ГОУ ВПО
КЕМЕРОВСКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ
ПИЩЕВОЙ ПРОМЫШЛЕННОСТИ

Г.И. СТАНЕВКО, Т.Г. КОЛЕСНИКОВА, В.А. ДАВЫДЕНКО

ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ: ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Для студентов вузов

Кемерово 2010

УДК 004.9 (075)
ББК 32.973-018
С 76

Рецензенты
В.Я. Карташов, доктор техн. наук, профессор;
В.С. Черкасов, канд. физ.-мат. наук, доцент.

Рекомендовано редакционно-издательским советом
Кемеровского технологического института пищевой
промышленности

Станевко Г.И.
С 76 Программирование и основы алгоритмизации: лабораторный практикум / Г.И. Станевко, Т.Г. Колесникова, В.А. Давыденко. – Кемерово: КемТИПП, 2010. – 127 с.
ISBN

Лабораторный практикум предназначен для проведения лабораторных работ по дисциплине «Программирование и основы алгоритмизации». В него включены 15 лабораторных работ. В каждой работе излагаются теоретически основы по теме выполняемой работы, приводятся демонстрационные примеры, предлагаются вопросы по контролю входных знаний и задания с вариантами для самостоятельного выполнения. Выбор изучаемых тем соответствует содержанию Государственного образовательного стандарта высшего профессионального образования.

Материал пособия охватывает широкий круг алгоритмов обработки структур данных. Лабораторный практикум предназначен для студентов специальности 220301 - Автоматизация процессов и производств (по отраслям) всех форм обучения.

УДК 004.9 (075)
ББК 32.973-018

ISBN

ОГЛАВЛЕНИЕ

Оглавление 3
Предисловие 4
Модульное программирование 5
Построение программного меню 11
Алгоритмы работы экрана в текстовом режиме 16
ООП. Разработка программы с объектом «Окно» 24
ООП. Наследование 27
ООП. Полиморфизм. Сокрытие полей и методов 32
Обработка символьной информации 36
Обработка данных типа записи 47
Формирование и обработка переменных файлового типа. Типизированные файлы 53
Формирование и обработка данных файлового типа. Текстовые файлы 65
Динамические структуры данных 75
Работа в графическом видеорежиме 82
Последовательности, рекуррентные соотношения 90
Рекурсивные алгоритмы 98
Алгоритмы на множествах 109
Библиографический список 123
Приложение 124

ПРЕДИСЛОВИЕ

Лабораторный практикум является составной частью методического обеспечения дисциплины «Программирование и основы алгоритмизации» и отвечает требованиям Федерального Государственного Образовательного Стандарта специальности 220301 – Автоматизация технологических процессов и производств.

В его состав входят 15 лабораторных работ. Каждая лабораторная работа посвящена определенной теме и построена по единому сценарию:

1. Изложены теоретические основы, вводящие в проблему тематики выполняемой работы. Изложение сопровождается демонстрационными примерами;
2. Сформулированы вопросы для контроля входных знаний;
3. Приведены задания, подлежащие выполнению.

Основное внимание уделено разработке алгоритмов решения задач по изучаемой проблеме.

В качестве языка программирования выбран язык высокого уровня Turbo Pascal 7.0.

Лабораторная работа

МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ

Основы теории

При разработке больших программ целесообразно часть подпрограмм и других ресурсов, таких как константы, объявление типов, собирать вместе и компилировать отдельно от основной программы в виде библиотек ресурсов или модулей.

Модуль – это автономно компилируемая коллекция программных ресурсов, предназначенная для использования другими модулями и программами.

Модульное программирование – это технология программирования, приводящая к значительному уменьшению сроков разработки программ и количества программных ошибок.

Все ресурсы модуля делятся на две группы: ***внешние*** – предназначенные для использования другими программными единицами, и ***внутренние*** – рабочие ресурсы данного модуля.

Структура модуля имеет вид:

Unit < имя модуля >;
Interface <интерфейсная секция>
Implementation <секция реализации>
[***BEGIN*** <секция инициализации> **]**
END.

Имя модуля *должно совпадать с именем файла*, в котором он содержится.

Интерфейсная секция содержит объявления ресурсов (в том числе заголовки подпрограмм), к которым возможны обращения извне.

Секция реализации содержит описание подпрограмм, объявленных в интерфейсной секции, и описание внутренних ресурсов модуля. Обращение к этим ресурсам возможно только из подпрограмм, описанных в том же модуле.

Секция инициализации содержит операторы, которые выполняют некоторые действия, необходимые для нормальной работы процедур модуля (например, открывают файлы,

изменяют цвет, выдают информацию о разработчике и т.п.) Операции этой секции выполняются один раз при включении модуля до начала выполнения основной программы. Эта секция в модуле может отсутствовать, что показано включением ее в квадратные скобки (**[ ]**).

В результате компиляции модуля система формирует одноимённый файл, имеющий расширение *.tpu.*

Среда языка Borland Pascal предусматривает три режима компиляции программы, использующей модули (главное меню, пункт *Compile*):

- *Compile* – компилируется только основная программа, все модули должны быть предварительно откомпилированы в *tpu–файлы* и размещены либо в текущем каталоге, либо в одном из каталогов, указанных как источники *tpu–файлов* в настройках среды (*Options/Directories*);
- *Make* – модули, для которых не обнаружены *tpu–файлы* компилируются из соответствующих *pas–файлов,* которые должны находиться либо в текущем каталоге, либо в одном из каталогов, указанных как источники *pas–файлов* в настройках среды (*Options/Directories*);
- *Build* – все ранее откомпилированные *tpu–модули* игнорируются и все модули компилируются из своих исходных файлов заново.

В процессе отладки модулей целесообразно использовать режим *Build,* а при отладке программы – режим *Compile*.

Внимание!
При запуске программы на выполнение (*Ctrl+F9*) программа и подключаемые к ней пользовательские модули должны быть размещены либо в текущем каталоге, либо в одном из каталогов, указанных в настройках среды (*Options/Directories*). В противном случае система выдаст ошибку (*15 File not found* – файл не найден).

Демонстрационный пример

```
Unit Mod_Mas1; {Имя файла: Mod_Mas1.pas}
Interface
{Блок объявления типов:}
    Type TM1_r = array[1..15] of real;
{Блок объявления подпрограмм:}
    Procedure Input (var n: byte; var X: TM1_r); {возвращает количество элементов массива X и их значения}
    Procedure Print (n: byte; X: TM1_r); {выводит на экран количество элементов массива X и их значения}
    Function Sum (n: byte; X: TM1_r): real; {возвращает сумму элементов массива X}
Implementation
Uses Crt;
Procedure Input;
    Var i: byte; {i – локальный ресурс – параметр цикла}
Begin
        writeln ('Введите:');
        write ('количество элементов массива = ');
        readln (n);
        writeln ('значения элементов: ');
        For i:= 1 to n do
        begin
                write ('X[', i, '] = ');
                readln (X[i]);
        end;
End;

Procedure Print;
    Var i: byte; {i – локальный ресурс – параметр цикла}
Begin
        writeln ('Значения элементов массива равны: ');
        For i:= 1 to n do
                writeln ('X[', i, '] = ', X[i]: 8: 2);
End;
```

```
Function Sum;
  Var  i: byte;  {i – локальный ресурс – параметр цикла}
       s: real;  {s – локальный ресурс для накопления суммы}
Begin
       s:= 0;
       For i:= 1 to n do
                          s:= s+X[i];
       Sum:= s; {присваивание выполнено для передачи значе-
                  ния суммы во внешнюю среду}
End;
```

```
BEGIN
       ClrScr;
       writeln ('Модуль разработан Ивановым М.Ф.');
       readln;
END.
```

Контроль входных знаний

1. Чем отличается модульное программирование от процедурного?
2. Как подключить пользовательский модуль к программе?
3. Какой пункт главного меню содержит информацию о размещении *tpu* и *pas–файлов*?
4. Ваши действия в случае, если при запуске программы на выполнение система выдаст ошибку '*15 File not found*'?
5. Модуль – это отдельно компилируемая или отдельно исполняемая конструкция языка?
6. Какому условию должны удовлетворять имя модуля и имя файла, хранящего этот модуль?
7. Какая секция модуля должна быть отражена в инструкции при передаче его в пользование?
8. Объявите в модуле подпрограмму-функцию и подпрограмму-процедуру нахождения номера наибольшего по значению элемента одномерного массива.

Задания для выполнения

Средствами модульного программирования решить задачу обработки одномерного числового массива (см. табл. 1).

Таблица 1

Варианты заданий

№ варианта	Задание
1	Найти сумму элементов до первого положительного и произведение остальных элементов.
2	Найти количество чётных элементов, попавших в интервал *[a, b]*.
3	Найти количество элементов, стоящих на нечётных местах и меньших заданного числа.
4	Найти количество элементов, кратных заданному числу и не попавших в интервал *[a, b]*.
5	Найти количество нечётных элементов до первого положительного и произведение элементов после него.
6	Найти произведение элементов, стоящих на нечётных местах и меньших заданного числа.
7	Найти сумму элементов кратных заданному числу, расположенных до первого положительного и произведение положительных элементов после него.
8	Найти количество элементов до первого отрицательного, попавших в интервал *[a, b]*, и количество отрицательных элементов после него.
9	Положительные элементы массива расставить в порядке убывания.
10	Чётные элементы массива расставить в порядке убывания.
11	Вставить заданное число после последнего отрицательного элемента массива.
12	Вставить заданное число после максимального элемента массива.

Окончание табл. 1

№ варианта	Задание
13	Найти первый элемент, кратный заданному числу и произведение отрицательных элементов после него.
14	Вставить минимальный элемент перед элементом, равным заданному значению.
15	Найти сумму элементов, стоящих на нечётных местах и не превышающих длину интервала *[a, b]*.
16	Вставить последний отрицательный элемент после минимального элемента массива.
17	Вставить минимальный элемент после первого, равного нулю.
18	Найти среднее значение чётных элементов до первого положительного и произведение отрицательных элементов после него.
19	Неположительные элементы массива расставить в порядке убывания.
20	Вставить заданное число перед максимальным элементом массива.
21	Вставить минимальный элемент после первого отрицательного элемента массива.
22	Удалить первый положительный элемент, расположенный после минимального элемента массива.
23	Положительные элементы массива расставить в порядке возрастания.
24	Вставить минимальный элемент после первого, кратного заданному числу.
25	Нечётные элементы массива расставить в порядке возрастания.

Лабораторная работа

ПОСТРОЕНИЕ ПРОГРАММНОГО МЕНЮ

Основы теории

Пользовательский интерфейс меню обычно имеет вид:

МЕНЮ
1. Ввод данных
2. Обработка данных
3. Вывод результатов
4. Выход

Ваш выбор –

Любой пункт меню может, в свою очередь, содержать вызов подменю.

Алгоритм построения меню основан на дополнительной структуре структурного программирования – *операторе выбора*:

Case <управляющий параметр-код> ***of***
{значения параметра:}
P1: <оператор 1>;
P2: <оператор 2>;
. . .
PN: <оператор N>
else < альтернативный оператор>;
end;

где *управляющий параметр-код* – выражение или переменная любого перечисляемого типа;
P1, P2, …, PN – значение управляющего параметра;
<оператор 1>, …,<оператор N> – операторы, вызывающие подпрограммы, ответственные за конкретные пункты меню, и могут быть простыми или составными;

альтернативный оператор – простой или составной оператор, выполняемый при значении управляющего параметра, не равного ни одному из значений *P1, P2, ..., PN*;
часть *else* как необязательная может отсутствовать.

Возврат в меню осуществляется через цикл по управляющему параметру; выход – через *<оператор N>* или *<альтернативный оператор>*.

Для организации размещения меню на экране используют оконный режим, который реализуется с помощью сервисных процедур модуля *CRT*:

Window (X1, Y1, X2, Y2: byte) – задание текущего прямоугольного окна с левой верхней вершиной *(X1, Y1)*, и правой нижней *(X2, Y2)*;

TextColor (C: byte) – выбор цвета символов на экране;

TextBackGround (C: byte) – выбор цвета фона активного окна;

GotoXY (X, Y: byte) – установка курсора в строку *X* и столбец *Y* текущего окна, позволяет размещать меню в удобном для каждой программы месте экрана, а текст – по центру текущего окна. Переменная *X* – номер позиции в строке экрана, *Y* – номер строки. Отсчёт позиций и строк экрана ведётся от левого верхнего угла, $1 \le X \le 80$, $1 \le Y \le 25$.

Действия процедуры *ClrScr* следует рассматривать как процесс заливки экрана выбранным цветом.

Алгоритм создания окна и задания цветов символов и фона:

- создаём окно;
- задаём цвет символов;
- задаём цвет фона под символами;
- заливаем окно цветом фона.

Перед завершением работы программы необходимо восстановить системные установки:

- размеры окна – экрана, задав координаты левой верхней вершины *(1, 1)* и правой нижней – *(80, 25);*
- жёлтый цвет символов и синий цвет фона.

Контроль входных знаний

1. Чем отличается оператор *Case* от условного оператора?
2. Как записать значение управляющего параметра символьного типа?
3. Возможно ли присутствие в одном операторе *Case* нескольких управляющих параметров?
4. Заголовок подпрограммы *MENU*, выводящей на экран текст меню и запрашивающей значение управляющего параметра для передачи его во внешнюю среду, имеет вид:

   Procedure MENU;

   Какая ошибка допущена в её заголовке?
5. Пункт *"Обработка данных"* главного программного меню содержит в свою очередь подменю, состоящее из *m* пунктов. Как организовать возврат из этого подменю в главное меню?

Задания для выполнения

Разработать программу, управляемую через программные меню. Пункты главного меню должны обеспечивать ввод, вывод и обработку исходных данных, кроме этого пункт *"Обработка данных"* должен обеспечивать вызов дополнительного подменю, подпрограммы которые реализуют обработку данных согласно решаемой задаче (см. табл. 2). В программе предусмотреть её реакцию на отсутствие требуемых данных.

Таблица 2

Варианты заданий

№ варианта	Задание
Организовать ввод и вывод элементов одномерного массива. Вывести значение требуемого элемента массива:	
1	a) первого чётного; b) наибольшего.
2	a) первого отрицательного; b) последнего, равного заданной величине.

Продолжение табл.2

№ варианта	Задание
3	a) первого, целая часть которого меньше заданной величины; b) последнего кратного произвольному значению *q*.
4	a) последнего, большего заданной величины; b) первого, дробная часть которого равна нулю.
Организовать ввод и вывод элементов одномерного массива. Вывести значение требуемых сумм элементов массива:	
5	a) всех, дробная часть которых равна нулю; b) меньших заданной величины.
6	a) попавших в заданный интервал; b) отрицательных.
7	a) не попавших в заданный интервал; b) чётных.
8	a) кратных произвольному значению *q;* b) положительных.
9	a) меньших заданной величины; b) дробная часть которых больше произвольного значения *q*.
10	a) положительных; b) принадлежащих заданному интервалу.
Организовать ввод и вывод элементов двумерного массива. Определить номер строки и номер столбца, на пересечении которых находятся требуемые элементы массива:	
11	a) наибольший; b) первый положительный.
12	a) наименьший; b) первый чётный.
13	a) первый, равный заданной величине; b) последний чётный.
14	a) последний отрицательный; b) первый, попавший в заданный интервал.

Окончание табл. 2

№ варианта	Задание
15	a) последний, не попавший в заданный интервал; b) первый отрицательный, принадлежащий главной диагонали.
Организовать ввод и вывод элементов одномерного массива. Сформировать новые массивы из элементов:	
16	a) чётных; b) попавших в заданный интервал.
17	a) больших некоторой заданной величины; b) совпадающих по значению со своими порядковыми номерами.
18	a) отрицательных; b) нечётных.
19	a) кратных некоторому числу; b) расположенных после элемента с заданным номером.
Организовать ввод и вывод элементов двумерного массива. Определить суммы требуемых элементов:	
20	a) принадлежащих главной диагонали; b) всех положительных.
21	a) расположенных выше главной диагонали; b) всех чётных.
22	a) расположенных ниже главной диагонали; b) всех, не превышающих заданной величины.
23	a) принадлежащих заданной строке; b) всех, целая часть которых меньше некоторой величины.
24	a) принадлежащих заданному столбцу; b) всех, попавших в заданных интервал.
25	a) до первого отрицательного; b) положительных, принадлежащих побочной диагонали.

Лабораторная работа

АЛГОРИТМЫ РАБОТЫ ЭКРАНА В ТЕКСТОВОМ РЕЖИМЕ

Основы теории

Экран в текстовом режиме представляется как совокупность строк и столбцов. Каждый символ располагается на так называемом ***знакоместе***, расположенном на пересечении строки и столбца.

В Turbo Pascal работа экрана в текстовом режиме обеспечивается средствами модуля ***CRT.***

Аббревиатура *CRT* расшифровывается по-русски как «электроннолучевая трубка». По умолчанию программа в Turbo Pascal использует режим, при котором на экране выделяется 25 строк и 80 столбцов.

Модуль *CRT* содержит типы, константы, переменные и подпрограммы, которые позволяют:

- выполнять вывод в заданное место экрана заданным цветом символа и фона;
- открывать на экране окна прямоугольной формы и выполнять вывод в пределах этих окон;
- очищать экран, окно, строку и её часть;
- обрабатывать ввод с клавиатуры: управлять курсором, проводить опрос клавиатуры;
- управлять встроенным динамиком.

Буфер клавиатуры – участок оперативной памяти, организованный по принципу очереди, в котором может храниться до 127 символов, вводимых с клавиатуры.

При подключённом модуле *CRT* можно выводить на дисплей строки и символы, содержащие в себе управляющие коды (коды 0..31). При этом они не будут оказывать управляющие воздействия, а будут изображаться на дисплее, согласно таблице изображения символов по их ASCII-коду. Исключение составляют лишь четыре кода:

#07 – вызывает один короткий звук динамика;

#08 – перемещает курсор влево на один символ, если есть куда сдвинуться в пределах строки; в противном случае не имеет эффекта;

#10 – перемещает курсор на строку ниже, не меняя текущего столбца. Если курсор находился в последней строке экрана, то информация выдаётся на первой строке нового экрана, то есть экран «прокручивается» на строку вверх;

#13 – перемещает курсор в начало текущей строки.

- Процедуры и функции модуля *CRT*
  - ✓ Работа с экраном в целом

Процедура *Window (X1, Y1, X2, Y2: byte)* – создаёт текущее окно, где *(X1, Y1)* – координаты верхней левой вершины, и *(X2, Y2)* – правой нижней вершины.

Координаты текущего окна хранятся в специальных переменных *WindMin: word* и *WindMax: word*. Для их определения используются функции *Lo (WM: word): byte* – левый и *Hi (WM: word): byte* – правый, поэтому

X1 = Lo (WindMin); Y1 = Hi (WindMin);
X2 = Lo (WindMax); Y2 = Hi (WindMax);

Процедура *ClrScr* – заливает текущее окно экрана текущим цветом.

- ✓ Позиционирование курсора

Процедура *GotoXY (X, Y: byte)* – устанавливает курсор в столбец *X*, строку *Y*.

Функция *WhereX: byte* – выдаёт номер текущего столбца.

Функция *WhereY: byte* – выдаёт номер текущей строки.

Процедура *ClrEOL* – стирает все символы строки справа от курсора.

Процедура *InsLine* – вставляет пустую строку на место текущей.

Процедура *DelLine* – удаляет текущую строку.

- ✓ Настройка цвета

Процедура *TextColor (C: byte)* – задаёт цвет символа.

Процедура *TextBackGround (C: byte)* – задаёт текущий цвет фона окна.

Для кодировки цвета используются десятичные числа 0..15 и 128.

Black = 0	{черный}	*DarkGrey = 8*	{темно-серый}
Blue = 1	{синий}	*LightBlue = 9*	{светло-серый}
Green = 2	{зеленый}	*LightGreen = 10*	{светло-зеленый}
Cyan = 3	{голубой}	*LightCyan = 11*	{светло-голубой}
Red = 4	{красный}	*LightRed = 12*	{розовый}
Magenta = 5	{фиолетовый}	*LightMagenta = 13*	{сиреневый}
Brown = 6	{коричневый}	*Yellow = 14*	{желтый}
LightGrey = 7	{светло-серый}	*White = 15*	{белый}
Blink = 128	{мерцание}		

Текущие цвета фона и символа можно определять через выше приведенные процедуры или через значение специальной переменной *TextAttr: word*, через которую цвет фона определяется как

(TextAttr div 16) mod 8,

а текущий цвет символа как

TextAttr mod 8,

где *div* – целочисленное деление,
mod – остаток от деления нацело.

Так, приведенные ниже операторы выполняют следующие действия:

```
TextAttr:= 16 + 14;      {выделяет красные символы на синем
                          фоне}
```

TextAttr:= 2·16 + 15; {выделяет белые символы на зеленом фоне}

✓ Подача звуковых сигналов

Процедура *Sound (Hz: word)* – включает звук с частотой *Hz* в герцах.

Процедура *NoSound* – выключает звук.

✓ Использование встроенного таймера

Delay (ms: word) – задержка процесса (пауза) в *ms* миллисекунд.

✓ Опрос клавиатуры

Функция *KeyPressed: Boolean* – описывает состояние буфера клавиатуры и принимает значение *true*, если в буфере есть хотя бы один символ, и *false*, если буфер пуст.

Функция *ReadKey: char* – анализирует буфер клавиатуры и если он не пуст, то в качестве результата возвращает первый символ буфера. В противном случае ожидается нажатие на любую клавишу.

Часто при организации диалога используются два цикла, организованные с участием этих функций.

Первый цикл – цикл ***ожидания нажатия*** любой клавиши:

repeat until Keypressed;

приводит к ожиданию нажатия любой клавиши, вырабатывающей код, при условии, что буфер клавиатуры пуст (*KeyPressed = False*). Если же буфер клавиатуры содержит хотя бы один код (*KeyPressed = True*), то этот цикл не приводит ни к каким действиям и управление передаётся следующему за ним оператору.

Таким образом, чтобы можно было корректно использовать циклы ожидания нажатия клавиш, необходимо предварительно очищать буфер клавиатуры от кодов случайно

нажатых пользователем клавиш. Для этого применяется второй цикл – цикл очистки буфера клавиатуры:

while KeyPressed do ch:= ReadKey; {*ch* – переменная типа *char*}

Информация, считанная при нажатии клавиши, поступает в буфер клавиатуры и занимает в нём очередной байт. Нажатие цифровых, буквенных и символьных клавиш посылает в буфер клавиатуры ASCII-код (один символ). Нажатие управляющих функциональных клавиш F1-F10, клавиш управления курсором, клавиш Enter и Escape, а также совокупности клавиш Alt +…, Ctrl +…, посылает в буфер клавиатуры не один, а два символа, первый из которых #0. Такая пара символов называется ***расширенным кодом клавиатуры***.

В таблице 3 приведены коды функциональных клавиш.

Таблица 3

Коды функциональных клавиш

Клавиша	Нажатие	Клавиша	Нажатие
A–Z	65–90	End	079
↑	072	Delete	083
→	077	Page Up	073
↓	080	Page Down	081
←	075	F1–F10	059–068
Ins	082	F11	0133
Home	071	F12	0134

Функция *Chr (x: byte): char* – возвращает символ, соответствующий ASCII-коду *x*, *x* – переменная, константа или выражение типа *byte* ($0 \le x \le 255$).

Пример 1.

writeln (Chr (17)); {Результат – символ ◄}

Функция *Ord (ch: char): byte* – возвращает числовой код, соответствующий символу *ch*.

Пример 2.

```
writeln (Ord (‘A’)); {Результат – числовой код = 65}
```

Распознать принадлежность нажатой клавиши к простому или расширенному коду можно по алгоритму, реализованному фрагментом примера 3.

Пример 3.

```
Var
    ch1, ch2: char;  {символьные значения нажатой клавиши}
    …
    ch1:= Readkey; {считываем нажатие клавиши}
    if ch1 = #0 then {если код расширенный}
                    begin
                          ch2:= Readkey; {считываем второй символ расширенного кода}
                          writeln (‘Расширенный код = ’, ord (ch2))
                    end
                else
                          writeln (‘Код = ’, ord (ch1));
```

Функции опроса клавиатуры в совокупности с оператором выбора *Case* позволяют реализовывать процессы обработки нажатий только на определённые клавиши и даже переопределять функции, закреплённые за этими клавишами.

Пример 4.

Переопределение функций, закреплённых за клавишами: PgUp, PgDn, → , ← ; обработка нажатий на эти клавиши и клавишу Enter.

```
Var
    ch1, ch2: char; {символьные значения нажатых клавиш}
    p: byte;        {ключ для оператора Case}
    k: integer;     {счётчик повторений цикла}
    …
```

```
BEGIN
    …
    k:= 0;
    repeat
        k:= k + 1;
        while KeyPressed do ch:= ReadKey;{очищаем буфер клавиатуры}
        ch1:= Readkey; {считываем нажатие клавиши}
        if ch1 = #0 then {если код расширенный – клавиша управляющая}
                    ch2:= Readkey; {считываем второй символ расширенного кода}
        p:= Ord (ch2); {определяем ASCII-код, нажатой клавиши}
        gotoXY (40, 12);
        Case p of
            73: begin GotoXY (40, 5); write (k, 'Маше дали кашу');
                end;{PgUp}
            81: begin GotoXY (40, 2); write (k, 'Маша съела кашу');
                end;{PgDn}
            77: begin GotoXY (60, 12); write (k, 'Маша ест кашу');
                end;{→}
            75: begin GotoXY (10, 12); write (k, 'Маша, ешь кашу!');
                end;{←}
        end;
    until (ch1 = #13) or (k = 20); {#13 – Enter}
        …
END.
```

Контроль входных знаний

1. Какие функции выполняет функция *ReadKey*?
2. К моменту выполнения оператора *Ch1:= ReadKey;* в буфере клавиатуры находились коды нажатия клавиш Delete, End и F1. По запросу этого оператора Вы нажали клавишу Enter. Код какой клавиши будет считан в переменную *ch1*?
3. Какой вид имеет цикл очистки буфера?

4. Как обрабатываются расширенные коды клавиатуры?
5. Какие функции выполняет функция *KeyPressed*?
6. В программе выполняется цикл, управляемый функцией *KeyPressed*. Как отразится на его выполнении случайное нажатие на любую клавишу?
7. При выводе ASCII-кодов 0..31 символы каких кодов не будут отображены на экране?

Задания для выполнения

1. Открыть 4 окна разного цвета и вписать:
   - в первое – тему лабораторной работы;
   - во второе – фамилию, имя, отчество, номер группы исполнителя и ASCII-коды фамилии;
   - интерфейсом третьего окна должен быть текст меню:

МЕНЮ
1. Ввод данных
2. Обработка данных
3. Вывод результатов
4. Выход

Ваш выбор –

   - в четвертом окне разместить символы, соответствующие ASCII-кодам из интервала *[0..n]*, где *n* – количество символов Вашего полного имени.
2. Во втором окне реализовать процесс отображения бегущей строки.
3. В третьем окне реализовать циклический процесс, управляемый клавишей Escape (код #27), в котором нажатие управляющей клавиши из совокупности: ←, →, ↑, ↓ вызывает изменение цвета символов соответствующего ей пункта меню.

Лабораторная работа

ООП. РАЗРАБОТКА ПРОГРАММЫ С ОБЪЕКТОМ «ОКНО»

Основы теории

Объектно-ориентированное программирование (ООП) – это технология программирования, базирующаяся на свойствах:

- инкапсуляции – объединения данных и методов их обработки в одну конструкцию при определении объектного типа;
- наследования – создания иерархии объектных типов с тем, чтобы поля данных и методы предков автоматически были полями данных и методами потомков;
- полиморфизма – такого определения методов в иерархии классов, чтобы метод с одним именем мог применяться к различным родственным объектам, сохраняя при этом возможность переопределять его собственными действиями.

Объектный тип в дальнейшем будем называть ***классом,*** *данные* – ***полями*** (аналогично типу ***record*** – запись), а *процедуры* и/или *функции*, обрабатывающие эти поля – ***методами***.

Объявление класса осуществляется в разделе объявления типов программы или модуля по структуре:

Type <имя класса> = ***object***
{объявление полей данных}
{объявление методов, содержащее только их заголовки}
end;

Полное объявление методов размещается в описательной части программы или в разделе *Implementation* модуля. При этом список формальных параметров опускается, а перед именем метода через точку указывается имя класса, к которому этот метод принадлежит.

Синтаксис полного объявления методов имеет вид:

- для метода-процедуры:

procedure <имя класса>**.**< имя метода>;
{объявление местных ресурсов}
<тело подпрограммы-процедуры>;

- для метода-функции:

function <имя класса>**.**< имя метода>;
{объявление местных ресурсов}
<тело подпрограммы-функции>;

Инициализацию полей класса принято возлагать на один из его методов, который является процедурой и, по установившейся традиции, поименован как метод ***Init***.

Переменная, объявленная через конкретный класс, называется ***экземпляром*** этого класса или ***объектом*** и объявляется по стандартным правилам объявления переменных.

Обращение к полям и методам объекта осуществляется через *составное имя*

<имя объекта>**.**<вызываемое поле или метод>

➢ Вспомогательный материал

1. Оператор присоединения, применяемый при работе с составными именами:

With … do begin … end;

2. Функция *TextColor (C: byte)* модуля *CRT* – устанавливает цвет символов (*C = 0..15; C = 128* – мерцание).
3. Функции *WindMax* и *WindMin* типа *word* хранят информацию о размерах текущего окна.

Контроль входных знаний

1. Что общего в данных типа *record* и типа *object*?
2. В чём заключается свойство инкапсуляции и на каком этапе конструирования объекта оно используется?
3. Какая часть процедур и функций включается в класс?
4. Методы *Init* и *MakeWin* класса *Win* объявлены как

```
Procedure Init (...);
Procedure MakeWin;
```

   Как оформить их заголовки при полном описании методов?
5. Какая функция при конструировании класса возлагается на метод *Init*?
6. Переменные *a, b, c* типа *real* являются полями некоторого класса. Как объявить метод их инициализации? Как организовать вызов этого метода из основной программы?
7. Чем обосновано отсутствие списка формальных параметров при объявлении прочих методов класса, кроме метода инициализации полей?

Задания для выполнения

Дан фрагмент объявления класса *Win*:

```
Type Win = Object  {класс ОКНО}
                   {Поля:}
                      x1, y1,          {координаты верхнего левого угла окна}
                      x2, y2,          {координаты нижнего правого угла окна}
                      colf: byte;      {цвет фона}
                   {Заголовки методов:}
                   {инициализация полей:}
                      Procedure Init (<список формальных параметров>);
                   {создание окна:}
                      Procedure MakeWin;
                   {задание размеров окна по оси OX:}
                      Function GetSizeX: byte;
           End;
```

```
Procedure Win. MakeWin;
Begin
    Window (x1, y1, x2, y2);   {изображение текущего окна на
                                экране}
    Textbackground (colf);     {установка фона: colf = 0..7}
    ClrScr;
End;
```

1. Дополнить класс функцией задания размеров окна по оси OY.
2. Разработать модуль, содержащий данный класс.
3. Разработать программу, выводящую на экран:
   - цвет и размеры окна;
   - окно выбранного Вами цвета;
   - три окна разных размеров и цветов;
   - случайное количество окон случайных цветов.
4. Дополнить класс методом восстановления размеров и цвета окна, предусмотренных по умолчанию.
5. Вписать в одно окно Вашу фамилию, в другое – имя. Дополнить класс методом вывода строковых данных в заданное окно.
6. Сформировать окно(а) и разместить в нём/них название лабораторной работы.

Лабораторная работа

ООП. НАСЛЕДОВАНИЕ

Основы теории

Условимся в дальнейшем *тип объекта* называть ***классом***, а *экземпляр класса* называть ***объектом.***

Наследование называют соотношение между классами, при котором один класс строится на базе другого посредством добавления полей и определения новых методов. При этом исходный класс, на базе которого выполняется построение, назы-

вают ***родительским***, ***предком*** или ***базовым***, а строящийся класс – ***дочерним***, ***потомком***, ***наследником*** или ***производным***.

Иерархия классов с наследованием образует ***дерево классов***, в корне которого находится родительский класс. Дерево может иметь несколько уровней и ветвей, на каждом уровне добавляются необходимые поля и методы.

Потомок наследует все поля и методы родителя и дополняет их собственными полями и методами.

В результате использования механизма наследования отпадает необходимость заново описывать в наследнике уже существующие в классе-родителе поля и методы. Требуется описать только те поля и методы, которых недостаёт в родителе.

Отношения между различными классами проекта принято иллюстрировать *диаграммой отношений классов*, такую диаграмму называют ***иерархией классов*** (Рис. 1).

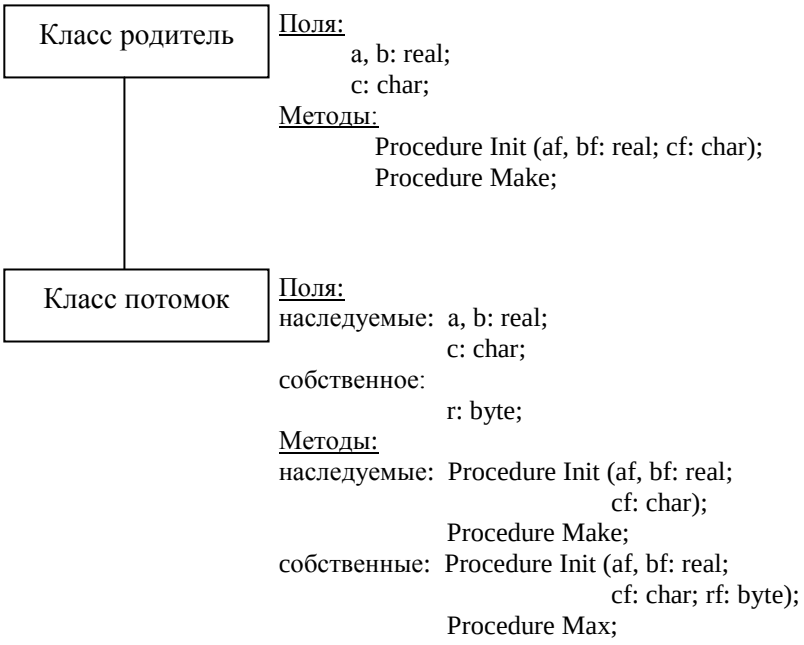


Рис. 1. Демонстрация иерархии классов

В иерархии классов методы можно *переопределять*, т.е. называть *одинаковыми именами*, поля же переопределять нельзя.

Поиск метода в иерархии классов выполняется следующим образом:

1. В первую очередь компилятор устанавливает тип класса.
2. Ищет метод в этом классе и если находит, то подключает его.
3. Если метода в данном классе нет, то идёт поиск в родительском классе. В случае успеха вызывается метод родителя.
4. Если метод в родителе не найден, то поиск продолжается в классах-предках (в направлении к корню дерева) до тех пор, пока вызванный метод не будет найден.

Если компилятор не обнаруживает метод, то он фиксирует ошибку 44 (*Field indentifier expented* – ожидается имя поля или метода класса).

- Объявление класса-потомка

  Структура объявления класса-потомка имеет вид:

Type <имя класса-потомка> = ***Object*** **(**<имя класса-родителя>**)**
<собственные поля класса-потомка>;
<собственные методы класса-потомка>;
End;

- Активизация полей и методов родительского класса.

Доступ к полям родителя осуществляется в потомке как к собственным. Обращение же к методам осуществляется через составное имя:

*<имя класса-родителя>***.***<вызываемый метод>;*

или через использование служебного слова ***inherited*** (наследуемый):

inherited <вызываемый метод>;

- Вспомогательный материал

Процедуры *Inc (k)* и *Dec (k)* – формируют соответственно следующее и предыдущее значения переменной *k*.

Процедуры *Inc (k, n)* и *Dec (k, n)* – соответственно увеличивают и уменьшают значения переменной *k* на *n*.

Контроль входных знаний

1. В чём заключается наследование?
2. Что иллюстрирует диаграмма иерархии классов?
3. Можно ли из класса-наследника вызвать метод класса-предка? Если такой вызов возможен, то объясните почему. Возможен ли вызов метода в обратном направлении?
4. Что произойдёт, если в классе-наследнике не предусмотрена инициализация полей родительского класса?
5. Класс-родитель *T_obj1* имеет поля *a1*, *b1*, *c1*, инициализация которых осуществляется через метод

```
Procedure Init (a1f, b1f, c1f: real);
Begin
        a1:= a1f;
        b1:= b1f;
        c1:= c1f;
End;
```

Класс-потомок *T_Obj2* располагает собственными полями *d, p.* Правильно ли реализован его метод инициализации?

```
Procedure Init (a1f, b1f, c1f: real; df, pf: integer); {Переопределённый метод}
Begin
        Init (a1f, b1f, c1f); {Вызов родительского метода}
        d:= df;
        p:= pf;
End;
```

Задания для выполнения

Дана иерархия классов:

Разработать:

- класс-потомок для вывода символа в заданную позицию первого окна;
- класс-потомок для *N*-кратного вывода символа с заданной позиции второго окна.

Лабораторная работа

ООП. ПОЛИМОРФИЗМ. СОКРЫТИЕ ПОЛЕЙ И МЕТОДОВ

Основы теории

Полиморфизм или ***сложное наследование*** **–** это свойство родственных объектов решать схожие по смыслу проблемы разными способами. Изменяя алгоритм того или иного метода в потомках, можно придавать этим потомкам отсутствующие у родителя специфические свойства. Для изменения метода необходимо *перекрыть* его в потомке, т.е. объявить в потомке одноимённый метод и реализовать в нём нужные действия. В результате в объекте-родителе и в объекте-потомке будут действовать *два одноимённых метода*, имеющие разную алгоритмическую основу и, следовательно, придающие объектам разные свойства. Это и называется полиморфизмом объектов.

Паскаль реализует механизм *простого и сложного полиморфизма.*

Простой полиморфизм – использование одинакового имени для методов с различной реализацией. Одноимённые методы в этом случае называются статически полиморфными и могут иметь различные списки формальных параметров. Такое название методов связано с тем, что размещение ссылок на них осуществляется в программе ещё *на этапе компиляции*, т.е. при так называемом ***раннем связывании.*** Каждый объект вызывает свой метод.

Сложный полиморфизм **–** это не только описанный выше механизм наследования и перекрытие методов родителя, но и их ***виртуальность,*** позволяющая *родительским методам обращаться к методам потомков*. При сложном полиморфизме *ссылки* на одноимённые методы формируются *в процессе выполнения программы*, т.е. при так называемом ***позднем связывании***.

Для *каждого объекта*, содержащего виртуальные методы, компилятор создаёт *таблицу виртуальных методов (ТВМ),* количество её элементов равно количеству виртуальных методов объекта. В этой таблице хранятся размеры виртуальных методов и адреса точек входа в каждый виртуальный метод

- Описание виртуальных методов
  Чтобы описать метод как виртуальный необходимо:
  - при описании метода во всех классах в конце заголовка метода добавить слово *virtual;*

Procedure <имя> (<список формальных параметров>); *virtual;*
Function <имя> (<список формальных параметров>): <тип функции>; *virtual;*

  - в каждый класс иерархии, содержащий виртуальные методы, включить специальный статический метод – конструктор. Для этого метода слово *Procedure* должно быть заменено словом ***Conctructor.*** Конструктор неявно выполняет настройку механизма позднего связывания (обеспечивает связь объекта с ТВМ).

Внимание! Метод *Conctructor* должен быть вызван до первого обращения к виртуальному методу, иначе происходит «зависание» компьютера!

Как правило, *Conctructor* используют для инициализации полей объекта и, следовательно, роль конструктора должен выполнять метод *Init*.

  - если метод, который является виртуальным в потомках, отсутствует в базовом классе, то его надо описать и в базовом классе, может быть, как метод с пустым телом.

Внимание!
Одноимённые виртуальные методы должны иметь одинаковый список формальных параметров, для функций – тип результата, или быть методами без параметров!

- Создание и использование процедуры с полиморфным объектом.

  Структура заголовка процедуры:

Procedure <имя> (*Var* <имя переменной>:<тип базового класса>);

При использовании процедуры достаточно в операторе вызова её указать переменную требуемого класса и вызов его полиморфного метода будет обеспечен.

- Сокрытие полей и методов

Поля и методы, объявленные за зарезервированным словом *privat*, становятся недоступными в программе. Их доступность ограничивается только рамками модуля, в котором они описаны.

Поля и методы, объявленные за словом *public*, становятся общедоступными.

Контроль входных знаний

1. Как понимаются в ООП свойства: инкапсуляция, наследование и полиморфизм?
2. Чем различается раннее и позднее связывание методов с кодом программы?
3. Какую роль выполняет *Constructor*?
4. Какую информацию содержит таблица виртуальных методов?
5. Вне классов описана процедура

Procedure DoSeek(var F:T_obj); {*T_obj* – базовый класс в иерархии классов}

Возможно ли использование в ней при вызове в качестве фактического параметра экземпляра наследуемого класса?

6. Когда возникает необходимость сокрытия полей и методов?

Задания для выполнения

Вызов полиморфных методов из методов базового класса выполняется на основе программ, разработанных в двух предыдущих лабораторных работах.

1. Демонстрация механизма простого (статического) полиморфизма:
   - назовите методы '*Вывод символа*' и '*N-кратный вывод символа*' одинаковым именем;
   - создайте одноимённый метод в базовом классе, включив в его тело вывод фразы '*Я – родитель*';
   - создайте в базовом классе новый метод *Run* – процедуру без параметров, содержащую обращение к методу построения окна и методу, выводящему фразу '*Я – родитель*';
2. Демонстрация механизма сложного полиморфизма – разработка виртуальных методов:
   2.1. Вызов виртуальных методов из методов базового класса:
      - объявите одноимённые методы виртуальными;
      - создайте конструкторы для каждого класса;
      - разработайте программу, содержащую конструирование объектов классов '*Окно*', '*Вывод символа*', '*N-кратный вывод символа*';
      - организуйте для каждого объекта вызов метода *Run* базового класса;
      - выполните программу;
      - оцените полученные результаты.
   2.2. Вызов виртуальных методов с использованием процедуры, содержащей полиморфный объект:
      - в интерфейсной части модуля вместо метода *Run* предыдущего задания, опишите процедуру, выполняющую те же операции, что и метод *Run*;

- в основной программе организуйте её вызов, обеспечивающий правильную работу виртуальных методов;
- вынесите процедуру *Run* из модуля в основную программу; продемонстрируйте её работу;

2.3. Оцените способы использования виртуальных методов.

3. Сокрытие полей и методов:

- организуйте варианты работы Ваших программ со скрытыми полями и методами.

Лабораторная работа

ОБРАБОТКА СИМВОЛЬНОЙ ИНФОРМАЦИИ

Основы теории

Часто при разработке программ возникает необходимость в обработке символьной информации. Такая информация может быть представлена в виде одного символа – ***литеры*** или в виде последовательности символов – ***строки*.**

Для объявления данных литерного типа в Паскале используется служебное слово *Char*, а для объявления данных строкового типа используется служебное слово *String*, за которым в квадратных скобках указывается значение максимально допустимой длины данной строки в пределах от 1 до 255 включительно. Если после слова *String* ничего не указано, то длина строки автоматически устанавливается равной 255.

В памяти компьютера под каждый символ отводится по одному байту, при этом нулевой байт всегда содержит информацию о текущей длине данной строки.

В выражениях строки и символы заключаются в апострофы. Например,

‘*a*’, ‘*A*’, ‘*сигнал*’, ‘*Иванов Р.Д.*’

Согласно кодировочной таблице ASCII строчные и заглавные буквы считаются различными.

Элементы строки следует рассматривать как индексные переменные, с изменением индекса в максимально допустимых пределах от 1 до 255.

- Объявление и инициализация данных

Тип ***Char*** может быть связан с константами и переменными.

Константы объявляются в виде:

Const <её имя> = ‘<символ>’;

Например,

Const ch1 = ‘+’; Const ch2 = ‘k’;
Const ch3 = ‘ ’;{эта константа соответствует пробелу}.

Переменные могут быть объявлены через ссылку на ранее объявленный тип, например,

```
Type Tchar = char;
Var ch1: Tchar;
```

или непосредственно, например,

```
Var ch1: char;
```

Задать конкретное значение переменной типа *Char* можно с использованием операторов присваивания или ввода.

Тип ***String*** аналогично типу *Char* также может быть связан с константами и переменными.

Константы объявляются в виде:

Const <её имя> = ‘<строка символов>’;

Например,

Const Str_L = ‘Обработка символьных данных’;

Объявление *переменных* можно выполнить через ранее объявленный тип и непосредственно.

- ✓ Объявления переменных через ранее объявленный тип

```
Const CStr = 12; {вспомогательная константа для объявления типа и переменной}
Type
  TStr1 = string[CStr]; {тип объёмом памяти в 12 байт}
  TStr2 = string[150];  {тип объёмом памяти в 150 байт}
  TStr3 = string;       {тип объёмом памяти в 256 байт}
Var
  Str1: TStr1; {строка, которая не должна содержать более 12 символов}
  Str2: TStr2; {строка, которая не должна содержать более 150 символов}
  Str3: TStr3; {строка, которая не должна содержать более 255 символов, включая пробелы}
```

- ✓ Непосредственное объявление переменных

```
Var
  Str1: string[CStr];
  Str2: string[150];
  Str3: string;
```

- ➢ Строковые выражения

Данные типа *Char* и *String* будем в дальнейшем называть ***строковыми***. Выражения, операндами которых служат строковые данные, называют ***строковыми выражениями***. Строковые выражения формируются из строковых констант, переменных, указателей функций и знаков операций ***сцепления и сравнения***.

Операция сцепления (+) применяется для сцепления (слияния) нескольких строк в одну строку. Длина результирующей строки не должна превышать 255. Например, слово *PasCal* можно получить как результат выполнения оператора:

Str:= ‘Pas’ + ‘Cal’;

или

Str:= ‘Pa’ + ‘s’ + ‘C’ + ‘al’;

а также как результат выполнения последовательности операторов:

Str1:= ‘Pas’; Str2:= ‘Cal’; Str:= Str1+Str2;

или

Str1:= ‘Pa’; Srt2:= ‘al’; Str:= Str1 + ‘s’ + ‘C’ + Str2;

Операции отношения записываются традиционным способом:

<, <=, >, >=, =, < >

- Операции над символами

Символы можно лишь присваивать и сравнивать друг с другом. При сравнении символов они считаются равными, если равны их ASCII-коды, и один символ больше другого, если имеет больший ASCII-код.

Например,

‘*R*’ = ‘*R*’
‘*r*’ > ‘*R*’ {код #114 > кода #82}

К символьным значениям и переменным могут быть применены также системные функции:

Chr (X: byte): char – возвращает символ ASCII-кода.
Ord (C: char): byte – возвращает ASCII-код символа *C*.

Pred (C: char): char – возвращает предшествующий *C* символ.
Succ (C: char): char – возвращает последующий за *C* символ.
UpCase (C: char): char – переводит символы '*a*'..'*z*' в верхний регистр '*A*'..'*Z*', возвращая все остальные, в том числе и кириллицу, в исходном виде.

- Операции над строками

Строки можно присваивать, сцеплять и сравнивать. Если при сцеплении длина строки получится длиннее, чем объявленная длина строки для переменной в левой части оператора присваивания, то излишек отсекается.

Сравнение строк происходит посимвольно, начиная от первого символа в строке. Строки равны, если имеют одинаковую длину и посимвольно эквивалентны. Если при посимвольном сравнении окажется, что один символ больше другого (его код больше), то строка, содержащая его, тоже считается большей. Остатки строк и их длины не играют роли. Любой символ всегда больше «пустого места».

К отдельным символам строки можно обратиться по номеру (индексу) данного символа в строке. Индекс определяется выражением целого типа аналогично записи индекса в элементе одномерного массива.

- Системные процедуры и функции обработки строк

Процедура *Delete (Var S: string; Pos, N: integer)* – удаляет из строки *S* подстроку, содержащую *N* символов, начиная с позиции *Pos*.

Например,

```
S:= 'ABCL+FGR';
Delete (S, 3, 4);
```

После выполнения этих операторов строка *S* = '*ABGR*'.

Процедура *Insert (Fragment: string; Var S: string; Pos: integer)* – вставляет в строку *S* подстроку *Fragment*, начиная с позиции *Pos*.

Например,

```
Fragment:= 'ABCD';
S:= 'abefg';
Insert (Fragment, S, 3);
```

Результатом выполнения этой процедуры будет строка *S* = '*abABCDefg*'.

Процедура *Str (X; Var S: string)* – преобразует числовое значение *X* в строковое и присваивает результат строке *S*, причём можно переводить числа как целые, так и вещественные.

Например:

```
X:= 2321;
Y:= 76.854;
Str (X, S1);
Str (Y:8:3, S2);    {без форматного преобразования результат будет выдан в форме с плавающей точкой}
```

В результате выполнения этих операторов *S1*= '*2321*', *S2*= '*76.854*'.

Процедура *Val (S: string; Var X; Var ErrCode: integer)* – переводит строковое значение *S* в значение числовой переменной *X* целого или вещественного типа, если данная строка действительно отвечает правилам записи чисел. Строка *S* не должна содержать незначащих пробелов в начале и в конце. Значение переменной *ErrCod* равно нулю, если при преобразовании не обнаружено ошибок, в противном случае значение *ErrCod* равно номеру позиции первого ошибочного символа.

Например,

```
S1:= '5621';
S2:= '-89.543';
S3:= '246.56 + 0.45';
Val (S1, X, Cod1);
Val (S2, Y, Cod2);
Val (S3, Z, Cod3);
```

Результатом выполнения этой последовательности операторов будут значения:

X = 5621, Cod1 = 0; Y = -89.543, Cod2 = 0; Cod3 = 4;

В последнем операторе выдан только код ошибки, так как разделительным знаком между целой и дробной частями является точка, а не запятая.

Функция *Pos (Fragment, S: string): byte* – возвращает номер первого элемента, с которого начинается первое вхождение подстроки *Fragment* в строку *S*. Если такой подстроки нет, то результат равен нулю.

Например:

```
Frag_1 = 'CD';
Frag_2:= 'DA';
S:= 'ABCDabcCD';
k1:= Pos (Frag_1, S);
k2:= Pos (Frag_2, S);
```

Переменная *k1* равна 3, так как первое появление подстрока '*CD*' начинается с третьей позиции строки *S*. Значение переменной *k2* равно 0, так как такой подстроки нет.

Функция *Copy (S: string; Pos, N: integer): string* – копирует *N* символов строки *S*, начиная с позиции *Pos*. Если *Pos* больше длины строки *S*, то результатом будет пробел, при *Pos* большем 255 выдаётся ошибка.

Например,

```
S1:= 'Весна идёт, весне дорогу!';
Frag_1:= Copy (S1, 13, 13);
Frag_2:= Copy (S, 9, 3);
```

Значением переменной *Frag_1* будет строка '*весне дорогу!*', а переменной *Frag_2* будет присвоено значение '*ёт,*'.

Функция *Concat (S1, S2, ..., Sn): string* – выполняет сцепление (слияние) строк *S1, S2, ..., Sn* аналогично *операции сцепления (+).*

Функция *Length (S: string): byte* – выдаёт текущую длину строки.

Например,

```
Var S1: string;
    S2: string[5];
        ...
    S1:= 'a+bsin (x)';
    S2:= 'aa dd7 cc bbb';
    L1:= Length (S1);
    L2:= Length (S2);
```

В результате получим *L1 = 9*, а *L2 = 5*, так как по объявлению количество символов в строке *S2* не должно быть больше *5*.

Процедура *Insert (Fragment: string; Var S: string; Pos: integer)* – вставляет в строку *S* подстроку *Fragment*, начиная с позиции *Pos*.

Например,

```
S1:= 'У Егорки';
S2:= ' отговорки!';        {строка начинается с пробела}
Insert (S1, S2, 1);
Insert (' всегда', S2, 9); {строка ' всегда' тоже начинается с
                            пробела}
```

В результате выполнения последовательности этих операторов будет сформирована строка *S2 = 'У Егорки всегда отговорки!'*.

Контроль входных знаний

1. Чем отличается тип *Char* от типа *String*?
2. Сколько байт памяти отводится под переменную *Sim*: *char*?
3. Сколько байт памяти потребуется для размещения переменной *Str*: *string[24]*?
4. Что общего и чем отличается переменная типа *string* от одномерного массива?
5. Как сравниваются строки?

6. Как на основе процедуры
   Val (*S*: *string*; *Var X*; *Var ErrCode*: *integer*)
   реализовать алгоритм поиска числовой подстроки в строке?
7. Каким свойством кода-ASCII можно воспользоваться при реализации алгоритма сортировки элементов строки в алфавитном порядке?

Задания для выполнения

В технологии модульного программирования выполнить задание согласно Вашего варианта.

Таблица 4

Варианты заданий

№ варианта	Задание
1	Дана непустая последовательность непустых слов из латинских букв; соседние слова отделены друг от друга запятой, за последним словом – точка. Определить количество слов, которые заканчиваются буквой ‘*w*’.
2	Дана непустая последовательность непустых слов из латинских букв; соседние слова отделены друг от друга запятой, за последним словом – точка. Определить количество слов, которые начинаются и оканчиваются одной и той же буквой.
3	Дана непустая последовательность непустых слов из латинских букв; соседние слова отделены друг от друга запятой, за последним словом – точка. Определить количество слов, которые содержат хотя бы одну букву ‘*d*’.
4	Дана непустая последовательность непустых слов из латинских букв; соседние слова отделены друг от друга запятой, за последним словом – точка. Определить количество слов, которые содержат ровно три буквы ‘*e*’.
5	Дана непустая последовательность непустых слов из латинских букв; соседние слова отделены друг от друга запятой, за последним словом – точка. Вывести все слова, отличные от слова ‘*hello*’.

Продолжение табл. 4

№ варианта	Задание
6	Дана непустая последовательность непустых слов из латинских букв; соседние слова отделены друг от друга запятой, за последним словом – точка. Вывести текст, составленный из последних символов всех слов текста.
7	Дана непустая последовательность непустых слов из латинских букв; соседние слова отделены друг от друга запятой, за последним словом – точка. Вывести текст, составленный из первых символов всех слов текста.
8	Дана непустая последовательность непустых слов из латинских букв; соседние слова отделены друг от друга запятой, за последним словом – точка. Вывести все слова, содержащие ровно две буквы ‘*d*’.
9	Дан текст, содержащий от 1 до 30 слов, в каждом из которых от 1 до 5 малых латинских букв, между словами запятая, за последним словом – точка. Вывести на экран эту же последовательность слов, но в обратном порядке.
10	Дан текст, содержащий цифры, латинские и русские буквы. Подсчитать сумму цифр, встречающихся в тексте.
11	По правилам машинописи после запятой в тексте всегда ставится пробел. Составить программу исправления такого рода ошибок в тексте.
12	Составить программу исправления ошибочного набора текста вида *«после символов ‘.’, ‘!’, ‘?’ должен стоять пробел»*.
13	Удвоить вхождение некоторой буквы в текст. Буква задается пользователем.
14	Дан текст. Вывести все слова, предварительно заменив в них первую букву на заглавную.
15	Дан текст. Составить программу проверки правильности написания сочетаний «жи»-«ши», «ча»-«ща», «чу»-«щу». Исправить ошибки.

Окончание табл. 4

№ варианта	Задание
16	Дан текст, содержащий от 1 до 30 слов, разделенных запятой, заканчивающийся точкой. Дописать после каждого слова количество вхождений в него заданного символа.
17	Дан текст, содержащий цифры, латинские и русские буквы. Найти максимальное число среди чисел, образованных входящими в текст цифрами.
18	Даны две строки. Составить третью, включив в нее только те символы, которые есть и в первой и во второй строке.
19	Дана строка, состоящая из *n* символов. Вывести ее на экран *n* раз, циклически сдвигая на 1 символ вправо. Пример: исходная строка – *sdfhjoutwer*, сдвиг на 1 символ вправо – *dfhjoutwers*.
20	Дан текст, содержащий от 1 до 30 слов, разделенных запятой, заканчивающийся точкой. Вывести на экран текст, центрируя каждое слово по середине экрана. Примечание: использовать процедуры модуля *CRT* не допускается.
21	Даны две строки. Определить, совпадают ли они. Если нет, сообщить номер позиции первого несовпадающего символа.
22	Дан текст, содержащий цифры, латинские и русские буквы. Найти сумму всех цифр, присутствующих в тексте.
23	Дана строка, содержащая минимум две буквы ‘z’. Изменить ее следующим образом: символы строки, расположенные между первой и последней буквой ‘z’, переставить в обратном порядке.
24	Дан текст, содержащий слова, разделенные пробелами. Найти в нем слова-рифмы для заданного слова (рифма – совпадение трех последних символов).
25	Дан текст, содержащий слова, разделенные пробелами. Найти в нем и вывести на экран слова-палиндромы (одинаково читающиеся слева направо и наоборот).

Лабораторная работа

ОБРАБОТКА ДАННЫХ ТИПА ЗАПИСИ

Основы теории

Запись – это структура данных, состоящая из фиксированного числа *разнотипных* компонент, называемых *полями записи*. Записи используются для представления разнородной, но логически связанной информации. Каждое поле записи имеет имя, которое даётся ему при объявлении записи.

Тип записи объявляется структурой

record
<поле 1> : <тип поля 1>;
<поле 2> : <тип поля 2>;
…
<поле n> : <тип поля n>;
end;

Тип полей записи может быть любым, в том числе и типом ранее объявленной записи, за исключением файлового типа. В Borland Pascal определены записи двух типов: записи с фиксированными полями и вариантные записи. В данной лабораторной работе рассматриваются только *записи с фиксированными полями.*

Данные типа записи можно использовать как простые переменные, так и формировать из них массивы. Объявление данных этого типа ничем не отличается от стандартных для языка способов.

Объявление простых переменных можно выполнить непосредственно при объявлении переменных или через ранее объявленный тип записи.

- ✓ Непосредственное объявление переменных:

Var <имя переменной> : *record*
<поле 1> : <тип поля 1>;
<поле 2> : <тип поля 2>;
…
<поле n> : <тип поля n>;
end;

- ✓ Объявление переменных через ранее объявленный тип записи:

Type <имя типа> = *record*
<поле 1> : <тип поля 1>;
<поле 2> : <тип поля 2>;
…
<поле n> : <тип поля n>;
end;

Var <имя переменной> : <имя типа>*;*

Объявление массива записей можно также выполнить непосредственно при объявлении переменных или через ранее объявленный тип массива записей.

- ✓ Непосредственное объявление переменных:

Var <имя массива> : *array [..] of record*
<поле 1> : <тип поля 1>;
<поле 2> : <тип поля 2>;
…
<поле n> : <тип поля n>;
end;

- ✓ Объявление переменных через ранее объявленный тип массива записей:

Type <имя типа массива> = *array [..] of* <тип записи>:

Var <имя массива> : <имя типа массива записей>*;*

В дальнейшем при изложении материала будем объявлять переменные при формировании списков параметров подпрограмм только через предварительно объявленные типы, как того требуют процедурная и модульная технологии программирования.

Пример 1. Объявление двух записей *Rec1* и *Rec2*, состоящих из 6 полей:

```
Type T_Rec = record
                FAM: string[12]; {одно поле строкового типа}
                A, B: real;      {два поля вещественного
                                  типа}
                N, K: integer;   {два поля целого типа}
                Ch1: char;       {поле символьного типа}
             end;

Var Rec1, Rec2: T_rec;
```

Пример 2. Объявление записи, содержащей *поле-запись*:

```
Type TAddress: record
                  Name_Np,              {населённый пункт}
                  Street,               {улица}
                  House_Fl:string[25];  {дом-квартира}
               end;

T_Stud = record
            FIO: string[30];      {фамилия, имя, отчество}
            Address: TAddress;    {адрес}
            Grupp: string[6];     {группа}
            NumTel: string[11];   {номер телефона}
         end;

{Объявление переменной типа T_Stud:}

Var Rec_Stud: T_Stud;
```

Пример 3. Объявление массива, содержащего 10 записей типа *T_Stud*:

```
Type TM1_Stud = array [1..10] of  T_Stud;

Var M1_Stud: TM1_Stud;
```

- Операции над записями

  Над записями можно выполнять следующие операции:

  - доступ к полям записи;
  - присваивание записей.

Доступ к полям записи осуществляется через полное *составное имя (путь к полю.поле)*. Элементы имени разделяются точкой.

К полям простой переменной *Rec_Stud* типа *T_Stud*, объявленной выше, можно обратиться непосредственно:

```
Rec_Stud.Grupp:= 'AM-72';
Rec_Stud.FIO:= 'Иванов Дмитрий Иванович';
Rec_Stud.Address.House_Fl:= 10 – 54;
```

или используя оператор присоединения:

{Вариант 1}

```
With Rec_Stud do
        begin
            Grupp:= 'AM-72';
            FIO:= 'Иванов Дмитрий Иванович';
            Address.House_Fl:= 10 – 54;
        end;
```

{Вариант 2}

```
With Rec_Stud do
        begin
            Grupp:= 'AM-72';
            writeln (FIO);
            With Address do
                        begin
                            writeln ('Адрес:');
                            readln (Name_Np);
                            readln (Street);
                            readln (House_Fl:);
                        end;
        end;
```

Доступ к *полям массива записей* осуществляется также через составное имя, первая составляющая которого является *i-ым элементом массива* записей:

```
With M1_Stud[i].Rec_Stud do
        begin
            Grupp:= 'AM-72';
            writeln (FIO);
            With Address do
                        begin
                            writeln ('Адрес:');
                            readln (Name_Np);
                            readln (Street);
                            readln (House_Fl:);
                        end;
        end;
```

Операция присваивания применима только к однотипным записям.

Контроль входных знаний

1. Чем отличается запись от одномерного массива?
2. Сколько байт памяти будет занимать поле '*Address*' переменной *M1_Stud[i]*?
3. Запишите варианты доступа к полю '*дом-квартира*' переменной *M1_Stud[i]*?
4. Переменные типа записи *M1_Rec* и *M2_Rec* объявлены как:

   Var M1_Rec, M2_Rec:TM1_Stud;

   Какие операции недопустимы и почему?
   - *if M1_Rec <= M2_Rec then … else …;*
   - *M2_Rec:= M1_Rec;*
   - *writeln (M1_Rec, ' ', M2_Rec);*
   - *write ('Введите данные студента ', i);*
     readln (M1_Rec[i]);

- *write (‘Введите фамилию, имя и отчество студента ’, i);*
  readln (M1_Rec[i].FIO);

5. Обоснуйте корректность выполнения оператора

Name_Np:= Street ;

Задания для выполнения

1. Разработать структуру базы данных по вариантам (см. табл. 5), включив в нее не более пяти полей данных разных типов.
2. В программе использовать оконный интерфейс – меню, обеспечивающее формирование базы данных, вывод базы на экран и завершение работы программы.
3. Заполнить базу данных не менее чем пятью записями.
4. Вывести данные на экран.

Таблица 5

Варианты заданий

№ варианта	Наименование базы данных
1	Студент – экзаменационная сессия
2	Библиотека – заказ книг
3	Автомобиль – ГИБДД
4	Студент – факультет
5	Студент – аттестация на лучшую группу
6	Отдел кадров – учет военнообязанных
7	Телефонная станция
8	Библиотека – учет выданной литературы
9	Учет товаров на складе
10	Отдел кадров – учет уволенных сотрудников по уважительной причине

Продолжение табл. 5

11	Отдел кадров – учет уволенных сотрудников по нарушению трудовой дисциплины
12	Прайс-лист магазина по продаже компьютеров
13	Фильмотека
14	Поликлиника
15	Аптечный склад
16	Компьютерный магазин – учет новых поступлений
17	Подписка на журналы
18	Абитуриент – учетные данные
19	База снабжения
20	Спортивные соревнования
21	Библиотека – учет новых поступлений
22	Компьютерный магазин – учет продаж
23	Абитуриент – результаты вступительных экзаменов
24	Бухгалтерия – начисление заработной платы по окладу
25	Бухгалтерия – начисление заработной платы с почасовой оплатой труда

Лабораторная работа

ФОРМИРОВАНИЕ И ОБРАБОТКА ПЕРЕМЕННЫХ ФАЙЛОВОГО ТИПА. ТИПИЗИРОВАННЫЕ ФАЙЛЫ

Основы теории

Довольно часто обработке подлежат большие объемы данных, как, например, сведения обо всех студентах в деканате, ведомости заработной платы, адреса ВУЗов города и т.п. Вводить многократно такие объемы с клавиатуры слишком долго и утомительно. Желательно хранить их на внешнем носителе и обращаться к ним из обрабатывающей программы при каждом счете.

Для этих целей в Турбо Паскале предусмотрены специальные объекты – *файлы.* Операции над ними сводятся к работе с внешними носителями. Файловая система Турбо Паскаля связана с двумя понятиями файла: файла физического и файла логического.

Под физическим файлом понимается поименованная область на внешнем носителе, хранящая последовательность однотипных компонентов.

Каждому физическому файлу в языке ставится в соответствие *файловая переменная определённого типа*, называемая ***логическим файлом***.

Количество компонентов файла заранее не оговаривается. Компонентами файла могут быть переменные любого типа, за исключением файлового типа.

➢ Объявление типа файла

Type < имя типа> = file of <тип компонент файла>;

Тип конкретных файлов объявляется следующим образом:

Пример 1. Объявление типа файла с компонентами записями:

```
Type TStud = record          {TStud – тип записи с полями:}
                Fam: string[15];   {фамилия,}
                Nom: integer;      {номер телефона.}
             end;
TfStud = file of Stud;       {TfStud – тип файла с компонентами – записями}
```

Пример 2. Объявление файла с компонентами вещественного типа:

```
Tf_Rr = file of real;        {Tf_Rr – тип файла с вещественными компонентами}
```

Пример 3. Объявление файла с компонентами целого типа:

```
Tf_Ri = file of integer;     {Tf_Ri – тип файла с компонентами целого типа}
```

➢ Объявление переменных файлового типа

Объявить переменную файлового типа можно непосредственно или через ранее определенный тип.

✓ Непосредственное объявление:

Var <список переменных > : < тип файла>;

Например,

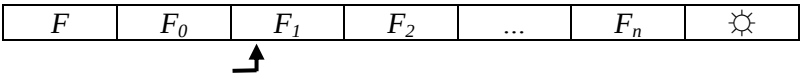
```
Var f_Rec: file of record      {f_Rec – файловая переменная типа
                                «запись» с полями:}
                  Fam: string[15]; {фамилия,}
                  Nom: integer;    {номер телефона.}
               end;
    f_Rr: file of real;        {f_Rr – файловая переменная вещественного типа }
    f_Ri: file of integer;     {f_Ri – файловая переменная целого типа}
```

✓ Объявление переменных через ранее объявленный тип:

Например,

```
Var f_Rec: file of TStud;      {f_Rec – переменная файлового типа TStud}
    f_Rr: Tf_Rr;               {f_Rr – переменная файлового типа Tf_Rr}
```

Условно физический файл можно изобразить как некоторую ленту:



где *F* – имя файла;

F0, F1, … , Fn – компоненты файла;

⮭ – указатель файла;

☼– признак конца файла.

Имя файла – <собственное имя-идентификатор>**.**<расширение>. В качестве расширения рекомендуем использовать *dan* или *dat*.

Компонент файла – это конкретное данное, хранящееся в файле. Нумерация компонентов файла начинается с нуля.

Указатель файла – позиция, которая доступна в данный момент времени для чтения или записи. Все действия с файлом производятся покомпонентно, причём в этих действиях участвует тот компонент файла, который обозначается текущим указателем. В результате выполнения операций текущий указатель может перемещаться, настраиваясь на тот или иной компонент файла.

Признак конца файла – системный код (символ #26), следующий за последним компонентом файла.

По способу доступа к компонентам различают файлы двух видов: файлы последовательного доступа и файлы прямого доступа. ***Файлом последовательного доступа*** называется файл, к компонентам которого обеспечивается доступ в той же последовательности, в какой они записывались (позиции 0, 1, 2, …). ***Файлом прямого доступа*** называется файл, доступ к компонентам которого осуществляется па адресу компонента – по номеру позиции.

Для работы с файловыми данными необходимо выполнить следующие действия:

1. Логически связать программное имя файла – переменную файлового типа с именем соответствующего физического файла.
2. Проверить файл на наличие в нём данных.
3. Открыть файл на запись или на чтение.
4. Выполнить действия, предусмотренные алгоритмом решения задачи.
5. Закрыть файл.

➢ Стандартные *процедуры и функции*

Процедура *Assign (var f; FileName: string)* – связывает файловую переменную *f* с именем физического файла, заданным в строке *FileName*. Имя физического файла – путь к файлу (пол-

ное имя или имя файла и его расширение, если файл находится в текущем каталоге), например:

```
Type TF_r = file of real;
Var f1, f2, f3: Tf_r;
        Fs: string[30];          {Fs – имя физического файла,
                                  определённое через оператор ввода
                                  или через оператор присваивания}
            …
BEGIN
            …
        Assign (f1, 'Fm.dat'); {связывание файловой переменной
                                с файлом в текущем каталоге}
        Assign (f2, 'C:\AM_1\ Fr.dan'); {связывание файловой
                                переменной с файлом в указанном
                                каталоге}
        Assign (f3, Fs);         {связывание файловой переменной
                                с файлом под именем, являющимся
                                значением переменной Fs}
            …
END.
```

Процедура *Rewrite (var f)* – создает (открывает) *новый файл* с логическим именем *f* для записи, если файл был непустым, то все его компоненты пропадают, признак конца файла перемещается в самое начало файла.

Процедура *Reset (var f)* – открывает файл с логическим именем *f* для чтения, устанавливает указатель в начало файла, то есть перемещает его на первую компоненту.

Процедура *Read (var f; t)* – считывает из файла. После выполнения процедуры переменной *t* присваивается значение, равное компоненту файла, на котором стоял указатель файла. Переменная *t* может быть простой или индексной, совпадающей по типу с типом компонентов файла.

Процедура *Write (var f; t)* – записывает в файл. После выполнения процедуры в файл *f* будет записан ещё один компонент, равный значению переменной *t.* Переменная *t* может быть простой или индексной, совпадающей по типу с типом компонентов файла.

Процедура *Seek (var f; n: longint)* – устанавливает указатель текущей позиции файла на компонент с номером *n*, начиная отсчёт с нуля.

Процедура *Close (var f)* – закрывает файл с именем *f*. Если файл был открыт, никогда не следует выходить из программы, предварительно не закрыв его. Связь файловой переменной с файлом при закрытии сохраняется, и при повторном использовании этого же файла в данной программе процедуру *Assign* применять ещё раз не требуется.

Процедура *Erase (var f)* – удаляет внешний файл, логически связанный с переменной *f* (применима лишь к закрытым файлам).

Процедура *Rename (var f; New_name: string)* – переименовывает физический файл, ранее связанный с файловой переменной *f*, в имя *New_name* (применима лишь к закрытым файлам).

Процедура *Truncate (var f)*– отсекает часть файла *f*, начиная с того компонента, который был бы считан последующей операцией считывания, и подтягивает на его место конец файла (применима лишь к открытым файлам).

Функция *Eof (f): Boolean* – определяет состояние конца файла. Значение функции равно *True,* если указатель файла находится на признаке его конца (за последней компонентой файла), и *False* в любом другом случае.

Функция *FilePos (f): longint* – определяет номер компонента, на который установлен в данный момент указатель файла.

Функция *FileSize (f): longint* – определяет значение, равное количеству компонентов в файле *f*. Если *FileSize (f) = 0*, то файл пуст, иначе файл содержит данные.

➢ Обработка компонентов файла

Основные операции над компонентами файла – это операции записи и чтения. На базе этих операций выполняются более сложные операции:

- создание файла – запись в файл требуемых данных;
- модификация файла – изменение всех или нескольких компонентов, добавление и удаление компонентов;
- поиск нужной информации.

Демонстрационные примеры

В примерах рассмотрены программные фрагменты, демонстрирующие процессы: формирования, чтения, корректировки и расширения файла данных вещественного типа.

Перед любым действием с файлом необходимо провести проверку на наличие в нём данных:

```
        …
Assign (f, Name_File); {физическому файлу, являющемуся значением переменной Name_File, назначили файловую переменную f}
Reset (f);             {открыли файл на считывание}
if Filesize (f) = 0 then writeln (‘Файл пуст’)
                   else begin … end; {реализовали проверку файла на наличие в нём данных}
        …
```

Пример 1. Формирование файла.

```
        …
Rewrite (f);      {открыли файл, чтобы записать в него данные}
          repeat
                write (‘число = ’);
                readln (a);
                write (f, a); {запись данного в файл}
                write (‘Продолжить? y/n’); readln (p);
          until p = ‘n’;
        …
Close (f);
        …
```

Пример 2. Чтение данных из файла.

В этом примере данные из файла считываются исключительно с целью их проверки.

- ✓ Фрагмент *последовательного доступа* к данным файла:

```
…
Reset (f);          {открыли файл для чтения}
while not Eof (f) do
                    begin
                        read (f, a); {считали в переменную a
                                      значение очередного компонента файла}
                        writeln (‘число = ’, a:8:2);
                    end;
…
Close (f); {закрыли файл}
…
```

- ✓ Фрагмент *прямого доступа* к данным файла:

```
…
Reset (f);          {открыли файл для чтения}
write (‘Введите номер позиции считываемого данного = ’);
readln (k);
    seek (f, k); {установили указатель файла на позицию с
                  номером k}
    read (f, a); {считали значение k-го компонента в переменную a}
    writeln (‘число = ’, a:8:2);
…
Close (f); {закрыли файл}
…
```

Пример 3. Корректировка файла.

Корректировка файла осуществляется только в том случае, *если файл не пуст*.

Процесс корректировки файла *последовательного доступа* требует:

- считать все данные из файла;
- откорректировать данные;
- записать все данные в файл.

Процесс корректировки файла *прямого доступа* требует:

- определить позицию корректируемого компонента;
- установить указатель файла в эту позицию и считать данное;
- сформировать новое значение данного и записать в файл.

Ниже приведен фрагмент реализации корректировки файла прямого доступа:

```
    …
write ('Введите номер позиции исправляемого данного = ');
readln (k);
        Reset (f);    {открыли файл для чтения}
        seek (f, k);  {подвели указатель файла к позиции k}
        read (f, a); {считали k-ый компонент в переменную a}
        writeln ('число = ', a:8:2);
        Close (f);   {закрыли файл}
write ('Введите новое значение = ');
readln (a);
        Rewrite (f); {открыли файл, чтобы записать в него данные}
        write (f, a); {записали новое значение переменной a в файл}
        Close (f);    {закрыли файл}
    …
```

Пример 4. Расширение файла за счёт внесения в него новых компонентов.

Дописывать новые компоненты можно только в конец файла. При реализации этого процесса необходимо:

- ввести значение нового компонента;
- установить указатель файла за последним компонентом (оператор *Seek (f, filesize (f));*);
- записать откорректированный компонент;
- закрыть файл.

```
…
write ('Введите новое значение = ');
readln (a);
    Seek (f, filesize (f)); {установили указатель файла за последним компонентом}
    write (f, a); {записали новое значение переменной a в файл}
…
```

Контроль входных знаний

1. Что общего и в чём различие физического и логического файлов?
2. Можно ли в программе одной и той же файловой переменной поставить в соответствие несколько физических файлов?
3. Конец файла устанавливается программистом или самой системой?
4. Можно ли поместить новый компонент в любое место файла?
5. При формировании файла имеет ли значение, какой доступ (последовательный или прямой) к его компонентам будет реализован в программе?
6. Как выполнить корректировку компонента файла, если его место в файле неизвестно?
7. При формировании файла можно использовать счётный цикл (записывать заданное количество компонентов) или другие циклы, обеспечивающие повторение записи, например, по запросу '*Ввод продолжить? Да – y, Нет – n*'. Какой структуре следует отдать предпочтение при изменяющемся объёме информации, подлежащей хранению в файле?

Задания для выполнения

Таблица 6

Варианты заданий

№ варианта	Задание
1	В файле из вещественных чисел заменить значение *k*-го компонента средним значением суммы положительных компонентов файла.
2	В файл из целых чисел добавить значение суммы первых *k* компонентов.
3	Компонент файла состоит из двух переменных: первая переменная – строка, вторая – число. Переместить в конец файла данное, для которого первая составляющая имеет наибольшую длину.
4	В файле из целых чисел определить, каких компонентов (положительных или отрицательных) в нём больше.
5	В файле из строковых данных определить наличие заданной строки и поместить ответ в качестве его первого компонента.
6	Из файла строковых данных удалить первый компонент, содержащий заданную строку.
7	Компонент файла состоит из двух переменных: первая переменная – строка, вторая – целое число. Удалить из файла первое данное, для которого вторая составляющая имеет нулевое значение.
8	Из файла символьных данных удалить последний компонент, содержащий заданный символ.
9	В файле из двоичных данных определить количество единиц и, если возможно, добавить полученное значение в файл.
10	Из файла вещественных чисел удалить все компоненты, находящиеся после первого чётного компонента.
11	Компонент файла состоит из двух переменных: первая переменная – строка, вторая – число. Удалить из файла данное, для которого вторая составляющая имеет наибольшее значение в своей группе.

Продолжение табл. 6

№ варианта	Задание
12	В файл из целых чисел добавить сумму значений первого и последнего его компонентов и заменить первый компонент на их разность.
13	Из файла двоичных чисел удалить все компоненты, находящиеся до первой единицы.
14	Компонент файла состоит из двух строковых переменных. Удалить из файла данное, для которого значение первой составляющей равно заданной строке.
15	Сформировать новый файл из целых положительных чисел исходного.
16	Расставить компоненты числового файла в порядке возрастания.
17	Компонент файла состоит из двух переменных: первая переменная – число, вторая – символ. Переместить в конец файла данное, для которого первая составляющая имеет наименьшее значение в своей группе.
18	Заменить первый чётный компонент числового файла номером его позиции.
19	Компонент файла состоит из двух переменных: первая переменная – число, вторая – символ. Поместить в качестве первого компонента файла последний компонент, значение первой составляющей которого больше заданного числа.
20	Добавить в файл из строковых данных компонент, равный количеству символов, содержащихся в его *k*-ом компоненте.
21	Компонент файла состоит из двух переменных: первая переменная – целое число, вторая – символ. Поменять местами *k*-ый компонент с компонентом, значение которого кратно заданному числу (в предположении, что такой компонент присутствует однократно).
22	Компонент файла состоит из двух переменных: первая переменная – число, вторая – символ. Поменять местами последний компонент с первым.

Окончание табл. 6

№ варианта	Задание
23	Компонент файла состоит из двух переменных: первая переменная – число, вторая – строка. Заменить *k*-ый компонент на компонент с наименьшей длиной строки второй составляющей.
24	Компонент файла состоит из двух переменных: первая переменная – строка, вторая – символ. Если в *k*-ом компоненте значение первого символа первой составляющей равно символу второй составляющей, то заменить обе составляющие допустимым новым символом или допустимой новой строкой.
25	Компонент файла состоит из двух переменных: первая переменная – число, вторая – символ. Удалить из файла все данные, для которых значение первой переменной больше заданного числа.

Лабораторная работа

ФОРМИРОВАНИЕ И ОБРАБОТКА ДАННЫХ ФАЙЛОВОГО ТИПА. ТЕКСТОВЫЕ ФАЙЛЫ

Основы теории

Текстовый файл – это последовательность символов, разбитая на строки длиной от 0 до 255 символов, поэтому его можно рассматривать как разновидность файла типа *file of char*.

В текстовых файлах помимо признака конца файла *Eof* используется признак конца строки *Eoln*. Признак *Eoln* представляет собой последовательность из двух символов кода ASCII – символа с кодом #13 (“возврат каретки”) и символа с кодом #10 (“перевод строки”), выставляемых средой автоматически.

Текстовый файл можно представить как страницу книги, в конце каждой строки которой стоит *Eoln*.

Объявление текстового файла в программе осуществляется через стандартный тип *text*:

Var f: text; {*f* – файловая переменная}

➢ Системные средства для работы с текстовыми файлами

Процедура *Assign (var f: text; 'имя файла')* – ставит *имя файла* в соответствие файловой переменной *f;*

Процедура *Rewrite (var f: text)* – открывает новый файл для записи и устанавливает указатель на начало файла.

Процедура *Writeln (var f: text; S1, S2, ..., SN)* – записывает значения переменных *S1, S2, ..., SN* в файл и дополнительно формирует признак конца строки *Eoln*.

Процедура *Write (var f: text; S1, S2, ..., SN)* – записывает значения переменных *S1, S2, ..., SN* в файл, но не формирует признак конца строки *Eoln*.

Процедура *Append (var f: text)* – открывает уже существующий файл, и устанавливает указатель на конец файла. После такого открытия в конец текстового файла можно вписать дополнительную информацию.

Процедура *Reset (var f: text)* – открывает файл для чтения и устанавливает указатель на начало файла.

Процедура *Read* (*var f: text; S1, S2, ..., SN*) – считывает элементы строки по одному без перехода на новую строку.

Процедура *Readln (var f: text; S1, S2, ..., SN)* – считывает первый элемент строки в переменную *S1* и осуществляет переход к началу следующей строки.

Процедура *Readln (var f: text)* – просто осуществляет переход к началу следующей строки.

Функция *Eoln (var f: text): Boolean* – возвращает значение *True*, если текущая файловая позиция – конец строки или конец файла, и *False* в противном случае.

Функция *Eof (var f: text): Boolean* – возвращает значение *True*, если текущая файловая позиция – конец файла, и *False* в противном случае.

Функция *SeekEoln (var f: text): Boolean* – возвращает значение *True,* если текущая файловая позиция – конец строки или

конец файла, и *False*, если перед ними стоят лишь пробелы и/или символы табуляции (#9).

Функция *SeekEof (var f: text): Boolean* – возвращает значение *True*, если текущая файловая позиция – конец файла, или перед ним стоят лишь пробелы, признак конца строки и/или символы табуляции, и *False* в противном случае.

Процедура *SetTextBuf (var f:text; var Buf [; BufSize: word])* – устанавливает размер буфера ввода – вывода текстового файла *f* равным *BufSize* байт. Должна выполняться перед открытием файла *f*. Буфер размещается в переменной *Buf*. Выражение *[; BufSise: word])* может отсутствовать, если переменная *Buf* в программе уже объявлена. По умолчанию размер буфера равен 128 байтам.

Процедура *Flush (var f: text)* – выводит текущее содержимое буфера файла *f* в физический файл, не дожидаясь заполнения буфера до конца. Имеет смысл только при записи в файл.

Процедура *Erase (var f: text)* – удаляет текстовый файл с диска.

Процедура *ReName (var f: text; NewName: string)* – переименовывает файл.

➢ Организация работы с текстовыми файлами

На практике обработка текстовых файлов сводится к считыванию всего файла в память, а затем к записи уже модифицированного файла на диск.

Записывать в текстовый файл и читать из текстового файла разрешается только переменные целого, вещественного, символьного и строкового типов;

При наборе каждая строка завершается клавишей Enter, что соответствует маркеру конца строки.

Признак конца файла формируется автоматически при программном закрытии файла и записи файла при его наборе в текстовом редакторе.

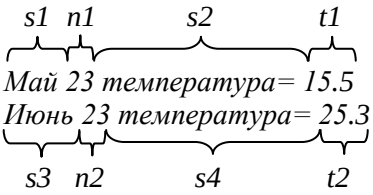
Числовые данные при записи в файл автоматически преобразуются в цепочку символов, а при считывании автоматически преобразуются к значению того типа переменной, которая используется в файловых операциях.

Текстовый файл может быть сформирован программным путём или набран в любом текстовом редакторе. Разделителем элементов является пробел.

Набор файла в текстовом редакторе осуществляется по общим правилам набора текста:

- элементы файла (слова: символы, строки, числа) разделяются пробелом;
- в конце каждой строки осуществляется переход на новую строку;
- готовый файл сохраняется; для идентификации текстовых файлов, а, следовательно, и быстрого поиска рекомендуется в качестве расширения использовать *txt, dat* или *dan.*

При считывании *элементы – числа* автоматически переводятся в значения объявленных типов. Например, файл содержит две строки:



При считывании по фрагменту:

```
Var s1, s2, s3, s4: string[20];
    n1, n2: integer;
    t1, t2: real;
    f: text;

BEGIN
        …
    readln (f, s1, n1, s2, t1);   readln (f, s3, n2, s4, t2);
    writeln (‘Средняя температура ’, n2, ‘-го числа = ’,
      ((t1+t2)/2):6:1);
        …
END.
```

получим результат:

Средняя температура 23-го числа = 20.4

При считывании по фрагменту:

```
var Str1, Str2: string;
    f1: text;
        …
BEGIN
        ...
    readln (f1, Str1);
    readln (f1, Str2);
        ...
END.
```

данные будут считаны в переменные типа *string*, цифровые элементы которых являются текстом.

Для выполнения *арифметических действий* над цифровыми элементами необходимо:

- выделить цифровые элементы из строк;
- перевести их значения из строковых данных в числовые;
- выполнить арифметические действия.

При формировании файловых данных программным путём необходимо назначить буфер ввода – вывода и обеспечить сброс данных из него после завершения записи данных в файл. Распознавание данных, считываемых из текстовых файлов, сформированных программным путём, связано с определёнными трудностями, поэтому рекомендуем формировать *исходные файлы* набором в текстовых редакторах.

При добавлении данных в конец файла (процедура *Append ( )*) рекомендуем сохранять *форму данных*, находящихся в файле, и добавляемые данные оформлять как строки.

Например,

```
    …
Append (f1);
writeln (f1, 'июль 23 температура 27.6');
    …
```

или

```
        …
str1:= 'июль 23 температура 27.6';
Append (f1);
writeln (f1,  str1);
        …
```

или

```
        …
s1:= 'июль';  s2:= '23';  s3:= 'температура';  s4:= '27.6';
str1:= s1+' '+s2+' '+s3+' '+s4; {элементы строки соединены
                                 пробелом}
Append (f1);
writeln (f1, str1);
        …
```

Контроль входных знаний

1. Чем отличается текстовый файл от файла типа *char*?
2. Чем отличается процедура *Rewrite (f)* от процедуры *Append (f)*?
3. Можно ли применить операцию *Erase (f)* для незакрытого файла?
4. Как осуществить считывание строки

Иванов С. П. 1990 5.6

без дополнительного преобразования цифровых данных в числовые для их дальнейшего использования?

5. В файле записаны данные результатов сдачи экзаменов группой студентов по предметам:

Предмет, Количество студентов, Оценка

Данные были считаны в массив переменных типа *string*. Какие действия надо выполнить, чтобы вычислить значение среднего балла?

6. Как организовать считывание из текстового файла данных заданной строки?
7. Как создать копию файла?

Задания к выполнению

Создать в любом текстовом редакторе текстовый файл. Считать данные с построчным выводом их на экран. Выполнить обработку данных по вариантам (см. табл. 7).

Таблица 7

Варианты заданий

№ варианта	Задание
1	Одним из элементов каждой строки текстового файла является целое число. Добавить в файл строку, в которой число является наименьшим из числовых элементов файла.
2	Каждая строка текстового файла содержит текстовые и два числовых элемента. Добавить в файл строку, в которой числовые элементы являются суммами соответствующих им элементов.
3	Текстовый файл содержит элементы типа *record*: *Type Trec = record* *Fam: string[15];* {Фамилия} *Sex: char;* {пол: M – мужской, W – женский} *Mark: byte;* {оценка} *end;* Добавить в текстовый файл строку данных первой девушки, получившей на экзамене отличную оценку.
4	В каждой строке текстового файла текстовые элементы чередуются с числовыми. Добавить в файл две строки: пустую и строку, в которой числовыми элементами являются суммы соответствующих числовых элементов первых двух строк.
5	В каждой строке текстового файла текстовые элементы чередуются с числовыми. Сформировать новый файл из строк исходного файла, все числовые элементы которых имеют положительное значение.
6	В текстовом файле, каждая строка которого содержит элементы вещественных и строковых типов, определить номер последней строки, содержащей только числа, принадлежащие заданному интервалу *[a, b]*.Добавить в файл строку любой длины, содержащей отрицательные числа.

Продолжение табл. 7

№ варианта	Задание
7	В каждой строке текстового файла текстовые элементы чередуются с числовыми. Добавить строку, в которой числовые элементы представляют количество положительных элементов соответствующих им элементов.
8	Первым элементом каждой строки текстового файла является число. Создать копию этого файла и найти сумму первых элементов всех строк.
9	Первым элементом каждой строки текстового файла является символ. Каждая строка текстового файла содержит данные – символы и данные – числа. Найти среднее значение чисел тех строк файла, которые начинаются с заданного символа. Добавить в файл новую строку, содержащую произвольное количество слов.
10	Сформировать копию исходного файла, дописав в начало каждой строки длину её первого элемента. Определить их суммарную длину.
11	Текстовый файл содержит элементы типа *record:* *Type Trec = record* *Fam: string[15];* {Фамилия} *Sex: char*; {пол: M – мужской, W – женский} *Mark: byte*; {оценка} *end;* Вписать в первую строку файла данные о количестве девушек, получивших на экзамене отличные оценки.
12	Текстовый файл содержит данные о наименовании проданного изделия, его количестве и цене. Определить суммарную стоимость проданных изделий. Дополнить файл данными о новом изделии.
13	Первым элементом каждой строки текстового файла является число. Каждая строка текстового файла содержит данные – символы и данные – числа. Строки файла, начинающиеся с заданного числа, скопировать в новый файл. Вычислить произведение числовых данных, входящих в первые строки каждого файла.

Продолжение табл. 7

№ варианта	Задание
14	В каждой строке текстового файла текстовые элементы чередуются с числовыми. Добавить в файл две строки: пустую и строку, в которой числовыми элементами являются суммы соответствующих числовых элементов первых двух строк.
15	Текстовый файл содержит данные о номере телефона, его владельце и суммарном времени разговоров за прошлый месяц. Сформировать новый файл – извещение о сумме выплат за разговоры, если стоимость минуты разговора равна некоторой заданной величине.
16	Текстовый файл содержит элементы типа *record*: *Type Trec= record* *Fam: string[15];* {Фамилия} *Sex: char;* {пол: M – мужской, W – женский} *Mark: byte;* {оценка} *end;* Добавить в текстовый файл строку: *средний балл* <значение>.
17	Каждая строка текстового файла содержит данные – строки и данные – числа и начинается с числа. Добавить в файл новую строку, первый элемент которой равен наименьшему числу среди первых элементов каждой строки.
18	Судейская коллегия массового лёгкоатлетического забега установила базовую величину для расчёта денежного вознаграждения равной *a* (у.е. / мин). Результаты забега фиксировались в файле, каждая строка которого имела вид: *Фамилия Имя, Год рождения, Результат* Определить призовое денежное вознаграждение победителю забега и добавить в файл строку данных, содержащую *Фамилия Имя, Год рождения, Результат, Призовая сумма.*

Окончание табл. 7

№ варианта	Задание
19	Каждая строка текстового файла содержит данные – символы и данные – числа. Вычислить произведение числовых данных, входящих в первую строку, начинающуюся с заданного символа и добавить в файл данные этой строки, изменив знаки данных – чисел на противоположные.
20	В каждой строке текстового файла символьные элементы чередуются с числовыми. Добавить строку, в которой числовые элементы являются средними значениями соответствующих чисел двух последних строк.
21	Текстовый файл содержит данные о владельце телефона, номере телефона и суммарном разговорном времени. Сформировать новый файл – извещение о выплатах за разговорное время.
22	Первым элементом каждой строки текстового файла является число. Строки, начинающиеся с чётного числа, скопировать в новый файл и добавить в него строку – сообщение о средней сумме этих чётных чисел.
23	Текстовый файл содержит элементы типа *record:* *Type Trec= record* *Fam: string[15];* {Фамилия} *Sex: char;* {пол: M – мужской, W – женский} *Year: word;* {год рождения} *end;* Добавить в текстовый файл строку данных о среднем возрасте лиц мужского пола, получивших на экзамене оценку «отлично».
24	Первым элементом каждой строки текстового файла является число. Строки, начинающиеся с отрицательных чисел, скопировать в новый файл и определить номер строки исходного файла, содержащей наибольшее из этих чисел.
25	Текстовый файл содержит данные о владельце телефона, номере телефона и суммарном разговорном времени. Добавить в файл строку – извещение о выплатах за наибольшее разговорное время.

Лабораторная работа

ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

Основы теории

Память современных компьютеров представлена 65536 перекрывающимися 64-Кбайтовыми сегментами. Начало каждого сегмента отстоит от соседнего на 16 байт, поэтому сегменты имеют 16-ричные адреса 0000h, 0010h, 0020h и т.д.

Статистические структуры данных – это структуры объявленного типа, память под которые отводится в процессе компиляции и остаётся неизменной на протяжении работы программы.

Динамические структуры данных – это структуры, которые можно создавать / уничтожать в процессе выполнения программы. При их появлении в программе выделяется фиксированный объём памяти для хранения *адреса* размещаемой переменной, а не самой переменной. Сама переменная / переменные размещается в динамической памяти – в *куче*. В куче можно выделять блоки для хранения данных только на период их использования. После завершения обработки этих данных, выделенные блоки необходимо возвращать в свободную область кучи и использовать для размещения новых данных. При динамическом размещении данных заранее не известны ни их тип, ни количество размещаемых данных, к ним нельзя обращаться по именам, как к статическим переменным.

- Адреса и указатели

Номера ячеек памяти – байтов называются адресами.

Каждая ячейка памяти имеет составной адрес

сегмент : *смещение*

Смещение указывает, сколько байт от начала сегмента необходимо пропустить, чтобы обратиться к нужному адресу.

Средством управления динамической памятью являются указатели. ***Указатель –*** это переменная, которая в качестве своего значения содержит адрес байта памяти.

Значение указателя не может быть в явном виде выведено на экран или печать. Его надо предварительно расшифровать: выделить сегмент и смещение.

Для этих целей используются функции:

для сегмента: *Seg (X): word;*
для смещения: *Ofs (X): word;*

где *X* – выражение любого типа, имя процедуры или функции.

Указатели могут обмениваться значениями через оператор присваивания *(:=)* и сравниваться как равно *(=)* или неравно *(<>)*. Два указателя считаются равными, если только равны их сегменты и смещения.

С помощью указателей можно размещать в динамической памяти данные любого типа, кроме файловых.

При работе с указателями рекомендуется использовать в начале или в конце идентификаторов букву *P* или сочетание *Ptr*.

Среди указателей выделяется один специальный *Nil* (нулевой, пустой), который ни на что не указывает. Указатель *Nil* считается константой, совместимой с любым ссылочным типом. Значение *Nil* присваивается указателю, когда его надо отменить или в начале инициализации программы. Это позволяет проверить значение указателя, прежде чем присвоить ему какое-либо значение.

Переменная-указатель связывается с некоторым типом данных. Такие указатели называются типизированными. Для их объявления используется значок ^, размещаемый перед соответствующим типом.

Например,

```
Type
    TIntPtr = ^integer;   {объявление целого динамического
                           типа:}
    TM1 = array[1..20] of real;
    TM1Ptr = ^TM1;   {объявление вещественного массива
                      динамического типа}
    TSimPtr = ^char;  {объявление символьного динамиче-
                       ского типа}
```

➢ Объявление указателей

Переменную-указатель можно объявить непосредственно или через ссылочный тип данных.

Например,

```
Var
   PtrA: ^real;        {непосредственное объявление указателя
                        на переменную вещественного типа}
```

или через ссылочный тип.

Например,

```
PtrInt: TIntPtr;        {объявление указателя на массив данных
                         через ссылочный тип:}
PtrM1: TM1Ptr;
или
PtrM1: ^TM1;
```

В Турбо Паскале можно объявлять указатель и не связывать его с каким-либо конкретным типом данных. Для этого служит стандартный тип *Pointer*.

Например,

```
Var
   P: Pointer;
```

Указатели такого рода называют *нетипизированными.* С их помощью удобно размещать данные, структура и тип которых меняются в ходе выполнения программы. Эти указатели совместимы с любыми типизированными указателями.

➢ Доступ к динамической переменной

Непосредственно к динамическим переменным можно обратиться по структуре:

<имя переменной>^.

Например,

```
PtrA^:= 2.085;
PtrM1^[i]:= A^;
for i:= 1 to Pn^ do begin ... end;
if Pm^ > Pn^ then ...;
```

Операции, в которых доступ к самой переменной осуществляется через указатель с завершающим символом ^, называются ***разыменованием указателя***.

➢ Доступ к адресам

Для определения адреса любого объекта программы, переменной, процедуры или функции применяются: операция **@**, равносильная ей функция *Addr (X)* и функция *Ptr (Seg (X), Ofs (X))*, возвращающие результат типа *Pointer*, в котором содержится адрес аргумента *X*.

Например,

```
P:= @X;                    P:= Addr (X);
P:= Ptr (Seg (X), Ofs (X)); P1:= Ptr (0020, 0003);
```

➢ Характеристики кучи

Характеристики кучи хранятся в стандартных переменных:

HeapOrg – начало кучи;
HeapEnd – её конец;
HeapPtr – текущая граница незанятой динамической памяти.

➢ Выделение и освобождение динамической памяти

Перед выполнением каких-либо действий над динамической переменной необходимо запросить для неё память (место в куче), а при завершении действий над ней, память освободить, т.е. вернуть в кучу.

Выделение памяти для типизированных и нетипизированных указателей осуществляется с использованием разных процедур:

New (var P: <типизированный указатель>);
GetMem (var P: Pointer; Size: word); {отводит место объёмом Size байт}

За одно обращение к процедуре можно зарезервировать не более 65521 байт.

Освобождение памяти для типизированных и нетипизированных указателей осуществляется тоже с использованием разных процедур:

Dispose (*var P: <типизированный указатель>);*
FreeMem (var P: Pointer; Size: word);

➢ Анализ состояния кучи. Функции *MaxAvail* и *MemAvail*

Эти функции анализируют количество свободной памяти, как ещё не использованной, так и получающейся в результате освобождения динамических переменных процедурами *Dispose* и *FreeMem*. Возвращаемые значения этих функций имеют тип *LongInt*.

Функция *MaxAvail* – возвращает размер в байтах наибольшего непрерывного участка кучи.

Функция *MemAvail* – возвращает размер в байтах общего свободного пространства кучи.

➢ Определение размеров типов, переменных и функций

Функция *SizeOf (X): word;* – возвращает объём в байтах, занимаемый *X*. Здесь *X* – имя переменной, типа или функции, например:

```
Type
    Dim = Array[1..10, 1..10] of real;
Const
    C: LongInt = 12245;
Var
    St: String;
Begin
    writeln (SizeOf (Dim):12, SizeOf (C):10, SizeOf (St));
End.
```

Контроль входных знаний

1. На каком этапе обработки программы осуществляется выделение памяти для размещения статических данных, а на каком – для размещения динамических данных?
2. Может ли значением указателя быть данное вещественного типа?
3. Переменные *x* и *y* объявлены как: *Var x, y: ^integer*; Каковы типы переменных *x* и *x^* ?
4. Переменные *p, q, r* описаны как

```
Var p, q: ^ byte;
      r: ^ char;
```

Какие из следующих операторов неправильные и почему?

```
p:= q;     q:= r;   p:= Nil;
r:= Nil;   q:= ^p;
if q > Nil then q^:= p^;
if q <> r then read (r^);
if q = p then writeln (q);
```

5. Имеется программа:

```
Var
    x: ^integer;
    y: integer;
BEGIN                  {a}
      New (x);         {b}
      x^:= 10;         {c}
      y:= x^ + 6;
      Dispose (x);     {d}
      writeln (y);     {e}
END.
```

Какие переменные существуют в каждой из точек *a, b, c, d, e* и каковы их значения в эти моменты?

6. Что будет выдано на печать в результате выполнения следующих операторов при *x^ = 5, y^ = 8*:

```
x^:= y^;
if x = y then x:= Nil else if x^ = y^ then y:= x;
if x = y then g^:= 4;
writeln (g^);
```

7. Оператор присваивания имеет вид:

```
P:=Ptr (seg (X^[i]), ofs (X^[i]);
```

Какое значение приобретет переменная *P* после выполнения этого оператора?

Задания для выполнения

1. Определить характеристики кучи.
2. Случайным образом сформировать одномерные вещественный и целый массивы и разместить их в динамической памяти.
3. Разработать процедуру вывода этих массивов.
4. Вывести размеры памяти, занятой этими массивами и адрес случайной переменной каждого массива.
5. Сформировать указатель на случайную переменную одного из массивов и на случайную переменную *A*.
6. Увеличить значение элемента массива (п.5) на значение динамической переменной *A*.
7. Определить размеры наибольшего свободного участка кучи и размеры всей свободной области.
8. Сформировать указатель строкового типа. Проверить его на нуль. Разместить по нему свою фамилию, имя и отчество. Определить объём памяти, занятой этой информацией.
9. Заменить информацию п.8 на информацию о дате своего рождения.

Лабораторная работа

РАБОТА В ГРАФИЧЕСКОМ ВИДЕОРЕЖИМЕ

Основы теории

Работа в графическом режиме выполняется с помощью предопределённых констант, типов, процедур и функций стандартного модуля *Graph*.

Пиксель – точка графического экрана**.**

Адаптер **–** устройство связи для взаимодействия между собой устройств с различным способом представления данных.

Драйвер – это программа, управляющая каким-либо устройством.

Разрешающая способность адаптера – количество пикселей на экране. Один и тот же адаптер может иметь несколько режимов работы с различной разрешающей способностью. Для режима *VGAHi* драйвера *VGA* разрешающая способность равна 640×480.

Инициализация **–** это процесс перемещения содержимого модуля с жёсткого диска в оперативную память и подача команды на выполнение определённой работы.

Программа, использующая графику, должна содержать:

1. предложение *Uses Graph* для подключения модуля к основной программе;
2. объявление процедуры *GrInit* – инициализации графического режима, и её вызов;
3. вызов процедуры *CloseGraph* – закрытие графического режима.

➢ Процедура инициализации графического режима

```
Procedure GrInit;
Var GraphDriver: integer;   {графический адаптер}
    GraphMode: integer;     {графический режим}
    ErrorCode: integer;     {код ошибки}
```

```
Begin
GraphDriver:= detect;{автораспознавание драйвера, в этом
                      случае установка параметра GraphMode не
                      требуется}
InitGraph (GraphDriver, GraphMode, '<полный путь к драйверу
                      VGA>'); {инициализация графического
                      режима: если драйвер находится в теку-
                      щем каталоге, то в последнем параметре
                      путь к нему можно не прописывать}
ErrorCode:= GraphResult;      {результат инициализации:}
If ErrorCode <> GrOk then     {GrOk – системная константа,
                      равная нулю, характеризующая
                      успешное завершение инициализа-
                      ции}
    begin
        writeln ('Ошибка графики', GraphErrorMsg (Error-
Code));
        writeln ('Работа программы прервана');
        Halt (1);
    end;
End;
```

➢ Системные процедуры

GraphDefaults – очищает экран, возвращает курсор в положение (0, 0) и устанавливает все параметры графической системы в исходное, предусмотренное по умолчанию, состояние.

ClearDevice – очищает экран и устанавливает курсор в положение (0, 0).

PutPixel (X, Y: integer; Pixel: word) – строит пиксель с координатами *(X, Y)*, цветом *Pixel* ($0 \le Pixel \le 15$).

Line (X1, Y1, X2, Y2: integer) – рисует отрезок прямой между двумя точками с координатами *(X1, Y1)* и *(X2, Y2)*.

MoveTo (X, Y: integer) – перемещает указатель текущей позиции в точку (*X, Y*).

LineTo (X, Y: integer) – рисует линию от текущей позиции до точки с координатами *(X, Y)*.

SetLineStyle (LineStyle: word; Pattern: word; Thickness: word) – устанавливает толщину линии и её стиль.

Некоторые значения константы *LineStyle*:

0 – сплошная линия;

1 – точечная линия;

2 – штрих пунктирная линия;

3 – пунктирная линия и т.д.

Значение параметра *Pattern* обычно задается равным нулю.

Константа *Thickness* может принимать значения:

l – толщина линии в один пиксель;

3 – толщина линии в три пикселя.

Bar (X1, Y1, X2, Y2: integer) – рисует закрашенный прямоугольник, используя текущий стиль линии и цвет. Координаты *(X1, Y1)* определяют верхний левый угол прямоугольника, а координаты *(X2, Y2)* – нижний правый угол.

Circle (X, Y: integer; Radius: word) – рисует окружность с центром *(X, Y)* и радиусом *Radius*.

OutTextXY (X, Y: integer; TextString: string) – выдаёт строку *TextString* на дисплей с позиции, имеющей координаты *(X, Y)*.

SetTextStyle (Font, Direction: word; CharSize: word) – устанавливает текущий текстовый шрифт, стиль и коэффициент увеличения символов.

Константа *Font* определяет размер текста:

0 – матричный шрифт 8×8 {по умолчанию};

1 – полужирный шрифт;

2 – светлый шрифт {тонкое начертание};

3 – книжная гарнитура {рубленый шрифт};

4 – готический шрифт.

Константа *Direction* определяет расположение текста:

l – горизонтально слева направо;

2 – вертикально снизу вверх.

CharSize (1..10) – устанавливает размер каждого символа.

SetColor (Color: word) – устанавливает цвет пера.

SetBkColor (Color: word) – устанавливает цвет фона.

FloodFill (X, Y: integer; Border: word) – заливает вокруг точки *(X, Y)* область, ограниченную замкнутой линией цвета *Border*.

SetFillStyle (Pattern: word; Color: word) – назначает шаблон заливки *(0 ≤ Pattern ≤ 12)* и цвет заливки *(0 ≤ Color ≤ 15)*.

➢ Работа с фрагментами изображений

Функция *ImageSize (X1, Y1, X2, Y2: integer): word* – возвращает размер памяти в байтах, необходимый для сохранения прямоугольной области экрана, охватывающей рисунок.

Процедура *GetMem (P: pointer; SizeOf: word)* – выделяет динамическую память для хранения рисунка.

Процедура *FreeMem (P: pointer; SizeOf: word)* – освобождает выделенную память.

В процедурах *GetMem* и *FreeMem* параметр *P* – указатель на первый байт выделенной динамической памяти, *SizeOf* – размер памяти, определённый функцией *ImageSize*.

Процедура *GetImage (X1, Y1, X2, Y2: integer; var P)* – записывает прямоугольную область экрана, содержащую изображение, в динамическую память по указателю *P*, назначенному в процедуре *GetMem*.

Процедура *PutImage (X1, Y1: integer; var P; Mode: word)* – восстанавливает изображение из буфера *P* в прямоугольник, левый верхний угол которого определён координатами *(X, Y)*. Параметр *Mode* даёт возможность определять режим вывода изображения: можно суммировать изображение на экране и изображение в буфере, можно уничтожать изображение, находящееся в определённой области, можно инвертировать изображение, содержащееся в буфере. Эти операции задаются константой *Mode*, принимающей значения:

CopyPut = 0 {операция *Mov* – замещение}
XorPut = 1 {операция *Xor* – исключающее ИЛИ}
OrPut = 2 {операция *Or* – ИЛИ}
AndPut = 3 {операция *And* – И}
NotPut = 4 {операция *Not* – НЕ}

Демонстрационный пример

```
Program P1;
    ...
Var X1, Y1, X2, Y2: integer;     {координаты прямоугольника, содержащего фрагмент изображения}
    Xp, Yp: integer;             {координаты точки для вывода фрагмента}
    Size: word;                  {размер прямоугольника, содержащего фрагмент изображения}
    P: pointer;                  {указатель на буфер – участок динамической памяти}
  …
BEGIN
    ...
  Size:= ImageSize (X1, Y1, X2, Y2);   {определение размера памяти, необходимой для размещения изображения, находящегося в области, ограниченной прямоугольником с координатами (X1, Y1, X2, Y2)}
  GetMem (P, Size);              {размещение указателя P в динамической памяти}
  GetImage (X1, Y1, X2, Y2, P^);       {запись фрагмента в буфер}
    …
  PutImage (Xp, Yp, P^, XorPut);       {вывод фрагмента из буфера на экран}
    …
  PutImage (Xp, Yp, P^, XorPut);       {стирание фрагмента}
    …
  FreeMem (P, Size);             {освобождение динамической памяти}
    …
END.
```

Контроль входных знаний

1. Как называется точка графического экрана?
2. Какие действия выполняет процедура инициализации графического режима?
3. Как залить заданным цветом замкнутую геометрическую фигуру?
4. Как организовать движение фрагмента изображения вдоль любой горизонтальной линии?
5. Перемещаясь по экрану жёлтого цвета, зелёный прямоугольник оставляет след в виде своих предыдущих копий. Как убрать оставляемый след?

Задания для выполнения

1. Вывести заданным шрифтом вертикально на экран наименование лабораторной работы.
2. Средствами модуля *Graph* нарисовать цветную фигуру и организовать движение её по заданной траектории.

Таблица 8

Варианты заданий

№ варианта	Шрифт	Фигура	Траектория движения
1	Готический	Снежинка, имеющая сердцевину в виде шара	Раскручивающаяся спираль
2	Рубленный	Три связанных разноцветных шара	Кубическая парабола $y = ax^3$
3	Матричный	Два частично перекрывающихся прямоугольника	Полукубическая парабола $y = ax^{\frac{3}{2}}$
4	Полужирный	Прямоугольник, поставленный на пьедестал	Закручивающаяся спираль
5	Книжная гарнитура	Шар на поставленном прямоугольнике	Синусоида $y(x) = a\sin(x + b)$

Продолжение табл. 8

№ варианта	Шрифт	Фигура	Траектория движения
6	Готический	Два прямоугольника, имеющих один общий угол	Отрезок прямой, проведённой из левого верхнего в правый нижний угол экрана
7	Матричный	Три шара, образующих снеговика	Косинусоида $y(x) = a\cos(x + b)$
8	Светлый	Три шара, образующих дракона	Синусоида $y(x) = a\sin(x^2 + b)$
9	Полужирный	Поставленный прямоугольник и шар у его подножия	Кубическая парабола $y = ax^3$
10	Книжная гарнитура	Гирлянда из трёх шаров	Удлинённая циклоида $\begin{cases} x = a\varphi - b\sin\varphi \\ y = a - b\sin\varphi \end{cases} \quad (a < b)$
11	Готический	Три шара, расположенных в виде треугольника	Верзьера Аньези $y = \dfrac{8a^3}{x^2 + 4a^2} \quad (a > 0)$
12	Матричный	Прямоугольник на шаре	Обыкновенная циклоида $\begin{cases} x = a(\varphi - \sin\varphi \\ y = a(1 - \cos\varphi) \end{cases}$
13	Светлый	Шар на треугольнике	Циссоида Диокла $y^2 = \dfrac{x^3}{a - x} \quad (a > 0)$
14	Готический	Ёлка из двух треугольников	Верзьера Аньези $y = \dfrac{8a^3}{x^2 + 4a^2} \quad (a > 0)$

Окончание табл. 8

№ варианта	Шрифт	Фигура	Траектория движения
15	Матричный	Два воздушных шара на нитках, привязанных к одной точке	Удлинённая циклоида $\begin{cases} x = a\varphi - b\sin\varphi \\ y = a - b\cos\varphi \end{cases}$
16	Светлый	Элементарный парусник, собранный из отрезков прямых линий	Раскручивающаяся спираль
17	Полужирный	Регулировщик на автомагистрали с поднятой вверх рукой	Строфоида $y^2 = x^2 \cdot \frac{a-x}{a+x} \quad (a>0)$
18	Книжная гарнитура	Ёлочная гирлянда, состоящая из двух шаров и треугольника	Окружность $x^2 + y^2 = r^2$
19	Готический	Гимнаст с разведёнными в стороны руками	Закручивающаяся спираль
20	Матричный	Пирамида, состоящая из трёх шаров	Синусоида $y(x) = a\sin(x+b)$
21	Светлый	Треугольник, одной из вершин поставленный на прямоугольник	Отрезок прямой, проведённой из левого верхнего в правый нижний угол экрана
22	Готический	Шар, поставленный на вершину треугольника	Косинусоида $y(x) = a\cos(x+b)$
23	Матричный	Треугольник, размещённых на двух шарах	Синусоида $y(x) = a\sin(x^2+b)$
24	Светлый	Три треугольника, создающих силуэт горных вершин	Строфоида $y^2 = x^2 \cdot \frac{a-x}{a+x} \quad (a>0)$
25	Полужирный	Пирамида, состоящая из двух прямоугольников и треугольника	Закручивающаяся спираль

Лабораторная работа

ПОСЛЕДОВАТЕЛЬНОСТИ, РЕКУРРЕНТНЫЕ СООТНОШЕНИЯ

Основы теории

Последовательность – функция, заданная на множестве натуральных чисел; обозначается обычно:

$$a_1, a_2, \ldots, a_n \quad или \quad \{a_n\}, n = 1,2,\ldots$$

С каждым элементом последовательности связан его номер, поэтому будем рассматривать последовательность как функцию её аргумента *n* и обозначать как *F = f (n).*

Например,

2, 4, 6, 8, 10, 12, … – числовая последовательность положительных чётных чисел;

0, 1, 1, 2, 3, 5, 8, … – числовая последовательность из чисел Фибоначчи;

1, 2, 6, 24, 120, … – числовая последовательность, в которой каждый член является произведением всех натуральных чисел от 1 до данного натурального числа *n.* В математике такое произведение называется факториалом (от английского слова *factor* – множитель), записывается в виде:

$$n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \ldots \cdot n$$

и читается как “эн факториал”. По определению *0!* принято считать равным единице.

Обобщённым членом последовательности называют значение её *n*–го члена, т.е. само число *f (n).*

Имеется два способа определения последовательности:

- явной формулой, выражающей обобщённый член как функцию от *n*, например, *f (n) = 2n*, для *n≥0*;
- уравнением, связывающим обобщённый член с одним или несколькими другими членами последовательно-

сти, в комбинации с одним или несколькими явными значениями первых членов, например,

$$f(n) = f(n-1) + n \quad \text{для } n>0 \qquad (1)$$
$$f(0) = 0 \qquad (2)$$

Равенство, выражающее член последовательности через один или несколько предыдущих членов, называется ***рекуррентным соотношением или рекуррентным уравнением.***

Уравнение (1) является рекуррентным, а условие (2) – ***начальным условием*** для этого уравнения. Начальное условие может быть указано для значения n, отличного от 0, например, для $n = 1$. Для некоторых рекуррентных соотношений, например, для рекуррентного соотношения

$$F(n) = F(n-1)+F(n-2), \qquad (3)$$

определяющего числа Фибоначчи, начальное условие задается при двух значениях n:

$$F(0) = 0; \quad F(1)= 1 \qquad (4)$$

➢ Области применения рекуррентных соотношений

Рекуррентные соотношения полезно использовать в программировании вычислений, связанных с рекуррентными последовательностями. Соотношения позволяют формально выражать зависимости между величинами, которые вычисляются последовательно, а на основе этих выражений легко строить циклы.

Для решения задачи нужно:

- понять, что величины образуют рекуррентную последовательность;
- записать соответствующее рекуррентное соотношение;
- определить начальные условия;
- сформулировать условие, управляющее повторением цикла.

Демонстрационные примеры

Пример 1. Определение рекуррентных соотношений для арифметической и геометрической прогрессий.

Члены арифметической прогрессии связаны соотношением

$$a_n = a_{n-1}+d,$$

а для членов геометрической прогрессии справедливо соотношение

$$a_n = a_{n-1} \cdot q$$

Задав разность d или знаменатель q и начальное условие – значение первого члена a_1, получим конкретную прогрессию, определяемую через рекуррентные соотношения:

$$a_1 = c \text{ (} c - \textit{константа)}$$
$$a_n = a_{n-1}+d$$

и

$$a_1 = c \text{ (} c - \textit{константа)}$$
$$a_n = a_{n-1} \cdot q$$

Пример 2. Определение рекуррентного соотношения для n!

По определению 0! принято считать равным 1, на основе этого можно записать

$$1! = 0! \cdot 1; \quad 2! = 1! \cdot 2; \quad 3! = 2! \cdot 3; \quad \ldots \quad ; \quad n! = (n-1)! \cdot n$$

Соотношение

$$n! = \begin{cases} 1, & \textit{если } n = 0 \\ (n-1)! \cdot n, & \textit{если } n > 0 \end{cases}$$

является рекуррентным при вычислении n!

Пример 3. Определение рекуррентного соотношения для членов степенного ряда.

Степенным рядом называется функциональный ряд вида

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n + \ldots = \sum_{n=0}^{\infty} a_n x^n$$

где $a_0, a_1, a_2, a_3, \ldots, a_n$ – постоянные коэффициенты.

Областью сходимости степенного ряда всегда является некоторый интервал *(-R, R)* с центром в точке $x = 0$, называемый ***интервалом сходимости*** степенного ряда. Число *R* называется радиусом сходимости степенного ряда. Во всех внутренних точках своего интервала сходимости степенной ряд сходится абсолютно. На концах же этого интервала ряд может сходиться, либо расходиться. Если радиус сходимости степенного ряда *R* равен нулю, то этот ряд сходится только в одной точке $x = 0$; если радиус сходимости бесконечен, то ряд сходится при всех значениях аргумента $x\ (-\infty < x < +\infty)$.

Функцию $y = f(x)$, непрерывную и имеющую производные всех порядков в промежутке (*-R, R*), можно представить как сумму степенного ряда вида

$$f(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \ldots + \frac{f^{(n)}(0)}{n!}x^n + \ldots,$$

называемого рядом Маклорена для данной функции *f(x)*.

Вычисление членов ряда Маклорена для функции

$$\sin x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + \frac{(-1)^n x^{2n+1}}{(2n+1)!} + \ldots,\ (-\infty < x < +\infty)$$

“в лоб” связано с вычислением степени и факториала. Получим рекуррентное соотношение, для чего найдём зависимость между двумя соседними членами ряда a_n и a_{n+1}:

$$a_n = \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Из этой формулы путём замены n на $n+1$ получим выражение для a_{n+1}:

$$a_{n+1} = \frac{(-1)^{n+1} x^{2(n+1)+1}}{[2(n+1)+1]!} = \frac{(-1)^{n+1} x^{2n+3}}{(2n+3)!}.$$

Определим величину $q = \frac{a_{n+1}}{a_n}$:

$$q = \frac{(-1)^{n+1} x^{2n+3}(2n+1)!}{(2n+3)!(-1)^n x^{2n+1}} = \frac{-x^2}{(2n+2)(2n+3)}.$$

Следовательно, рекуррентное соотношение вычисления членов ряда Маклорена для функции $y = sin(x)$ имеет вид:

$$a_{n+1} = a_n \cdot \frac{(-x^2)}{2(n+1)(2n+3)}, \quad (n = 0,1,2,3,\ldots)$$
$$при\ a_0 = x.$$

Степенные ряды широко применяются для приближённого вычисления значения функции $y = f(x)$ при аргументе x, принадлежащем области сходимости ряда. Алгоритмизация таких вычислений связана с итерационным процессом вычисления частичной суммы ряда.

Контроль входных знаний

1. Почему последовательность можно рассматривать как функцию, зависящую от номера ее элемента?
2. Чем является обобщённый член последовательности?
3. Как можно определить последовательность?

4. Какое соотношение называется рекуррентным?
5. Какой смысл заложен в понятие *“начальное условие”* для рекуррентного уравнения?
6. Что необходимо определить при разработке алгоритмов решения задач, связанных с рекуррентными уравнениями?
7. Опишите этапы получения рекуррентного соотношения для вычисления членов степенного ряда.
8. Сформулируйте задачу приближённого вычисления функции $y = f(x)$ с заданной точностью на основе разложения этой функции в ряд?
9. Во всех ли алгоритмах, реализующих вычисление функции через разложение её в ряд, начальное значение суммы следует полагать равным нулю?

Задания для выполнения

Вычислить значение функции с точностью $\varepsilon = 0.0001$ при конкретном значении x и a.

Таблица 9

Варианты заданий

№ варианта	Функция и ее разложение в ряд	Область сходимости ряда
1	$a^x = \sum_{n=0}^{\infty} \frac{(x \cdot \ln a)^n}{n!}$	$a > 0$ $-\infty < x < +\infty$
2	$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$	$-\infty < x < +\infty$
3	$\cos x = \sum_{n=0}^{\infty} \left[(-1)^n \frac{x^{2n}}{(2n)!} \right]$	$-\infty < x < +\infty$
4	$arctgx = \sum_{n=0}^{\infty} \left[(-1)^n \frac{x^{2n+1}}{2n+1} \right]$	$x^2 \le 1$
5	$arctgx = \frac{\pi}{2} - \sum_{n=0}^{\infty} \left[(-1)^n \frac{1}{(2n+1) \cdot x^{2n+1}} \right]$	$x^2 \ge 1$

Продолжение табл. 9

№ варианта	Функция и ее разложение в ряд	Область сходимости ряда
6	$ln(1+x)=\sum_{n=0}^{\infty}\left[(-1)^{n+1}\frac{x^n}{n}\right]$	$-1 < x \le +1$
7	$ln(1-x)=\sum_{n=0}^{\infty}\left[(-1)^{n}\frac{x^n}{n}\right]$	$-1 \le x < +1$
8	$ln\,x=\sum_{n=0}^{\infty}\left[(-1)^{n+1}\frac{(x-1)^n}{n}\right]$	$0 < x \le 2$
9	$ln\,x=2\cdot\sum_{n=1}^{\infty}\left[\frac{1}{2n-1}\left(\frac{x-1}{x+1}\right)^{2n-1}\right]$	$x > 0$
10	$ln\,x=\sum_{n=1}^{\infty}\left[\frac{1}{n}\left(\frac{x-1}{x}\right)^{n}\right]$	$x \ge \frac{1}{2}$
11	$shx=\sum_{n=0}^{\infty}\frac{x^{2n+1}}{(2n+1)!}$ *)	$-\infty < x < +\infty$
12	$chx=\sum_{n=0}^{\infty}\frac{x^{2n}}{(2n)!}$ **)	$-\infty < x < +\infty$
13	$thx=1+2\cdot\sum_{n=1}^{\infty}(-1)^{n}\cdot e^{-2nx}$ ***)	$x > 0$
14	$sin^2\,x=\sum_{n=1}^{\infty}\left[(-1)^{n+1}\cdot\frac{2^{2n-1}\cdot x^{2n}}{(2n)!}\right]$	$-\infty < x < +\infty$
15	$cos^2\,x=1-\sum_{n=1}^{\infty}\left[(-1)^{n+1}\cdot\frac{2^{2n-1}\cdot x^{2n}}{(2n)!}\right]$	$-\infty < x < +\infty$
16	$sin^3\,x=\frac{1}{4}\cdot\sum_{n=1}^{\infty}\left[(-1)^{n+1}\cdot\frac{3^{2n-1}-3}{(2n+1)!}\cdot x^{2n+1}\right]$	$-\infty < x < +\infty$

Окончание табл. 9

№ варианта	Функция и ее разложение в ряд	Область сходимости ряда
17	$cos^3 x = \frac{1}{4} \cdot \sum_{n=0}^{\infty} \left[(-1)^n \cdot \frac{3^{2n}+3}{(2n)!} \right] \cdot x^{2n}$	$-\infty < x < +\infty$
18	$ln\frac{1+x}{1-x} = 2 \cdot \sum_{n=1}^{\infty} \left[\frac{1}{2n-1} \cdot x^{2n-1} \right]$	$x^2<1$
19	$ln\frac{1+x}{1-x} = 2 \cdot \sum_{n=1}^{\infty} \left[\frac{1}{(2n-1) \cdot x^{2n-1}} \right]$	$x^2>1$
20	$ln\frac{x}{x-1} = \sum_{n=1}^{\infty} \frac{1}{n \cdot x^n}$	$x^2>1$
21	$ln\frac{x}{x-1} = \sum_{n=1}^{\infty} \frac{x^n}{n}$	$x^2<1$
22	$\frac{x}{x+1} = \sum_{n=1}^{\infty} (-1)^{n-1} \cdot x^{n-1}$	$-\infty < x < +\infty$, $x \neq -1$
23	$\frac{x}{(1+x)^2} = \sum_{n=1}^{\infty} (-1)^{n-1} \cdot n \cdot x^{n-1}$	$-\infty < x < +\infty$ $x \neq -1$
24	$\frac{x}{(1+x)^2} = \sum_{n=1}^{\infty} n \cdot x^n$	$x^2<1$
25	$\frac{\pi}{4} = 1 - 2 \cdot \sum_{n=1}^{\infty} \frac{1}{(4n-1) \cdot (4n+1)}$	-

---

*) – функция гиперболический синус $shx = \frac{e^x - e^{-x}}{2}$

**) – функция гиперболический косинус $chx = \frac{e^x + e^{-x}}{2}$

***) – функция гиперболический тангенс $thx = \frac{shx}{chx}$

Лабораторная работа

РЕКУРСИВНЫЕ АЛГОРИТМЫ

Основы теории

Слово ***рекурсия*** произошло от латинского *recursion* – ***возвращение***.

Алгоритм решения задачи называется ***рекурсивным***, если в его теле происходит возвращение к самому же себе. Рекурсивными могут быть только подпрограммы. Это вытекает из того, что подпрограммы позволяют дать любой последовательности действий (операторов) имя, с помощью которого можно эту последовательность действий вызывать.

Рекурсия тесно связана с соответствующим ей рекуррентным соотношением. Применительно к рекурсии рекуррентное соотношение следует рассматривать как структуру, состоящую из двух утверждений: ***базового и рекурсивного***.

Утверждение, определяющее начальные условия в рекуррентном соотношении, составляет ***базовое утверждение***, а утверждение, определяющее закон получения последующего элемента последовательности через значение её предыдущего элемента составляет ***рекурсивное утверждение***.

Например, в рекуррентном соотношении, определяющем вычисление *n!*

$$n! = \begin{cases} 1, & при\ n = 1 \quad - утверждение\ 1, \\ n \cdot (n-1)!, & при\ n > 1 \quad - утверждение\ 2 \end{cases} \qquad (1)$$

первое утверждение является базовым, второе – рекурсивным.

Это соотношение можно описать рекурсивной подпрограммой-функцией или подпрограммой-процедурой.

- ✓ Подпрограмма-функция:

```
Function Fact_V1 (n: integer): longint;  {n >= 1}
Begin
    if n = 1 then Fact_V1:= 1  {базовое утверждение}
          else Fact_V1:= n*Fact_V1 (n-1); {рекурсивный вызов}
End;
```

✓ Подпрограмма-процедура:

```
Procedure Fact_V2 (n: integer; var P: longint); {P – значение n! –
                                                 результат}
Begin
    if n = 1 then P:= 1 {базовое утверждение}
        else Fact_V2 (n-1, P);{рекурсивный вызов}
    P:= n*P;
End;
```

Сравнительный анализ объявленных подпрограмм и рекуррентного соотношения для вычисления *n!* позволяет сделать вывод:

► Если для некоторой задачи определено рекуррентное соотношение, то использование рекурсии является фактически отображением этого рекуррентного соотношения средствами алгоритмического языка.

Однако рекурсия имеет место быть и при реализации алгоритмов, для которых рекуррентные соотношения не так прозрачны, как для числовых последовательностей.

Например, при реализации *итерационного метода уточнения отделённого корня* трансцендентного или алгебраического уравнения, рекуррентность можно описать словесно:

1. Задать начальное приближение к корню
2. Вычислить значение корня по формуле

$$корень = \begin{cases} уточнённому\ значению, если\ точность достигнута, \\ последующему\ приближению, если\ точность не\ достигнута. \end{cases}$$

При реализации алгоритмов обработки одномерных или двумерных массивов рекуррентность проявляется при формировании индекса (номера) последующего элемента массива через индекс (номер) его предыдущего элемента, то есть при применении формулы:

$$i := i + hi,$$

где *hi* – шаг изменения индекса элемента массива.

Программы, в которых используются рекурсивные подпрограммы, отличаются простотой, наглядностью и компактностью текста.

Однако за эти качества приходится расплачиваться неэкономным использованием оперативной памяти, так как выполнение рекурсивных подпрограмм *требует значительно большего размера оперативной памяти во время выполнения*, чем не рекурсивных, реализующих решение той же задачи обычными циклическими алгоритмами. При каждом рекурсивном вызове для параметров подпрограммы и её локальных переменных выделяются новые ячейки в памяти, называемой ***стеком***.

Стек – это определённым образом организованная структура в составе оперативной памяти (область динамической памяти).

Данные в стек записываются и считываются из него по принципу *"Первым пришёл, последним ушёл"*.

Например, к моменту считывания из стека в нём находилось последовательность символов, составляющая слово МАГАЗИН. При считывании символов из стека и выводе их на экран полученная последовательность будет составлять слово НИЗАГАМ.

Максимальное число рекурсивных вызовов, которое происходит во время выполнения подпрограммы, называется ***глубиной рекурсии***.

Число рекурсивных вызовов в каждый конкретный момент времени называется ***текущим уровнем рекурсии***.

➢ Организация управления рекурсивным вызовом

Рекурсивный вызов не может быть бесконечным. Управляющей структурой в рекурсивных подпрограммах является условный оператор, построенный на основе рекуррентного соотношения:

if L then Q1 else Q2;

где *L* – условное выражение;

Q1, Q2 – операторы.

Условное выражение *L* и оператор *Q1* формируются на основе базового утверждения рекуррентного соотношения. Формирование оператора *Q2* отражает рекурсивное содержание второго утверждения рекуррентного соотношения.

- Схема выполнения действий в рекурсивных подпрограммах.
  - ✓ Формы рекурсивных подпрограмм

В рекурсивных алгоритмах можно выделить 3 группы операторов.

1. В первую группу *P1* объединим операторы, расположенные до условного оператора.
2. Ко второй группе *Rec* отнесём операторы, реализующие рекурсивное утверждение; операторы этой группы также можно представить тремя подгруппами:
   - в подгруппу *S1* объединим операторы, расположенные до рекурсивного вызова;
   - к подгруппе *RecB* отнесём оператор рекурсивного вызова;
   - операторы, расположенные после рекурсивного вызова, объединим в подгруппу *S2*.
3. В третью группу *P2* объединим операторы, расположенные после условного оператора.

Любой оператор, за исключением оператора *RecB*, в конкретной реализации рекурсивной подпрограммы может отсутствовать.

Выполнение рекурсивной подпрограммы осуществляется по схеме:

- *“Как бы в цикле”* – по ветви *“НЕТ”* базового условия выполняются операторы первой группы – *P1* и рекурсивной группы *Rec*. Значения параметров рекурсивной подпрограммы и локальных переменных при каждом рекурсивном вызове записываются в стек. Процесс выполнения действий на этом этапе принято называть ***рекурсивным спуском***.

- При выполнении базового условия выполняются операторы ветви *“ДА”* и над данными стека (данные считываются в последовательности сверху – вниз) опять *“как бы в цикле до полного опустошения стека”* автоматически начинают выполняться операторы третьей группы *P2*. Выполнение рекурсивной подпрограммы завершается, как только все данные из стека будут выбраны. Процесс выполнения действий на данном этапе принято называть ***рекурсивным возвратом.***

В подгруппах рекурсивной группы *Rec* операторы выполняются аналогичным образом.

Структура рекурсивных подпрограмм может принимать три разных формы:

1. Форма, в которой результат выполнения подпрограммы формируют операторы, расположенные до рекурсивного вызова (с формированием результата на рекурсивном спуске):

```
Begin
    …
    P1;
    if <условие> ... Rec;
    …
End;
```

2. Форма с формированием результата операторами, расположенными после рекурсивного вызова (с формированием результата на рекурсивном возврате):

```
Begin
    …
    if <условие> … Rec;
    P2;
    …
End;
```

3. Форма с формированием результата операторами, расположенными как до, так и после рекурсивного вызова (с формированием результата как на рекурсивном спуске, так и на рекурсивном возврате):

```
Begin
    …
    P1;
    if <условие> … Rec;
    P2;
    …
End;
```

или

```
Begin
    …
    begin
        S1;
        if <условие> RecB;
        S2;
    end;
    …
End;
```

Демонстрацию, подтверждающую описанную схему выполнения рекурсивной подпрограммы и формы, проведём на примерах вычисления факториала по функции *Fact_V3*, *Fact_V4* и процедуры *RacPak*, формирующей массив цифр, составляющих запись числа (распаковка числа).

Вычисление факториала с использованием функции *Fact_V3:*

```
Function Fact_V3 (n, k: integer; P: longint): longint;
                        {n – натуральное число, для которого
                        вычисляется n!}
                        {k – значение первого сомножителя}
                        {P – текущее значение факториала = i!}
```

```
Begin
    P:= P*k;            {Накопление факториала стоит до оператора рекурсивного вызова – форма 1}
    if k = n then Fact_V3:= P
        else Fact_V3:= Fact_V3 (n, k+1, P);  {При каждом вызове в стек будет записываться значение n, последующее значение k и текущее значение P}
End;
```

При вызове этой функции из основной программы необходимо положить значения параметров *k* и *P* равными единице.

Результаты трассировки значений параметров этой подпрограммы приведены в таблице 10.

Таблица 10

Трассировка значений параметров функции *Fact_V3* для *n=4*

Текущий уровень рекурсии	Рекурсивный спуск		Рекурсивный возврат
0	P:= 1; {P=1, k=1;}	Fact_V3 (4,1,1);	n! = 24
1	P:= 1*1; {P=1, k=2;}	Fact_V3 (4,2,1);	Fact_V3:= 24
2	P:= 1*2; {P=2, k=3;}	Fact_V3 (4,3,2);	Fact_V3:= 24
3	P:= 2*3; {P=6, k=4;}	Fact_V3 (4,4,6);	Fact_V3:= 24
4	P:= 6*4; {P=24}	Fact_V3:= 24;	Fact_V3:= 24

Вычисление факториала с использованием функции *Fact_V4:*

```
Function Fact_V4 (n: integer): longint;
Var P: longint;                 {P – текущее значение факториала}
Begin
    if n = 1 then P:= 1
        else P:= Fact_V4 (n-1);  {При каждом вызове в стек будет записываться значение n и текущее значение P}
    Fact_V4:= P*n;              {Накопление факториала стоит после оператора рекурсивного вызова – форма 2}
End;
```

Таблица 11 содержит результаты трассировки значений параметров и локальных переменных функции *Fact_V4.*

Таблица 11

Трассировка значений параметров и локальных переменных функции *Fact_V4* для *n=4*

Текущий уровень рекурсии	Рекурсивный спуск		Рекурсивный возврат
0	n=4	P:= Fact_V4 (4)	n!= 24
1	n=4;	P:= Fact_V4 (3)	Fact_V4:= 6*4; {Fact_V4=24}
2	n=3;	P:= Fact_V4 (2)	Fact_V4:= 2*3; {Fact_V4=6}
3	n=2;	P:= Fact_V4 (1)	Fact_V4:= 1*2; {Fact_V4=2}
4	n=1;	P:= 1	Fact_V4:= 1*1; {Fact_V4=1}

Распаковка числа с использованием процедуры *RacPak*:

```
Procedure RacPak (N, i: integer; var k: integer; var x: TM1_i); {N – исходное целое число}
                                                                {i – текущее значение номера элемента в массиве x}
                                                                {при вызове подпрограммы необходимо значение i положить равным единице}
                                                                {k –количество цифр в числе}
                                                                {x – имя цифрового массива}
Type TM1_i = array [1..10] of integer; k ≤ 10}
Var ct, d: integer;          {ct – целая часть,
                              d – дробная часть от деления числа на 10}
Begin
    ct:= N div 10;  d:= N mod 10; {операторы, формирующие результат, расположены как до, так и после рекурсивного вызова – форма 3}
    if ct = 0 then k:= i
              else RacPaк (ct, i+1, k, x);
    x[i]:= d;
End;
```

Результаты трассировки значений её параметров представлены в таблице 12.

Таблица 12

Трассировка значений параметров и локальных переменных процедуры *RacPak* для *n=4*

Текущий уровень рекурсии	Рекурсивный спуск		Рекурсивный возврат
0	n=9357;	i:= 1;	k=4; X{7, 5, 3, 9};
1	ct:= 935; d:= 7;	i:= 1; RacPak (9357, 2, k, x);	X[1]:= 7;
2	ct:= 93; d:= 5;	i:= 2; RacPak (935, 3, k, x);	X[2]:= 5;
3	ct:= 9; d:= 3	i:= 3; RacPak (93, 4, k, x);	X[3]:= 3;
4	ct:= 0; d:= 9	i:= 4; k:= 4;	X[4]:= 9;

Всё выше сказанное позволяет сделать вывод:

► При разработке рекурсивной подпрограммы необходимо свести её алгоритм к одной из рассмотренных форм.

➢ Области применения рекурсивных алгоритмов

В принципе любой циклический алгоритм может быть реализован как рекурсивный. Ответ на вопрос, какой реализации отдать предпочтение, зависит от квалификации программиста. Однако при разработке алгоритмов построения графических объектов, например, фракталов, обработки текстовых данных, сортировки массивов, обработке древовидных структур данных и решении задач во многих других областях применение рекурсий неоспоримо.

Контроль входных знаний

1. Можно ли алгоритм ввода в память элементов двумерного массива реализовать как рекурсивный?
2. Какую функцию в рекуррентном соотношении или рекуррентном описании вычислительного процесса выполняет базовое утверждение?
3. При глубине рекурсии, равной *5*, сколько раз при пошаговом выполнении подпрограммы выполнится

группа операторов *P2*, размещённая после рекурсивной группы *Rec*?

4. По какому принципу осуществляется запись данных в стек и считывание данных из стека? Продемонстрируйте действие этого принципа на примере записи и считывания элементов одномерного массива.
5. Почему реализация рекурсивных алгоритмов возможна только подпрограммами?

Задания для выполнения

1. Описать в виде формулы или словесно рекурсивность задачи Вашего варианта (см. табл. 13).
2. Разработать и отладить программу.
3. Разработать таблицу трассировки параметров рекурсивной подпрограммы и заполнить её.
4. Разработать блок-схему решения задачи циклическим алгоритмом.

Таблица 13

Варианты заданий

№ варианта	Дополнительные условия к задаче	Вид подпрограммы	Форма
Из символов строки сформировать одномерный массив и вывести значения его элементов:			
1	в прямом порядке	процедура	1
2	в прямом порядке	процедура	2
3	в обратном порядке	процедура	1
4	в обратном порядке	процедура	2
Реализовать алгоритм уточнения отделённого корня алгебраического или трансцендентного уравнения			
5	методом половинного деления	процедура	любая
6	методом половинного деления	функция	любая
7	методом итераций	процедура	любая
8	методом итераций	функция	любая

Продолжение табл. 13

№ варианта	Дополнительные условия к задаче	Вид подпрограммы	Форма
Определить в строке:			
9	количество появлений заданного символа	функция	любая
10	позицию появления заданного символа	процедура	любая
Алгоритм Евклида вычисления наибольшего общего делителя двух чисел *a* и *b* заключается в том, что на каждой итерации в случае неравенства этих чисел, большее число заменяется разностью большего и меньшего. Процесс итераций заканчивается, когда числа окажутся равными.			
11	реализовать алгоритм Евклида	функция	1
12	реализовать алгоритм Евклида	процедура	2
13	реализовать алгоритм Евклида	функция	3
Реализовать алгоритм уточнения экстремума унимодальной функции			
14	методом половинного деления	процедура	любая
15	методом половинного деления	функция	любая
16	методом золотого сечения	процедура	любая
17	методом золотого сечения	функция	любая
Реализовать алгоритм вычисления значений функции $y = f(x)$ для аргумента x, изменяющегося на интервале $[a, b]$ с шагом h.			
18	значения *x* и *y* представить массивами	процедура	1
19	значения *x* и *y* представить массивами	процедура	2
В одномерном массиве определить:			
20	количество положительных элементов	процедура	2
21	сумму отрицательных элементов	функция	1
22	произведение элементов, принадлежащих интервалу $[a, b]$	процедура	2

Окончание табл. 13

№ варианта	Дополнительные условия к задаче	Вид подпрограммы	Форма
23	количество элементов, не принадлежащих интервалу *[a, b]*	функция	1
24	номер первого отрицательного элемента	процедура	любая
25	количество чётных элементов	функция	2

Лабораторная работа

АЛГОРИТМЫ НА МНОЖЕСТВАХ

Основы теории

Понятие ***множество*** является одним из основных *в современной математике* и трактуется как неупорядоченная совокупность неповторяющихся объектов. *В программировании* – это ограниченный упорядоченный набор различных элементов одного (базового) типа. ***Базовый тип*** – это совокупность значений, из которых могут быть образованы множества. В качестве базового может быть использован любой тип, кроме вещественного.

✓ Объявление базовых типов

```
Type T_Day = (Monday, Tuesday, Wednesday, Thursday, Friday,
             Saturday, Sunday );        {перечислимый тип}
  T_Symbol = char;                      {тип символов ASCII-кода}
  T_Digits = 1..100;                    {диапазонный числовой тип}
  T_Letter = 'А'..'Я';                  {диапазонный тип прописных
                                         букв русского алфавита}
```

✓ Объявления множеств

Множества, используемые в программе, могут быть объявлены либо в разделе объявления типов:

```
Type <имя типа> = Set of <тип элементов>;
Var <имя множества> : <имя типа>;
```

либо в разделе описания переменных:

Var <имя множества> : Set of <тип элементов>;

Например,

Type T_Symbol = Set of char;
Var Symbol: T_Symbol;

или

Var Symbol: S*et of char;*

Всего во множестве может быть не более 256 различных элементов. Множество, не содержащее ни одного элемента, называется ***пустым***.

- Формирование (конструирование) множеств

В программе элементы множества задаются в квадратных скобках, через запятую. Если элементы идут подряд, то можно использовать диапазон.

Элементы множества могут задаваться константами, переменными и выражениями базового типа, например,

[ ] – пустое множество;
[6] – множество из одного элемента;
[2, 5, 7, 9] – множество из нескольких целых чисел;
[1, k] – множество, состоящее из целого числа 1 и текущего значения переменной *k*;
*[k..3*k]* – множество целых чисел от значения переменной *k* до результата выражения *3*k*;
[*Monday, Tuesday, Wednesday*] – множество, состоящее из трёх элементов перечислимого типа.

В программах часто используются множества-константы:

Const <имя константы> = [<её элементы>];

Например,

Const YesOrNo = [‘Y’, ‘y’, ‘N’, ‘n’]; {множество-константа, содержащая четыре символа}

CharDig = [‘0’..‘9’]; {множество-константа, содержащая *десять символов* – цифр}

Digits = [0..9]; {множество-константа, содержащая *десять цифр*}

➢ Операции над множествами

Объединением двух множеств называется множество элементов, принадлежащих обоим множествам. Знак операции “+”.

Например,

1. A1:= [2, 5..8]; A2:= [4, 5, 7];
   A1+A2 = [2, 4..8];
2. [‘A’, ‘B’]+[‘C’, ‘c’] = [‘A’, ‘B’, ‘C’, ‘c’];
3. B1 = [‘a’..‘z’]; B2 = [‘A’];
   B1+B2 = [‘A’, ‘a’..‘z’];

Пересечением двух множеств называется множество элементов, содержащее только общие элементы. Знак операции “*”.

Например,

1. [‘a’, ‘k’] * [‘A’, ‘K’] = [ ];
2. A1:= [2, 5..8]; A2:= [4, 5, 7];
   A1*A2 = [5, 7];

Разностью двух множеств называется множество, содержащее те элементы первого множества, которые не являются элементами второго множества. Знак операции “–”.

Например,

1. [3, 5, 6] – [2, 4] = [3, 5, 6];
2. A1:= [2, 5..8]; A2:= [4, 5, 7];
   A1 – A2 = [2, 6, 8];
3. A1 = [‘A’..‘Z’]; A1 = A1 – [‘A’];
   A1 = [‘B’..‘Z’]; – из множества A1 исключили элемент ‘A’.

Операция определения ***принадлежности элемента множеству.*** Эта логическая операция обозначается служебным словом ***in*** и имеет результат *true*, если элемент принадлежит множеству, и *false* в противном случае.

Например,

- выражение *8 in [5..12]* имеет значение *true*, так как *8* входит в диапазон чисел *от 5 до 12*;
- выражение ‘*ё*’ *in [*‘*a*’..‘*z*’*]* имеет значение *false*, так как буквы ‘*ё*’ нет в латинском алфавите.

➢ Сравнение множеств

Для сравнения множеств используются *операции отношения*, принимающие значение *true* или *false*, в зависимости от результата сравнения:

= – равенство (совпадение) двух множеств;
< > – неравенство двух множеств;
<=, < – проверка на вхождение первого множества во второе;
>=, > – проверка на вхождение второго множества в первое.

Например,

[3..7] = [3, 4, 5..7];
[‘a’, ‘b’] < > [‘a’, ‘б’];
[‘л’, ‘м’, ‘н’] < [‘a’..‘я’];
[25..50] > [30..35].

➢ Размещение множеств в памяти

Множество любого типа представляется в памяти ЭВМ в виде последовательности нулей и единиц в соседних битах. При этом:

- максимальный размер памяти для размещения множества составляет 256 бит, т. е. 32 байта; байты множества располагаются в порядке возрастания адресов памяти;
- множества с базовыми типами *byte* и *char* занимают 32 байта;

- множества с использованием интервальных типов могут занимать меньший размер памяти, но он всегда равен целому числу байт;
- нумерация битов в отведённой последовательности байт строго фиксирована и ведётся справа налево (0..255). Некоторому элементу *x* соответствует бит с порядковым номером *Ord (x)*. Таким образом, значение бита является индикатором наличия того или иного элемента во множестве (1 – элемент присутствует, 0 – отсутствует);
- начало нумерации битов в отведённой памяти всегда кратно восьми и выбирается по максимуму, а общее количество байтов выбирается по минимуму с учётом возможности размещения всех элементов базового типа.

Например, для типов:

- *Type T1 = Set of char;*

и

T2 = Set of byte;

размер памяти составляет по 32 байта, то есть

SizeOf (T1) = 32 байта;
SizeOf (T2) = 32 байта;

- *Type T3 = Set of 5..13;*

и

T4 = Set of 9..20;

размер памяти составляет по 2 байта, то есть

SizeOf (T3) = 2 байта;
SizeOf (T4) = 2 байта;

Распределение битов для множеств:

- *m2* = [7..12, 15] типа *T2* имеет вид (см. табл.14).

Таблица 14

Распределение битов для множества *m2*

Байт	2								1							
Номер бита	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Значение	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0

Значения битов с 16 по 255 равны 0. Включение во множество *m2* новых элементов из диапазонов 0..6 и 16..255 вполне допустимо.

- *m4* = [12, 14, 16, 19] типа *T4* распределение битов будет иметь вид (см. табл. 15).

Таблица 15

Распределение битов для множества *m4*

Байт	3								2							
Номер бита	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8
Значение	0	0	0	0	1	0	0	1	0	1	0	1	0	0	0	0

Попытка присоединения к данному множеству любого элемента из диапазона номеров битов в пределах этих двух байтов будет успешной, вопреки объявленному типу. Числа, выходящие за пределы выделенной области (числа от 0 до 7 и от 24 до 255) в состав множества включить не удастся. На эти две ситуации система программирования никак не среагирует.

➢ Вспомогательный материал

При разработке алгоритмов на множествах часто используются следующие функции:

Function Eoln – использует стандартную файловую переменную *Input: Text*, связанную с клавиатурой и возвращает зна-

чение *true*, если достигнут конец строки. При этом конец строки фиксируется нажатием клавиши Enter;

Function Low (*m*: <тип базового множества>) – возвращает значение наименьшего элемента базового множества;

Function High (*m*: <тип базового множества>) – возвращает значение наибольшего элемента базового множества.

Демонстрационные примеры

Пример 1. Подпрограмма ввода элементов множества:

- набираем строку элементов множества *m*, заканчивающуюся нажатием клавиши Enter;
- начальное подмножество *m* задаём пустым, т.е. *m:= [ ]*;
- в цикле присоединяем к множеству *m* каждый элемент набранной строки.

```
Procedure ReadSet (var m: Tm); {type Tm = Set of Tb; тип Tb – базовый тип}
Var e: Tb;
Begin
    writeln ('Наберите строку элементов множества,
            заканчивающуюся нажатием клавиши Enter');
    m:= [ ];
    while not Eoln do
                    begin
                        read (e); m:= m + [e];
                    end;
End;
```

Пример 2. Подпрограмма вывода элементов множества:

- просматриваем все элементы базового множества, начиная с наименьшего *Low (Tb)*, и заканчивая наибольшим элементом *High (Tb);*
- проверяем принадлежность каждого из них результирующему множеству;
- в случае положительного исхода проверки выводим соответствующий элемент на экран.

```
Procedure WriteSet (m: Tm);
Var e: Tb;
Begin
      For e:= Low (Tb) to High (Tb) do
                  if e in m then Write (e, ‘ ’);
      writeln;
End;
```

Пример 3. Дано натуральное число *n*. Сформировать множество из цифр, не входящих в десятичную запись этого числа.

```
Procedure Digit (n: LongInt; var m: Tm);
Begin
      m:= [0..9];
      while n <> 0 do
                        begin
                              m:= m – [n mod 10];
                              n:= n div 10;
                        end;
End;
```

Пример 4. Нахождение минимального элемента множества.

В алгоритме использован тот факт, что элементы любого множества всегда упорядочены по возрастанию.

```
Function Min_Set (m: Tm): Tb;
Var e: Tb;
Begin
      e:= Low (Tb ); {начальное значение минимального элемен-
                        та}
      while not (e in m) do e:= Succ (e);
      Min_Set:= e;
End;
```

Пример 5. Дано натуральное число *n.* Вывести в порядке возрастания цифры, не входящие в десятичную запись этого числа.

```
Program P1;
Uses Crt;
Type  Tb = 0..9; {Tb – базовый перечислимый тип}
      Tm = Set of Tb;
Var n: LongInt;
    m: Tm;
BEGIN
        ClrScr;
        write ('Введите число = '); readln (n);
        Digit (n, m);
        WriteSet (m);
        readln;
END.
```

Контроль входных знаний

1. Вычислить значения отношений или указать, что они ошибочны:
   а) [5] <> [5, 5, 5];
   б) ['a', 'c'] = ['c', 'a'];
   в) [5, 6, 7, 8] = [5..8];
   г) [2, 3, 5, 7] <= [1..9];
   д) [4, 7..9] <= [3..6, 9];
   е) [ ] <= ['0'..'9'];
   ж) 'ф' in ['и'..'я'];
   з) 125 = [125].
2. Вычислить значения выражений:
   а) [1..5] + [3..9];
   б) [2..6] * [5..8];
   в) [1..6] – [4..7];
   г) [ ] * [3..8];
   д) ['a', 'd', 'n'] + ['5', '7'];
   е) ['5', '8'] * ['1', '9'].
3. Может ли тип *integer* быть взят в качестве базового для множества?

4. Дан тип *type Tm = Set of 13..20*.
   - Сколько байтов памяти будет отведено под данный тип?

- Какой вид будет иметь распределение битов для множества [16, 18, 20]?
- Как отреагирует система программирования на попытку присоединить к множеству следующие элементы: 6, 15, 27?
- Равны ли множества:
  [1, 2, 3, 1] и [3, 2, 1];
  [‘3’, ‘7’, ‘4’] и [‘3’, ‘4’, ‘7’];
  [3, 7, 4] и [3, 4, 7];
  [‘a’, ‘b’, ‘B’] и [‘b’, ‘B’, ‘a’]?

5. Можно ли построить множество из элементов:
   5, ‘a’, 9, ‘3’?

Задания для выполнения

Представить алгоритмы (в виде блок-схем) и решить задачи Вашего варианта (см. табл. 16).

Таблица 16

Варианты заданий

№ варианта	Задание
1	1. В строке-формуле подсчитать общее количество знаков ‘+’, ‘-’, ‘*’, ‘/’. Вывести на печать в порядке возрастания символы этой строки и их ASCII-коды. 2. Одномерный числовой массив содержит *N* произвольных целых чисел. Построить множество, содержащее элементы этого массива, принадлежащие интервалу *[a, b]*.
2	1. Одномерный массив содержит *N* произвольных целых чисел. Построить множество, содержащее чётные числа из этого массива. 2. Вывести на печать в алфавитном порядке буквы, входящие в Вашу фамилию, и их ASCII-коды.
3	1. Дана строка произвольного текста. Напечатать в порядке возрастания буквы, входящие в него более двух раз, и вывести их ASCII-коды. 2. Дано число. Определить количество различных цифр, содержащихся в его десятичной записи.

Продолжение табл. 16

№ варианта	Задание
4	1. Дана строка произвольного текста. Построить множество, содержащее гласные буквы русского языка, входящие в этот текст. Вывести на печать в алфавитном порядке элементы этого множества и их ASCII-коды. 2. Вывести в порядке возрастания цифры, входящие в десятичную запись натурального числа N.
5	1. Дана строка произвольного текста. Напечатать в алфавитном порядке буквы текста, входящие в него не менее двух раз, и их ASCII-коды. 2. Даны два одномерных массива, содержащие целые числа из интервала [0..255]. Определить числа, входящие в оба массива.
6	1. Одномерный массив содержит N произвольных целых чисел. Построить множество, содержащее нечётные числа из этого массива. 2. Вывести на печать в алфавитном порядке буквы, входящие в Вашу фамилию, но не входящие в Ваше имя, и их ASCII-коды.
7	1. Дана строка произвольного текста. Построить множество, содержащее согласные буквы русского языка, входящие в этот текст. Вывести на печать в алфавитном порядке элементы этого множества и их ASCII-коды. 2. Вывести в порядке возрастания цифры, не входящие в десятичную запись натурального числа N.
8	1. Дана строка произвольного текста. Построить множество, содержащее прописные гласные буквы русского языка, входящие в этот текст. Вывести на печать в алфавитном порядке элементы этого множества и их ASCII-коды. 2. Вывести в порядке убывания цифры, входящие в десятичную запись натурального числа N.
9	1. Даны два одномерных массива, содержащие целые числа из интервала [0..255]. Определить какой массив содержит больше чётных чисел. 2. Дана строка произвольного текста. Можно ли из букв, входящих в неё, сформировать заданное слово? Вывести в алфавитном порядке буквы этого слова и их ASCII-коды.

Продолжение табл. 16

№ варианта	Задание
10	1. Даны две строки произвольного текста. Определить какая из этих строк содержит больше гласных букв. Вывести на печать в алфавитном порядке гласные буквы, входящие в Ваше полное имя, и их ASCII-коды. 2. Одномерный числовой массив содержит *N* произвольных целых чисел. Построить множество, содержащее элементы этого массива, кратные некоторому заданному числу.
11	1. Дана строка произвольного текста. Построить множество, содержащее символы-цифры, входящие в этот текст. Вывести на печать в алфавитном порядке элементы этого множества и их ASCII-коды. 2. Построить множество из элементов одномерного массива, стоящих на нечётных местах.
12	1. Вычислить сумму цифр, входящих в десятичную запись натурального числа *N*. 2. Дана строка произвольного текста. Вывести те знаки препинания, которых нет в строке, и их ASCII-коды.
13	1. Даны два числа. Проверить наличие в них одинаковых цифр. 2. Дана строка. Вывести те знаки препинания, которые встречаются в ней только один раз и их ASCII-коды.
14	1. Даны два числа. Определить цифры, входящие в запись как первого, так и второго числа. 2 Строка состоит из трёх слов. Определить, какое из слов содержит наибольшее количество гласных букв. Вывести ASCII-коды всех символов этого слова.
15	1. Даны два слова. Вычеркнуть из первого слова те буквы, которые не встречаются во втором слове, и вывести их ASCII-коды 2. Дано многозначное число. Определить множество цифр, которые встречаются в его записи более одного раза.
16	1. Даны три арифметических выражения. Определить количественное соответствие в них арифметических скобок: ( ), [ ], { }. Вывести ASCII-коды этих скобок. 2. В строке, содержащей три числа, есть число, записанное в научной форме. Найти сумму этих чисел.

Продолжение табл. 16

№ варианта	Задание
17	1. Даны три алгебраических выражения: $a + b * c;$ $a/b - c;$ $a + b - c.$ Построить множество знаков арифметических действий, содержащихся в них. Вывести на печать элементы сформированного множества и их ASCII-коды. По совокупности вводимых с клавиатуры знаков арифметических действий выбрать выражение и вычислить его значение при заданных значениях переменных *a*, *b* и *c*. 2. Даны два числа. Определить цифры, не входящие в запись этих чисел.
18	1. Дано целое многозначное число. Вывести чётные цифры, содержащиеся в его записи. 2. Текст содержит повествовательные, вопросительные и восклицательные предложения. Определить в нём количество предложений. Вывести в алфавитном порядке начальные буквы каждого предложения и их ASCII-коды.
19	1. Дана строка произвольного текста. Напечатать в алфавитном порядке буквы текста, входящие в него не более двух раз, и их ASCII-коды. 2. Дано число. Верно ли, что оно записано в научной форме? Вывести нецифровые элементы записи числа в научной форме и их ASCII-коды.
20	1. Даны два многозначных целых числа. Верно ли, что для их записи был использован один и тот же набор цифр? 2. Строка состоит из трёх слов. Определить, какое из слов содержит наибольшее количество согласных букв. Вывести на печать ASCII-коды этого слова.
21	1. Дана строка произвольного текста. Вывести те знаки препинания, которые встречаются в ней более одного раза и их ASCII-коды. 2. Даны три числа. Найти цифры, которые есть в записи каждого из этих чисел.

Окончание табл. 16

№ варианта	Задание
22	1. Текст содержит повествовательные, вопросительные и восклицательные предложения. Определить в нём количество предложений. Вывести ASCII-коды первого слова текста. 2. Дано число. Верно ли, что в нём есть рядом стоящие нечётные цифры?
23	1. Дан произвольный текст. Определить присутствующее в нём числовое арифметическое выражение и вывести ASCII-коды его арифметических действий. 2. Даны два числа. Определить, в каком из них больше чётных цифр.
24	1. Дан текст – праздничное поздравление. Определить имена поздравляемых. Вывести ASCII-коды букв первого по списку имени. 2. Дано число. Верно ли, что в нём присутствует комбинация двух рядом стоящих цифр, сумма которых равна заданному числу?
25	1. Дана строка. Определить первое вхождение в неё звонкой согласной. Вывести в алфавитном порядке ASCII-коды всех звонких согласных русского языка. 2. Дано число. Вычислить сумму наибольшей чётной и наименьшей нечётной цифр, входящих в его запись.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Иванова Г.С. Основы программирования: Учебник для вузов. – М.; Изд-во МГТУ им Н.Э. Баумана, 2001. – 392 с., ил. (Сер. Информатика в техническом университете).
2. Керниган Брайан В., Пайк Роб. Практика программирования/ Пер. с англ. – СПб.: Невский Диалект, 2001. – 381 с.: ил.
3. Ливитин, Ананий В. Алгоритмы: введение в разработку и анализ. : Пер. с англ. – М.: Издательский дом ”Вильямс”, 2006. – 576 с.: ил. – Парал. тит англ.
4. Моргун А.Н. Справочник по Turbo Pascal для студентов. – М.: Издательский дом ”Вильямс”, 2006. – 608 с. : ил.
5. Сборник задач по программированию: учеб. пособие / А.И. Мишенин. – М.: Финансы и статистика; ИНФРА-М, 2009.– 224 с.
6. Хусаинов Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си (+CD): Учебное пособие. – Финансы и статистика, 2004. – 464 с.: ил.

ПРИЛОЖЕНИЕ

ASCII – Американский стандартный код информационного обмена (Модифицированная альтернативная кодировка ГОСТа)

001		002		003		004		005	
006		007		008		009		010	
011		012		013		014		015	
016		017		018		019		020	
021		022		023		024		025	
026		027		028		029		030	
031		032		033	!	034	"	035	#
036	$	037	%	038	&	039	'	040	(
041	)	042	*	043	+	044	,	045	-
046	.	047	/	048	0	049	1	050	2
051	3	052	4	053	5	054	6	055	7
056	8	057	9	058	:	059	;	060	<
061	=	062	>	063	?	064	@	065	A
066	B	067	C	068	D	069	E	070	F
071	G	072	H	073	I	074	J	075	K
076	L	077	M	078	N	079	O	080	P
081	Q	082	R	083	S	084	T	085	U
086	V	087	W	088	X	089	Y	090	Z
091	[	092	\	093	]	094	^	095	_
096	`	097	a	098	b	099	c	100	d
101	e	102	f	103	g	104	h	105	i
106	j	107	k	108	l	109	m	110	n
111	o	112	p	113	q	114	r	115	s
116	t	117	u	118	v	119	w	120	x
121	y	122	z	123	{	124	¦	125	}
126	~	127	⌂	128	А	129	Б	130	В

Продолжение приложения

131	Г	132	Д	133	Е	134	Ж	135	З
136	И	137	Й	138	К	139	Л	140	М
141	Н	142	О	143	П	144	Р	145	С
146	Т	147	У	148	Ф	149	Х	150	Ц
151	Ч	152	Ш	153	Щ	154	Ъ	155	Ы
156	Ь	157	Э	158	Ю	159	Я	160	а
161	б	162	в	163	г	164	д	165	е
166	ж	167	з	168	и	169	й	170	к
171	л	172	м	173	н	174	о	175	п
176	▒	177	▒	178	▓	179	┃	180	┤
181	╡	182	╢	183	╖	184	╕	185	╣
186	║	187	╗	188	╝	189	╜	190	╛
191	┐	192	└	193	┴	194	┬	195	├
196	─	197	┼	198	╞	199	╟	200	╚
201	╔	202	╩	203	╦	204	╠	205	═
206	╬	207	╧	208	╨	209	╤	210	╥
211	╙	212	╘	213	╒	214	╓	215	╫
216	╪	217	┘	218	┌	219	▉	220	▄
221	▌	222	▐	223	▀	224	р	225	с
226	т	227	у	228	ф	229	х	230	ц
231	ч	232	ш	233	щ	234	ъ	235	ы
236	ь	237	э	238	ю	239	я	240	Ё
241	ё	242	Є	243	є	244	Ї	245	ї
246	Ў	247	ў	248	°	249	•	250	·
251	√	252	№	253	¤	254	▪	255	

Коды 0..31 и код 127 относятся к управляющим и на экране не отображаются. Код 32 соответствует пробелу.

УЧЕБНОЕ ИЗДАНИЕ

Станевко Галина Ивановна
Колесникова Татьяна Геннадьевна
Давыденко Вероника Анатольевна

Программирование и основы алгоритмизации: лабораторный практикум

Лабораторный практикум

Для студентов вузов

Зав. редакцией: А.С. Обвинцева
Редактор: Н.В. Шишкина
Технические редакторы: Т.В. Васильева, Е.К. Матвеева
Художественный редактор: Л. П. Токарева

ЛР № _____ от _____
Подписано в печать _____ Формат $60\times84^{1/16}$
Бумага типографская. Гарнитура Times
Уч.-изд. л. ____. Тираж _____ экз.
Заказ № ____

Оригинал-макет изготовлен в редакционно-издательском центре Кемеровского технологического института пищевой промышленности
650056, г. Кемерово, б-р Строителей, 47

ПЛД № _____ от _____
Отпечатано в лаборатории множительной техники
Кемеровского технологического института пищевой промышленности
650056, г. Кемерово, ул. Красноармейская, 52