# Министерство образования Республики Беларусь

УПРАВЛЕНИЕ ОБРАЗОВАНИЯ МОГИЛЕВСКОГО ОБЛАСТНОГО ИСПОЛНИТЕЛЬНОГО КОМИТЕТА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Могилевский государственный политехнический колледж»

# Домашняя контрольная работа №2

# По дисциплине: «Основы алгоритмизации и программирования»

Группа ПО-455

## Выполнил О. И. Чижик

Шифр 14

# 2020

**22 Охарактеризуйте назначение системных диалогов.**

В приложениях часто приходится выполнять стандартные действия: открывать и сохранять файлы, задавать атрибуты шрифтов, выбирать цвета палитры, производить контекстный поиск и замену и т. п.

Разработчики Delphi позаботились о том, чтобы включить в библиотеку простые для использования компоненты, реализующие соответствующие диалоговые окна. Они размещены на странице Dialogs. В таблице 1 приведен перечень этих диалогов.

**Таблица 1. Системные диалоги и их фрагменты**

| **Пикто- грамма** | **Компонент** | **Страница** | **Описание** |
| --- | --- | --- | --- |
|  | **OpenDialog** «Открыть файл» | Dialogs | Предназначен для создания окна диалога «Открыть файл». |
|  | **SaveDialog** «Сохранить файл» | Dialogs | Предназначен для создания окна диалога «Сохранить файл как». |
|  | **OpenPictureDialog** «Открыть рисунок» | Dialogs | Предназначен для создания окна диалога «Открыть рисунок», открывающего графический файл. Начиная с Delphi 3. |
|  | **SavePictureDialog** «Сохранить рисунок» | Dialogs | Предназначен для создания окна диалога «Сохранить рисунок» — сохранение изображения в графическом файле. Начиная с Delphi 3. |
|  | **FontDialog** «Шрифты» | Dialogs | Предназначен для создания окна диалога «Шрифты» — выбор атрибутов шрифта. |
|  | **ColorDialog** «Цвет» | Dialogs | Предназначен для создания окна диалога «Цвет» — выбор цвета. |
|  | **PrintDialog** «Печать» | Dialogs | Предназначен для создания окна диалога «Печать». |
|  | **PrinterSetupDialog** «Установка принтера» | Dialogs | Предназначен для создания окна диалога «Установка принтера». |
|  | **FindDialog** «Найти» | Dialogs | Предназначен для создания окна диалога «Найти» — контекстный поиск в тексте. |
|  | **ReplaceDialog** «Заменить» | Dialogs | Предназначен для создания окна диалога «Заменить» — контекстная замена фрагментов текста. |
|  | **FileListBox** (список файлов) | Win3.1 | Отображает список всех файлов каталога. |
|  | **DirectoryListBox** (структура каталогов) | Win3.1 | Отображает структуру каталогов диска. |
|  | **DriveComboBox** (список дисков) | Win3.1 | Выпадающий список доступных дисков. |
|  | **FilterComboBox** (список фильтров) | Win3.1 | Выпадающий список фильтров для поиска файлов. |
|  | **DirectoryOutline** (дерево каталогов) | Samples | Пример компонента, используемого для отображения структуры каталогов выбранного диска. |

Последние четыре компонента в таблице 1 являются не законченными диалогами, а их фрагментами, позволяющими строить свои собственные диалоговые окна.

Все диалоги являются невизуальными компонентами, так что место их размещения на форме не имеет значения. При обращении к этим компонентам вызываются стандартные диалоги, вид которых зависит от версии Windows и настройки системы. Так что при запуске одного и того же приложения на компьютерах с разными системами диалоги будут выглядеть по-разному. Например, при русифицированной версии Windows все их надписи будут русскими, а при англоязычной версии надписи будут на английском языке.

Основной метод, которым производится обращение к любому диалогу, — **Execute**. Эта функция открывает диалоговое окно и, если пользователь произвел в нем какой-то выбор, то функция возвращает **true**. При этом в свойствах компонента-диалога запоминается выбор пользователя, который можно прочитать и использовать в дальнейших операциях. Если же пользователь в диалоге нажал кнопку Отмена или клавишу Esc, то функция **Execute** возвращает **false**. Поэтому стандартное обращение к диалогу имеет вид:

**if** <имя компонента-диалога>.Execute then

<операторы, использующие выбор пользователя>;

**60 Опишите технологию OLE.**

Аббревиатура OLE обозначает Objects Linked and Embedded (Присоединенные И Встроенные Объекты - ПИВО). Данные, разделяемые между приложениями называются *OLE объектом.*Приложение, которое может содержать OLE объекты, называют *OLE контейнером (OLE Container).*Приложение, данные из которого можно включить в OLE контейнер в виде OLE объекта, называют *OLE сервером.*

Например, MicroSoft Word может включать в документ графические объекты, аудио- и видеоклипы и множество других объектов (такой документ иногда называют *составным документом - compound document* ).

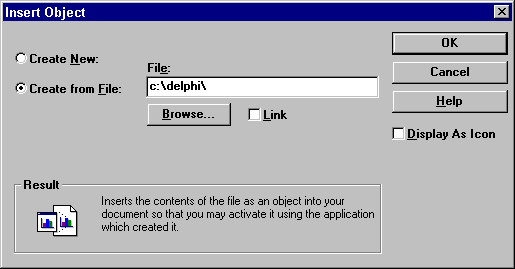
Как следует из названия, OLE объекты можно либо присоединить к OLE контейнеру, либо включить в него. В первом случае данные будут храниться в файле на диске, любое приложение будет иметь доступ к этим данным и сможет вносить изменения. Во втором случае данные включаются в OLE контейнер и только он сможет просматривать и модифицировать эти данные.

OLE является дальнейшим развитием идеи разделяемых между приложениями данных. Если с помощью DDE можно было работать с текстом, то OLE позволяет легко встроить в приложение обработку любых типов данных. Как и в случае с DDE, для правильной работы приложения-клиента (OLE контейнера) требуется наличие приложения OLE сервера. Каждый раз, когда в программе-клиенте пользователь обращается к OLE объекту с целью просмотра или редактирования данных (обычно двойной щелчок мышкой на объекте), запускается приложение-сервер, в котором и происходит работа с данными.

В природе существует несколько видов OLE, отличающихся по способу активации OLE сервера. OLE версии 1 запускает сервер в отдельном окне. OLE 2 реализует то, что называется *in-place activation and editing.*В данном случае сервер запускается “внутри” приложения-клиента, модифицирует вид системного меню, линейки инструментов и др. Развитие идеи OLE привело к появлению *OLE automation*- приложение-клиент может выполнить часть кода сервера. Тип OLE объекта, помещенного в программу-клиент, определяется тем, какую версию OLE поддерживает сервер.

**Объект TOLEContainer**

Объект TOLEContainer находится на странице System Палитры Компонент и нужен для создания приложений OLE-контейнеров. TOLEContainer скрывает все сложности, связанные с внутренней организацией OLE и предоставляет программисту достаточно простой интерфейс. Построим простейшее приложение с использованием OLE объекта. Создайте новый проект и поместите на форму TOLEContainer, в Инспекторе Объектов дважды щелкните мышкой на свойство ObjClass или ObjDoc - появится стандартный диалог Windows “Insert Object” (см. рис.1)

Рисунок 1, Диалог вставки объекта

В этом диалоге есть список всех зарегистрированных в системе OLE-серверов (регистрация происходит при инсталляции программы). Тип OLE-объекта определяется как раз тем сервером, который Вы укажете. Если Вы создаете новый объект (Create New), то при нажатии кнопки OK запустится программа OLE-сервер, в которой и формируется новый объект. После выхода из программы-сервера новый OLE объект *включается* (embedded object) в программу. OLE объект можно создать используя уже имеющийся файл в формате одного из OLE-серверов. Для этого нужно выбрать пункт Create from File (см. рис.2)

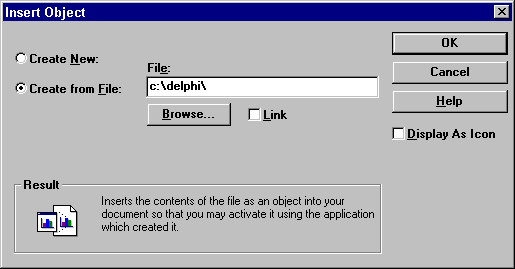


Рисунок 2, Создание объекта из существующего файла.

Выбранный объект можно как включить в приложение, так и присоединить, отметив пункт Link.

**70 Дайте определение указателя. Опишите базовые операции с указателями в языке Паскаль.**

Как следует из названия, **переменные-указатели** это особый тип переменных, которые не содержат значения, а указывают на них - на ту ячейку памяти, где они фактически располагаются. И хотя справедливо считается, что использование указателей может приводить к трудно контролируемым ошибкам в программе, всё же переменные-указатели — это очень эффективный инструмент для управления объектами в оперативной памяти компьютера.

   Конечно, ячейка памяти — это структура размером в один байт. Объекты же, с которыми работает программа, в основном намного большего размера. Соответственно, указатель содержит в себе адрес только первого байта той области оперативной памяти компьютера, где располагается данный объект. Зная тип и соответственно размер объекта, можно прочитать его целиком.

### Описание переменных - указателей

   Указатель описывается ключевым словом Pointer. По первой букве ключевого слова принято называть переменные - указатели с первой буквы P:

  var PIndexer: Pointer;*//нетипизированный указатель*

   Ключевым словом Pointer задаётся так называемый нетипизированный указатель, по аналогии с нетипизированным файлом. Нетипизированный указатель содержит просто адреc некой ячейки памяти. Объект, располагающийся начиная с этой ячейки, соответственно может быть совершенно любого типа и размера.

   Также в Delphi существуют и типизированные указатели. Они могут указывать на объект соответствующего типа и размера. Именно "могут указывать", потому что это по прежнему адрес одной - первой ячейки области памяти, где располагается объект. И далее его использование в программе зависит от программиста.   Итак, типизированный указатель описывается ключевым словом означающим данный тип, перед которым ставится значок ^:

  var PInteger: ^Integer;*//указатель на переменную целого типа*

         PText: ^String;*//указатель на переменную типа String*

   Также можно описать любой свой тип, и задать переменную-указатель данного типа:

type TMyType = Record

   X: Integer;

   S: String;  
end;

var PMyPointer: ^TMyType;

   Естественно, можно определить тип, для описания через него переменных - указателей. Делается это в том числе и потому, что, например, в процедурах и функциях в качестве параметров можно использовать только заранее описанный тип данных. Например, следующее описание задаёт функцию с параметром, являющимся указателем ранее описанного типа, результат которой также является указателем данного типа:

type TPMyPointer = ^TMyType;

function MyFunc(Point: TMyPointer): TMyPointer;

Описанный указатель без присвоенного значения указывает на совершенно неопределённую ячейку памяти. Попытка использовать такой указатель чревата крахом программы. Поэтому всем указателям нужно явно присваивать значения.

1. Указателю можно присвоить значение другого указателя. В результате оба указателя будут указывать на одну и ту же ячейку памяти. Также указателю можно присвоить пустое значение с помощью ключевого слова nil:

var P1, P2: Pointer;

begin  
   P1:=P2;*//Присвоение указателю значения другого указателя*

  P2:=nil;*//Присвоение указателю "пустого" значения*

end;

   Указатель со значением nil не адресует никакой ячейки памяти и единственное, что с ним можно сделать - это сравнить с другим указателем или со значением nil.

2. Значение типизированного указателя можно увеличить или уменьшить на размер области памяти, занимаемой объектом данного типа. Для этого служат операции инкремента и декремента:

type P: ^Integer;

begin

  inc(P);*//увеличение значения указателя на 4 байта (размер типа Integer)*

  dec(P);*//уменьшение значения указателя на 4 байта (размер типа Integer)*

end;

   Попытка выполнить операциии inc либо dec с нетипизированным указателем вызовет ошибку на этапе компиляции, так как компилятору неизвестно, насколько именно изменять значение указателя.

3. Процедурой New можно создать область памяти сответствующего типа и присвоить её адрес указателю (инициировать указатель):

var PInt: ^Integer;

begin

  New(PInt);*//Указатель PInt получает значение адреса созданной области памяти типа Integer*

end;

   Поскольку с областью памяти, созданной с помощью процедуры New, не связана ни одна переменная, но там содержится реальное используемое значение, то можно считать, что это значение связано с некой "безымянной переменной". Обращаться к ней по имени переменной невозможно, а можно оперировать только используя указатель.

   Также задать указателю адрес объекта можно с помощью операции, называемой "взятие адреса", которая обозначается значком @. При этом создавать область памяти уже не нужно, так как она создана предварительным описанием данного объекта:

var MyVar: TMyType;*//Описание переменной, при этом выделяется область памяти соответствующего размера*

    P: ^TMyType;*//Задаётся указатель соответствующего типа*

Begin

  P:=@MyVar;*//Указатель получает адрес области памяти, занимаемой переменной MyVar*

end;

4. Если область памяти уже создана и её адрес присвоен указателю, то в ячейку памяти, адресуемую данным указателем, можно записать значение объекта, соответствующего типу указателя. Для этого служит операция, обозначаемая также значком ^, стоящим после имени указателя, например: P^. Эта операция называется "разыменование указателя". Также с помощью этой операции со значением в данной ячейке памяти можно делать всё что нужно:

var MyVar: Integer;

    P: ^Integer;

Begin

  P:=@MyVar;*//Указатель получает адрес области памяти, занимаемой переменной MyVar*

  P^:=2;*//В ячейку памяти по адресу переменной MyVar записывается значение 2*  
  Form1.Caption:=IntToStr(P^+3);*//В заголовке Формы появится число 5*

end;

   С обычными переменными всё просто, но возникает вопрос, как получить значение по адресу указателя, если тип переменной - запись с несколькими полями? Аналогично:

type TMyRec = Record

    N: Integer;

    S: String;

end;

var MyRec: TMyRec;

    PRec: ^TMyRec;

Begin

 PRec:=@MyRec;*//Указатель получает адрес области памяти, занимаемой переменной MyRec*

  PRec^.S:='Строка данных';*//С помощью указателя производится изменение строкового поля записи*

  PRec^.N:=256;*//С помощью указателя производится изменение числового поля записи*

end;

   Далее, а как получить значение, если указатель это элемент массива, например:

var PArray: Array[1..100] of ^Integer;

    X: Integer;

   Ни PArray^[10], ни PArray[10]^ не являются правильными выражениями. Нужно использовать скобки:

X:=(PArray[10])^;

5. Память, выделенную процедурой New, всегда нужно явно освобождать. Освободить область памяти, адресуемую указателем, инициированным с помощью New, можно процедурой Dispose:

var MyVar: TMyType;

    P: ^TMyType;

Begin

  P:=@MyVar;

  Dispose(P);*//Освобождение области памяти, адресуемой указателем P*

end;

   При выполнении процедуры Dispose указатель снова приобретает неопределённое значение, не равное даже nil, и его использование может привести к неопределённым результатам, даже к краху программы.

**113 На канве компонента Image при нажатии кнопки «Рисовать» построите изображение кораблика. Использовать рисование по точкам. При помощи компонентов RadioButton задавать способ вывода изображения – мгновенно, замедленно. Для замедления использовать компонент Timer. Выполните задание, добавив, если необходимо, защиту от неправильного ввода данных. Форма обязательно должна содержать кнопку для закрытия приложения, кнопку очистки компонентов и главное меню, дублирующее основные действия.**

**РЕШЕНИЕ:**

Сначала напишем процедуру для осуществления задержки в том случае, когда рисовать кораблик нужно с задержкой. В качестве параметра будем передавать время задержки в миллисекундах. Для быстрого рисования будем передавать 0, для замедленного 1000, что составит задержку в 1 секунду.

procedure Delay(Value: Cardinal);

var

F, N: Cardinal;

begin

N := 0;

while N <= (Value div 10) do

begin

SleepEx(1, True);

Application.ProcessMessages;

Inc(N);

end;

F := GetTickCount;

repeat

Application.ProcessMessages;

N := GetTickCount;

until (N - F >= (Value mod 10)) or (N < F);

end;

Создадим форму приложения и разместим на ней компоненты TImage, TMainMenu, TButton, 2 компонента TRadioButton для выбора скорости прорисовки. Разместим на компоненте TImage стартовую картинку. Укажем заголовки элементов формы и самой формы. Настроим по умолчанию быструю прорисовку.

Внешний вид формы в процессе разработки приведен на рисунке 1.

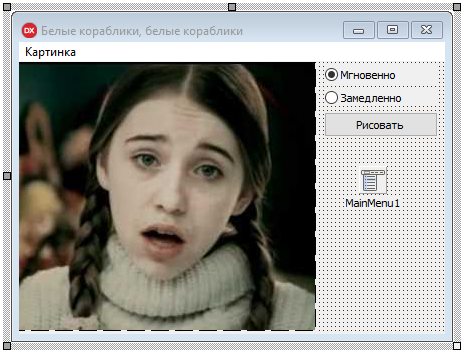


Рисунок 1. Внешний вид формы на этапе разработки.

Добавим пункты главного меню, отвечающие за прорисовку. Зададим для компонента Button1 обработчик который будет очищать рисунок и выводить изображение кораблика с указанной в переменной D скоростью.

procedure TForm1.Button1Click(Sender: TObject);

begin

Image1.Canvas.FillRect(Rect);{Очищаем компонент}

with Image1.Canvas do

begin

MoveTo(50, 250);

LineTo(Image1.Width-50, 250);

delay(D);

LineTo(Image1.Width-5, 210);

delay(D);

LineTo(5, 210);

delay(D);

LineTo(50, 250);

delay(D);

MoveTo(150, 210);

LineTo(150, 20);

delay(D);

LineTo(170, 30);

delay(D);

LineTo(150, 40);

delay(D);

MoveTo(150,20);

LineTo(30, 190);

delay(D);

LineTo(Image1.Width-35,190);

delay(D);

LineTo(150, 40);

end;

end;

Для компонентов RadioButton установим обработчики, которые задают переменную D управляющую скоростью прорисовки.

procedure TForm1.RadioButton1Click(Sender: TObject);

begin

D:=0;

end;

procedure TForm1.RadioButton2Click(Sender: TObject);

begin

D:=1000;

end;

Создадим обработчик вызывающийся при создании главной формы, он проинициализирует значения переменной D по умолчанию и создаст объект TRect помогающий при очистке Image.

procedure TForm1.FormCreate(Sender: TObject);

begin

Rect.Left:=0;

Rect.Top:=0;

Rect.Right:=Image1.Width;

Rect.Bottom:=Image1.Height;

D:=0;

end;

Создадим обработчики главного меню приложения.

procedure TForm1.N2Click(Sender: TObject);

begin

Form1.RadioButton1.Checked:=true;

D:=0;

Button1.Click;

end;

procedure TForm1.N3Click(Sender: TObject);

begin

Form1.RadioButton2.Checked:=true;

D:=1000;

Button1.Click;

end;

procedure TForm1.N5Click(Sender: TObject);

begin

Application.Terminate;

end;

Вид главной формы приложения после запуска представлен на рисунке 2.

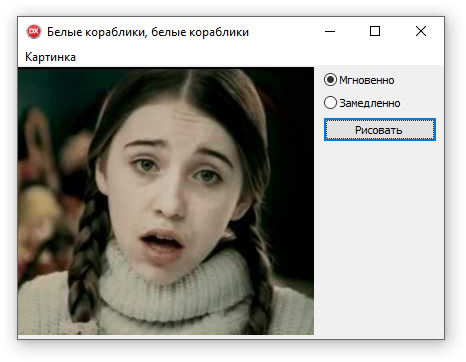


Рисунок 2. Главная форма приложения.

Вид главной формы приложения с нарисованным изображением кораблика представлен на рисунке 3.

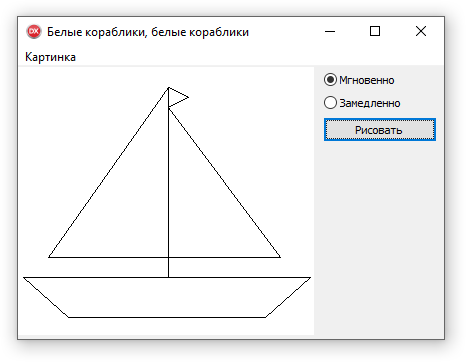


Рисунок 3. Главная форма приложения с нарисованным корабликом.

**131 Введите строку и букву. Вывести, сколько раз буква встречается в строке. Добавить меню, дублирующее кнопки. Выполните задание, добавив, если необходимо, защиту от неправильного ввода данных. Форма обязательно должна содержать кнопку для закрытия приложения.**

**РЕШЕНИЕ:**

Создадим форму приложения и разместим на ней компоненты TEdit, TMainMenu, TButton, TLabel для задания строки, буквы и вывода результата вычислений, а также собственно запуска вычислений. Укажем заголовки элементов формы и самой формы.

Внешний вид формы в процессе разработки приведен на рисунке 4.

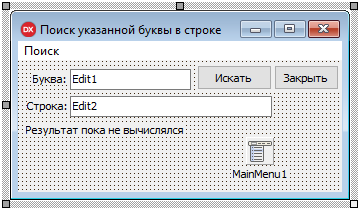


Рисунок 4. Форма на этапе разработки.

Добавим обработчики главного меню приложения для поиска и завершения работы программы, а так же для кнопки закрытия приложения.

procedure TForm1.Button2Click(Sender: TObject);

begin

Application.Terminate;

end;

procedure TForm1.N2Click(Sender: TObject);

begin

Button1.Click;

end;

procedure TForm1.N4Click(Sender: TObject);

begin

Application.Terminate;

end;

Реализуем алгоритм поиска в обработчике нажатия кнопки «Искать»

procedure TForm1.Button1Click(Sender: TObject);

var

i:integer;

c:char;

begin

N:=0;{обнулим переменную поиска}

{проверим входные данные на наличие символов

}

Label3.Caption:='Результат пока не вычислялся';

if Edit1.Text='' then

begin {Не указано буквы для поиска}

ShowMessage('Не указана буква для поиска');

exit;

end;

if length(Edit1.Text)>1 then

begin {Указано более одной буквы}

ShowMessage('Указана более чем одна буква для поиска');

exit;

end;

if Edit2.Text='' then

begin {Указана пустая строка для поиска}

ShowMessage('Не указана строка для поиска');

exit;

end;

S:=Edit2.Text;{Реализуем сам поиск}

for i := 1 to length(s) do

if edit1.text=s[i] then

inc(n);

{Выводим результат}

Label3.Caption:='Результат поиска, найдено позиций:'+inttostr(N);

end;

Готовая форма рабочего приложения приведена на рисунке 5.

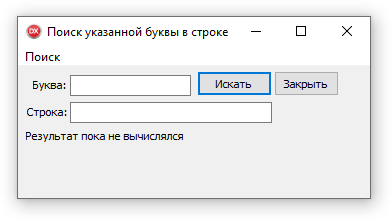


Рисунок 5. Рабочая форма приложения.

Результат работы приложения приведен на рисунке 6.

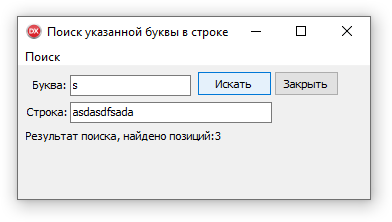


Рисунок 6. Результат поиска.

**ТЕСТ:**

Результат обработки ошибок приведён на рисунках 7-9.

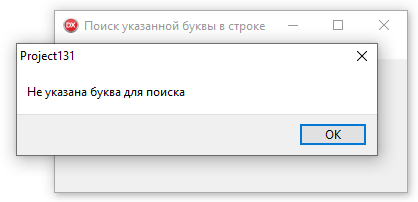


Рисунок 7. Обработка пустой буквы для поиска

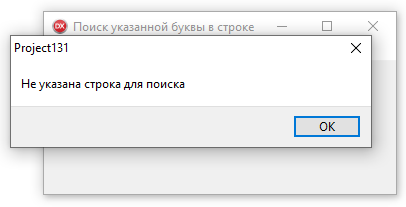


Рисунок 8. Обработка пустой строки для поиска.

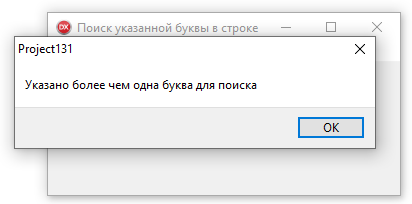


Рисунок 9. Обработка двух и более символов в качестве буквы.

**156 Посчитайте количество кратных ребер в графе, заданном матрицей смежности вершин. Ввод должен осуществляться либо с клавиатуры, либо из файла (организовать выбор пользователя).**

**РЕШЕНИЕ:**

Создадим форму приложения и разместим на ней компоненты TEdit, TMainMenu, TButton, TLabel для задания строки, буквы и вывода результата вычислений, а так же собственно запуска вычислений. Укажем заголовки элементов формы и самой формы.

Внешний вид формы в процессе разработки приведен на рисунке 10.

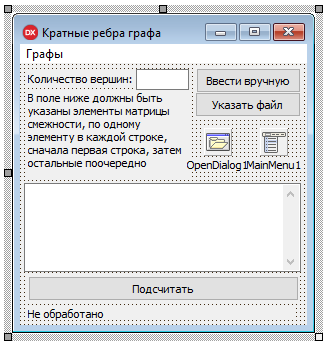


Рисунок 10. Внешний вид формы в процессе разработки.

Зададим обработчики для пунктов основного меню приложения.

procedure TForm1.N2Click(Sender: TObject);

begin

Button1.Click;

end;

procedure TForm1.N3Click(Sender: TObject);

begin

Button2.Click;

end;

procedure TForm1.N4Click(Sender: TObject);

begin

Button3.Click;

end;

procedure TForm1.N6Click(Sender: TObject);

begin

Application.Terminate;

end;

Зададим возможность загрузки матрицы смежности из файла в компонент Memo1 при помощи диалога выбора файла. Перед загрузкой файла необходимо указать количество вершин в графе.

procedure TForm1.Button2Click(Sender: TObject);

begin

if Edit1.Text<>'' then

begin

if Opendialog1.Execute then

Memo1.Lines.LoadFromFile(Opendialog1.filename)

end

else ShowMessage('Сначала необходимо указать количество вершин');

end;

После указания входных данных в виде матрицы смежности можно приступить к её анализу. Анализ условия приводит к тому, что работа производится с ориентированным графом. В матрице смежности запись о ребре графа между вершинами I и J будет отражена элементами матрицы A[i,j] и A[j,i], если элемент равен нулю то ребро отсутствует, в противном случае элемент указывает на вес ребра соответствующего направления. Таким образом произведение элементов матрицы A[i,j] и A[j,i] будет равно 0 если ребро не кратное, и отлично от нуля для кратных рёбер. При просмотре матрицы каждое ребро будет просмотрено 2 раза, следовательно результирующее значение счетчика кратности нужно будет разделить на 2.

procedure TForm1.Button3Click(Sender: TObject);

var

i,j:integer;

N,cnt,cod:integer;

A:array[1..10,1..10] of integer;

s:string;

begin

{}

N:=strtoint(Edit1.text);

cnt:=0;

{проверяем хватает ли нам элементов для матрицы}

if memo1.Lines.Count<N\*N then

begin

ShowMessage(' Указано меньше чем нужно вершин для матрицы');

Exit;

end

else

begin

{заполняем матрицу смежности}

for i:=1 to N do

for j := 1 to N do

begin

A[i,j]:=strtoint(memo1.Lines[cnt]);

inc(cnt);

end;

{считаем кратные ребра по всем вершинам}

cnt:=0;

Memo1.Lines.Add(' ');

for i:=1 to N do

begin

s:='';

for j := 1 to N do

begin

if i<>j then

begin

if A[i,j]\*A[j,i]<>0 then inc(cnt);

end;

s:=s+Inttostr(A[i,j])+' ';

end;

Memo1.Lines.Add(s);

end;

{Выводим результат подсчета}

cnt:=cnt div 2;

Label3.Caption:='Кратных ребер: '+inttostr(cnt);

end;

end;

**ТЕСТ:**

Ситуация с недостаточным количеством элементов матрицы смежности представлена на рисунке 11.

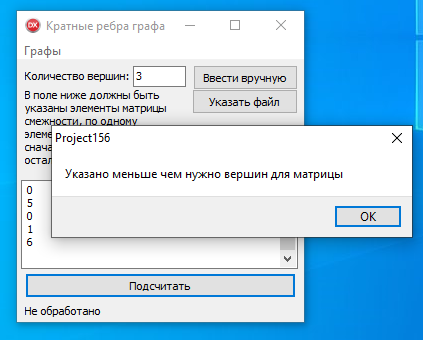


Рисунок 11. Недостаточное количество вершин матрицы.

Ситуация в которой не указано количество вершин графа представлена на рисунке 12.

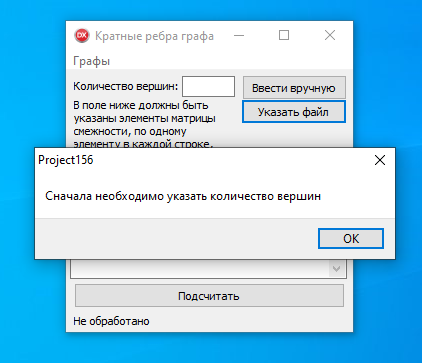


Рисунок 12. Не указано количество вершин графа.

Результат работы программы при верных исходных данных представлен на рисунке 13.

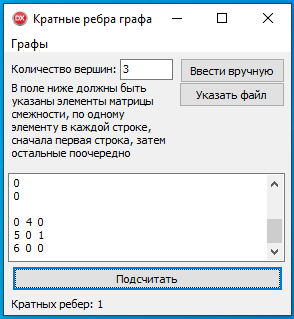


Рисунок 13. Результат подсчета кратных ребер.

В папке с программой размещены 2 тестовых файла в которых сохранена информация для загрузки данных по смежным матрицам с именами 4v2s.txt и 3v1s.txt для проверки матриц с 4 вершинами и 2 смежными ребрами, и 3 вершинами и 1 смежными ребрами соответственно.

**Список использованных источников**

1. ГОСТ ИСО/ МЭК 2382-99. Информационные технологии. Словарь. Ч. 1. Основные термины.
2. ГОСТ19.701-90 (ИСО 5807-85). Описание символов. Правила применения символов и выполнения схем.
3. СТУ СМК 01-32-2019. Стандарт учреждения. Общие требования к оформлению текстовых документов».
4. Макаров, В.Л. Программирование и основы алгоритмизации / В.Л. Макаров. – Санкт-Петербург: 2003.
5. Котов, В.М. Информатика. Методы алгоритмизации / В.М. Котов. – Нар. асвета: 2000.
6. Зубок, Д.А. Основы программирования в среде TURBO PASCAL / Д.А Зубок, С.В. Краснов, А.В. Маятин. – Санкт-Петербург: 2009.
7. Беляева, И.В. Основы программирования на языке TURBO PASCAL / И.В. Беляева. – Ульяновск: 2011.
8. Окулов, С.М. Основы программирования / С.М. Окулов. – М.: БИНОМ. Лаборатория знаний, 2008.
9. Ключарев, А. А. Учебное пособие «Структуры и алгоритмы обработки данных» / А. А. Ключарев, В. А. Матьяш, С. В. Щекин. – Санкт-Петербург: 2004.
10. Internet: https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0\_%D1%81%D0%BC%D0%B5%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D0%B8#:~:text=%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0%20%D1%81%D0%BC%D0%B5%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D0%B8%20%D0%B3%D1%80%D0%B0%D1%84%D0%B0%20G%20%D1%81,%D0%B3%D1%80%D0%B0%D1%84%D0%B0%20%D0%B2%20j%2D%D1%8E%20%D0%B2%D0%B5%D1%80%D1%88%D0%B8%D0%BD%D1%83.