	Федеральное государственное бюджетное образовательное учреждение высшего образования «Башкирский государственный аграрный университет»	Приложение к ОПОП ВО
		Методические указания к практическим занятиям

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к практическим занятиям и самостоятельной работе

по дисциплине

Б1.О.25.02 Разработка профессиональных приложений

Направление подготовки
13.03.02 Электроэнергетика и электротехника

Профиль подготовки
Электроснабжение

Квалификация (степень) выпускника
бакалавр

Уфа 2024

УДК 621.31:631(075.8)
ББК 40.76

Рекомендовано к изданию методической комиссией энергетического факультета (протокол № 7 от 21.03.2024 г.)

Составитель: д.т.н., профессор Галиуллин Р.Р.

Рецензент: к.т.н., доцент Кафиев И.Р.

Ответственный за выпуск: зав. кафедрой ЭТП к.т.н., доцент Ахметшин А.Т.

г. Уфа, БГАУ, Кафедра электроснабжения и автоматизации технологических процессов

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Дисциплина «Разработка профессиональных приложений» ведется в рамках модуля «Информационные технологии и программирование» составляет основу для подготовки специалистов по направлению «Электроэнергетика и электротехника».

Целью изучения дисциплины является формирование у будущего инженера системы знаний и практических навыков, необходимых для решения основных задач, связанных с созданием профессиональных приложений для ЭВМ в электроснабжении.

Задача дисциплины «Разработка профессиональных приложений» вытекают из требований ФГОС и квалифицированной характеристики выпускника и сводятся к изучению основных задач электроснабжения с применением алгоритмов и компьютерных программ.

Изучение дисциплины «Разработка профессиональных приложений» предполагает предварительное освоение следующих дисциплин в рамках общеобразовательного курса: основы информатики и математика.

Методические указания к практическим занятиям по разработке профессиональных приложений составлены в соответствии с типовой и рабочей программой дисциплины.

Материалы по всем темам составлены на основе действующих библиографических источников и нормативных документов в области систем электроснабжения.

В состав методических указаний дисциплины «Разработка профессиональных приложений» входит 4 темы, в каждой из которых приведены основные расчетные формулы и алгоритмы, примеры расчетов.

В результате решения практических задач по дисциплине «Разработка профессиональных приложений» студенты должны приобрести навыки и умения по созданию профессиональных приложений для ЭВМ, пригодных для практического применения в системах электроснабжения.

1 РАЗРАБОТКА МНОГООКОННЫХ И МНОГОМОДУЛЬНЫХ ПРИЛОЖЕНИЙ

Для использования в разрабатываемом приложении нескольких окон необходимо создать новые формы (*File* → *New* → *Form*), при этом создаются и новые файлы Unit.dfm и Unit.pas. По умолчанию все дополнительные окна невидимы. Возможно и создание дополнительных модулей с используемыми подпрограммами без формы. Для этого необходимо выполнить следующее: *File* → *New* → *Unit*. Чтобы можно было пользоваться дополнительными формами и ее модулями, необходимо в разделе *uses* вызывающего модуля прописать имя вызываемого модуля (*File* → *Use* → *Unit*).

Пример. Разработать приложение, которое позволяет вводить текст из нескольких строк, содержащих фамилию и имя (с произвольным количеством пробелов между ними), выводить в отельном окне фамилии или имена, находить в тексте заданное имя и выводить в отдельном окне информацию о разработчике программы. Разработать подпрограмму, удаляющую из строки лишние пробелы, и поместить ее в отдельный модуль (использовать эту подпрограмму при поиске имени и выводе фамилий или имен).

Алгоритм решения:

1. Разработка схемы взаимодействия форм и модулей проекта. В состав проекта входят четыре модуля и три окна:

• Unit1 — основной модуль (с основной формой-окном Form1, где расположены MainMenu и Memo для ввода списка фамилий с именами);

• Unit2 — дополнительный модуль (с дополнительной формой-окном Form2, где расположены две кнопки Button и ListBox для вывода списка имен или фамилий);

• Unit3 — дополнительный модуль (с дополнительной формой-окном Form3, где расположены две метки Label и Image для вывода информации о разработчике программы);

• Unit4 — дополнительный модуль (без формы), содержащий процедуру удаления лишних пробелов, вызываемую из основного модуля Unit1 и дополнительного модуля Unit2;

Стрелки на схеме указывают на модули и формы, использующиеся при активизации пунктов меню и кнопок (рис. 16).

2. Дизайн Form1 (см. рис. 15). При этом Memo1.Name → M.

3. Далее: *File* → *New* → *Form* (создается форма Form2 и Unit2).

4. Дизайн Form2 (см. рис. 15). При этом ListBox1.Name → LB.

5. Вернувшись в Unit1, делаем следующее:

File → *Use* → *Unit* (Unit1 связывается с Unit 2);

File → *New* → *Form* (создается форма Form 3 и Unit 3).

6. Дизайн Form3 (см. рис. 15).

• для размещения на форме изображения используется библиотечный компонент TImage (из раздела Additional);

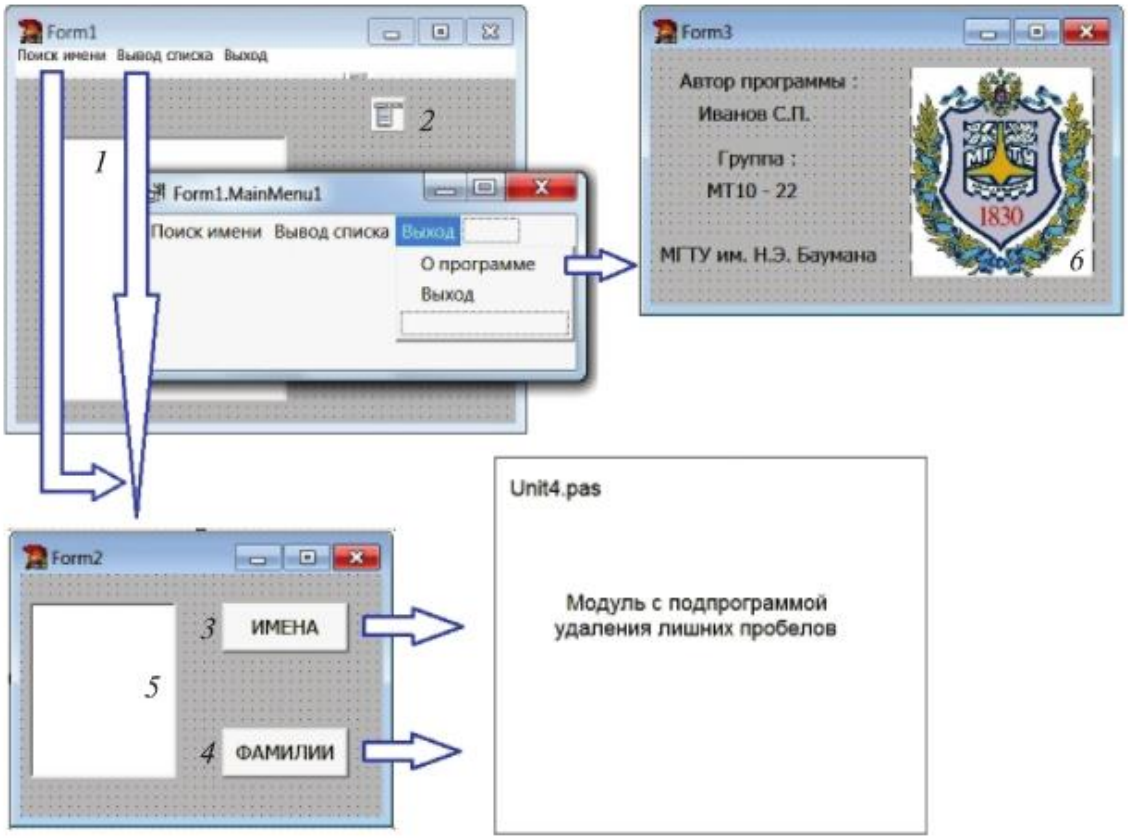

Рисунок 1.1 – Схема взаимодействия:
1 – Memo1; 2 – MainMenu1; 3 – Button1; 4 – Button2; 5 – ListBox1; 6 - Image1

• для помещения рисунка в Image1 используется его свойство Picture, с помощью которого загружается нужный файл с изображением;

• для масштабирования изображения под размер Image1 нужно установить для его свойств следующие значения: Autosize = False; Proportional = = True.

7. Вернувшись в Unit1, делаем следующее:

File → *Use* → *Unit* (Unit1 связывается с Unit 3);

File → *New* → *Unit* (создается Unit 4).

В этот модуль прописываем процедуру удаления лишних пробелов:

```
...
interface
procedure udalenie(var st:string);
implementation
procedure udalenie;
begin
  while pos('_ _', st)<> 0 do
     delete(st,pos('_ _',st),1);
  if st[1]='_'
     then
        delete(st,1,1);
```

```
  if st[length(st)]='_'
      then
        delete(st,length(st),1);
end;
```

8. Вернувшись в Unit1, делаем следующее:

File → *Use* → *Unit* (Unit1 связывается с Unit4).

Далее записываем обработчики событий для следующих пунктов меню:

Поиск:

```
...
var
  i:byte;
  naiden:boolean;
  st, name:string;
begin
  name:=InputBox('Поиск','Искомое имя -  ','');
  naiden:=false;
  for i:=0 to M.Lines.Count-1 do
    begin
      st:=M.Lines[i];
      udalenie(st);
      if name=copy(st,pos('_',st)+1, length(st)-pos('_',st))
         then
            begin
               naiden:=true;
               break;
             end;
          end;
   if naiden
     then
       ShowMessage('Это имя в строке №'+IntToStr(i+1))
     else
        ShowMessage('Нет этого имени');
end;
```

Вывод списка:

```
...
begin
  Form2.Visible:=true;
end;
```

О программе:

```
...
begin
  Form3.Visible:=true;
end;
```

Выход:

```
...
begin
  close;
end;
```

9. Перейдя в Unit2, делаем следующее:

File → *Use* → *Unit* (Unit2 связывается с Unit1);

File → *Use* → *Unit* (Unit2 связывается с Unit4).

Далее записываем обработчики событий для следующих кнопок:

Имена:

```
...
var
  i:byte;
  st:string;
begin
  LB.Clear;
  for i:=0 to Form1.M.Lines.Count-1 do
begin
    st:=Form1.M.Lines[i];
    udalenie(st);
    LB.Items.Add(copy(st,pos('_',st)+1, length(st)-
pos('_',st));
end;
end;
```

Фамилии:

```
...
var
  i:byte;
  st:string;
begin
  LB.Clear;
  for i:=0 to Form1.M.Lines.Count-1 do
begin
    st:=Form1.M.Lines[i];
    udalenie(st);
    LB.Items.Add(copy(st,1,pos('_',st)-1);
end;
end;
```

10. И в конце сохраняем весь проект:

File → *Save All*.

Примечание. В качестве окна с информацией о программе можно использовать стандартное окно Delphi (*File* → *New* → *Other* → *Deplhi Files* → *AboutBox*).

Задание для самостоятельной работы

Выполнив это задание, можно подготовиться к рубежному контролю по теме «Файлы, многооконные и многомодульные приложения».

1. Разработать программу, содержащую меню, которое позволяет:

- вводить текст из нескольких строк;
- находить самую длинную строку;
- выводить в отдельном окне с помощью его кнопки количество пробелов или количество знаков препинания в тексте;
- выводить информацию об авторе программы в двух отдельных окнах (разработанном и стандартном).

2. Разработать подпрограмму, подсчитывающую количество пробелов в строке, и подпрограмму, подсчитывающую количество знаков препинания в строке, которые поместить в отдельный модуль.

2 РАЗРАБОТКА И ОТЛАДКА ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ СТРУКТУРИРОВАННЫХ ТИПОВ ДАННЫХ – ФАЙЛОВ

Тема посвящена организации вычислительных процессов с использованием внешних файлов, которые в программе ассоциируются с помощью структурированных файловых или "буферных" переменных.

Файл представляет собой именованный набор данных одного типа на внешнем носителе. Файл используется для размещения исходных данных, результатов вычислений, а также и текста исходной программы. Внешние носители - дискета, жесткий диск ("Винчестер"), лазерный диск (CD). Удобство использования файла заключается в возможности сохранения информации пользователем.

Файлы объявляются либо в подразделе VAR, либо в подразделе TYPE раздела описаний Паскаль-программы, а именно:

Var имя : file of базовый тип;

или

TYPE имя типа = FILE OF базовый тип;

Var имя : имя типа;

Обмен данными между основной программой и файлом на внешнем носителе осуществляется через файловую или буферную переменную. В описании "имя" есть имя файловой переменной. В Паскале различают текстовые, типизированные и нетипизированные файлы. В данной работе речь идет только о типизированных и текстовых файлах.

Текстовые файлы - это файлы последовательного доступа, т.е. для того, чтобы обратиться к какой-либо компоненте файла, необходимо перебрать все расположенные до нее; длина компоненты такого файла меняется. **Типизированные файлы** (описание приведено выше) - файлы прямого доступа, т.е. к любой компоненте файла можно обратиться непосредственно, поскольку длина каждой компоненты файла известна и задается типом буферной переменной. Таким образом, в текстовых файлах размер компоненты меняется, а для типизированных файлов он постоянен и определяется базовым типом. Описание текстового файла имеет вид:

VAR имя : TEXT;

В Паскале существуют два особых файла со стандартными именами INPUT (ввод) и OUTPUT (вывод). Оба они имеют тип TEXT. При вводе-выводе информации через стандартное устройство (экран дисплея) они формируются автоматически.

Существуют два вида взаимодействия с файлами: чтение и запись информации. Перед работой необходимо связать файловую переменную F с файлом по имени ”NAME” при помощи оператора

ASSIGN (F,’NAME’);

а после окончания работы файл закрыть

CLOSE (F);

причем имя файла может быть полным, т.е. указывается не только имя файла, но и полный путь к нему.

Чтение из файла данных:

RESET (F);

READ (F, список переменных программы для чтения, ввода);

Запись в файл данных

REWRITE (F);

WRITE (F, список переменных программы, для записи, вывода);.

Текстовые файлы открываются либо только для чтения, либо только для записи, типизированные файлы можно открывать одновременно и для записи, и для чтения. Помните, что файл, который необходимо открыть в программе, должен существовать на внешнем носителе. Кроме того, если вы пытаетесь открыть уже существующий типизированный файл с информацией оператором REWRITE (F); , то после его открытия вся информация будет стерта.

Функции и процедуры работы с файлами

для типизированных и текстовых файлов

процедура **ERASE(VAR F)** – стирает внешний файл, ассоциированный с буферной переменной F;

процедура **RENAME(VAR F, 'NEWNAME')** – переименовывает внешний файл;

функция EOF(VAR F): BOOLEAN **– имеет значение TRUE, если текущая позиция в файле находится за последним символом файла или если в файле нет компонент, в противном случае значение функции FALSE, т.е. функция определяет признак конца файла;**

для текстовых файлов

процедура APPEND(VAR F) **– открывает существующий внешний текстовый файл для добавления информации, текущий указатель файла устанавливается в его конец;**

функция **EOLN(VAR F):BOOLEAN** – определяет признак конца строки;

3) для типизированных файлов

функция FILEPOS(VAR F):LONGINT **– возвращает (определяет) текущую позицию указателя в файле;**

функция **FILESIZE(VAR F):LONGINT** –возвращет (определяет) текущий размер файла;

процедура SEEK(VAR F, N:LONGINT) **– устанавливает текущую позицию указателя файла на указанный элемент N;**

процедура TRUNCATE(VAR F) **– усекает размер файла до текущей позиции указателя в файле.**

Примеры

1. Создайте типизированный файл для записи данных вещественного типа и считайте из уже существующего в текущем каталоге текстового файла данные целого типа.

PROGRAM USER11;

```
VAR X1, X2, X3 :REAL;
I, I1, I2, I3 :INTEGER;
XR : FILE OF REAL;
YR :TEXT;
BEGIN
ASSIGN (XR, 'X.DAT'); {файл для записи }
ASSIGN (YR,'Y.DAT'); { файл для чтения }
WRITELN('ВВЕДИТЕ ТРИ ПЕРЕМЕННЫЕ');
READ (X1, X2, X3);
REWRITE(XR); WRITE (XR,X1,X2,X3); {запись в файл X.DAT}
RESET (YR);
READ(YR, I1, I2, I3); {чтение из файла Y.DAT}
I := I1 + I2 + I3;
WRITELN ('I1= ',I1:10,'I2 = ',I2:10,'I3 = ',I3:10, 'I=',I :10);
CLOSE (XR); CLOSE (YR); END.
```

2. Прочитайте из предварительно созданного в рабочем каталоге файла pr1.txt

информацию, представленную в следующем виде:

```
мама  49
папа   50
сын 18
дочь 15
```

После чего к каждому году в программе прибавьте два года и запишите новую информацию в файл pr2.txt в следующем виде:

```
51 мама
52 папа
20 сын
17 дочь
program pr1;
var r1,r2:text;
s,s1,s2:string;
d,d1,code:integer;
begin
assign(r1,'pr1.txt'); assign(r2,'pr2.txt');
reset(r1); rewrite(r2);
while not eof(r1) do begin
while not eoln(r1) do begin
readln(r1,s);
s2:=copy(s,7,2); val(s2,d,code); writeln(s2,d);
delete(s,5,9); writeln(s); d1:=d+2;
writeln(r2,' ',d1,' ',s);
end;end;
close(r1); close(r2);
end.
```

3. Считайте из предварительно созданного в рабочем каталоге файла pr3.txt информацию, представленную в следующем виде:
12
14
34
56
234
-123
и к каждому считанному значению d прибавьте sin(π/3).

```
program pr2;
var r1:text;
d:integer; d1:real;
begin
assign(r1,'pr3.txt'); reset(r1);
while not eof(r1) do begin
readln(r1,d); d1:=d+sin(pi/3);
writeln(' ',d,' ',d1); end; close(r1); end.
```

3 РАЗРАБОТКА И ОТЛАДКА ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ ТИПИЗИРОВАННЫХ ФАЙЛОВ

Типизированные файлы

Файл, определенный стандартным или пользовательским типом данных, называется типизированным. Общая форма объявления типизированных файлов имеет вид:

Var <имя файла>: File of <тип компонент>;

Здесь тип компонент может быть любым типом данных, определенных в Pascal, но только не файловым. Для работы с типизированными файлами используются уже знакомые нам процедуры и функции: Write, Read, Seek, Filesize, Filepos, а также процедура Truncate:

Truncate(<имя файловой переменной>)

Она удаляет все компоненты в файле, начиная с того над которым находиться указатель.

Одной из главных особенностей типизированного файла является возможность прямого обращения к его отдельным компонентам. Это достигается за счет, того что заранее известен тип компонент файла. Рассмотрим два примера кода, в первом из которых обращение к элементам файла происходит последовательно, а во втором прямо.

Пример 1.

Вычислить среднее арифметическое элементов файла.

Решение:

```
program type_fail;
uses crt;
var
f: file of integer; {объявление файловой переменной}
x, n, i: integer;
sr: real;
begin
assign (f, 'file.dat'); {создание файла}
rewrite(f); {открытие файла в режиме записи}
write('Количество элементов '); read(n);
for i:=1 to n do
begin
readln(x); {ввод элемента}
write(f, x); {запись элемента в файл}
end;
clrscr; n:=0;
reset(f); {открытие файла в режиме чтения}
while not eof(f) do {выполнять пока не достигнут конец файла}
begin
read(f, x);
n:=n+x;
end;
sr:=n/filesize(f);
close(f); {закрытие файла}
write('Среднее арифметическое = ', sr);
end.
```

Рисунок 3.1 – Рабочее окно

Пример 2.

Поменять строки в файле местами.

```
program type_fail;
uses crt;
var
f: file of string;
i: integer;
s, s1, s2, s3: string;
begin
assign (f, 'file.dat');
s1:='Pascal';
s2:=' на ';
s3:='Программирование';
rewrite(f);
write(f, s1, s2, s3); {запись строк в файл}
reset(f);
write('Выводим как есть: ');
while not eof(f) do {вывод содержимого файла}
begin
read(f, s);
write(s);
end;
writeln;
write('Вывод после обработки: ');
for i:=2 downto 0 do
begin
seek(f, i); {смена позиции указателя}
read(f, s);
write(s);
end;
close(f);
end.
```

Рисунок 3.2 – Рабочее окно программы

Таким образом, напрашивается вывод, что типизированные файлы несколько функциональней в обработке, чем текстовые. Далее разберем последний пункт данной статьи, а именно третий вид файлов — бестиповые файлы.

Бестиповые (нетипизированные) файлы

Бестиповые файлы предназначены для записи участков памяти компьютера на внешний носитель и считывания их в память. В отличие от типизированных файлов, нам не нужно знать информация какого типа хранится в них. А все потому, что данные из файлов, не имеющих типа, считываются в виде байт, после чего они «подстраиваются» под переменную, в которую происходит считывание.

Общая форма записи нетипизированных файлов

Var <идентификатор>: File;

отличается от типизированных отсутствием части of <тип данных>. Кроме того, немного изменяется принцип действия процедур Reset и Rewrite. К ним прибавляется второй параметр типа Word:

reset(<имя файловой переменной>, <значение>)

rewrite(<имя файловой переменной>, <значение>)

Здесь «*значение*» — это новый размер буфера, который по умолчанию равен 128 байтам. В качестве минимального значения можно указать 1 байт, а максимального — 64 кбайт (число в байтах).

Также в бестиповых файлах для записи и чтения информации используются не стандартные процедуры Read и Write, а две новые: **BlokRead** и **BlockWrite**. Рассмотрим каждую из них.

Процедура BlockRead

Данная процедура считывает из файла заданное число записей, которые помещаются в память.

Общая форма записи:

BlockRead(<имя файловой переменной>, <x>, <количество байт>, <y>);

x, y – обычные переменные, в первую помещается прочитанные данные, во вторую – количество считанных байт. В случае удачи y (y – необязательный параметр) будет иметь тоже значение, что и третий параметр.

Процедура BlockWrite

Для записи информации в бестиповый файл предназначена процедура BlockWrite:

BlockWrite(<имя файловой переменной>, <x>, <количество байт>, <y>);

Параметры процедуры BlockWrite точно такие же, как и у BlockRead. Да и принцип их одинаков, за исключением того, что первая записывает данные в файл, а вторая считывает их из него.

В следующей программе данные сначала записываются в нетипизированный файл, а затем выводятся из него посредством рассмотренных нами операций.

```
program no_type_fail;
uses crt;
var
f: file;
x, i ,n: byte;
begin
clrscr;
assign(f, 'f');
rewrite(f, 1);
write('n = '); readln(n);
for i:=1 to n do
begin
x:=n-i;
blockwrite(f, x, 1); {запись в файл}
end;
reset(f, 1);
while not eof(f) do
begin
blockread(f, x, 1); {чтение из файла}
write(x,' ');
end;
close(f);
readkey;
end.
```

Рисунок 3.3 – Данные в программе

Возможно, что ваша среда программирования не поддерживает работу с файлами, не имеющими типа. Поэтому прежде чем начать искать ошибку в коде, стоит узнать про данную функцию.

4 ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ TURBO DELPHI. АНИМАЦИИ ИЗОБРАЖЕНИЙ

Графические изображения наносятся на поверхность визуальных компонентов Turbo Delphi. Как правило, для этого используется компонент TImage. Поверхности соответствует свойство Canvas, к которому применяются методы вывода графических примитивов (точка, линия, прямоугольник, эллипс и др.). Для вывода изображений используются два объекта Pen и Brush. Карандаш Pen служит для рисования (проведения линий и контуров), а кисть Brush — для закраски.

Из всех свойств этих объектов выделим следующие:

- `Pen.Color` — цвет (по умолчанию — черный, т. е. `clBlack`);
- `Pen.Width` — толщина (по умолчанию 1 пиксел);
- `Brush.Color` — цвет кисти (по умолчанию такой же, какой и цвет фона).

Процедуры (методы) рисования примитивов:

Точка:

```
Pixels[x,y]:=C;
```

Здесь `C` — цвет точки (например, `clRed` — красный); `x`, `y` — координаты точки в пикселах на поверхности *Image* (начало координат находится в левом верхнем углу *Image*, ось *x* направлена направо, а *y* — вниз).

Линия:

`LineTo(x,y)` — процедура, которая проводит линию заданного цвета и толщины в точку с координатами `x`, `y` из текущей точки.

`MoveTo(x,y)` — процедура, переводящая карандаш (без рисования) в точку с координатами `x` и `y`.

Прямоугольник:

```
Rectangle(x1,y1,x2,y2);
```

Здесь `x1` и `y1` — координаты левой верхней точки; `x2` и `y2` — координаты правой нижней точки прямоугольника.

Эллипс:

```
Ellipse(x1,y1,x2,y2);
```

Здесь `x1` и `y1` — координаты левой верхней точки; `x2` и `y2` — координаты правой нижней точки невидимого прямоугольника, в который вписывается изображаемый эллипс.

Дуга эллипса:

```
Arc(x1,y1,x2,y2,x3,y3,x4,y4);
```

Здесь `x1` и `y1` — координаты левой верхней точки; `x2` и `y2` — координаты правой нижней точки невидимого прямоугольника, в который вписывается невидимый эллипс; `x3` и `y3` — координаты начала изображаемой дуги эллипса; `x4` и `y4` — координаты конца этой дуги.

Процедура заливки краской выбранного цвета:

```
FloodFill(x,y,clBlack,fsBorder),
```

где x и y — координаты любой точки внутри закрашиваемой области, ограниченной линией черного цвета.

Процедура вывода текста:

```
TextOut(x,y,'Надпись'),
```

Здесь x, y — координаты левого верхнего угла прямоугольной области вывода текста Надпись.

Тексту соответствует объект Font. Его некоторые свойства:

Font.Size — размер (например, 30);

Font.Name — тип (например, 'TimesNewRoman');

Font.Color — цвет (например, clGreen).

Более подробно о графических возможностях Turbo Delphi можно узнать в специальном пособии.

Пример. Разработать простое графическое приложение, в котором при нажатии на кнопку «Рисунок» появляется изображение японского флага с подписью.

Перед созданием приложения желательно нарисовать эскиз с указанием основных (узловых) точек рисунка (рис. 17). В данном случае размер Image1 составляет 300 на 400 пикселей, в его центре расположен флаг размером 200 на 100 пикселей, а диаметр красного круга — 50 пикселей.

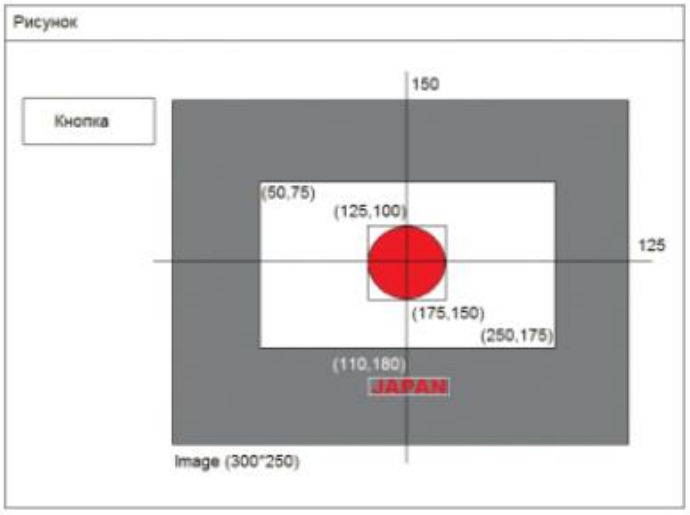

Рисунок 4.1 – Эскиз рисунка

Обработчик события для кнопки «Рисунок» (Button1) будет выглядеть следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
Begin
  with Image1.Canvas do
```

```
            Rectangle(50,75,250,175);
    Ellipse(125,100,175,150);
    Brush.Color:=clBlack;
    FloodFill(5,5,clBlack,fsBorder);
    Brush.Color:=clRed;
    FloodFill(150,120, clBlack, fsBorder);
    Font.Name:='Arial';
    Font.Size:=20;
    Font.Color:=clRed;
    TextOut(110,180,'Japan');
  end;
end;
```

Для моделирования движения какого-либо объекта на поверхности рисунка необходимо в цикле выполнять следующие действия:

- удаление нарисованного объекта (рисование его цветом фона);
- расчет нового положения объекта (новых координат узловых точек);
- рисование объекта в новом положении;
- задержка изображения на экране.

При этом движение любой сложности на плоскости можно моделировать, используя композицию трех преобразований:

- *смещение*. Определяющие параметры — величины смещения (dx, dy);
- *поворот*. Определяющие параметры — угол поворота на каждом шаге ($\Delta\varphi$) и координаты полюса поворота (x_p, y_p);
- *масштабирование*. Определяющие параметры — координаты полюса масштабирования (x_m, y_m) и коэффициенты масштабирования по оси x и оси y (k_x, k_y).

Для обеспечения повторений указанного выше алгоритма очень удобно использовать невизуальный компонент TTimer (из раздела System), который обеспечивает повторение необходимых действий через заданный промежуток времени. Повторяющиеся действия записываются в обработчике события OnTimer.

В качестве глобальных переменных опишем x (координата, отражающая изменение положения красного круга) и dx (смещение круга на каждом шаге по времени). Для задания их начальных значений используем событие OnFormCreate.

```
...
implementation
var
  x:integer;
  dx:shortint;
```

Событие OnFormCreate:

```
...
begin
   x:=125;
   dx:=2;
end;
```

Обработчик кнопки «Движение»:

```
...
begin
  Timer1.Enabled:=true;
end;
```

Событие OnTimer:

```
...
begin
  with Image1.Canvas do
    begin
      Brush.Color:=clWhite;
      Pen.Color:=clWhite;
      FloodFill(x+25,125,ClBlack, fsBorder);
      Ellipse(x, 100, x+50, 150);
      if (x+50>245) or (x<55)
        then
           dx:=-dx;
      x:=x+dx;
      Brush.Color:=clRed;
      Pen.Color:=clBlack;
      Ellipse(x, 100, x+50, 150);
      FloodFill(x+25,125,clBlack,fsBorder);
    end;
end;
```

Можно также добавить кнопку «Остановка», обработчик которой:

```
...
begin
  Timer1.Enabled:=true;
end;
```

На рис. 18 показан внешний вид формы и окна приложения в окончательном варианте.

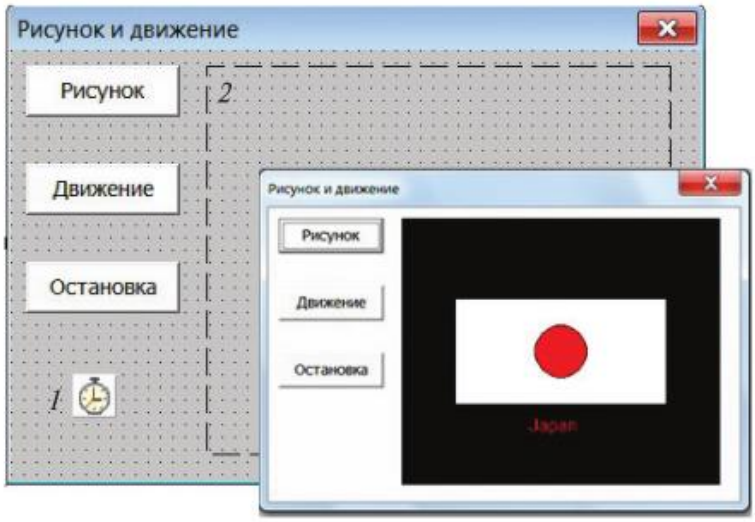

Рисунок 4.2 – Форма и окно приложения:
1 – Timer1; 2 – Image1

Задания для самостоятельной работы

1. Создать цветной рисунок, на котором изобразить дом с окнами, дерево, солнце на небе и сделать подпись. Моделировать горизонтальное движение солнца и изменение цвета неба и окон при выходе солнца за пределы рисунка.

2. Изобразить катящийся с горы мяч на синем фоне. Подпись под рисунком выполнить шрифтом желтого цвета.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Алексеев Ю. Е., Ваулин А. С., Куров А. В. Практикум по программированию. Обработка числовых данных: учеб. пособие для вузов / Алексеев Ю. Е., Ваулин А. С., Куров А. В.; ред. Трусов Б. Г. - Москва: Изд-во МГТУ им. Н. Э. Баумана, 2008. - 285 с.

2. Иванова Г. С., Ничушкина Т. П., Пугачев Е. К. Объектно-ориентированное программирование: учебник для вузов / Иванова Г. С., Ничушкина Т. Н., Пугачев Е. К.; ред. Иванова Г. С. - 3-е изд., стер. - Москва: Изд-во МГТУ им. II. Э. Баумана, 2007. - 366 с.

3. Исаев А. Л. Информатика. Конспект лекций: Исаев А. Л.; МГТУ им. И. Э. Баумана. - Москва: Изд-во МГТУ им. И. Э. Баумана, 2016. - 54 с.

4. Исаев А. Л. Информатика. Конспект практических занятий: учебно-методическое пособие / Исаев А. Л.; МГТУ им. Н. Э. Баумана (национальный исследовательский у-т). - Москва: Изд-во МГТУ им. И. Э. Баумана, 2019. -112с.

5. Парфилова Н. И., Пылькин А. Н., Трусов Б. Г. Программирование. Основы алгоритмизации и программирования: учебник для вузов / Парфилова И. И., Пылькин А. Н., Трусов Б. Г.; ред. Трусов Б. Г. - Москва: Академия, 2012. - 231 с.

6. Парфилова Н. И., Пылькин А. П. Трусов Б. Г. Программирование. Структурирование программ и данных: учеб, пособие для вузов / Парфилова Н. И. Пылькин А. П., Трусов Б. Г.; ред. Трусов Б. Г. - Москва: Академия, 2012. - 237 с.