Министерство образования и науки Украины

Севастопольский национальный технический университет

Программирование на языке Паскаль:

методические указания к лабораторным работам по дисциплине

“Основы программирования и алгоритмические языки”

для студентов дневной и заочной формы обучения

направления 6.0804 –“Компьютерные науки”

Часть 2

Севастополь
2006

УДК 004.42 (075.8)

Программирование на языке Паскаль: методические указания к лабораторным работам по дисциплине “Основы программирования и алгоритмические языки” для студентов дневной и заочной формы обучения направления 6.0804 – “Компьютерные науки”, часть 2/ Сост. **В.Н. Бондарев.**— Севастополь: Изд-во СевНТУ, 2006. — 67с.

Цель указаний: оказать помощь студентам направления «Компьютерные науки» при выполнении лабораторных работ по дисциплине «Основы программирования и алгоритмические языки» в интегрированной среде программирования Turbo Pascal (Borland Pascal).

Методические указания составлены в соответствии с требованиями программы дисциплины "Основы программирования и алгоритмические языки" для студентов направления 6.0804 и утверждены на заседании кафедры информационных систем протокол №12 от 4 июля 2005 года.

Допущено учебно-методическим центром СевНТУ в качестве методических указаний.

Рецензент: Кожаев Е.А., к.т.н., доцент кафедры кибернетики и вычислительной техники

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цели и задачи лабораторных работ

Основная цель выполнения настоящих лабораторных работ – получение практических навыков создания и отладки программ на языке Паскаль, использующих сложные типы данных. В результате выполнения лабораторных работ студенты должны углубить знания основных теоретических положений дисциплины "Основы программирования и алгоритмические языки", решая практические задачи на ЭВМ.

Студенты должны получить практические навыки работы в интегрированной среде программирования Turbo (Borland) Pascal, навыки разработки программ, использующих текстовые и типизированные файлы, записи, списковые и древовидные структуры данных, динамическую память, объекты и модули.

Выбор вариантов и график выполнения лабораторных работ

При изучении раздела "Программирование на языке Паскаль" (часть 2) выполняется пять лабораторных работ. Все работы выполняются в течение 4 академических часов.

В лабораторных работах студент решает заданную индивидуальным вариантом задачу. Варианты заданий приведены к каждой лабораторной работе и уточняются преподавателем. Номер варианта выбирается в соответствии с номером зачетной книжки студента. Вариант вычисляется как остаток от деления числа, соответствующего двум последним цифрам в номере зачетной книжки, на 30. Например, две последние цифры зачетки 46. Тогда остаток деления 46 на 30 будет равен 16. Следовательно, студент выбирает вариант 16.

Лабораторная работа выполняется в два этапа. На первом этапе, выполняемом самостоятельно, студент должен подготовить следующее:

- проработать по конспекту и рекомендованной литературе, приведенной в конце настоящих методических указаний, основные теоретические положения лабораторной работы и ответить на контрольные вопросы;
- разработать алгоритм решения задачи и составить его структурную схему;
- описать структурную схему алгоритма;
- написать программу на языке Паскаль и описать ее;
- оформить результаты первого этапа в виде заготовки отчета по лабораторной работе (**студенты-заочники** первый этап выполнения лабораторных работ оформляют в виде контрольной работы, которую сдают на кафедру до начала зачетно-экзаменационной сессии).

На втором этапе, выполняемом в лабораториях кафедры, студент должен:

- отладить программу;
- разработать и выполнить тестовый пример.

По завершении лабораторной работы готовится и защищается отчёт по лабораторной работе.

Студенты дневного отделения должны выполнять и защищать работы **строго по графику**. График выполнения лабораторных работ:

- лабораторная работа 7 – 12-ая неделя осеннего семестра;
- лабораторная работа 8 – 14-ая неделя осеннего семестра;
- лабораторная работа 9 – 2-ая неделя весеннего семестра;
- лабораторная работа 10 – 3-ая неделя весеннего семестра;
- лабораторная работа 11 – 4-ая неделя весеннего семестра;

Требования к оформлению отчета

Отчёт по лабораторной работе оформляется каждым студентом индивидуально. В общем случае отчёт должен включать: название и номер лабораторной работы; цель работы; вариант задания и постановку задачи; разработку и описание алгоритма решения задачи; описание средств языка Паскаль, привлекаемых для решения задачи; разработку и описание программы; выводы по работе; приложения. Содержание отчета указано в методических указаниях к каждой лабораторной работе.

Постановка задачи представляет изложение содержания задачи, цели её решения. На этом этапе должны быть полностью определены исходные данные, перечислены задачи по преобразованию и обработке входных данных, дана характеристика результатов, описаны выходные данные.

Разработка алгоритмов решения задачи предполагает математическую формулировку задачи и описание решения задачи на уровне схем алгоритмов. Схемы алгоритмов должны быть выполнены в соответствии с требованиями ГОСТ, сопровождаться описанием и таблицей используемых в алгоритме имён с указанием их смыслового значения.

Описание средств языка содержит общие сведения об управляющих конструкциях и декларациях языка, используемых в лабораторной работе.

Разработка и описание программы включает описание вычислительного процесса на языке Паскаль. Кроме текста программы, в отчёте представляется её пооператорное описание, даётся характеристика конструкций языка, обеспечивающих решение задачи конкретной лабораторной работы.

В приложении приводится текст программы, распечатки, полученные во всех отладочных прогонах программы с анализом ошибок, результаты решений тестового примера.

ЛАБОРАТОРНАЯ РАБОТА №7
ПРОГРАММИРОВАНИЕ ОПЕРАЦИЙ НАД СТРОКАМИ И ТЕКСТОВЫМИ ФАЙЛАМИ

1. Цель работы

Изучение основных операций над строками и файлами, программирование операций обработки строк текстовых файлов, исследование свойств файловых переменных.

2. Варианты заданий

1. Написать программу, которая считывает из текстового файла три предложения и выводит их в обратном порядке.
2. Написать программу, которая считывает текст из файла и выводит на экран только предложения, содержащие введенное с клавиатуры слово.
3. Написать программу, которая считывает текст из файла и выводит на экран только строки, содержащие двузначные числа.
4. Написать программу, которая считывает английский текст из файла и выводит на экран слова, начинающиеся с гласных букв.
5. Написать программу, которая считывает текст из файла и выводит его на экран, меняя местами каждые два соседних слова.
6. Написать программу, которая считывает текст из файла и выводит на экран только предложения, не содержащие запятых.
7. Написать программу, которая считывает текст из файла и определяет, сколько в нем слов, состоящих не более чем из четырех букв.
8. Написать программу, которая считывает текст из файла и выводит на экран только цитаты, то есть предложения, заключенные в кавычки.
9. Написать программу, которая считывает текст из файла и выводит на экран только предложения, состоящие из заданного количества слов.
10. Написать программу, которая считывает английский текст из файла и выводит на экран слова текста, начинающиеся и оканчивающиеся на гласные буквы.
11. Написать программу, которая считывает текст из файла и выводит на экран только строки, не содержащие двузначных чисел.
12. Написать программу, которая считывает текст из файла и выводит на экран только предложения, начинающиеся с тире, перед которым могут находиться только пробельные символы.
13. Написать программу, которая считывает английский текст из файла и выводит его на экран, заменив каждую первую букву слов, начинающихся с гласной буквы, на прописную.
14. Написать программу, которая считывает текст из файла и выводит его на экран, заменив цифры от 0 до 9 на слова «ноль», «один»...«девять», начиная каждое предложение с новой строки.

15. Написать программу, которая считывает текст из файла, находит самое длинное слово и определяет, сколько раз оно встретилось в тексте.
16. Написать программу, которая считывает текст из файла и выводит на экран сначала вопросительные, а затем восклицательные предложения.
17. Написать программу, которая считывает текст из файла и выводит его на экран, после каждого предложения добавляя, сколько раз встретилось в нем введенное с клавиатуры слово.
18. Написать программу, которая считывает текст из файла и выводит на экран все его предложения в обратном порядке.
19. Написать программу, которая считывает текст из файла и выводит на экран сначала предложения, начинающиеся с однобуквенных слов, а затем все остальные.
20. Написать программу, которая считывает текст из файла и вычисляет количество открытых и закрытых скобок. Дописать вычисленные значения в конец каждой строки. Результаты записать в новый файл.
21. Написать программу, которая считывает текст из файла и выводит на экран предложения, содержащие максимальное количество знаков пунктуации.
22. Даны два текстовых файла. Удалить из этих файлов строки, которые имеют одинаковые номера, но сами не являются одинаковыми. Результаты записать в новый файл.
23. В каждой строке текстового файла найти наиболее длинную последовательность цифр. Значение ее длины преобразовать в строку, которую записать в начале строки файла. Результаты записать в новый файл.
24. Даны два текстовых файла. Создать третий файл из символов, которые записаны в позициях с одинаковыми номерами, но различаются между собой.
25. Дан текстовый файл с программой. Исключить комментарии в тексте Pascal-программы.
26. Написать программу, которая подсчитывает количество пустых строк в текстовом файле.
27. Написать программу, которая находит максимальную длину строки текстового файла и печатает эту строку.
28. Пусть текстовый файл разбит на непустые строки. Написать программу для подсчета числа строк, которые начинаются и оканчиваются одной и той же литерой.
29. Пусть текстовый файл разбит на непустые строки. Написать программу для подсчета числа строк, которые состоят из одинаковых литер.
30. Написать программу, переписывающую содержимое текстового файла t2 в текстовый файл t1 (с сохранением деления на строки).

3. Основные сведения и методические указания

3.1 Множества

В задачах обработки строк часто используют множества. Множеством в языке Pascal называется конечный набор однотипных данных. Объекты, из которых состоит множество, называются элементами множества. От массива множество отличается отсутствием индексации его элементов. Описание множества выполняется либо в разделе типов, либо в разделе описания переменных. При описании множества используется зарезервированное слово set:

```
Type <имя типа>=set of <базовый тип>;
```

Базовым типом множества может быть любой упорядоченный скалярный тип, состоящий не более чем из 256 элементов. Пример описания множества:

```
Type Bukvi=set of char;
```

Допустимыми значениями множественного типа данных являются все возможные подмножества, составленные из значений базового типа. Если переменная типа «множество» описана как set of 1..3, то она может принимать значения из следующего набора: {(1, 2, 3), (1,2), (1, 3), (2, 3), (1), (2), (3), ()}.

Константы множественного типа записываются с помощью квадратных скобок и списка элементов. Примеры:

```
Const
 alphabet = ['A'..'Z',  'a'..'z'];
 empty = [];
 digits = [0..9];
```

Отсутствие списка элементов в квадратных скобках означает пустое множество. Зарезервированное слово in используется для определения принадлежности элемента множеству:

```
Var ch:char;
If  ch in alphabet  then  ...
```

Над множествами языка Pascal можно выполнять те же операции, что и над математическими множествами, то есть находить их объединение, пересечение, разность. Если S1 и S2 — константы или переменные множественного типа, то S1 + S2 будет их объединением, S1 * S2 — пересечением, S1 - S2 — разностью. К множествам могут применяться операции отношений: = (равенство), <> (неравенство), <= (является подмножеством), >= (является надмножеством). Чтобы добавить во множество какой-нибудь элемент, следует объединить его с множеством, состоящим из единственного элемента:

```
S := S + [ch];
```

Также можно использовать процедуру Include(S, ch). Имеется также процедура Exclude для исключения элемента из множества. У этой процедуры первый параметр означает множество, а второй — исключаемый элемент.

Рассмотрим пример. Пусть требуется создать множество из литер, найти в нем элемент с минимальным кодом, распечатать все элементы множества в алфавитном порядке, выход из программы создания множества осуществить при вводе символа '*'. Работу программы организовать с использованием процедур и пунктов меню.

```
Program  DemoSet;
Uses Crt; { Указание имени модуля, содержажего процедуру ClrScr}
Var        S:Set of 'A'..'z'; { описание множества }
           C:Char;
    Number:Integer;

Procedure Konstr;  {Процедура создания множества}
begin
     S:=[];
     Writeln('Вводите буквы от A до Z');
     Readln(C);
     While C<>'*' Do {Вводим литеры во множество, до тех пор пока
                      не будет введен символ '*'}
     begin
       S:=S+[C];     {Добавление элемента во множество с помощью
                      операции объединения двух множеств S и [C]}
       Readln(C)
    end
end;

Procedure Smin;  { Процедура поиска литеры с  минимальным  кодом,
                      входящей во множество S}
begin
   C:='A';
   While not (C in S) Do
         C:=Succ(C);
   Writeln('Литера с минимальным кодом, входящая в множество S ', C);
   Writeln('Нажмите клавишу Enter');
   Readln;
end;

Procedure Rasp;  { Процедура печати элементов множества S в
                      алфавитном порядке}
begin
  Writeln('Литеры, входящие в множество:');
  For C:='A' To 'z' Do
```

```
    If C in S Then Writeln(C);
    Writeln('Нажмите клавишу Enter');
    Readln;
end;

 {===========основная программа============}
begin

   {бесконечный цикл, обеспечивающий вывод на экран
    пунктов меню}
   While True Do
    Begin
     ClrScr; { очистка экрана }
     Writeln('1-Формирование множества');
     Writeln('2-Поиск элемента с мин. кодом');
     Writeln('3-Печать множества');
     Writeln('4-Выход');
     Writeln('-----------------------------');
     Writeln('Введите номер пункта меню');
     Readln(Number);
     Case Number Of { вызов необходимой процедуры по номеру}
      1:Konstr;
      2:Smin;
      3:Rasp;
      4:Exit { Встроенная процедура выхода}
     End
   End.
```

3.2. Строковый тип

Строка представляет собой последовательность литер и в стандартном языке Pascal описывается в виде упакованного одномерного массива, компоненты которого есть литеры, а тип индекса задается диапазоном 1..N, где N>1:

```
Var stroka: packed array[1..N] of char;
```

В языке Turbo Pascal строковый тип может быть также описан с помощью зарезервированного слова String. Пример описания строковой переменной в языке Turbo Pascal:

```
Var stroka1 : String[10];
```

Целое число в квадратных скобках задает длину строки. Максимально допустимая длина строки типа string составляет 255 символов и назначается строковой переменной по умолчанию, если длина не указана. В примере резервируется для

переменной stroka1 одиннадцать байтов памяти. Дополнительный байт содержит фактическую длину строковой переменной. Длина строки равна значению нулевого элемента массива, т.е. stroka1[0].

В языке Pascal имеется набор процедур и функций для работы со строками. Строке можно присваивать значение строковой константы:

```
stroka1:='ABC';
```

К строкам можно применять операцию *конкатенации,* которая обозначается знаком «+». Конкатенация — это объединение строк:

```
stroka2:=stroka1+'DEF';
```

Результатом такой последовательности операторов будет строка 'ABCDEF'. Длина строковой переменной stroka2 должна задаваться с учетом суммарной длины слагаемых. Строковая константа в исходном тексте программы должна размещаться в пределах одной строки текста. Чтобы задать длинное строковое значение, занимающее в тексте программы несколько строк, можно воспользоваться операцией конкатенации. Операцию слияния строк strl и str2 выполняет и функция Concat(strl, str2).

Функция Length(str) определяет длину аргумента str строкового типа.

Процедура Delete(str, start, nrem) используется для того, чтобы удалить nrem символов в строковой переменной str, начиная с позиции start.

Имеется также процедура Insert, предназначенная для вставки одной строки в другую. Чтобы вставить строку (константу или значение строковой переменной) instr в строковую переменную str, начиная с позиции start, данную процедуру следует вызвать следующим образом:

```
Insert(instr, str, start);
```

Функция Copy(strl, start, n) копирует n символов строки strl, начиная с позиции start. Это строковое значение можно присвоить другой строковой переменной.

Функция Pos(substr, str) определяет начальную позицию подстроки substr в строке str. Например, результат следующего вызова равен 5:

```
Pos(' F(x) ', 'Let F(x) = 2x')
```

Если подстрока не встретилась в строке, значение Pos равно нулю.

Имеются две процедуры преобразования числовых значений в строковые и наоборот — Str и Val. Процедура Str должна вызываться следующим образом:

```
Str(num, strnum);
```

Здесь num — значение числового типа, а strnum — переменная строкового типа. Процедура присваивает переменной strnum строковое значение, представляющее собой символьное представление значения переменной num. Процедура Val выполняет обратное преобразование. Обращение к ней имеет вид:

```
Val(strnum, num, errcode);
```

Третий параметр в этой процедуре равен нулю при успешном выполнении преобразования. В том случае, когда первый параметр содержит символы, недопустимые при записи числа, значение параметра errcode по окончании работы процедуры будет равно номеру позиции с ошибочно заданным символом.

Две строки можно сравнивать при помощи операций >, <, <=, >=. При этом сравниваются коды символов, начиная с первых символов строк.

3.3. Файловый тип

Файловый тип языка Паскаль представляет последовательность однотипных элементов. Число элементов в определении файлового типа в отличие от массива не фиксируется. В зависимости от типа элементов файлы подразделяются на текстовые и бинарные. Текстовые файлы предназначены для хранения текстов (например, текстов программ на языке Pascal), а бинарные используются для хранения данных разных типов.

Текстовый файл содержит последовательность символов, организованных в строки. Каждая строка файла заканчивается *меткой конец строки*, состоящей из пары управляющих символов: возврат каретки CR = #13 и перевод строки LF = #10. Заканчивается файл меткой конец файла (управляющий символ #26). Работу с текстовыми файлами обеспечивает модуль system среды программирования Turbo Pascal, который не указывают в операторе uses.

Для описания текстовых файлов в языке Pascal используется стандартный файловый тип text. Прежде чем приступить к операциям над текстовыми файлами, необходимо описать переменную типа text:

```
Var in_file : text;
```

Процедура Assign связывает имя внешнего файла (файла, который физически расположен на жестком диске) с файловой переменной. Например:

```
Assign(in_file, 'C:\work\ivanov\my_file');
```

Здесь my_file имя внешнего файла (физический файл), а in_file – файловая переменная (логический файл), используемая в программе в качестве параметра процедур работы с файлами.

После установления связи внешнего файла с файловой переменной внешний файл надо открыть для записи или чтения. При открытии файла выполняются необходимые системные операции, подготавливающие файл к записи или считыванию

информации. Текстовый файл in_file открывается процедурой Reset(in_file) только для чтения, а процедурой Rewrite(in_file) — только для записи. После открытия файла процедурой Rewrite файл считается пустым. Если файл с таким именем уже существовал, то данные, хранившиеся ранее в этом файле, становятся недоступными. Если требуется открыть существующий файл в режиме записи с возможностью дописывания данных в конец, то используют процедуру Append(<имя файловой переменной>). По завершении обработки файла программой он должен быть закрыт. После закрытия файла связанный с ним внешний файл обновляется, а файловая переменная может быть связана с другим внешним файлом. Закрывается файл с помощью процедуры Close(in_file).

Доступ к текстовому файлу организуется последовательно, то есть программа не может в любой момент времени считать из него произвольную порцию информации или произвести запись в произвольное место файла. Файл представляет собой линейную последовательность элементов, каждый из которых имеет свой номер. Указатель файла указывает на текущую позицию файла, из которой выполняется чтение значений или запись. Указатель файла при считывании очередного элемента файла перемещается к следующему элементу.

К файловым переменным применима стандартная функция Eof(<файловая переменная>), которая принимает значение TRUE, если указатель файла указывает на конец файла, и значение FALSE в ином случае.

Для записи значений в файл или чтения из него используются процедуры:

- записать в файл

  Write(<файловая переменная>,<список вывода>);
- считать из файла

  Read(<файловая переменная>,<список ввода>).

Например: Read(in_file, X, Y); Здесь список ввода представлен переменными X и Y, которым присваиваются значения двух очередных элементов из файла, связанного с файловой переменной in_file.

Запись признака конца строки в текстовый файл выполняется процедурой Writeln(<файловая переменная>).

Для обнаружения признака конца строки при чтении текстового файла используется встроенная функция Eoln(<файловая переменная>), принимающая значение TRUE, если из файла считан символ, за которым стоит признак конца строки, и значение FALSE в противном случае. Процедура Readln(<файловая переменная>,<список ввода>) пропускает все символы до начала следующей строки текстового файла (т.е. переводит указатель файла на новую строку), а затем присваивает значения в соответствии со списком ввода.

Текстовый файл может использоваться для хранения числовых значений. При считывании таких значений из файла или их записи в файл происходит автоматическое преобразование из числового формата в символьный и наоборот.

При обращениях к стандартным функциям и процедурам ввода-вывода автоматически производится проверка на наличие ошибок. При обнаружении ошибки программа прекращает работу и выводит на экран соответствующее сообщение. С

помощью директив компилятора {$I+} и {$I-} автоматическую проверку ошибок ввода-вывода можно включить или выключить. Если автоматическая проверка отключена, ошибки ввода-вывода, возникающие при работе программы, не приводят к ее останову. Стандартная функция IOResult возвращает код ошибки. Нулевое значение кода ошибки означает нормальное завершение операции ввода-вывода.

3.4. Пример программы обработки строк файлов

Пусть требуется написать программу, которая считывает заданный текстовый файл и выводит номера строк, в которых содержится введенное пользователем слово. Длина строки текста не превышает 80 символов. В программировании *словом* принято называть последовательность символов, ограниченную разделителями. В качестве разделителей используются пробелы, запятые, точки, двоеточия, точки с запятой. Текст не содержит переносов слов.

В предыдущих лабораторных работах был рассмотрен общий порядок действий при создании программы. Будем придерживаться его и впредь.

Исходные данные и результаты.

Исходные данные:

1. Текстовый файл неизвестного размера, состоящий из строк длиной не более 80 символов. Поскольку по условию переносы отсутствуют, можно ограничиться поиском слова в каждой строке текста отдельно. Для ее хранения введем строковую переменную длиной 80 символов.
2. Слово-образец, вводимое для поиска с клавиатуры. Для его хранения также выделим строку длиной 80 символов.

Результатом работы программы является номер строки, в которую входит слово. Представим его в программе целым значением.

Для хранения длины строки будем использовать именованную константу, а для хранения фактического количества символов в строке – переменную целого типа. Для работы с файлом потребуется файловая переменная соответствующего типа.

Алгоритм решения задачи.

1. Построчно считывать текст из файла.
2. Просматривать каждую строку и искать в ней первое вхождение слова. При каждом обнаружении слова печатать номер строки и саму строку.

Детализируем второй пункт алгоритма. Для обнаружения слова организуем цикл просмотра символов считанной строки. Если очередной символ строки не является разделителем, то добавляем его в строку, которая предназначена для хранения слова, считываемого из текущей строки текста. Когда очередной символ строки разделитель, то это означает, что в строке выделено очередное слово и его можно сравнить со словом-образцом .

Программа и тестовые примеры.

Текст программы с комментариями приведен ниже.

```
Program  search_word_in_file;
```

```
Const
        delimiters = [' ','.',',',':',';']; {разделители слов}
        len=80;  {максимальная длина строки файла}
Var
        in_file : text;
        pattern_word,                   {введенное слово-образец}
        word_intext,                    {слово, найденное в тексте}
        stroka : string[len];           {считываемая строка файла}
        k, n, strokalength : word;
        file_name : string[20];
Begin
WriteLn( 'Введите имя текстового файла');
WriteLn;
ReadLn(file_name);
WriteLn('Введите образец для поиска');
WriteLn;
ReadLn(pattern_word);

{Отключение автоматической проверки ошибок ввода-вывода}
{$I-}
Assign(in_file, file_name);
Reset(in_file);
if   IOResult > 0   then
Halt;           {Если ошибка открытия файла, то останов}

 {чтение строк до конца файла}
n := 0;
while   not Eof(in_file)   do
begin
    Inc(n); {увеличить номер строки}
    ReadLn(in_file, stroka);

    {запомнить длину текущей строки}
    strokalength := Length(stroka);

    {выделение слова из текущей строки}
    k := 1;
    word_intext := ' ';
    while   k <= strokalength do
    begin
        if (stroka[k] in delimiters) or (k=strokalength) then
        begin   {обнаружен конец слова}
            if k=strokalength then  {включение последней буквы в слово}
            word_intext := word_intext+stroka[k];
            {проверка совпадения с образцом}
```

```
            if word_intext = pattern_word  then
            begin
               WriteLn('Слово ', pattern_word,' найдено в строке ', n:4, ':');
               WriteLn(stroka);
               Break; {слово-образец обнаружено, прервать поиск}
            end;
            word_intext := '';
         end
         else   {если не конец слова, то добавить букву в слово}
               word_intext := word_intext+stroka[k];
         Inc(k);
      end;
end;
Close(in_file);
WriteLn('Для завершения работы нажмите <Enter>');
ReadLn;
end.
```

Перечень символов-разделителей содержится в константе множественного типа delimiters. После того как пользователем заданы имя текстового файла (в данном случае путь доступа к файлу должен быть указан полностью, вместе с расширением), а также слово для поиска, указанный файл открывается, и строки текста последовательно считываются из него. В каждой считанной строке выделяются слова, которые сравниваются с образцом для поиска. Это делается до тех пор, пока не будет найдено искомое слово, либо будут перебраны все слова, принадлежащие текущей строке. Все необходимые для этого операции выполняются во внутреннем цикле while...do и условных операторах. Проверка прекращается, если обнаружено первое появление слова в строке. В этом случае на печать выводятся номер строки и сама строка.

Для тестирования программы требуется создать файл с текстом, в котором заданное слово встречается:

- в начале строки;
- в конце строки;
- в середине строки;
- несколько раз в одной строке;
- как часть других слов, находящаяся в начале, середине и конце этих слов;
- между различными разделителями.

Длина хотя бы одной из строк должна быть равна 80 символам. Для тестирования программы следует выполнить ее, по крайней мере, два раза: введя с клавиатуры слово, содержащееся в файле, и слово, которого в нем нет.

4. Порядок выполнения работы

1. Выбрать способ представления исходных данных задачи и результатов.
2. Разработать алгоритм решения задачи, разбив его при необходимости на отдельные подпрограммы.
3. Разработать программу на языке Pascal.
4. Разработать тестовые примеры, следуя указаниям, изложенным в разделе 3.
5. Выполнить отладку программы
6. Получить результаты работы программы для различных режимов ее работы.
7. Используя окно просмотра Watch, выяснить значение файловой переменной до и после выполнения процедуры Assign, а также до и после выполнения процедур Reset и Rewrite.

5. Содержание отчета

Цель работы, вариант задания, структурная схема алгоритма решения задачи с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, значения файловой переменной на разных этапах выполнения программы, выводы.

6. Контрольные вопросы

1. Какие типы данных используются в качестве базовых при построении множественных типов?
2. Приведите пример описания множественного типа
3. Какие операции определены над множественными типами и каков их приоритет?
4. Приведите примеры выполнения операций над множественными типами.
5. Напишите программу создания множества из букв латинского алфавита.
6. Чем отличается текущая длина строки от ее общей длины?
7. Где сохраняется текущая длина строки?
8. Как инициализируется строка при ее объявлении?
9. Как ввести и вывести строку?
10. Какие библиотечные функции имеются для работы со строками?
11. Объясните, что означают следующие термины: логический и физический файл, текстовый и бинарный файл, последовательный и прямой метод доступа.
12. Каким образом открывают и закрывают файлы?
13. Как выполняется запись данных в файл последовательного доступа, чтение из файла?
14. Как обнаружить конец файла? Как обнаружить конец строки текстового файла?

15. Как связать файловую переменную с файлом расположенным на внешнем устройстве?

ЛАБОРАТОРНАЯ РАБОТА №8
ПРОГРАММИРОВАНИЕ ОПЕРАЦИЙ НАД ЗАПИСЯМИ И ТИПИЗИРОВАННЫМИ ФАЙЛАМИ

1. Цель работы

Изучение способов обработки логически связанных данных различных типов. Создание программ, использующих записи и типизированные файлы.

2. Варианты заданий

1. Описать структуру с именем STUDENT, содержащую следующие поля:
- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по возрастанию номера группы;
- чтение данных из этого файла, корректировку данных в файле по номеру записи;
- вывод на дисплей фамилий и номеров групп для всех студентов, включенных в массив, если средний балл студента больше 4.0;
- если таких студентов нет, вывести соответствующее сообщение.

2. Описать структуру с именем STUDENT, содержащую следующие поля:
- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по возрастанию среднего балла;
- чтение данных из этого файла;
- вывод на дисплей фамилий и номеров групп для всех студентов, имеющих оценки 4 и 5;
- если таких студентов нет, вывести соответствующее сообщение.

3. Описать структуру с именем STUDENT, содержащую следующие поля:
- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку 2;
- если таких студентов нет, вывести соответствующее сообщение.

4. Описать структуру с именем AEROFLOT, содержащую следующие поля:
- название пункта назначения рейса;
- номер рейса;
- тип самолета.

Написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа AEROFLOT; записи должны быть упорядочены по возрастанию номера рейса;
- чтение данных из этого файла;
- вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры;
- если таких рейсов нет, выдать на дисплей соответствующее сообщение

5. Описать структуру с именем AEROFLOT, содержащую следующие поля: Q название пункта назначения рейса;
- номер рейса;
- тип самолета.

Написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа AEROFLOT; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;
- чтение данных из этого файла;
- вывод на экран пунктов назначения и номеров рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры;
- если таких рейсов нет, выдать на дисплей соответствующее сообщение.

6. Описать структуру с именем WORKER, содержащую следующие поля:
- фамилия и инициалы работника;
- название занимаемой должности;
- год поступления на работу.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа WORKER; записи должны быть размещены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры;
- если таких работников нет, вывести на дисплей соответствующее сообщение.

7. Описать структуру с именем TRAIN, содержащую следующие поля:
- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;
- чтение данных из этого файла;
- вывод на экран информации о поездах, отправляющихся после введенного с клавиатуры времени;
- если таких поездов нет, выдать на дисплей соответствующее сообщение.

8. Описать структуру с именем TRAIN, содержащую следующие поля:
- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN; записи должны быть упорядочены по времени отправления поезда;
- чтение данных из этого файла;
- вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры;
- если таких поездов нет, выдать на дисплей соответствующее сообщение.

9. Описать структуру с именем TRAIN, содержащую следующие поля:

- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN; записи должны быть упорядочены по номерам поездов;
- чтение данных из этого файла;
- вывод на экран информации о поезде, номер которого введен с клавиатуры;
- если таких поездов нет, выдать на дисплей соответствующее сообщение.

10. Описать структуру с именем MARSH, содержащую следующие поля:
- название начального пункта маршрута;
- название конечного пункта маршрута;
- номер маршрута.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа MARSH; записи должны быть упорядочены по номерам маршрутов;
- чтение данных из этого файла;
- вывод на экран информации о маршруте, номер которого введен с клавиатуры;
- если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

11. Описать структуру с именем MARSH, содержащую следующие поля:
- название начального пункта маршрута;
- название конечного пункта маршрута;
- номер маршрута.

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа| MARSH; записи должны быть упорядочены по номерам маршрутов;
- чтение данных из этого файла;
- вывод на экран информации о маршрутах, которые начинаются или оканчиваются в пункте, название которого введено с клавиатуры;
- - если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

12. Описать структуру с именем NOTE, содержащую следующие поля:
- фамилия, имя;
- номер телефона;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа NOTE; записи должны быть упорядочены по датам рождения;
- чтение данных из этого файла;
- - вывод на экран информации о человеке, номер телефона которого введен с клавиатуры;
- если такого нет, выдать на дисплей соответствующее сообщение.

13. Описать структуру с именем NOTE, содержащую следующие поля:
- фамилия, имя;
- номер телефона;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа NOTE; записи должны быть размещены по алфавиту;
- чтение данных из этого файла;
- - вывод на экран информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры;
- - если таких нет, выдать на дисплей соответствующее сообщение.

14. Описать структуру с именем NOTE, содержащую следующие поля:
- фамилия, имя;
- номер телефона;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа NOTE; записи должны быть упорядочены по трем первым цифрам номера телефона;
- чтение данных из этого файла;
- вывод на экран информации о человеке, чья фамилия введена с клавиатуры;
- если такого нет, выдать на дисплей соответствующее сообщение.

15. Описать структуру с именем ZNAK, содержащую следующие поля:
- фамилия, имя;
- знак Зодиака;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью процедур или функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK; записи должны быть упорядочены по датам рождения;
- чтение данных из этого файла;

- - вывод на экран информации о человеке, чья фамилия введена с клавиатуры;
- если такого нет, выдать на дисплей соответствующее сообщение.

16. Описать структуру с именем ZNAK, содержащую следующие поля:
- фамилия, имя;
- знак Зодиака;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK; записи должны быть упорядочены по датам рождения;
- чтение данных из этого файла;
- вывод на экран информации о людях, родившихся под знаком, название которого введено с клавиатуры;
- если таких нет, выдать на дисплей соответствующее сообщение.

17. Описать структуру с именем ZNAK, содержащую следующие поля:
- фамилия, имя;
- знак Зодиака;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK; записи должны быть упорядочены по знакам Зодиака;
- чтение данных из этого файла;
- вывод на экран информации о людях, родившихся в месяц, значение которого введено с клавиатуры;
- если таких нет, выдать на дисплей соответствующее сообщение.

18. Описать структуру с именем PRICE, содержащую следующие поля:
- название товара;

название магазина, в котором продается товар;
- стоимость товара в руб.

Написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа PRICE; записи должны быть размещены в алфавитном порядке по названиям товаров;
- чтение данных из этого файла;
- вывод на экран информации о товаре, название которого введено с клавиатуры;
- если таких товаров нет, выдать на дисплей соответствующее сообщение

19. Описать структуру с именем PRICE, содержащую следующие поля:
- название товара;
- название магазина, в котором продается товар;
- стоимость товара в руб.

Написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа PRICE; записи должны быть размещены в алфавитном порядке по названиям магазинов;
- чтение данных из этого файла;
- вывод на экран информации о товарах, продающихся в магазине, название которого введено с клавиатуры;
- если такого магазина нет, выдать на дисплей соответствующее сообщение.

20. Описать структуру с именем ORDER, содержащую следующие поля:
- расчетный счет плательщика;
- расчетный счет получателя;
- перечисляемая сумма в руб.

Написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа ORDER; записи должны быть размещены в алфавитном порядке по расчетным счетам плательщиков;
- чтение данных из этого файла;
- вывод на экран информации о сумме, снятой с расчетного счета плательщика, введенного с клавиатуры;
- если такого расчетного счета нет, выдать на дисплей соответствующее сообщение.

21. Описать структуру с именем STUDENT, содержащую следующие поля:
- номер;
- фамилия и имена;
- год рождения;
- год поступления в университет;
- структура OCENKI, содержащая четыре поля: физика, математика, программирование, история;

Написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов отличников;
- если таких студентов нет, вывести соответствующее сообщение.

22. Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены номеру;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших одну оценку 3;
- если таких студентов нет, вывести соответствующее сообщение.

23. Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по фамилиям;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших все двойки;
- если таких студентов нет, вывести соответствующее сообщение.

24. Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по номеру;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших все пятерки;
- если таких студентов нет, вывести соответствующее сообщение.

25. Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших одну оценку 4, а все остальные – 5;
- если таких студентов нет, вывести соответствующее сообщение.

26. Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, которые начинаются с литеры ‘A’, и их оценки;
- если таких студентов нет, вывести соответствующее сообщение.

27. Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по номеру;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, которые начинаются с литеры ‘Б’, и год их рождения;
- если таких студентов нет, вывести соответствующее сообщение.

28. Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по фамилиям;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, которые начинаются с литеры ‘Б’или ‘Г’, и год их поступления;
- если таких студентов нет, вывести соответствующее сообщение.

29. Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по фамилиям;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, имеющих средний балл выше среднего балла группы;
- если таких студентов нет, вывести соответствующее сообщение.

30. Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью процедур или функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по номеру;
- чтение данных из этого файла;
- вывод на дисплей фамилий и года рождения студентов, не получивших ни одной тройки;
- если таких студентов нет, вывести соответствующее сообщение.

3. Основные сведения и методические указания

3.1 Записи

Запись (комбинированный тип) – это структурированный тип данных, состоящий из нескольких компонент (полей). В отличие от массивов компоненты записи могут иметь разные типы, и доступ к ним осуществляется не по индексу, а по имени поля. При определении записи задаются имя и тип каждого

поля. Описание записи начинается служебным словом record и заканчивается словом end. В качестве примера рассмотрим запись, содержащую анкетные данные студента:

```
Type
 Student=record
            First_name: string[15];
            Last_name: string[15];
            birthday:  record
                          day:1..31;
                          month:1..12;
                          year:integer
                          end;
            end;
 Var
      S:Student;
      Gruppa: array[1..30] of Student;
```

К каждому компоненту записи можно обратиться, используя имя переменной и имя поля, разделенные точкой. Такие имена в программах называются *составными именами*. Например,

```
S.First_name:= 'ИВАНОВ';
```

Поле с именем birthday – тоже запись и состоит из трех полей. Доступ к этим полям осуществляется добавлением через точку имени соответствующего поля:

```
S. birthday.day:=15;
S. birthday.month:=4;
S. birthday.year:=1978;
```

Приведенные операторы присваивания можно записать короче, если использовать оператор присоединения:

```
With < имя переменной типа запись> Do
Begin
      <операторы>
End
```

Здесь в операторах вместо имен переменных типа запись указываются только имена полей:

```
With S. birthday Do
    Begin
```

```
    day:=15;
    month:=4;
    year:=1978
  End;
```

Записи могут входить в качестве компонентов в другие переменные. Например, переменная Gruppa – это массив, состоящий из 30 записей. Оператор присваивания нового значения полю month пятого элемента массива имеет вид

```
Gruppa[5].birthday.month:=12;
```

Легко видеть, что к каждой записи группы доступ осуществляется при помощи индекса.

Приведенный ниже фрагмент программы подсчитывает число студентов, родившихся в 1978 году:

```
K:=0;
For i:=1 To 30 Do
  With Gruppa[i].birthday  Do
    If year=1978 then
          K:=K+1;
 Writeln(` число родившихся в 1978 году = `, K:2);
```

При определении записи в неё можно включать вариантную часть. Например, опишем запись person, предназначенную для хранения данных либо о преподавателе, либо о студенте. Такая запись будет содержать фиксированную часть, в которой будет указана фамилия персоны и адрес, а также вариантную часть, в которой для преподавателя будет указываться должность (post) и количество трудов, а для студента средний балл и группа.

```
Type person=record
          FIO: string;
          Address: string;
          Case who: (teacher,student) of
                Teacher: (post: string; number:integer);
                Student: ( mark:real; gruppa:string)
          End;
```

Вариантная часть записи начинается словом case. После ее окончания в записи не могут появляться никакие другие поля, поэтому слово Case не закрывается служебным словом End. За словом case следует *поле признака* с указанием его типа – это поле who. Если это поле примет значение teacher, то рассматриваемая запись будет включать поля, соответствующие первому варианту, иначе – второму. Тип поля признака должен быть перечислимым.

Для переменной P типа person можно записать следующие операторы присваивания:

```
P.FIO:='СТОЛЯРОВ';
P.address:='ул. Тенистая д.5 кв. 7';
P.who:=teacher;
P.post:='docent';
P.number:=52;
```

Задав значение поля признака who, следует присваивать новые значения только полям того варианта, которые помечены соответствующей константой выбора.

3.2. Типизированные и нетипизированные файлы

Бинарные файлы в языке Turbo Pascal подразделяются на типизированные и нетипизированные. Бинарные файлы допускают прямой доступ к элементам файла, то есть в соответствии с их номерами. Описание типизированного файла имеет вид:

```
Var Typed_file : file of <тип >;
```

Здесь <тип> может быть любым типом за исключением файлового. Элементами типизированного файла являются значения указанного типа.

При работе с типизированными файлами используются процедуры Assign, Reset и Rewrite. Текстовый файл, открытый процедурой Reset, доступен только для чтения, а типизированный файл доступен как для чтения, так и для записи. Особенность использования процедур Read и Write для типизированных файлов заключается в том, что каждый из параметров этих процедур должен быть принадлежать заданному типу. Например:

```
Var  F: file of integer;
       X: integer;
…
Read(F,X);
```

Процедуры ReadLn и WriteLn применяются *только* к текстовым файлам.

В языке Turbo Pascal имеются специальные процедуры и функции для работы с типизированными файлами. Функция FilePos(F), где F –типизированная файловая переменная, возвращает номер текущего элемента файла. Функция FileSize(F) возвращает количество элементов файла. Процедура Seek(F,N) перемещает указатель текущего элемента в позицию N. Применяя последовательно процедуры Seek(F,N) и Read(F,X), можно из файла считывать элементы в произвольном порядке.

Чтобы эффективно выполнять операции ввода-вывода для внешних файлов, в языке Pascal целесообразно использовать нетипизированные файлы, так как при работе с ними можно применять быстрые дисковые операции низкого уровня. Нетипизированные файлы рассматриваются как последовательность байтов. Описание нетипизированной файловой переменной имеет вид

```
VarUntyped_file : file;
```

Такая файловая переменная связывается с внешним файлом обычным образом. В числе параметров процедур Reset и Rewrite для нетипизированных файлов кроме файловой переменной имеется необязательный второй параметр типа Word, например:

```
Reset(untyped_file, n);
```

Дополнительный параметр n описывает размер индивидуальной записи в файле в байтах. Если параметр n отсутствует, его значение по умолчанию принимается равным 128.

При работе с нетипизированными файлами для чтения и записи значений используются процедуры BlockRead и BlockWrite. Эти процедуры требуют организации в памяти буфера (область памяти, предназначенная для временного хранения данных). В качестве буфера обычно используются массивы.

3.3 Пример программы обработки записей типизированного файла

Имеются записи о сотрудниках, содержащие фамилию и инициалы, год рождения и оклад сотрудника. Требуется написать программу с использованием процедур и функций, которая:

- вводит записи о сотрудниках с клавиатуры в типизированный файл и упорядочивает его по размеру оклада;
- осуществляет поиск сотрудников в файле по фамилии и вычисляет средний оклад этих сотрудников;
- отображает содержимое типизированного файла.

Исходные данные и результаты.

Исходные данные:

Сведения об одном сотруднике будем представлять в виде записи. При этом фамилию и инициалы представим строкой из 15 символов, год рождения – целым типом, оклад – вещественным типом. Поскольку количество записей о сотрудниках не оговорено, будем хранить все сведения в типизированном файле, а не массиве.

Результаты. В результате работы программы требуется вывести на экран требуемые элементы файла. Так как результаты представляют собой выборку из исходных данных, дополнительная память для них не отводится. Кроме того, необходимо подсчитать средний оклад для найденных сотрудников. Для этого введем переменную вещественного типа.

Алгоритм решения задачи.

1. Ввести с клавиатуры сведения о сотрудниках в типизированный файл. При этом будем прекращать ввод, если пользователь введет символ ‘*’.
2. Выполнить сортировку файла по окладу, используя возможность прямого доступа к элементам типизированного файла. В качестве метода сортировки будем использовать метод прямого обмена (пузырьковую сортировку);
3. Обеспечить вывод сведений о сотруднике:
   a. ввести с клавиатуры фамилию;
   b. выполнить поиск сотрудника в файле по фамилии;
   c. увеличить суммарный оклад и счетчик количества сотрудников;
   d. вывести сведения о сотруднике или сообщение об его отсутствии;
   e. вывести средний оклад.
4. Отобразить содержимое типизированного файла.

Каждый из указанных пунктов алгоритма представим в форме процедуры, которые можно будет вызывать из основной программы в произвольном порядке. При этом контроль над порядком вызова процедур полностью возлагается на пользователя.

Программа и тестовые примеры

```
Program Demo_TypedFiles;
Uses Crt;{подключить модуль управления клавиатуры и монитора}
Type person=record        {запись о сотруднике}
        FIO:string[15];   {фамилия и инициалы}
        Year:integer;     {год рождения}
        Salary:real;      {оклад}
        End;

Var   f:file of person;          {файл с записями о сотрудниках}
      Elem1,Elem2:person;  {два элемента файла типа запись}

{===============создание файла==================}
Procedure Create_file;
Begin

{Установить указатель в позицию, определяемую размером файла.
Если создается новый файл, то в начало. Если записи добавляются в существующий файл, то в конец}
Seek(f,FileSize(f));

Writeln(‘Вводите сведения о сотрудниках.’);
Writeln(‘Для выхода вместо фамилии напечатайте символ *’);
While True Do
```

```
        Begin
                Write('Введите фамилию и инициалы (обязательно)');
                Readln(Elem1.FIO);
                If Elem1.FIO='*' Then Break; {Выход, если введена *}
                Write('Введите год рождения');
                Readln(Elem1.Year);
                Writeln('Введите оклад');
                Readln(Elem1.Salary);
                Write(f,Elem1)
        End
End;

{==============сортировка файла====================}
Procedure Sort_file;                    {Метод пузырька}
Var i,j:integer;
Begin
  Seek(f,0);        {Указатель файла установить в начало}
  For i:=filesize(f)-1 downto 1 do {цикл, задающий число проходов}
    For j:=0 to i-1 do     {сравнивать элементы от 0-го до (i-1)-го}
        Begin
        Seek(f,j);    {Указатель установить на j-ый элемент}
        Read(f,Elem1,Elem2);                {Прочитать две записи}
        If Elem1.Salary>Elem2.Salary then       {Сравнить оклады}
                Begin
                Seek(f,j);          {Вернуть указатель в j-ую позицию}
                Write(f,Elem2,Elem1); {Поменять местами записи}
                End
        End
End;

{===========вывод содержимого файла==============}
Procedure Print_file;
Begin
Seek(f,0);          {Указатель файла установить в начало}
Writeln('Фамилия':15, 'Год':5, 'Зарплата':10);
While not eof(f) do       {Просмотр всех записей до конца файла}
        Begin
                Read(f,Elem1);
                Writeln(Elem1.FIO:15, Elem1.Year:5, Elem1.Salary:10:2);
        End
Readln; {Ожидание нажатия клавиши Enter}
End;

{====поиск сотрудников и вычисление ср. оклада==========}
Procedure Search_persons;
```

```
Var   Found:boolean;  {признак успеха поиска}
      S:string[15];           {строка с фамилией}
      N_persons:integer;      {количество найденных сотрудников}
      SummaSalary:Real;       {суммарный оклад сотрудников}

Begin
N_persons:=0;
SummaSalary:=0;
While True Do
  Begin
  Writeln('Введите фамилию или *');
  Readln(s);
  If s='*' Then Break;   {Прерывание поиска, если введена *}
  Found:=False;
  Seek(f,0);        {Указатель файла установить в начало}

  {Пока не конец файла и пока не найден  сотрудник с введенной
  фамилией продолжать поиск}
  While not eof(f) and not Found Do
      Begin
      Read(f,Elem1); {Прочитать очередную запись из файла}
      If (Pos(s,Elem1.FIO)<>0) and  {Есть ли в  FIO фамилия s?}
        (Elem1.FIO[Length(s)+1]=' ') {Есть ли после нее пробел?}
            Then
            Begin
            {Вывод сведений о найденном сотруднике, увеличение
            счетчика найденных сотрудников и вычисление суммар-
            ного оклада}
            Writeln(Elem1.FIO:15, Elem1.Year:5, Elem1.Salary:10:2);
            N_persons:=N_persons+1;
            SummaSalary:=SummaSalary+Elem1.Salary;
            Found:=True; {Сотрудник найден, прервать цикл}
            End;
      End;
      If not Found Then              {Если не найден}
            Writeln('Такого сотрудника нет');
End;
      If N_persons<>0 Then
            Writeln('Средний оклад=',SummaSalary/N_persons);
Readln;     {Ожидание нажатия клавиши Enter для фиксации окна
просмотра}
End;

{==========основная программа==================}
Begin
```

```
Assign(f,'person.dat'); {Установить связь логич. и физич. файлов}
{$I-}          {отключить проверку ошибок ввода-вывода}
Reset(f);      {открыть файл для чтения и добавления записей}
{$I+}          {включить проверку ошибок ввода-вывода}
If IOResult=0 Then      {Если файл успешно открыт, то}
        writeln('Добавление записей в существующий файл')
 else  {Если файла нет, то создать новый файл и открыть его}
        begin
        Rewrite(f); {Создание и открытие файла для чтения/записи}
        writeln('Запись в новый файл');
        end;

readln;       {Ожидание нажатия клавиши Enter }

{бесконечный цикл, обеспечивающий вывод на экран
пунктов меню}
While True Do
Begin
ClrScr;                  { очистка экрана }
Writeln('1-Создание файла');
Writeln('2-Сортировка файла');
Writeln('3-Вывод содержимого файла');
Writeln('4-Поиск сотрудников и вычисление ср. оклада');
Writeln('5-Выход');
Writeln('-----------------------------');
Writeln('Введите номер пункта меню');
Readln(Number);
Case Number Of        {Вызов необходимой процедуры по номеру}
        1:Create_file;     {Вызов процедуры создания файла}
        2:Sort_file;       { Вызов процедуры сортировки файла}
        3:Print_file;      {Вызов процедуры просмотра файла}
        5:Search_persons;      {Вызов процедуры поиска сотрудника}
        4:Exit                 { Встроенная процедура выхода}
End
End.
```

Программа достаточно подробно прокомментирована. В тексте основной программы осуществляется открытие файла либо с помощью процедуры Reset, если файл существует, либо с помощью процедуры Rewrite, если файл 'person.dat' на диске не обнаружен. Это позволяет один раз ввести исходные данные в файл, а затем его многократно использовать. Если потребуется начать работу с новым файлом, то старый файл следует удалить с диска средствами операционной системы.

Все необходимые действия с файлом выполняются с помощью процедур, вызываемых из пунктов меню, отображаемых на экране. При заполнении файла

требуется обязательно вводить инициалы, которые должны отделяться от фамилии пробелом. Фамилия должна начинаться с первой позиции каждой строки. В дальнейшем это свойство используется в процедуре поиска сотрудников.

Рассмотрим подробнее процедуру Search_persons. Здесь цикл поиска сотрудников по фамилии организован как бесконечный цикл, выход из которого выполняется при вводе вместо фамилии символа ‘*”. В процедуре введена переменная Found для того, чтобы после окончания цикла поиска сотрудника в файле было известно, завершился ли он успешно. При этом проверка совпадения фамилии сотрудника производится в два этапа. Сначала с помощью функции Pos проверяется, входит ли подстрока, представляющая фамилию, в строку, содержащую фамилию и инициалы. Если входит, то функция Pos вернет ненулевое значение. После этого проверяется, есть ли непосредственно после фамилии пробел (если пробела нет, то искомая фамилия является частью другой, и эта запись нам не подходит).

При отладке программы возможны проблемы при поиске фамилий, заданных русскими буквами. Это обусловлено разной кодировкой букв русского алфавита в операционных системах MS DOS (кодировка ASCII) и Windows (кодировка ANSI). Для того чтобы при работе программы нормально отображались русские сообщения, рекомендуется исходный текст программы подготовить с помощью редактора Far, выбрав кодировку DOS нажатием клавиши F8. Для ввода фамилий при работе программы на русском языке потребуется установить на компьютер драйвер клавиатуры, допускающий необходимые переключения в режиме DOS.

4. Порядок выполнения работы

1. Выбрать способ представления исходных данных задачи и результатов.
2. Разработать алгоритм решения задачи, разбив его на отдельные процедуры и функции.
3. Разработать программу на языке Pascal.
4. Разработать тестовые примеры, следуя указаниям раздела 3.
5. Выполнить отладку программы.
6. Получить результаты работы программы для различных режимов ее работы.

5. Содержание отчета

Цель работы, вариант задания, структурная схема алгоритма решения задачи с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, выводы.

6. Контрольные вопросы

1. Что понимают под комбинированным типом данных?
2. Приведите пример описания комбинированного типа.

3. Как осуществляется обращение к полям комбинированного типа?
4. Для чего применяют оператор присоединения?
5. Приведите пример использования оператора присоединения.
6. Приведите пример описания вариантных записей.
7. Как обратиться к полям вариантов?
8. В чем заключается отличие дискриминанта оператора варианта и дискриминанта вариантной записи?
9. Как выполняется описание пустых полей вариантов?
10. Чем отличаются последовательный доступ к компонентам файла и прямой доступ?
11. В чем состоит различие между типизированными и нетипизированными файлами?
12. Назовите и объясните специальные функции и процедуры для работы с типизированными файлами.
13. Как записать и считать значения из нетипизированного файла?

ЛАБОРАТОРНАЯ РАБОТА №9
ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ СПИСКОВ

1. Цель работы

Изучение списковых структур данных и приобретение навыков разработки и отладки программ, использующих динамическую память. Исследование особенностей использования переменных ссылочного типа.

2. Варианты заданий

Представить одну из приведенных ниже таблиц в виде линейного списка L, элементами которого являются строки таблицы. Написать процедуры организации, добавления элемента в список, исключения элемента из списка, просмотра списка, а также одну из процедур в соответствии с вариантом, приведенным ниже.

Значения и количество записей в таблице студент выбирает самостоятельно. Исходные данные при организации списка должны вводиться из предварительно подготовленного файла. Признаком окончания ввода данных в файл должна служить специальная запись. Количество строк таблицы в программе не задается.

1. Таблица 1. Процедуру, которая вставляет в начало списка L новый элемент E.

2. Таблица 2. Процедуру, которая вставляет в начало списка L новый элемент E.

3. Таблица 3. Процедуру, которая вставляет в начало списка L новый элемент E.

4. Таблица 1. Процедуру, которая вставляет в конец списка L новый элемент E.

5. Таблица 2. Процедуру, которая вставляет в конец списка L новый элемент E.

6. Таблица 3. Процедуру, которая вставляет в конец списка L новый элемент E.

7. Таблица 1. Процедуру, которая вставляет новый элемент E после первого элемента непустого списка L.

8. Таблица 2. Процедуру, которая вставляет новый элемент E после первого элемента непустого списка L.

9. Таблица 3. Процедуру, которая вставляет новый элемент E после первого элемента непустого списка L.

10. Таблица 1. Процедуру, которая вставляет в список L новый элемент E1 за каждым вхождением элемента E.

11. Таблица 2. Процедуру, которая вставляет в список L новый элемент E1 за каждым вхождением элемента E.

12. Таблица 3. Процедуру, которая вставляет в список L новый элемент E1 за каждым вхождением элемента E.

13. Таблица 1. Процедуру, которая вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом.

14. Таблица 2. Процедуру, которая вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом.

15. Таблица 3. Процедуру, которая вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом.

16. Таблица 1. Процедуру, которая вставляет в непустой список L, элементы которого упорядочены по возрастанию значений одного из полей таблицы, новый элемент E так, чтобы сохранилась упорядоченность.

17. Таблица 2. Процедуру, которая вставляет в непустой список L, элементы которого упорядочены по возрастанию значений одного из полей таблицы, новый элемент E так, чтобы сохранилась упорядоченность.

18. Таблица 3. Процедуру, которая вставляет в непустой список L, элементы которого упорядочены по возрастанию значений одного из полей таблицы, новый элемент E так, чтобы сохранилась упорядоченность

19. Таблица 1. Процедуру, которая удаляет из непустого списка L первый элемент.

20. Таблица 2. Процедуру, которая удаляет из непустого списка L первый элемент.

21. Таблица 3. Процедуру, которая удаляет из непустого списка L первый элемент.

22. Таблица 1. Процедуру, которая удаляет из непустого списка L второй элемент, если такой есть.

23. Таблица 2. Процедуру, которая удаляет из непустого списка L второй элемент, если такой есть.

24. Таблица 3. Процедуру, которая удаляет из непустого списка L второй элемент, если такой есть.

25. Таблица 1. Процедуру, которая удаляет из непустого списка L за каждым вхождением элемента E один элемент, если такой есть и он отличен от E.

26. Таблица 2. Процедуру, которая удаляет из непустого списка L за каждым вхождением элемента E один элемент, если такой есть и он отличен от E.

27. Таблица 3. Процедуру, которая удаляет из непустого списка L за каждым вхождением элемента E один элемент, если такой есть и он отличен от E.

28. Таблица 1.Процедуру или функцию, которая проверяет, есть ли в списке L хотя бы два элемента с равными значениями второго поля.

29. Таблица 2.Процедуру или функцию, которая проверяет, есть ли в списке L хотя бы два элемента с равными значениями второго поля.

30. Таблица 3.Процедуру или функцию, которая проверяет, есть ли в списке L хотя бы два элемента с равными значениями второго поля.

Таблица 1

N	Фамилия	Имя	Отчество	Оценки		
				Математика	История	Физика

Таблица 2

N Поезда	Станция отправления	Станция назначения	Время отправления	Время прибытия	Стоимость билета

Таблица 3

N	ФИО	Год рождения	Пол	Семейное состояние	Количество детей	Окдад

3. Основные сведения и методические указания

3.1. Ссылочный тип

В предыдущих лабораторных работах рассматривались *статические переменные*. Память под такие переменные отводится заранее на этапе компиляции программы. Иногда возникает необходимость в использовании переменных, которые создаются и уничтожаются в процессе выполнения программы. Такие переменные называют *динамическими*, а память, которая для них выделяется – *динамической памятью*. Для работы с динамической памятью исполь-

зуют *переменные ссылочного типа (указатели)*. Задание ссылочного типа выполняется следующим образом:

<задание ссылочного типа>::=^<имя типа>

Например:

```
Type
      IntPtr=^Integer;
      Massive=Array[1..10] of Real;
Var
      P:IntPtr;
      Rabmas:^Massiv;
```

Здесь ссылочная переменная P – это указатель, значением которого является адрес области памяти, в которой хранится целое значение. Ссылочная переменная RABMAS – указатель на область памяти, где хранится массив.

В языке Turbo Pascal различают типизированные и нетипизированные указатели. Приведенные выше указатели могут ссылаться лишь на данные соответствующего типа и называются типизированными. Нетипизированный указатель не связывается с каким-либо типом данных и описывается как переменная типа Pointer:

```
Var Ukaz:Pointer;
```

Такой указатель используют для ссылки на данные, тип и структура которых меняются во время выполнения программы.

После описания ссылочной переменной выделяется область динамической памяти, на которую она будет ссылаться. Для этого используют процедуру New(<имя ссылочной переменной>), которая выделяет необходимый объем памяти и присваивает переменной ссылочного типа значение, соответствующее начальному адресу выделенной памяти.

Для того чтобы записать значение в выделенную память, используют переменную с указателем

<переменная с указателем>::=<ссылочная переменная>^

Например:

```
New(P); P^:=5;
```

В этом случае в область динамической памяти, на которую ссылается указатель P, записывается значение 5 (рисунок 1). На рисунке символом "*" обозначено (в общем случае неизвестное программисту) значение ссылочной переменной P.

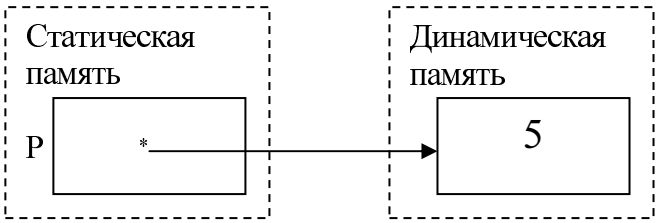

Рисунок 1. – Указатель и динамическая память

Для уничтожения объектов в динамической памяти применяется процедура Dispose(<имя ссылочной переменной>). Данная процедура освобождает область динамической памяти, на которую ссылается параметр процедуры, после чего эта область становится доступной для дальнейшего распределения между другими динамическими переменными.

Для работы с нетипизированными указателями в языке Turbo Pascal можно использовать процедуры GetMem и FreeMem. Процедура GetMem выделяет область динамической памяти заданного объема, а процедура FreeMem освобождает эту область памяти. Вызов этих процедур имеет следующий синтаксис:

```
GetMem(<имя ссылочной переменной>, <объем памяти>);
FreeMem (<имя ссылочной переменной>, <объем памяти>);
```

Здесь ссылочная переменная должна иметь тип Pointer, а объем памяти не должен превышать 65521 байт.

3.2. Списковые структуры данных

Из динамических структур в программах наиболее часто используют различные линейные списки, стеки, очереди, деревья. Они различаются способами связи отдельных элементов и допустимыми операциями над ними.

Линейные структуры данных предназначены для отображения линейных связей. Линейные структуры данных могут быть последовательными и списковыми. *Последовательной* называется структура, в которой логический порядок следования элементов задается их физическим порядком. Примером последовательной структуры данных является массив. *Списковой* называется такая структура данных, при которой логический порядок следования элементов задается путем отсылок, то есть каждый элемент списка, кроме последнего, содержит указатель на следующий элемент (или предыдущий). Доступ к первому элементу списка выполняется с помощью специального указателя – указателя на вершину (голову) списка.

Односвязным линейным списком называют список, в котором предыдущий элемент ссылается на следующий. *Двусвязный линейный список* – это список, в котором предыдущий элемент ссылается на следующий, а следующий – на предыдущий. *Односвязный циклический список* – это односвязный линейный

список, в котором последний элемент ссылается на первый. *Стек* – это односвязный список, в котором компоненты добавляются и удаляются только со стороны вершины списка. *Очередь* – это односвязный список, в котором компоненты добавляются в конец списка, а удаляются со стороны вершины списка.

Для задания списковых структур необходимо определить элемент списка в виде записи, в состав которой входит поле-указатель на элемент этого же типа. Для этого в языке Pascal разрешено определять ссылочные типы на еще не описанные типы значений:

```
Type
   Data=<тип данных элемента списка>;
   Ukaz=^Element;
   Element=Record
                D:Data;
                Next:Ukaz    { поле-указатель  на элемент}
           End;
```

Определив ссылочный тип Ukaz, можно с его помощью построить следующий связный однонаправленный список (рисунок. 2).

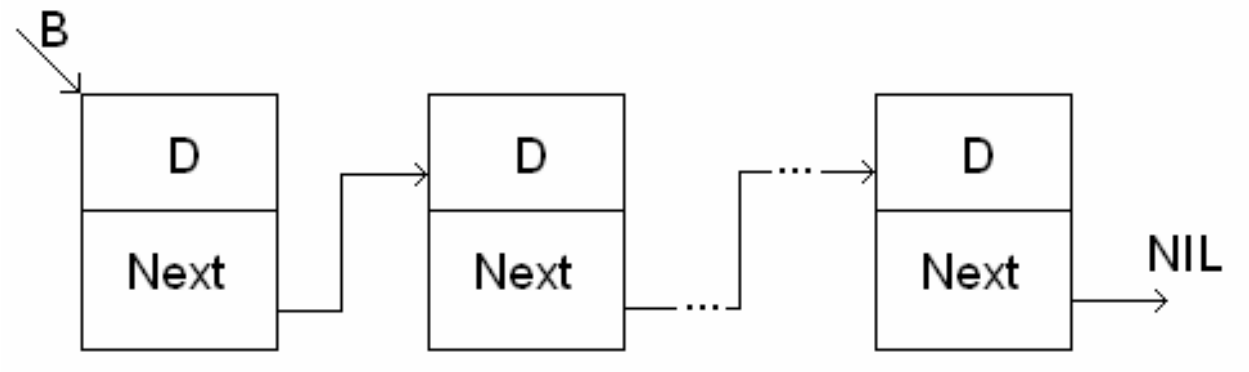

Рисунок 2. – Связный однонаправленный список

Здесь B является указателем на вершину списка. Каждый элемент списка содержит поле Next – указатель следующего элемента и поле данных D. Поле Next последнего элемента содержит пустую ссылку NIL.

Если Temp – переменная ссылочного типа, указывающая на элемент списка, предшествующий удаляемому, то чтобы исключить элемент из списка достаточно выполнить следующий оператор:

```
Temp^.Next:=Temp^.Next^.Next;
```

Чтобы вставить элемент в произвольное место списка, кроме начала, необходимо изменить значения двух указателей. Пусть ссылочная переменная Place указывает на элемент, после которого необходимо вставить новый эле-

мент. Новый элемент адресуется указателем NewE. Тогда операция вставки реализуется операторами:

```
NewE^.Next:= Place^.Next; Place^.Next:=NewE;
```

При добавлении элемента в начало списка или исключении начального элемента необходимо использовать указатель вершины списка B. Операция удаления реализуется одним оператором

```
B:=B^.Next;
```

Операция добавления – двумя операторами:

```
NewE^.Next:=B; B:=NewE;
```

3.3. Пример программы работы с очередью

Рассмотрим программу, выполняющую организацию очереди, добавление элемента в очередь, удаление элемента из очереди. Очередь – это линейный список, добавление элементов в который выполняется с одной стороны, а исключение – с другой. При этом используется специальные указатели начала и конца очереди. В примере элементами очереди являются записи, содержащие сведения о грузополучателе. Каждая запись содержит следующие поля: фамилия, имя, отчество грузополучателя, адрес.

```
Program DemoQuery;
Uses Crt;
Type
   Data=  Record  {Описание  записи о грузополучателе}
             FIO:string[15];
            Adr:string[30]
          End;

   Ukaz=^Query; {Описание указателя на элемент очереди}
   Query=Record
             Inf:Data;
             Next:Ukaz
          End;
Var
   NewE,Left,Right,Temp:Ukaz;   {Указатели}
                      Z:Data;   {Запись, добавляемая в очередь}
                    Key:Char;   {номер пункта меню программы}

Procedure Org; {Процедура организации очереди}
```

```
Begin
    Writeln('Выполняется процедура организации очереди');
    Writeln('Для выхода из процедуры вводите символ * ');
    Writeln('======================================== ');
    Writeln('Введите фамилию и инициалы грузополучателя');
    Readln(Z.Fio);
    Writeln('Введите адрес грузополучателя ');
    Readln(Z.Adr);
    If Z.Fio='*' Then Exit;  {Выход из процедуры при вводе '*'}
    New(NewE);               {Создание нового элемента      }
    NewE^.Inf.Fio:=Z.Fio;    {Заполнение его полей           }
    NewE^.Inf.Adr:=Z.Adr; ;
    NewE^.Next:=nil;
    Right:=NewE; { Right - указатель хвоста очереди }
    Left:=NewE;{ Left - указатель головы очереди }
    While True Do            {Повторение этих же действий     }
     Begin
      Writeln('Введите фамилию и инициалы грузополучателя');
      Readln(Z.Fio);
      Writeln('Введите адрес грузополучателя');
      Readln(Z.Adr);
      If Z.Fio='*' Then Exit;
      New(NewE);
      NewE^.Inf.Fio:=Z.Fio;
      NewE^.Inf.Adr:=Z.Adr;
      NewE^.Next:=Nil;
      Right^.Next:=NewE; {Связь с предыдущим элементом}
      Right:=NewE          {Перемещение указателя хвоста очереди}
     End
End;

Procedure Dob;  {Добавление элемента в конец очереди}
Begin
    Writeln('Введите фамилию и инициалы грузополучателя');
    Readln(Z.Fio);
    Writeln('Введите адрес грузополучателя');
    Readln(Z.Adr);
    If Z.Fio='*' Then Exit;
    New(NewE);
    NewE^.Inf.Fio:=Z.Fio;
    NewE^.Inf.Adr:=Z.Adr;
    NewE^.Next:=nil;
    If Right=Nil Then          {Если добавляется первый элемент, то}
      Left:=NewE               {инициализировать указатель головы}
    Else
```

```
      Right^.Next:=NewE; {иначе добавить в конец очереди}
    Right:=NewE;
End;

Procedure Udal; {Процедура исключения элемента}
Begin
  Writeln('Исключается головной элемент очереди');
  Writeln('Нажмите клавишу Enter');
  Readln;
  If Left<>Nil Then    {Если очередь не пустая, то}
  Begin
    Temp:=Left;        {запоминаем указатель на  голову очереди}
    Left:=Left^.Next;  {указатель головы смещаем на 2-ой элемент}
    Dispose(Temp);  {освобождаем память от головного элемента}
    If Left=Nil Then   {Если удалили последний элемент, то}
      Right:=Nil;       {указатель на конец очереди равен Nil}
  End
End;

Procedure Prosmotr; {Процедура просмотра очереди}
Var i:integer;      {Просмотр выполняется от головы к хвосту}
Begin
    Writeln('Очередь содержит следующие элементы:');
    Temp:=Left;
    i:=1;
    While Temp<>nil Do
     Begin
       Writeln(i,' ',Temp^.Inf.Fio,' ',Temp^.Inf.Adr);
       Temp:=Temp^.Next;
       i:=i+1;
    End;
    Writeln('Нажмите клавишу Enter');
    Readln;
End;
{==============основная программа===============}
begin
     Right:=Nil;
     Left:=Nil;
     Repeat
     ClrScr; { очистка экрана }

   {вывод на экран пунктов меню}
     Writeln('1-Организация очереди');
     Writeln('2-Добавление элемента в очередь');
     Writeln('3-Удаление элемента из очереди');
```

```
    Writeln('4-Просмотр очереди');
    Writeln('5-Выход');
    Writeln('-----------------------------');
    Writeln('Нажмите клавишу от 1 до 5');

    Key:=ReadKey; {считывание кода нажатой клавиши}
    Case Key Of { вызов необходимой процедуры по номеру}
     '1':Org;
     '2':Dob;
     '3':Udal;
     '4':Prosmotr;
     End
    Until Key='5' {выход из программы}
End.
```

4. Порядок выполнения работы

1. Выбрать вид линейного списка, подходящий для решения задачи.
2. Разработать алгоритм решения задачи, разбив его на отдельные процедуры и функции, так, как указано в варианте задания.
3. Разработать программу на языке Pascal.
4. Разработать тестовые примеры, которые предусматривают проверку корректности работы программы для пустых списков.
5. Выполнить отладку программы.
6. Получить результаты работы программы для различных режимов ее работы.

5. Содержание отчета

Цель работы, вариант задания, структурные схемы процедур с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, выводы.

6. Контрольные вопросы

1. Чем отличаются статические переменные от динамических переменных?
2. Как осуществляется задание ссылочных переменных?
3. Для чего используется переменная с указателем?
4. Каково назначение процедур New и Dispose, GetMem и FreeMem?
5. Какие операции определены над ссылочными переменными?
6. Какие операции определены над переменными с указателем?
7. Что понимают под линейной структурой данных?

8. Что называется последовательной структурой и списковой структурой данных?
9. Определите различные виды линейных односвязных списков.
10. Как описать элемент списковой структуры на языке Паскаль?
11. Напишите на языке Паскаль процедуры для работы с однонаправленными списками:
    - создания списка;
    - добавления элемента в список;
    - удаления элемента из списка.

ЛАБОРАТОРНАЯ РАБОТА №10 ПРОГРАММИРОВАНИЕ НЕЛИНЕЙНЫХ СТРУКТУР ДАННЫХ

1. Цель работы

Изучение нелинейных структур данных и приобретение навыков разработки и отладки программ, использующих древовидные структуры. Исследование особенностей работы с поисковыми бинарными деревьями.

2. Варианты заданий

Представить приведенную в предыдущей работе таблицу в виде бинарного дерева. Написать процедуры создания и обхода дерева, а также одну из процедур или функций, приведенных ниже. Значения полей и количество записей в таблице студент выбирает самостоятельно. Исходные данные в разрабатываемой программе должны вводиться из предварительно подготовленного файла. Признаком окончания данных должна служить специальная запись.

1. Таблица 1. Процедуру, которая присваивает параметру E элемент из самого левого листа непустого дерева T.
2. Таблица 2. Процедуру, которая присваивает параметру E элемент из самого левого листа непустого дерева T.
3. Таблица 3. Процедуру, которая присваивает параметру E элемент из самого левого листа непустого дерева T.
4. Таблица 1. Процедуру, которая определяет уровень, на котором находится элемент E в дереве T.

5. Таблица 2. Процедуру, которая определяет уровень, на котором находится элемент E в дереве T.

6. Таблица 3. Процедуру, которая определяет уровень, на котором находится элемент E в дереве T.

7. Таблица 1. Процедуру, которая вычисляет среднее арифметическое всех элементов непустого дерева T (по одному из полей таблицы, которое имеет числовое значение)

8. Таблица 2. Процедуру, которая вычисляет среднее арифметическое всех элементов непустого дерева T (по одному из полей таблицы, которое имеет числовое значение)

9. Таблица 3. Процедуру, которая вычисляет среднее арифметическое всех элементов непустого дерева T (по одному из полей таблицы, которое имеет числовое значение)

10. Таблица 1. Процедуру, которая заменяет в дереве T все элементы меньшие, чем некоторое положительное число A, на это число (по одному из полей таблицы, которое имеет числовое значение).

11. Таблица 2. Процедуру, которая заменяет в дереве T все элементы меньшие, чем некоторое положительное число A, на это число (по одному из полей таблицы, которое имеет числовое значение).

12. Таблица 3. Процедуру, которая заменяет в дереве T все элементы меньшие, чем некоторое положительное число A, на это число (по одному из полей таблицы, которое имеет числовое значение).

13. Таблица 1. Процедуру, которая печатает элементы всех листьев дерева T.

14. Таблица 2. Процедуру, которая печатает элементы всех листьев дерева T.

15. Таблица 3. Процедуру, которая печатает элементы всех листьев дерева T.

16. Таблица 1. Процедуру, которая находит в непустом дереве T длину (число ветвей) пути от корня до вершины с элементом E.

17. Таблица 2. Процедуру, которая находит в непустом дереве T длину (число ветвей) пути от корня до вершины с элементом E.

18. Таблица 3. Процедуру, которая находит в непустом дереве T длину (число ветвей) пути от корня до вершины с элементом E.

19. Таблица 1. Процедуру, которая подсчитывает число вершин на n-ом уровне непустого дерева T.

20. Таблица 2. Процедуру, которая подсчитывает число вершин на n-ом уровне непустого дерева T.

21. Таблица 3. Процедуру, которая подсчитывает число вершин на n-ом уровне непустого дерева T.

22. Таблица 1. Написать рекурсивную процедуру или функцию, которая определяет, входит ли элемент в дерево T.

23. Таблица 2. Написать рекурсивную процедуру или функцию, которая определяет, входит ли элемент в дерево T.

24. Таблица 3. Написать рекурсивную процедуру или функцию, которая определяет, входит ли элемент в дерево Т.

25. Таблица 1. Написать рекурсивную процедуру или функцию, которая определяет число вхождений элемента Е в дерево Т.

26. Таблица 2. Написать рекурсивную процедуру или функцию, которая определяет число вхождений элемента Е в дерево Т.

27. Таблица 3. Написать рекурсивную процедуру или функцию, которая определяет число вхождений элемента Е в дерево Т.

28. Таблица 1. Написать рекурсивную процедуру или функцию, которая вычисляет сумму элементов (по одному из полей таблицы) непустого дерева Т.

29. Таблица 2. Написать рекурсивную процедуру или функцию, которая вычисляет сумму элементов (по одному из полей таблицы) непустого дерева Т.

30. Таблица 3. Написать рекурсивную процедуру или функцию, которая вычисляет сумму элементов (по одному из полей таблицы) непустого дерева Т.

3. Основные сведения и методические указания

3.1. Бинарные деревья и алгоритмы их обработки

Нелинейные структуры предназначены для отображения связей подчиненности элементов данных. Примером нелинейных структур являются иерархические древовидные структуры. В представлении данных наибольшее распространение получили *бинарные деревья*. Любая вершина бинарного дерева связана только с вершиной ближайшего высшего уровня и с двумя вершинами низшего уровня.

Вершина (узел) дерева, которая не имеет вышестоящей вершины (предка), называется *корнем*. Считается, что корень дерева расположен на первом уровне. Каждая вершина уровня k имеет одного предка на уровне k-1. Максимальный уровень дерева называется его *глубиной*. Среди любых пар непосредственно связанных вершин можно выделить предка и дочернюю вершину. Вершина дерева, которая не имеет дочерних вершин, называется *листом*.

В связном списке каждый элемент содержит указатель на другой элемент. Бинарное дерево похоже на связный список с тем отличием, что каждый узел содержит два указателя: на левое бинарное поддерево и правое бинарное поддерево. Ниже приведен пример описания узла бинарного дерева:

```
TYPE
   TreePtr=^Tree;
   Tree=Record
     Data: String;     {тип значения, хранимый в вершине}
     Left,Right: TreePtr  {указатели на левое и правое поддерево}
```

```
End;
```

Наиболее распространенным условием организации бинарных деревьев является упорядоченность. Элементы дерева в этом случае снабжаются ключевыми признаками. Каждый элемент в упорядоченном дереве имеет на своей левой ветви элементы с меньшими, чем у него, значениями ключей, а на правой ветви элементы с большими значениями ключей.

Упорядоченные бинарные деревья обеспечивают быстрый поиск записи по ее ключу. Такие деревья называют *деревьями бинарного поиска*. Ниже приведен пример описания функции поиска записи по ключу в бинарном дереве.

```
Function Search(Root:TreePtr; Node:string):TreePtr;
{ Node – ключевое (поисковое) значение, например типа string;
  Root – указатель на корень дерева (поддерева)}
Var Found:Boolean; {признак успешности поиска}

{ Если дерево содержит вершину Node то функция
  возвращает указатель на эту вершину}

 Begin
   Found:=False;
   While (Root<>Nil) and (not Found) Do
   {повторять цикл пока не дошли до листа или не нашли значение}

     If Root^.Data=Node Then  {Проверить значение вершины}
       Found:=True            {значение найдено}
     Else                     {значение не найдено}
       If Root^.Data>Node Then
         Root:=Root^.Left;    {поиск в левом поддереве}
       Else
         Root:=Root^.Right;   {поиск в правом поддереве}
  Search:=Root;
End;
```

Рассмотрим алгоритм построения упорядоченных бинарных древовидных структур. Пусть имеется неупорядоченный массив A[1],A[2],...,A[n]. Первый элемент A[1] назовем корнем. Сравним второй элемент A[2] с первым элементом A[1]. Если A[2]>A[1], запишем в правый указатель элемента A[1] адрес элемента A[2], иначе адрес A[2] запишем в левый указатель. На следующем шаге сравниваем A[3] с A[1]. При A[3]>A[1] сравниваем A[3] с элементом адрес, которого хранится в правом указателе, а если этого адреса нет, то записываем в правый указатель элемента A[1] адрес элемента A[3]. Если же в правом указателе элемента A[1] указан адрес элемента A[2], то сравниваем A[3] с A[2] и заносим адрес элемента A[3] в правый или левый ука-

затель элемента A[2] в зависимости от результатов сравнения. Процесс повторяется для каждого нового элемента. В этом случае новые элементы всегда присоединяются к листьям дерева. Алгоритм добавления элемента в дерево может быть сформулирован рекурсивно:

```
Function AddTree(Top:TreePtr; NewNode: String):TreePtr;
                {Top – указатель на корень дерева (поддерева);
                 NewNode – добавляемое значение элемента}
Begin
    If Top=Nil Then {указатель нулевой, если дошли до листа}
      Begin
        New(Top);          {выделить память для нового узла}
        Top^.Data:=Node;{записать в узел значение}
        Top^.Left:=Nil;     {указатели на поддеревья пустые}
        Top^.Right:=Nil;
      End
     Else
      IF Top^.Data>NewNode Then {поиск места вставки}
        Top^.Left:=AddTree(Top^.Left,NewNode)
      Else
        Top^.Right:=AddTree(Top^.Right,NewNode);
     AddTree:=Top
End;
```

Эффективность поиска в бинарном дереве в значительной степени зависит от его симметричности. Под *симметричным* понимается дерево, которое состоит из *n* уровней, причем *n-1* уровень занят полностью. Если листья дерева располагаются только на двух соседних уровнях *n-1* и *n*, а *n-1=trunc(log2 M)*, то такое дерево называется *выровненным* (М-количество вершин дерева).

Формирование упорядоченного выровненного дерева можно выполнить с помощью следующего алгоритма. Исходные записи должны иметь вид упорядоченного массива. За корень дерева принимается средняя запись массива, которая разделяет массив примерно на две равные части. Средние записи в обеих частях массива образуют вершины второго уровня, а массив оказывается разделенным на 4 части. Средние записи каждой из четырех частей помещаются на третьем уровне бинарного дерева. Процесс продолжается до тех пор, пока все записи массива не будут внесены в дерево.

Чтобы распечатать дерево бинарного поиска в заданном порядке используют различные *стратегии обхода дерева*:

- обход сверху;
- обход слева направо;
- обход снизу.

Для обхода дерева применяют процедуры:

- обработки корня дерева или поддерева;

- обход левого поддерева;
- обход правого поддерева.

Если перечисленные процедуры выполняются в порядке 1-2-3, то выполняется обход сверху; если в порядке 2-1-3, то выполняется обход слева направо; если в порядке 2-3-1, то выполняется обход снизу. Для печати дерева с сохранением относительного порядка используют обход слева направо. Пример процедуры обхода двоичного дерева слева направо:

```
Procedure LR(Top:TreePtr);
Begin
  If Top<>Nil Then
     Begin
       LR(Top^.Left);
       Writeln(Top^.Data);
       LR(Top^.Right)
      End
End;
```

3.2. Пример программы обработки бинарного дерева

Пусть требуется написать процедуры создания, добавления листа в бинарное дерево, просмотра бинарного дерева, отображения структуры дерева, а также рекурсивную процедуру, которая подсчитывает число вершин на n-ом уровне непустого дерева T (корень считать вершиной 1-го уровня).

Ниже приведен текст соответствующей программы с комментариями. С каждым узлом дерева связана запись, состоящая из фамилии и адреса грузополучателя. Дерево упорядочено по фамилиям.

```
Program TreeProcess;
Uses Crt;
Type
    Zap=Record  {Описание  записи о грузополучателе}
        FIO:string[15];
        Adr:string[30]
       End;

    TreePtr=^Tree; {Описание узла дерева}
    Tree=Record
        Data:Zap;
        Left,Right:TreePtr
        End;

Var    Top:Treeptr;
          Z:Zap;
  Level,N,i:Integer;
```

```
    Number:Integer;

 {Функция добавляющая лист к дереву}
 Function AddTree (Top:TreePtr;Newnode:Zap):TreePtr;
 Begin
      If Top=Nil THEN
        Begin
          New(Top);
          Top^.Data:=Newnode;
          Top^.Left:=Nil;
          Top^.Right:=Nil;
        End
       Else
        If Top^.Data.Fio>Newnode.Fio Then
           Top^.Left:=AddTree(Top^.Left,Newnode)
        Else
           Top^.Right:=AddTree(Top^.Right,Newnode);
       Addtree:=Top
 End;

Procedure OrgTree;
 Begin
  Writeln('Выполняется процедура организации дерева');
  Writeln('Для выхода из процедуры вводите символ * ');
  Writeln('========================================= ');
  Top:=nil;
  While True Do
   Begin
    Writeln('Введите фамилию и инициалы грузополучателя');
    Readln(Z.Fio);
    Writeln('Введите адрес грузополучателя');
    Readln(Z.Adr);
    If Z.Fio='*' Then Exit;
    Top:=Addtree(Top,Z);
   End
 End;

Procedure DobL;
 Begin
   Writeln('Выполняется процедура добавления листа');
   Writeln('Для выхода из процедуры вводите символ * ');
   Writeln('========================================= ');
   Writeln('Введите фамилию и инициалы грузополучателя');
   Readln(Z.Fio);
   Writeln('Введите адрес грузополучателя');
```

```
  Readln(Z.Adr);
  If Z.Fio='*' Then Exit;
  Top:=Addtree(Top,Z);
End;

Procedure Prosmotr(Top:TreePtr);
{Процедура просмотра значений узлов дерева слева направо}
Begin
  If Top<>Nil Then
     Begin
       Prosmotr(Top^.Left);
       Writeln(i,' ',Top^.Data.Fio,' ',Top^.Data.Adr);
       i:=i+1;
       Prosmotr(Top^.Right)
     End;
End;

Procedure Otobr(Top:TreePtr;Otstup:Integer);
{Процедура отображения структуры дерева.
 Дерево отображается повернутым на 90 градусов против
 часовой стрелки. Узлы дерева, находящиеся на одном
 уровне, отображаются с  одинаковым отступом от края
 экрана.}

Begin
  If Top<>Nil Then
     Begin
       Otstup:=Otstup+3;
       Otobr(Top^.Right,Otstup);
       Writeln(' ':Otstup,Top^.Data.Fio);
       Otobr(Top^.Left,Otstup);
     End
End;


Procedure NodeCount(Top:TreePtr; Level:Integer; Var N:Integer);
{Процедура подсчета количества вершин уровня Level}
Begin
   If (Level>=1) and (Top<>Nil) Then
    Begin
      If Level=1 Then N:=N+1;
      NodeCount(Top^.Left,Level-1,N);
      NodeCount(Top^.Right,Level-1,N);
    End
End;
```

```
{==============основная программа===============}
Begin
   { цикл, обеспечивающий вывод на экран  пунктов меню}
   Repeat
    ClrScr; { очистка экрана }
    Writeln('1-Организация двоичного дерева');
    Writeln('2-Добавление листа к дереву');
    Writeln('3-Просмотр дерева');
    Writeln('4-Подсчет количества вершин на n-ом уровне');
    Writeln('5-Выход');
    Writeln('------------------------------');
    Writeln('Введите номер пункта меню');
    Readln(Number);
    Case Number Of { вызов необходимой процедуры по номеру}
     1:OrgTree;
     2:Dobl;
     3:Begin
       Writeln('Выполняется процедура просмотра дерева');
       Writeln('======================================== ');
       i:=0;
       Prosmotr(Top);
       Otobr(Top,1);
       Writeln('Нажмите клавишу Enter');
       Readln
      End;
     4:Begin
       Writeln('Выполняется процедура подсчета количества');
       Writeln('вершин на n-ом уровне');
       Writeln('======================================== ');
       Write('Введите значение уровня-->');
       Read(Level);
       N:=0;
       NodeCount(Top,Level,N);
       Writeln;
       Writeln('На уровне ',Level,' находится ',N,' вершин');
       Writeln('Нажмите клавишу Enter');
       ReadKey
      End;
    End;
   Until Number=5; {выход из цикла, если введено 5}
End.
```

4. Порядок выполнения работы

1. Описать элемент бинарного дерева в соответствии с вариантом задачи.
2. Разработать алгоритм решения задачи, разбив его на отдельные процедуры и функции, так, как указано в варианте задания.
3. Разработать программу на языке Pascal.
4. Разработать тестовые примеры, которые предусматривают проверку корректности работы программы в разных режимах.
5. Выполнить отладку программы.
6. Получить результаты работы программы для различных режимов ее работы.

5. Содержание отчета

Цель работы, вариант задания, структурные схемы процедур с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, выводы.

6. Контрольные вопросы

1. Что понимают под нелинейной структурой данных?
2. Что называется бинарным деревом данных?
3. Как построить упорядоченное бинарное дерево?
4. Отчего зависит эффективность поиска в бинарном дереве?
5. Что понимают под симметричным и выровненным деревом?
6. Как формируется выровненное дерево?
7. Напишите на языке Паскаль процедуры для работы с бинарными деревьями:
   - создания упорядоченного бинарного дерева;
   - добавления элементов в дерево;
   - удаления листа дерева;
   - процедуры обхода дерева.

ЛАБОРАТОРНАЯ РАБОТА №11
ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ОБЪЕКТОВ И МОДУЛЕЙ

1. Цель работы

Изучение базовых понятий объектно-ориентированного программирования, приобретение навыков разработки и отладки программ, использующих

модули и объекты. Исследование особенностей работы с поисковыми бинарными деревьями.

2. Варианты заданий

Создать модуль, содержащий описание объекта, который представляет бинарное дерево в соответствии с вариантом из предыдущей лабораторной работы. Объект должен обладать возможностью добавления новых элементов, удаления существующих, поиска элемента по ключу, обхода дерева, а также выполнять дополнительные операции в соответствии с вариантом задания.

Написать программу для представления таблицы, заданной в предыдущей лабораторной работе в виде бинарного дерева. Программа должна содержать меню, позволяющее проверить все методы объекта. Предусмотреть возможность ввода данных из файла и с клавиатуры.

3. Основные сведения и методические указания

3.1. Модули

Модуль в языке Turbo Pascal – это отдельно хранимая и независимо компилируемая программа. Модуль содержит коллекцию программных ресурсов, которые можно использовать в других программах. Модуль содержит описания типов данных, переменных и других объектов, а также подпрограммы. Модуль сам по себе не является выполнимой программой – его объекты используются другими программными единицами. Подпрограммы, входящие в модуль, можно написать, отладить и откомпилировать один раз, а использовать многократно. Доступ к ресурсам модуля из других программ обеспечивает оператор uses, в котором указывается имя модуля. Этот оператор размещается в разделе описаний программы сразу после заголовка. Если в программе используется не один модуль, а несколько, необходимо указать имена всех модулей, перечислив их через запятую. Исключением является модуль System, ссылка на который необязательна. Этот модуль содержит, в частности, процедуры файлового ввода-вывода, процедуры и функции для работы со строками и некоторые другие.

Модуль начинается заголовком:

unit <имя модуля>;

Имя модуля выбирается в соответствии с правилами языка Pascal. Файл, содержащий модуль, обязан иметь то же имя, что и модуль. Структура модуля изображена на рисунке 3.

Unit <имя модуля>
Interface
Интерфейсная секция
Implementation
Секция реализации
Секция инициализации

Рисунок 3 – Структура модуля

Модуль состоит из трех секций. Первая — *интерфейсная секция* — содержит описания типов, переменных и других объектов данных, которые можно использовать в других программах или модулях. Она начинается с зарезервированного слова interface. Для процедур и функций в интерфейсной секции указывают только заголовки, а их полные описания содержатся в секции реализации. *Секция реализации* начинается с зарезервированного слова implementation. Все описания, содержащиеся в секции реализации, являются локальными, их область действия — данный модуль. Здесь же содержатся полные описания функций и процедур модуля. Последняя часть модуля — *секция инициализации* (начинается словом begin, а заканчивается словом end). Она может быть пустой или включать в себя исполняемые операторы, реализующие необходимые действия по инициализации (например, присваиванию начальных значений переменным) модуля. Пример модуля будет приведен ниже.

В результате компиляции модуля на диске создается файл с расширением tpu.

3.2. Объекты. Основные понятия

Программа, написанная с использованием методологии объектно-ориентированного программирования (ООП), состоит из объектов, которые взаимодействуют между собой. *Объект* – это реальная либо абстрактная сущность, которая моделирует часть предметной области программы. С точки зрения программиста объект состоит из атрибутов и методов. *Атрибуты* описывают свойства объекта, а *методы* – характерное для объекта поведение. Поэтому программная реализация объекта представляет собой объединение данных (атрибутов) и процедур их обработки (методов). В языке Turbo Pascal для описания объектов применяется тип object, который можно считать обобщением структурного типа record. Переменные объектового типа называются *экземплярами* объекта.

В отличие от типа «запись» объектовый тип содержит не только поля, описывающие данные, но также процедуры и функции, описания которых входят в описание объекта. Эти процедуры и функции называются *методами*. Методам объекта доступны его атрибуты (поля данных). Непосредственно в объектовом типе задаются только заголовки методов, а их полные описания должны быть сделаны вне объектового типа. Приведем пример описания объекта:

```
Type
Location = object
             X,Y : Integer;
             procedure Init(InitX, InitY: Integer);
             function GetX : Integer;
             function GetY : Integer;
             end;
```

Объект описывается с помощью зарезервированных слов object и end, между которыми находятся описания полей данных (атрибутов) и методов. В приведенном примере объект содержит два поля X и Y для хранения значений координат, а также заголовки двух функций (GetX, GetY) и процедуры (Init) — методов данного объекта. В отличие от других описаний описание объектного типа может находиться только на самом верхнем уровне программной единицы (программы, модуля), в которой используется этот тип. В разделе описаний процедур и функций такое описание содержаться не может.

Зарезервированное слово private позволяет ограничить доступ к полям объекта. В следующем примере доступ к полям X и Y возможен только посредством методов объектного типа Location:

```
Type  Location = object
                   procedure Init(InitX, InitY : Integer);
                   function GetX : Integer;
                   function GetY : Integer;
                   private
                    X, Y : Integer;
                   end;
```

В секции private могут находиться и методы объекта.

Полное описание методов, то есть описание их реализации, должно находиться после описания объекта (обычно в секции реализации модуля). В этом случае имя метода является составным и складывается из имени объекта и метода, разделяемых точкой:

```
procedure Location.Init(InitX, InitY : Integer);
begin
  X := InitX;
  Y := InitY;
end;

function Location.GetX : Integer;
begin
  GetX := X;
end;
```

```
function Location.GetY : Integer;
begin
  GetY := Y;
end;
```

После того как объект описан, в программе можно использовать его экземпляры, то есть переменные указанного объектного типа:

```
Var   GrFigurePosition: Location;
```

3.3. Инкапсуляция

Инкапсуляция – это механизм, который позволяет защитить данные (атрибуты) и методы объекта от некорректного использования. В соответствии с принципами инкапсуляции атрибуты объекта не могут быть доступными непосредственно. Доступ к атрибутам (полям данных) осуществляется *только через соответствующие методы объекта.* Например, чтение значений атрибутов X и Y экземпляра объекта GrFigurePosition возможно только с помощью вызова его методов GetX и GetY:

```
Xpos:= GrFigurePosition.X;
Ypos:= GrFigurePosition.Y;
```

При этом процедура инициализации Init и функции GetX и GetY уже не существуют как отдельные самостоятельные единицы. Это неотъемлемые части объектного типа Location. Если в программе имеется несколько экземпляров объектов (переменных) указанного типа, то для хранения данных каждого экземпляра объекта резервируется своя собственная область памяти, а указатели на точки входа в процедуру и функции являются общими. Вызов каждого метода возможен только с помощью составного имени, явно указывающего, для обработки каких полей данных предназначен данный метод.

3.4. Наследование

Объектно-ориентированный подход позволяет определить новый объект, как потомок другого объекта:

```
Type
  Point = object(Location)
              Visible : Boolean;
              procedure Show;
              procedure Hide;
```

```
        function IsVisible : Boolean;
        procedure MoveTo(NewX, NewY : Integer);
end;
```

В объектовом типе Point присутствуют все поля и методы типа Location. Описанное свойство называется *наследованием*, то есть потомок наследует свойства родителя. Наследником здесь является объект Point, представляющий точку на экране компьютера, а родителем — объект Location.

Такое определение объектового типа содержит следующие элементы:

- поля X и Y, унаследованные от Location;
- собственное поле Visible;
- унаследованные методы Init, GetX, GetY;
- новые методы Show, Hide, IsVisible, MoveTo.

Что касается совместимости объектных типов, то экземпляру объекта родительского типа можно присваивать значения объекта-наследника, но не наоборот.

3.5. Виртуальные методы и полиморфизм

Связь между *виртуальными* методами и вызывающими их процедурами устанавливается во время выполнения программы (это называется *поздним связыванием,* в отличие от *раннего связывания,* когда связь устанавливается во время компиляции). В описании виртуального метода после заголовка метода указывается зарезервированное слово virtual. Заголовки виртуальных методов родителя и наследника должны совпадать. Инициализация экземпляра объекта, имеющего виртуальные методы, выполняется с помощью специального метода — *конструктора.* Конструктор присваивает полям объекта начальные значения и выполняет другие необходимые действия по инициализации объекта. В заголовке метода-конструктора слово procedure заменяется словом constructor. Действия, обратные действиям конструктора, выполняет еще один специальный метод – *деструктор.* Его описание начинается со слова destructor.

Конструктор создает указатель на *таблицу виртуальных методов,* которая содержит адреса всех виртуальных методов объекта и в дальнейшем используется для их поиска. При вызове виртуального метода по его имени определяется адрес, а затем по этому адресу передается управление. Если имеется несколько экземпляров объектов одного типа, то конструктор должен быть вызван отдельно для каждого экземпляра объекта при его инициализации. В противном случае возможен сбой в работе программы.

У каждого объектного типа имеется собственная таблица виртуальных методов, что позволяет одному и тому же оператору вызывать разные процедуры. Таким образом, вызов одного и того же метода будет работать по-разному, в зависимости от того какой объект его вызывает. Это свойство называют *полиморфизмом.*

Конструктор и деструктор могут быть и «пустыми», то есть не содержать операторов. Весь необходимый код в этом случае создается автоматически.

3.6. Динамическое создание объектов

Для работы с динамическими объектами используется расширенный синтаксис процедур New и Dispose. Обе процедуры в этом случае содержат в качестве второго параметра имя конструктора или деструктора объекта:

```
New(P, <имя конструктора>)
Dispose(P,<имя деструктора>).
```

Здесь P — указатель на переменную объектного типа.

Действие процедуры New в случае расширенного синтаксиса равносильно действию следующих операторов:

```
New(P); P^.<имя конструктора>;
```

Вызов процедуры Dispose эквивалентен такой последовательности:

```
P^.<имя деструктора; Dispose(P);
```

3.7. Пример модульной программы для работы с объектом «очередь»

Требуется создать модуль, содержащий описание объекта, который представляет очередь. Объект должен обладать возможностью организации очереди, добавления элемента в очередь, удаления элемента из очереди, просмотра очереди. Написать программу для представления данных о грузополучателях в виде очереди. Программа должна содержать меню, позволяющее проверить работу всех методов объекта.

В качестве основы для данной программы возьмем пример программы работы с очередью из лабораторной работы №3. При этом все необходимые типы данных, в том числе и объект, представляющий очередь, опишем в модуле DynQuery. Данный модуль будет подключаться к программе в операторе Uses. Программа будет выводить на экран соответствующие пункты меню и вызывать необходимые методы объекта.

В программе создается экземпляр объекта Query, описание которого приведено в модуле DynQuery. Данный объект содержит описание двух полей-указателей на начало (Left) и конец очереди (Right), а также описание всех методов, необходимых для работы с очередью. При этом доступ к значениям указателей имеют только методы объекта, так как указатели описаны в секции Private.

```
Unit DynQuery; {Модуль для работы с очередью}

{=====интерфейсная секция модуля================}
```

```
Interface
Type
    Data=  Record  {Описание  записи о грузополучателе}
                FIO:string[15];
                Adr:string[30]
               End;

    Ukaz=^ElQuery; {Описание указателя на элемент очереди}
    ElQuery=Record
                 Inf:Data;
                 Next:Ukaz
              End;

    Query=object {Описание объекта}
              Procedure Init;    {Метод инициализирующий указатели}
              Procedure Org;          {Метод организации очереди}
              Procedure Dob;          {Метод добавляющий элемент}
              Procedure Udal;         {Метод исключающий элемент}
              Procedure Prosmotr;   {Метод для просмотра очереди}
              Private                       {Закрытые поля объекта}
              Left:Ukaz;                {Указатель на голову очереди}
              Right:Ukaz;               {Указатель на хвост очереди}
              End;

{======секция реализации модуля==================}
Implementation

Procedure Query.Init;  {Метод инициализирующий указатели}
Begin
       Right:=Nil;         {Начальное значение указателей Nil}
       Left:=Nil;
End;

Procedure Query.Org; {Метод (процедура)организации очереди}
Var   Z:Data;               {Запись, добавляемая в очередь}
       NewE:Ukaz;
Begin
    Writeln('Выполняется процедура организации очереди');
    Writeln('Для выхода из процедуры вводите символ * ');
    Writeln('========================================= ');
    Writeln('Введите фамилию и инициалы грузополучателя');
    Readln(Z.Fio);
    Writeln('Введите адрес грузополучателя ');
    Readln(Z.Adr);
    If Z.Fio='*' Then Exit;  {Выход из процедуры при вводе '*'}
```

```
    New(NewE);              {Создание нового элемента      }
    NewE^.Inf.Fio:=Z.Fio;    {Заполнение его полей          }
    NewE^.Inf.Adr:=Z.Adr; ;
    NewE^.Next:=nil;
    Right:=NewE; { Right - указатель хвоста очереди }
    Left:=NewE;   { Left - указатель головы очереди }
    While True Do           {Повторение этих же действий    }
     Begin
      Writeln('Введите фамилию и инициалы грузополучателя');
      Readln(Z.Fio);
      Writeln('Введите адрес грузополучателя');
      Readln(Z.Adr);
      If Z.Fio='*' Then Exit;
      New(NewE);
      NewE^.Inf.Fio:=Z.Fio;
      NewE^.Inf.Adr:=Z.Adr;
      NewE^.Next:=Nil;
      Right^.Next:=NewE; {Связь с предыдущим элементом}
      Right:=NewE          {Перемещение указателя хвоста очереди}
     End
End;

Procedure Query.Dob;  {Добавление элемента в конец очереди}
Var    Z:Data;     {Запись, добавляемая в очередь}
       NewE:Ukaz;
Begin
    Writeln('Введите фамилию и инициалы грузополучателя');
    Readln(Z.Fio);
    Writeln('Введите адрес грузополучателя');
    Readln(Z.Adr);
    If Z.Fio='*' Then Exit;
    New(NewE);
    NewE^.Inf.Fio:=Z.Fio;
    NewE^.Inf.Adr:=Z.Adr;
    NewE^.Next:=nil;
    If Right=Nil Then    {Если добавляется первый элемент, то}
      Left:=NewE         {инициализировать указатель головы   }
    Else                 {иначе                               }
      Right^.Next:=NewE;   {связать конец очереди с элементом}
    Right:=NewE;
End;

Procedure Query.Udal; {Метод (процедура) исключения элемента}
Var Temp:Ukaz;
Begin
```

```
  Writeln('Исключается головной элемент очереди');
  Writeln('Нажмите клавишу Enter');
  Readln;
  If Left<>Nil Then    {Если очередь не пустая, то}
  Begin
     Temp:=Left;           {запоминаем указатель на  голову очереди}
     Left:=Left^.Next;     {смещаем его на второй элемент}
     Dispose(Temp);  {освобождаем память от головного элемента}
     If Left=Nil Then   {Если удалили последний элемент, то}
        Right:=Nil;         {указатель на конец очереди равен Nil}
  End
End;

Procedure Query.Prosmotr; {Просмотр очереди}
Var    i:integer;      {Просмотр выполняется от головы к хвосту}
         Temp:Ukaz;
Begin
     Writeln('Очередь содержит следующие элементы:');
     Temp:=Left;
     i:=1;
     While Temp<>nil Do
      Begin
        Writeln(i,' ',Temp^.Inf.Fio,' ',Temp^.Inf.Adr);
        Temp:=Temp^.Next;
        i:=i+1;
     End;
     Writeln('Нажмите клавишу Enter');
     Readln;
End;

{======секция инициализации модуля===============}
Begin
End.


Program DemoQueryObject;
Uses Crt, DynQuery; {Подключение модулей}
Var    Key:Char;          {номер пункта меню программы}
         Q:Query;           {Создание экземпляра объекта}
begin
      Q.Init;                {Инициализация указателей очереди}
      Repeat
        ClrScr;                      { очистка экрана }
        {--------------------вывод на экран пунктов меню----------------}
        Writeln('1-Организация очереди');
```

```
        Writeln('2-Добавление элемента в очередь');
        Writeln('3-Удаление элемента из очереди');
        Writeln('4-Просмотр очереди');
        Writeln('5-Выход');
        Writeln('-----------------------------');
        Writeln('Нажмите клавишу от 1 до 5');
        Key:=ReadKey; {считывание кода нажатой клавиши}
        Case Key Of { вызов методов по номеру пункта меню}
              '1':Q.Org;
              '2':Q.Dob;
              '3':Q.Udal;
              '4':Q.Prosmotr;
        End
    Until Key='5' {выход из программы}
End.
```

4. Порядок выполнения работы

1. Разработать модуль, содержащий описание вспомогательных типов и объект, представляющий бинарное дерево, взяв в качестве основы программу из предыдущей лабораторной работы.
2. Разработать основную программу, в которой создается экземпляр объекта и обеспечивается вызов необходимых методов объекта.
3. Выполнить компиляцию программы. При этом в интегрированной среде в подпункте меню Options→Directories необходимо указать пути, где будут размещаться .tpu и .exe файлы, а также исходный текст модуля (unit directories), например: c:\bp\work.
4. Разработать тестовые примеры, которые предусматривают проверку корректности работы программы в разных режимах.
5. Выполнить отладку программы.
6. Получить результаты работы программы для различных режимов ее работы.

5. Содержание отчета

Цель работы, вариант задания, текст модуля и программы с комментариями, тестовые примеры, результаты вычислений, выводы.

6. Контрольные вопросы

1. Что понимают под модулем в языке Turbo Pascal?
2. Назовите составные части модуля и объясните их назначение.
3. Как подключить модуль к программе?

4. Какое требование предъявляется к имени файла, содержащего модуль?
5. Что такое объект? Как его описать на языке Turbo Pascal?
6. Что понимают под методом объекта? Как описываются методы?
7. В какой части модуля обычно описывают объекты?
8. Что такое наследование? Как определить объект-потомок?
9. Как определить экземпляр объекта и присвоить значения его полям?
10. Как определяется виртуальный метод? Какие дополнительные методы необходимо при этом описать?
11. Что такое полиморфизм?
12. Как создать динамический экземпляр объекта?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ковалюк Т.В. Основи программування.—К.: Видавнича група BHV,2005.—384 с.
2. Марченко А.И., Марченко Л.А. Программирование в среде Turbo Pascal 7.0. – К.: Век+,2000.—464 с.
3. Немнюгин С.А. Turbo Pascal: Практикум 2-е изд. – СПб.: Питер, 2005.—268 с.
4. Немнюгин С.А. Turbo Pascal:Учебник 2-е изд. – СПб.:Питер, 2003.—468 с.
5. Н.Вирт Алгоритмы и структуры данных М:Мир,1989г.-360с.