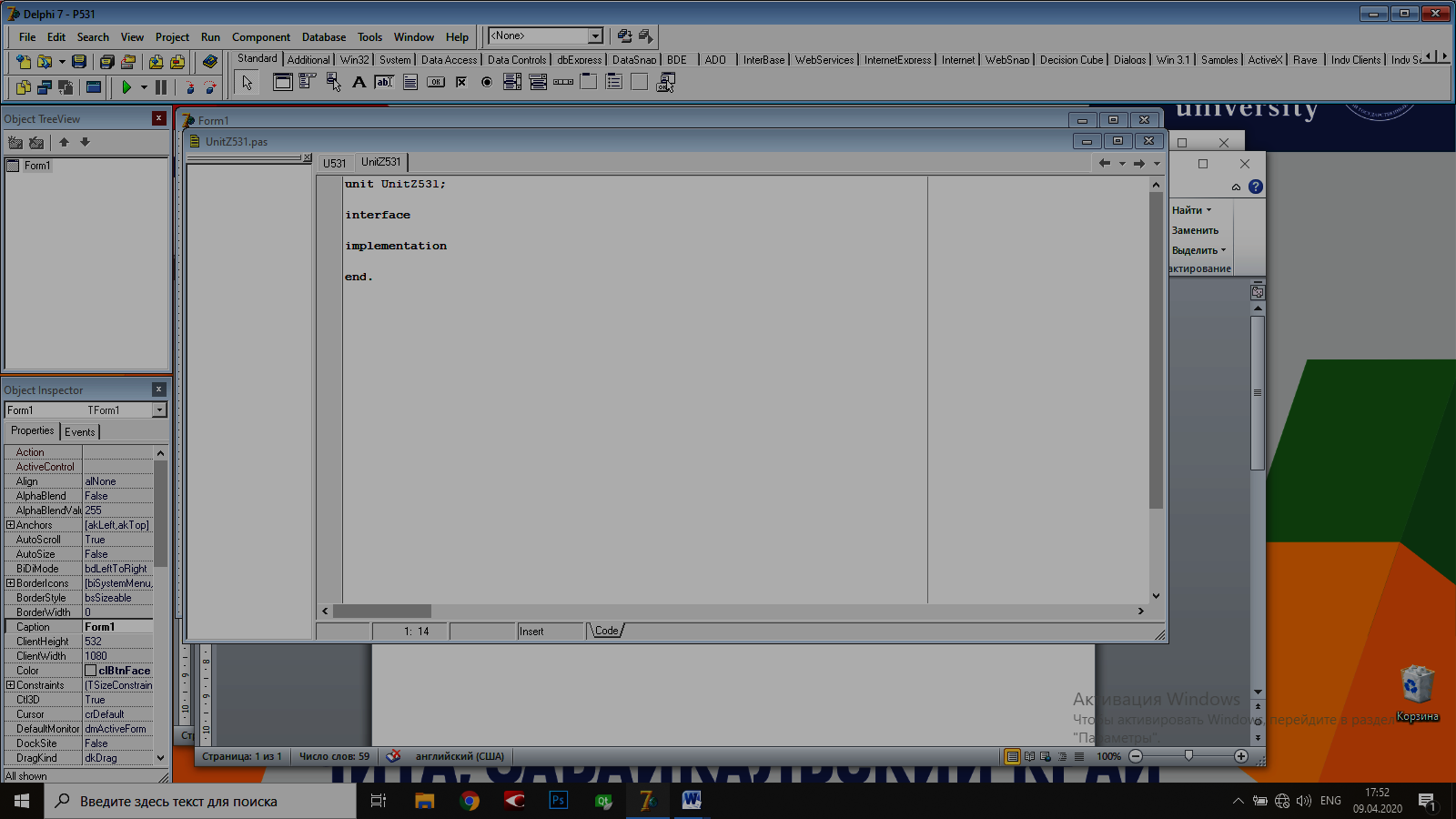
Алгоритм создания приложения для работы с односвязанным списком , задача №531

1. **Создать папку**, в которой будут храниться все файлы проекта,

В данном примере адрес, созданной папки: **c:\SOLO\Z531**

1. **Открыть** папку **Z531** и в ней создать текстовый файл, в котором будут храниться результаты тестирования проекта.
2. **Запустить** Delphi 7 и сохранить проект и модуль формы под именами:

**P531, U531 в** папке **Z531.**



1. **Создать свой модуль**, в котором будут описаны тип указателя на узел и тип узла. Реализованы операции для работы со списком. Этот список позволит определиться с компонентами меню и другими компонентами:
2. Создание списка.
3. Удаление списка.
4. Вывод результата обработки списка в текстовый файл.
5. Сохранение списка в типизированном файле.
6. Создание списка, используя данные из типизированного файла.

В данном случае модуль имеет имя **UnitZ531** и заготовка создана в окне редактора кода. В интерфейсную часть модуля добавим:

1. Стандартные модули:

**uses SysUtils, Dialogs;**

Как и в предыдущем примере будем использовать диалоговое окно для открытия и закрытия файлов и функции преобразования типов из модуля **SysUtils**.

б) Добавим секцию:

type

Puzel = ^Zp; // создали тип указателя на узел

Zp = record // узел - запись, содержащая поля **х** и **next**

x: real;

next: PUzel;

end;

FZap = file of real; //тип файла для хранения х1,х2,х3, …,хn

Добавим в интерфейсную часть создаваемого модуля заголовки операций, которые будут использоваться при работе со списком:

{Операции для работы с односвязанным списком }

procedure AddFirst(var f: PUzel; a: PUzel);{Вставить узел **a** первым в список}

procedure AddAfter(var old:PUzel; a: PUzel);{Вставить узел **a** после **old**}

{Построить список; **f** -указатель на голову списка}

procedure BuildSpisok(var f: PUzel);

{Вывод списка в текстовый файл}

procedure WriteSpText(var f: PUzel;var ftxt:Text);

{Создать из существующего списка новый}

procedure NewSpisok(var f: PUzel; var fn: PUzel);

{Выделить из списка первый узел и вернуть его пользователю}

procedure DelFirstElement(var f,a: PUzel);

{Выделить из списка узел, следующий за узлом old и вернуть его польз-лю}

procedure DelElement(var old,a: PUzel);

procedure DelSpisok(var f: PUzel); //Удалить список

В секции **implementation** реализуем перечисленные процедуры:

procedure AddFirst(var f: PUzel; a: PUzel);{Вставить узел **a** первым в список}

begin

a^. next := f; //Добавить узел **а** в голову списка

f :=a;

end;

Процедура получает указатель на голову списка **f**  (по ссылке, т.к. **f** будет изменен в теле процедуры) и указатель на добавляемый узел **а** (вопрос о том, где и как он создавался оставим на потом). На рисунке это выглядит так:

|  |  |
| --- | --- |
|  |  |

1) **а**

**F**

Перечёркнутая ячейка означает, что поле узла **next = nil.**

2) Добавляем **а** к списку:

**а**

**F**

3) Остаётся перевести указатель **f** на новую голову списка

a

f

Теперь два указателя на голову списка, но **а** – это указатель на отдельный узел, **значение которого будет изменяться** при создании нового узла, т.е. **а** в дальнейшем не будет показывать на голову списка.

Следующая процедура:

procedure AddAfter(var old:PUzel; a: PUzel);{Вставить узел **a** после **old**}

begin

a^.next := old^.next; //Связываем узел **а** с узлом, следующим за old

old^.next := a; // Связываем узел old с узлом **а.**

end;

Алгоритм, выполняемый процедурой AddAfter, можно изобразить так:

а old

f

Процедура AddAfter получает указатель на вставляемый узел **а** и указатель на узел из списка **old**, после которого будет вставлен узел **а.** Это реализуется так: вначале связываем узел **а** с узлом, следующим за узлом **old**

1. а old

f

2. Затем указатель old^.next переводим на **а**

а old

f

Разберём процедуру построения списка с «нуля», т.е. при f=nil.

procedure BuildSpisok(var f: PUzel); { **f** -указатель на голову списка}

var

a,d :PUzel; //а – указатель на созд. узел, а **d** на последний узел списка

ch: char; //переменная для выхода из процедуры

begin

f:= nil; //если убрать этот оператор, то эта процедура позволит добавлять //узлы в существующий список, для которого f <> nil

repeat

new(a); //выделить память под новый узел

{Заполнить поля узлы, на который указывает **а** }

a^.x := StrToFloat(InputBox('Введите реальное число',' ',' '));

a^.next := nil;

if (f = nil) then

begin

AddFirst(f,a); //вставить узел **а**  в голову списка f

d:= f; //запомнить указатель на последний узел списка

end

else

begin

AddAfter(d,a); //добавить узел в хвост списка

d := a; //запомнить указатель на последний узел списка

end;

ch:= InputBox('Для завершения ввода нажмите Y',' ',' ')[1];

until (ch = 'Y') or (ch ='y');

end;

{Создать из существующего списка новый}

{ f- указатель на существующий список, fn- на создаваемый}

procedure NewSpisok(var f: PUzel; var fn: PUzel);

var

xn: real; //перем., в которой будет храниться значение из последнего

// узла существующего списка

p,a,d: PUzel; //указатели на текущий, создаваемый и последний узлы

begin

xn := EndElemSp(f); //функция EndElemSp будет описана ниже

{определение первого узла нового списка }

p:= f; //текущий указатель для существующего списка

fn:=nil;

new(a);

a^.x := p^.x - xn;

AddFirst(fn,a);

d:= fn;

{создание следующих узлов нового списка }

p:= p^.next;

while not(p^.next = nil) do

begin

new(a); //выделяем память под узел

a^.x := p^.x - xn; //заполняем поле **х** узла **а**

AddAfter(d,a); //вставляем узел **а** после последний элемент списка

d := a; //перемещаем **d** на последний элемент списка

p:=p^.next;

end;

end;

Вернёмся к функции EndElemSp(f). Задача этой функции вернуть значение, поля **х** последнего узла списка, на который указывает **f**. Оно потребуется для создания узлов нового списка.

function EndElemSp(var f: PUzel):real;

var

p: PUzel;

begin

p:= f; //установить указатель p на первый узел списка f

{в цикле перемещение p с головы до хвоста списка и возвращение p^.x}

while not(p = nil) do

begin

if (p^.next = nil) then Result:= p^.x;

p:=p^.next;

end;

end;

Осталось рассмотреть операции удаления узлов из списка. Первая операция –

удаление узла из головы списка. По ссылке функция получает указатели: на голову списка f и указатель на удаляемый узел.

procedure DelFirstElement(var f,a: PUzel);

begin

a := f; //после этой операции **а** указывает на первый узел списка

f := f^.next; //указатель **f** перемещён на второй элемент списка

a^.next := nil; //теперь первый узел полностью выделен из списка

end; //после завершения процедуры узел **а** можно удалить или //использовать для построения нового списка

Следующая процедура не используется в данной задаче. Она интересна в смысле выделения узла из заданной позиции списка

procedure DelElement(var old,a: PUzel);

begin

a := old^.next;

old^.next := a^.next;

a^.next := nil;

end;

Графический алгоритм процедуры DelElement :

1. Дан список f и указатель, который будет указывать на узел, следующий за old.

old

f

Свяжем указатель **а** с узлом следующим за old: a := old^.next;

2. old

f

а

3.Теперь нужно перенаправить указатель на элемент, следующий за а:

old

f

а

Эту задачу и выполняет оператор old^.next := a^.next;

Остаётся разорвать связь узла **а** со следующим за ним узлом, что реализуется оператором a^.next := nil;

В результате получим:

**а** old

**F**

Теперь переходим к удалению списка из оперативной памяти. Эту операцию нужно выполнить перед закрытием проекта, Иначе указатели будут обнулены, а выделенная под них память (процедура new) останется недоступной в данном сеансе работы с компьютером.

Удаление динамической переменной, память под которую была выделена процедурой  **new,** реализуется процедурой Dispose(a), где параметр **а**-удаляемая переменная. Код процедуры удаления списка:

procedure DelSpisok(var f: PUzel); //Удалить список

var

a: PUzel;

begin

while (f <> nil) do

begin

DelFirstElement(f,a);

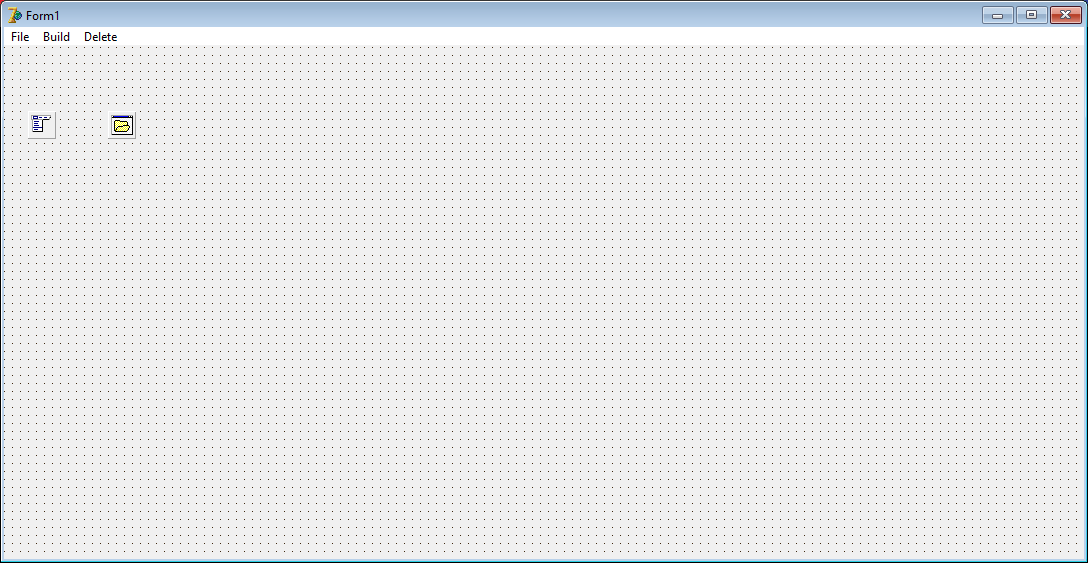
Dispose(a);

end;

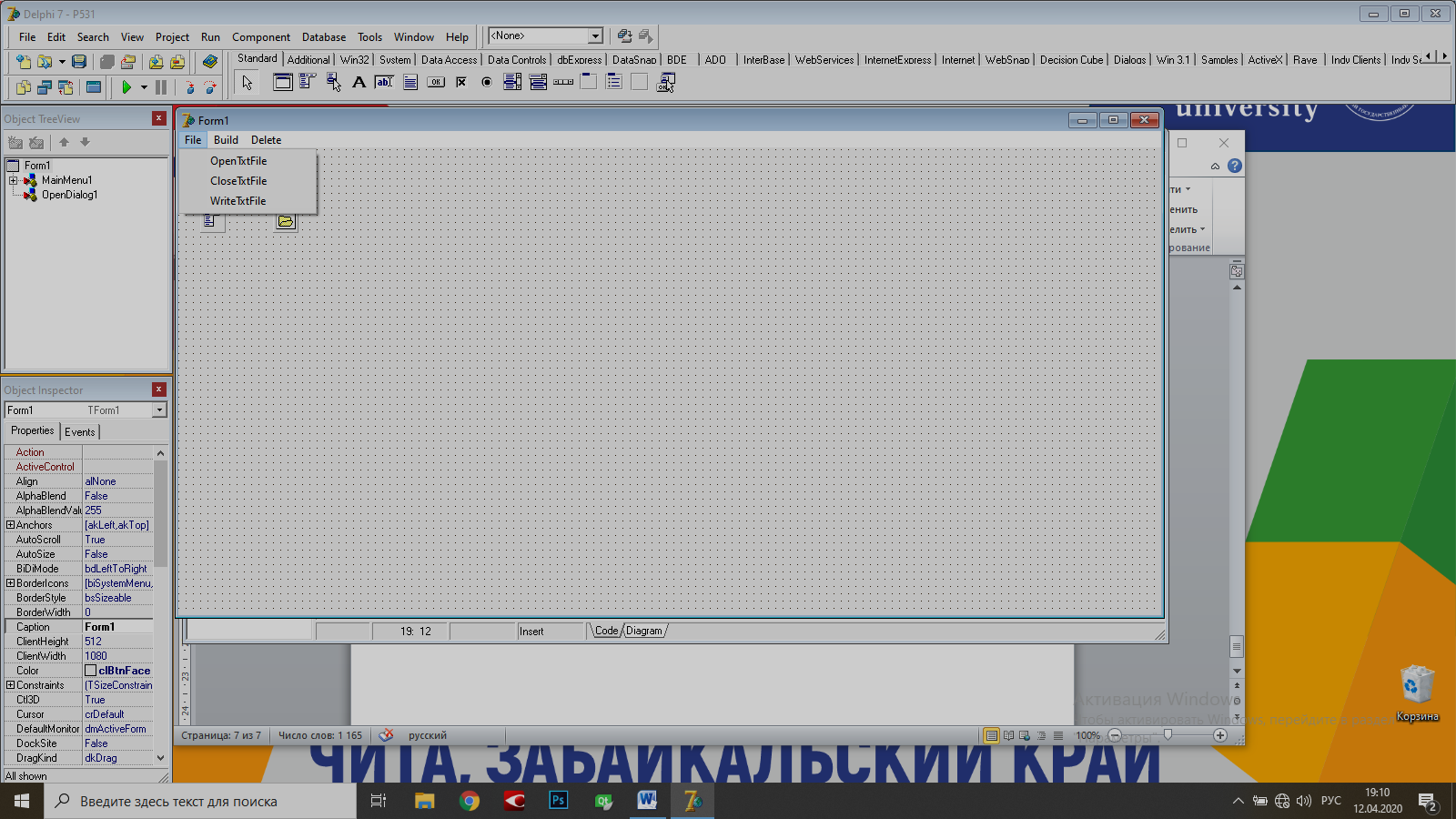
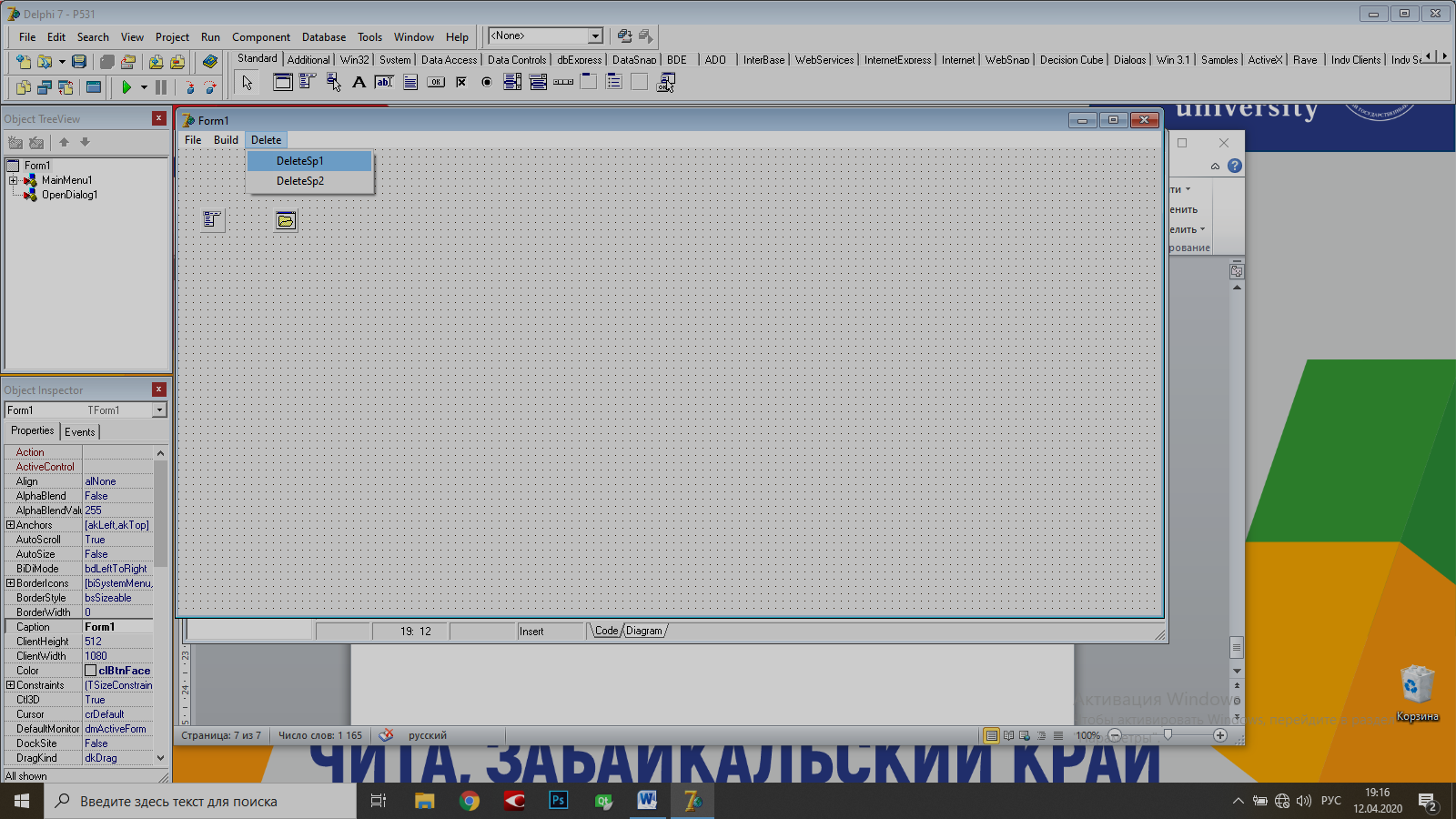
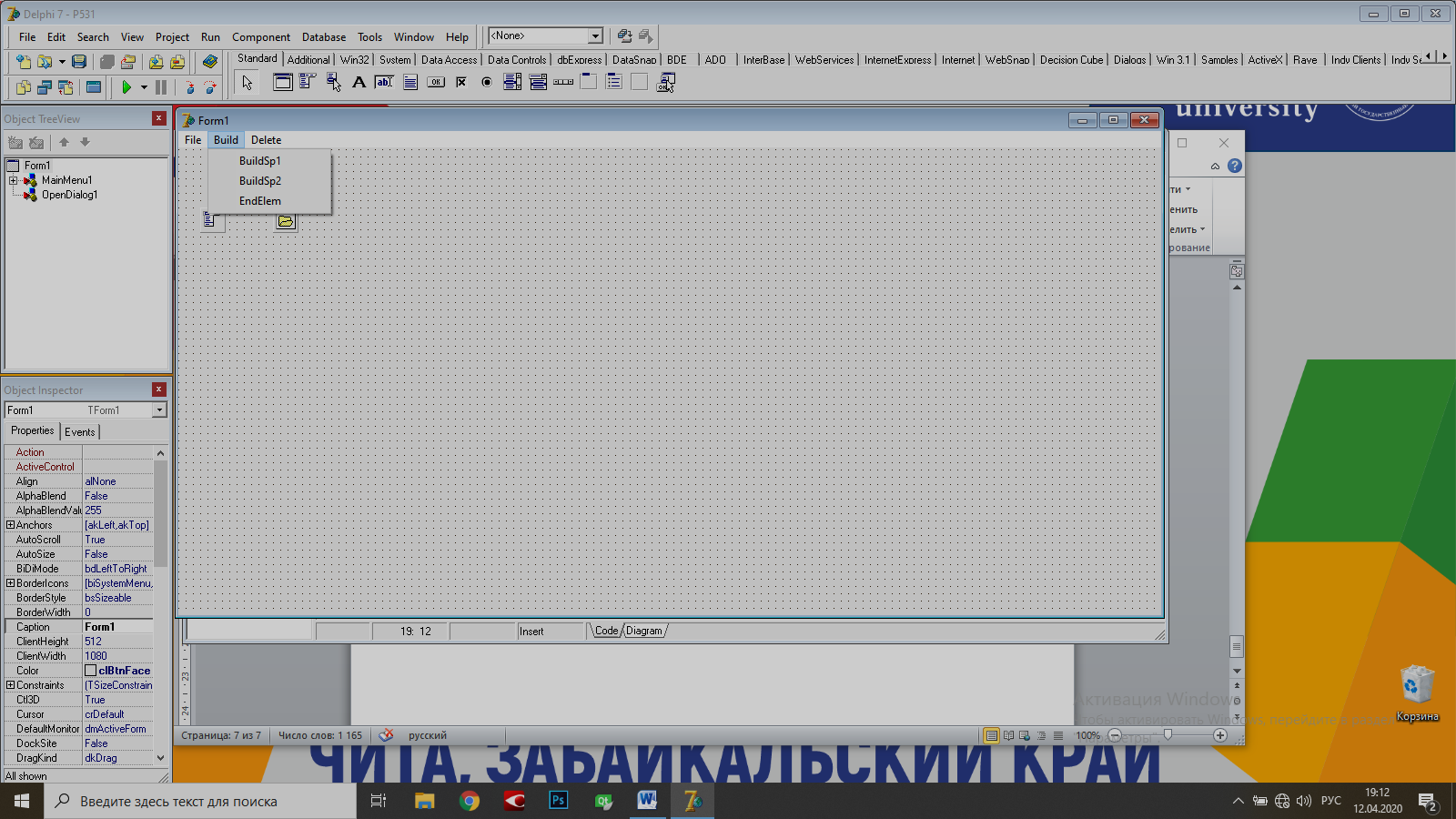
end;

Алгоритм ясен из приведённого кода.

Модуль для реализации задачи готов. Остаётся создать оконный интерфейс проекта. Также как и в работе с динамическим массивом, на форму данного проекта поместим компоненты MainMenu и OpenDialog, которые находятся на страницах Standard и Dialogs и создаём главное меню:



В главном меню созданы три раздела: File, Build, Delete. Каждый из них имеет следующие подразделы:

Создадим обработчики событий для элементов меню. В результате интерфейсная часть модуля формы примет вид:

unit U531;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Menus, UnitZ531;

type

TForm1 = class(TForm)

MainMenu1: TMainMenu;

File1: TMenuItem;

OpenTxtFile1: TMenuItem;

CloseTxtFile1: TMenuItem;

WriteTxtFile1: TMenuItem;

Build1: TMenuItem;

BuildSp11: TMenuItem;

BuildSp21: TMenuItem;

Delete1: TMenuItem;

DeleteSP11: TMenuItem;

DeleteSp21: TMenuItem;

OpenDialog1: TOpenDialog;

EndElem1: TMenuItem;

procedure OpenTxtFile1Click(Sender: TObject);

procedure CloseTxtFile1Click(Sender: TObject);

procedure WriteTxtFile1Click(Sender: TObject);

procedure BuildSp11Click(Sender: TObject);

procedure BuildSp21Click(Sender: TObject);

procedure DeleteSP11Click(Sender: TObject);

procedure DeleteSp21Click(Sender: TObject);

procedure EndElem1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

Sp1,Sp2: PUzel; //указатели на основной и результирующий списки

ftxt: text;

implementation

{$R \*.dfm}

{Коды обработчиков событий. Часть из них описана в работе DinMas136a }

procedure TForm1.OpenTxtFile1Click(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

CloseFile(ftxt);

end;

procedure TForm1.WriteTxtFile1Click(Sender: TObject);

var

n: integer;

p: PUzel;

begin

n:= StrToInt(InputBox('Для нач-го списка n=1, для результата n=2',' ',' '));

if (n=1) then p:=Sp1 else p:=Sp2;

if (n = 1) then writeln(ftxt,'Begin spisok')

else writeln(ftxt,'Rezult spisok');

while not (p = nil) do

begin

write(ftxt, p^.x:5:2,' ');

p:= p^.next;

end;

writeln(ftxt,' ')

end;

procedure TForm1.BuildSp11Click(Sender: TObject);

begin

BuildSpisok(Sp1);

end;

procedure TForm1.BuildSp21Click(Sender: TObject);

begin

NewSpisok(Sp1,Sp2);

end;

procedure TForm1.DeleteSP11Click(Sender: TObject);

begin

DelSpisok(Sp1);

end;

procedure TForm1.DeleteSp21Click(Sender: TObject);

begin

DelSpisok(Sp2);

end;

procedure TForm1.EndElem1Click(Sender: TObject);

var n: real;

begin

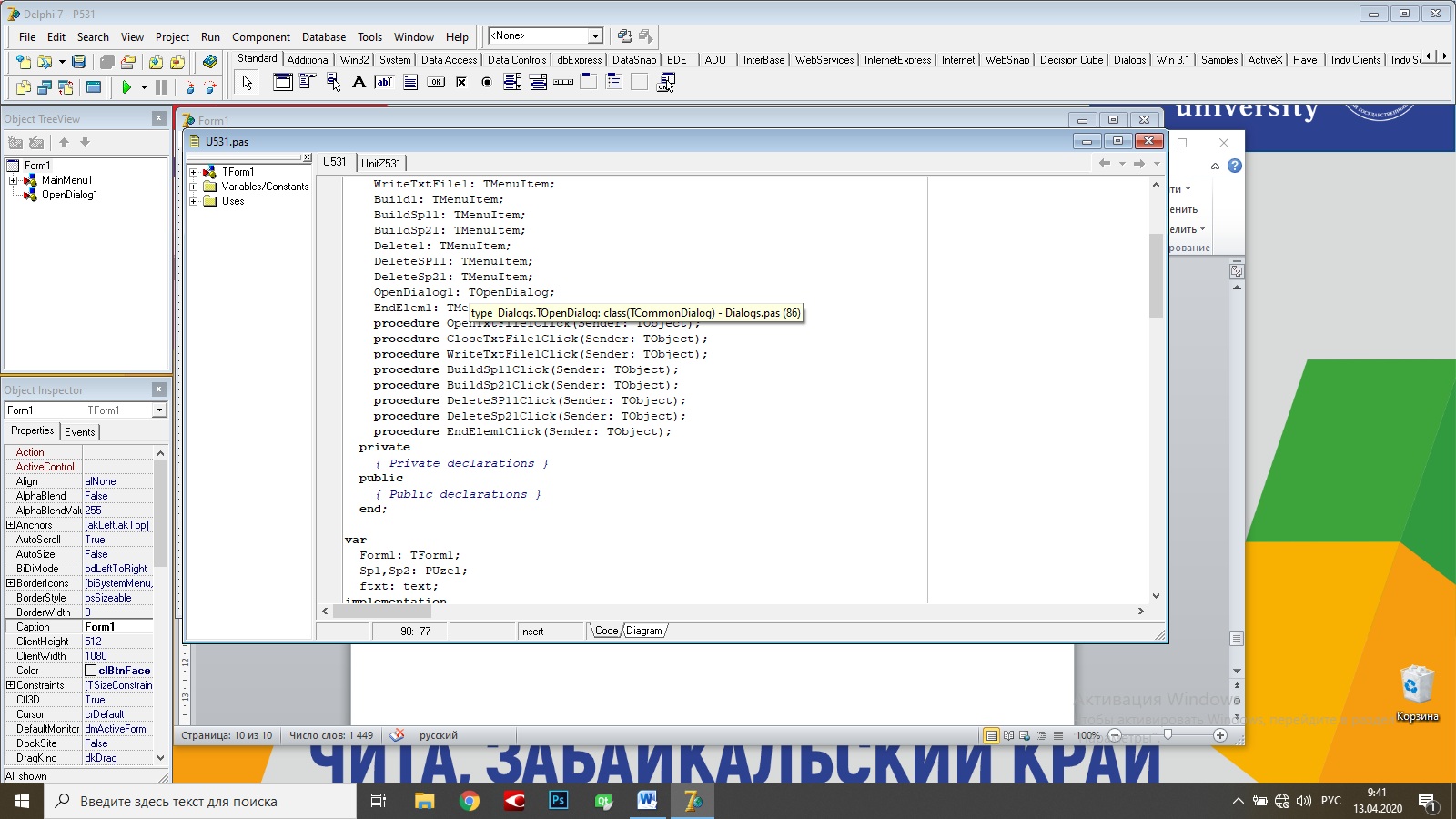
n:= EndElemSp(Sp1);

ShowMessage('n=' + FloatToStr(n));

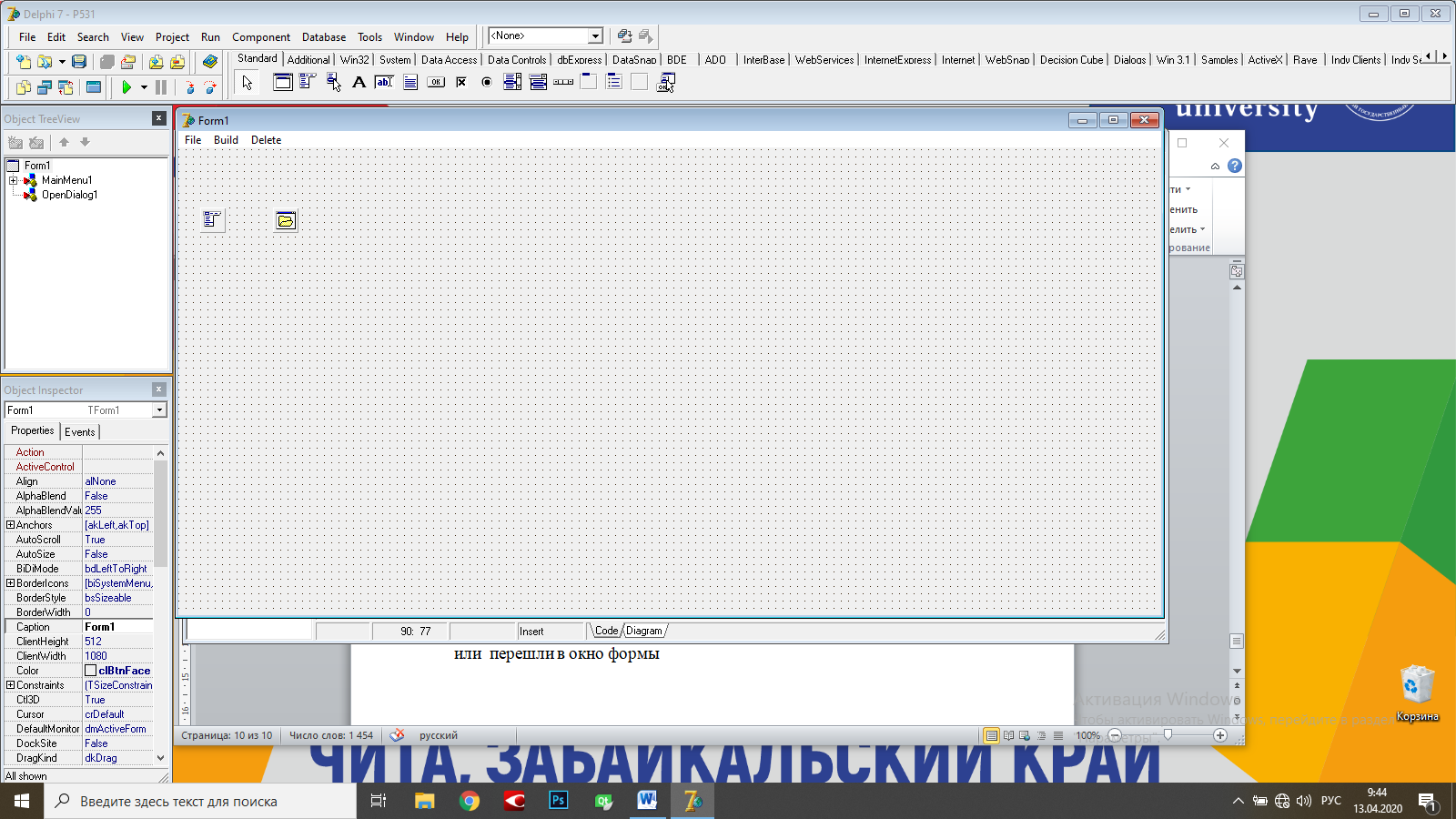
end;

end.

Остаётся проверить работу созданного приложения. Если мы находимся в окне модуля формы

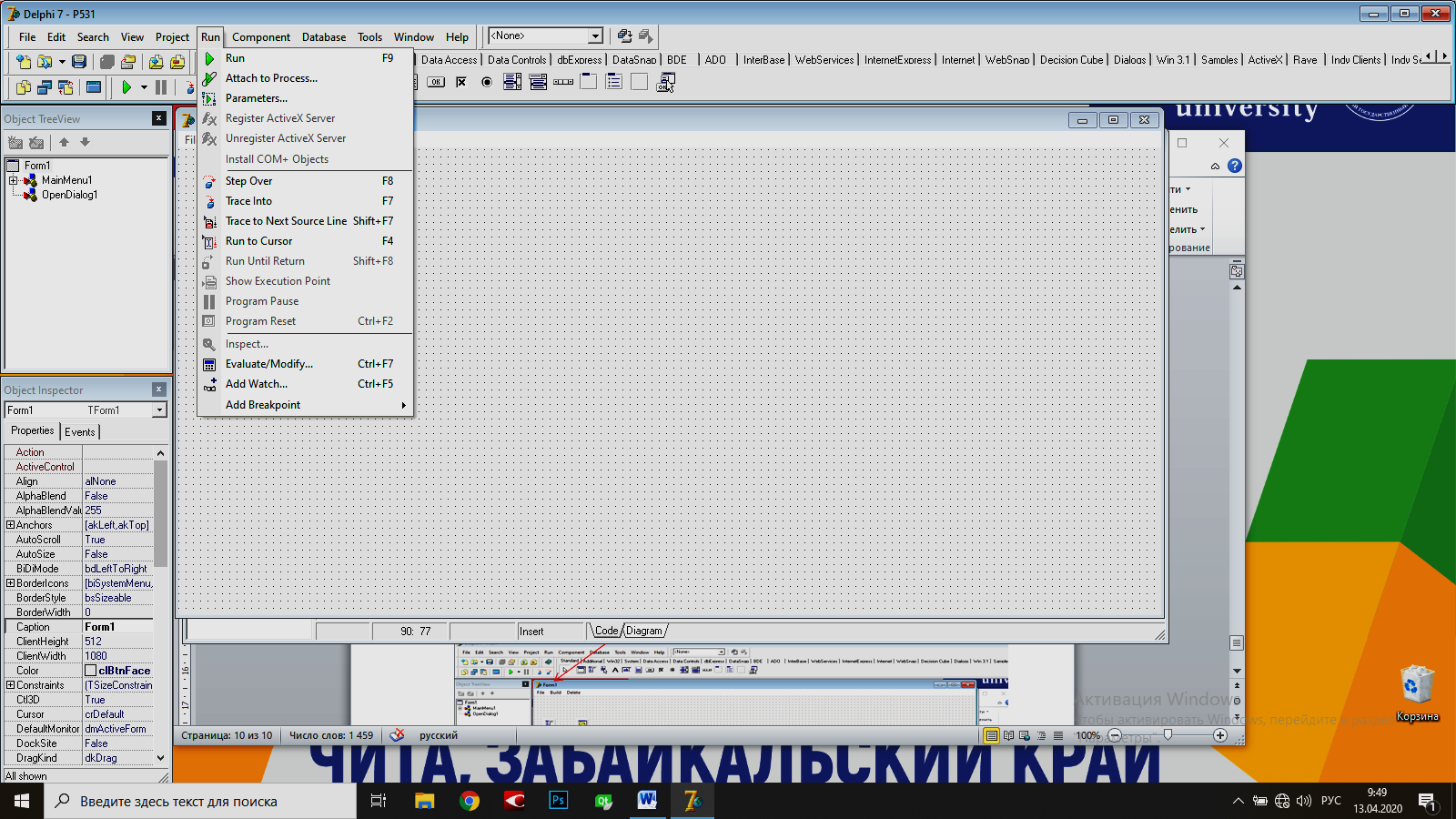


или перешли в окно формы

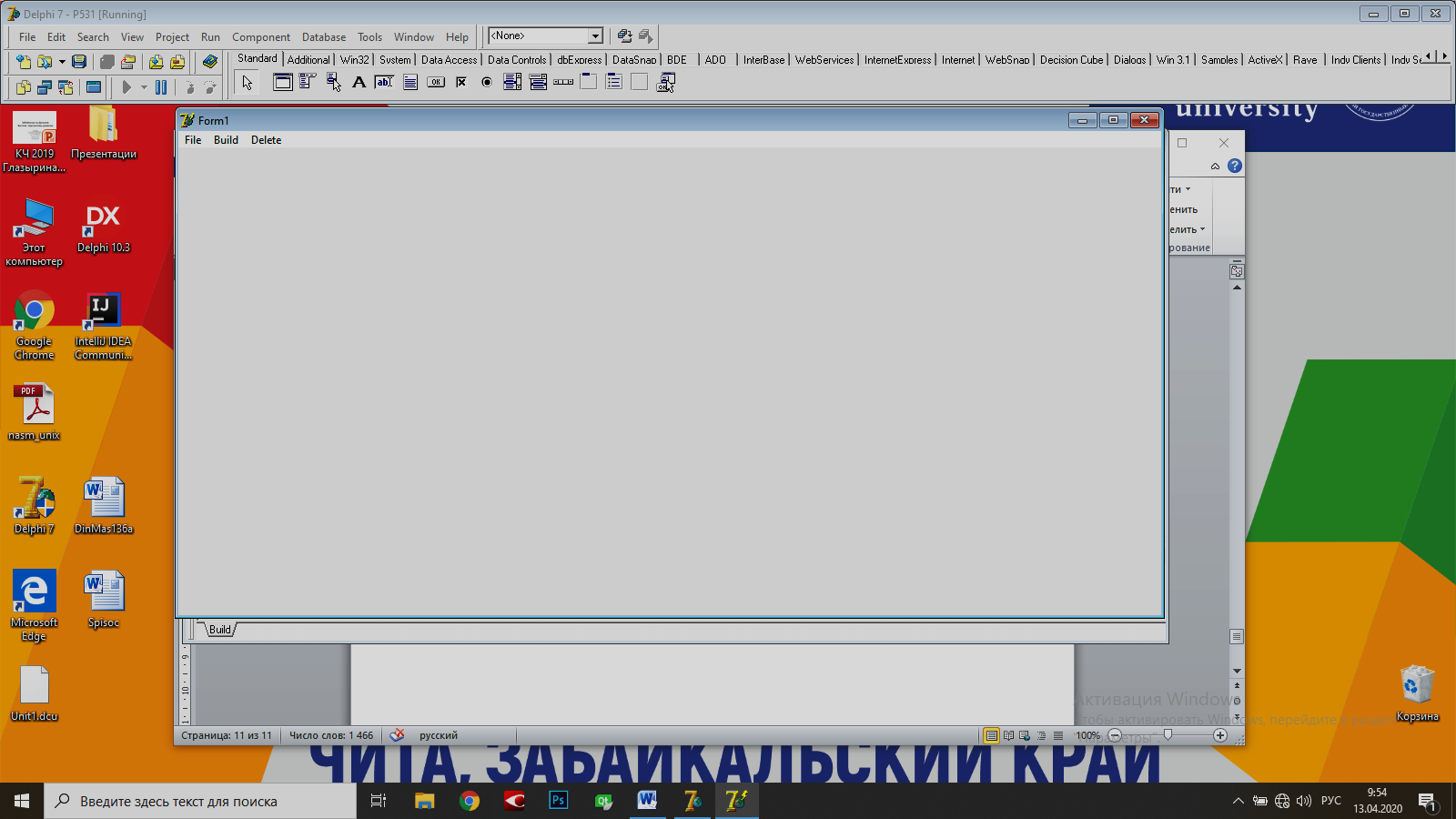


Выполним команду Run|Run главного меню или используем « горячую «

клавишу F9.

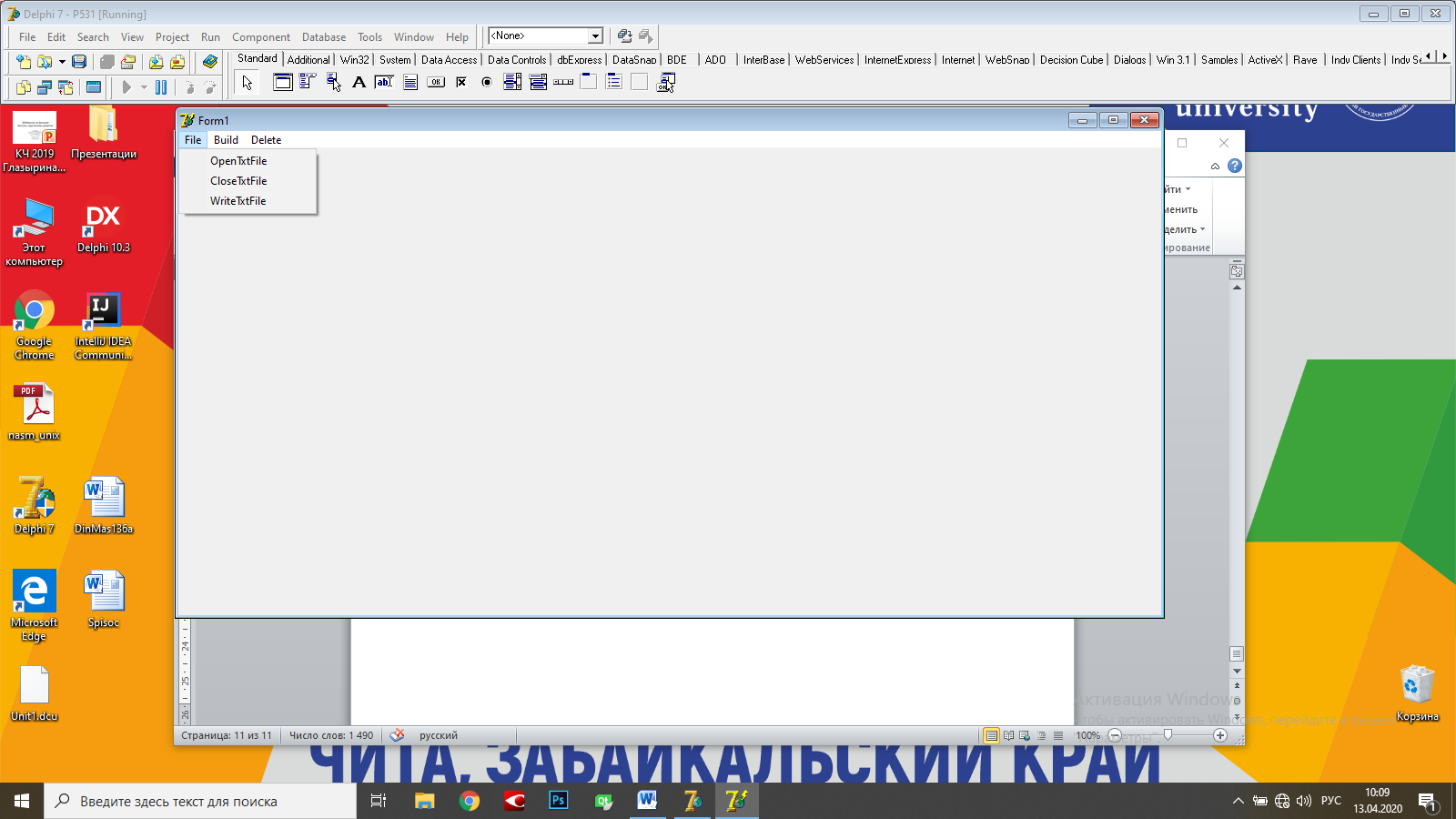


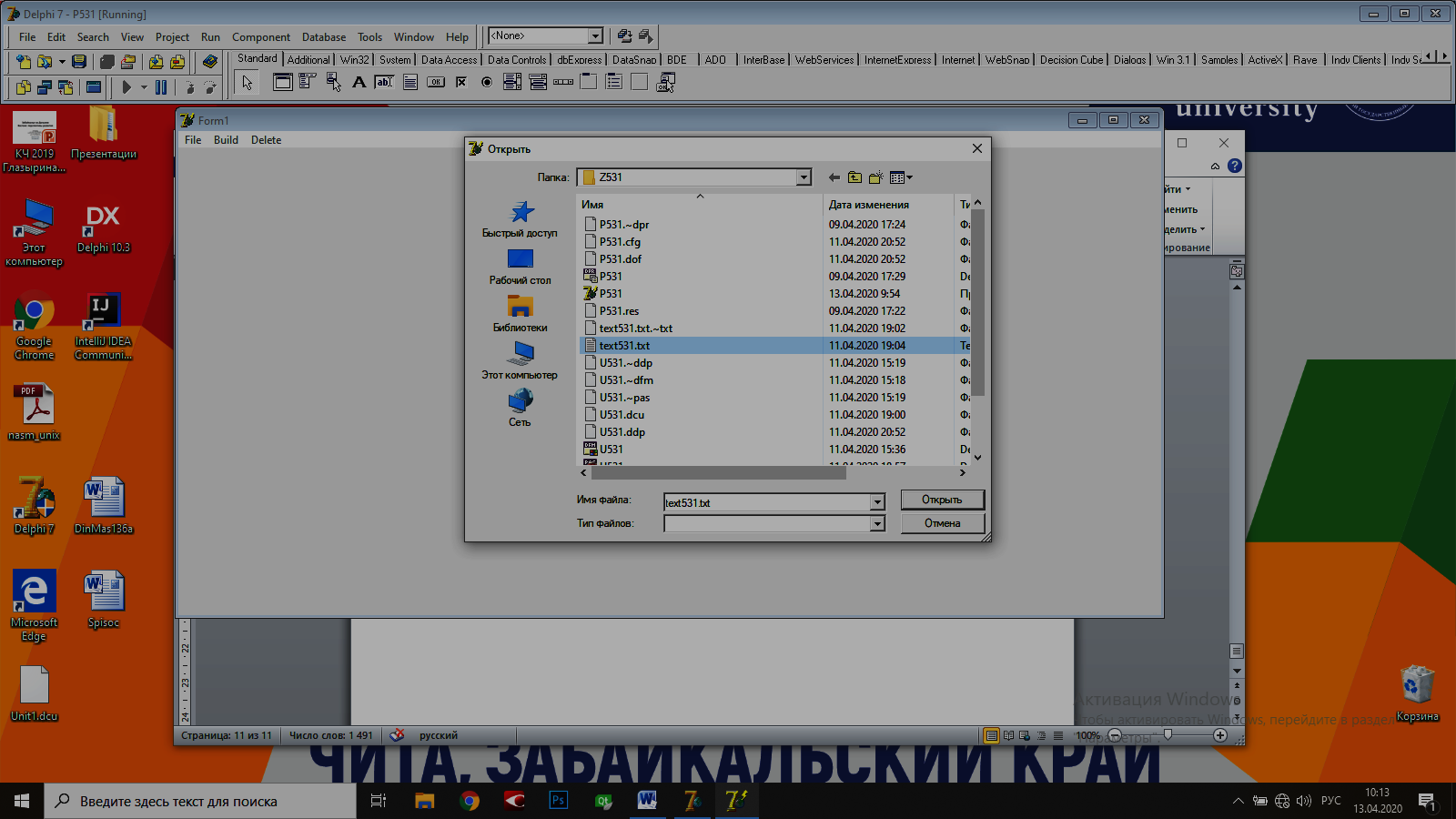
Получим окно выполняемой программы



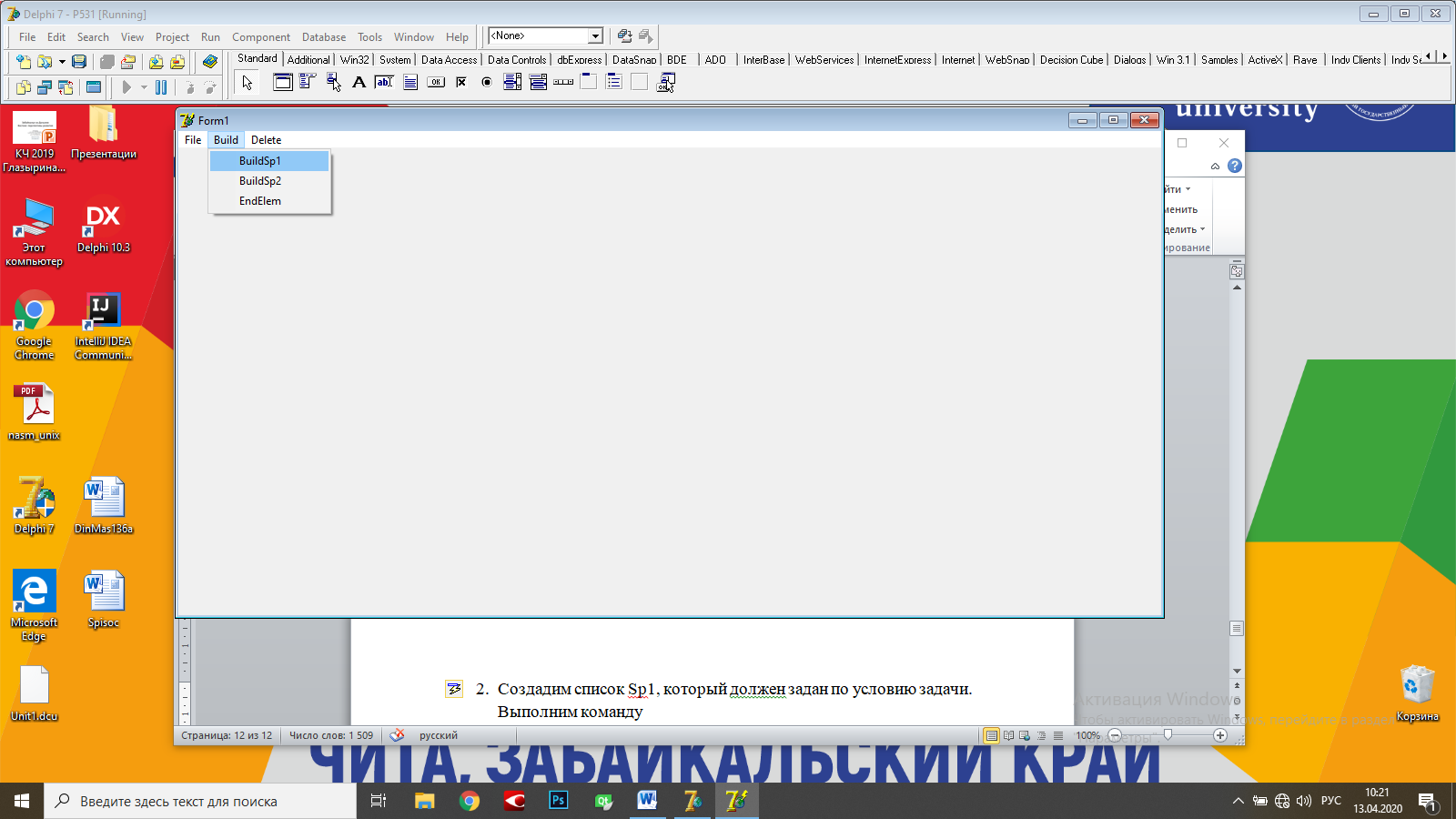
Это окно имеет строку заголовка и строку меню выполняемого приложения. Проверим работу созданных ранее обработчиков событий.

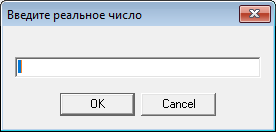
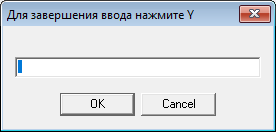
1. Откроем текстовый файл:

 получим окно, в котором выберем открываемый файл



1. Создадим список Sp1, который должен задан по условию задачи. Выполним команду Build|BuildSp1

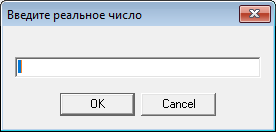
 последовательно будут появляться окна:

 и 

При вводе реального числа в качестве разделителя целой и дробной частей используется символ ‘ **,** ‘ , а не ‘ **.**’ Для продолжения ввода можно нажимать на любую клавишу, кроме **Y**.

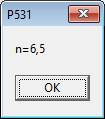
1. После заполнения Sp1 выведем его в текстовый файл командой:

File|WriteTxtFile. Появится окно, в котором мы выберем список, выводимый в текстовый файл. Для Sp1 введём 1.



1. Для просмотра текстового файла нужно:
2. Закрыть текстовый файл, команда File|ClosetxtFile.
3. Перейти в папку, в которой находятся файлы данного приложения. Открыть файл txt531.txt
4. Для дальнейшей работы снова откроем текстовый файл. См. пункт 1.
5. Создадим второй список, команда Build|BuildSp2.
6. Проверим работу функции EndElemSp командой Build|EndElem

Получим окно, в которое выведено значение последнего элем. списка:

 - всё правильно.

1. Выведем второй список в текстовый файл File|WriteTxtFile.
2. Закроем текстовый файл File|ClosetxtFile.
3. Удалим первый список: Delete|DeleteSp1.
4. Удалим первый список: Delete|DeleteSp2.
5. Закрываем приложение.
6. Просматриваем содержимое текстового файла. Убеждаемся в правильной работе приложения.

