перевод сообщения, возможная причина ошибки и рекомендации для ее устранения
52	Invalid EXTRN definition	«неправильное определение extrn» — Вероятно, фрагмент программы на языке Ассемблера не объявлен в программе или модуле
53	Too many EXTRN ltfinition	«слишком много определений extrn» — Максимально допустимое число определений EXTRN в Obj-файле равно 256
54	OF expected	ожидается зарезервированное слово OF
55	INTERFACE expected	ожидается зарезервированное слово INTERFACE
56	Invalid relocatable	«неправильно определена смещенная ссылка» — Вероятно, во фрагменте программы на языке Ассемблера неправильно задано смещение адреса
57	THEN expected	ожидается зарезервированное слово Then
58	TO or DOWNTO expected	ожидается зарезервированное слово To или downto
59	Undefined forward	не завершено опережающее объявление
60	Too many procedures	«слишком много процедур» — Превышено максимально допустимое (512) количество подпрограмм, объявленных в программе или модуле
61	Invalid Typecast	неверно описанное преобразование типов
62	Division by zero	деление на нуль
63	Invalid file type	неправильно задан файловый тип
64	Cannot Read or Write variables of type	«нельзя читать или писать переменные этого типа» — Предпринята попытка чтения/записи данных, не обрабатываемых процедурами Read/Readln и Write/Writeln.
65	Pointer variable expected	ожидается переменная ссылочного типа
66	String variable expected	ожидается строковая переменная
67	String expession expected	ожидается выражение строкового типа
68	Circular unit reference	«циклические ссылки модулей» — Не допускается, чтобы два модуля ссылались друг на друга
69	Unit name mismatch	«неправильное имя модуля» — Модуль, имя которого задано в директиве Uses, не найден
70	Unit version mismatch	«неверная версия модуля» — Модуль, подключаемый к программе, был изменен после компиляции
71	Duplicate unit name	имя модуля дублируется в директиве uses
72	Unit file format error	ошибка в спецификации файла модуля
73	Implementation expected	ожидается зарезервированное слово implementation

Код ошибки	Сообщение об ошибке	Перевод сообщения, возможная причина ошибки и рекомендации для ее устранения
74	Constant and case types do not match	типы констант и селектора в операторе case не соответствуют друг другу
75	Record variable expected	ожидается переменная комбинированного типа
76	Constant out of range	константа не укладывается в допустимый диапазон
77	File variable expected	ожидается переменная файлового типа
78	Pointer expression expected	ожидается выражение ссылочного типа
79	Integer or real expression expected	ожидается выражение типа integer или real
80	Label not within current block	метка находится вне текущего блока
81	Label already defined	метка ранее уже объявлена
82	Undefined label in preceding statement part	необъявленная метка в предыдущей части раздела операторов
83	Invalid @ argument	неправильный аргумент оператора @
84	UNIT expected	ожидается зарезервированное слово unit
85	";" expected	ожидается точка с запятой
86	":" expected	ожидается двоеточие
87	"," expected	ожидается запятая
88	"(" expected	ожидается открывающая круглая скобка
89	")" expected	ожидается закрывающая круглая скобка
90	"=" expected	ожидается знак равенства
91	":=" expected	ожидается знак присваивания
92	"[" or "(" expected	ожидаются знаки "[" или "("
93	"]" or ")" expected	ожидаются знаки ']' или ')'
94	"." expected	ожидается точка
95	".." expected	ожидается горизонтальное двоеточие
96	Too many variables	«слишком много переменных» — Максимально допустимый размер памяти для размещения переменных равен 64К: – глобальных, объявленных в программе или модуле; – локальных, объявленных в подпрограмме
97	Invalid FOR control variable	«неправильная переменная цикла в операторе for» — Переменная цикла в операторе for должна принадлежать одному из простых типов
98	Integer variable expected	ожидается переменная целого типа
99	File and procedure types are not allowed here	в данном контексте файловый и процедурный типы недопустимы

Код ошибки	Сообщение об ошибке	Перевод сообщения, возможная причина ошибки и рекомендации для ее устранения
100	String length mismatch	«неправильная длина строки» — Длина строковой константы не соответствует количеству элементов символьного массива.
101	Invalid ordering of fields	неправильный порядок следования полей
102	String constant expected	ожидается константа строкового типа
103	Integer or real variable expected	ожидается переменная типа integer или real
104	Ordinal variable expected	ожидается переменная порядкового типа
105	INLINE error	ошибка в директиве inline
106	Character expession expected	ожидается выражение символьного типа
107	Too many relocation items	«слишком много перемещаемых элементов» — Размер таблицы перемещаемых символов в объектном файле превышает 64К байта. Это означает, что программа слишком велика, чтобы ее смог обработать компоновщик Турбо-Паскаля.
108	Overflow in arithmetic operation.	переполнение при выполнении арифметической операции
109	No enclosing For, While or Repeat statement.	использование операторов Break и Continue вне пределов цикла
112	Case constant out of range	«в операторе case константа не вписывается в допустимый диапазон» — Целочисленные константы, используемые в операторе case, должны находиться в пределах от -32768 до 32767.
113	Error in statement	ошибка в операторе
114	Cannot call an interrupt procedure	не вызывается процедура прерывания
116	Must be in 8087 mode to compile this	«для компиляции данной конструкции должен быть установлен режим сопроцессора» — Данная конструкция может быть откомпилирована только в режиме {$ N+}.
117	Target address not found	«заданный адрес отсутствует» — Не обнаружен оператор, расположенный по заданному адресу.
118	Included files are not allowed here	«в заданном месте не допускается включение файла» — Исходные файлы нельзя включать внутри раздела операторов.
119	No inherited methods are accesible here.	неправильное использование ключевого слова Inherited

Код ошибки	Сообщение об ошибке	Перевод сообщения, возможная причина ошибки и рекомендации для ее устранения
121	Invalid Qualifier	«неправильно указан квалификатор» — Возможны следующие причины появления сообщения: задан индекс переменной, не объявленной как массив; задано поле переменной, не объявленной как запись; в качестве указателя используется переменная, не объявленная как ссылочная
122	Invalid variable reference	неправильная ссылка на переменную
123	Too many symbols	«слишком много символов» — Длина текста программы или модуля превышает 64К байта
124	Statement part too large	«слишком велик раздел операторов» — Превышен максимально допустимый размер раздела операторов программы (около 24К байт)
126	Files must be var parameters	«файлы должны быть параметрами-переменными» — Параметры файлового типа в объявлении подпрограмм должны описываться как переменные.
127	Too many conditional symbols	«слишком много условных символов» — Недостаточно памяти для размещения имен, указанных в директивах условной компиляции Рекомендуется укоротить одно или несколько символических имен
128	Misplaced conditional directive	несоответствие директив условной компиляции
129	ENDIF directive missing	отсутствует директива {$ ENDIF}
130	Error in initial conditional defines	«ошибка в установке условных определений» — Условия компиляции, заданные в Options/Compiler/ Conditional defines, должны разделяться пробелами, запятыми или точками с запятой
131	Header does not match previous definition	«заголовок не соответствует предыдущему определению» — Заголовок подпрограммы, заданный в секции связи модуля или в объявлении, использующем forward, не соответствует данному заголовку
132	Critical disk error	серьезная ошибка дискового накопителя
133	Cannot evaluate this expression	данное выражение невозможно вычислить
134	Expession incorrectly terminated	«неправильно завершено выражение» — Вероятно, отсутствует точка с запятой
135	Invalid format specifier	неправильная спецификация формата

Код ошибки	Сообщение об ошибке	Перевод сообщения, возможная причина ошибки и рекомендации для ее устранения
136	Invalid indirect reference	«неправильная косвенная ссылка» — Возможно, используется переменная типа absolute, базовая переменная которой не объявлена в данном модуле
137	Structured variables are not allowed here	в данном контексте структурные переменные недопустимы
138	Cannot evaluate without System unit	нельзя вычислить без модуля System
139	Cannot access this symbol	«невозможен доступ к данному символу» — Доступ к некоторым идентификаторам, например, переменным, возможен только после начала фактического выполнения программы
140	Invalid floating-point operation	«неправильная операция с плавающей точкой» » — Операция над двумя значениями вещественного типа привела к переполнению или делению на нуль
141	Cannot compile overlay to memory	нельзя компилировать код оверлейной программы в память
142	Procedure or function variable expected	ожидается переменная типа procedure или function
143	Invalid procedure or function reference	неправильная ссылка на процедуру или функцию
144	Cannot overlay this unit	«данный модуль нельзя сделать оверлейным» — Предпринята попытка объявить оверлейным модуль, который был откомпилирован без директивы {$0+ }.
145	Too many nested scopes	использование недопустимо большого числа вложенных элементов языка
146	File access denied	попытка использовать файл, предназначенный только для считывания, как выходной
147	Object type expected	ожидается объектный тип
148	Local object types are not allowed	«локальный объектный тип недопустим» — Объектный тип не должен объявляться в подпрограмме
149	VIRTUAL expected	ожидается зарезервированное слово virtual
150	Method identifier expected	ожидается идентификатор метода
151	Virtual constructors are not allowed	«виртуальные конструкторы недопустимы» — Метод-конструктор может быть только статическим
152	Constructor identifier expected	ожидается идентификатор конструктора
153	Destructor identifier expected	ожидается идентификатор деструктора

Код ошибки	Сообщение об ошибке	Перевод сообщения, возможная причина ошибки и рекомендации для ее устранения
154	Fail only allowed withing constructors	«процедуру Fail можно использовать только внутри конструкторов»
155	Invalid combination of opcode and operands	«неправильное сочетание кода операции и операндов» — Код операции Ассемблера не допускает данного сочетания операндов
156	Memory reference expected	«ожидается ссылка на элемент памяти» — Вероятно, регистровый операнд не заключен в квадратные скобки
157	Cannot add or subtract relocatable symbols	сложение или вычитание переместимых символов недопустимо
158	Invalid register combination	указана неправильная комбинация регистров
159	286/287 Instructions are not enabled	«не разрешены инструкции процессора 286/287» — Отсутствует директива компилятора {$G+}, разрешающая использовать операции 286/287
160	Invalid symbol reference	«неправильная ссылка на символ» — Данный символ нельзя использовать в ассемблерном операнде
161	Code generation error	ошибка при генерации кода
162	ASM expected	ожидается зарезервированное слово ASM

Сообщения об ошибках на шаге выполнения

Сообщения об ошибках на шаге выполнения имеют следующий формат:

Run-time error < номер > at < сегмент >:< смещение >,

где < номер > – номер ошибки,

< сегмент >:< смещение > – адрес, по которому произошла ошибка.

Ошибки шага выполнения делятся на четыре группы:

ошибки операционной системы MS DOS: 1-99,

ошибки ввода-вывода: 100–149,

серьезные ошибки: 150–199,

грубые ошибки: 200–255.

Таблица 37. Ошибки операционной системы DOS

Код ошибки	Сообщение	Причина появления ошибки
1	Invalid function number	«неверный номер функции» — Предпринята попытка вызова несуществующей функции DOS.
2	File not found	файл не найден
3	Path not fond	путь не найден
4	Too many open files	«слишком много открытых файлов» — Операционная система MS DOS позволяет открывать не более 15 файлов
5	File access denied	«запрещен доступ к файлу» — Вероятно, предпринята попытка записи в файл, предназначенный только для считывания.
6	Invalid file handle	«неправильный обработчик файла» — Данное сообщение выдастся в том случае, если при вызове MS DOS передается неправильный спецификатор файла. Вероятно, искажена файловая переменная
12	Invalid file access code	«неправильный код доступа к файлу» — Задано недопустимое значение переменной *FileMode*
15	Invalid drive number	неправильный номер диска
16	Cannot remove current directory	нельзя удалять текущую директорию
17	Cannot rename across drives	при переименовании файла нельзя указывать другое устройство

Таблица 38. Ошибки ввода-вывода

Ошибки, относящиеся к данной группе, вызывают завершение программы, если она откомпилирована в режиме {$I+}. В режиме {$I-} программа продолжает выполняться, а код ошибки возвращается функцией *IOResult*.

Код ошибки	Сообщение	Причина появления ошибки
100	Disk read error	«ошибка чтения с диска» — Предпринята попытка чтения после конца файла
101	Disk write error	«ошибка записи на диск» — Диск целиком заполнен
102	File not assigned	«файл не назначен» — Файловой переменной не назначен файл с помощью процедуры *Assign*
103	File not open	файл не открыт
104	File not open for input	файл не открыт для ввода
105	File not open for output	файл не открыт для вывода
106	Invalid numeric format	«неправильный числовой формат» — Числовое значение, считанное из текстового файла, имеет неправильный формат

Таблица 39. Серьезные ошибки

Код ошибки	Сообщение	Причина появления ошибки
150	Disk is write-protected	диск защищен от записи
151	Unknown unit	неизвестное устройство
152	Drive not ready	дисковод не готов к работе
153	Unknown command	неизвестная команда
154	CRC error in data	ошибка в данных на диске
156	Disk seek error	ошибка поиска на диске
157	Unknown media type	неизвестный тип носителя
158	Sector not found	сектор не найден
159	Printer out of paper	в принтере нет бумаги
160	Device write fault	ошибка при записи на устройство
161	Device read fault	ошибка устройства при чтении
162	Hardware failure	отказ аппаратных средств

Таблица 40. Грубые ошибки

Возникновение этих ошибок всегда приводит к немедленной остановке программы.

Код ошибки	Сообщение	Причина появления ошибки
200	Division by zero	деление на нуль
201	Range check error	«выход за допустимые границы» — Вычисленное или присвоенное значение не укладывается в допустимый диапазон
202	Stack overflow error	переполнение стека
203	Heap overflow error	переполнение области динамической памяти
204	Invalid pointer operation	«неправильная операция с указателем» — Используемая ссылочная переменная содержит *nil* или адрес за пределами динамической памяти.
205	Floating point overflow	переполнение в операции с плавающей точкой
206	Floating point underflow	потеря порядка в операции с плавающей точкой
207	Invalid floating point operation	неправильная операция с плавающей точкой
208	Overlay manager not installed	«монитор оверлеев не инициирован» — Вероятно, не была вызвана процедура *OvrInit* или при ее вызове произошла ошибка
209	Overlay file read error	ошибка при чтении оверлейного файла
210	Object not initialized	объект не инициализирован
211	Call to abstract method	«обращение к абстрактному методу» — Предпринята попытка создания экземпляра объекта абстрактного типа, содержащего один или несколько абстрактных методов. Однако абстрактные типы существуют только для того, чтобы можно было наследовать от них и перекрывать абстрактные методы
212–214		Эти сообщения связаны с ошибками, возникающими при работе с пакетом Turbo Vision.

Приложение В.
Описание числовых файлов для лабораторных работ

Текстовые файлы DATF.TXT и DATI.TXT содержат соответственно вещественные и целые числа, набитые по 10 чисел в строке, DATF.BIN и DATI.BIN содержат вещественные и целые числа во внутримашинной кодировке. Файлы SIMBOL.TXT, RUS.TXT и LAT.TXT содержат символьную информацию. Имена, строение и метод доступа к данным файлов приведены ниже.

1. Файл **DATI.TXT** – последовательный символьный (текстовый) набор целочисленных данных, содержит 1000 чисел в виде 50 строк (записей) по 20 чисел в каждой строке. Значения чисел не превосходят 999 по абсолютной величине.

Пример открытия файла и чтения из него чисел с 310 по 315:

```
VAR
mas : array[1..5] of integer;
fin : text;
. . .
BEGIN
. . .
Assign(fin,'D:\LAB1\DATI.TXT');
Reset(fin);
 пропуск 300 чисел в виде 15 строк по 20 чисел:
for i:=1 to 15 readln(fin);
 пропуск первых 9 чисел (301-309)
и чтение 6 чисел (310-315):
for i:=1 to 9 read(fin);
for i:=1 to 6 read(fin,mas[i]);
close(fin);  закрытие файла
. . .
```

Примечание: Если рассматривать этот набор данных, как содержащий последовательность символов – цифр, пробелов и других служебных символов, то при чтении его в символьный массив рекомендуется пропускать концы строк. Например, при чтении из этого набора данных в одномерный символьный массив **Txt** следует использовать функцию **Eoln**:

```
for i:= ... do
if not Eoln(fin) then Read(fin, Txt[i])
else Readln(fin);
```

Если желательно дополнительно также пропускать все пробелы (и символы табуляции), то лучше использовать функцию **SeekEoln** :

```
for i:= ... do
```

```
if not SeekEoln(fin) then Read(fin, Txt[i])
else Readln(fin);
```

И, наконец, если при чтении желательно брать только цифры, то можно в программу добавить "шаг назад" при каждом вводе не цифрового значения:

```
for i:= ... do
 begin
 Read(fin, Txt[i])
 if (Txt[i] < '0') or (Txt[i] > '9') then i:=i-1;
end;
```

2. Файл **DATF.TXT** – последовательный символьный (текстовый, форматный) набор вещественных данных, содержит 500 чисел в виде 50 строк (записей) по 10 чисел в каждой строке. Значения чисел не превосходят 1000000 по абсолютной величине. Пример открытия файла и чтения из него чисел с 32 по 35:

```
VAR
mas : array[1..4] of real;
fin : text;
. . .
BEGIN
. . .
Assign(fin,'D:\LAB1\DATF.TXT');
Reset(fin);
 пропуск 31 числа:
for i:=1 to 31 read (fin);
 чтение 4 чисел:
for i:=1 to 4 read(fin,mas[i]);
close(fin);
. . .
```

Примечание: Если читать из этого набора данных в символьный массив, рекомендуется пользоваться приемами, описанными для файла DATI.TXT.

3. Файл **DATI.BIN** – файл прямого доступа, представляет набор целочисленных данных, содержит 1000 двухбайтовых целых чисел. Должен описываться как файл из данных целого типа или массивов целого типа. Значения чисел не превосходят 999 по абсолютной величине. Пример открытия файла и чтения из него каждого 5-го числа:

```
VAR
dann,fict : integer;
fin : file of integer;
. . .
BEGIN
```

```
Assign(fin,'D:\LAB1\DATI.BIN');
Reset(fin);
While not Eof do
begin
for i:=1 to 4 do read(fin,fict); { пропуск 4 чисел, чте-
нием их в фиктивную переменную}
Read(fin,dann); { чтение числа }
. . .
end;
close(fin);    {закрытие файла }
```

4. Файл **DATF.BIN** – бесформатный (двоичный, внутримашинный) файл прямого доступа, представляет набор вещественных данных, содержит 500 шестибайтовых вещественных чисел. Должен подключаться к файловой переменной вещественного типа. Значения чисел не превосходят 99 по абсолютной величине. Пример открытия файла и чтения из него всех чисел после 90-го по 10 штук:

```
TYPE
MAS10 = array[1..10] of real;
VAR
dann : MAS10;
fin : file of MAS10;
. . .
BEGIN
. . .
Assign(fin,'D:\LAB1\DATI.BIN');
Reset(fin);
 пропуск 90 чисел, чтением их в переменную dann длиной 10
чисел 9 раз
for i:=1 to 9 do
 read(fin,dann);
While not Eof do {пока не кончится файл}
begin
 Read(fin,dann); { чтение очередных 10 чисел: }
. . .
end;
close(fin); { закрытие файла }
. . .
```

5. Файл **SIMBOL.TXT** – символьный (текстовый) файл последовательного доступа, представляет набор символьных данных, содержит 410 элементов (без переводов строк). Этот набор данных можно связывать как с текстовой, так и с типизированной (типа `char`) файловой переменной.

Пример 1. Чтение из файла с 20-го по 50-ый символы в одномерный символьный массив:

```
VAR
mas : array[0..30] of char;
fin : text;
. . .
BEGIN
. . .
Assign(fin,'D:\LAB1\SIMBOL.TXT');
Reset(fin);
 пропуск 19 символов:
for i:=1 to 19 read (fin);
 чтение 31 символа, с 20-го по 50-й:
for i:=0 to 30 read(fin,mas[i]);
close(fin);   закрытие файла
. . .
```

Пример 2. Чтение из файла с каждого четвертого символа в одномерный символьный массив длиной 40 байт:

```
TYPE
SimMas = array[1..40] of char;
VAR
Sim : SimMas;
fin : file of char;
i,j : integer; {i- счетчик элементов массива, j - номер
символа в файле }
. . .
BEGIN
. . .
Assign(fin,'D:\LAB1\SIMBOL.TXT');
Reset(fin);
. . .
j:=3; { нумерация в файле начинается с 0! }
for i:=1 to 40 do
begin
Seek(fin,j)          { ищем нужный номер символа в файле }
read(fin,Sim[i]);   { читаем его в очередной элемент мас-
сива }
j:=j+4;       { устанавливаем новый номер }
end;
close(fin); { закрытие файла }
. . .
```

6. Файл **RUS.TXT** – символьный (текстовый) файл последовательного доступа, представляет набор строк, длиной не более 80 символов кириллицы в строке. Файл может читаться только как текстовый.

Пример 1. Чтение из файла с 5-го по 9-ю строки в одномерный строчный массив:

```
TYPE Ss = string[80];
VAR    mas : array[1..5] of Ss;
       fin : text;
 . . .
BEGIN
 . . .
 Assign(fin,'D:\LAB1\RUS.TXT');
 Reset(fin);
  { пропуск 4-х строк: }
 for i:=1 to 4 do ReadLn (fin);
 { чтение 5-и строк: }
 for i:=1 to 5 do ReadLn (fin,mas[i]);
 close(fin); { закрытие файла }
. . .
```

7. Файл **LATS.TXT** – символьный (текстовый) файл. Может рассматриваться как файл последовательного доступа, если читать в строковые переменные, и как файл последовательного или прямого доступа, при чтении в символьные переменные. В последнем случае содержит он служебные символы возврата каретки и перевода строки. Представляет набор строк, длиной строго по 50 символов ASCII в каждой строке, не считая двух служебных символов.

Пример 1. Чтение из файла ASCII символов в символьный массив [2x55] элементов:

```
VAR
 mas : array[1..2,1..55] of char;
 fin : text;
 . . .
BEGIN
 . . .
 Assign(fin,'D:\LAB1\LATS.TXT');
 Reset(fin);
 for i:=1 to 2 do
  for j:=1 to 55 do
   if not Eol(Fin) then
    Read (fin,mas[i])
   else {пропуск конца записи (служебных символов)}
   begin
    ReadLn(fin);
    j:=j-1; { отмена смещения в массиве }
   end;
 close(fin); { закрытие файла }
. . .
```

Приложение Г.
Таблица ASCII-кодов (с альтернативной кодировкой)

Первая половина таблицы – стандартный набор кодов символов

№ п/п	Код 16-й	Символ	№ п/п	Код 16-й	Символ	№ п/п	Код 16-й	Символ	№ п/п	Код 16-й	Символ
0	00	(null)	32	20		64	40	@	96	60	`
1	01	☺	33	21	!	65	41	A	97	61	a
2	02	☻	34	22	"	66	42	B	98	62	b
3	03	♥	35	23	#	67	43	C	99	63	c
4	04	♦	36	24	$	68	44	D	100	64	d
5	05	♣	37	25	%	69	45	E	101	65	e
6	06	♠	38	26	&	70	46	F	102	66	f
7	07	•	39	27	'	71	47	G	103	67	g
8	08	◘	40	28	(	72	48	H	104	68	h
9	09	○	41	29	)	73	49	I	105	69	i
10	0A	◙	42	2A	*	74	4A	J	106	6A	j
11	0B	♂	43	2B	+	75	4B	K	107	6B	k
12	0C	♀	44	2C	,	76	4C	L	108	6C	l
13	0D	♪	45	2D	-	77	4D	M	109	6D	m
14	0E	♫	46	2E	.	78	4E	N	110	6E	n
15	0F	☼	47	2F	/	79	4F	O	111	6F	o
16	10	►	48	30	0	80	50	P	112	70	p
17	11	◄	49	31	1	81	51	Q	113	71	q
18	12	↕	50	32	2	82	52	R	114	72	r
19	13	‼	51	33	3	83	53	S	115	73	s
20	14	¶	52	34	4	84	54	T	116	74	t
21	15	§	53	35	5	85	55	U	117	75	u
22	16	▬	54	36	6	86	56	V	118	76	v
23	17	↨	55	37	7	87	57	W	119	77	w
24	18	↑	56	38	8	88	58	X	120	78	x
25	19	↓	57	39	9	89	59	Y	121	79	y
26	1A	→	58	3A	:	90	5A	Z	122	7A	z
27	1B	←	59	3B	;	91	5B	[	123	7B	{
28	1C	└	60	3C	<	92	5C	\	124	7C	|
29	1D	↔	61	3D	=	93	5D	]	125	7D	}
30	1E	▲	62	3E	>	94	5E	^	126	7E	~
31	1F	▼	63	3F	?	95	5F	_	127	7F	Δ

Вторая половина таблицы – альтернативный набор кодов символов

№ п/п	Код 16-й	Символ	№ п/п	Код 16-й	Символ	№ п/п	Код 16-й	Символ	№ п/п	Код 16-й	Символ
128	80	А	160	A0	а	192	C0	└	224	E0	р
129	81	Б	161	A1	б	193	C1	┴	225	E1	с
130	82	В	162	A2	в	194	C2	┬	226	E2	т
131	83	Г	163	A3	г	195	C3	├	227	E3	у
132	84	Д	164	A4	д	196	C4	─	228	E4	ф
133	85	Е	165	A5	е	197	C5	┼	229	E5	х
134	86	Ж	166	A6	ж	198	C6	╞	230	E6	ц
135	87	З	167	A7	з	199	C7	╟	231	E7	ч
136	88	И	168	A8	и	200	C8	╚	232	E8	ш
137	89	Й	169	A9	й	201	C9	╔	233	E9	щ
138	8A	К	170	AA	к	202	CA	╩	234	EA	ъ
139	8B	Л	171	AB	л	203	CB	╦	235	EB	ы
140	8C	М	172	AC	м	204	CC	╠	236	EC	ь
141	8D	Н	173	AD	н	205	CD	═	237	ED	э
142	8E	О	174	AE	о	206	CE	╬	238	EE	ю
143	8F	П	175	AF	п	207	CF	╧	239	EF	я
144	90	Р	176	B0	░	208	D0	╨	240	F0	Ё
145	91	С	177	B1	▒	209	D1	╤	241	F1	ё
146	92	Т	178	B2	▓	210	D2	╥	242	F2	Є
147	93	У	179	B3	│	211	D3	╙	243	F3	є
148	94	Ф	180	B4	┤	212	D4	╘	244	F4	Ї
149	95	Х	181	B5	╡	213	D5	╒	245	F5	ї
150	96	Ц	182	B6	╢	214	D6	╓	246	F6	Ў
151	97	Ч	183	B7	╖	215	D7	╫	247	F7	ў
152	98	Ш	184	B8	╕	215	D8	╪	248	F8	°
153	99	Щ	185	B9	╣	217	D9	┘	249	F9	•
154	9A	Ъ	186	BA	║	218	DA	┌	250	FA	·
155	9B	Ы	187	BB	╗	219	DB	█	251	FB	√
156	9C	Ь	188	BC	╝	220	DC	▄	252	FC	№
157	9D	Э	189	BD	╜	221	DD	▌	253	FD	¤
158	9E	Ю	190	BE	╛	222	DE	▐	254	FE	■
159	9F	Я	191	BF	┐	223	DF	▀	255	FF	

СОДЕРЖАНИЕ

Учебное издание

Информатика

Лабораторный практикум
по программированию
на Турбо-Паскале

ЛР № 020309 от 30.12.96

Подписано в печать 26.08.02 Формат 60x90$^1/_{16}$ Бумага кн.-журн. Печать офсетная.
Печ. л. 12,4. Гарнитура Таймс. Тираж 1000 экз. Зак. 161.

РГГМУ, 195196, СПб, Малоохтинский пр. 98
Российский государственный гидрометеорологический университет
Отпечатано в ООО «АСпринт»

38 = 00