

Документ подписан простой электронной подписью  
Информация о владельце:  
ФИО: Локтионова Оксана Геннадьевна  
Должность: проректор по учебной работе  
Дата подписания: 19.04.2022 14:05:31  
Уникальный программный ключ:  
0b817ca911e6668abb13a5d426d39e5f1c11eabbf73e943df4a4851fda56d089

## МИНОБРАЗОВАНИЯ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

Юго-Западный государственный университет  
(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ

Проректор по учебной работе

Локтионова

2016 г.



## ЗАПИСИ И ФАЙЛЫ

Методические указания по выполнению лабораторной работы  
для студентов направления подготовки 09.03.01

Курск 2016

УДК 621.3

Составитель: Э.И. Ватутин

Рецензент

Кандидат технических наук, доцент *В.С. Панищев*

**Записи и файлы:** методические указания по выполнению лабораторных работ по дисциплине «Программирование» / Юго-Зап. гос. ун-т; сост.: Э.И. Ватутин; Курск, 2016. 22 с.

Методические рекомендации содержат сведения по разработке программ, оперирующих записями и файлами, на современных языках программирования высокого уровня.

Предназначены для студентов направления подготовки 09.03.01 «Информатика и вычислительная техника».

Текст печатается в авторской редакции

Подписано в печать \_\_\_\_\_. Формат 60x84 1/16.

Усл. печ. л.    Уч. – изд.л.    Тираж 30 экз. Заказ    . Бесплатно.

Юго-Западный государственный университет

305040, Курск, ул. 50 лет Октября, 94.

## Содержание

Записи .....	4
Файлы .....	10
Пример программы .....	17
Индивидуальные задания .....	18
Содержание отчета.....	21
Контрольные вопросы .....	21
Библиографический список.....	21

## Записи

Целью работы является получение практических навыков при разработке программ, оперирующих структурными типами данных.

Иногда в программах возникает необходимость совместного хранения группы различных значений. Если тип значений совпадает, то для этого можно использовать массивы, в случае его несовпадения используют другую конструкцию языка – записи (в некоторых языках аналогичные конструкции называются структурами), которые относятся к структурным типам. Очень простым примером, поясняющим использование записей, является рассмотрение простой базы данных сотрудников. Для каждого сотрудника необходимо хранить ряд полей: фамилия, имя, отчество, дата рождения, пол и пр., которые представляют собой переменные различных типов. Можно завести для их хранения группу массивов, один из которых будет хранить имена, другой – даты рождения и т.д., однако более удобным в данном случае будет использование записей, при этом каждому сотруднику будет соответствовать свой элемент массива с набором т.н. *полей*.

В простейшем случае тип запись объявляется в программе следующим образом:

```
Имя_типа = record  
  Список_полей_1: Тип1;  
  Список_полей_2: Тип2;  
  ...  
  Список_полей_N: ТипN;  
end;
```

В списках полей через запятую записываются идентификаторы полей, входящих в состав записи (аналогично объявлению переменных). В пределах записи идентификаторы полей должны быть уникальны. Рассмотрим пример записи, используемой в качестве вспомогательной структуры данных при организации поиска файлов:

```
TSearchRec = record  
  Time, Size, Attr: Integer;
```

```
Name: TFileName;
ExcludeAttr: Integer;
FindHandle: THandle;
FindData: TWin32FindData;
end;
```

Записи можно комбинировать с другими структурными типами, создавая, например, массивы записей или записи с полями типа запись или массив. Однако при определении записи запрещены рекуррентные ссылки на тот же тип данных, т.к. в данном случае компилятор не сможет определить размер переменной объявляемого типа в памяти. Например, приведенная ниже ситуация является ошибочной:

```
type
  TRecord = record
    a: TRecord; // Ошибка: рекуррентные определения типов запрещены!
    b: Integer;
  end;
```

Обращение к полям записи производится через символ «.» следующим образом:

```
Имя_переменной.Имя_поля
```

Например, для приведенной выше записи TSearchRec возможны следующие обращения к ее полям:

```
var
  Rec: TSearchRec;
begin
  Rec.Time := 1;
  Writeln(Rec.Size);
```

Аналогичным образом через точку в Delphi производится обращение к элементам классов (например, к свойствам и событиям компонентов формы).

Для более короткого обращения к полям записей существует специальный оператор присоединения `with`, который позволяет не указывать имя записи при обращении к ее полям. В общем виде он записывается следующим образом:

```
with Имя_переменной_типа_записи1, ..., Имя_переменной_типа_записиN do
  Оператор;
```

При его использовании в составе оператора можно обращаться к именам полей перечисленных записей напрямую без указания имени переменной. С его использованием предыдущий пример может быть оформлен в следующем виде:

```
with Rec do begin
  Time := 1;
  Writeln(Size);
end;
```

При использовании оператора `with` область видимости полей записи частично перекрывает другие области видимости (глобальную, локальную в подпрограмме и т.д.), поэтому при использовании записей обычно стремятся к тому, чтобы имена полей не совпадали с именами элементов перекрываемых областей видимости, иначе все преимущества оператора присоединения теряются, т.к. обращения с использованием символа «.» приходится применять для элементов перекрываемых областей видимости.

С записью целиком разрешается производить только одну операцию – побитовое копирование содержимого. Все остальные действия осуществляются вручную путем обращения к соответствующим полям записей. Например:

```
type
  TRec = record
    X, Y: Integer;
    Z: Extended;
  end;

var
  A, B: TRec;
  C: Integer;

begin
  with A do begin
    X := 1;
    Y := 2;
    Z := 3.5;
  end;

  B := A; // Эквивалентно B.X := A.X; B.Y := A.Y; B.Z := A.Z;

  with B do
    Writeln(X, Y, Z);

  C := A.X*B.X + A.Y*C.Y;
```

end.

Поля записи хранятся в памяти последовательно друг за другом, однако в состав записи может входить т.н. вариантная часть, поля которой располагаются в одной области памяти поверх друг друга. Это может быть использовано, например, для одновременного хранения информации об объектах различного типа или для обращения к простым частям более сложных типов без применения адресной арифметики. Запись с вариантной частью в общем случае записывается следующим образом:

```

type
  Имя_типа = record
    Список_полей1: Тип1;
    Список_полей2: Тип2;
    ...
    Список_полейN: ТипN;
  case Тэг: Тип of
    Список_констант1: (Вариантная_группа_полей1);
    Список_констант2: (Вариантная_группа_полей2);
    ...
    Список_константN: (Вариантная_группа_полейN);
  end;

```

Правилами языка разрешается использовать только одну вариантную часть в составе записи, которая располагается после обычных полей. Переменные, входящие в состав различных вариантных групп полей, располагаются поверх друг друга. Определение полей в составе вариантной группы полей не отличается от определения других полей записи. В составе вариантной части не могут быть использованы некоторые типы, например, длинные строки и динамические массивы, однако вместо этого можно использовать указатели на них. Тэг и списки констант не играют решающей роли и достались в наследство от более старых версий языка Паскаль, поэтому могут использоваться по усмотрению программиста.

В качестве достаточно простого примера рассмотрим, как можно обратиться к старшей и младшей половинам целого числа типа Integer с использованием записи с вариантной частью.

```

type
  TInteger = record
    case Byte of

```

```

    0: (Number: LongInt);
    1: (LowPart, HighPart: Word);
end;

var
  A: TInteger;

begin
  A.Number := Random(High(Integer));

  Writeln('A = ', A, ' high part = ', A.HighPart, ' low part = ', A.LowPart);
  // К старшей части можно обращаться иначе: Word((Pointer(Integer(@A)+2))^)
end.

```

При записи констант типа запись поля записи указываются в круглых скобках через символ «;», каждому полю должно соответствовать значение, отделяемое от его имени символом «:».

```

Имя_константы: Тип_константы = (
  Поле1: Значение1;
  Поле2: Значение2;
  ...
  ПолеN: ЗначениеN
);

```

Некоторые поля в подобной записи могут быть опущены, при этом их значение окажется неопределенным. Приведем пример записи, представляющей собой экзаменационный вопрос теста. Константа со значениями полей, соответствующая конкретному вопросу, может быть задана в следующем виде:

```

type
  TQuestion = record
    Question: String;
    AnswersCnt: Integer;
    Answers: array [1..MAX_ANSWERS_CNT] of String;
    RightAnswer: Integer;
  end;

const
  Q1: TQuestion = (
    Question: 'Процессор исполняет программы в виде';
    AnswersCnt: 4;
    Answers: (
      'двоичного кода',
      'Delphi',
      'C++',
      'ассемблера'
    );
    RightAnswer: 1
  );

```

При хранении полей в памяти более быстрое обращение к ним достигается в случае, если поля имеют т.н. *выравнивание* на определенное



число байт (это является следствием тонкостей в организации работы шины между процессором и памятью и иерархии кэш-памяти). Компилятор Delphi старается располагать поля так, чтобы их стартовый адрес был кратен определенному значению, которое задается в опциях компилятора и по умолчанию составляет 8 байт. Аналогичного поведения можно добиться путем использования в коде программы директив компилятора `{ $A1 }`, `{ $A2 }`, `{ $A4 }`, `{ $A8 }`.

При использовании некоторых SIMD-команд ассемблера их операнды должны быть выровнены на 16 байт, в противном случае процессор выдает ошибку. Delphi не поддерживает данный тип выравнивания, что приводит к необходимости выполнения данного выравнивания путем адресной арифметики, что достаточно неудобно.

В качестве поясняющего примера рассмотрим следующую запись.

```
type
  TRecord = record
    a: Byte;
    b: LongInt;
  end;
```

Без учета выравнивания ее поля должны располагаться в памяти подряд, а размер переменной должен составлять 5 байт. На практике поле `b` записи выравнивается в памяти на 4 (иногда подобный подход называется выравниванием на натуральный размер, англ. *natural alignment*), между полями образуется неиспользуемое пространство из 3 байт, а размер переменной типа запись в памяти составляет 8 байт.

Если переменные подобного типа разместить в массиве, то он будет включать в себе множество «щелей» между полями, обращение к которым будет происходить относительно быстро, однако в целом память будет расходоваться неэффективно. Избежать данной ситуации можно путем изменения расположения данных. Например, вместо массива записей (англ. *array of structures*, AoS) можно использовать запись с полями-массивами (англ. *structure of arrays*, SoA), в некоторых случаях можно изменить порядок

следования полей записи. При необходимости принудительного отключения выравнивания полей записей независимо от текущих настроек компилятора перед ключевым словом `record` может быть использовано ключевое слово `packed`:

```
type
  TRecord = packed record
    a: Byte;
    b: LongInt;
  end;
```

## Файлы

При работе с диском программист абстрагирован от деталей физической реализации дискового хранилища, будь то обычный жесткий диск, flash-накопитель, RAID-массив или NAS. Дисковая память имеет определенную логическую структуру: набор разделов (или томов, англ. `partition` или `volume`) с соответствующими им буквами логических дисков («C:», «D:» и т.д.), деревом каталогов и расположенными в них файлами. Именно файл является минимальной логической единицей, с которой может работать программа при долговременном энергонезависимом хранении данных.

С точки зрения программы файл представляет собой именованную последовательность байт и очень похож на массив. При работе с файлами, расположенными на HDD, предпочтительными являются операции последовательного обращения к данным, т.к. при этом достигается максимальная скорость обмена (при случайном обращении скорость обмена может быть в десятки раз меньше). Для небольших файлов операционная система Windows применяет кэширование, располагая их в свободных областях оперативной памяти, что при повторном обращении к файлу существенно повышает скорость обмена, однако может привести к потере данных в случае внезапной потери питания.

Для работы с файлами в Delphi существует минимально необходимый набор подпрограмм, осуществляющий базовые действия. Основная стратегия работы с файлами состоит из открытия файла, осуществления необходимого набора действий (обычно чтение или запись данных) и закрытия файла. При отсутствии острой необходимости файлы не рекомендуется держать открытыми на протяжении работы программы (в том числе потому, что на это расходуются ресурсы операционной системы).

При работе с файлами их подразделяют на текстовые и бинарные. Текстовый файл хранит информацию в виде, доступном для редактирования с использованием любого текстового редактора. В бинарных файлах информация сохраняется в том виде, в котором с ней оперирует процессор. Для редактирования бинарных файлов можно использовать специальные HEX-редакторы, однако зачастую это неудобно. С точки зрения особенностей обработки в программе бинарные файлы подразделяются на типизированные и нетипизированные. Особенности работы с файлами каждого типа подробно рассмотрены ниже. Работа с файлами в Delphi производится с использованием файловой переменной, которой сопоставляется конкретный файл (в операционной системе ей соответствует определенный дескриптор, англ. *handle*). В файле существует понятие текущей позиции – файлового указателя, который при каждом чтении или записи автоматически смещается вперед на прочитанное/записанное число байт. При необходимости файловый указатель может быть перемещен к нужному месту вручную с использованием соответствующих подпрограмм (например, процедуры *Seek*).

Одним из поддерживаемых типов файлов являются текстовые файлы. Примерами таких файлов могут выступать файлы с расширением «.txt», «.pas», «.bat». Для объявления файловой переменной данного типа существует предопределенный тип *TextFile* (ранее в языке Паскаль он именовался как *Text*). Сопоставление файловой переменной имени

обрабатываемого файла производится путем обращения к подпрограмме `AssignFile` (ранее она называлась `Assign`):

```
var
  F: TextFile;
begin
  AssignFile(F, 'имя файла');
```

Если указано только имя файла (например, «1.txt»), считается, что он размещен в текущем каталоге (при необходимости данное поведение может быть изменено). При размещении файла по другому пути его имя может включать в себя путь в соответствии с правилами именования путей. Например, «C:\Temp\2.txt» – обращение к файлу в папке Temp на диске C, «\\srv1\data\3.txt» – обращение к файлу в общей папке (англ. shared folder) data на машине srv1 (т.н. UNC-путь, англ. Universal Naming Convention) или «..\..\4.txt» – обращение к файлу в папке на два уровня выше текущей.

При необходимости чтения содержимого файла он открывается с использованием подпрограммы `Reset`, принимающей в качестве единственного параметра файловую переменную. Если открываемый файл не существует, в программе произойдет исключительная ситуация и ее выполнение будет прервано. Чтобы этого избежать, можно использовать механизм обработки исключений или с использованием опции компилятора `{SO-}` запретить формирование исключения. При этом о неудаче (отсутствии файла) можно узнать по коду ошибки, который можно получить с использованием функции `IOResult`. Другой стратегией является предварительная проверка существования файла с использованием функции `FileExists`.

Если в файл планируется запись новой информации, для открытия служит подпрограмма `Rewrite`. При ее использовании файловый указатель устанавливается в начало файла, предыдущее содержимое файла удаляется. В случае отсутствия файла на диске он создается. Если существует

необходимость дописать данные в конец существующего файла, он может быть открыт с использованием подпрограммы `Append`, отличающейся от рассмотренных выше подпрограмм открытием существующего файла и установкой файлового указателя в конец.

Для чтения информации из текстового файла применяется хорошо знакомая по работе с консольными приложениями пара процедур `Read` и `Readln`, первым параметром которых должна быть указана файловая переменная. Для записи информации в текстовый файл применяются процедуры `Write` и `Writeln`, первым параметром которых также должна быть файловая переменная (при работе с окном консоли и клавиатурой по умолчанию используются предопределенные файловые переменные `Input` и `Output`, которые можно не указывать первым параметром).

После завершения действий с файлом его необходимо закрыть, для чего применяется подпрограмма `CloseFile` (ранее называвшаяся `Close`), принимающая в качестве единственного параметра файловую переменную.

Только при закрытии файла гарантируется, что все временные буферы в оперативной памяти будут действительно сброшены на диск и в файле окажется вся информация, поэтому необходимо всегда закрывать открытые файлы. Кроме того, к открытому с использованием стандартных средств файлу не смогут получить доступ другие приложения операционной системы, что в ряде случаев может вызвать определенные проблемы.

В качестве достаточно простого примера работы с текстовыми файлами рассмотрим программу, которая формирует текстовый файл, записывая в него значение ограниченной строки и несколько целых чисел.

```
var
  F: TextFile;
  S: String[25] = 'Пример текстового файла';
  I: Integer;

begin
  // Сопоставление файловой переменной имени файла
  AssignFile(F, 'File.txt');

  // Открытие файла для записи
  ReWrite(F);
```

```

// Запись значений
Writeln(F, S);
for I := -5 to 5 do
  Writeln(F, I);

// Закрытие файла
CloseFile(F);
end.

```

Другим способом является сохранение значений переменных в том же самом двоичном виде, в котором они используются для обработки процессором, без преобразования в текстовое представление. Примерами подобных двоичных файлов является большинство файлов («.exe», «.doc», «.dll»). Рассмотрим сперва особенности работы с нетипизированными бинарными файлами.

Работа с ними производится с использованием файловой переменной типа `File`, которой перед началом работы аналогично рассмотренному выше необходимо сопоставить имя файла с использованием процедуры `AssignFile`. Далее файл следует открыть для чтения или записи, однако в данном случае при использовании процедур `Reset` или `Rewrite` после файловой переменной необходимо указывать второй параметр целого типа, соответствующий минимальной порции данных, с использованием которой производится обмен данными с файлом. Ранее указанный параметр влиял на размер буфера в оперативной памяти, который использовался для операций с файлом, сегодня действия с файлами производятся с использованием WinAPI-функций операционной системы и указанный параметр не оказывает существенного влияния на скорость выполняемых операций и обычно принимается равным 1.

Чтение и запись нетипизированных бинарных файлов производится с использованием процедур `BlockRead` и `BlockWrite`:

```

procedure BlockRead(var F: File; var Buf; Count: Integer; [var Result: Integer]);
procedure BlockWrite(var F: File; const Buf; Count: Integer; [var Result: Integer]);

```

Кроме файловой переменной `F` они в качестве обязательных параметров принимают буфер `Buf`, который используется соответственно как источник

или приемник данных, а также размер порции данных `Count` в байтах, которую требуется прочитать или записать. При этом параметр `Buf` является нетипизированным и никаких проверок соответствия размера буфера и числа читаемых или записываемых байт не производится, ответственность целиком и полностью лежит на программисте. Последний параметр `Result` не является обязательными и используется достаточно редко: в нем производится возврат числа реально записанных или прочитанных байт. Если данное значение отличается от значения параметра `Count`, это говорит о наличии ошибок в работе дисковой подсистемы (в простейшем случае это может быть следствием нехватки дискового пространства, более неприятной является ситуация наличия аппаратных ошибок, что в итоге может привести к частичной или полной потере данных).

После завершения обработки нетипизированного бинарного файла он должен быть закрыт с использованием процедуры `CloseFile`.

Ниже приведен пример, показывающий сохранение той же информации (ограниченная строка и ряд целых чисел) в бинарной форме.

```
var
  F: File;
  S: String[25] = 'Пример текстового файла';
  I: Integer;

begin
  AssignFile(F, 'File.txt');
  Rewrite(F, 1);

  BlockWrite(F, S, SizeOf(S));
  for I := -5 to 5 do
    BlockWrite(F, I, SizeOf(I));

  CloseFile(F);
end.
```

Среди содержимого сформированного файла можно увидеть обрывки текстовой информации, однако остальная информация хранится в двоичном виде. Так, например, первый байт файла, имеющий двоичное значение  $17_{16} = 23_{10}$ , представляет собой текущую длину ограниченной строки, после которой располагаются двоичные значения, соответствующие целым числам в дополнительном коде. Обычно хранение информации в бинарном виде

требует меньше места и зачастую используется там, где не требуется ручное редактирование данных с использованием сторонних текстовых редакторов.

Если бинарный файл предназначен для хранения однотипной информации (например, записей таблицы базы данных, имеющих один и тот же двоичный формат), то для его обработки могут быть использованы средства работы с типизированными бинарными файлами. При этом файловая переменная определяется как

```
var
  F: File of Тип_записи;
```

При записи и чтении файла используются процедуры `Read` и `Write`, однако в качестве параметров кроме указания файловой переменной допускается использовать только переменные типа `Тип_записи`, указанного при объявлении файловой переменной.

В качестве примера рассмотрим простую программу, осуществляющую чтение из типизированного бинарного файла.

```
type
  TComplex = record
    Re, Im: Double;
  end;

const
  N = 5;

var
  F: File of TComplex;
  Arr: array [1..N] of TComplex;
  I: Integer;

begin
  AssignFile(F, 'File.txt');
  Reset(F);

  for I := 1 to N do
    Read(F, Arr[I]);

  CloseFile(F);
end.
```

Обычно на практике бинарные файлы имеют более сложный формат, зачастую включающий в себя заголовок, версию формата файла, контрольную сумму содержимого и т.п., поэтому работа с ними как правило ведется как с нетипизированными файлами. При необходимости текстовые



файлы также могут быть открыты как бинарные, однако зачастую работа с ними в подобном режиме представляется неудобной.

При чтении содержимого файлов можно узнать, достигнут ли конец файла (попытка дальнейшего чтения будет вызывать ошибки), для чего можно воспользоваться функцией `Eof`, возвращающей значение типа `Boolean`. При работе с текстовыми файлами существует также функция `Eoln`, которая возвращает истину в случае, если файловый указатель находится в конце строки. Текущую позицию файлового указателя можно узнать с использованием функции `FilePos`, а размер файла – с использованием функции `FileSize`.

## Пример программы

Задача. Сравнить двоичное содержимое заданной пары бинарных файлов и, в случае его несовпадения, выдать номера и значения несовпадающих байт.

```

program bv;
{$APPTYPE CONSOLE}

uses
  SysUtils;

{ Имена сопоставляемых файлов }
const
  FileName1 = 'C:\test1.tst';
  FileName2 = 'C:\test2.tst';

type
  TBuf = array of Byte;

{ Буферы для размещения данных }
var
  Buf1, Buf2: TBuf;

{ Подпрограмма для загрузки данных в буфер }
procedure LoadFile(FileName: String; var Buf: TBuf);
var
  F: File;
begin
  AssignFile(F, FileName);
  Reset(F, 1);

  SetLength(Buf, FileSize(F));

  BlockRead(F, Buf[0], Length(Buf));

  CloseFile(F);
end;

```

```

{ Подпрограмма для сравнения буферов }
function CompareBufs(const Buf1, Buf2: TBuf): Boolean;
var
  I: Integer;
begin
  Result := False;

  if Length(Buf1) <> Length(Buf2) then begin
    Writeln('Lengths differ: ', Length(Buf1), ' bytes and ', Length(Buf2), ' bytes');
    exit;
  end;

  for I := 0 to High(Buf1) do
    if Buf1[I] <> Buf2[I] then begin
      Writeln('Differ at byte ', I, ': ', IntToHex(Buf1[I], 2), 'h and ', IntToHex(Buf2[I], 2),
'h');
      exit;
    end;

  Result := True;
end;

begin
  { Чтение файлов }
  LoadFile(FileName1, Buf1);
  LoadFile(FileName2, Buf2);

  { Сравнение прочитанных данных }
  if CompareBufs(Buf1, Buf2) then
    Writeln('OK')
  else
    Writeln('Not equal');

  Readln;
end.

```

## Индивидуальные задания

1. Дан набор csv-файлов, хранящий значения счетчика количества напечатанных на принтере страниц с момента загрузки компьютера. Построить график изменения значения счетчика в заданном временном диапазоне. Учесть, что в каждом новом файле значение счетчика начинается с нуля, хотя реальное количество страниц, напечатанное, например, за месяц, в момент создания файла ненулевое.
2. Дан набор csv-файлов, хранящий значения счетчика количества напечатанных на принтере страниц с момента загрузки компьютера. Определить количество напечатанных страниц в заданном временном диапазоне и среднее количество печатаемых страниц в день за все время измерений. Учесть, что в каждом новом файле значение счетчика начинается с нуля, хотя реальное количество страниц, напечатанное, например, за месяц, в момент создания файла ненулевое.

3. Дан набор csv-файлов, хранящий значения счетчика количества напечатанных на принтере страниц с момента загрузки компьютера. «Склеить» данные файлов в один файл формата csv (первая строка – заголовок, остальные строки – записи с данными).
4. Файлы в указанной пользователем папке имеют имена «0», «1», «2», ..., «148», «149» и т.д. Преобразовать имена файлов следующим образом: «0000», «0001», «0002», ..., «0148», «0149» (количество цифр в имени файла задается пользователем).
5. Пользователем вводится папка и маска (например, «C:\Program Files\Borland» и «\*.pas»). Подсчитать суммарное количество строк в текстовых файлах, располагающихся в заданной папке и соответствующих заданной маске. При наличии в указанной папке вложенных папок необходимо также осуществить их сканирование.
6. Если по указанному пути находится файл backup.rar, переименовать его в backupDDMMYY.rar, где DD, MM и YY – день, месяц и год создания файла. Если по указанному пути количество файлов превышает заданное, удалить несколько наиболее старых файлов. Количество файлов задается в текстовом файле конфигурации Config.ini.
7. Задан текстовый файл с данными о днях рождения группы людей. Сформировать еще один текстовый файл, в котором указанные лица будут упорядочены в порядке убывания количества дней, оставшегося до их дней рождения.
8. Задан текстовый файл. Добавить в начало каждой из его строк  $N$  пробелов (сделать отступ).  $N$  задается пользователем.
9. Осуществить рекурсивное сканирование всех подпапок в заданной корневой папке. В каждой из просмотренных папок удалить файл с расширением .exe, имя которого совпадает с именем папки (если такой файл существует).

10. Определить частоты встречаемости символов в заданном текстовом файле. Вывести полученные значения частот в текстовый файл result.txt в виде гистограммы.

11. Заданы две папки. Провести синхронизацию содержимого папок (включая вложенные подпапки) по следующим правилам: если в одной из папок отсутствует папка или файл, присутствующие в другой, скопировать их в папку; если время модификации файлов в папках отличается, заменить более старый файл новым.

12. В указанном файле на языке Delphi найти все идентификаторы переменных и вывести их в отдельный файл.

13. В указанном файле на языке Delphi удалить все комментарии и сохранить полученный текст программы в виде отдельного файла.

14. В указанном файле на языке Delphi найти все изображения строковых констант и вывести их в отдельный файл.

15. В указанном файле на языке Delphi найти все изображения целочисленных констант и вывести их в отдельный файл.

16. В указанном файле на языке Delphi найти все изображения вещественных констант и вывести их в отдельный файл.

17. В указанном файле на языке Delphi найти все имена рекуррентных подпрограмм и вывести их в отдельный файл.

18. Вычислить энтропию указанного пользователем файла по формуле

$$H = -\sum_{i=1}^n p_i \log_2 p_i, \text{ где } p_i - \text{вероятность встречаемости } i\text{-го символа.}$$

Осуществить сжатие указанного файла одним из архиваторов и вычислить энтропию для полученного архива. Сравнить полученные значения.

19. В заданном текстовом файле найти и вывести все IP-адреса (см. файл log.txt). IP-адрес записывается как 4 целых числа из диапазона 0...255 каждое, разделенных точками (например, «169.254.54.12»).

20. Задан текстовый файл, каждая строка которого представляет собой отдельный экзаменационный вопрос. Сформировать из указанных вопросов

$N$  экзаменационных билетов по  $K$  вопросам в каждом при условии, что одни и те же вопросы должны повторяться минимальное число раз в различных билетах.

## Содержание отчета

1. Титульный лист.
2. Индивидуальное задание.
3. Краткое описание стратегии решения.
4. Листинг программы.
5. Тестовые примеры, результаты тестирования.
6. Выводы.

## Контрольные вопросы

1. Для чего применяются записи?
2. Как производится обращение к полям записей?
3. В заключается функционал оператора присоединения?
4. Как определяется размер записи в памяти компьютера? Как на него влияет выравнивание полей?
5. Как логически организовано хранение информации на жестком диске компьютера?
6. Какие типы файлов поддерживаются при работе с ними в Delphi?
7. Какие подпрограммы используются при работе с файлами?

## Библиографический список

1. Емельянов С.Г., Ватутин Э.И., Панищев В.С., Титов В.С. Процедурно-модульное программирование на Delphi: учебное пособие. М.: Аргмак-Медиа, 2014. 352 с.

2. Зотов И.В., Ватутин Э.И., Борзов Д.Б. Процедурно-ориентированное программирование на C++: учебное пособие. Курск: КурскГТУ, 2008. 211 с.