Беляков А.Ю. | tt@59.ru
Версия от 18.05.2017

Интенсив по основам программирования на языке Pascal

Среда разработки PascalABC.NET

На кого рассчитан интенсив:

- раньше программировал, есть базовая подготовка, но давно не было практики самостоятельной работы и многое забыл;
- начинающий разработчик, посещал лекции и внимательно слушал, но не всё понял и было мало практики самостоятельной работы;
- нерегулярно посещал занятия как лекционные, так и практические, есть начальные навыки разработки, но не хватает умений в проектировании и знаний о технологиях программирования.

Что тут можно найти:

- обработка текстовых и типизированных файлов,
- обработка строк,
- динамические массивы и множества,
- циклы и ветвления,
- подпрограммы и модульное программирование,
- построение диаграмм и графиков,
- анализ данных, поиск и сортировка,
- обработка исключений.

Введение. Обязательно к прочтению.

Программа исполняет в точности, что ей укажешь и так, как ей позволяют настройки среды исполнения, компилятор и подключенные модули. Программы, написанные на языке Pascal, могут незначительно отличаться для таких сред исполнения как FreePascal, Delphi, PascalABC.NET. Эти незначительные изменения могут привести к тому, что код программы не будет компилироваться из другой среды.

То, что будет рассмотрено в этом интенсиве, проверено и работает в PascalABC.NET, но может работать при незначительных изменениях и в других средах. Сам PascalABC и платформа .NET постоянно совершенствуются, поэтому, со временем приведенные шаблоны программ могут устареть морально. Для начала работы скачайте с официального сайта и установите PascalABC.NET. Создайте папку и сохраняйте в неё все разрабатываемые программы. Я буду называть их шаблонами, так как именно так я их и рассматриваю, как шаблоны для последующего использования. В тексте каждого шаблона есть комментарии, которые позволяют понять суть кода и при некотором опыте настроить его под свои нужды.

Текст каждого шаблона структурирован – код написан с отступами в соответствии с вложенностью операторов друг в друга. Обязательно используйте этот стиль оформления, начиная с первых шагов в программировании, так как вам самим будет проще понимать и редактировать ваш код, а также вашим коллегам и начальнику.

Стиль не только дисциплинирует внешние проявления в кодировании, но и формирует алгоритмическое мышление. Не пренебрегайте стилем (стиль CamelCase, Венгерская нотация и др.), даже в наименовании переменных, констант, классов и подпрограмм. Как правило, в рамках организации (где вы будете работать) вам нужно будет привыкать к установленному там стилю, но, в большинстве случаев, принятые подходы схожи и это уже не будет для вас пыткой.

Тема 1. Обработка текстовых файлов

Программа может обойтись без диалога с пользователем, но иногда они необходимы, хотя бы для того, чтобы пользователь могу увидеть результаты работы программы до её закрытия. Поэтому первые шаблоны посвящены организации диалогов с пользователем. И начнем мы с простого **Шаблона #0**: программа запускается в отдельном окне, что-то выводит на экран, не закрывается до нажатия клавиши. Создай новый файл, сразу сохрани его в папку, скопируй туда этот код (можно без комментариев):

```
uses Crt; // подключаем модуль для отображения программы в окне
begin
  writeln('Это первая программа'); // выводим на экран
  repeat
    // пустой цикл: повторяй до нажатия на любую клавишу
  until KeyPressed;
end.
```

Для запуска из среды нажми **Shift+F9**, но чтобы откомпилировать в отдельный исполняемый файл (с расширением exe) - **Ctrl+F9** (файл появится в той же папке).

Следующий **Шаблон #1**: запускается в отдельном окне, выводит дату и время, ожидает нажатия клавиши q (от quit - выход), проверяются все комбинации символа при вводе с этой клавиши.

```
uses Crt, System; // + модуль для работы с системными функциями
var c: char; // переменная для хранения символа
    d: DateTime; // переменная для хранения даты и времени
begin
  d:=DateTime.Now; // получаем текущую дату и время
  writeln(d.Day,'.',d.Month,'.',d.Year); // на экран дату
  writeln(d.Hour,'.',d.Minute,'.',d.Second); // на экран время
  repeat // цикло ДО наступления истинности условия
    c:=readkey; // нажатую клавишу запоминаем
  until c in ['q','Q','й','Й']; // входит ли во множество ?
end.
```

Не все клавиши имеют отображаемый символ, поэтому можно анализировать не сам символ, а его код в таблице символов.

Шаблон #2 – программа отображает дату и время в установленном формате и не закрывается до нажатия клавиши Escape (код клавиши равен 27):

```
uses Crt, System; // подключаем модули
var s: string; // переменная для хранения строки
begin
  s:=String.Format('{0:dd/mm/yy HH:mm:ss}', DateTime.Now);
  // получаем значение DateTime и форматируем его по шаблону
  writeln(s); // выводим на экран
  repeat until ord(readkey)=27; // цикл до нажатия клавиши с кодом 27
end.
```

Если можно выделить в программе такую часть кода, которая имеет самостоятельное функциональное назначение и может быть использована неоднократно в текущей или другой программе, то её называют подпрограммой, дают ей имя и выносят из основного текста.

В следующем шаблоне используется входной файл input.txt (файл создайте в текстовом редакторе и расположите в той же папке, что и программа):

```
10
20
40
30
50
```

Шаблон #3 – программа использует подпрограмму pause() для организации удержания окна после выполнения программы, программа открывает текстовый файл для чтения и первую строку выводит на экран:

```
uses Crt; // подключаем модуль для отображения программы в окне

procedure pause(); // подпрограмма задерживает экран до нажатия Escape
begin
```

```
  writeln('Программа завершена...'); // пишет на экран
  repeat until ord(readkey)=27; // ждёт нажатия клавиши с кодом 27
end;

var input: Text; // объявляем переменную для текстового файла
    s: string; // переменная для хранения строки

begin
  AssignFile(input, 'input.txt'); // ассоциируем файл с переменной
  Reset(input); // открываем файл для чтения
  Readln(input, s); // читаем из файла одну строку
  Writeln(s); // пишем строку на экран
  CloseFile(input); // закрываем файл

  pause(); // запустить подпрограмму pause
end.
```

При работе с файлами возможна ситуация ошибки чтения (нет файла, имя другое, строки в файле уже закончились и т.п.). Если не предусмотреть в программе код для обработки ошибок времени исполнения, то программа будет «вылетать». Как справиться с подобной ситуацией, показано в **Шаблоне #4** - программа содержит два пути исполнения (блок try обеспечивает попытку выполнения программы, а при неудаче производится переход к блоку except), выход из программы осуществляется по нажатию на клавишу Enter:

```
uses Crt;
var input: Text;
    s: string;
begin
  try // начало блока, где возможна ошибка
    AssignFile(input, 'input.txt');
    Reset(input); // открываем файл для чтения
    Readln(input, s); // читаем из файла одну строку
    Writeln(s); // пишем строку на экран
    CloseFile(input); // закрываем файл
  except // если была ошибка, то переходим сюда
    writeln('Ошибка чтения файла...');
  end;
  ReadLn(); // закрыть программу при нажатии Enter
end.
```

Следующий **Шаблон #5** показывает способ обработки всех строк текстового файла - программа открывает файл для чтения, с помощью цикла с предусловием читает последовательно содержимое строк в

целочисленную переменную, накапливает сумму чисел и их количество в соответствующих переменных, выводит на экран среднее арифметическое чисел с точностью до двух знаков после запятой:

```
uses Crt;
var input: Text;
    elm, sum, count: integer;
begin
  AssignFile(input, 'input.txt');
  Reset(input);
  sum := 0; // устанавливаем сумму в ноль
  count := 0; // устанавливаем счетчик в ноль
  while not eof(input) do // пока не конец файла
  begin // объединяем группу операторов в тело цикла
    Readln(input, elm); // читаем из файла одно число
    sum := sum + elm; // добавляем его к сумме
    inc(count); // увеличиваем счетчик на +1
  end;  // объединяем группу операторов в тело цикла
  CloseFile(input);

  Writeln(sum/count:0:2); // ответ с точностью до 2-х знаков
  ReadLn();
end.
```

А что если нужно выбрать из первого файла только те числа, которые больше среднего арифметического и сохранить их в другой файл. Получается, что нужно сначала пройти все строки первого файла, чтобы определить среднее и потом ещё раз пройти, чтобы выбрать нужные числа. Работа с файловой системой существенно медленнее, чем с ячейками оперативной памяти, поэтому имеет смысл отказаться от второго прохода по файлу, а перенести значения на первом проходе в оперативную память в массив или список и работать уже с элементами этих структур.

Шаблон #5 - программа читает входной файл построчно и размещает значения в динамический массив, размер массива увеличивается по мере чтения из файла, после чтения подсчитывается среднее значение, далее открывается новый файл для записи, запускаем цикл по всем элементам массива и в выходной файл записываем только те из них, которые больше среднего (формат вывода – номер элемента, символ табуляции, значение элемента):

```
uses Crt;
var input, output: Text;
    avg: real; // переменная для хранения вещественного числа
    elm, sum, i: integer;
    arr: array of integer; // динамический массив целых чисел
begin
  AssignFile(input, 'input.txt');
  Reset(input);
  sum := 0; // устанавливаем сумму в ноль
  SetLength(arr, 0); // устанавливаем размер массива
  while not eof(input) do
  begin
    Readln(input, elm); // читаем из файла одно число
    sum := sum + elm; // добавляем его к сумме
    SetLength(arr, arr.Length+1); // увеличиваем массив
    arr[arr.Length-1]:=elm; // записываем очередной элемент
  end;
  CloseFile(input);

  avg:=sum/arr.Length; // вычисляем среднее

  AssignFile(output, 'output.txt');
  Rewrite(output); // открыть файл для записи
  for i:=0 to arr.Length-1 do // индекс по всем элементам массива
    if arr[i]>avg then
      writeln(output,i,chr(9),arr[i]);
  CloseFile(output);

  ReadLn();
end.
```

После работы данной программы мы получим выходной файл output.txt с таким содержимым:

```
2    40
4    50
```

Не удаляйте данный файл, так как впоследствии мы будем использовать его для дополнения в него записей.

Первая часть программы читает построчно текстовый файл и размещает прочитанное в массив. Такая задача является стандартной и довольно часто встречается, поэтому имеет смысл выделить эту часть кода в подпрограмму. Ранее мы уже использовали подпрограмму pause(), вида процедура - она выполняла некоторые действия, но не возвращала значения. Более того мы не передавали в процедуру никаких значений. Зачем нужно что-то передавать в подпрограмму?

Чтобы можно было разнообразить её действия, например, в подпрограмму pause() можно передавать в качестве аргумента код клавиши, по нажатию на которую нужно закрывать программу. В следующем шаблоне будет соответствующий пример.

Теперь будем использовать подпрограмму вида функция: она будет принимать на вход строковое значение – имя файла, а возвращать в тело основной программы значение типа массив строк (это строки, считанные из файла). Будем использовать не массив целых чисел, а массив строк для универсальности функции, так как рассчитываем на её дальнейшее использование в других программах. Может быть, потребуется читать текстовый файл, в котором не обязательно должны быть именно целые числа, а, наоборот, это могут оказаться значения любого типа данных или даже смешанные строки. Получив массив строк, уже потом в самой программе, мы можем правильно строки разобрать на данные нужного типа.

Итак, **Шаблон #6** – это программа, которая запускает функцию чтения файла, результаты работы функции передаёт в массив строк, потом параметрическим (for) циклом подсчитывается среднее значение с использованием преобразования строкового элемента массива к целому типу данных (StrToInt), далее открывается для дозаписи уже существующий файл output.txt (не стирается то, что было ранее, а добавляется в конец файла), запускаем цикл по всем элементам массива и в выходной файл дописываем только те из них, которые меньше или равны среднему, перед завершением программы срабатывает пауза, которую можно снять нажатием на пробел:

```
uses Crt;

procedure pause(code: byte); // подпрограмма задерживает экран
begin
  repeat until ord(readkey)=code; // ждёт нажатия клавиши с кодом code
end;

function read_array(filename: string): array of String;
var input: Text;
    elm: String;
    arr: array of String;
begin
```

```
  AssignFile(input, filename);
  Reset(input);
  SetLength(arr, 0); // устанавливаем размер массива
  while not eof(input) do
  begin
    Readln(input, elm); // читаем из файла одно число
    SetLength(arr, arr.Length+1); // увеличиваем массив
    arr[arr.Length-1]:=elm; // записываем очередной элемент
  end;
  CloseFile(input);
  result:=arr; // возвращаем значение в тело основной программы
end;

var output: Text;
    avg: real; // переменная для хранения вещественного числа
    sum, i: integer;
    arr: array of String; // динамический массив строк
begin // это начало программы
  arr:=read_array('input.txt'); // вызываем подпрограмму чтения файла

  sum := 0; // устанавливаем сумму в ноль
  for i:=0 to arr.Length-1 do // индекс по всем элементам массива
    sum:=sum+StrToInt(arr[i]); // из строки в число и + его к сумме
  avg:=sum/arr.Length; // вычисляем среднее

  AssignFile(output, 'output.txt');
  Append(output); // открыть файл для дозаписи
  for i:=0 to arr.Length-1 do // индекс по всем элементам массива
    if StrToInt(arr[i])<=avg then
      writeln(output,i,chr(9),arr[i]);
  CloseFile(output);

  pause(32); // пауза до нажатия клавиши пробел (код = 32)
end.
```

После работы данной программы мы получим дополненный выходной файл output.txt с таким содержимым:

```
2    40
4    50
0    10
1    20
3    30
```

Тема 2. Модульное программирование

Постепенно разрабатываемые и исследуемые тут программы становятся всё больше и сложнее, что затрудняет процесс понимания и редактирования. Мы уже сделали попытку к снижению сложности кода, разбив его на отдельные участки, называемыми подпрограммами. Но можно сделать и следующий шаг: перенести подпрограммы в отдельный модуль. Модуль – это отдельный файл с подпрограммами, который нельзя запустить самостоятельно как программу. Модуль можно подключить из программы и использовать находящиеся в нём процедуры и функции. Модульное программирование не только снижает сложность кода программы, но и даёт возможность последующего использования подпрограмм в других разработках.

В нашем шаблоне #6 были две подпрограммы: процедура для задержки экрана и функция для чтения строк из файла в массив. Обе эти подпрограммы имеют универсальное назначение и могут быть применены в других программах, поэтому имеет смысл вынести их в отдельный модуль для последующего использования.

Чтобы разработать модуль нужно просто создать новый файл для кода, перенести туда необходимые подпрограммы, добавить заголовочную часть с описанием подпрограмм и сохранить модуль в той же папке, что и основная программа. Обобщенная структура модуля такова:

```
unit Utils; // unit нужно писать, чтобы компилировалось как модуль
            // далее идёт наименование модуля на ваш выбор

interface // начало интерфейсной части

  // тут декларируем названия и сигнатуры подпрограмм

implementation // начало части реализации подпрограмм

  // тут размещаем тексты подпрограмм

end.
```

Итак, в качестве **шаблона #7** я приведу два файла: модуль Utils и текст программы. Обратите внимание, что в модуле есть две подпрограммы с именем pause. Так можно делать, если мы хотим иметь немного отличающиеся по функционалу процедуры или функции, но с общим назначением и названием. Например, общее назначение - задерживать экран с результатами работы программы, варианты реализации: а) до нажатия любой клавиши; б) до нажатия клавиши по выбору программиста. Чтобы это было возможно (использование одного имени для нескольких подпрограмм), нужно после декларирования подпрограммы обозначить, что она перегружаемая (**overload**), а также соблюдать уникальность сигнатур (у подпрограмм должно хоть что-то различаться: количество аргументов, типы данных).

Текст модуля:

```
unit Utils; // синтаксис для модуля

interface // интерфейсная часть
  procedure pause(); overload;
  procedure pause(code: byte); overload;
  function read_array(filename: string): array of String;

implementation // часть реализаций подпрограмм

uses Crt;

procedure pause();
begin
  repeat until KeyPressed;
end;

procedure pause(code: byte);
begin
  repeat until ord(readkey)=code;
end;

function read_array(filename: string): array of String;
var input: Text;
    elm: String;
    arr: array of String;
begin
  AssignFile(input, filename);
  Reset(input);
  SetLength(arr, 0);
  while not eof(input) do
  begin
    Readln(input, elm);
```

```
    SetLength(arr, arr.Length+1);
    arr[arr.Length-1]:=elm;
  end;
  CloseFile(input);
  result:=arr;
end;

end.
```

Текст программы:

```
uses Crt, Utils; // подключаем наш модуль

var output: Text;
    i, temp: integer;
    arr: array of String;

begin
  arr:=read_array('input.txt');

  AssignFile(output, 'output.txt');
  Rewrite(output); // открыть файл для записи
  for i:=arr.Length-1 downto 0 do // в обратном порядке
  begin
    temp:=StrToInt(arr[i]); // строку перевести в целое число
    temp:=(temp div 10) mod 10; // узнать цифру десятков
    if not odd(temp) then // если десятков четное количество
      writeln(output,i,chr(9),arr[i],' - четное');
  end;
  CloseFile(output);

  pause(27); // пауза до нажатия Escape
end.
```

Программа использует вспомогательные подпрограммы из модуля Utils. В начале программы идет чтение строк текстового файла input.txt в массив строк arr, затем открывается другой файл output.txt для записи и туда переносятся только те строки, в которых находятся числа с четным количеством десятков. Результаты выводятся в обратном порядке. Пример выходного файла:

```
2    40 - четное
1    20 - четное
```

Вы, конечно, обратили внимание, что в нашем входном файле значения не отсортированы. Очень удобно использовать модульное программирование для реализации шаблонных задач, таких как про-

читать значения из файла, отсортировать их, обратно записать в файл. Оцените **Шаблон #8**:

```
uses Crt, Utils; // подключаем наш модуль
const filename = 'input.txt';
var arr: array of String;
begin
  arr:=read_array(filename); // читаем в массив из файла
  sort_array(arr); // сортируем значения
  write_array(filename, arr); // пишем массив в файл

  pause(27); // пауза до нажатия Escape
end.
```

Программа короткая и элегантная, но, очевидно, что для её функционирования нужны доработки в модуле:

```
unit Utils; // синтаксис для модуля

interface // интерфейсная часть
  procedure pause(); overload;
  procedure pause(code: byte); overload;
  function read_array(filename: string): array of String;
  procedure write_array(filename: String; arr: array of String);
  procedure sort_array(var arr: array of String);

implementation // часть реализаций подпрограмм

uses Crt;

procedure pause();
begin
  repeat until KeyPressed;
end;

procedure pause(code: byte);
begin
  repeat until ord(readkey)=code;
end;

function read_array(filename: string): array of String;
var input: Text;
    elm: String;
    arr: array of String;
begin
  AssignFile(input, filename);
  Reset(input);
  SetLength(arr, 0);
  while not eof(input) do
  begin
```

```
    Readln(input, elm);
    SetLength(arr, arr.Length+1);
    arr[arr.Length-1]:=elm;
  end;
  CloseFile(input);
  result:=arr;
end;

procedure write_array(filename: String; arr: array of String);
var i: integer;
    output: Text;
begin
  AssignFile(output, filename);
  Rewrite(output);
  for i:=0 to arr.Length-1 do
    writeln(output,arr[i]);
  CloseFile(output);
end;

procedure swap(var a: String; var b: String);
var c: String;
begin
  c:=a; a:=b; b:=c;
end;

procedure sort_array(var arr: array of String);
var i, j: integer;
begin // пузырьковая сортировка
  for j:=0 to arr.Length-1-1 do
    for i:=1 to arr.Length-1-j do
      if arr[i-1]>arr[i]
        then swap(arr[i-1],arr[i]);
end;

end.
```

Обратите внимание, что в модуль были добавлены две подпрограммы: процедура сортировки методом «пузырька» `sort_array` и процедура `swap`, которая меняет местами значения элементов. И только одна из них попала в интерфейсную часть. Дело в том, что в интерфейсную часть как раз и принято размещать описание только тех подпрограмм, которые мы хотим сделать видимыми извне модуля и, соответственно, доступными для использования, а те, которые мы сделали как вспомогательные (нужные для работы самого модуля), можно скрыть.

Есть ещё один нюанс - в некоторых местах аргументам подпрограмм предшествует служебное слово var (от variable - переменная),

но во многих местах оно отсутствует. Причина в том, что аргумент можно передавать в подпрограмму как значение, а можно – по ссылке. При передаче значения в теле подпрограммы создается дополнительная переменная соответствующего типа и в самой подпрограмме работа производится уже с новой переменной. Все изменения, которые претерпевает новая переменная в рамках работы подпрограммы, не будут видны из основной программы. Если же мы хотим, чтобы в подпрограмме производилась работа именно с той же переменной, что мы потом будем использовать в основной программе, то нужно передавать не само значение, а ссылку на него. Именно такой порядок установлен в большинстве языков программирования и для его осуществления используются специальные ключевые слова, в частности в Паскале это var. Однако PascalABC во многих случаях идет по пути упрощения работы программиста и позволяет ему в данном случае не писать (можно писать, можно не писать) явно слово var – компилятор сам распознает необходимость в нём и передает аргумент по ссылке. Уточню, что в Delphi и во FreePascal (а также во многих других языках программирования) нет таких послаблений, поэтому я рекомендую не привыкать к отклонениям от стандарта. К тому же вы можете быть неправильно поняты другими разработчиками, да и начальнику такой стиль программирования может не понравиться.

Тема 3. Графики функций и анализ данных

Шаблон #9 посвящен часто встречающейся теме: произвести парсинг строк с данными, сгруппировать данные в массив, представить их в графическом виде – **построить диаграмму** и сохранить в графический файл.

Данные находятся во входном текстовом файле kinopoisk.txt в таком формате – на каждый фильм приходится по четыре строки:

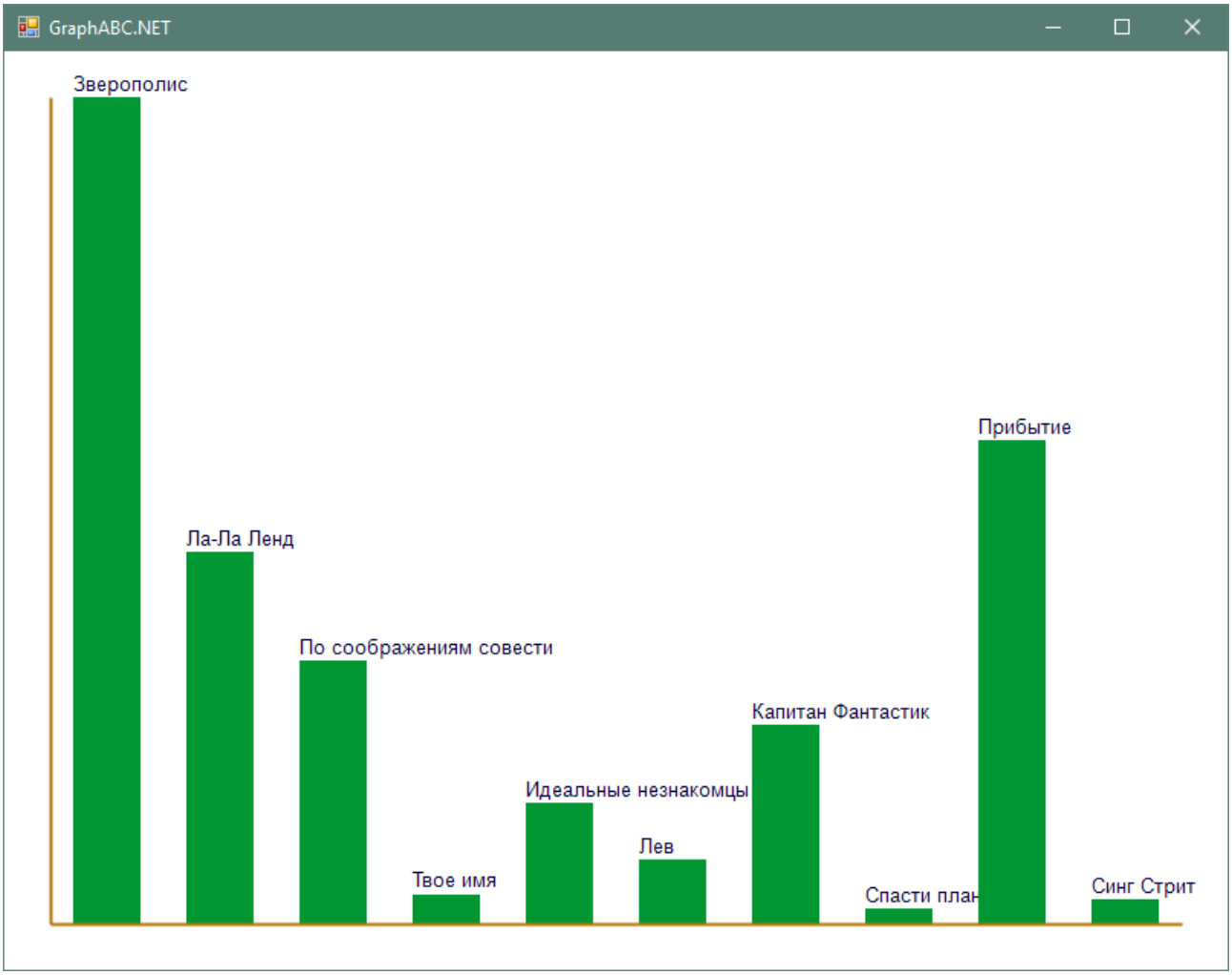
```
1.
Зверополис (2016)
Zootopia
8.382 (170 618)
2.
Ла-Ла Ленд (2016)
La La Land
8.344 (76 829)
3.
По соображениям совести (2016)
Hacksaw Ridge
8.086 (54 199)
4.
Твое имя (2016)
Kimi no na wa.
8.086 (5 887)
5.
Идеальные незнакомцы (2016)
Perfetti sconosciuti
7.702 (24 989)
6.
Лев (2016)
Lion
7.700 (13 187)
7.
Капитан Фантастик (2016)
Captain Fantastic
7.694 (41 049)
8.
Спасти планету (2016)
Before the Flood
7.681 (3 176)
9.
Прибытие (2016)
Arrival
7.643 (99 824)
10.
Синг Стрит (2016)
Sing Street
7.636 (4 985)
```

Нужно выбрать названия фильмов без года (это вторая строка в четверке строк) и количество голосовавших за фильм без их рейтинга (это число целое из скобок в последней строке в каждой четверке строк). То есть из входного файла нужно выбрать такую информацию:

```
Зверополис    170618
Ла-Ла Ленд    76829
По соображениям совести  54199
Твое имя      5887
Идеальные незнакомцы     24989
Лев     13187
Капитан Фантастик  41049
Спасти планету     3176
Прибытие      99824
Синг Стрит    4985
```

И на основе этих данных построить диаграмму, так чтобы столбики вместились в размеры экрана и сопровождались подписями названий фильмов. Учтите, что во входном файле количество проголосовавших не просто находится внутри круглых скобок, но ещё и имеет пробел – как делитель числа для наглядности отображения, который нужно будет удалить.

Вот такой должен быть итог программы на экране и аналогичный в виде графического файла в папке программы:



Текст шаблона #9:

```
uses GraphABC;

Type
  TLang = record
    name: string[24]; // название фильма
    number: integer;  // количество голосовавших
end;

function parse(s: string): integer;
var count, gl, gr: integer; str: string;
begin
  gl:=pos('(', s); // левая граница числа
  gr:=pos(')', s); // правая граница числа
  count:=gr-gl-1;  // количество символов числа
  str:=Copy(s,gl+1,count); // забрать из строки число
  Delete(str, pos(' ', str), 1); // удалить пробел
  result:=StrToInt(str); // вернуть выбранное число
end;

const
    ots = 30;  // отступы от краев экрана
    pole = 15; // поля вокруг столбиков диаграммы
var txt: text; // переменная для работы с файлом
    lst: array of string; // массив строк файла
    arr: array of TLang; // массив структур, выбранных из файла
    lst_count, count,  i: Integer;
    s: string; // строка из файла
    // масштабы по горизонтали и по вертикали
    msX: integer; // например, 600 пикселей / 10 фильмов = 60
    msY: real; // например, 400 пикселей / 80000 голосов = 0,005

begin
  AssignFile(txt, 'kinopoisk.txt'); Reset(txt);
  lst_count:=0; // сбрасываем размер массива строк
  while not eof(txt) do // пока не конец файла
  begin
    readln(txt, s); // читаем очередную строку из файла
    inc(lst_count); // увеличиваем размер массива строк
    SetLength(lst, lst_count); // увеличиваем сам массив
    lst[lst_count-1]:=s; // запоминаем строку из файла
  end;
  CloseFile(txt); // закрываем файл
  // теперь все данные находятся в массиве
  // запись про 1 фильм занимает 4 строки
  count := 0;
  for i:=0 to lst_count-1 do // для всех элементов массива
    if (i+1) mod 4 = 0 then  // делаем выборку по 4 строки
    begin
      inc(count); // 4 строки дают 1 новый элемент
      SetLength(arr, count); // увеличиваем размер массива
      // извлекаем название и складываем в поле name
```

```
      arr[count-1].name := Copy(lst[i-2], 1, pos('(', lst[i-2])-2);
      // извлекаем количество и складываем в поле number
      arr[count-1].number := parse(lst[i]);
    end;
  // теперь у нас есть массив структур про фильмы

  SetWindowSize( 800, 600 ); // назначаем размеры экрана
  ClearWindow(rgb(255,255,255)); // устанавливаем цвет заливки фона

  // вычисляем масштаб, чтобы диаграмма заняла полный размер экрана
  msX := (WindowWidth-2*ots) div count; // масштаб по горизонтали
  msY := (WindowHeight-2*ots) / arr[0].number; // по вертикали

  SetFontColor(rgb(0,0,55)); // цвет надписей

  for i:=0 to count-1 do // для всех выбранных фильмов
  begin
    SetBrushColor( RGB(0,150,50) ); // цвет заливки столбика
    FillRectangle(ots+i*msX+pole,
        WindowHeight-ots-Round(arr[i].number*msY),
        ots+(i+1)*msX-pole, WindowHeight-ots);

    SetBrushColor( RGB(255,255,255) ); // цвет заливки фона текста
    // выводим название фильма
    TextOut(ots+i*msX+pole, WindowHeight-ots-
        Round(arr[i].number*msY)-TextHeight(arr[i].name),
        arr[i].name);
  end;
  // рисуем оси координат
  SetPenColor(RGB(180,120,0)); Pen.Width := 2;
  Line(ots,WindowHeight-ots,WindowWidth-ots,WindowHeight-ots);
  Line(ots,ots,ots,WindowHeight-ots);

  SaveWindow('graph.bmp'); // сохраняем диаграмму в файл
end.
```

Диаграммы строятся по дискретным табличным значениям. В свою очередь, графики функций выглядят непрерывными и строятся исходя из расчетов значений функции во множестве точек графика. Но в реальности экран монитора дискретный, поэтому можно аппроксимировать график совокупностью прямых отрезков малой величины, то есть достаточно с некоторым шагом рассчитать значения функции в отдельных точках и соединить их прямыми линиями.

Шаблон #10 показывает пример **построения графика функции**:

$$y = (x/6)^5 + \cos(x) - 2$$

В шаблоне показано как сформировать массив координат точек графика, затем на его основе определить масштабы отображения гра-

фика по оси абсцисс и ординат и далее отобразить функцию кусочно-непрерывным графиком.

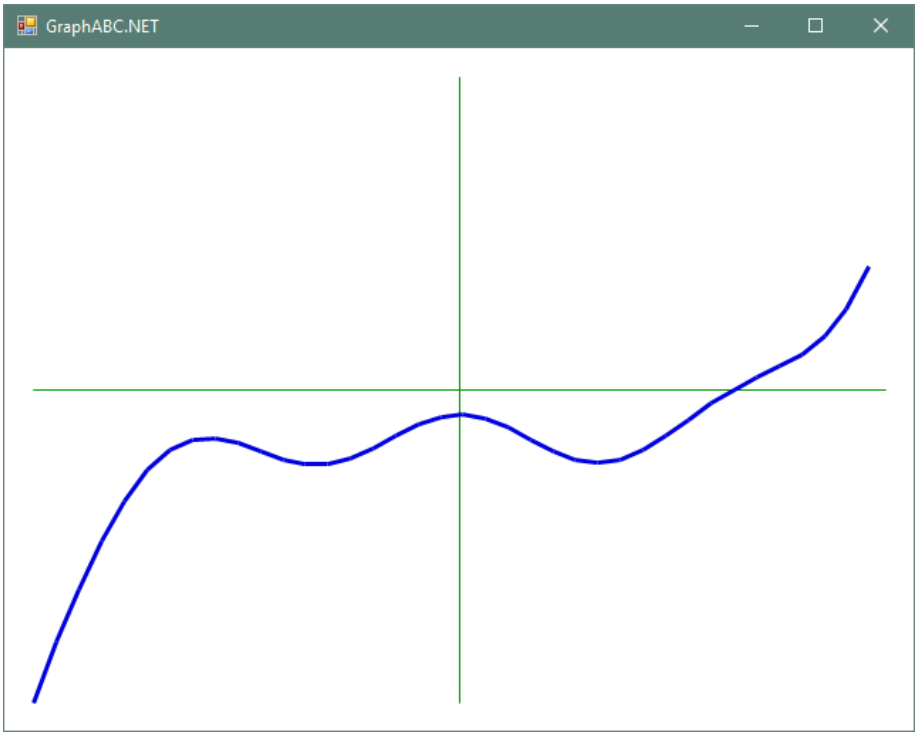
Текст шаблона #10:

```
uses GraphABC; // чтобы сделать exe-шник нажми Ctrl+F9

const
  step=10;  // шаг вывода значений
  ots=20;   // отступ от края
  dx=0.5;   // шаг графика

Type // координаты точек графика
  Txy = record
  x: real;
  y: real;
end;

function func(x: real): real;
begin // сама функция
  result := power(x/6,5) + cos(x) - 2;
end;

function arrT(L,R: real): array of Txy;
var i: integer; x,y: real;
    arr: array of Txy;
begin // построение массива с координатами точек графика
  x:=L; // начальная координата слева
  i:=0; // начальное количество точек графика
  repeat // цикл заполнения массива координат
    inc(i); SetLength(arr,i); // увеличиваем размер массива
    y:=func(x); // вычисляем значение функции в новой точке
    arr[i-1].x:=x; // записываем координату x
    arr[i-1].y:=y; // записываем координату y
    x:=x+dx; // делаем шаг вправо по оси x
  until x>R; // пока не дойдем до правой границе
  result:=arr; // возвращаем массив координат
end;

var sx,sy,i: integer;
    x,y,mtX,mtY,int_L, int_R,min_Y,max_Y,mm_Y: real;
    arr: array of Txy;

begin
  int_L:=-3*Pi; int_R:=+3*Pi; // левая и правая границы

  arr:=arrT(int_L,int_R); // сформируем массив значений

  // найдем минимум и максимум
  min_Y:=arr[0].y; max_Y:=arr[0].y;
  for i:=1 to arr.Length-1 do
  begin
   if arr[i].y>max_Y then max_Y:=arr[i].y;
```

```
    if arr[i].y<min_Y then min_Y:=arr[i].y;
  end;
  // подберем масштаб графика
  if abs(max_Y)>abs(min_Y)
    then mm_Y:=abs(max_Y)
    else mm_Y:=abs(min_Y);
  mtX:=(int_R-int_L)/(WindowWidth-2*ots); // масштаб по горизонтали
  mtY:=(mm_Y)/((WindowHeight-2*ots)/2); // масштаб по вертикали

  sy:=trunc(WindowHeight/2);  // середина экрана по высоте
  sx:=trunc(WindowWidth/2);   // середина экрана по ширине

  ClearWindow(RGB(0,0,255));   // зальём фон
  SetPenColor(RGB(5,222,5)); // установим цвет осей
  Line(ots,sy,WindowWidth-ots,sy);  // ось X
  Line(sx,ots,sx,WindowHeight-ots); // ось Y

  SetPenWidth(3); SetPenColor(RGB(255,255,5)); // установим цвет гра-
фика

  i:=0; // нарисуем линии графика
  MoveTo(sx+trunc(arr[i].x/mtX),sy-trunc(arr[i].y/mtY));
  for i:=1 to arr.Length-1 do // линии между точками графика
    LineTo(sx+trunc(arr[i].x/mtX),sy-trunc(arr[i].y/mtY));
end.
```

Обратите внимание, что значения y вещественные числа, а координаты точек на экране – целые, так как пиксели не могут быть не целыми. Поэтому при выводе линий значения функции приходится округлять до целого числа (функции round, trunc и т.п.).

Результаты работы этого шаблона:



Шаблон #11 - произвести анализ данных и найти экстремум функции на интервале:

При анализе функций часто встречается задача поиска минимума или максимума значения функции на определенном интервале. Если функция несложная, то можно обойтись поиском производной от функции и решением соответствующего уравнения (когда производная равна нулю). Однако этот процесс может быть довольно сложным, в то время как поиск экстремума функции программным способом прост и занимает мало времени. По поводу возражений о приблизительности вычислений можно заметить следующее: если задача решается для практических нужд, то и получаемые данные следует рассматривать в рамках практики, то есть в программе можно указать ту точность вычислений, которая удовлетворит заказчика.

В данном шаблоне для поиска точки минимума используется метод бисекции, когда интервал делится пополам и для последующего поиска выбирается та половина, в которой значения функции меньше. Процесс последовательного деления интервала пополам продолжается до тех пор пока не будет достигнута заданная точность вычисления значения x (когда разница между левой и правой границами интервала станет не больше, чем установленная погрешность вычисления).

```
uses Crt; // ищем экстремум функции
const e=0.0001; // точность

function func(x: real): real;
begin
  result := power(x/6, 5)+cos(x)-2;
end;

var xL,xR,yL,yR,x,y: real;
begin
  xL:=0; xR:=3*Pi; // левая и правая границы
  yL:=func(xL); yR:=func(xR);
  x:=(xL+xR)/2;

  repeat // ищем минимум
    if (yR<yL) // если график идет вниз
      then begin xL:=x; yL:=func(xL); end  // то пойду направо
      else begin xR:=x; yR:=func(xR); end; // иначе пойду налево
```

```
    x:=(xL+xR)/2;   // середина нового интервала
    writeln(x:0:6); // выводим для контроля значений
  until (xR-xL<=e); // до достижения заданной точности
  y:=func(x); // значение функции в найденной точке минимума
  writeln(chr(10), ' x = ', x:0:3,
          chr(10), ' y = ', y:0:3);
  readkey; // пауза
end.
```

Шаблон #12 - произвести анализ данных и найти пересечение функции с осью абсцисс.

В данном шаблоне представлено решение аналогичной задачи по анализу функции с поиском точки пересечения графика с осью абсцисс методом бисекции. Отличием является только момент выбора половины интервала из исходного – выбираем ту часть где есть нулевое значение функции:

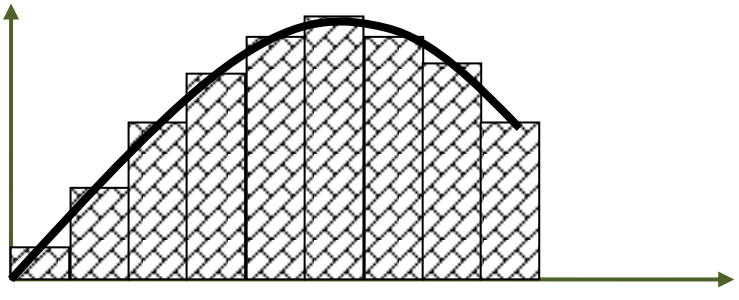
```
uses Crt; // ищем где функция = 0
const e=0.0001; // точность

function func(x: real): real;
begin
  result := power(x/6, 5)+cos(x)-2;
end;

var xL,xR,yL,yR,x,y: real;
begin
  xL:=0; xR:=3*Pi; // левая и правая границы
  x:=(xL+xR)/2; y:=func(x); // делим интервал пополам

  repeat // ищем пересечение
    if (y<0) // выбираем часть с нулём
      then xL:=x  // пойду направо, там 0
      else xR:=x; // пойду налево, там 0
    x:=(xL+xR)/2; y:=func(x); // середина нового интервала
    writeln(x:0:6); // выводим для контроля значений
  until (xR-xL<=e); // до достижения заданной точности

  writeln(chr(10), ' x = ', x:0:3,
          chr(10), ' y = ', y:0:3);
  readkey;
end.
```

Шаблон #13 - произвести анализ данных и вычислить интеграл на интервале (как площадь под линией графика).

Если функция сложная, то могут возникнуть проблемы с вычислением её интеграла на заданном диапазоне. В этом случае можно воспользоваться следующим шаблоном:

```
uses Crt; // ищем интеграл как площадь под кривой

function func(x: real): real;
begin
  result := power(x/6, 5)+cos(x)-2;
end;

var xL,xR,x,y,dx,s,ds: real; step,i: integer;
begin
  xL:=0; xR:=6.063; // левая и правая границы
  step:=100; // начальное количество шагов
  dx:=(xR-xL)/step; // ширина прямоугольника
  x:=xL; s:=0; // начальные значения

  for i:=1 to step do // суммируем прямоугольники
  begin
    y:=func( x+dx/2 ); // высота прямоугольника
    ds:=Abs(y*dx); // площадь прямоугольника
    s:=s+ds; // накапливаем интеграл
    x:=x+dx; // шагаем по x дальше
  end;

  writeln('s = ', s:0:6);
end.
```

В шаблоне представлен подход аппроксимации площади непрерывной функции совокупностью площадей прямоугольников:



Обратите внимание, что при вычислении площади прямоугольника используется функция, возвращающая модуль числа (то есть без знака). Так сделано ввиду того, что функция может иметь отрицательные значения, но площадь фигуры всегда положительна.

Шаблон #14 - произвести анализ данных и вычислить интеграл с заданной точностью.

Очевидно, что чем меньше ширина прямоугольников, тем лучше аппроксимация и точнее вычисление интеграла. Поэтому имеет смысл рассмотреть следующий шаблон, в котором за основу взят цикл вычисления интеграла из предыдущей программы, но добавлен внешний цикл по ширине прямоугольников. На каждой итерации происходит сравнение вычисленного интеграла с предыдущим значением ширины прямоугольников и с уменьшенным. Цикл завершается при достижении заданной точности, то есть когда разница между вычисленными интегралами на двух соседних шагах укладывается в установленные границы. Начальную ширину прямоугольников может установить программист через переменную step, которая определяет количество шагов вычисления интеграла (количество прямоугольников). После каждой итерации внешнего цикла программы происходит увеличение количества прямоугольников.

```
uses Crt; // ищем интеграл как площадь под кривой
const e=0.0001; // точность

function func(x: real): real;
begin
  result := power(x/6, 5)+cos(x)-2;
end;

var xL,xR,x,y,dx,s,sold,ds: real; step,i: integer;
begin
  xL:=0; xR:=6.063; // левая и правая границы
  step:=50; // начальное количество шагов
  s:=0;
  repeat
    sold:=s;
    dx:=(xR-xL)/step; // ширина прямоугольника
    x:=xL; s:=0; // начальные значения
    for i:=1 to step do // суммируем прямоугольники
    begin
      y:=func( x+dx/2 ); // высота прямоугольника
      ds:=Abs(y*dx); // площадь прямоугольника
      s:=s+ds; // накапливаем интеграл
      x:=x+dx; // шагаем по x дальше
    end;
    step:=step+10; // увеличиваем количество шагов
  until (abs(s-sold)<=e); // проверяем точность
  writeln('s = ', s:0:5);
end.
```

Тема 4. Символы и строки

Шаблон #15 - построить таблицу символов. В данном шаблоне показано как можно вывести таблицу символов:

1) на экран посимвольно,

2) в файл построчно,

3) в файл из массива строк.

Обращаю ваше внимание, что в старых версиях Паскаля можно было обойтись функцией `Chr(i)`, которая возвращала символ в однобайтной кодироваке по его порядковому номеру в таблице символов. В настоящее время следует использовать функцию `ChrAnsi(i)`, если вы хотите корректно выводить национальные символы. Также обратите внимание на удобный метод `WriteAllLines`, который непосредственно пишет все элементы массива в файл (без объявления файловой переменной, без открытия файла, без последующего его закрытия, без организации цикла).

Текст шаблона #15:

```
uses Crt; // вывести таблицу символов
var i, count: integer;
    Str: string;
    f: Text;
    arr: array of string;
begin
  TextColor(Yellow); // цвет текста
  TextBackground(Blue); // цвет фона
  ClrScr; // залить цветом фона экран

  // ChrAnsi(i) - возвращает символ в кодировке ANSI
  // выводим на экран посимвольно
  for i:=32 to 255 do // без служебных символов
    if (i+1) mod 16 = 0  // по 16 в строке
      then Writeln(ChrAnsi(i)) // в конце Enter
      else Write(ChrAnsi(i));

  // выводим в файл построчно
  AssignFile(f, 'tablecode.txt'); Rewrite(f);
  str:=''; // стираем строку для накопления символов
  for i:=32 to 255 do
  begin
    str:=str+ChrAnsi(i); // накапливаем символы в строке
```

```
    if (i+1) mod 16 = 0 then
    begin
      Writeln(f, str); // пишем в файл строку символов
      str:=''; // стираем строку для следующей итерации
    end;
  end;
  CloseFile(f);

  // выводим в файл массив строк
  count:=0; // длина массива строк
  str:='';
  for i:=32 to 255 do
  begin
    str:=str+ChrAnsi(i);
    if (i+1) mod 16 = 0 then
    begin
      Inc(count);
      SetLength(arr,count); // увеличиваем массив
      arr[count-1]:=str; // записываем в последний элемент
      Str:='';
    end;
  end;
  // все элементы массива пишем в файл построчно
  WriteAllLines('codelines.txt', arr);

  ReadKey;
end.
```

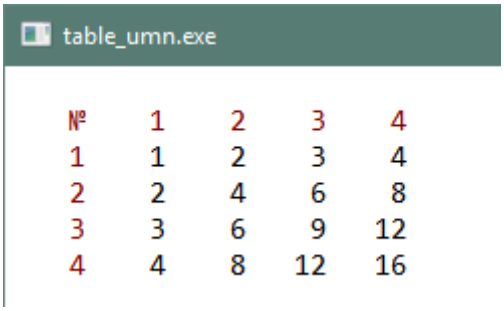
Один из результатов работы данной программы:

```
!"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~□
ЂЃ‚ѓ„…†‡€‰Љ‹ЊЌЋЏ
ђ‘’“”•–—□™љ›њќћџ
 ЎўЈ¤Ґ¦§Ё©Є«¬-®Ї
°±Ііґµ¶·ё№є»јЅѕї
АБВГДЕЖЗИЙКЛМНОП
РСТУФХЦЧШЩЪЫЬЭЮЯ
абвгдежзийклмноп
рстуфхцчшщъыьэюя
```

#task_15

Попробуйте *самостоятельно* на основе данного шаблона разработать программу, которая выводит квадратную таблицу умножения размерностью N (1<N<15). Саму размерность может определить поль-

зователь программы в ходе диалога. Программа спрашивает у пользователя: 'Введите размерность таблицы (1..15) - '. Если пользователь ввел число не из указанного диапазона, то программа возвращается к своему вопросу до тех пор, пока не будет введено разрешенное значение. Далее с экрана стирается диалог с пользователем и выводится искомая таблица умножения. Обратите внимание, что названия строк и столбцов должны быть выделены цветом. Ниже приведен пример вывода таблицы умножения:



Шаблон #16 - проверить строку на «Палиндром» с помощью специальной функции:

Обычно строку воспринимают как массив символов. Индексация символов в строке существенно облегчает работу. Далее приведен пример обработки строки с целью проверки её на палиндромность. Палиндром - это такая строка, которая читается слева-направо и справа-налево одинаково, без учета знаков препинания, пробелов и иных символов, а также регистра букв.

```
uses Crt;

function palindrom(s: string): Boolean;
const z = ',. !?-:/\(){}[]';
var s0, s1, s2: string; i: Byte;
begin
  s0:=LowerCase(s); // переводим все символы в нижний регистр

  s1:='';
  for i:=1 to Length(s0) do
    if Pos(s0[i], z) = 0 // забираем только разрешенные символы
      then s1 := s1 + s0[i];

  s2:='';
  for i:=Length(s1) downto 1 do
```

```
    s2 := s2 + s1[i]; // переворачиваем строку

  Result := s1=s2;
end;

var s: string;
begin
  s:='Аргентина манит негра!';
  if palindrom(s)
    then write(s, ' - палиндром')
    else write(s, ' - не палиндром');
  readkey;
end.
```

#task_16

Доработка эффективности кода.

Надо отметить, что во многих случаях, при реализации практических проектов, ставится задача на оптимизацию кода (снизить потребляемые ресурсы, увеличить скорость выполнения кода). Например, в данной функции завершающая часть кода посвящена тому, чтобы создать перевернутую строку и сравнить её с исходной. То есть каждый раз резервируется место под лишнюю переменную `s2` и гонится полный цикл по всей длине строки `Length(s1)`, с последующим сравнением двух строк:

```
s2:='';
for i:=Length(s1) downto 1 do
  s2 := s2 + s1[i];
Result := s1=s2;
```

Однако можно было бы поступить иначе, если просто сравнивать последовательно первый и последний символ, потом второй и предпоследний и так далее до середины строки. При таком алгоритме можно было бы не создавать лишнюю переменную и сократить перебор в два раза. Для одной проверки может быть это и несущественно, но это будет иметь большое значение при обработке значительной по объему базы (особенно в рамках работ по направлению BIG DATA).

Проведем имитацию работы с базой данных. Создайте входной файл `palindroms.txt`, расположите его в той же папке, что и программа и наполните его палиндромами:

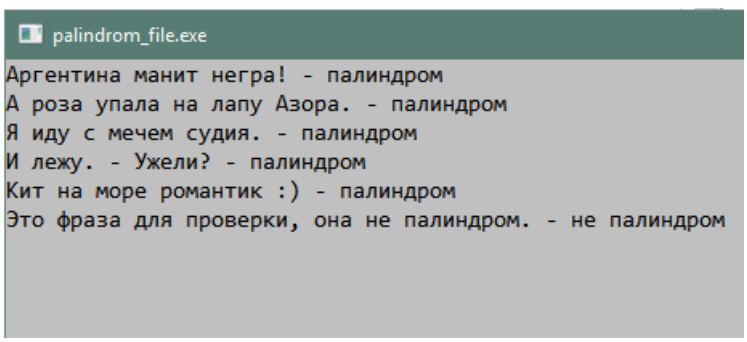
```
Аргентина манит негра!
А роза упала на лапу Азора.
Я иду с мечем судия.
```

```
И лежу. - Ужели?
Кит на море романтик :)
Это фраза для проверки, она не палиндром.
```

Всего шесть строк, включая и последнюю - для контроля. Если у вас под рукой доступ в интернет, то наберите ещё 10-20 палиндромов из сети и разместите их во входном файле. Немного преобразим программу - **Шаблон #16+**, чтобы она работала с массивом строк:

```
uses Crt; // Шаблон #16+

function palindrom(s: string): Boolean;
const z = ',. !?-:/\(){}[]';
var s0, s1, s2: string; i: Byte;
begin
  s0:=LowerCase(s); // переводим все символы в нижний регистр

  s1:='';
  for i:=1 to Length(s0) do
    if Pos(s0[i], z) = 0 // забираем только разрешенные символы
      then s1 := s1 + s0[i];

  // эта часть функции неэффективна
  s2:='';
  for i:=Length(s1) downto 1 do
    s2 := s2 + s1[i]; // переворачиваем строку

  Result := s1=s2;
end;

var s: string;
    arr: array of string;
begin
  arr:=ReadAllLines('palindroms.txt'); // из файла в массив строк
  foreach s in arr do // для всех элементов массива
    if palindrom(s)
      then writeln(s, ' - палиндром')
      else writeln(s, ' - не палиндром');
  readkey;
end.
```

Апробируйте работоспособность этого кода, у вас должен получиться вот такой результат:



Теперь ваша задача состоит в том, чтобы *самостоятельно* доработать функцию `palindrom`, как было объяснено ранее, то есть отказаться от создания переменной `s2`, для хранения перевернутой строки и проверять совпадение символов, начиная с концов строки и до её середины (для сокращения времени потоковой проверки).

Следующий **шаблон #17** посвящен решению задачи перевода строки с числовыми данными в массив чисел.

Во многих современных языках программирования есть набор функций для парсинга строк (выборки нужной информации). Но не всегда стандартных наборов функций хватает на все возникающие виды практических задач, поэтому нужно уметь понимать и самому создавать базовые методы работы со строками, например, функция для парсинга строки с числами в массив чисел:

```
uses Crt;
// функция парсинга: из строки в массив чисел
function StrToArr(s: string; c: char): array of real;
var arr: array of real;
    gl, gr, count: integer;
    str: string;
begin
  count:=0; // количество чисел в строке
  gl:=1; // граница слева
  repeat
    gr:=pos(c, s, gl); // граница справа текущего числа
    if gr=0 // если это последнее число в строке
      then gr:=length(s)+1; // граница по длине строки
    str:=Copy(s, gl, gr-gl); // вырезать текущее число
    if pos(',', str)>0 // если запятая вместо точки
      then str[pos(',', str)]:='.'; // то заменить на '.'
    inc(count); SetLength(arr, count); // увеличить массив
    arr[count-1]:=StrToFloat(str); // записать в массив
    gl:=gr+1; // сдвинуть левую границу
  until gl>length(s); // пока не конец строки
  Result := arr; // вернуть массив чисел
end;

var f: Text;
    s: string;
    arr: array of real;
    i: integer;
    sum: real;
begin // найти среднее-арифметическое чисел в строке
  AssignFile(f, 'numeric.txt'); Reset(f);
```

```
  Readln(f, s); // прочитать из файла строку для парсинга
  Close(f);
  try // сделать попытку парсинга строки
    // преобразовать строку по указаному символу разделителю
    arr:=StrToArr(s, '|');
    sum:=0; // сбросить сумму
    foreach var r in arr do // перебрать элементы массива
    begin
      WriteLn(chr(9), r); // выводим для контроля
      sum:=sum+r; // накапливаем сумму
    end;
    WriteLn(chr(10),'среднее = ', sum/arr.Length:0:3);
  except // если неудачный парсинг
    WriteLn('Ошибка чтения');
  end;
  readkey;
end.
```

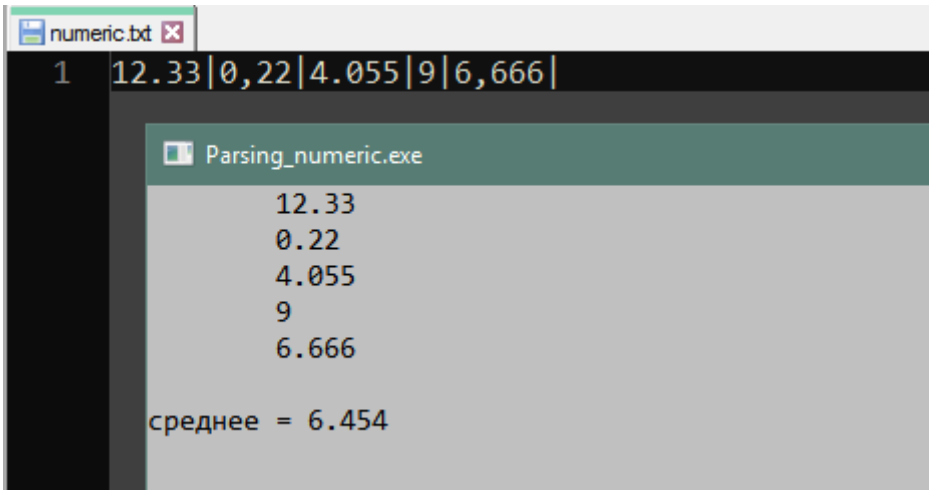
Входной поток данных находится в файле numeric.txt в одной строке в виде вещественных чисел, записанных через символ-разделитель, например, «Пробел», «Табуляция», «|» или любой иной, выбранный пользователем. Количество чисел в строке заранее неизвестно, поэтому будем работать с динамической структурой. В шаблоне представлена функция `StrToArr`, в которую можно передать строку и символ-разделитель и получить обратно динамический массив. В функции предусмотрена проверка, некорректного для Паскаля, ввода вещественного числа, когда в качестве выделения целой и дробной частей используют символ «запятая», тогда как Паскаль воспринимает вещественное число, написанное через символ «точка». Кроме того в функции предусмотрена возможность чтения последнего числа во входной строке, даже если оно не завершается символом-разделителем. Посмотрите на примеры допустимых строк:

```
12.33|0.22|4.055|9|6.666|
12.33|0.22|4.055|9|6.666
12,33|0,22|4.055|9|6.666|
12,33 0,22 4,055 9 6,666
```

Возьмите любую из представленных строк, можете ещё добавить к ней чисел, и поместите её в файл numeric.txt. Файл и программу расположите в одной папке и можете провести испытания шаблона. Об-

ратите внимание на то, что перебор всех значений массива осуществляется с помощью специального оператора цикла `foreach`. Его отличие от стандартного параметрического в том, что не нужно определять «вручную» границы изменения параметра.

Ниже вы можете увидеть содержимое входного файла и результаты работы с ним:



#task_17

Вы, наверное, обратили внимание, что в шаблоне используется защищаемый блок `try`, так как велика вероятность обнаружения в строке некорректного формата записи. Вариантов ошибочного ввода может быть множество и все не учесть заранее. Ранее в функции `StrToArr` мы рассмотрели только два возможных варианта неточности ввода. Предлагаю вам добавить *самостоятельно* в функцию обработку (исправление) ещё одной ошибки: пусть из-за дребезга клавиатуры (или тремора рук) в строке могут появляться такие числа, в которых символ «точка» или «запятая» дублируются (или даже повторяются большее количество раз). Возникает задача – автоматизировать исправление этого вида ошибок, то есть заменять такие числа:

```
12..33 4,,055 6....666
```

на такие:

```
12.33 4.055 6.666
```

Тема 5. Поиск и сортировка

ПОИСК В ОТСОРТИРОВАННОМ СПИСКЕ

Ранее мы уже подробно обсуждали метод бисекции, когда весь диапазон поиска делится на каждой итерации пополам для сужения диапазона поиска. На каждом шаге выбирается та половина диапазона, в которую входит искомый результат. Данный подход особенно эффективен для поиска б большом массиве значений, так как может уменьшить количество шагов поиска на два и более порядков (то есть в 100 или более раз). Применим данный подход для поиска информации в некоторой базе данных – это может быть информация о жителях города-миллионника или поиск слова в словаре (в помощь любителю кроссвордов). Пример входного файла:

```
1 9601.05 a misc
2 66.69 абсолютно adv
3 27.56 абсолютный adj
4 67.84 август noun
5 33.99 авиация noun
6 81.04 автобус noun
7 108.72 автомат noun
8 84.49 автомобиль noun
9 157.61 автор noun
10 32.04 авторитет noun
```

Обратите внимание, что в каждой строке четыре подстроки: номер по порядку, рейтинг слова, само слово, часть речи. Следовательно, нужно в программу добавить функцию парсинга строк, такую, чтобы она одну строку разбивала на массив подстрок, из которого, в свою очередь, можно будет брать интересующий нас элемент – само слово. Ранее мы уже разрабатывали функцию парсинга из строки в массив чисел. Возьмём её за основу и немного упростим для нашего следующего **шаблона #18**, решающего оптимальным образом задачу поиска строки, её позиции и рейтинга. Алгоритм работы шаблона такой: сначала считать все строки файла в массив строк, потом запустить цикл поиска искомой строки, в котором делим интервал поиска пополам, сравниваем искомое слово с тем, что нашлось в середине диапазона, если искомое слово больше, то переходим в нижнюю половину, если искомое слово меньше, то переходим в верхнюю половину, если искомое слово равно, то выводим на экран ответ

```
uses Crt;
// функция парсинга: из строки в массив подстрок
function StrToArrStr(s: string; c: char): array of String;
var arr: array of String;
    gl, gr, count: integer;
    str: string;
begin
  count:=0; // количество подстрок в строке
  gl:=1; // граница слева
  repeat
    gr:=pos(c, s, gl); // граница справа текущей подстроки
    if gr=0 // если это последнее подслово в строке
      then gr:=length(s)+1; // граница по длине строки
    str:=Copy(s, gl, gr-gl); // вырезать текущую подстроку
    inc(count); SetLength(arr, count); // увеличить массив
    arr[count-1]:=str; // записать в массив
    gl:=gr+1; // сдвинуть левую границу
  until gl>length(s); // пока не конец строки
  Result := arr; // вернуть массив подстрок
end;

var arr: array of String;
    str, tmp: String;
    gL, gR, num, steps: integer;
    b: boolean;
begin
  arr:=ReadAllLines('5000слов.txt');
  str:='программа'; // искомое слово
  gL:=0; gR:=arr.Length-1; // границы
  steps:=0; // кол-во шагов поиска
  b:=false; // слово пока не найдено

  repeat
    num:=(gL+gR) div 2; // позиция в середине списка
    tmp:=StrToArrStr(arr[num],' ')[2]; // слово из позиции
    inc(steps); // увеличиваем кол-во шагов
    if str>tmp then gL:=num; // слово в нижней половине
    if str<tmp then gR:=num; // слово в верхней половине
    if str=tmp then // слово нашли
      begin b:=true; break; end;
    writeln(gL:8,gR:8); // выводим для контроля значений
  until (gR-gL<2); // интервал поиска сузился

  writeln('всего шагов поиска - ', steps);
  if b then
    begin
      writeln('слово - ', str);
      writeln('позиция - ', num);
      writeln('рейтинг - ', StrToArrStr(arr[num],' ')[1]);
    end
    else writeln('слово - ', str, ' - не найдено');
end.
```

СОРТИРОВКА

В таких задачах, как сортировка не лишним будет оценить трудоёмкость и время исполнения алгоритма. Трудоёмкость понимают как количество выполняемых операций (минимальное, максимальное, среднее).

Шаблон для оценки времени исполнения алгоритма.

```
program P_Vrem; // Delphi 7
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Classes,
  Windows;

var time: Int64;
begin
  time:=GetTickCount; // Windows
  // тут сортировка
  time:=(GetTickCount-time);
  Writeln(time:5,' msek');
  readln;
end.
```

Шаблон генерации тестового файла со списком случайных чисел.

```
program P_List; // Delphi 7
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Classes,
  Windows;
const count = 100;
var lst: TStringList;
    i: Integer;
begin
  lst:=TStringList.Create;
  Randomize;
  for i:=1 to count do
    lst.Add(IntToStr(Random(100)));
  lst.SaveToFile('rnd.txt');
```

```
  readln;
end.
```

Шаблон использования встроенного метода сортировки.

```
program P_List_Sort; // Delphi 7
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Classes,
  Windows;
const count = 100;
var lst: TStringList;
    i: Integer;
begin
  lst:=TStringList.Create;
  lst.LoadFromFile('rnd.txt');
  for i:=0 to lst.Count-1 do
    if Length(lst[i])<2
      then lst[i]:=' '+lst[i];

  lst.Sort;

  for i:=0 to lst.Count-1 do
    Writeln(lst[i]);
  lst.SaveToFile('rnd+.txt');

  readln;
end.
```

Шаблон сортировки методом обмена («пузырьковая»).

```
program P_Sort_Buble; // Delphi 7
{$APPTYPE CONSOLE}
uses SysUtils; // пузырьковая сортировка - обменом

procedure swap(var a: Integer; var b: integer);
var tmp: Integer;
begin
  tmp:=a; a:=b; b:=tmp;
end;

var arr: array of Integer;
    i, j, count: Integer; f: Text;
begin
  AssignFile(f, 'rnd.txt'); Reset(f);
  count:=0;
```

```
  while not Eof(f) do
  begin
    Inc(count);
    SetLength(arr, count);
    Readln(f, arr[count-1]);
  end;
  CloseFile(f);

  for i:=count-1 downto 2 do
  for j:=0 to i-1 do
    if arr[j]>arr[j+1]
      then swap(arr[j], arr[j+1]);

  for i:=0 to Count-1 do
    Writeln(arr[i]);

  readln;
end.
```

Шаблон сортировка методом выбора.

```
program P_Sort_; // Delphi 7
{$APPTYPE CONSOLE}
uses SysUtils; // сортировка выбором

procedure swap(var a: Integer; var b: integer);
var tmp: Integer;
begin
  tmp:=a; a:=b; b:=tmp;
end;

var arr: array of Integer;
    i, j, count, imax: Integer; f: Text;
begin
  AssignFile(f, 'rnd.txt'); Reset(f);
  count:=0;
  while not Eof(f) do
  begin
    Inc(count);
    SetLength(arr, count);
    Readln(f, arr[count-1]);
  end;
  CloseFile(f);

  for i:=count-1 downto 1 do
  begin
  imax:=0;
```

```
  for j:=0 to i do
    if arr[j]>arr[imax]
      then imax:=j;
  swap(arr[imax], arr[i]);
  end;

  for i:=0 to Count-1 do
    Writeln(arr[i]);

  readln;
end.
```

Шаблон сортировки методом вставки.

```
program P_Sort_Insert; // Delphi 7
{$APPTYPE CONSOLE}
uses SysUtils; // сортировка вставкой

procedure swap(var a: Integer; var b: integer);
var tmp: Integer;
begin
  tmp:=a; a:=b; b:=tmp;
end;

var arr: array of Integer;
    i, j, count, tmp: Integer; f: Text;
begin
  AssignFile(f, 'rnd.txt'); Reset(f);
  count:=0;
  while not Eof(f) do
  begin
    Inc(count);
    SetLength(arr, count);
    Readln(f, arr[count-1]);
  end;
  CloseFile(f);

  for i:=2 to count-1 do
  begin
    tmp:=arr[i];
    for j:=i downto 1 do
      if tmp<arr[j-1]
        then arr[j]:=arr[j-1]
        else Break;
    arr[j]:=tmp;
  end;
```

```
  for i:=0 to Count-1 do
    Writeln(arr[i]);

  readln;
end.
```

Шаблон скоростной сортировки (методом бисекции).