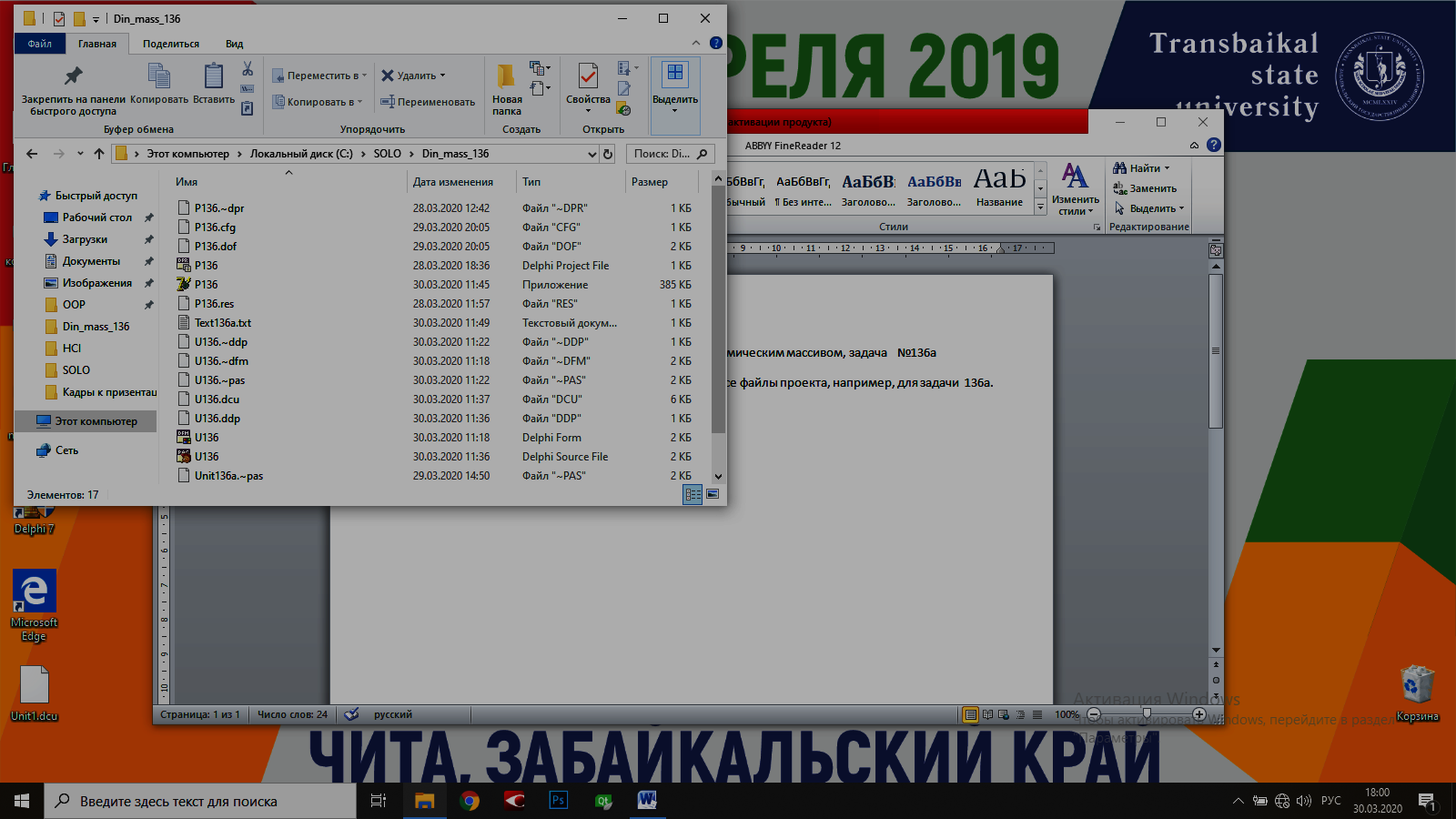
# Алгоритм создания приложения для работы с динамическим массивом, задача №136а

1. **Создать папку**, в которой будут храниться все файлы проекта, например, для задачи 136а.

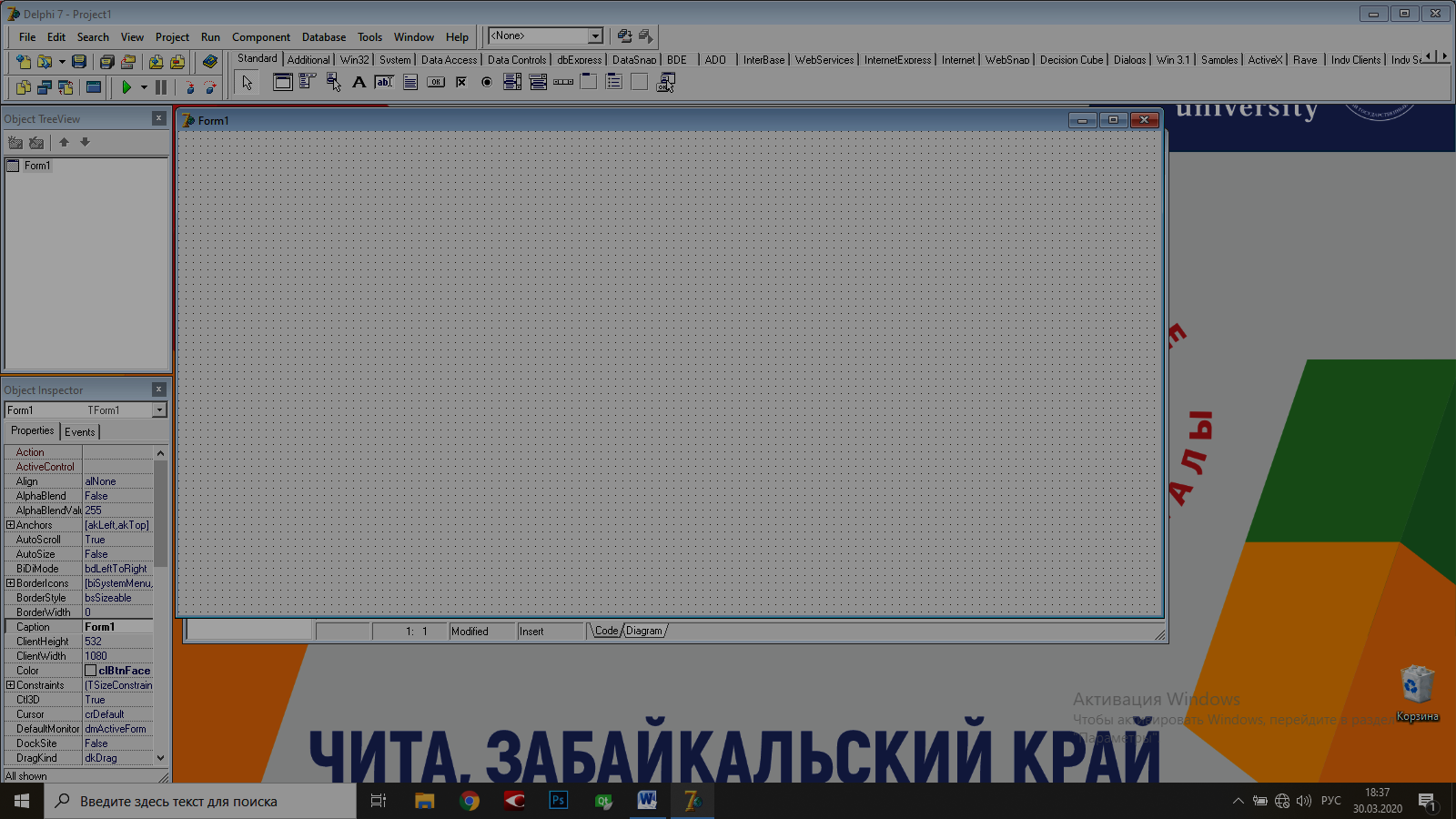
В данном примере адрес, созданной папки: **c:\SOLO\Din\_mass\_136**



1. **Создать проект** для своего варианта; пошаговая реализация:

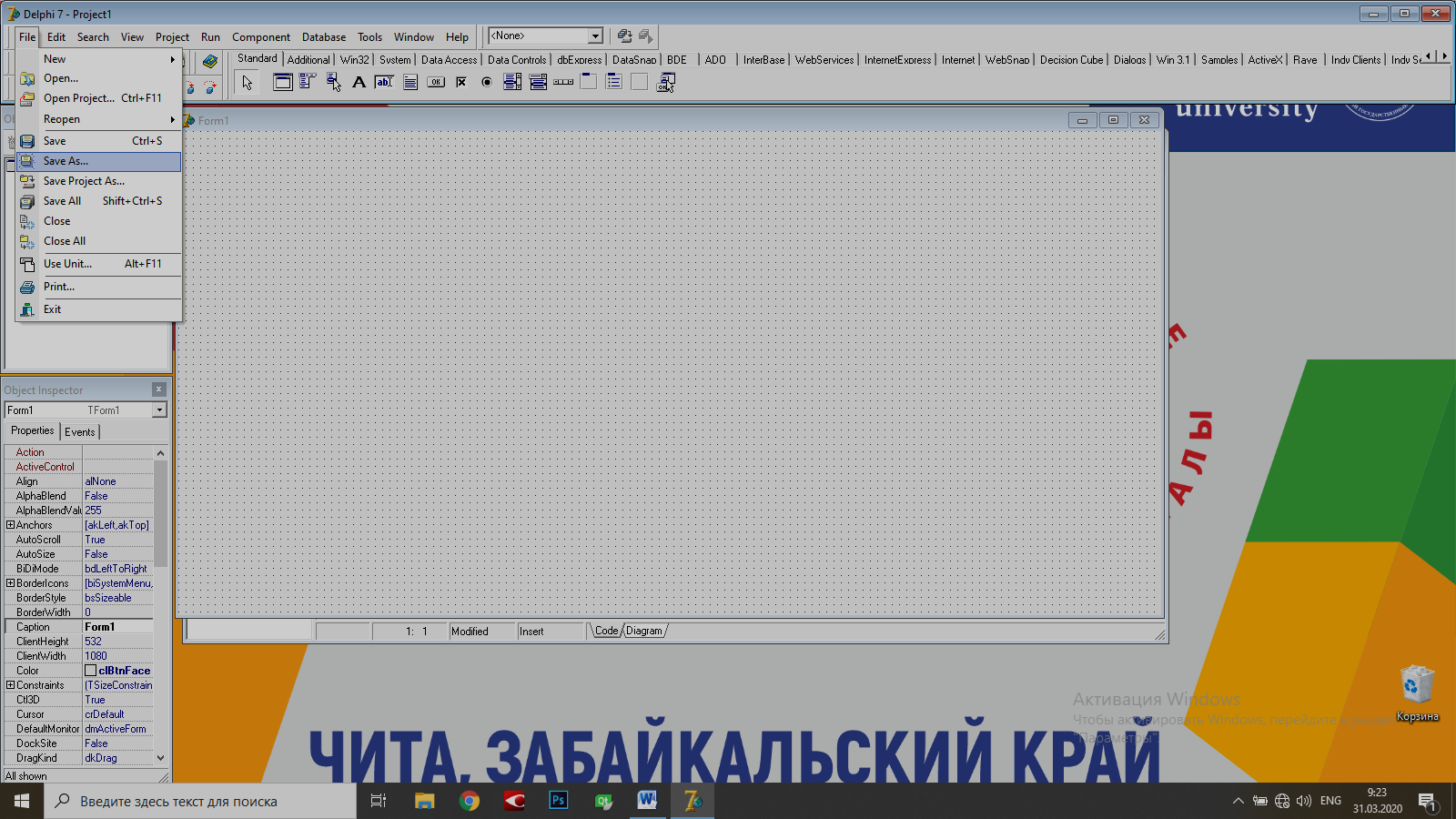
 Двойной щелчок левой кнопкой мыши (ЛКМ) на ярлыке.

В результате загружается основное окно Интегрированной Среды Разработки (IDE) в Delphi 7:



1. **Сохраним проект** в созданном каталоге, например, в **c:\SOLO\DM \_136**

Для этого выполним команду File из главного меню и в появившемся окне выберем команду Save As

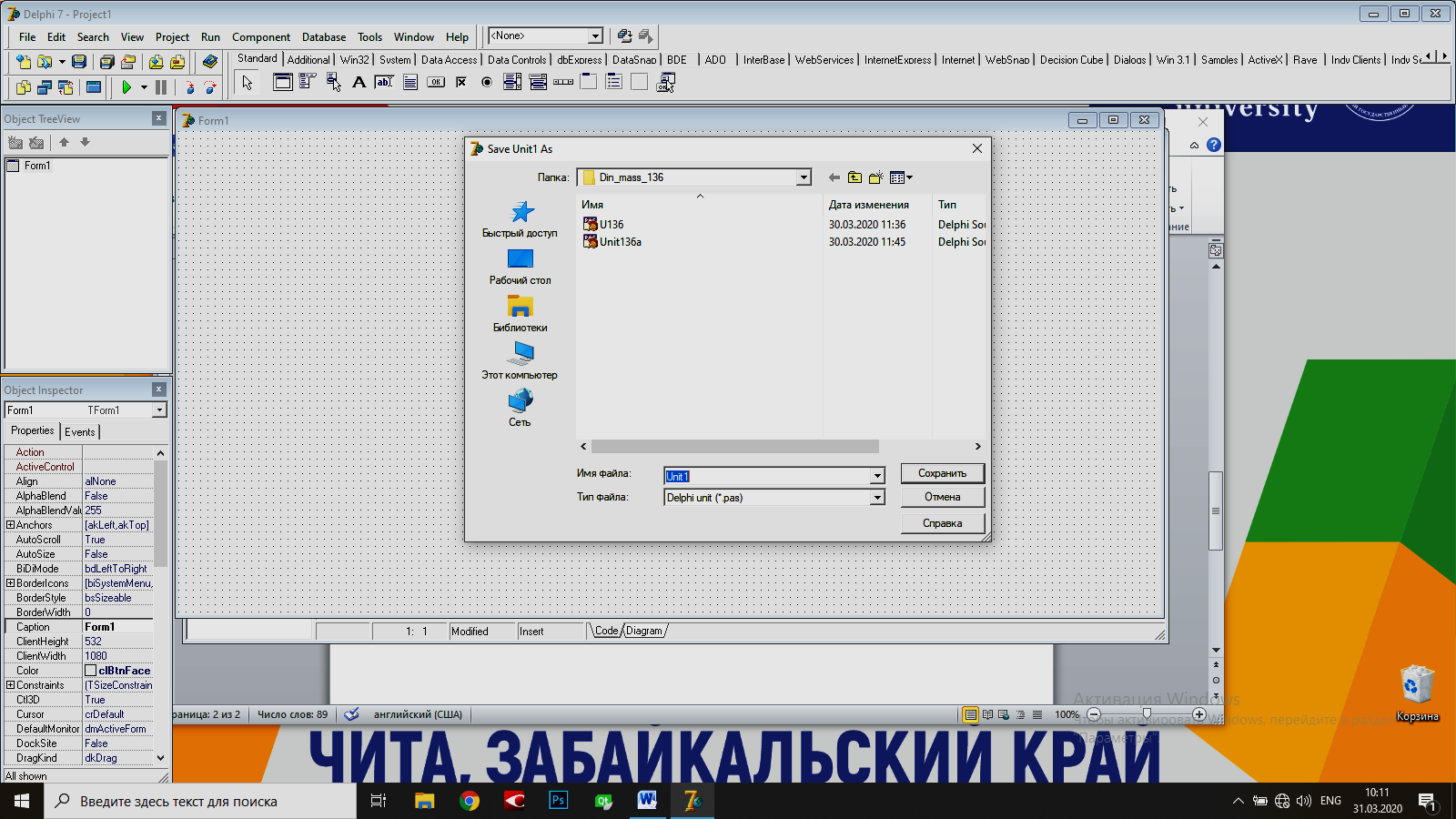


В окне **Save Unit1 AS** выберем:

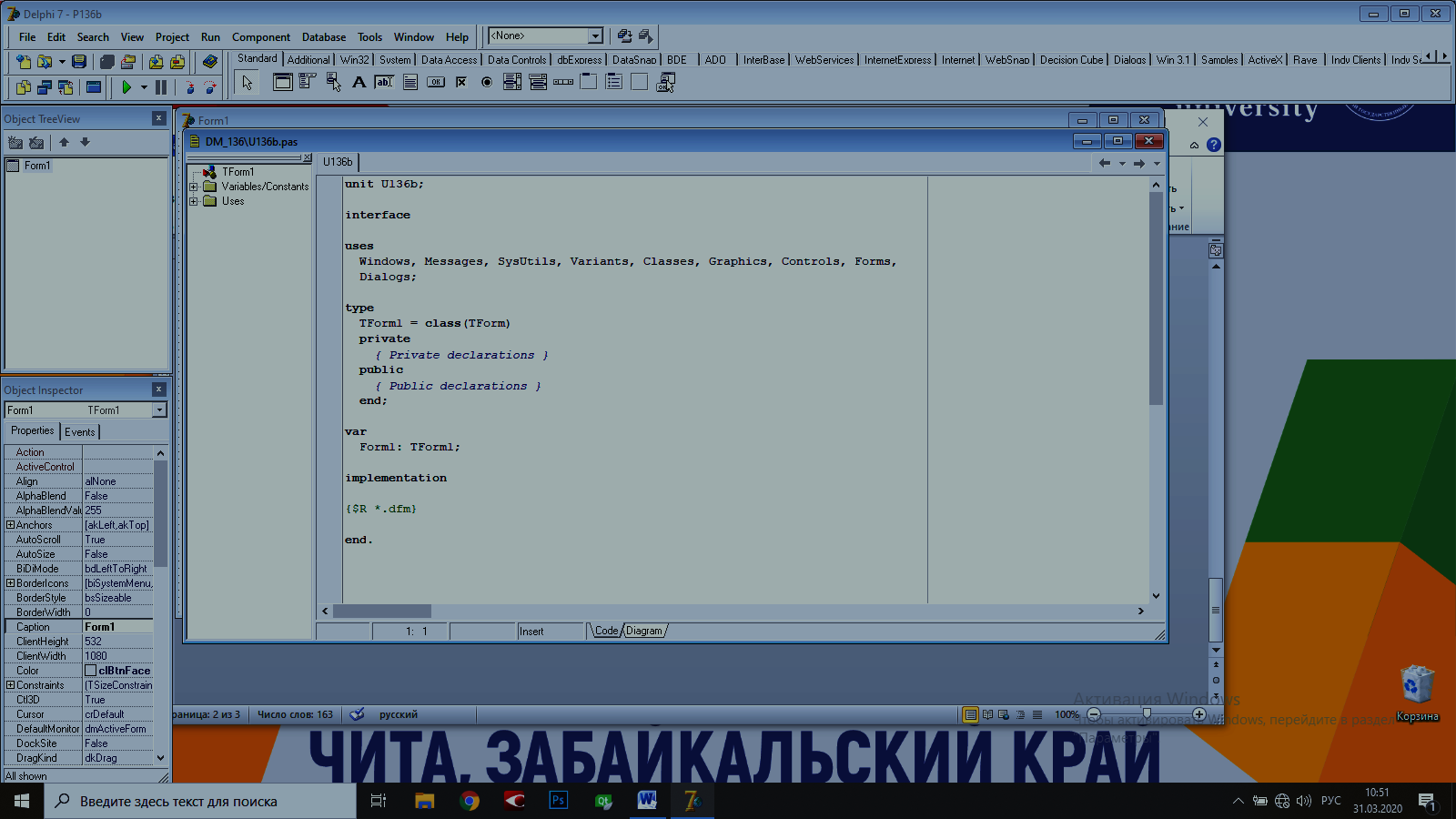
вначале папку, в которой будут храниться файлы создаваемого проекта,

затем имя модуля формы. Имена модулей лучше начинать заглавной буквой **U,** а далее информация, к какому проекту относится данный модуль, например, U136a.

Остаётся сохранить выбранное имя модуля в выбранной папке.



Аналогичную операцию нужно повторить для сохранения проекта. Имя проекта также желательно начинать заглавной буквой **P**, а далее краткая информация о задаче проекта, например**, P136b**. В результате этих операций заголовки проекта будут выглядеть так:

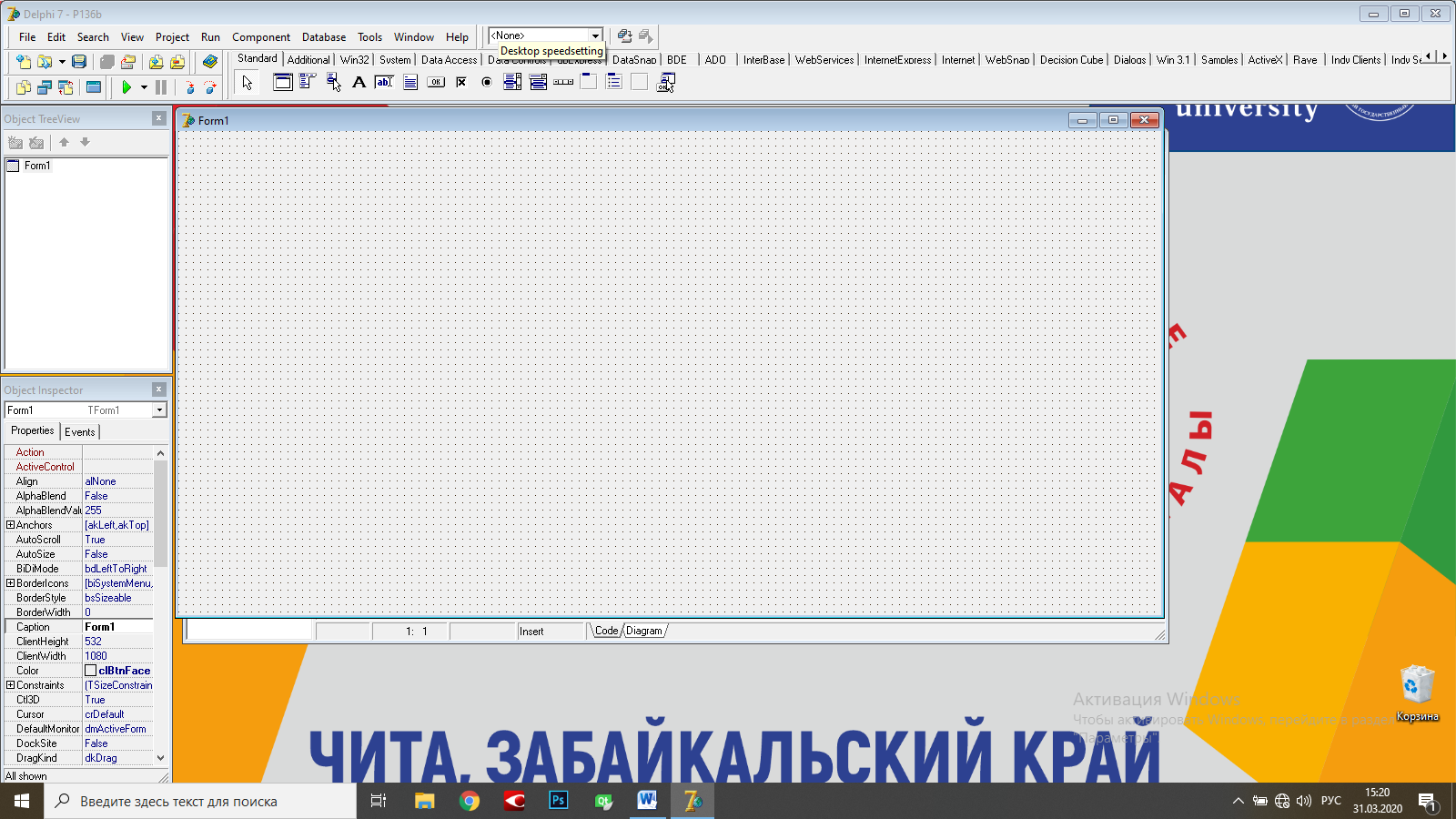


1. **Создать строку меню** с операциями, реализуемыми в проекте. В начале определим список этих операций, что позволит легко создавать элементы меню и выпадающие элементы подменю:

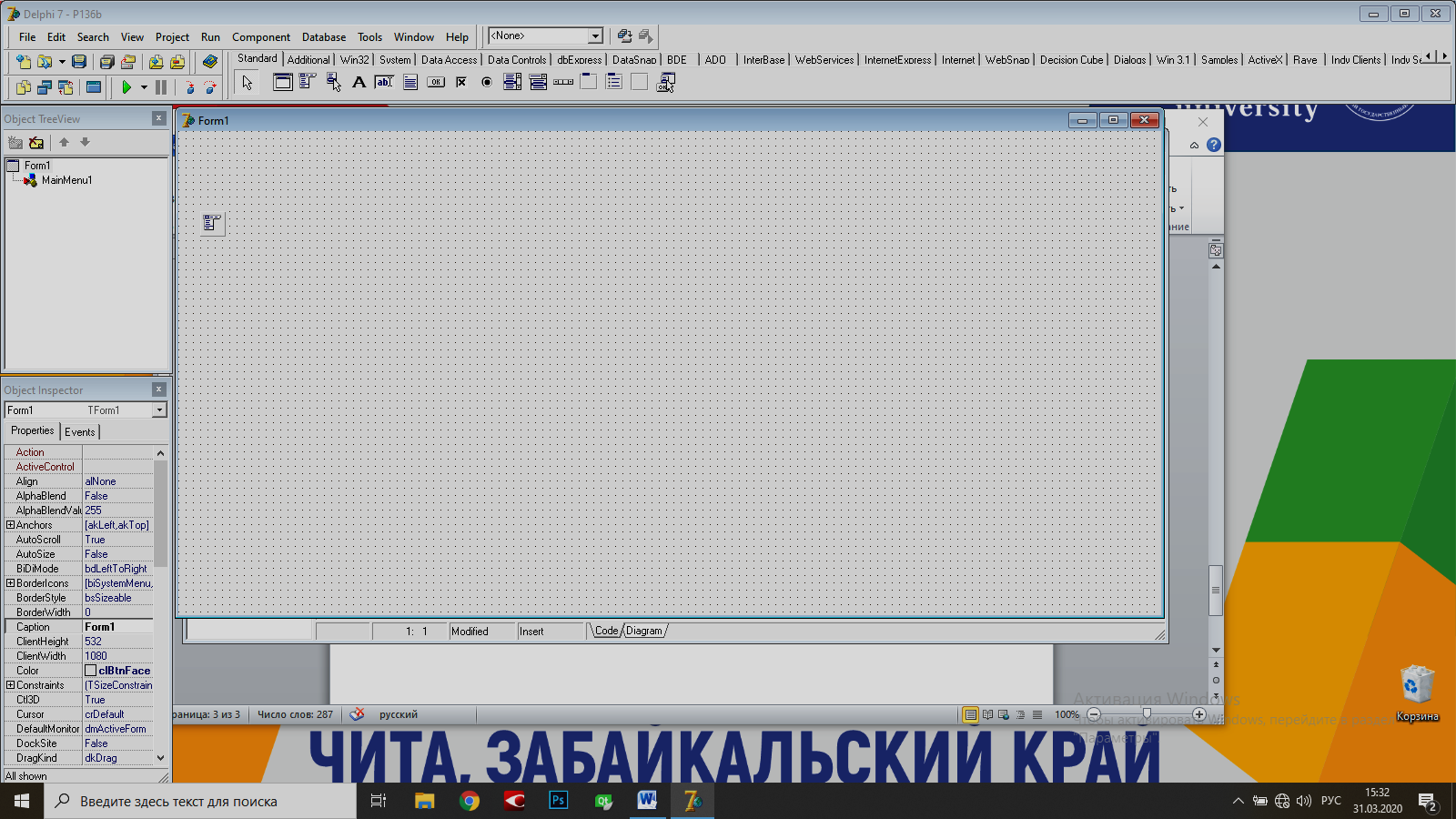
* Заполнение массива, возможны два варианта: вручную и случайными числами.
* Запись массива: в текстовый файл или в типизированный.
* Получение результата, например, среднеарифметическое элементов массива или сумма квадратов элементов.
* Открытие файлов: текстового или типизированного.
* Закрытие файлов: текстового или типизированного.
* Удаление динамического массива.

Для создания меню нужно перейти в окно формы, т.к. заголовок этого окна не менялся, то по умолчанию оно называется **Form1.**

В Delphi 7 имеются две компоненты: **MainMen**u – главное меню и **PopupMenu** - всплывающее меню. Обе расположены на странице **Standard.** Будем использовать **MainMen**u.

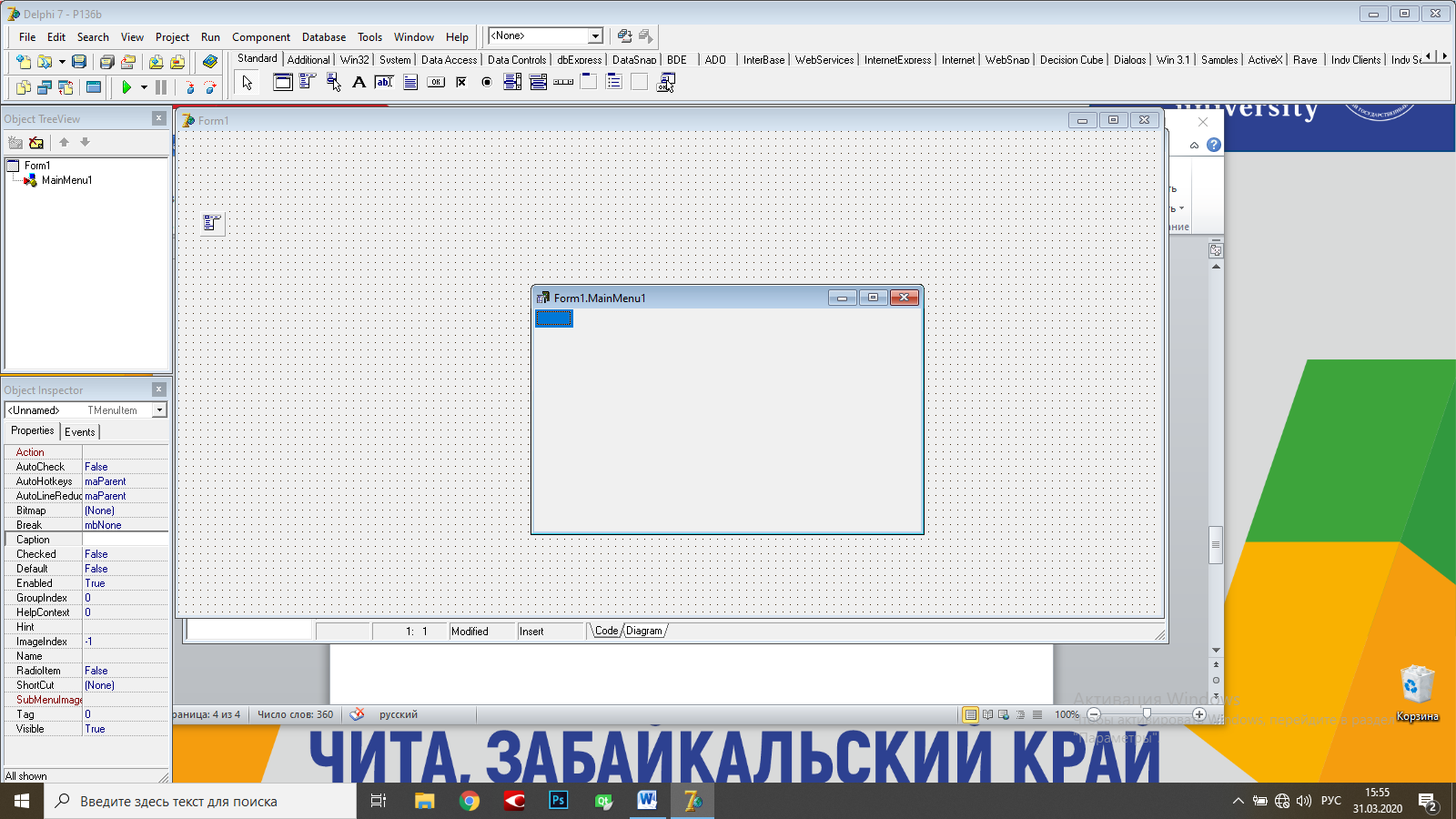


Для переноса **MainMen**u в окно формы щёлкаем ЛКМ по логотипу **MainMen**u, а затем щёлкаем ЛКМ в окне формы:

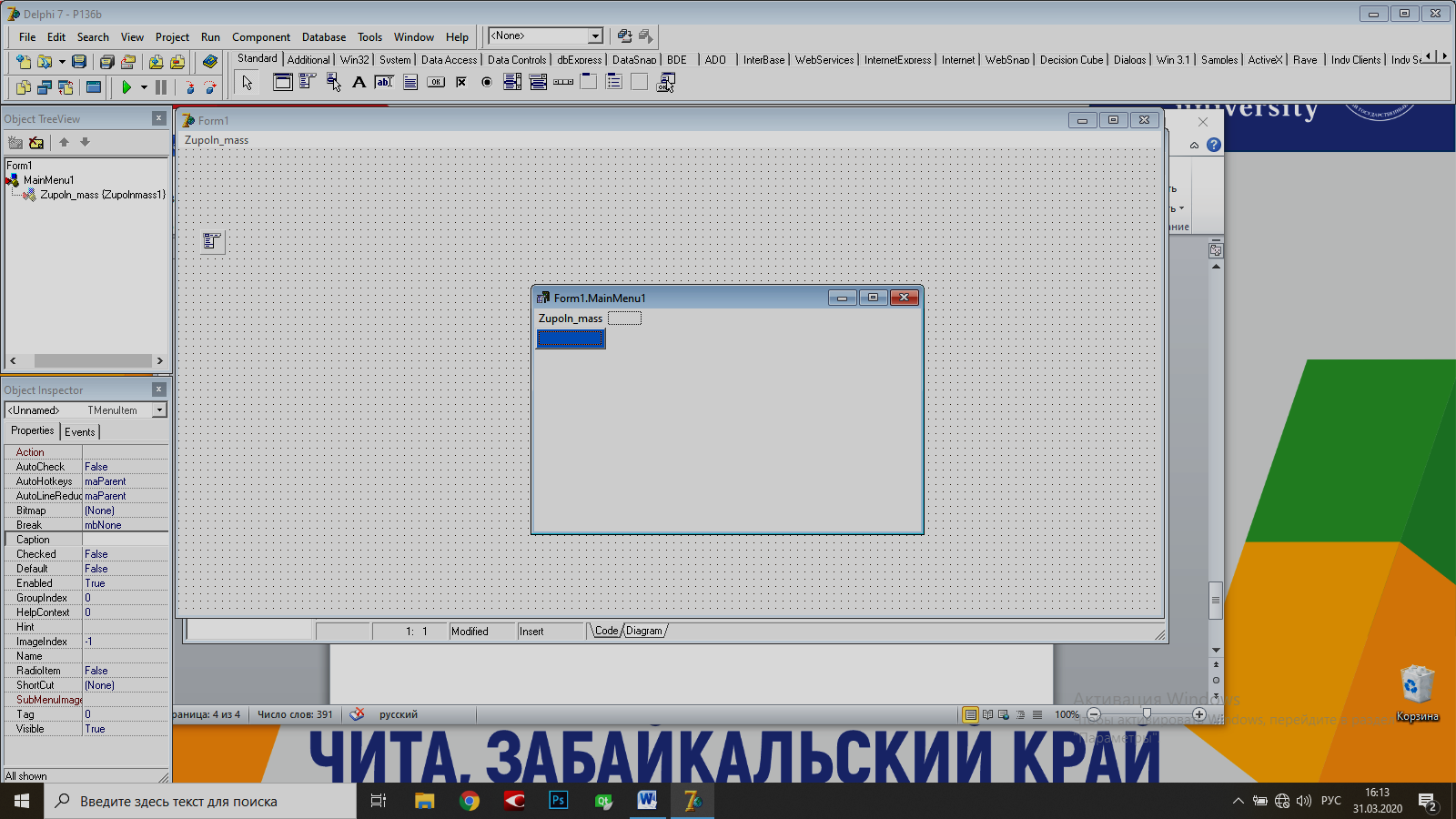


Компонент **MainMen**u помещён на форму. **Это не визуальный компонент**, т.е. при работе приложения **будет видно меню**, сгенерированное компонентом, а не сам компонент. Его при проектировании можно поместить в любую точку окна формы, лишь бы не мешал проектированию.

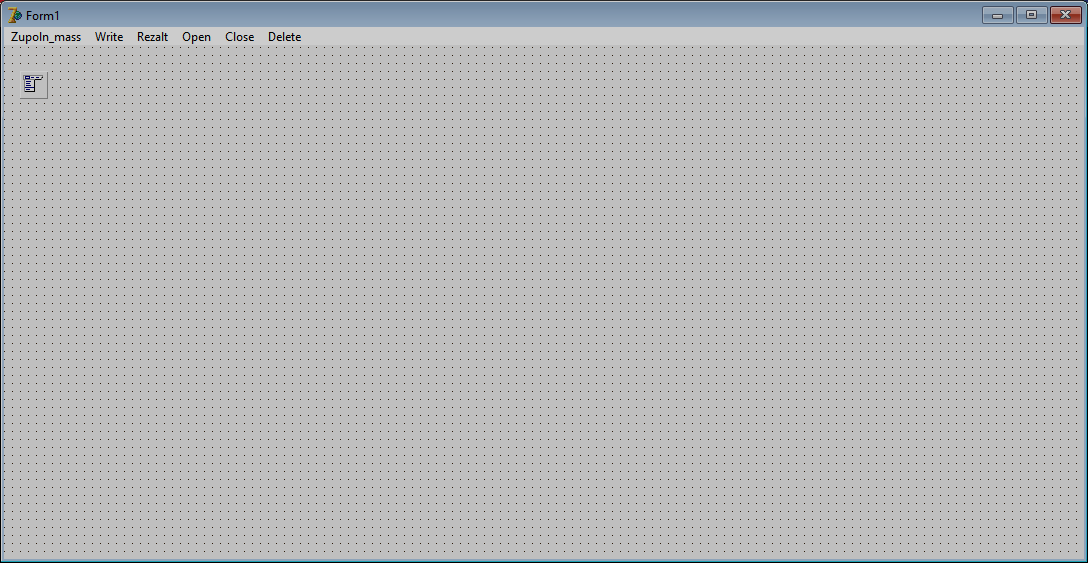
Основное свойство **MainMen**u - **Items**. Это свойство заполняется с помощью специального конструктора меню. Вызвать его можно двойным щелчком ЛКМ на компоненте **MainMen**u. Есть и другой способ (найти его самостоятельно) . В результате вызова конструктора откроется окно:



В нём можно сгенерировать всё меню. В нашем случае заголовок первого элемента меню – заполнение массива. Наберём его латинскими буквами в свойстве Caption инспектора объектов (Object Inspector). При нажатии клавиши ввод получим:



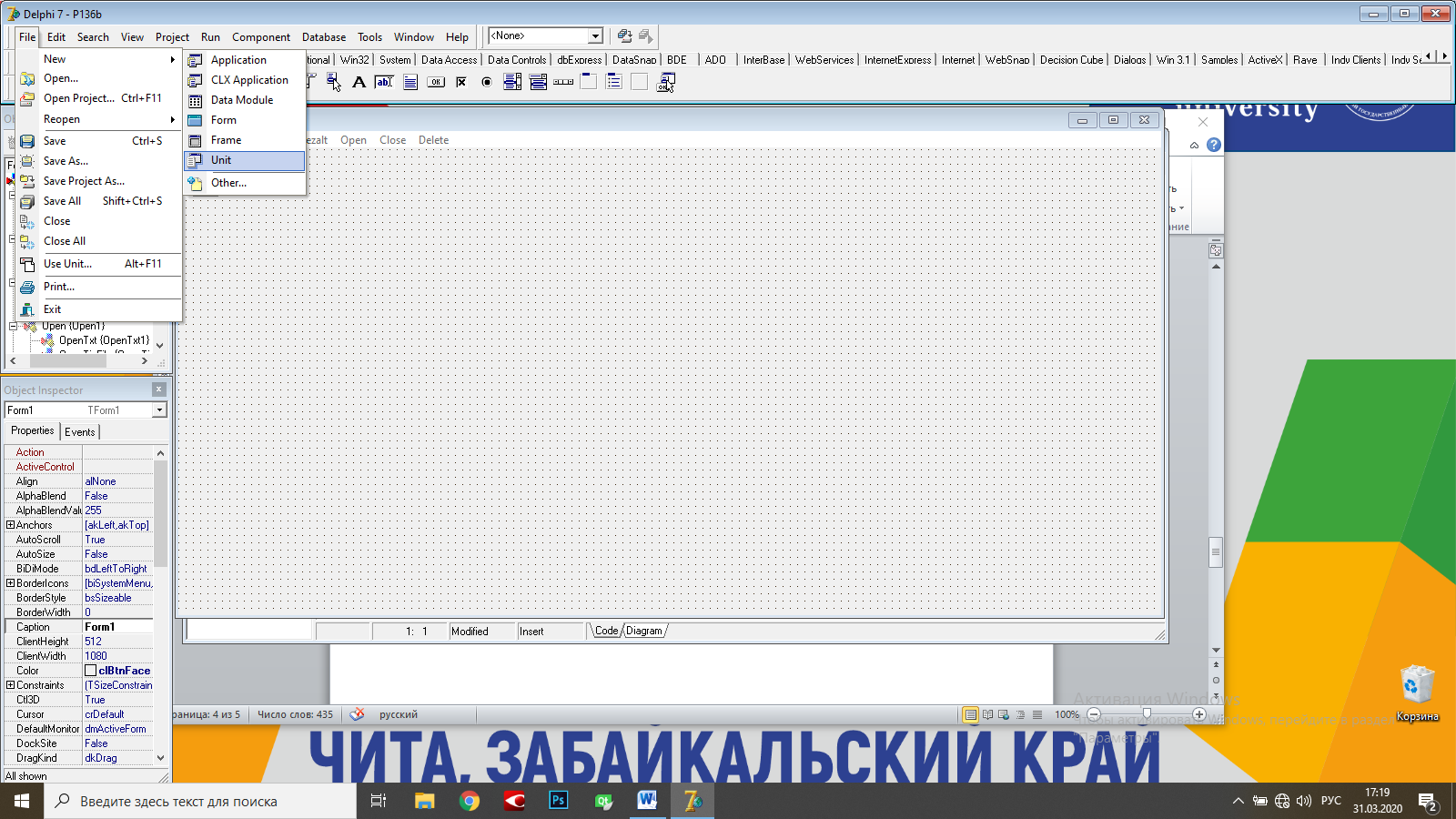
Точно также, нажимая ЛКМ на одном из прямоугольников, можно заполнить либо элемент подменю, либо элемент основного меню. В результате получим меню:



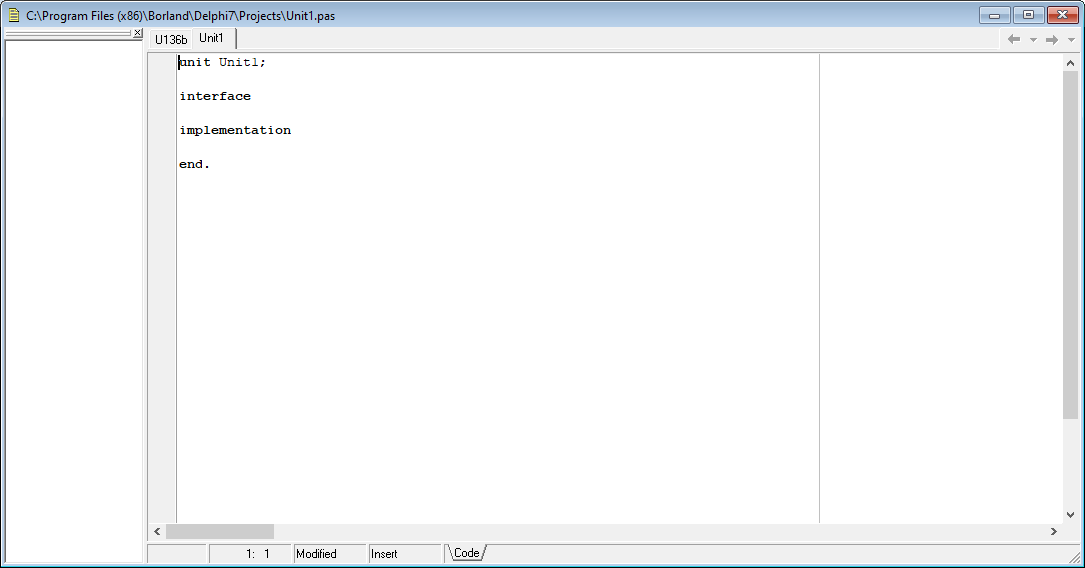
1. Для того, чтобы операции меню реализовывались, необходимо написать процедуры их выполняющие. Для этого нужно:

* создать модуль;
* откомпилировать его;
* добавить в модуль формы.

Выполнив команду **File| New|Unit**, рисунок ниже:



получим окно редактирования для создания модуля.



Вначале изменим Unit1 на Unit136b, чтобы понимать в каком проекте он используется, или чтобы имена модулей на пересекались. Для этого Выполняем команду**: File|Save As…** и в окне **Save As…** выбираем паку, в которой будет храниться модуль и его имя:



В окне результат сохранения создаваемого файла. Остаётся наполнить содержанием болванку модуля **Unit136b**.

В проекте придётся заполнять массив и выводить его в текстовый файл. Поэтому придётся обращаться к функциям преобразования типа, например,

StrToInt, использовать окна ввода /вывода. Для этого в модуле необходимо подгрузить стандартный модуль **SysUtils**.

Для того, чтобы не вводить имена файлов вручную, используем, показанное выше, окно **Open**, но для этого нужно подгрузить модуль **Dialogs.** Поэтому начало модуля имеет вид:

**unit** Unit136b;

**interface**

*//Модули содерж. ф-ии преобраз типа, диалог*

*// Далее нужно описать типы, используемые в приложении:*

**uses** SysUtils,Dialogs;

**type**

DinMas = **array** **of** **integer**; *//Тип динамического массива*

Tipfile = **file** **of** **integer**; *//Тип типизированного файла*

*// И наконец, опишем заголовки процедур, используемых в приложении:*

*// Ручное заполнение массива*

**procedure** ZapMassHand(**var** a: DinMas);

*// Вывод массива в текстовый файл*

**procedure** WriteMassText(**var** a: **array** **of** **integer**; **var** f: **TextFile**);

*// Интерфейсная секция модуля закончилась.*

*// В секции implementation напишем реализацию двух функций создаваемого модуля:*

**implementation**

**procedure** ZapMassHand(**var** a: DinMas);

**var**

n,i : **word**;

k: **string**[10]; *//Номер элемента в массиве*

**begin**

n:= StrToInt(InputBox('Количество элементов в массиве ',' ',' '));

SetLength(a,n); *//Выделение памяти для динамического массива*

*{ Заполнение массива}*

**for** i:=0 **to** n-1 **do**

**begin**

k:= IntToStr(i);

a[i]:= StrToInt(Inputbox('input' + k + 'element',' ',' '));

**end**;

**end**;

*// Вывод массива в текстовый файл*

**procedure** WriteMassText(**var** a: **array** **of** **integer**; **var** f: **TextFile**);

**var**

n,i: **integer**; *//n- количество элементов в строке*

**begin**

n:= StrToInt(InputBox('введите кол. элементов в строке ',' ',' '));

writeln(f, 'Заданный массив');

**write**(f, a[0]:5,' ');

**for** i:=1 **to** High(a) **do**

**if** (i **mod** n = 0) **then** writeln(f, a[i]:5)

**else** **write**(f, a[i]:5,' ');

writeln(f,' ');

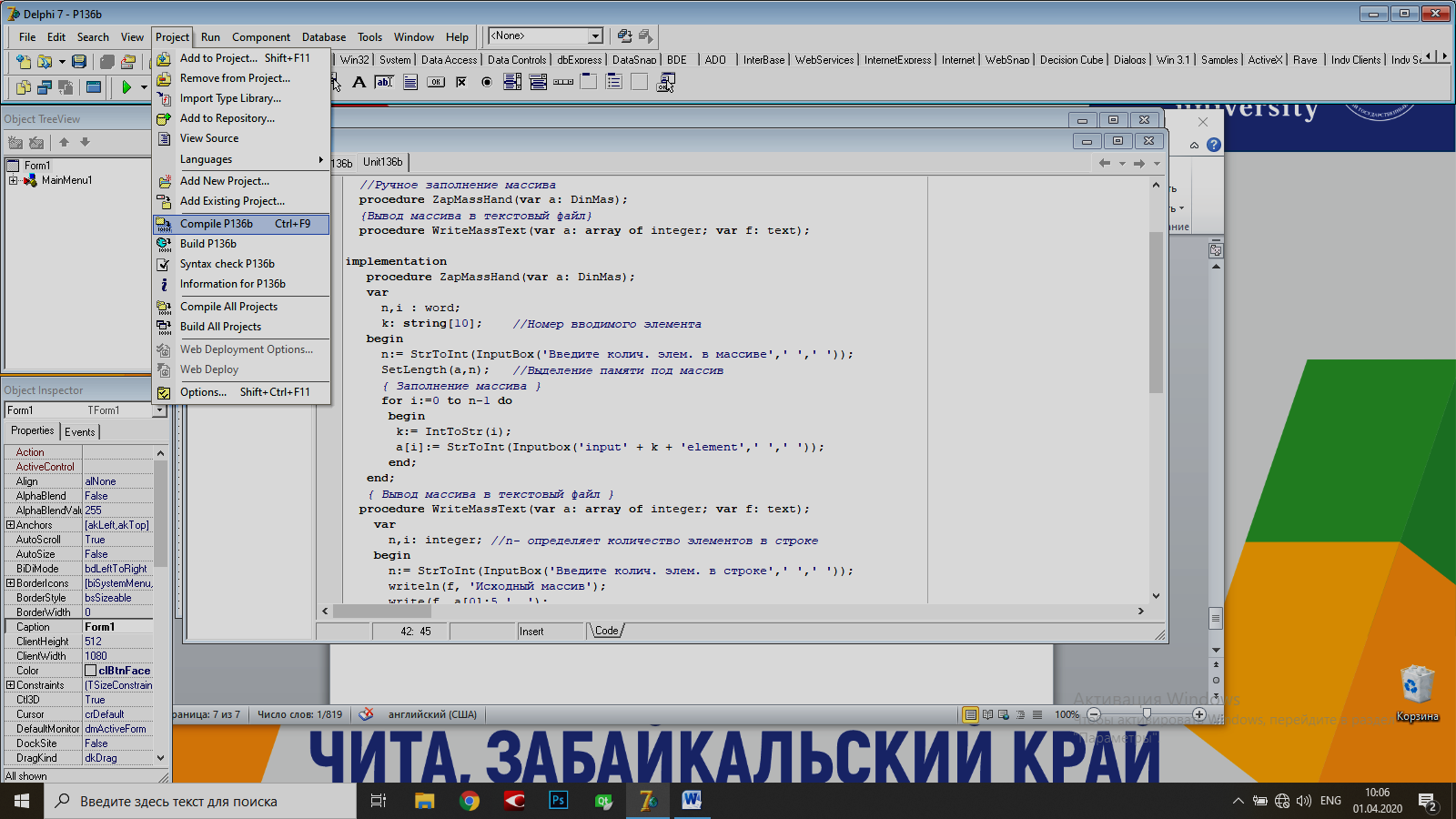
**end**;

**end**.

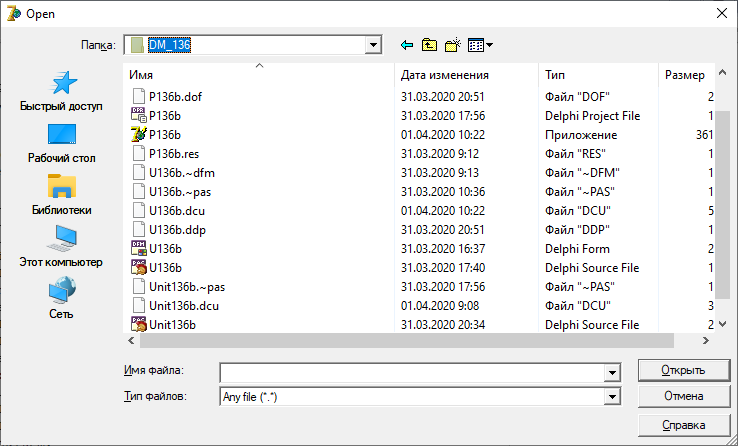
Почему в модуле реализованы только две процедуры, хотя выше обсуждалось большее количество операций? Дело в том, что часть из них будет реализована в модуле формы, а **другая будет добавляться в текст создаваемого модуля** после того как будет проверена работа уже написанных операций.

После создания текста модуля сохраним его в папке, в которой будут сохранены все файлы проекта. Результат такого сохранения показан выше. Осталось откомпилировать созданный модуль и подгрузить его к проекту.

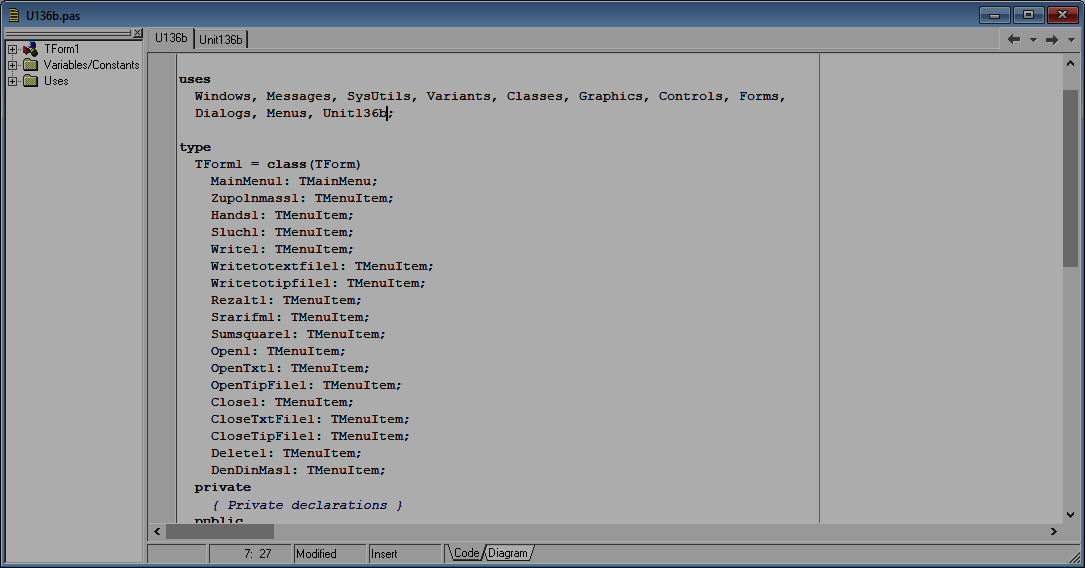
Компиляцию реализуем командой: **Project|CompileP136b** :



Либо можно использовать горячие клавиши: **Ctrl+F9.** Если теперь открыть папку, в которой хранятся файлы проекта, то увидим, что компиляция прошла успешно. Появился модуль Unit136b.dcu.

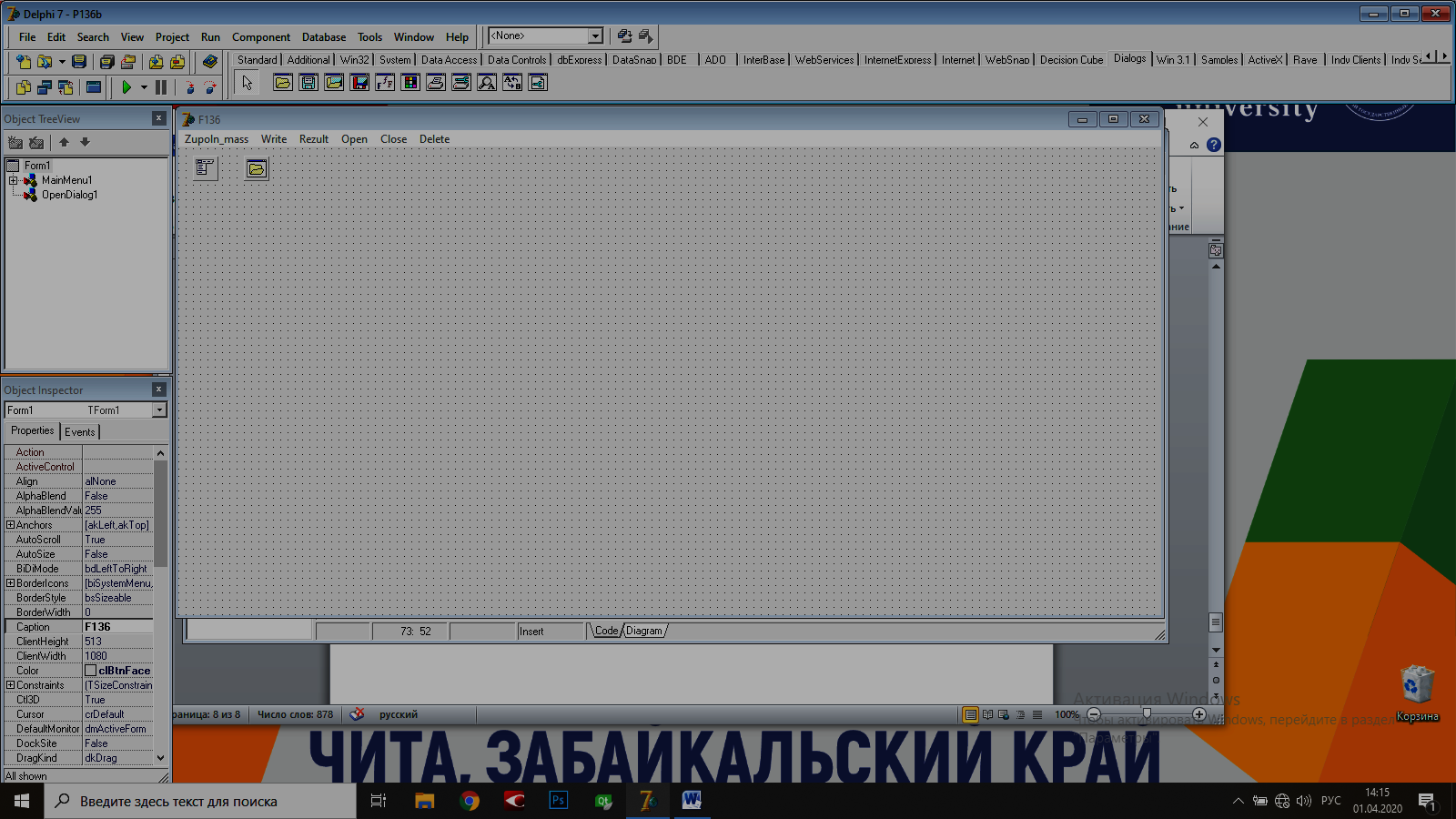


В окне модуля формы в uses предложение добавим имя(без расширения) созданного модуля:



Для приложения нужно:

1. Добавить на форму компонент **OpenDialog**, который находится на странице **Dialogs** и переносится на форму также как компонент **MainMenu;**



1. Написать обработчики событий тестируемых операций. Предварительно добавим в секцию **var** модуля формы:

**var**

**Form1: TForm1;**

глобальные переменные, значение которых понятно из коментария:

dm: DinMas; //имя динамического массива

fDM: Tipfile; //файловая переменная для работы с типизир файлом

ftxt: TextFile; //файловая переменная для работы с текстовым файлом

Теперь эта секция выглядит так:

**var**

**Form1: TForm1;**

**dm: DinMas; //**имя динамического массива

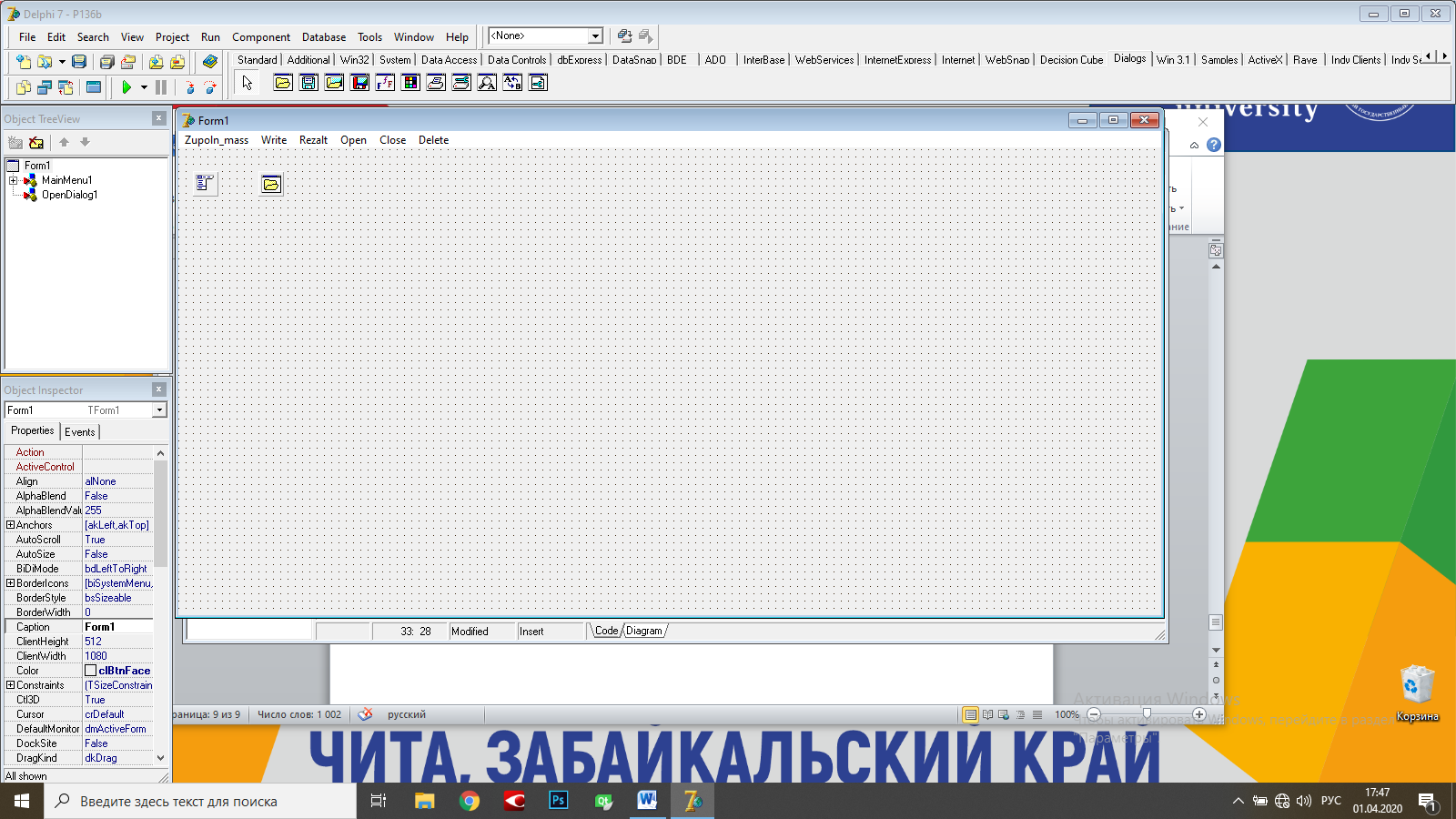
**fDM: Tipfile; //**файловая переменная для работы стипизир файлом

**ftxt: TextFile; //**файловая переменная для работы с текст файлом

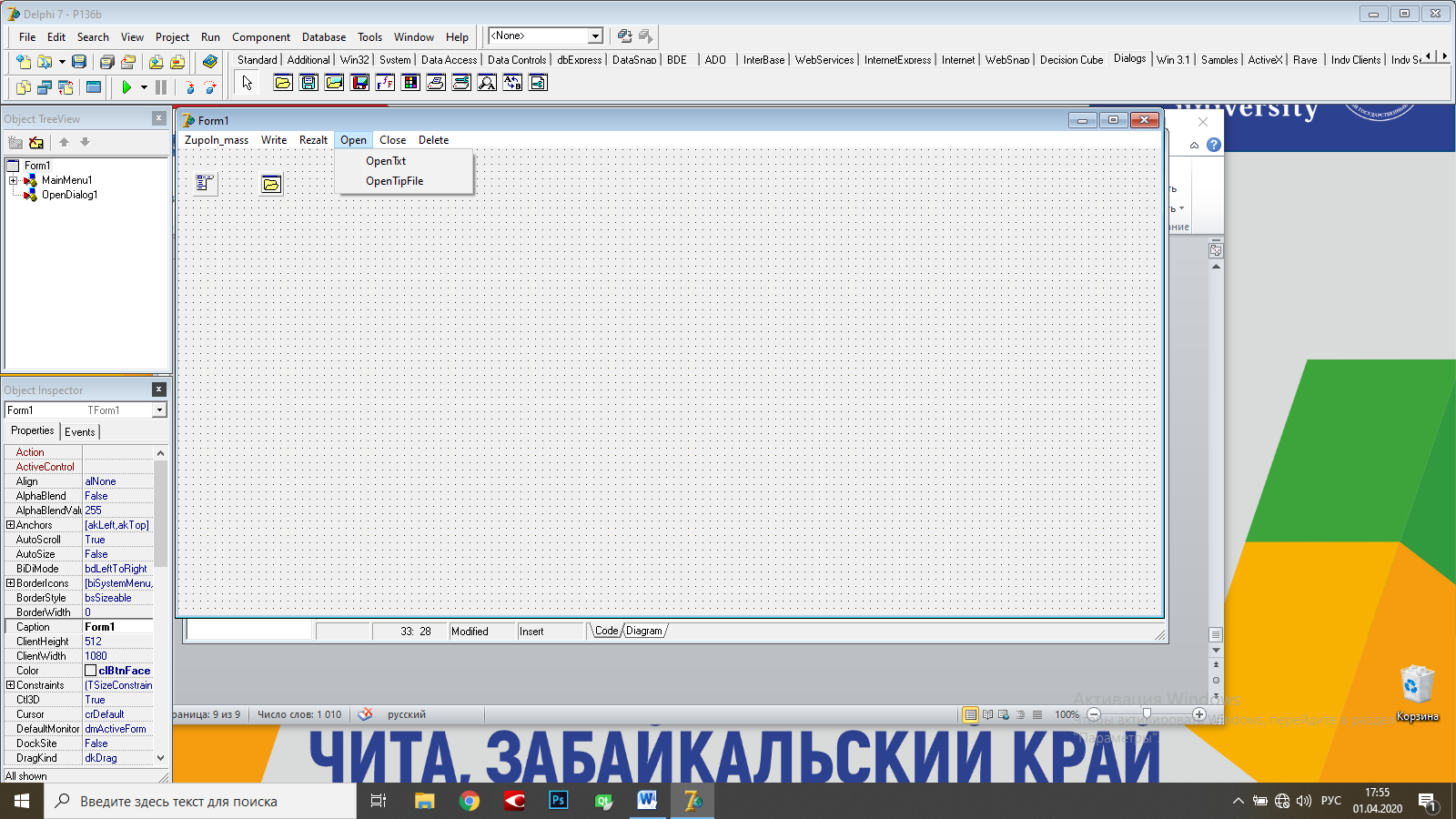
**implementation**

1. Остаётся написать обработчики событий для кнопок меню и запустить приложение. Будет создано несколько обработчиков, остальные создаются по образу и подобию, т.е. их нужно создать самостоятельно и проверить в деле.

Перейдём в окно формы, имя окна не изменяли и по умолчанию оно имеет имя - **Form1**.



Переходим к опции **Open** щёлкаем на ней ЛКМ:



В появившемся окне подменю - двойной щелчок ЛКМ на опции OpenTxt. Delphi в описании класса TForm1создаст заголовок обработчика события при щелчке ЛКМ на кнопке OpenTxt:

TForm1 = class(TForm)

MainMenu1: TMainMenu;

Zupolnmass1: TMenuItem;

Hands1: TMenuItem;

Sluch1: TMenuItem;

Write1: TMenuItem;

Writetotextfile1: TMenuItem;

……………{под многоточием скрыто часть данных класса TForm1}

CloseTipFile1: TMenuItem;

Delete1: TMenuItem;

DenDinMas1: TMenuItem;

OpenDialog1: TOpenDialog;

procedure OpenTxt1Click(Sender: TObject);

Помимо заголовка: procedure OpenTxt1Click(Sender: TObject); в секции

**implementation** модуля формы создаётся заготовка для реализации обработчика события:

implementation

{$R \*.dfm}

procedure TForm1.OpenTxt1Click(Sender: TObject);

begin

end;

end.

Остаётся написать код выполняемый обработчиком при его вызове. Вот один из вариантов:

procedure TForm1.OpenTxt1Click(Sender: TObject);

var s:string;

begin

if not OpenDialog1.Execute then exit;

s:= OpenDialog1.FileName;

AssignFile(ftxt,s);

Append(ftxt);

end;

Локальный параметр **s** нежен для хранения имени открываемого файла. Оператор **if not OpenDialog1.Execute then exit;** при неудачном открытии диалогового окна прерывает работу обработчика. В случае нормального открытия диалогового окна в нём пользователь выбирает открываемый файл и его имя сохраняется в свойстве **FileName** объекта **OpenDialog1.** Понятно, можно обойтись и без переменной **s.** Она визуально упрощает список параметров следующего оператора  **AssignFile(ftxt,s);.** Этот оператор выполняет туже задачу, что и **Assign** в диалектах Turbo Pascal, ABC. В данном случае он связывает глобальную файловую переменную **ftxt**  с файлом по имени **s**. (Оператор **Assign** выполняет в Delphi другую задачу).

**Append(ftxt); -** открывает текстовый файл для добавления строк в конец файла.

Точно также создаём заготовки обработчиков CloseTxtFile, DelDinMas, Hands, Writetotextfile. Появляются 4 заготовки в **interface** секции

procedure OpenTxt1Click(Sender: TObject);

procedure CloseTxtFile1Click(Sender: TObject);

procedure DenDinMas1Click(Sender: TObject);

procedure Hands1Click(Sender: TObject);

procedure Writetotextfile1Click(Sender: TObject);

и 4 в **implementation**

procedure TForm1.OpenTxt1Click(Sender: TObject);

var s:string;

begin

if not OpenDialog1.Execute then exit;

s:= OpenDialog1.FileName;

AssignFile(ftxt,s);

Append(ftxt);

end;

procedure TForm1.CloseTxtFile1Click(Sender: TObject);

begin

end;

procedure TForm1.DenDinMas1Click(Sender: TObject);

begin

end;

procedure TForm1.Hands1Click(Sender: TObject);

begin

end;

procedure TForm1.Writetotextfile1Click(Sender: TObject);

begin

end;

end.

Остаётся заполнить заготовки:

procedure TForm1.CloseTxtFile1Click(Sender: TObject);

begin

CloseFile(ftxt);

end;

procedure TForm1.DenDinMas1Click(Sender: TObject);

begin

SetLength(dm,0);

end;

procedure TForm1.Hands1Click(Sender: TObject);

begin

ZapMassHand(dm);

end;

procedure TForm1.Writetotextfile1Click(Sender: TObject);

begin

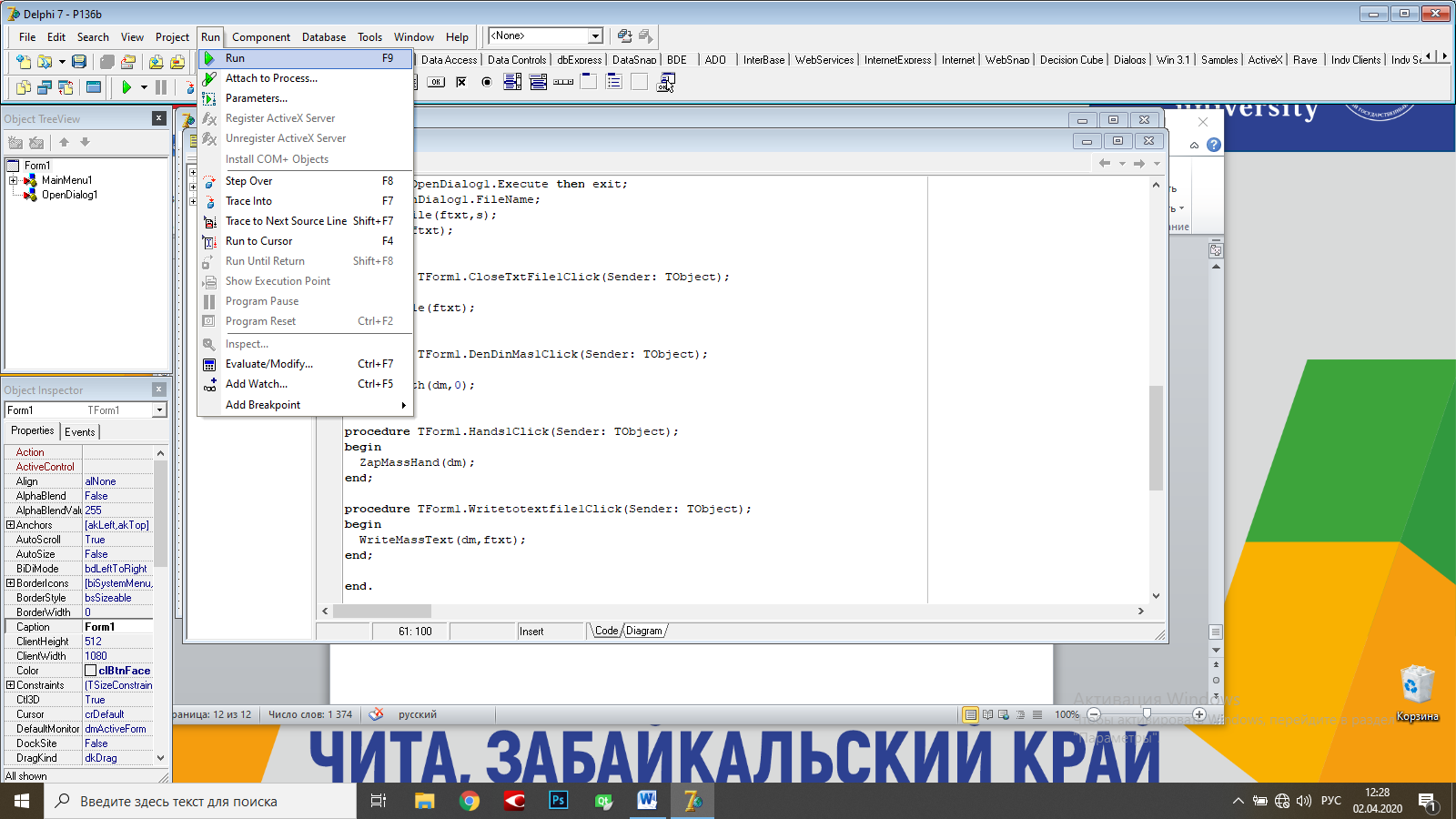
WriteMassText(dm,ftxt);

end;

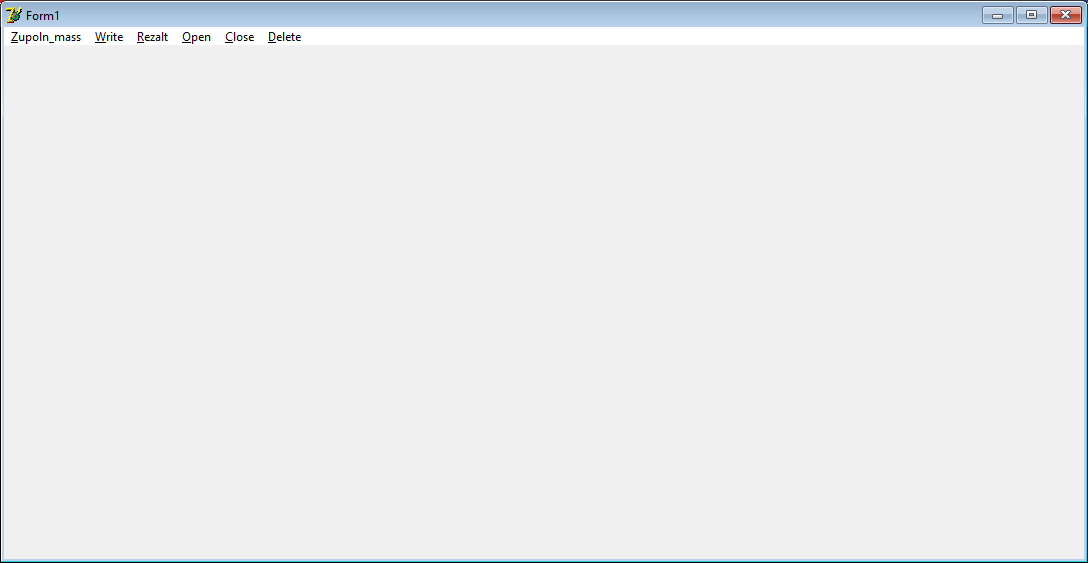
end.

Следует отметить, что параметры **ftx, dm** в обработчиках **- глобальные**, т.е. описаны в модуле формы и их не надо передавать в списке формальных параметров в заголовках обработчиков.

Остаётся проверить работу приложения. Если приложение открыто, то выполняем команду главного меню IDE: Run| Run или нажимаем клавишу F9 на клавиатуре:

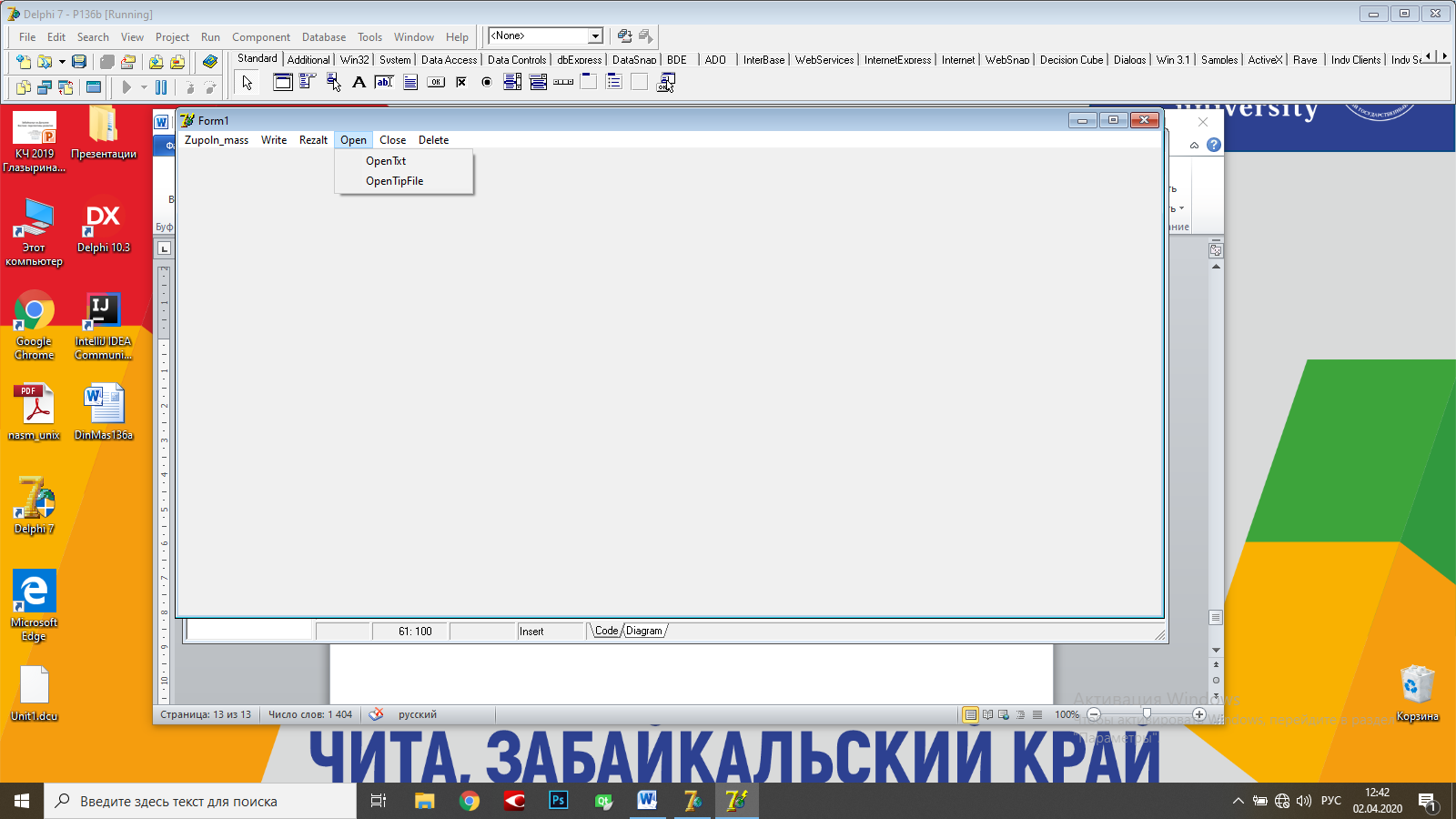


Приложение будет запущено на выполнение, если всё сделано правильно, то получим главное окно для работы с приложением:

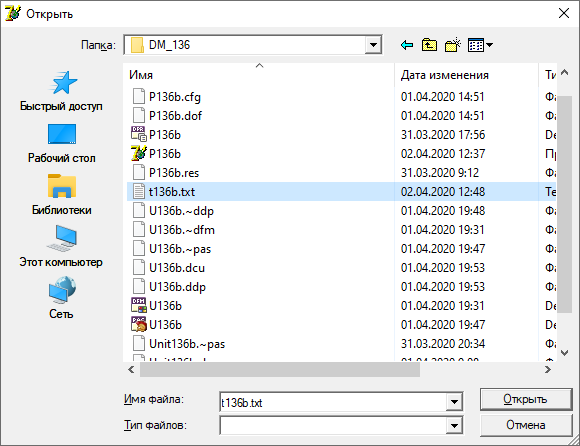


Последовательно проверим работу созданных обработчиков:

1. Откроем для записи предварительно созданный текстовый файл:

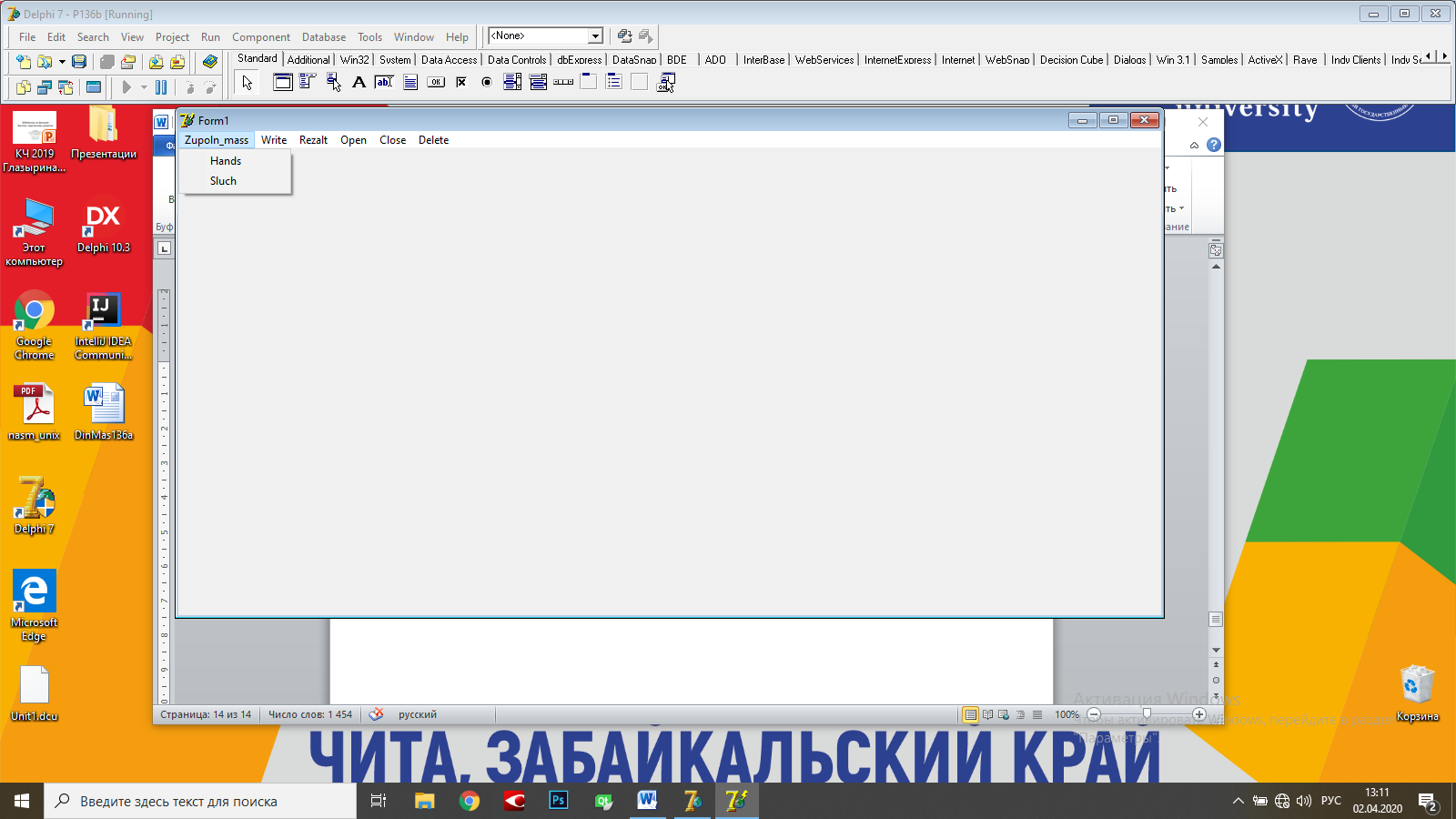


В окне Open выберем файл, в который будет записываться текст:

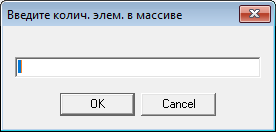


В данном случае это файл t136b.txt , т.е. обработчик **OpenTxt** выполнил свою задачу: создал окно открыть, в котором можно выбрать папке с нужным файлом; а далее связать выбранный файл с файловой переменной и открыть его.

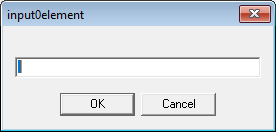
1. Проверим заполнение динамического массива:

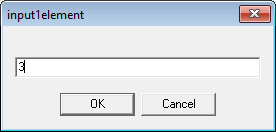


Выполним команду **Zapoln\_mass|Hands** , в появившемся окне



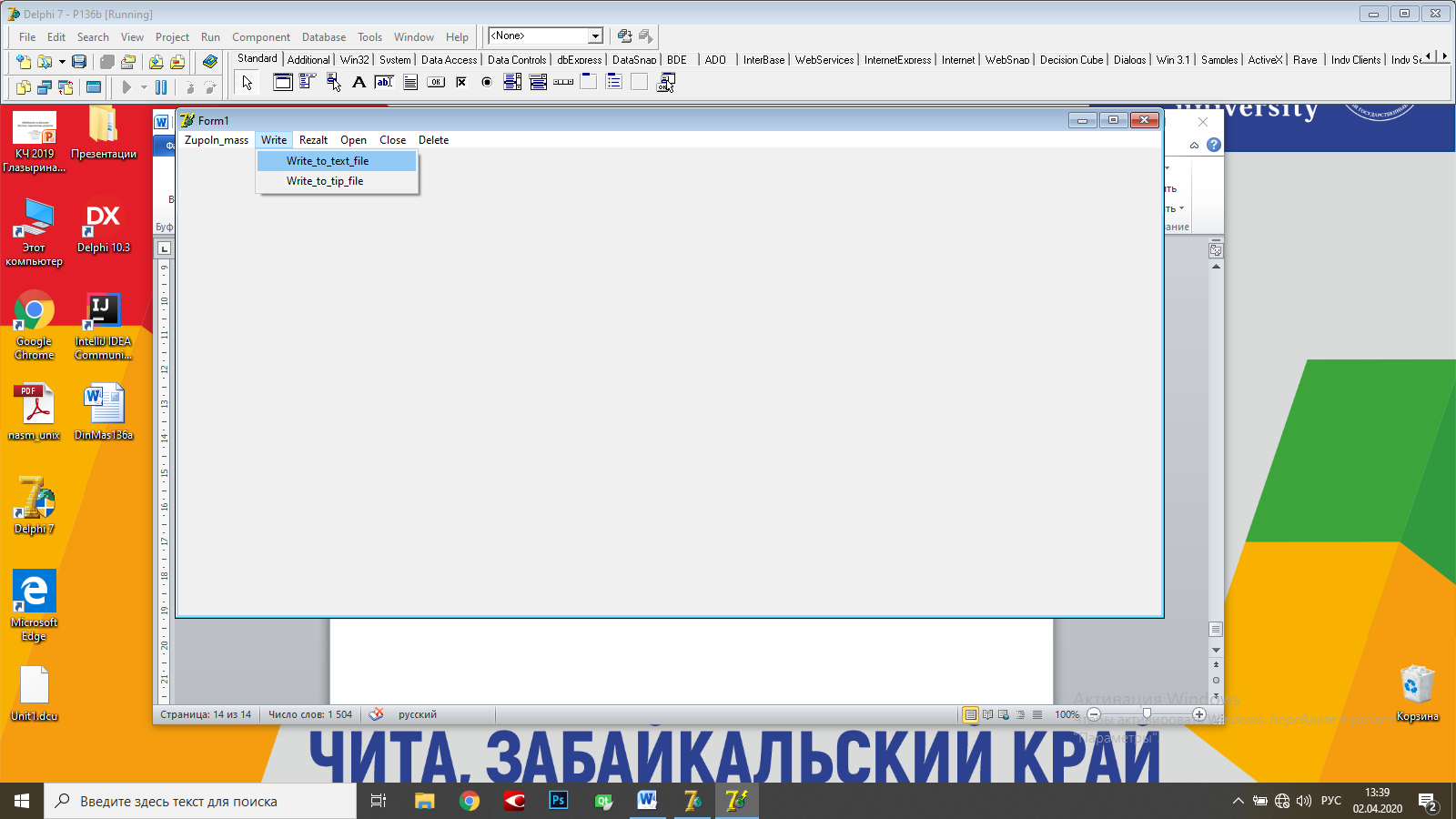
введём количество элементов в массиве, например, 5. Далее будут появляться окна для ввода i-го элемента:

 затем

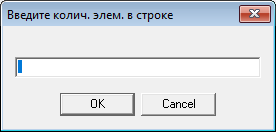
 и т.д. Всего окон будет 5.

Сбоев при вводе элементов не было, т.е. выполнил свою задачу.

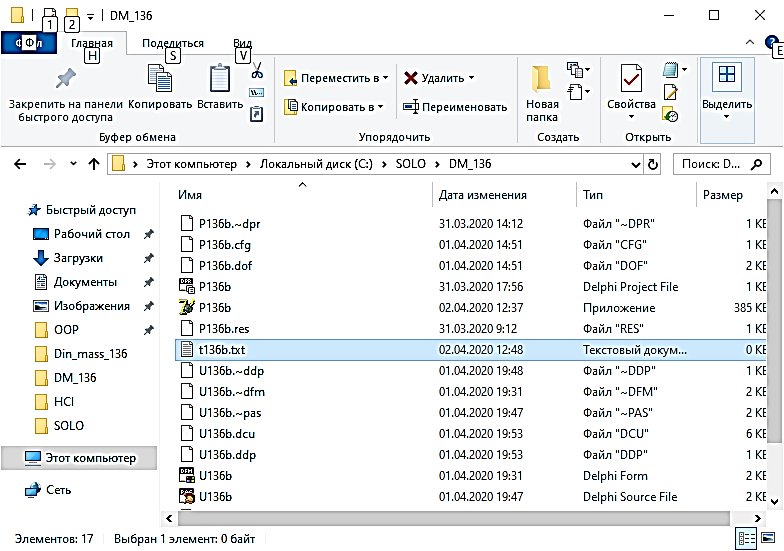
1. Теперь перепишем динамический массив в текстовый файл, т.е. выполним команду Write|Write\_to\_text\_file:



Перед выполнением этой команды **обязательн**о должна быть выполнена команда **OpenTxt**. При выполнении появляется окно, в котором определяется количество элементов в строке:



Остаётся проверить, что записано в текстовый файл. Свернём приложение, перейдём в папку, где находятся файлы данного проекта:



Открываем текстовый файл, но он пуст. Всё потому, что данные, выводимые в файл застряли в буфере обмена. Вытолкнуть их можно при помощи процедуры **Flush**(параметр-файловая переменная), которая используется для завершения обмена с файлом без его закрытия. Так как **Flush** не включена в текст обработчика Write\_to\_text\_file придётся закрыть текстовый файл и тогда увидим, что в него записывалось.

Закроем файл, выполнив команду **CloseTxtFile** из менюсозданного приложения, одновременно проверив работу этого обработчика.



Теперь файл не пуст. Осталось проверить работу обработчика, который удаляет динамический массив.

1. Удалим динамический массив, команда Delete|DenDinMas. Массив опустеет, т.е. переменная dm=nil.
2. Откроем текстовый файл, команда Open|OpenTxt.
3. Команда приведёт к аварийному останову проекта. Это произошло потому, что обработчике Write\_to\_text\_file массив пуст, dm=nil. Кстати, этот результат говорит о недостатке в конструкции Write\_to\_text\_file. В ней нет проверки условия dm=nil.

# Динамические массивы

Одним из мощнейших средств языка Delphi являются динамические массивы. Их основное отличие от обычных массивов заключается в том, что они хранятся в динамической памяти. Этим и обусловлено их название.

Размер динамического массива можно изменять во время работы программы.

Динамический массив объявляется без указания границ:

var  
DynArray: array of Integer;

Переменная DynArray представляет собой ссылку на размещаемые в динамической памяти элементы массива. Изначально память под массив не резервируется, количество элементов в массиве равно нулю, а значение переменной DynArray равно nil.

Работа с динамическими массивами:

SetLength(DynArray, 50); // Выделить память для 50 элементов

При изменении размера массива значения всех его элементов сохраняются. При этом последовательность действий такова: выделяется новый блок памяти, значения элементов из старого блока копируются в новый, старый блок памяти освобождается.

При уменьшении размера динамического массива лишние элементы теряются.

При увеличении размера динамического массива добавленные элементы не инициализируются никаким значением и в общем случае их значения случайны.

Определение количества элементов производится с помощью функции Length:

N := Length(DynArray); // N получит значение 100

Элементы динамического массива всегда индексируются от нуля. Доступ к ним ничем не отличается от доступа к элементам обычных статических массивов:

DynArray[0] := 5; // Присвоить начальному элементу значение 5  
DynArray[High(DynArray)] := 10; // присвоить конечному элементу значение 10

К динамическим массивам, как и к обычным массивам, применимы функции Low и High, возвращающие минимальный и максимальный индексы массива соответственно. Для динамических массивов функция Low всегда возвращает 0.

Освобождение памяти, выделенной для элементов динамического массива, осуществляется установкой длины в значение 0 или присваиванием переменной-массиву значения nil (оба варианта эквивалентны):

SetLength(DynArray, 0); // Эквивалентно: DynArray := nil;

Необязательно по окончании использования динамического массива освобождать выделенную память, поскольку она освобождается автоматически при выходе из области действия переменной-массива.

Также, как и при работе со строками, при присваивании одного динамического массива другому, копия уже существующего массива не создается.

var  
 A, B: array of Integer;  
begin  
 SetLength(A, 100); // Выделить память для 100 элементов  
 A[0] := 5;  
 B := A; // A и B указывают на одну и ту же область памяти!  
 B[1] := 7; // Теперь A[1] тоже равно 7!  
 B[0] := 3; // Теперь A[0] равно 3, а не 5!  
end.

В приведенном примере, в переменную B заносится адрес динамической области памяти, в которой хранятся элементы массива A (другими словами, ссылочной переменной B присваивается значение ссылочной переменной A).

Для непосредственной работы с динамическими массивами, существует ряд функций, это:

* функция **Length — определяющая длину массива;**
* функция **High — возвращающая наибольший индекс массива;**
* функция **Low —** **возвращающая наименьший индекс массива;**

Для усечения динамического массива можно использовать функции: Copy, Finelize.

# Источники

А.Я. Архангельский, Язык Pascal и основа программирования в Delphi, §3.1.3, стр.166