# Двусявязный список

Разберём работу с двусвязанным списком. Нужно решить задачу (№533 из задачника):

Даны натуральное число **n**, действительные числа **x1, x2, …,xn**. Вычислить

**(x1+x2 +2xn) + (x2+x3 +2xn-1) + (x3+x4 +2xn-2) + … (xn-1+xn +2x2) .**

Если числа хранятся в списке, то этот список отличается от использованного в предыдущей задаче. В том списке узлы имели только один указатель – на следующий узел, т.е. в этом списке удобно передвигаться только в одном направлении, от головы списка к хвосту. Из условия задачи ясно, что нам придётся двигаться по списку в **двух** направлениях: от головы до хвоста и от хвоста к голове.

Реализовать такой список можно, изменить структуру узла: добавить ещё один указатель, который будет указывать на узел, предшествующий данному.

Графически это выглядит так:

pred x next

где **pred -** указатель напредыдущий узел, а **next** на следующий. Х- данные, которые хранятся в узле (это не обязательно простой тип, как в данной задаче, часто это поле типа записи).

Список, составленный из узлов выше приведённого типа называется двусвязанным списком.

Реализовать тип узла и тип указателя очень просто, в интерфейсной части модуля, который будет содержать типы и процедуры, необходимые для решения задачи, напишем:

unit UnitZ533;

interface

uses SysUtils, Dialogs;

type

PUzel = ^Zp2;

Zp2 = record

x: real;

next: PUzel; // указатель на следующий узел

**pred: PUzel;** // указатель на предыдущий узел

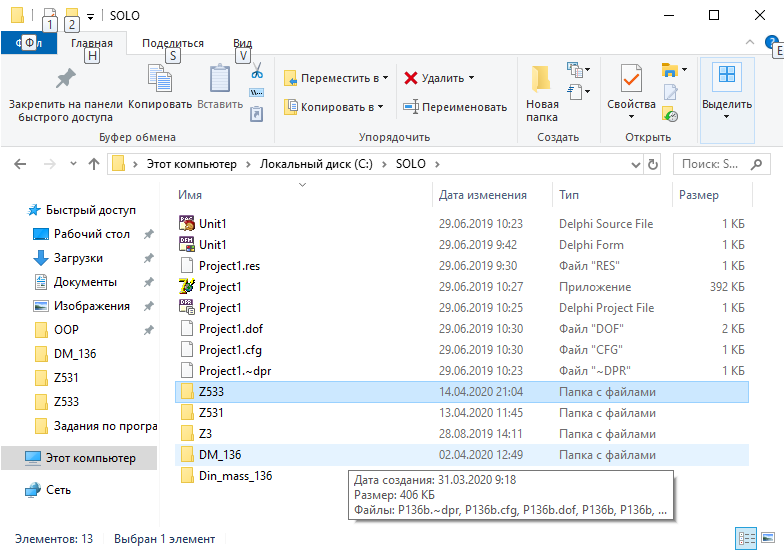
end;

FZap = file of real;

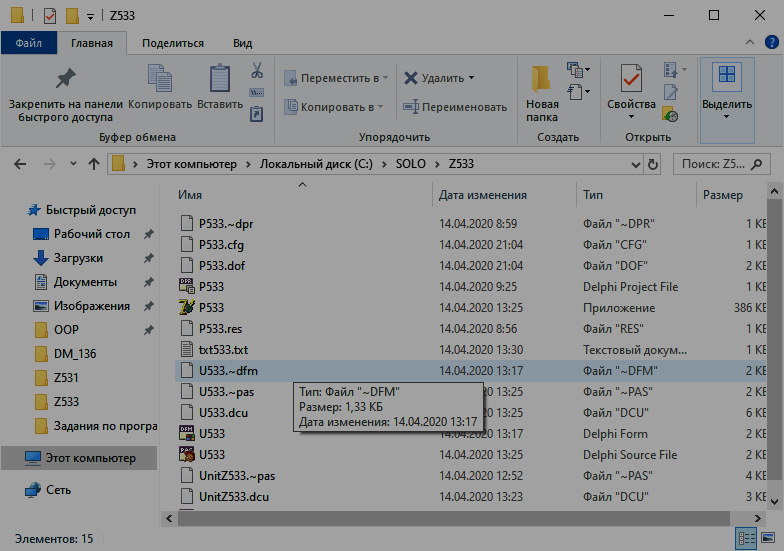
Кстати, если сравнить тексты UnitZ533 и UnitZ531, они отличаются только выделенной строчкой. Это, надеюсь, подсказывает как увеличить скорость написания модуля UnitZ533.

Для создания приложения определимся с последовательностью действий;

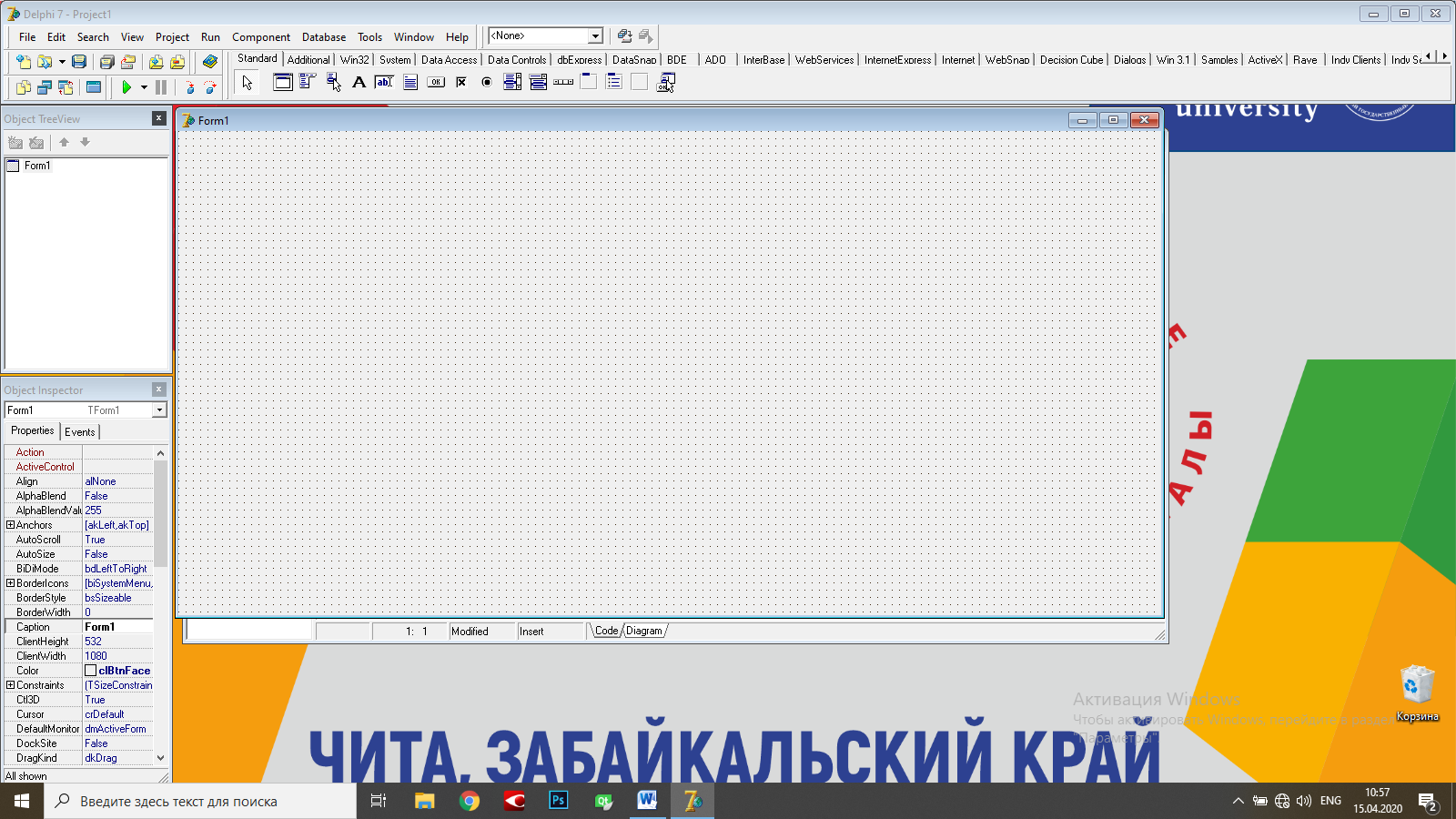
1. Создать папку для проекта задачи Z533.



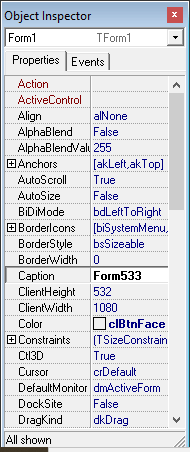
1. Открыть папку Z533 и в ней создать текстовый файл для хранения результатов запуска создаваемого проекта

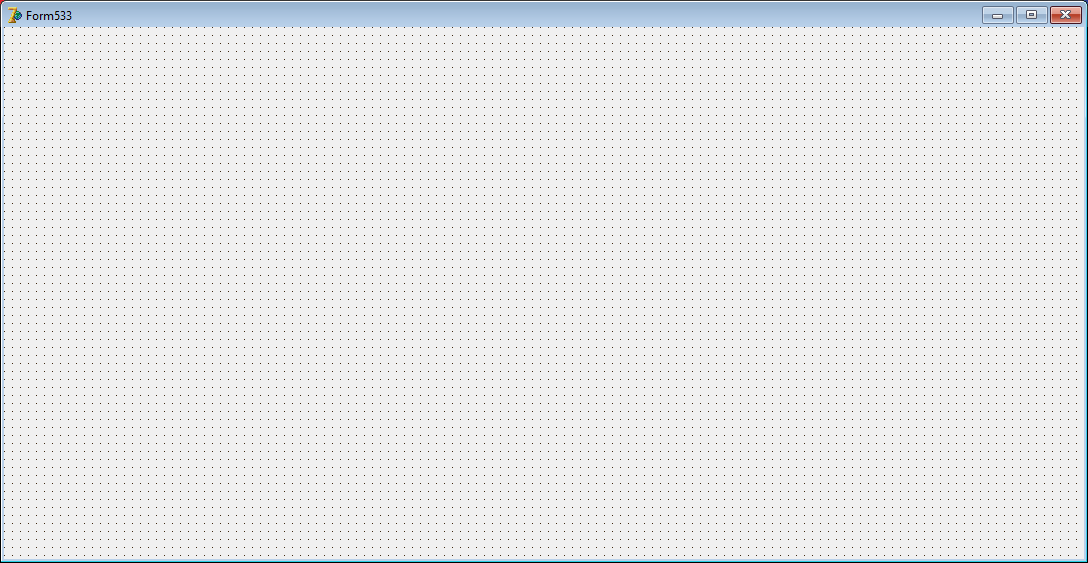


1. Запустить Delphi

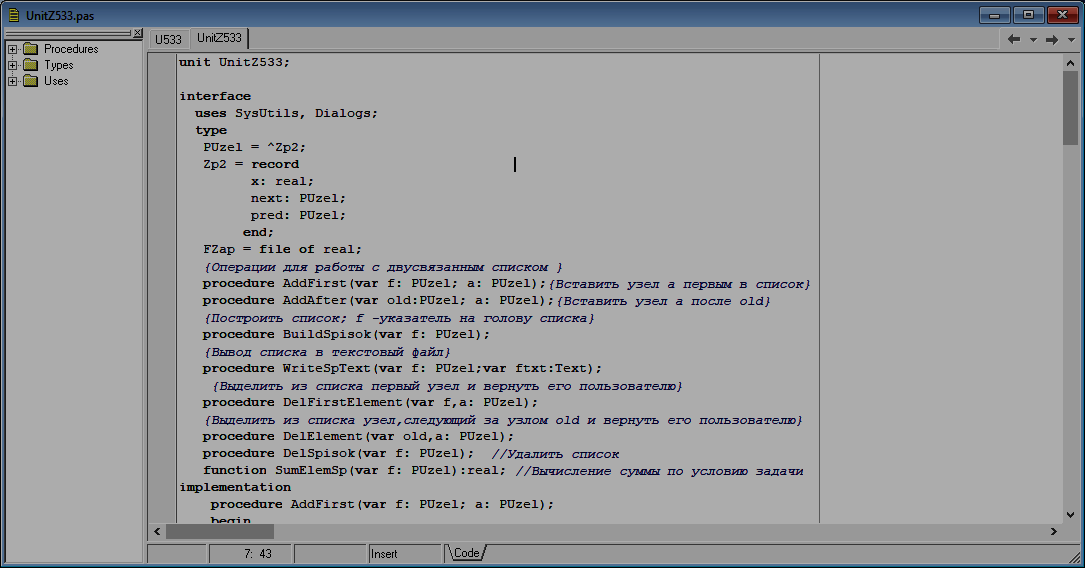


1. Изменим название формы с **Form1** на **Form533.** Для этого перейдём в окно Objekt Inspektor и на странице Propertits свойству Caption напишем Form533

 Название окна формы изменится на Form533



1. Сохраним проект и модуль формы под именами P533, U533.
2. Операции для реализации проекта:
3. Создать тип узла и тип указателя на узел.
4. Процедура создания списка.
5. Процедура удаления списка.
6. Функция вычисления суммы по заданному алгоритму.
7. Создать заготовку модуля, в которой будут реализованы операции 6).



Наполним заготовку следующим текстом:

**unit** UnitZ533;

**interface**

**uses** SysUtils, Dialogs;

**type**

PUzel = ^Zp2;

Zp2 = **record**

x: real;

next: PUzel;

pred: PUzel;

end;

FZap = file of real;

{Операции для работы с двусвязанным 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);

{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 пользователю}

**procedure** DelElement(var old,a: PUzel);

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

**function** SumElemSp(var f: PUzel):real; //Вычисление суммы по условию задачи

**implementation**

procedure AddFirst(var f: PUzel; a: PUzel);

begin

a^.next := f;

if f<> nil then f^.pred := a;

f:=a;

end;

Графически схему работы процедуры **AddFirst** можно представить:

Дан узел **а** у него поля next и pred равны nil.

**а**

x

Список может быть пустым: f = nil. Тогда оператор a^.next := f; не меняет поле next узла **а.** Второй оператор  if f<> nil then f^.pred := a; не срабатывает т.к. f = nil. Третий оператор f:=a; завершает операцию вставки, перемещая указатель f на вставляемый узел, получаем список:

**а**

x

**f**

Во втором случае список содержит хотя бы один узел:

**f**

x

Тогда оператор a^.next := f; свяжет узел **а** с головным элементом списка

**а**

x

x

**f**

Второй оператор  if f<> nil then f^.pred := a; свяжет головной элемент **f** с **а:**

**а**

x

x

**f**

Остаётся перевести указатель **f** на новую голову списка f:=a;

**а**

x

x

**f**

procedure AddAfter(var old:PUzel; a: PUzel);

begin

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

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

{Если узел **old** не последний в списке, то связать следующий за **old** узел с **а**}

if a^.next <> nil then a^.next^.pred := a;

a^.pred := old; // связываем узел **а** с узлом **old**

end;

Создайте графическую схему работы процедуры AddAfter, используя комментарий для каждого оператора AddAfter.

**procedure** BuildSpisok(var f: PUzel);

**var**

a,d :PUzel;

ch: char;

**begin**

f:= nil;

**repeat**

new(a);

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

a^.next := nil;

if (f = nil) then

begin

AddFirst(f,a);

d:= f;

end

else

begin

AddAfter(d,a);

d := a;

end;

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

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

**end**;

Как видим, текст процедуры BuildSpisok полностью совпадает с соответствующим текстом для односвязанного списка UnitZ531. Т.е. создавая

UnitZ533, эту процедуру можно скопировать из UnitZ531.

{Вывод списка в текстовый файл}

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

**var**

p: PUzel; //текущий указатель, перемещается от головы до хвоста списка

s: string; //строка, которая будет выводиться в текстовый файл

y: real; //значение поля данных узла

begin

p:= f; // текущий указатель указывает на голову списка

while not(p = nil) do

begin

y:= p^.x; //получить значение поля данных текущего узла

s:=Format('%-7.3g',[y]); //Создать отформатированную строку

write(ftxt, s);

p:= p^.next; //перевести указатель на следующий узел

end;

end;

В процедуре WriteSpText используется функция Format, которая объявлена в модуле SysUtils *(стр.196 из книги: А.Я.Архангельский, Язык Pascal и основа программирования в Delphi. Учебное пособие – М.: ООО «Бином-Пресс», 2004г.-496с.)*

Вывод можно организовать так как это реализовано в проекте для односвязанного списка.

**function** SumElemSp(var f: PUzel):real;

var

p,d: PUzel;

s: real;

**begin**

p:= f;

{Определение указателя на последний элемент списка}

while not(p = nil) do

begin

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

p:=p^.next;

end;

{Вычисление суммы элементов списка по заданному правилу}

p:= f;

s:= 0;

while not(p^.next = nil) do

begin

s:= s + (p^.x + p^.next^.x +2\*d^.x);

p:= p^.next;

d:= d^.pred;

end;

result := s;

end;

procedure DelFirstElement(var f,a: PUzel);

begin

a := f; // сохраняем указатель на голову списка

f := f^.next; a^.next := nil; //отсоединяем первый элемент списка

if f<>nil then f^.pred := nil; //в головном узле списка поле pred = nil

end;

Алгоритм следующей процедуры рассмотреть самостоятельно.

procedure DelElement(var old,a: PUzel);

begin

if (old^.next = nil) then a:= nil //old последний узел в списке

else

if (old^.next^.next = nil) then //old предпоследний узел в списке

begin

a := old^.next;

a^.pred:= nil;

old^.next:= nil;

end

else

begin //за old не менее двух узлов в списке

a := old^.next;

old^.next := a^.next;

old^.next^.pred:= old;

a^.next := nil;

a^.pred:= nil;

end;

end;

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

var

a: PUzel;

begin

while (f <> nil) do

begin

DelFirstElement(f,a);

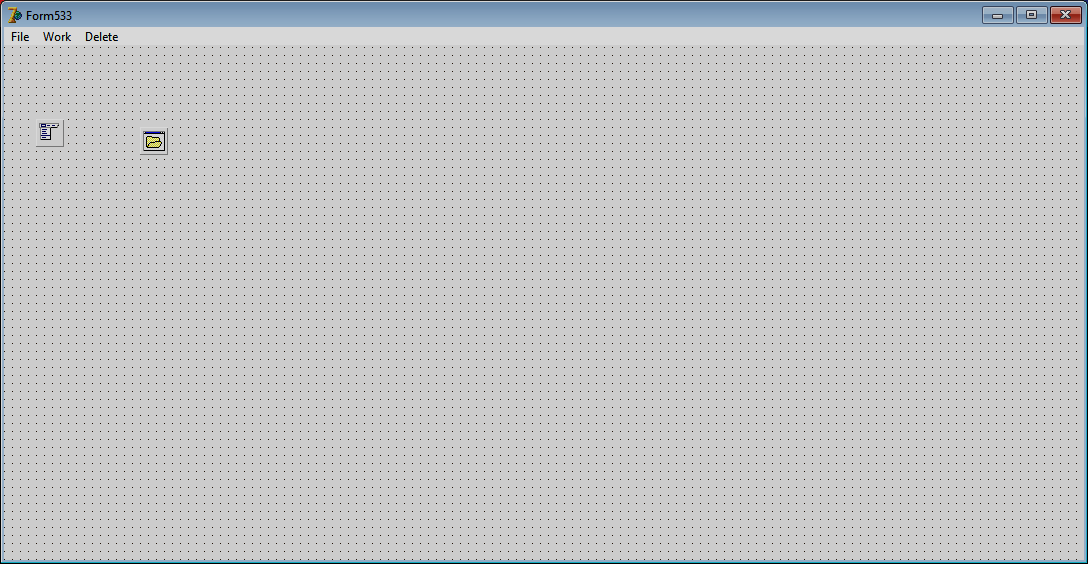
Dispose(a);

end;

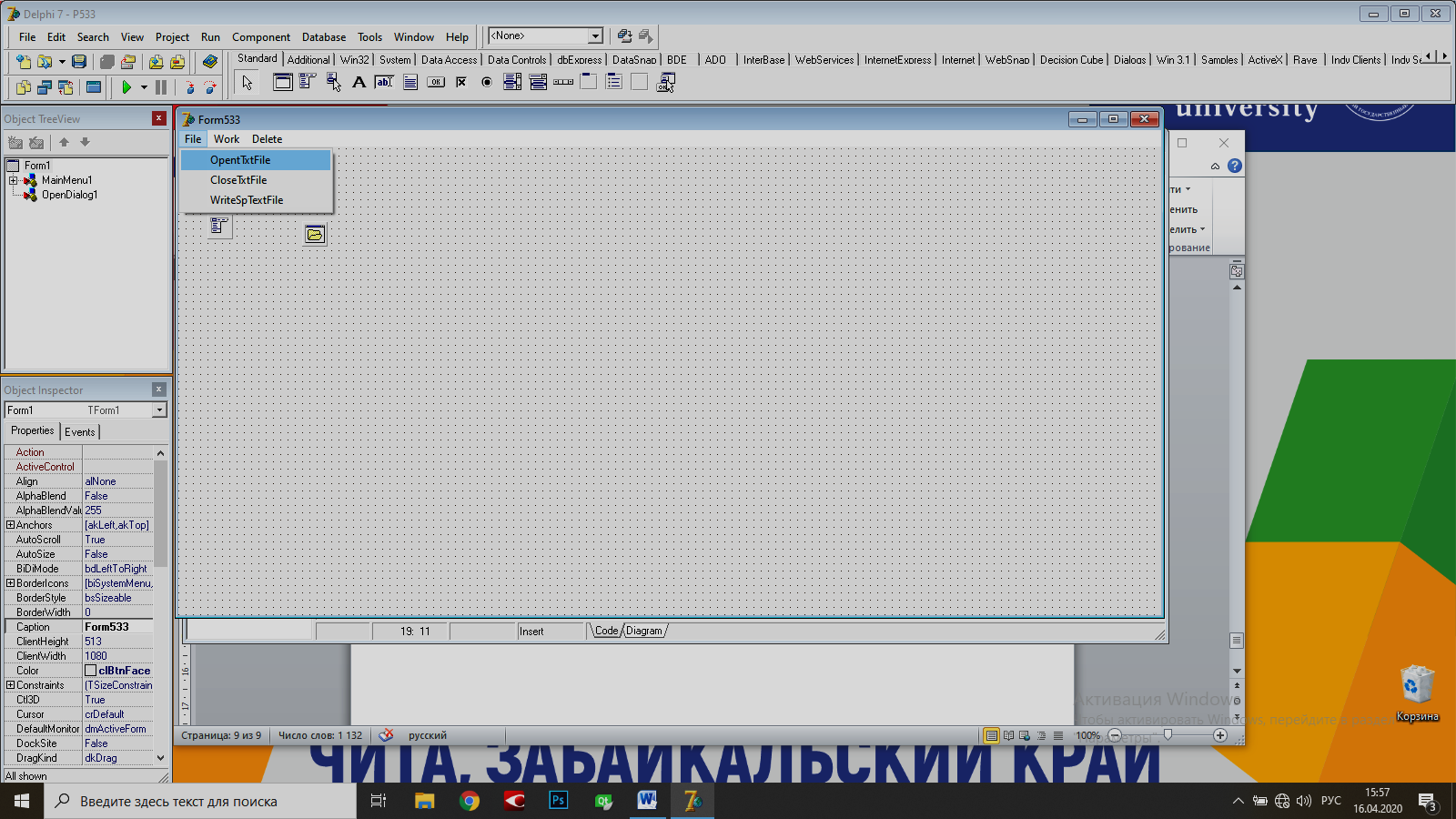
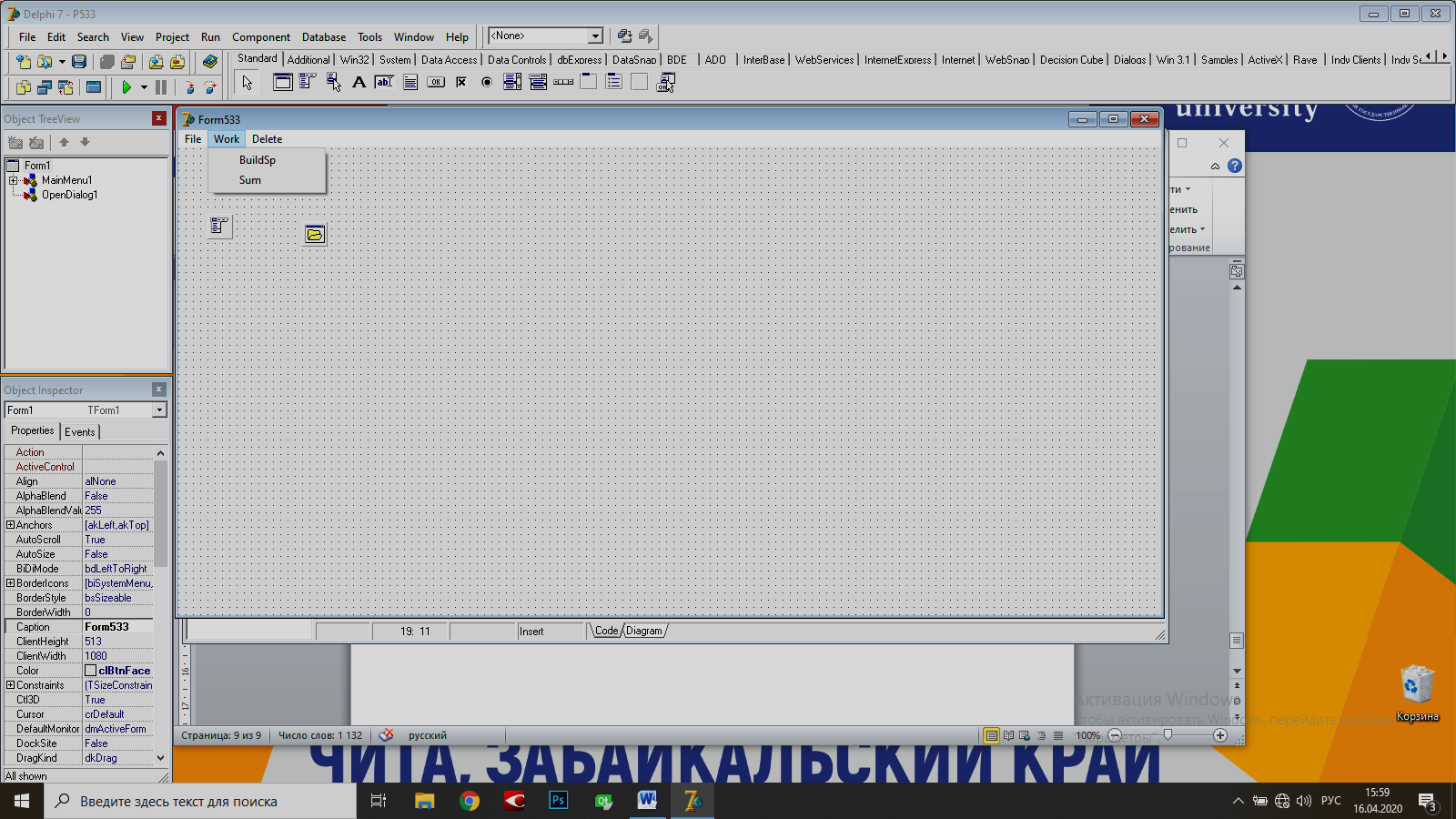
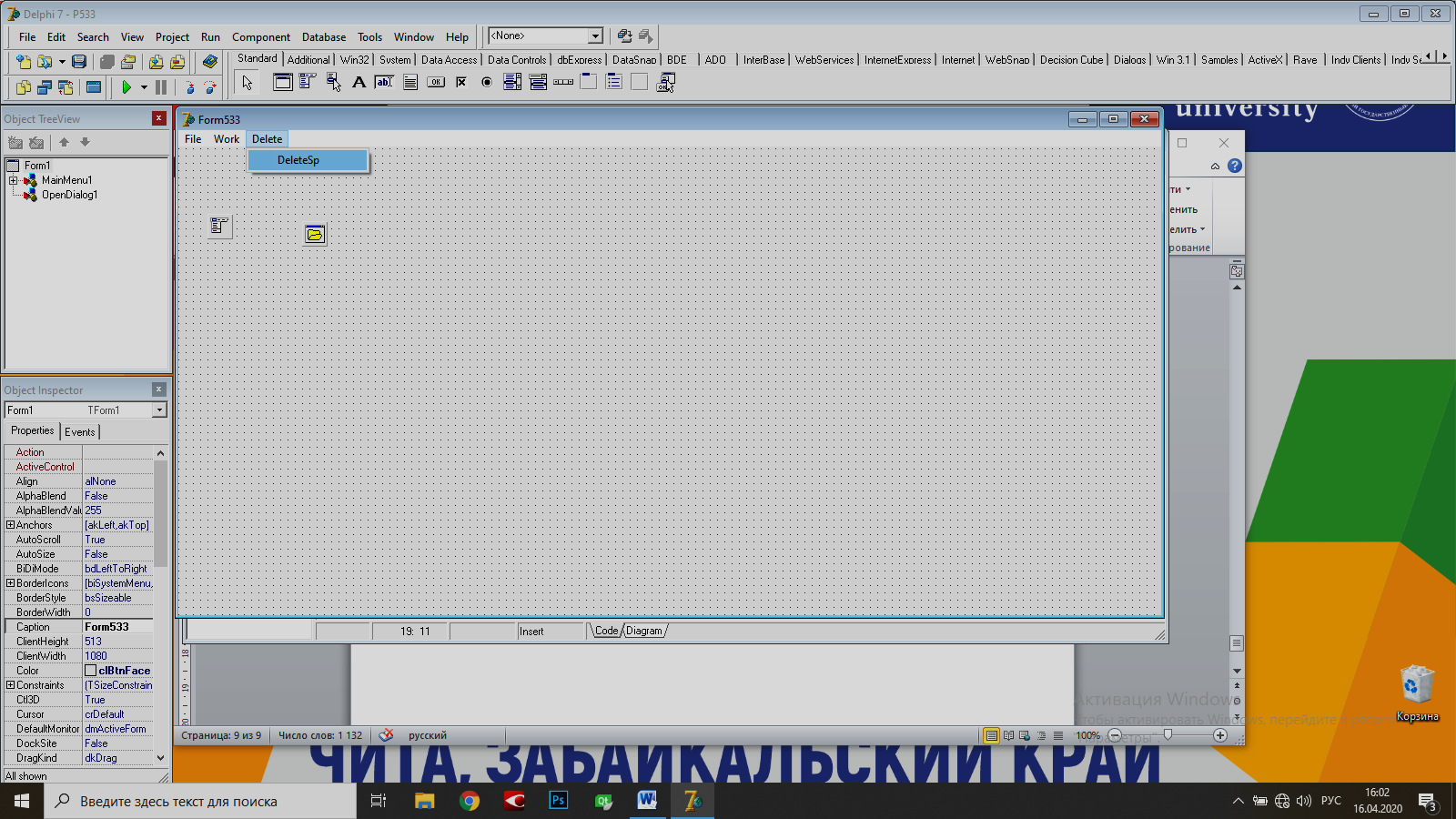
end;

end.

Итак модуль создан. Оформим центр управления проектом. Переходим в окно Form533 и переносим туда компоненты **MainMenu** и **OpenDialog,** которые находятся на страницах Standard и Dialog.



Для решения задачи в меню нам хватит три элемента: File, Work и Delete, каждый из которых будет содержать свои элементы:

   
Остаётся написать обработчики событий для созданных элементов меню, но вначале в окне редактора модуля U533 в uses предложение имя созданного

модуля UnitZ533:

uses

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

Dialogs, UnitZ533, Menus;

Затем в секцию var добавим глобальные параметры:

var

Form1: TForm1;

Sp1: PUzel; //указатель на список

ftxt: text; //файловая переменная

Обработчики событий создаём также как в предыдущем проекте, т.е. в окне Form533 последовательно щёлкаем ЛКМ на каждом из элементов меню и в интерфейсной части модуля U533 получим заголовки обработчиков событий, а в секции реализации заготовки процедур для написания кода обработчиков:

procedure OpentTxtFile1Click(Sender: TObject);

procedure CloseTxtFile1Click(Sender: TObject);

procedure WriteSpTextFile1Click(Sender: TObject);

procedure BuildSp1Click(Sender: TObject);

procedure Sum1Click(Sender: TObject);

procedure DeleteSp1Click(Sender: TObject);

…

implementation

{$R \*.dfm}

procedure TForm1.OpentTxtFile1Click(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.WriteSpTextFile1Click(Sender: TObject);

begin

writeln(ftxt,'Содержимое двусвязанного списка ');

WriteSpText(Sp1,ftxt);

writeln(ftxt,' ')

end;

procedure TForm1.BuildSp1Click(Sender: TObject);

begin

BuildSpisok(Sp1);

end;

procedure TForm1.Sum1Click(Sender: TObject);

begin

writeln(ftxt, ' ');

write(ftxt,'Искомая сумма равна = ');

writeln( ftxt,SumElemSp(Sp1):7:2);

end;

procedure TForm1.DeleteSp1Click(Sender: TObject);

begin

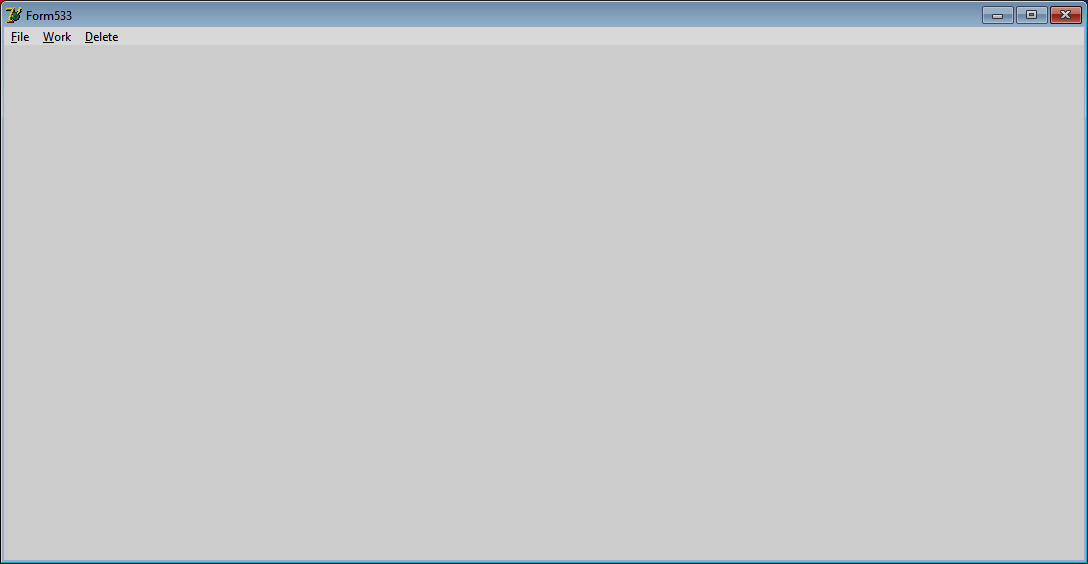
DelSpisok(Sp1);

end;

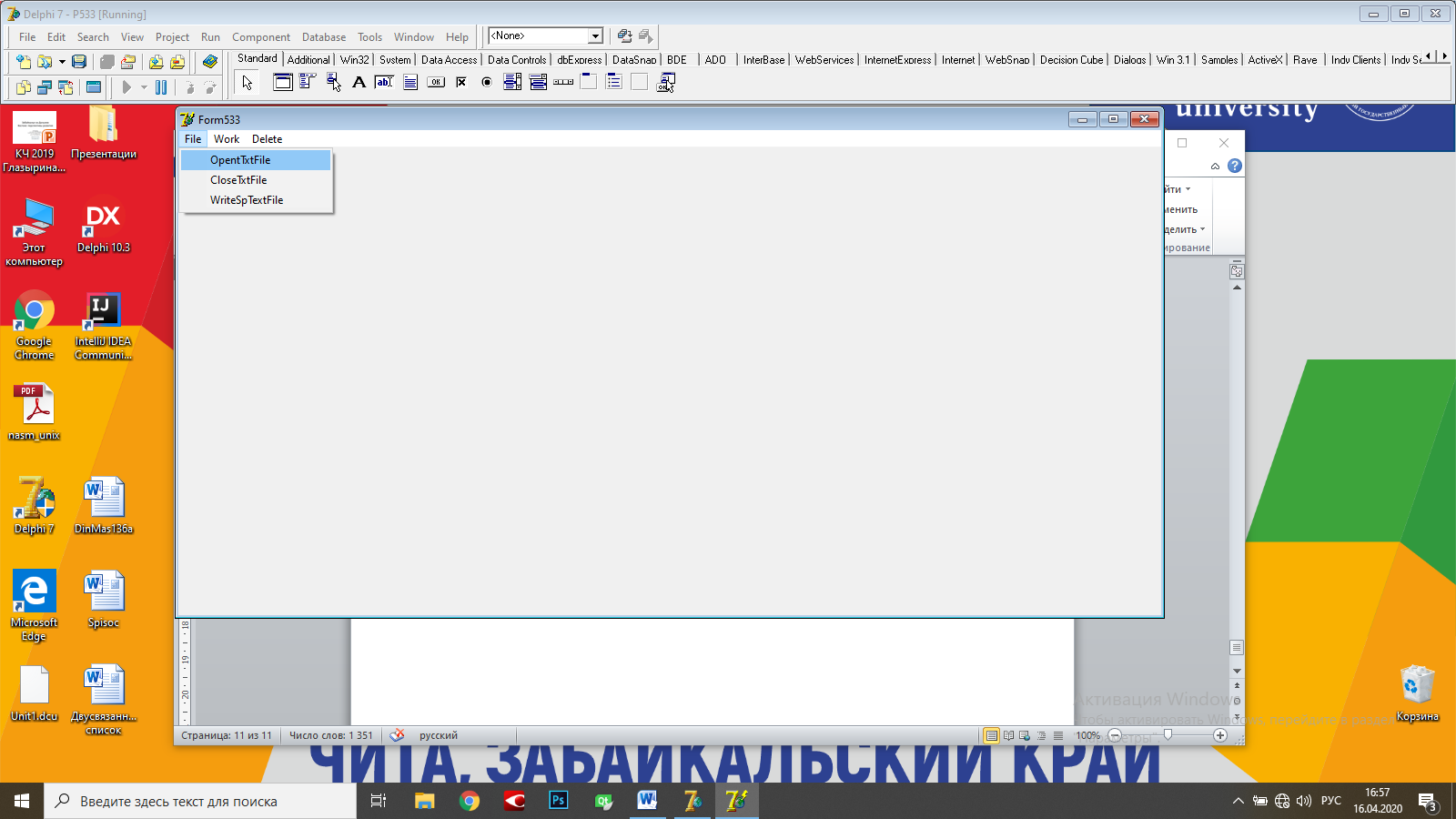
end.

Как видите, если создан модуль , в котором написаны все подпрограммы для решения задачи, то создать обработчики событий не представляет труда.

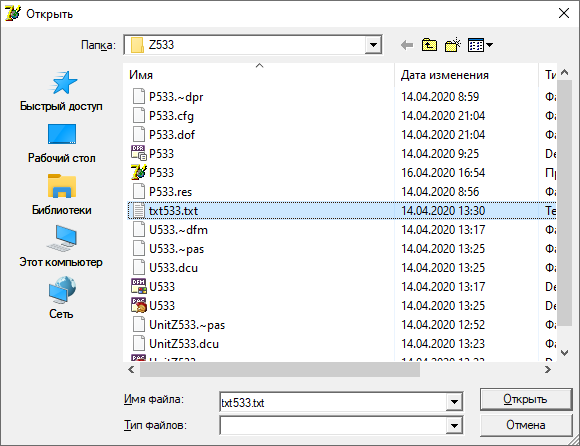
Когда все подготовительные операции выполнены, можно запускать проект на выполнение: команда Run|Run, или f9.



Теперь последовательно выполним следующие операции:

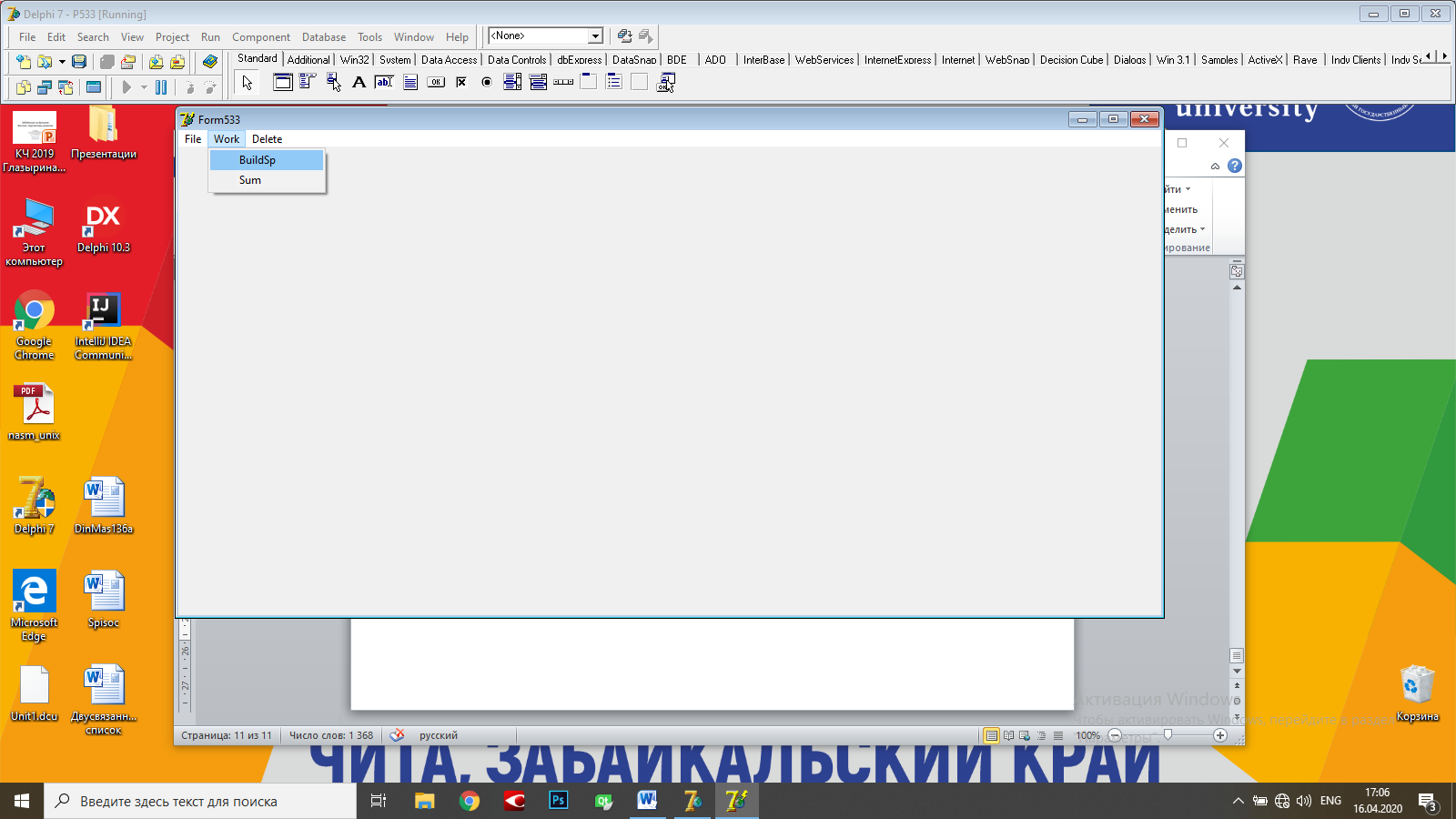


Получим

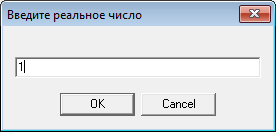
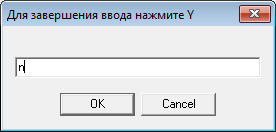


т.е. найдём нужную папку, в ней текстовый файл и откроем его.

Следующая операция – создание списка: Work|BuildSp



После выполнения команды циклически будут появляться окна для заполнения списка:

В первом окне вводим реальное число. Учтите, что **целая и дробная части разделяются запятой**. Если нажмёте на точку, или на букву, будет аварийный останов приложения. Пока мы не готовы перехватывать такие ситуации. Для досрочного прекращения работы приложения выполните команду: **Project|Program Reset** или Ctrl+F2

Во втором окне для продолжения ввода можно нажать любую клавишу, кроме Y.

Для проверки работы функции SumElemSp введём пять чисел: 1, 2, 3, 4, 5.

Запишем значения узлов списка в текстовый файл: **File|WriteSpTextFile**

Вычислим значение суммы элементов списка по заданному алгоритму:

**Work|Sum**

Затем завершим работу приложения:

1. **File|CloseTxtFile**
2. **Delete|DeleteSp эта операция обязательна.**
3. Выйдем из приложения ×
4. Откроем текстовый файл и посмотрим результат.



Содержимое текстового файла совпадает с ожидаемым результатом.