МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ
Колледж Пятигорского института (филиал) СКФУ

ОП. 06 ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ЛАБОРАТОРНЫХ ЗАНЯТИЙ

Специальности СПО

09.02.01 Компьютерные системы и комплексы

Квалификация: специалист по компьютерным системам

Пятигорск 2023

Методические указания для лабораторных занятий по дисциплине ОП.06 Основы алгоритмизации и программирования составлены в соответствии с требованиями ФГОС СПО. Предназначены для студентов, обучающихся по специальности 09.02.01 Компьютерные системы и комплексы.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Программа Основы алгоритмизации и программирования предусматривает изучение языков программирования.

При изучении предмета следует соблюдать единство терминологии и обозначения в соответствии с действующими стандартами, Международной системной единицы (СИ).

В результате освоения учебной дисциплины обучающийся должен

уметь

- разрабатывать и анализировать алгоритмы для решения поставленных задач;
- определять сложность алгоритмов;
- реализовывать типовые алгоритмы в виде программ на актуальных языках программирования;
- использовать средства проектирования для создания и графического отображения алгоритмов;
- оформлять код программ в соответствии со стандартом кодирования;
- выполнять проверку, отладку кода программы.

знать

- понятие алгоритмизации, свойства алгоритмов, общие принципы построения алгоритмов, основные алгоритмические конструкции;
- классификация языков программирования;
- понятие системы программирования;
- основные элементы языка, структура программы;
- методы реализации типовых алгоритмов;
- операторы и операции, управляющие структуры, структуры данных, классы памяти;
- понятие подпрограммы, библиотеки подпрограмм;
- объектно-ориентированная модель программирования, основные принципы объектно-ориентированного программирования на примере алгоритмического языка: понятие классов и объектов, их свойств и методов, инкапсуляции и полиморфизма, наследования и переопределения.

Лабораторное занятие 1. Составление и оформление блок-схем простых алгоритмов.

Тема 1. Понятие алгоритма. Способы описания алгоритма. Базовые алгоритмические конструкции.

Цель: изучить способы построения блок-схем и написание программного кода.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

Блок-схемой будем называть такое графическое представление алгоритма, когда отдельные действия (или команды) представляются в виде геометрических фигур – *блоков*. Внутри блоков указывается информация о действиях, подлежащих выполнению. Связь между блоками изображают с помощью линий, называемых *линиями связи*, обозначающих передачу управления.

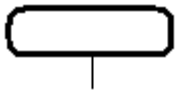
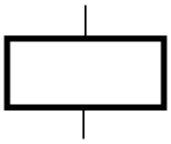
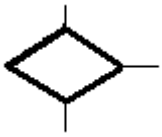
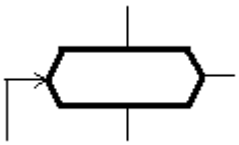
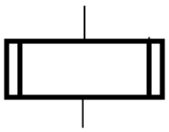
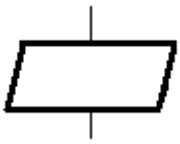
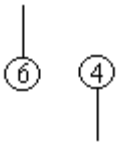
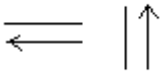
Существует Государственный стандарт, определяющий правила создания блок-схем. Конфигурация блоков, а также порядок графического оформления блок-схем регламентированы ГОСТ 19.701-90 "Схемы алгоритмов и программ".

Правила составления блок-схем:

1. Каждая блок-схема должна иметь блок «*Начало*» и один блок «*Конец*».
2. «*Начало*» должно быть соединено с блоком «*Конец*» линиями потока по каждой из имеющихся на блок-схеме ветвей.
3. В блок-схеме не должно быть блоков, кроме блока «*Конец*», из которых не выходит линия потока, равно как и блоков, из которых управление передается «в никуда».
4. Блоки должны быть пронумерованы. *Нумерация* блоков осуществляется сверху вниз и слева направо, номер блока ставится вверху слева, в разрыве его начертания.
5. Блоки связываются между собой линиями потока, определяющими последовательность выполнения блоков. Линии потоков должны идти параллельно границам листа. *Если линии идут* ***справа налево*** *или* ***снизу вверх****, то стрелки в конце линии обязательны*, в противном случае их можно не ставить.
6. По отношению к блокам линии могут быть *входящими* и *выходящими*. Одна и та же линия потока является выходящей для одного блока и входящей для другого.
7. От блока «*Начало*» в отличие от всех остальных блоков линия потока только выходит, так как этот блок – первый в блок-схеме.
8. Блок «*Конец*» имеет только вход, так как это последний блок в блок-схеме.
9. Для простоты чтения желательно, чтобы линия потока входила в блок «Процесс» сверху, а выходила снизу.
10. Чтобы не загромождать блок-схему сложными пересекающимися линиями, линии потока можно разрывать. При этом в месте разрыва ставятся *соединители*, внутри которых указываются номера соединяемых

блоков. В блок-схеме не должно быть разрывов, не помеченных соединителями.

11. Чтобы не загромождать блок, можно информацию о данных, об обозначениях переменных и т.п. размещать в *комментариях* к блоку.

Название блока	**Обозначение блока**	**Назначение блока**
1	**2**	**3**
Терминатор		Начало/Конец программы или подпрограммы
Процесс		Обработка данных (вычислительное действие или последовательность вычислительных действий)
Решение		Ветвление, выбор, проверка условия. В блоке указывается условие или вопрос, который определяет дальнейшее направление выполнения алгоритма
Подготовка		Заголовок счетного цикла
Предопределенный процесс		Обращение к процедуре
Данные		Ввод/Вывод данных
Соединитель		Маркировка разрыва линии потока
Комментарий		Используется для размещения пояснений к действиям
Горизонтальные и вертикальные потоки		Линии связей между блоками, направление потоков

Задание:

1) Создать алгоритм с помощью блок-схем и написать программу для вычисления суммы двух чисел.

2) Создать алгоритм с помощью блок-схем и написать программу для вычисления разности двух чисел.

3) Создать алгоритм с помощью блок-схем и написать программу для вычисления площади квадрата.

4) Создать алгоритм с помощью блок-схем и написать программу для вычисления объема куба.

5) Создать алгоритм с помощью блок-схем и написать программу для сравнения двух чисел между собой, для определения большего.

6) Создать алгоритм с помощью блок-схем и написать программу для сравнения двух чисел между собой, для определения меньшего.

Вопросы для самоконтроля

1. Понятие алгоритма.
2. Свойства и виды алгоритмов.
3. Способы описания алгоритмов: псевдокоды.
4. Правила составления блок-схем.
5. Стандарты графического оформления алгоритмов.
6. Линейные алгоритмы.
7. Разветвляющиеся алгоритмы.
8. Циклические алгоритмы.
9. Критерии «хорошего» алгоритма.

Лабораторное занятие 2. Составление и оформление блок-схем простых алгоритмов.

Тема 1. Понятие алгоритма. Способы описания алгоритма. Базовые алгоритмические конструкции.

Цель: изучить способы построения блок-схем и написание программного кода.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

Воспользоваться онлайн конструктором блок-схем, для выполнения лабораторного занятия.

Задание:

1) Создать алгоритм с помощью блок-схем и написать программу для определения четности или нечетности введенного числа.

2) Создать алгоритм с помощью блок-схем и написать программу для определения числа, является оно положительным или отрицательным.

3) Создать алгоритм с помощью блок-схем и написать программу для вычисления корней квадратного уравнения.

4) Создать алгоритм с помощью блок-схем и написать программу при вводе числа будет выдавать день недели, а в ином случае ошибку.

5) Создать алгоритм с помощью блок-схем и написать программу для сравнения трех чисел между собой, для определения большего.

6) Создать алгоритм с помощью блок-схем и написать программу для сравнения трех чисел между собой, для определения меньшего.

Вопросы для самоконтроля

1. Понятие алгоритма.
2. Свойства и виды алгоритмов.
3. Способы описания алгоритмов: псевдокоды.
4. Правила составления блок-схем.
5. Стандарты графического оформления алгоритмов.
6. Линейные алгоритмы.
7. Разветвляющиеся алгоритмы.
8. Циклические алгоритмы.
9. Критерии «хорошего» алгоритма.

Лабораторное занятие 3. Проектирование и оформление алгоритмов сортировки.

Тема 2. Основные методы и этапы проектирования алгоритмов. Эффективность и сложность алгоритма.

Цель: изучить алгоритмы сортировки одномерных массивов.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

Сортировка одномерных массивов.

В практике программиста часто случается так, что нужно разместить какие-либо данные в определенном порядке. В Паскале для таких случаев предусмотрена сортировка. Существует два основных алгоритма сортировки. Первый из них - **метод прямого выбора**. Ее смысл заключается в том, что за счет вложенности циклов каждый элемент массива сравнивается с остальными.

То есть, если у нас 10 чисел, то сначала первое из них будет сравниваться до тех пор, пока не будет найдено другой, например, большее его (если мы сортируем просто по возрастанию). Далее так же будет сравниваться 2, 3, 4 … элементы с последующими, но не с теми, которые уже отсортированы.

Прелесть этого вида сортировки заключается в том, что она очень проста для начинающего программиста, и все обычно начинают именно с неё. Давайте же рассмотрим пример. Пусть нам дан массив из 20 элементом и нам нужно отсортировать по убыванию.

```
Program SortMas;
var
i, j, k: integer;
mas: array[1..20] of integer;
Begin
        randomize;
        for i := 1 to 20 do mas[i] := random(100);
        writeln;
        writeln('массив до сортировки');
        for i := 1 to 20 do write(mas[ i], ' ');
        for i := 1 to 19 do
        for j := i + 1 to 20 do
        if mas[i] < mas[j] then
        begin
                k := mas[i];
                mas[i] := mas[j];
                mas[j] := k;
        end;
        writeln;
        writeln('массив после сортировки');
        for i := 1 to 20 do write(mas[ i], ' ');
End.
```

Обратите внимание на 2 первые строки после Begin. Здесь мы вызываем процедуру генерации случайных чисел. То есть просто заполняем наш массив числами от 1 до 100. Далее в цикле выводится для пользователя первоначальный массив.

Далее следует сам алгоритм сортировки, для новичков объясним, что это 2 цикла for, первый из них, в данной случае, идет с самого начала и до N-1 (1 первого до предпоследнего элемента).

Далее вложенный for, где используется уже другой счетчик j. Он изменяется от текущего i-того, увеличенного на 1 и до конца. И далее мы проверяем 2 элемента массива в условии и, если все нас устраивает, то производим обмен переменными. В последнем цикле мы выводим отсортированный массив.

Заметим, что первоначальный массив при сортировке не сохраняется. Кроме того, хочу вам показать, как можно сделать так, чтобы первая половина массива была отсортирована по возрастанию, а вторая - по убыванию.

```
Program SortPopolam;
Var i, j, k: integer;
mas: array[1..10] of integer;
begin
	randomize;
	for i := 1 to 10 do mas[i] := random(101);
	writeln('массив до сортировки');
	for i := 1 to 10 do write(mas[ i], ' ');
	for i := 1 to 4 do
	for j := i + 1 to 5 do
	if mas[ i] > mas[ j] then
	begin
		k := mas[ i];
		mas[i] := mas[j];
		mas[j] := k;
	end;
	for i := 5 to 9 do
	for j := i + 1 to 10 do
	if mas[ i] < mas[ j] then
	begin
		k := mas[ i];
		mas[ i] := mas[j];
		mas[j] := k;
	end;
	writeln;
	writeln('массив после сортировки');
	for i := 1 to 10 do write(mas[ i], ' ');
end.
```

Надеюсь, что вы заметили, что здесь 2 алгоритма сортировки, первый идет с первого элемента по 5, а второй - с 5 по 10. Они отличаются только условиями, в этом и заключается этот нехитрый способ.

Как видите, здесь нет ничего особенного. И еще один пример по сортировке выбором. Пусть нудно отсортировать массив таким образом, чтобы числа в нем располагались в порядке возрастания своих последних разрядов. Допустим, если у нас массив 17 23 18 70, то после сортировки он должен выглядеть так: 70 23 17 18. Вот код этой программы.

```
Program SortOstatok;
Var   a: array[1..10] of Integer;
i, j, k: Integer;
begin
        randomize;
        for i := 1 to 10 do a[ i] := random(100);
        writeln('массив до сортировки');
        for i := 1 to 10 do write(a[ i], ' ');
        for i := 1 to 9 do
        for j := i + 1 to 10 do
        if a[i] mod 10 > a[j] mod 10 then begin
                k := a[i];
                a[i] := a[j];
                a[j] := k;
        end;
        writeln;
        writeln('массив после сортировки');
        for i := 1 to 10 do write(a[i], ' ');
end.
```

Эта программа схожа с первой, их отличие лишь в том, что разное условие для перестановки элементов массива. В первой это «if mas[i] < mas[j] then», а во втрой «if a[i] mod 10 > a[j] mod 10 then» То есть мы располагаем наши элементы в таком необычном порядке благодаря тому, что просто проверяем остатки их деления на 10, то есть и сравниваем их последние цифры.

Теперь подробнее о **сортировке пузырьком**. Его сущность заключается в том, что мы сравниваем соседние элементы парами: 1 и 2, 2 и 3, 3 и 4, 4 т.д. И если наш элемент удовлетворяет условию сортировки, то мы его проталкиваем в конец массива, он как бы всплывает, прямо как «пузырек». От этого и название этого алгоритма. Вот пример программы с этим алгоритмом.

```
Program SortPusirkom;
Var mas: array[1..20] of integer;
i, j, k: integer;
begin
        randomize;
        for i := 1 to 20 do mas[i] := random(100);
        writeln('массив до сортировки');
        for i := 1 to 20 do write(mas[ i], ' ');
        for i := 1 to 19 do
        for j := 1 to 20 - i do
        if mas[j] > mas[j + 1] then begin
                k := mas[j];
```

```
            mas[j] := mas[j + 1];
            mas[j + 1] := k;
        end;
        writeln;
        writeln('Отсортированный массив: ');
        for i := 1 to 20 do
        write(mas[i], ' ');
end.
```

Ее алгоритм чем-то похож на сортировку выбором. Здесь так же 2 цикла for, но второй цикл идет с 1 до 20 - i.

Кроме того, существует второй, не менее известный алгоритм сортировки «пузырьком» с флагом. Флаг - это переменная типа boolean. Из-за этого меняется структура алгоритма, но сущность остается прежней. Вот он.

```
Program SotrSFlagom;
Var a: array[1..20] of integer;
i, j, L: integer;flag: boolean;
begin
        randomize;
        for i := 1 to 20 do
        a[i] := random(100);
        writeln('массив до сортировки');
        for i := 1 to 20 do write(a[i], ' ');
        i:=0;
        repeat
        i := i + 1;
        flag:=false;
        for j := 19 downto i do
        if a[j] < a[j + 1] then begin
            L := a[j]; a[j] := a[j + 1]; a[j + 1] := L;
            flag := true;
        end;
        until not flag;
        writeln;
        writeln('массив после сортировки');
        for i := 1 to 20 do write(a[i], ' ');
end.
```

Этот алгоритм наиболее сложный для запоминания, но уверяю Вас, что его и не нужно зазубривать, а понимать, что за чем идет. Здесь используется внешний цикл repeat … until not flag; Он существует до тех пор, пока флаг не окажется истинным, то есть flag := true; А это может произойти, если наше условие сортировки выполняется. И наш элемент тоже всплывает, как пузырек. Итак, надеюсь, что вы поняли, в чем отличие этих 2 алгоритмов сортировок и теперь сможете отсортировать любой массив так, как Вам нужно.

Вопросы для самоконтроля

1. Основные методы и этапы проектирования алгоритмов.
2. Нисходящее и восходящее проектирование.
3. Структурное и модульное программирование

Лабораторное занятие 4. Проектирование и оформление алгоритмов поиска.

Тема 3. Алгоритмы поиска и сортировки. Различные комбинации алгоритмических конструкций.

Цель: изучить алгоритмы поиска элементов в массиве.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

Последовательный поиск элемента в массиве

Линейный или последовательный поиск - самый простой из алгоритмов поиска элемента в массиве.

Алгоритм заключается в обходе всех элементов массива, как правило, слева на право, и сравнения их с искомым значением. Если значения элемента и ключа совпадают, то поиск возвращает индекс элемента.

По скольку линейный алгоритм, обходит массив последовательно, он очень медленный.

Тем не менее этот метод используется для поиска:

- на небольших массивах данных;
- в потоковой обработке данных;
- поиске минимального и максимального значения массива;
- на одиночных неупорядоченных больших массивах.

Программа для последовательного поиска элемента массива:

```
program SequentialSearch;
const
arrayLength = 10;
var
inputArray : array [1..arrayLength] of integer;
index, key: integer;
{функция для последовательного поиска}
function LinearSearch(k : integer): integer;
var i : integer;
begin
	i := 1;
	while (i <= arrayLength) do
	begin
		if inputArray[i] = k then
		begin
			LinearSearch := i;
			Exit;	{досрочное завершение функции}
		end;
		i := i + 1;
	end;
	{сюда попадаем в случае если ничего не нашли}
	LinearSearch := -1;
end;
begin
randomize;
```

```
writeln ('Исходный массив: ');
{заполнение случайными числами}
for index := 1 to arrayLength do
        begin
                inputArray[index] := random(100);
                write (inputArray[index]:4);
        end;
 writeln;
 write('Искомое значение ');
 readln(key);
index := LinearSearch(key);
if index = -1 then
writeln ('Элемент не найден')
else
writeln ('Индекс элемента в массиве ', index);
readln;
end.
```

Функция для линейного поиска с использованием цикла for

В функции используется цикл while do, можно легко модифицировать программу использовав цикл for.

```
function LinearSearch(k : integer): integer;
var i : integer;
begin
        for i := 1 to arrayLength do
        if inputArray[i] = k then
        begin
                LinearSearch := i;
                Exit;     {выход из функции последовательного поиска}
        end;
{возвращаем -1 если ничего не нашли}
LinearSearch := -1;
end;
```

Максимальный элемент массива

Алгоритм поиска **максимального элемента** неупорядоченного массива заключается в следующем:

- сначала мы предполагаем, что наибольший элемент находиться в начале массива;
- сохраняем значение первого элемента в переменной;
- затем мы сравниваем его с другими элементами массива один за другим, если какой-либо элемент больше, чем наш предполагаемый максимум, то обновляется значение переменной;
- после обхода всего массива, возвращаем максимальный элемент.

Код программы для поиска максимального элемента массива:

```
program Maximal;
const
arrayLength = 10;
var
```

```
inputArray : array [1..arrayLength] of integer;
maximum, i: integer;
begin
	randomize;
	writeln ('Исходный массив: ');
	{заполнение случайными числами}
	for i := 1 to arrayLength do
	begin
		inputArray[i] := random(100);
		write (inputArray[i]:4);
	end;
	writeln;
	{поиск максимального значения}
	{считаем что первый элемент и есть максимальный}
	maximum := inputArray[1];
	for i := 2 to arrayLength do
	if maximum < inputArray[i] then    {если текущее значение больше
максимального}
	maximum := inputArray[i];        {присваиваем максимуму текущее
значение}
	write('Максимальный элемент массива ', maximum);
	readln;
end.
```

Рекурсивный алгоритм нахождения максимального значения элемента массива

Найти максимальный элемент массива, можно также - рекурсивно. Реализация метода немного сложнее предыдущего, однако полезна для изучения принципов рекурсивных вызовов функций.

```
program Maximum;
const
arrayLen = 10;
var
  inputArr : array [1..arrayLen] of integer;
  max, i: integer;
function MaxElement(maximal, index: integer):integer;
begin
	if index > arrayLen then
	MaxElement := maximal
	else
	begin
		if inputArr[index] > maximal then
		maximal := inputArr[index];
		MaxElement := MaxElement(maximal, index + 1); {рекурсивный
	вызов}
	end;
end;
begin
	randomize;
```

```
        writeln ('Исходный массив: ');
        for i := 1 to arrayLen do
        begin
                inputArr[i] := random(100);
                write (inputArr[i]:4);
        end;
        writeln;
        {рекуррентный поиск максимума}
        max := inputArr[1];
        max := MaxElement(max, 2);
        write('Наибольший элемент==', max);
        readln;
end.
```

Минимальный элемент массива

Найти минимальный элемент массива очень просто. Если это упорядоченный массив, то достаточно вернуть первое или последнее значение, в зависимости от того, как отсортированы данные, от наименьшего к наибольшему или от наибольших к наименьшим. Это очень простая задача.

В случае с неотсортированным массивом, задача поиска минимального значения элемента сводиться к полному обходу всех элементов и выбора из них - минимума.

Код программы для поиска минимального, по значению, элемента неупорядоченного массива

```
program Minimal;
const
arrayLength = 10;
var
inputArray : array [1..arrayLength] of integer;
minimum, i: integer;
begin
        randomize;
        writeln ('Исходный массив: ');
        {заполнение случайными числами}
        for i := 1 to arrayLength do
        begin
                inputArray[i] := random(100);
                write (inputArray[i]:4);
        end;
        writeln;
        {поиск минимального значения}
        {считаем что первый элемент и есть минимальный}
        minimum := inputArray[1];
        for i := 2 to arrayLength do
        if minimum > inputArray[i] then         {если минимум больше текущего}
        minimum := inputArray[i];             {присваиваем ему текущее значение}
        write('Минимальный элемент массива==', minimum);
        readln;
end.
```

Найти *минимальное значение*, можно также, с использованием рекурсивного алгоритма.

Рекурсивный алгоритм поиска минимального элемента в одномерном массиве

```
program MinimalElement;
const
arrayLen = 10;
var
inputArr : array [1..arrayLen] of integer;
min, i: integer;
function MinElement(minimal, index: integer):integer;
begin
	if index > arrayLen then
	MinElement := minimal
	else
	begin
		if inputArr[index] < minimal then
		minimal := inputArr[index];
		MinElement := MinElement(minimal, index + 1); {рекурсивный вызов}
	end;
end;
begin
	randomize;
	writeln ('Исходные данные: ');
	for i := 1 to arrayLen do
	begin
		inputArr[i] := random(100);
		write (inputArr[i]:4);
	end;
	writeln;
	{рекуррентный поиск минимального значения}
	min := inputArr[1];
	min := MinElement(min, 2);
	write('Минимальный элемент==', min);
	readln;
end.
```

Бинарный поиск элемента в массиве

Бинарный поиск (binary search) - алгоритм поиска индекса элемента в упорядоченном массиве, в нем используется деление массива на половины, по это й причине алгоритм называют **методом деления пополам**.

Метод бинарного поиска достаточно прост для понимания, в то же время он очень эффективен. Поскольку на каждой итерации количество элементов в рабочей области массива уменьшается вдвое.

Описание алгоритма бинарного поиска

- определяем значение элемента в средине рабочей области массива и сравниваем его с искомым;
- если они равны, выводим значение;

- если значение середины больше искомого, то поиск продолжается в первой половине, иначе во второй;
- проверяем не сошлись ли границы рабочей области, если да - искомого значения нет, нет - переходим на первый шаг.

Рекурсивная реализация бинарного поиска

```
program BinSearch1;
const
arrayLength = 15;
var
sortedArray : array [1..arrayLength] of integer;
i, k, index : integer;
function BinSearch(key, leftIndex, rightIndex : integer):integer;
var
middleIndex : integer;
begin
	if leftIndex > rightIndex then
	BinSearch := -1
	else
	begin
		middleIndex := (leftIndex + rightIndex) div 2;
		if sortedArray[middleIndex] = key then
		begin
			BinSearch := middleIndex;
		end
		else
		begin
			if sortedArray[middleIndex] > key then
			BinSearch := BinSearch(key, leftIndex, middleIndex) {рекурсивный вызов функции}
		else
		BinSearch := BinSearch(key, middleIndex + 1, rightIndex); {рекурсивный вызов функции}
		end;
	end;
end;
begin
	writeln('Исходный массив: ');
	{заполнение массива числами}
	for i := 1 to arrayLength do
	begin
		sortedArray[i] := i * 2;
		write(sortedArray[i]:4);
	end;
	writeln;
	write('Введите значение искомого элемента==');
	readln(k);
	index := BinSearch(k, 1, arrayLength);
	if index = -1 then
```

```
        writeln ('Элемент не найден')
        else
        writeln ('Индекс элемента в массиве ', index);
        readln;
end.
```

Итеративная реализация алгоритма бинарного поиска

```
program BinSearch2;
const
elementsCount = 15;
var
sortedArr : array [1..elementsCount] of integer;
i, key, res : integer;
function BinarySearch(key : integer):integer;
var
firstIndex, midIndex, lastIndex: integer;
begin
        firstIndex := 1;
        lastIndex := elementsCount;
        while firstIndex < lastIndex do
        begin
                midIndex := (firstIndex + lastIndex) div 2;
                if sortedArr[midIndex] = key then
                begin
                        BinarySearch := midIndex;
                        Exit; {выход из процедуры если индекс найден}
                end
                else
                begin
                        if sortedArr[midIndex] > key then
                        lastIndex := midIndex
                        else
                        firstIndex := midIndex + 1;
                end;
        end;
        {сюда попадаем в случае если ничего не нашли}
        BinarySearch := -1;
end;
begin
        writeln('Исходный массив: ');
        for i := 1 to elementsCount do
        begin
                sortedArr[i] := i * 3;
                write(sortedArr[i]:4);
        end;
        writeln;
        write('k = ');
        readln(key);
```

```
    res := BinarySearch(key);
    if res = -1 then
    writeln ('Поиск завершен, элемент не найден')
    else
    writeln ('Индекс элемента в массиве ', res);
    readln;
end.
```

Вопросы для самоконтроля

1. Алгоритмы поиска.
2. Алгоритмы сортировки.
3. Вложенные циклы.
4. Вспомогательные алгоритмы.
5. Различные комбинации алгоритмических конструкций.
6. Алгоритм Евклида.
7. Декомпозиция алгоритма.

Лабораторное занятие 5. Проектирование и оформление сложных алгоритмов.

Тема 3. Алгоритмы поиска и сортировки. Различные комбинации алгоритмических конструкций.

Цель: изучить символьный алгоритм и составной алгоритм.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

Тип данных char.

В большинстве применений компьютера алфавитно-цифровая информация используется на ряду числовой информацией, прежде чем мы с Вами сможем написать программу, которая манипулирует алфавитно-цифровыми знаками - то есть литерами, нам потребуется тип данных для их представлений.

В языке pacal для этого существует тип данных char, также как переменная типа integer может хранить одно число, так и переменная типа char может хранить один символ.

Давайте рассмотрим какие значения может принимать переменная типа char в программе, на не большом примере:

```
Program char_;
uses crt;
var alpha: Char;
begin
clrscr;
alpha := 'p';
alpha := '+';
alpha := '2';
alpha := ' ';
readln;
end.
```

Переменная типа char может принимать значения в виде буквы, и все значения обязательно заключать в одинарные кавычки, тогда программа посчитает этот как символ.

Также переменная может принимать значение в виде знаков - +, -, =, и т.д.

В виде цифры - 1, 2, 3, и т.д.

И также хотим заметить, что символ два не является числом(цифрой), которая может учавствовать в арифметических операциях, а это уже просто символ.

И в переменной может содержаться пробел, хотя мы его и не видим на экране, но всё же это есть символ - значение типа char.

Вы можете увидеть все символы во Free Pascal в таблице кодов, перейдя в меню - Tools->Ascii table.

Теперь давайте напишем простую программу, которая запрашивает ввод двух литер, и сравнивает их. Как можно сравить литеры? - просто, каждая литера имеет свой номер, если номер одной литеры больше другой, то первая литера естественно больше:

```
Program char_;
uses crt;
var a, b: Char;
begin
	clrscr;
	write('Введите две литеры без пробела - ');
	readln(a, b);
	write('Первая литера ');
	if (a < b) then
	write('меньше второй')
	else
	if (a = b) then
	write('равная второй')
	else
	write('больше второй');
	readln;
end.
```

Как Вы уже должны были заметить литеры вводятся не через пробел, так как он тоже считается литерой.

Также для типа char есть специальные функции, а именно:

Succ() - возвращает следующий символ;

Pred() - возвращает предыдущий символ;

Chr() - возвращает значение кода литеры;

Ord() - возвращает значение литеры по коду;

В первом случае функция будет выводить следующий символ, после символа, который мы дадим ей на обработку, во втором случае всё также, только функция будет возвращать предыдущий символ. Далее идёт функция, которая будет выводить номер литеры данной ей на обработку. И последняя функция по заданному ей номеру литеры, будет выводить саму литеру.

Составное условие.

В этом уроке мы с Вами рассмотрим конструкцию условия, в которой мы будем проверять сразу несколько совпадений - например равно ли первое число второму и второе третьему. Для такой проверки используются зарезервированные слова, которые дают понять программе, что дальше будет ещё одно условие, но в этом случаем будет истина, если оба условия верны, но также можно проверять их не зависимо друг от друга.

Давайте рассмотрим на примере использование сразу несколько условий.

```
Program IF_ELSE;
uses crt;
var num1, num2, num3: Integer;
begin
	clrscr;
	write('Введите три числа через пробел - ');
	readln(num1, num2, num3);
	if ((num1 = num2)and(num2=num3)) then
	writeln('Все числа равны!')
	else
```

```
        writeln('Числа не равны!');
        readln;
end.
```

Мы совместили два условия при помощи команды and(и), также можно было добавить ещё условий. И ещё можно проверять на правильность одно условие из двух, например:

```
Program IF_ELSE;
uses crt;
var num1, num2, num3: Integer;
begin
        clrscr;
        write('Введите три числа через пробел - ');
        readln(num1, num2, num3);
        if ((num1 = num2)or(num2=num3)or(num1=num3)) then
        writeln('Два числа равны!')
        else
        writeln('Числа не равны!');
        readln;
end.
```

Наше условие будет проверять первое условие, если нет, то второе, а затем третье, пока не будет совпадений, если нет то возьмёт иначе. Мы использовали команду or(или).

Вопросы для самоконтроля

1. Алгоритмы поиска.
2. Алгоритмы сортировки.
3. Вложенные циклы.
4. Вспомогательные алгоритмы.
5. Различные комбинации алгоритмических конструкций.
6. Алгоритм Евклида.
7. Декомпозиция алгоритма.

Лабораторное занятие 6. Изучение инструментария среды программирования.

Тема 4. Классификация и генеалогия актуальных языков программирования. Основные элементы языка.

Цель: изучить элементы интерфейса Lazarus.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

Lazarus - среда быстрой разработки программного обеспечения для компилятора Free Pascal, аналогичная Delphi.

Данный проект базируется на оригинальной кроссплатформенной библиотеке визуальных компонентов Lazarus Component Library (LCL).

Кроссплатформенное программное обеспечение - это программное обеспечение, работающее более чем на одной аппаратной платформе и/или операционной системе.

Free Pascal - это компилятор языков Pascal и Object Pascal, работающий под Windows, Linux, Mac OS X, FreeBSD, и другими ОС.

Таким образом, разработанные приложения могут функционировать практически под любой операционной системой.

Все, что вы видите на экране во время работы различных приложений, все элементы (кнопки, бегунки, меню и т.п.) можно реализовать в Lazarus.

В Lazarus используется технология визуального программирования. Пользователь для создания графического интерфейса приложения использует готовые компоненты, значки которых находятся на панели компонентов. После того как он помещает компонент на форме, программный код для него генерируется автоматически. Вручную остается запрограммировать только те действия, которые будет выполнять это приложение.

Процесс создания приложения можно разделить на следующие этапы:

1. Создание проекта. В результате на экране появляется пустая форма (окно будущего приложения).

2. Создание графического интерфейса проекта - расположение необходимых элементов, задание размеров, изменение свойств;

3. Написание программного кода, который определит, что будет делать ваша программа.

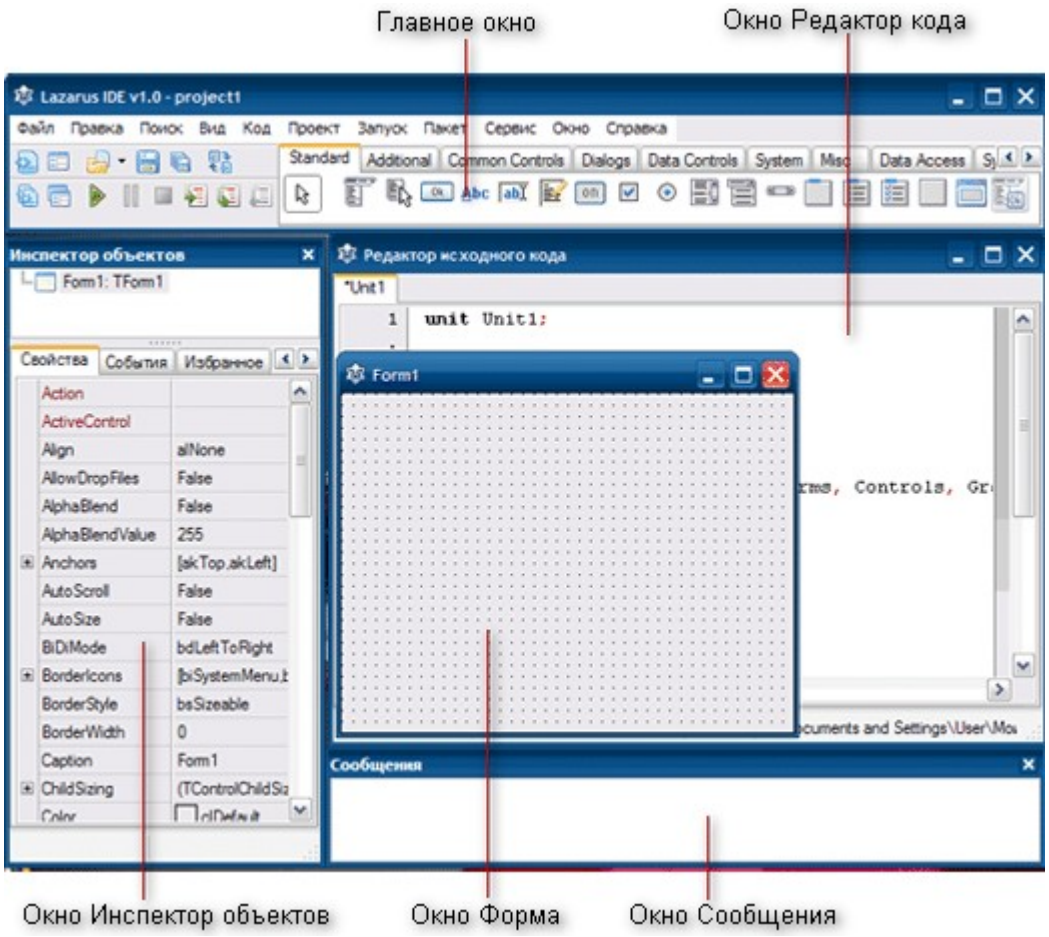
4. Отладка программы.

Чтобы познакомится с основными инструментами среды разработки, запустим среду программирования.

Для этого выполните команду:

Пуск => Все программы => Lazarus => Lazarus.

При этом запускается оболочка создания приложений, называемая интегрированной средой разработки IDE (Integrated Development Environment). На экране появиться набор окон.



Вы видите все основные инструменты среды разработки Lazarus:

1. Окно формы - окно будущего приложения.

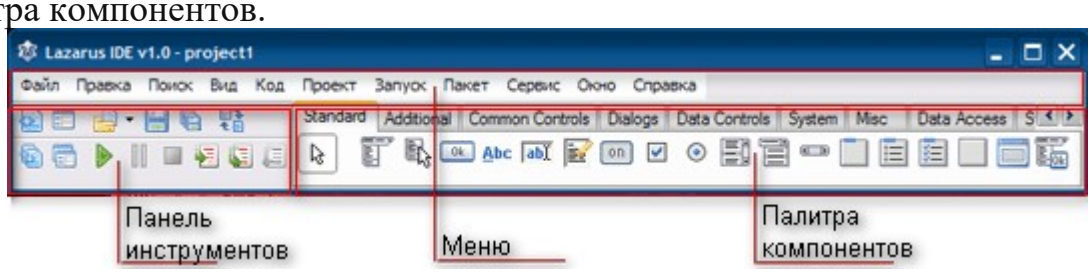
2. Главное окно, содержащее три панели: меню, панель инструментов, палитру компонентов. Палитру компонентов вы будете использовать для выбора необходимых вам для создания пользовательского интерфейса компонент.

3. Окно Инспектор объектов, содержащее файлы проекта и окно со вкладкой Свойства, в котором вы будете настраивать свойства помещенных на форму объектов.

4. Окно Редактор исходного кода, в котором вы будете писать программный код.

Дадим появившимся окнам краткую характеристику.

Главное окно. Здесь располагаются меню, панель инструментов и палитра компонентов.

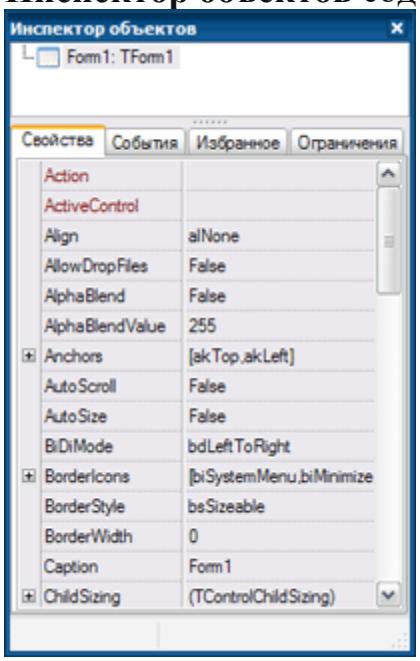


На Палитре компонентов, представляющей собой множество тематических вкладок, располагаются визуальные и не визуальные компоненты для вашей будущей программы.

Не визуальные компоненты видны только на первом этапе создания приложения - при редактировании.

Главное окно остается открытым все время работы IDE. Закрывая его, вы, тем самым, закрываете Lazarus и все открытые в нем окна.

Инспектор объектов содержит четыре страницы

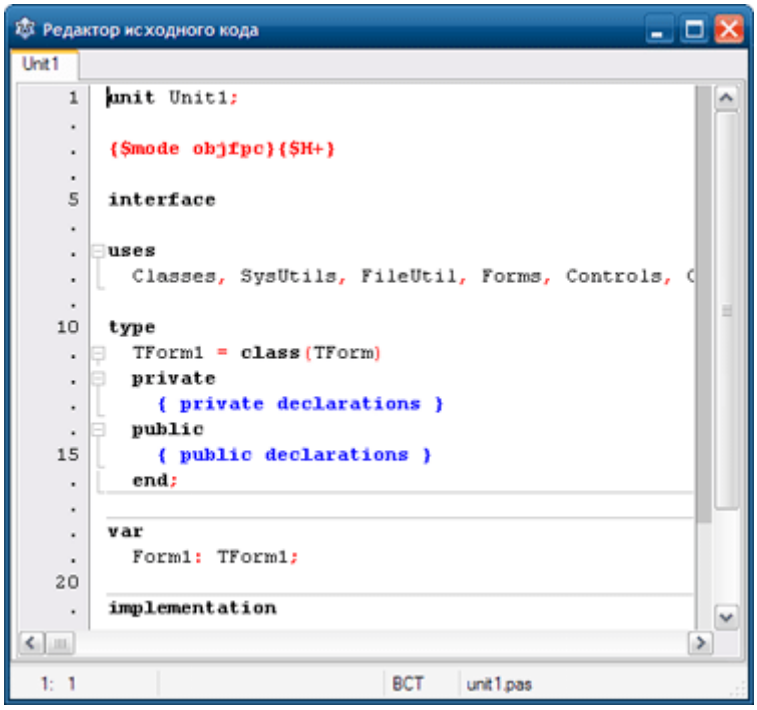


На первой странице «Свойства» постоянно отображаются все доступные свойства выбранного компонента. В левой колонке содержится список всех свойств выделенного в данный момент компонента, в правой - значения свойств.

Значения свойств можно менять еще до запуска проектируемой программы. Например, для будущего окна вашего приложения (формы) свойство Name имеет значение Form1. Для изменения имени достаточно изменить его в Инспекторе объектов.

На второй странице «События» находятся возможные обработчики событий для выбранного компонента. В левой колонке расположены названия события, в правой - соответствующие процедуры.

Окно Редактора кода. На момент первого запуска оно имеет заголовок Unit1.



В окне Редактор исходного кода вы будите писать программный код программы, и само окно очень похоже на обычный текстовый редактор. Для

удобства при редактировании текста программы строки пронумерованы, предусмотрено выделение цветами:

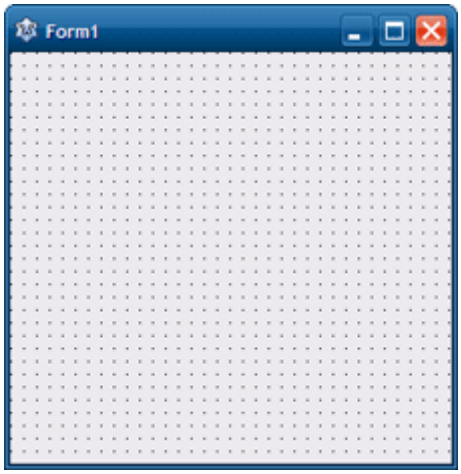
- все служебные слова выделяются жирным шрифтом;
- знаки препинания становятся красными;
- строки с ошибками выделяются коричневым цветом;
- комментарии могут заключаться в фигурные скобки {} и выделяются синим.

Текст программы разбивается на части - процедуры и функции.

Основную работу программист производит именно здесь.

Проектировщик форм. У каждого Windows-приложения должно быть хотя бы одно окно.

Lazarus при первом запуске автоматически предлагает пользователю новый проект, открывая пустую форму под названием Form1, и назначает его главным окном.



Перенося на него элементы из палитры компонентов, вы тем самым, предварительно оформляете его.

Главное окно в проекте может быть только одно. Все другие создаваемые окна будут дочерними. Закрывая главное окно стандартной кнопкой закрытия окна, или программно, вы закрываете и все дочерние окна.

Вопросы для самоконтроля

1. Классификация языков программирования.
2. Понятие системы программирования.
3. Основные элементы языка.
4. Структура типовой программы.
5. Особенности среды программирования.

6. **Лабораторное занятие 7. Подготовка структуры программы в среде программирования.**

Тема 4. Классификация и генеалогия актуальных языков программирования. Основные элементы языка.

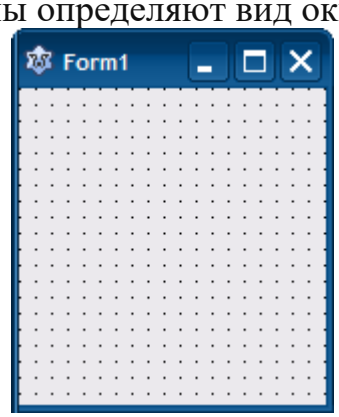
Цель: изучить основные компоненты Lazarus.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

Компонент TForm

Форма (объект типа TForm) является основой программы. Свойства формы определяют вид окна программы.



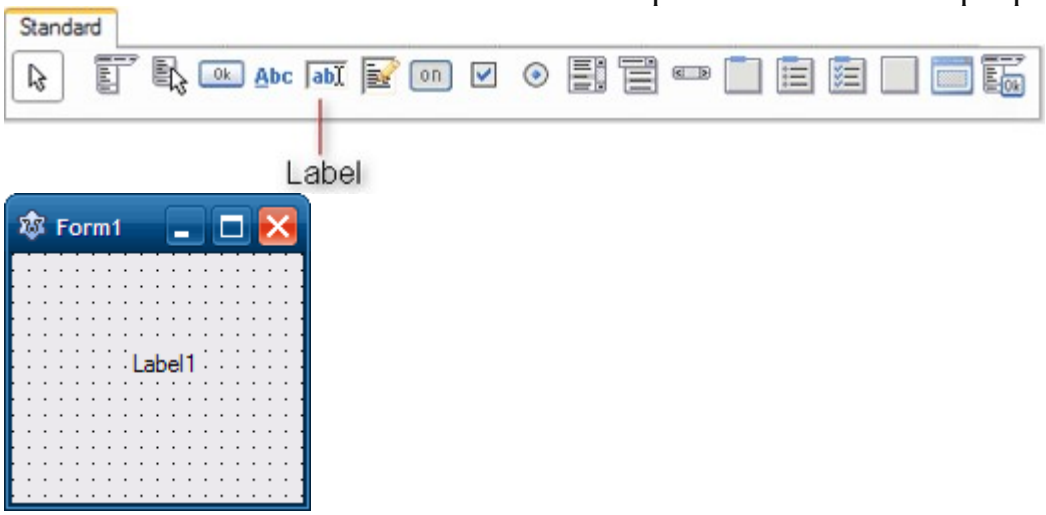
Основные свойства формы

Свойство	***Описание***
Name	*Имя формы. В программе имя формы используется для управления формой и доступа к компонентам формы.*
Caption	*Текст заголовка окна.*
Top	*Расстояние от верхней границы формы до верхней границы экрана.*
Left	*Расстояние от левой границы формы до левой границы экрана.*
Width, Height	*Ширина, высота формы.*
Icon	*Значок в заголовке диалогового окна, обозначающий кнопку вывода системного меню.*
Color	*Цвет фона.*
Font	*Шрифт. Шрифт, используемый по «умолчанию» для компонентов, находящимися на поверхности формы.*
Canvas	*Поверхность, на которую можно вывести графику.*

Компонент TLabel

Компонент TLabel (Надпись)

Компонент Label (Надпись) используется для вывода на форму текста, который пользователь не может изменить во время выполнения программы.

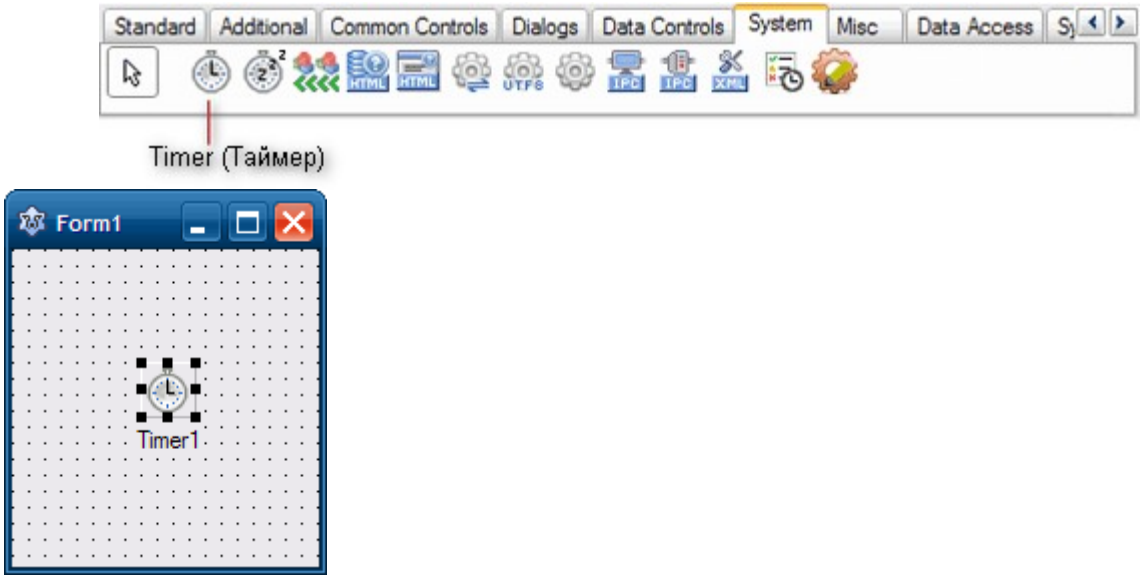


Основные свойства

Свойство	*Описание*
Name	*Имя компонента. Используется в программе для доступа к компоненту и его свойствам.*
Caption	*Отображаемый в поле надписи текст.*
Left	*Расстояние от левой границы поля вывода до левой границы формы.*
Top	*Расстояние от верхней границы поля вывода до верхней границы формы.*
Width,Height	*Ширина, высота поля вывода.*
AutoSize	*Признак того, что размер поля определяется его содержимым.*
WordWrap	*Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку (значение свойства* AutoSize *должно быть* False*).*
Alignment	*Задает способ выравнивания текста внутри поля:*taLeftJustify — *выровнивание по левому краю;*taCenter – выравнивание *по центру;*taRightJustify — *Выравнивание по правому краю*
Font	*Параметры шрифта, используемые для отображения текста:* Font.Name — *вид шрифта*;Font.Size – *размер шрифта*;Font.Color – *цвет шрифта*.
ParentFont	*Признак наследования компонентом характеристик*

	шрифта формы, на которой находиться компонент. Если значение свойства равно True*, то текст выводиться шрифтом, установленным для формы.*
Color	*Цвет фона области вывода текста.*
Transparent	*Управляет отображением фона области вывода текста. Значение* True *делает область вывода текста прозрачной, (область не закрашивается цветом, заданным свойством* Color*).*
Visible	*Позволяет скрыть текст (*False*) или сделать его видимым (*True*).*

Компонент TTimer



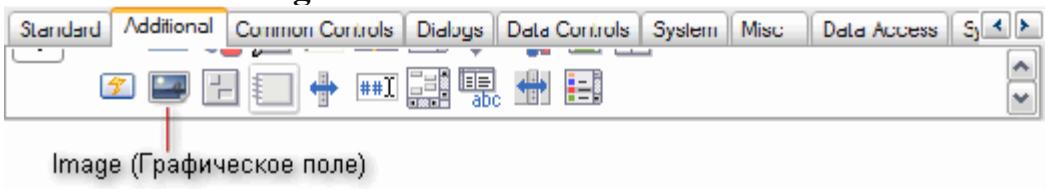
С помощью таймера (Timer) можно запрограммировать выполнение определенного кода через равные интервалы времени. Когда таймер установлен на форме, система периодически генерирует событие OnTimer. Для пользователя таймер невидим.

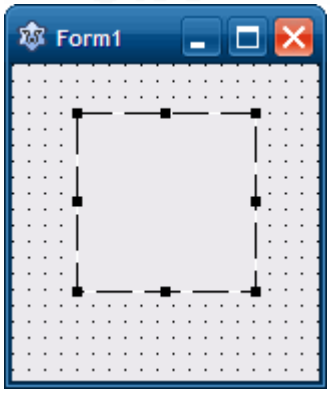
Основные свойства

Свойство	*Описание*
Name	*Имя компонента. Используется для доступа к компоненту.*
Interval	*Интервал времени между генерацией событий OnTimer, выраженный в миллисекундах (мс). Отсчет времени начинается с момента установки свойства* Enabled *в* True.
Enabled	*Разрешение работы. При значение* True *таймер включается,* False — выключается.

Чтобы отключить таймер, нужно присвоить свойству Enabled значение False или свойству Interval — значение 0

Компонент TImage





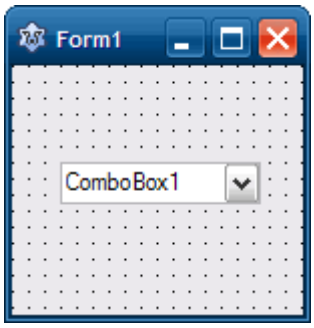
*Компонент (*TImage*) обеспечивает вывод на поверхность формы иллюстраций, представленных в* bmp*–формате (чтобы компонент можно было использовать для отображения иллюстраций в формате* JPG*, надо подключить модуль* JPEG *– указать имя модуля в директиве* uses*).*

Основные свойства

Свойство	***Описание***
Picture	Иллюстрация, которая отображается в поле компонента.
Width, Height	Размер компонента. Если размер компонента меньше размера иллюстрации, и значение свойств AutoSize, Strech и Proportional равно False, то изображается часть иллюстрации.
Proportional	Признак автоматического масштабирования картинки без искажения. Чтобы масштабирование было выполнено, значение свойства AutoSize должно быть False.
Strech	Признак автоматического масштабирования (сжатия или растяжения) иллюстрации в соответствии с реальным размером компонента. Если размер компонента не пропорционален размеру иллюстрации, то иллюстрация будет искажена.Обратите внимание: свойство Stretch не влияет на файлы рисунков типа .ico.
AutoSize	Признак автоматического изменения размера компонента в соответствии с реальным размером иллюстрации.
Center	Признак определяет расположение картинки в поле компонента по горизонтали, если ширина картинки меньше ширины поля компонента. Если значение свойства равно False, то картинка прижата к правой границе компонента, если True – то картинка располагается по центру.

Visible	Отображается ли компонент и соответственно, иллюстрация на поверхности формы.
Canvas	Поверхность, на которую можно вывести графику.

Компонент TComboBox



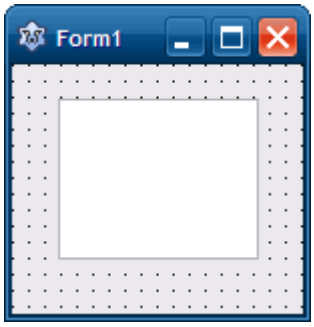
*Компонент (*TComboBox*) дает возможность ввести данные в поле редактирования путем набора на клавиатуре или выбором из списка.*

Основные свойства

Свойство	*Описание*
Name	*Имя компонента. Используется в программе для доступа к компоненту и его свойствам.*
Text	*Текст, находящийся в поле ввода-редактирования*
Items	*Элементы списка – массив строк*
Count	*Количество элементов списка*
Sorted	*Признак необходимости автоматической сортировки (*True*) после добавления очередного элемента.*
ItemIndex	*Номер выбранного элемента. Элементы списка нумеруются с нуля. Если в списке ни один из элементов не выбран, то значение равно минус 1.*
DropDownCount	*Количество отображаемых элементов в раскрытом списке. Если количество элементов списка больше чем* DropDownCont *, то появляется вертикальная полоса прокрутки.*
Left	*Расстояние от левой границы компонента до левой границы формы.*
Top	*Расстояние от верхней границы компонента до*

	верхней границы формы.
Width	*Ширина компонента.*
Height	*Высота компонента (поля ввода-редактирования).*
Font	*Шрифт, используемый для отображения элементов списка.*
ParentFont	*Признак наследования свойств шрифта родительской формы.*

Компонент TListBox



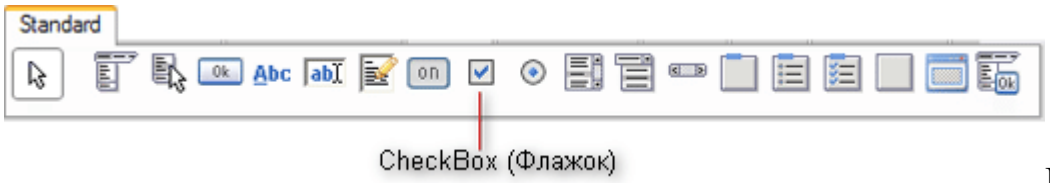
*Компонент (*TListBox*) представляет собой список, в котором можно выбрать нужный элемент.*

Основные свойства

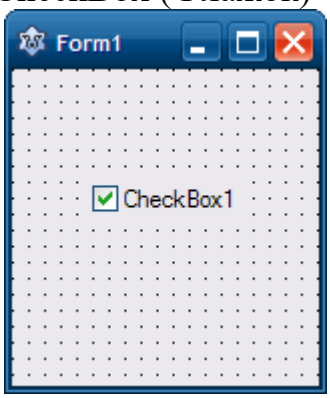
Свойство	*Описание*
Name	*Имя компонента. Используется в программе для доступа к компоненту и его свойствам.*
Items	*Элементы списка – массив строк*
Count	*Количество элементов списка*
Sorted	*Признак необходимости автоматической сортировки (*True*) после добавления очередного элемента.*
ItemIndex	*Номер выбранного элемента. Элементы списка нумеруются с нуля. Если в списке ни один из элементов не выбран, то значение равно минус 1.*
Left	*Расстояние от левой границы списка до левой границы формы.*
Top	*Расстояние от верхней границы списка до верхней границы формы.*

Width	*Ширина поля списка.*
Height	*Высота поля списка.*
Font	*Шрифт, используемый для отображения элементов списка.*
ParentFont	*Признак наследования свойств шрифта родительской формы.*

Компонент TCheckBox



Компонент CheckBox (Флажок)



*Компонент Флажок (*TCheckBox*) предоставляет пользователю два варианта выбора – его можно установить или снять. Установленный флажок отмечен галочкой. Когда флажки объединены в группу, пользователь может установить или снять любые флажки группы. Одновременно может быть включено несколько флажков.*

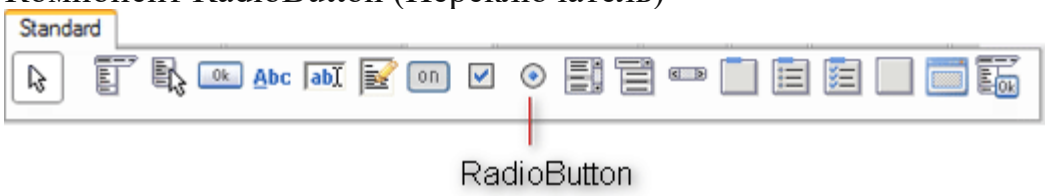
Основные свойства

Свойство	*Описание*
Name	*Имя компонента. Используется в программе для доступа к компоненту и его свойствам.*
Caption	*Текст, который находиться справа от флажка.*
Checked	*Состояние, внешний вид флажка: если флажок установлен (в квадратике есть «галочка»), то значение* True; *если флажок сброшен (нет «галочки»), то* False.
State	*Состояние флажка. В отличие от свойства* Checked, *позволяет различать установленное, сброшенное и промежуточное состояния. Состояние флажка определяет одна из*

	констант: cbChecked *(установлен);* cbGrayed *(серый, неопределенное состояние);* cbUnChecked *(сброшен).*
AllowGrayed	*Свойство определяет, может ли флажок быть в промежуточном состоянии: если* AllowGrayed = False, *то флажок может быть только установленным или сброшенным, если* AllowGrayed = True, *то допустимо промежуточное состояние.*
Left	*Расстояние от левой границы флажка до левой границы формы.*
Top	*Расстояние от верхней границы флажка до верхней границы формы.*
Width, Height	*Ширина, высота поля вывода поясняющего текста*
Font	*Шрифт, используемый для отображения поясняющего текста.*
ParentFont	*Признак наследования характеристик шрифта родительской формы.*

Компонент TRadioButton

Компонент RadioButton (Переключатель)



Компонент TRadioButton (переключатель) *в отличии от флажка (CheckBox), позволяют выбрать только один из предложенных вариантов.*

Когда пользователь устанавливает один из переключателей, все остальные переключатели группы автоматически снимаются

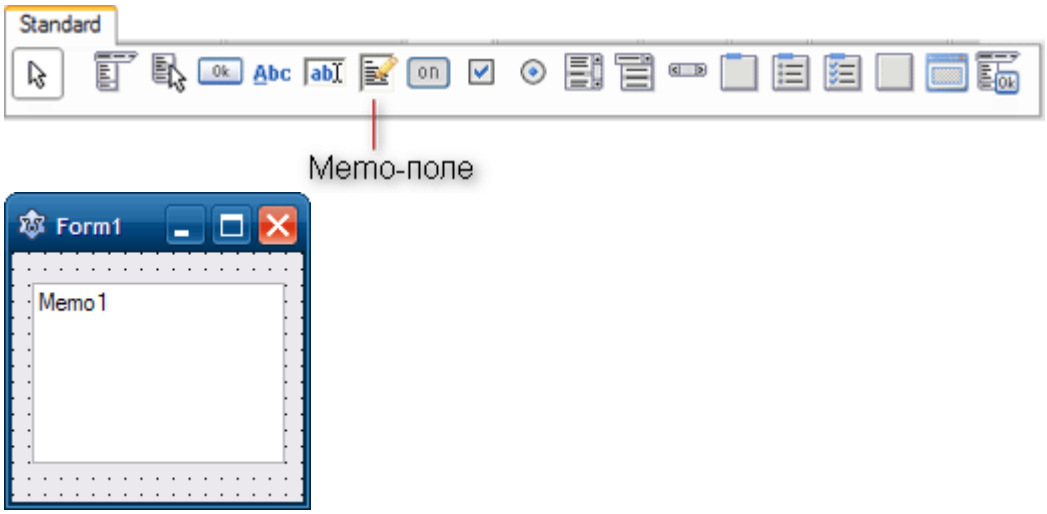
Если надо организовать несколько групп переключателей, то каждую группу следует представить компонентом TRadioGroup.

Основные свойства

Свойство	*Описание*
Name	*Имя компонента. Используется в программе для доступа к компоненту и его свойствам.*

Caption	*Текст, который находиться справа от кнопки.*
Checked	*Состояние, внешний вид кнопки: если кнопка выбрана, то значение* True *, если не выбрана значение* False.
Left	*Расстояние от левой границы флажка до левой границы формы.*
Top	*Расстояние от верхней границы флажка до верхней границы формы.*
Width, Height	*Ширина, высота поля вывода поясняющего текста*
Font	*Шрифт, используемый для отображения поясняющего текста.*
ParentFont	*Признак наследования характеристик шрифта родительской формы.*

Компонент TMemo



Основное предназначение компонента TMemo — работа с большим количеством строк (ввод, отображение и редактирование текстового материала).

Для работы с буфером обмена можно использовать общепринятые горячие клавиши: Ctrl-X — вырезать, Ctrl-C — копировать; Ctrl-V — вставить.

Основные свойства

Свойство	**Описание**
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам, в частности для доступа к тексту, введенному в поле редактирования.
Text	Текст, находящийся в поле Memo. Рассматривается как единое целое.

Lines	Массив строк, соответствующий содержимому поля. Доступ к строке осуществляется по номеру. Строки нумеруются с нуля.
Lines.Count	Количество строк текста в поле Memo.
Left	Расстояние от левой границы поля до левой границы формы.
Top	Расстояние от верхней границы поля до верхней границы формы.
Width, Height	Ширина, высота поля.
Font	Шрифт, используемый для отображения вводимого текста.
ParentFont	Признак наследования свойств шрифта родительской
WantReturns	Клавиша для ввода конца строки: TRUE – клавиша ENTER; FALSE – сочетание клавиш CTRL + ENTER.
WordWrap	Переход в начало следующей строки при вводе длинных строк: TRUE –производится автоматически; FALSE – не производиться.При включенной горизонтальной полосе прокрутке это свойство игнорируется.
ScrollBar	Использование полосы прокрутки, если текст большой и не помещается в компоненте Memo: ssNone — Нет полосы прокрутки; ssHorizontal — Установлена горизонтальная прокрутка; ssVertical — Установлена вертикальная прокрутка; ssBoth — Установленs две полосы прокрутки.
ReadOnly	Разрешает или запрещает редактирование текста. (Программно все равно текст можно добавлять).

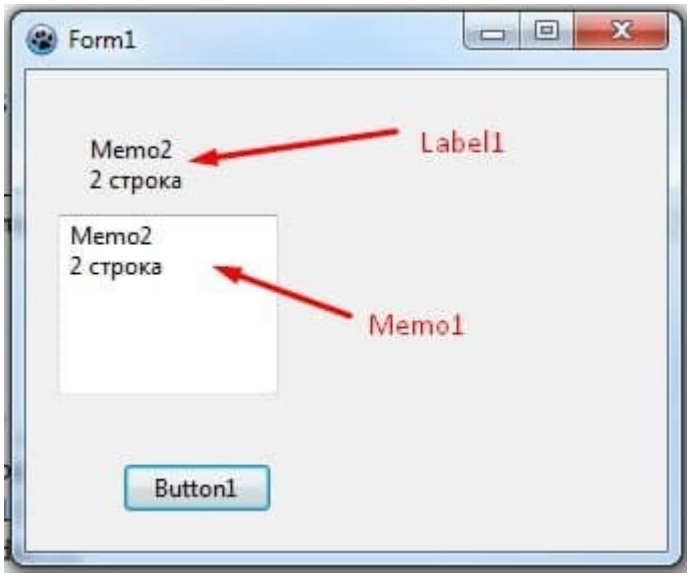
Для сохранения содержимого текстового поля Memo в файл используется функция SaveToFile(‘mytetxt.txt’), а для извлечения - LoadFromFile(‘mytetxt.txt’), где mytetxt.txt — текстовый файл, расположенный в каталоге программы.

Дополнение.

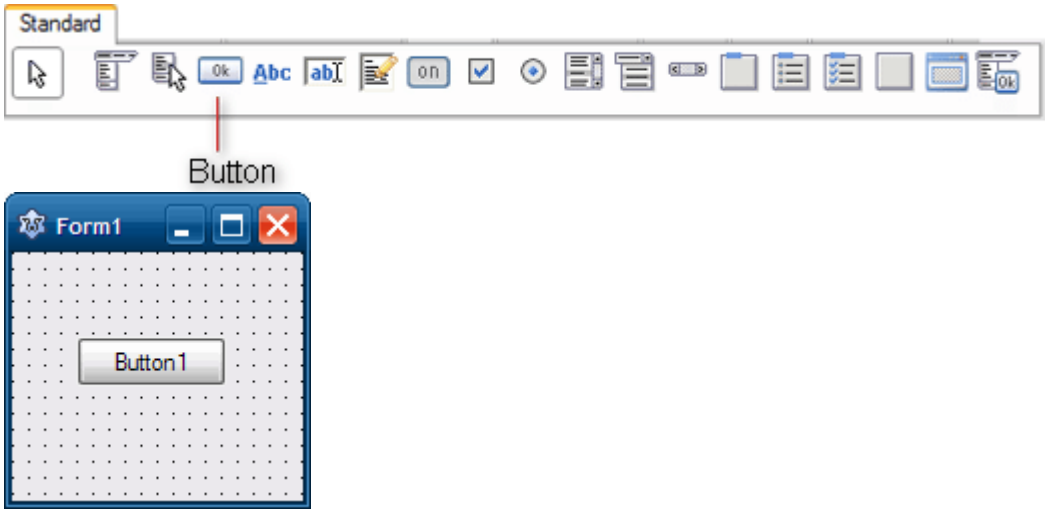
Для примера разместите на форму элементы button, memo и label. И вставте код ниже в кнопку.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
label1.caption := Memo1.Text
end;
```

После запуска программы при нажатии на кнопку текст в label поменяется на текст расположенный в memo.



Компонент TButton



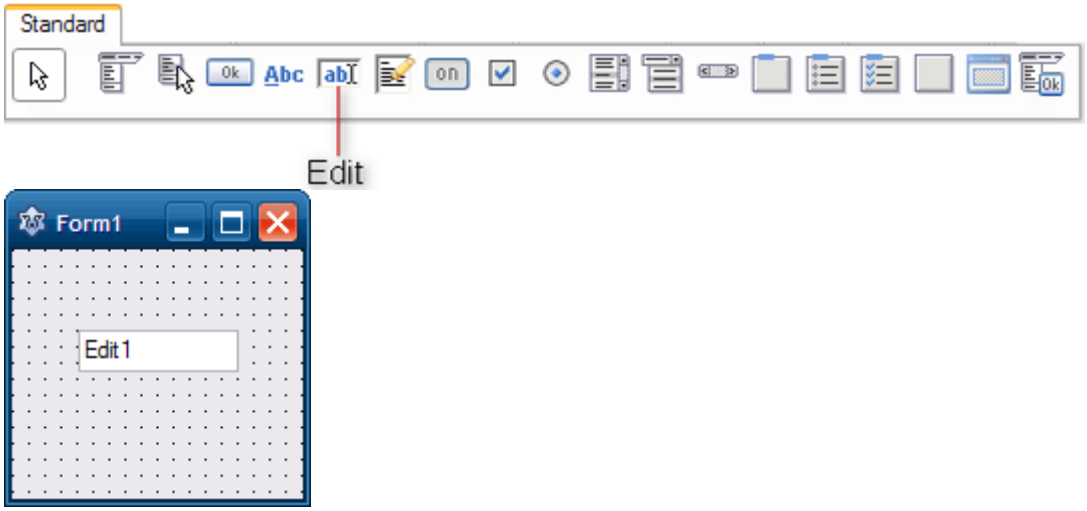
Компонент Button (Кнопка) – командная кнопка, с помощью которой пользователь может вызывать выполнение какого-либо действия.

Основные свойства

Свойство	*Описание*
Name	*Имя компонента. Используется в программе для доступа к компоненту и его свойствам.*
Caption	*Текст на кнопке.*
Left	*Расстояние от левой границы кнопки до левой границы формы.*
Top	*Расстояние от верхней границы кнопки до верхней границы формы.*
Width, Height	*Ширина, высота кнопки.*
Enabled	*Признак доступности кнопки.* True —*кнопка доступна* False — *кнопка недоступна.*

	Например, в результате щелчка на кнопке событие Click *не возникает.*
Visible	*Позволяет скрыть текст.* False – текст видим. True – текст невидим.
Hint	*Контекстная подсказка – текст, который появляется рядом с указателем мыши при наведении указателя (для того чтобы текст появился, надо чтобы значение свойства* ShowHint *было* True*).*
ShowHint	*Разрешает (*True*) или запрещает (*False*) отображение подсказки при наведении указателя на кнопку.*

Компонент TEdit



*Компонент (*TEdit*) представляет из себя поле ввода-редактирования строки символов.*

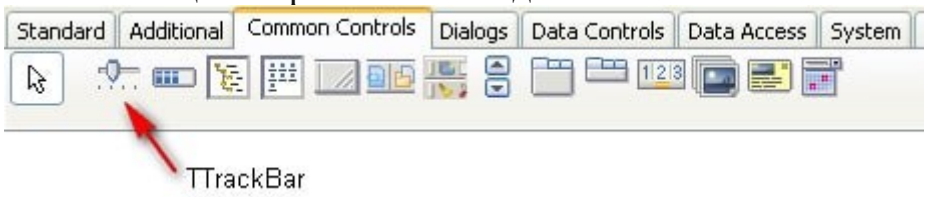
Основные свойства

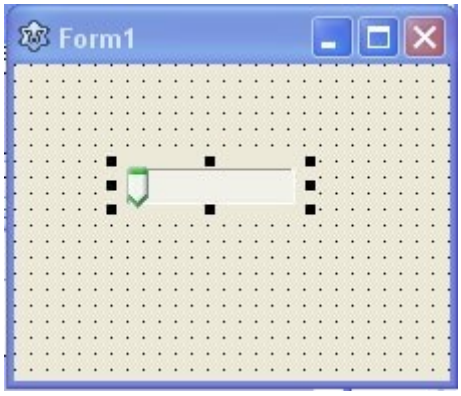
Свойство	*Описание*
Name	*Имя компонента. Используется в программе для доступа к компоненту и его свойствам, в частности для доступа к тексту, введенному в поле редактирования.*
Text	*Текст, находящийся в поле ввода и редактирования.*
Left	*Расстояние от левой границы компонента до левой границы формы.*
Top	*Расстояние от верхней границы компонента до верхней границы формы.*
Width, Height	*Ширина, высота поля.*
Font	*Шрифт, используемый для отображения вводимого*

	текста.
ParentFont	*Признак наследования компонентом характеристик шрифта формы, на которой находиться компонент. Если значение свойства равно* True*, то при изменении свойства* Font *формы автоматически меняется значение свойства* Font *компонента.*
Enabled	*Используется для ограничения возможности изменить текст в поле редактирования. Если значение свойства равно* False *, то текст в поле редактирования изменить нельзя.*
Visible	*Позволяет скрыть текст (*False*) или сделать его видимым (*True*).*

Компонент TTrackBar


Движок (TTrackBar) обычно применяется там, где надо в визуальном режиме выставить с помощью мыши какое-либо приближенное значение с помощью перетаскивания движка по шкале.





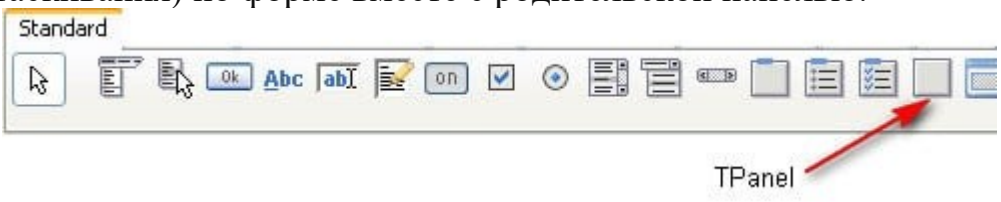
Внешний вид движка настраивается с помощью следующих свойств.

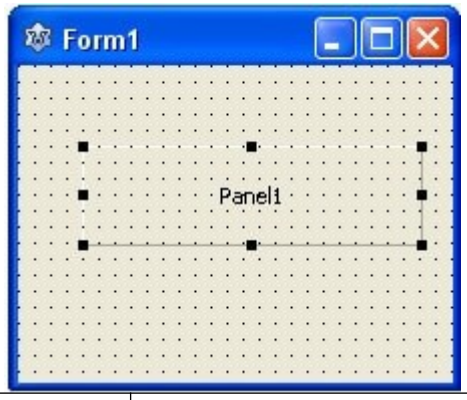
Свойство	Название
Frequecy	Частота засечек
Min Max	Минимальная и максимальная допустимые границы
Orientation	Ориентация: горизонтальная (значение trHorizontal) или вертикальная (значение trVertical)
Selstart SelEnd	Начало и конец «оптимального» диапазона в рамках границ Min/Max по аналогии с приборами управления. Область оптимального управления выделяется дополнительными засечками и другим цветом

SliderVisible	Видимость движка
ThumbLength	Толщина полосы движка в пикселях
TickMarks	Положение засечек. Возможные значения: tmBottomRight (снизу); tmTopLeft (сверху); tmBoth (с обоих сторон)
TickStyle	Способ отображения засечек на движке. Возможные значения: tsAuto (автоматически); tsManual (программно); tsNone (вообще не отображать)

Компонент TPanel

Компонент Панель (TPanel) предназначена для объединения произвольных элементов управления с возможностью их перемещения (перетаскивания) по форме вместе с родительской панелью.

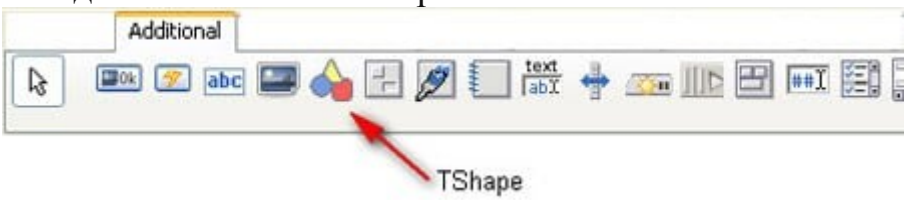


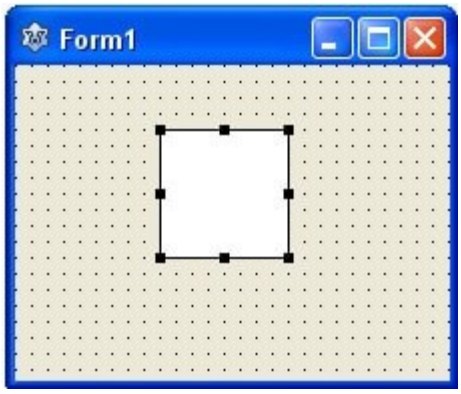


Свойство	Название
BovelInner и BovelOuter	Задают стили оформления внутренней и внешней рамок панели. Они могут принимать одно из четырех значений: bvNone Отсутствует blLowered «Вдавленная» «рамка bvRaised «Выпуклая» рамка bvSpace «Плоская» рамка
BovelWidth	Определяет расстояние между внутренней и внешней рамками (в пикселях)
BorderWidth	Определяет ширину рамки вокруг панели в пикселях

Компонент TShape

Компонент  Фигура (TShape) предназначен для отображения на форме различных геометрических фигур. Конкретная форма геометрического объекта задается в свойстве Shape.





Возможны следующие значения свойства Shape:.

Значение	Форма фигуры
stCircle stEllipse stRectangle stRoundRect stRoundSquare stSquare	Круг Эллипс Прямоугольник Прямоугольник с округленными краями Квадрат с округленными углами Квадрат

Цвет фигуры определяется кистью объекта (свойство Brush), границы фигуры – пером (свойство Pen).

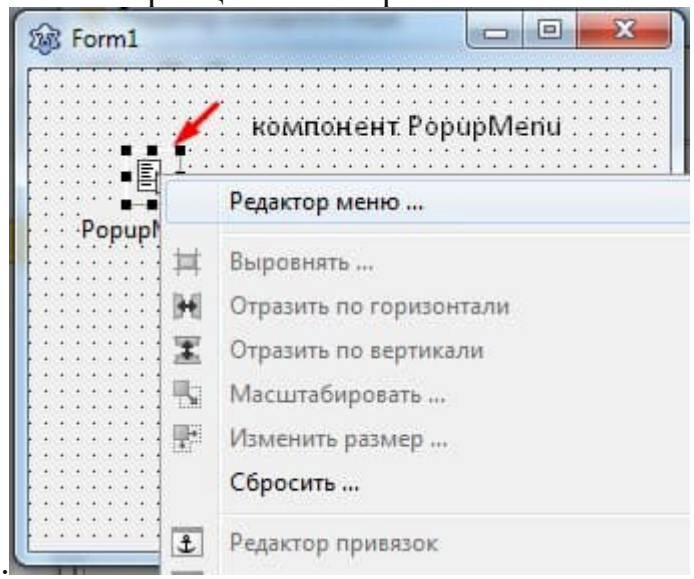
Компонент TPopupMenu

Компонент TPopupMenu — служит для появления всплывающего меню когда вы щелкаете правой кнопкой мыши по приложению. У Lazarus очень простая реализация этого всплывающего меню.

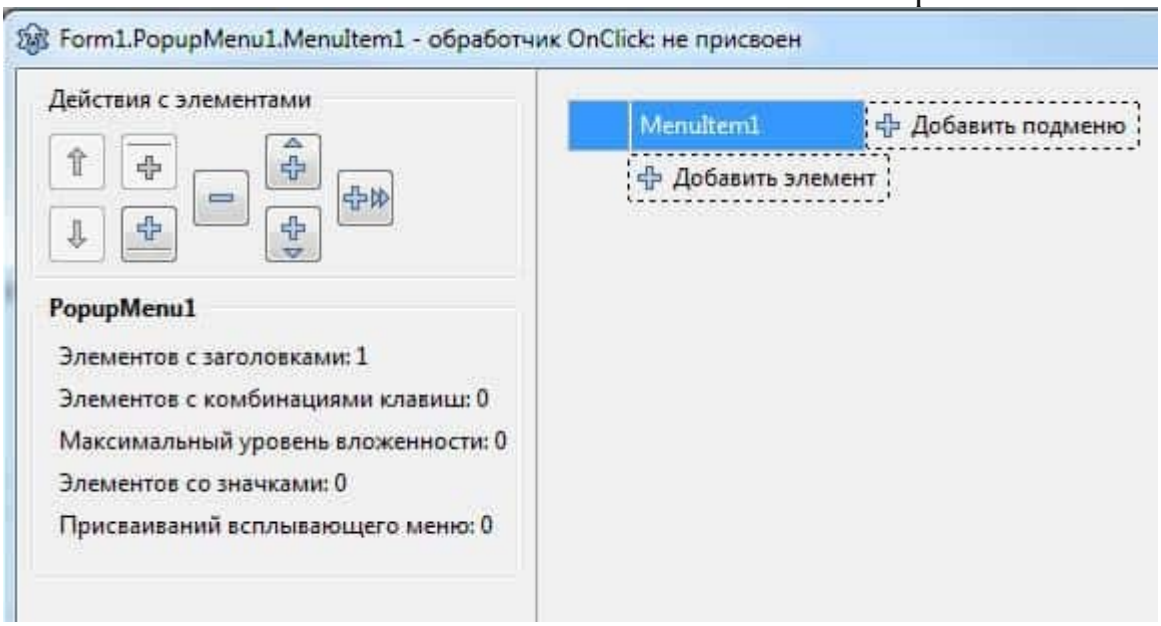
Создание всплывающего меню

Создать всплывающее меню очень просто. Просто перетащите компонент TPopupMenu на форму. Разместить его можно в любом месте т.к

элемент не визуальный. Теперь щелкните правой кнопкой мыши и выберите «Редактор меню …».



Вы увидите 1 пункт меню, уже созданный для вас. Вы можете изменить его свойство, например, свойство Caption на что-то вроде «мое первое меню». Для этого нужно щелкнуть элемент, чтобы выбрать его, а затем изменить свойство в Инспекторе объектов.



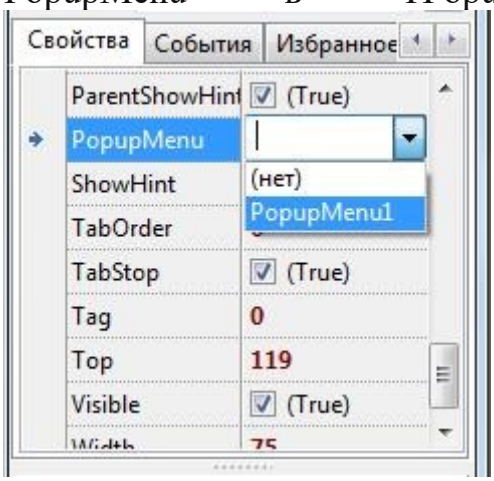
Если вы хотите добавить больше элементов меню, то вам необходимо щелкнув левой кнопкой мыши по «Добавить элемент» или «Добавить подменю» и после этого появится новый пункт меню. Слева есть форма «Действия с элементами» с помощью которой вы можете удалять, вставлять или менять порядок элементов меню.

Использование всплывающего меню 3 способами

Способ 1: использование свойства PopupMenu (без кода)

Вы можете всплывающее меню без написания кода! Если вы установите свойство PopupMenu компонента на только что подготовленное TPopupMenu, то при щелчке правой кнопкой мыши по этому компоненту появится меню.

Для примера разместите TButton на форме. Установите его свойство PopupMenu в TPopupMenu из выпадающего меню.



Теперь запустите проект (F9 или Run-> Run) и щелкните правой кнопкой мыши на кнопке, чтобы увидеть всплывающее меню.

Способ 2: использование кода для отображения всплывающего меню

Использование свойства PopupMenu для отображения всплывающего меню просто и подходит практически для всех целей. Иногда вы можете захотеть сделать несколько пользовательских кодов, прежде чем показывать всплывающее меню. Тогда использование этого метода — хорошая идея.

Разместите другой компонент TButton на форму (предположим, он называется Button2). В его событии OnMouseDown (Инспектор объектов-> События-> OnMouseDown) введите следующий код:

```
procedureTForm1.Button2MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
ifButton = mbRight then
PopupMenu1.PopUp;
end;
```

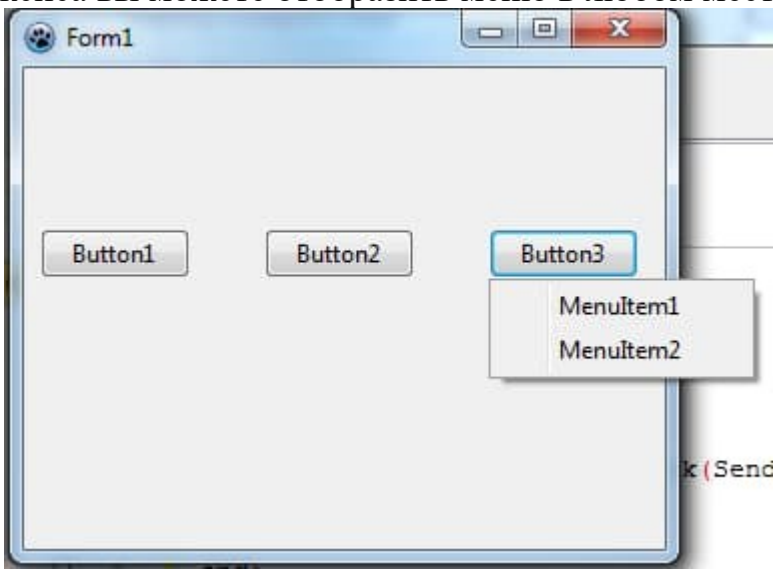
Теперь запустите проект (F9 или Run-> Run) и щелкните правой кнопкой мыши по кнопке, и вы увидите всплывающее меню.TPopupMenu.PopUp показывает меню в позиции указателя мыши.

Способ 3: показ меню по левому клику (фиксированное положение)

Также существует возможность добавить всплывающее меню с фиксированным положение. Для этого перетащите еще один TButton (предположим, он называется Button3), а затем дважды щелкните по нему и введите:

```
procedureTForm1.Button3Click(Sender: TObject);
var
pt, pt2: TPoint;
begin
pt.x:=Button3.Left;
pt.y:=Button3.Top+Button3.Height;
pt2:=ClientToScreen(pt);
PopupMenu1.PopUp(pt2.x, pt2.y);
end;
```

Теперь запустите проект и нажмите левой кнопкой миши по кнопке Button3. Вы увидите меню, которое появится внизу кнопки. Процедура PopUp должна показывать меню по указателю мыши, а не внизу кнопки. Как мы это сделали? С помощью синтаксиса PopUp (x, y). С помощью этого синтаксиса вы можете отобразить меню в любом месте экрана.



Компонент TToggleBox

Компонент TToggleBox это кнопка с надписью, имеющая 2 логических состояния — *нажата* или *не нажата*. Перевод кнопки из одного состояния в другое переключается одиночным щелчком мыши.

Для проверки статуса кнопки (активна она или нет) можно воспользовавшись командой **ToggleBox.Checked;**. Вы можете использовать свойства *Checked* как обычное логическое значение.

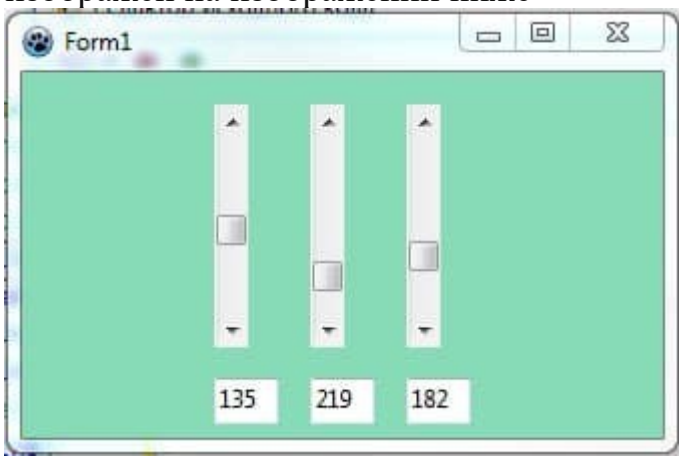
Компонент TScrollBar

Компонент TScrollBar в Lazarus является элемент управления, который позволяет пользователю прокручивать содержание связанного управления путем перемещения ползунка. Перемещать ползунок можно как мышкой, так и с клавиатуры. С помощью ползунка **TScrollBar** мы можем изменять целое число, не выходящее за пределы диапазона чисел Min и Max определеное в свойстве компонента. Для изменения ориентации ползунка используется свойство Kind принимающее значения sbHorizontal и sbVertical

Для примера использования TScrollBar можно взять цветовую модель RGB для этого разместим на форме 3 TScrollBar каждый из которых будет отвечать за свой цвет из RGB т.е красный, зеленый и голубой и 3 элемента edit для отображения в них значений наших ползунков. Пропишите в свойстве min в каждом ползунке 0, а в свойстве max 255. Затем находим в ScrollBar событием OnChange и двойным щелчком открываем редактор кода куда вставляем код

```
procedure TForm1.ScrollBar3Change(Sender: TObject);
begin
Form1.color:=RGBToColor(ScrollBar1.Position,ScrollBar2.Position,ScrollBar3.Position);
edit1.text:=ScrollBar1.Position.ToString;
edit2.text:=ScrollBar2.Position.ToString;
edit3.text:=ScrollBar3.Position.ToString;
end;
```

для двух других элементов **TScrollBar** в событие OnChange выбираем из выпадающего списка созданное событие и запускаем проект, результат изображен на изображении ниже



Вопросы для самоконтроля

1. Классификация языков программирования.
2. Понятие системы программирования.
3. Основные элементы языка.
4. Структура типовой программы.
5. Особенности среды программирования.

Лабораторное занятие 8. Реализация простых циклических алгоритмов.

Тема 5. Методы реализации типовых алгоритмов. Операторы и операции.

Цель: изучить работу циклических алгоритмов.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

Цикл с параметром

Оператор цикла применяется при выполнении расчетов или других действий, повторяющихся определенное количество раз. Оператор имеет вид:

For i:= N1 To N2 Do "оператор";

либо

For i:= N1 DownTo N2 Do "оператор";

Здесь i - параметр цикла (переменная порядкового типа),

N1, N2 - начальное и конечное значения параметра цикла i.

N1, N2 могут быть константами, переменными или выражениями порядкового типа.

Напомним, что "оператор" может иметь вид: Begin "операторы" end;

В случае связки "To" цикл выполняется при условии N1 <= N2 и происходит с единичным возрастанием параметра цикла i от N1 до N2. В случае связки DownTo цикл выполняется при условии N1 >= N2 и происходит с единичным уменьшением параметра цикла i от N1 до N2.

В операторе цикла не разрешается присваивать параметру цикла какое-либо значение.

После окончания цикла значение параметра цикла "i" неопределенно.

Оператор цикла часто применяется для суммирования значений некоторой последовательности чисел или значений функции при известном числе операций суммирования. Напомним некоторые определения, связанные с расчетом суммы последовательности.

Сумма членов последовательности величин

$a_1, a_2, a_3, \ldots, a_n$

называется конечной суммой

$S_n = a_1 + a_2 + a_3 + \ldots + a_n$

Для некоторых последовательностей известны формулы расчета конечных сумм, например:

при $a_n = a_n\text{-}1 + d$; $S_n = (a_1 + a_n)*n/2$; - арифметическая прогрессия,

при $a_n = a_n\text{-}1 * q$; $S_n = (a_1 - a_n*q)/(1-q)$; - геометрическая прогрессия,

где d и q - постоянные числа.

Здесь N-ый член последовательности выражается через (N-1)-ый член. Такие зависимости называются реккурентными.

Конечная сумма последовательности может быть неизвестна, тогда для ее расчета применяется алгоритм суммирования членов последовательности в цикле от 1 до N. Приведем пример расчета конечной суммы последовательности: $12 + 32 + 52 + \ldots + (2*N-1)_2$; $S_n = N*(4*N_2-1)/3$;

```
PROGRAM SUM_K;                    { расчет конечной суммы }
var
```

```
a, S, Sn, i, N : word;
Begin
        write('Введите число членов суммы N=');
        readln(N);
        S:= 0;
        For i:= 1 to N do
        begin                       {  цикл суммирования }
                a := Sqr(2*i-1);
                S:= S+a
        end;
        Sn := N*(4*N*N-1) div 3;
        Writeln('Конечная сумма S=',  S:10:2);
        Writeln('Расчет конечной суммы по формуле Sn=',  Sn:10:2);
        Writeln('Нажми Enter');
        ReadLn;
End.
```

В некоторых случаях "N"-ый член последовательности определяется через сумму предыдущих членов, например,

$a_n = p*S_{n-1}$,

тогда

$S_n = S_{n-1} + a_n = S_{n-1}*(1+p)$,

и конечную сумму можно рассчитать по формуле:

$S_n = S_0*(1+p)N$,

где "S_0" - начальная сумма.

Рассмотрим программу вычисления конечной суммы денежного вклада в банк через N месяцев при ежемесячной процентной ставке "pr" (5% соответствует pr=5).

```
PROGRAM VKLAD;        { расчет конечной суммы вклада в банк }
var
S, Sn, pr : Real;
i, N      : Integer;
Begin
        Write('Введите начальную сумму вклада S=');
        readln(S);
        Write('Введите процент по вкладу pr=');
        readln(pr);
        Write('Введите количество месяцев вклада N=');
        readln(N);
        For i:= 1 to N do S:= S*(1+pr/100);     { цикл произведений }
        Writeln('Конечная сумма вклада S=',  S:10:2);
        { Оператор для расчета "Sn" напишите самостоятельно }
        Writeln('Расчет конечной суммы вклада по формуле Sn=', Sn:10:2);
        Writeln('Нажмите Enter');
        readln;
End.
```

Часто применяются вложенные операторы цикла. Например, если необходимо провести все варианты расчета при изменении нескольких параметров в заданных диапазонах.

Составим программу расчета функции y = A*sin(x) - cos(x)/A; при изменении аргумента "x" в диапазоне от 0 до Pi с шагом Pi/100 и при изменении параметра "A" в диапазоне от 1 до 3 с шагом 0.5.

```
Program tabl;
var y, x, a, dx : real;
i, j: integer;
Begin
        Writeln(' Расчет по формуле:  y=A*sin(x)-cos(x)/A; ');
        Writeln('-----------------------------------------------');
        Writeln('|   X   |  A=1.0 | A=1.5 | A=2.0 | A=2.5 | A=3.0 |');
        Writeln('-----------------------------------------------');
        dx := pi/100;
        for i:= 0 to 100 do
        begin    { внешний цикл изменения аргумента "X" }
                 x:= dx*i;
                 Write( x:8:4 );
                 for j := 1 to 5 do
                 begin{ вложеннный цикл изменения параметра "A" }
                          A := 0.5*(j+1);
                          y := A*sin(x)-cos(x)/A;    Write(y:8:4)
                 end;
                 Writeln;                          {перевод курсора на новую
строчку}
                 if ((i+1) mod 20) = 0 then readln{задержка прокрутки экрана до
нажатия Enter}
          end;
          readln;
End.
```

Вопросы для самоконтроля

1. Переменные.
2. Типы данных.
3. Объявление и инициализация переменных.
4. Область действия и время существования переменных.
5. Константы.
6. Операторы и операции.
7. Ввод – вывод данных.
8. Операторы присваивания.

Лабораторное занятие 9. Реализация алгоритмов обработки одномерных массивов.

Тема 5. Методы реализации типовых алгоритмов. Операторы и операции.

Цель: изучить работу одномерных массивов.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

Одномерные массивы

Одномерный массив – это именованная последовательность, состоящая из пронумерованных элементов одного типа. Элементы могут быть любого имеющегося в **Pascal** (за исключение файлового) типа данных. Номер, также называемый индексом, имеет каждый элемент массива. Индекс должен быть порядкового типа. Одномерный массив можно объявить как в качестве переменной:

var <имя переменной>: array[m..n] of <тип элементов>;

так и типа:

type <имя типа> = array[m..n] of <тип элементов>;

Здесь **m** – номер первого элемента, а **n** – последнего. Например, если диапазон задан так: [1..10], то это означает, что определен одномерный массив размерностью в 10 элементов, с индексами от 1 до 10.

Для обращения к элементу массива нужно указать его имя и номер: mas[i], тут mas – имя, i – номер. В программе ниже мы объявим массив и произведем простые операции над его элементами.

```
program array_primer;
uses crt;
var mas, A: array[1..10] of real;
begin
	clrscr;
	mas[1]:=32;
	mas[5]:=13;
	mas[9]:=43;
	A[1]:=(mas[9]-mas[1])*mas[5];
	write(A[1]:5:2);
	readkey;
end.
```

В каком-то смысле с массивами можно работать, как и с обычными переменными, но представьте, например ситуацию, когда необходимо заполнить массив, состоящий из десятков или тысяч элементов. Это будет удобней сделать посредством цикла. Следующая конструкция заполняет массив числами и выводит их на экран.

```
for i:=1 to n do
begin
mas[i]:=i;
write(mas[i]:3);
end;
```

Если необходимо, чтобы массив состоял из значений, введенных с клавиатуры, то просто замените присвоение на оператор read. Также бывают ситуации, когда требуется заполнить массив случайными числами. Программа ниже поочередно присваивает каждому элементу случайную величину.

```
program array_random;
uses crt;
var i: integer;
mas: array[1..100] of integer;
begin
        clrscr;
        randomize;
        for i:=1 to 100 do
        begin
                mas[i]:=random(10);
                write(mas[i]:2);
        end;
        readkey;
end.
```

Широко распространены задачи связанные с разного рода алгоритмами применимыми к массивам. Среди них особенно популярны методы поиска и сортировки элементов.

Вопросы для самоконтроля

1. Переменные.
2. Типы данных.
3. Объявление и инициализация переменных.
4. Область действия и время существования переменных.
5. Константы.
6. Операторы и операции.
7. Ввод – вывод данных.
8. Операторы присваивания.

Лабораторное занятие 10. Реализация алгоритмов обработки двумерных массивов.

Тема 6. Операторы отношения. Операторы цикла.

Цель: изучить алгоритм работы двумерных массивов.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

Двумерные массивы

Тем, кто знакомым с математическими матрицами, будет не трудно освоить и **двумерные массивы в Pascal**. Матрица – это математический объект, представляющий собой прямоугольную таблицу. Таблица состоит из элементов, которые находятся на пересечении строк и столбцов, определяющих их, то есть i-ая строка и j-ый столбец задают адрес k-ому элементу матрицы (k_{ij}). Двумерные массивы абсолютно аналогичны математическим матрицам, поэтому их можно представить так:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & & & \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}$$

В отличие от одномерных массивов, двумерные характеризуются в программе парой индексов, один из которых соответствует номеру строки, другой – столбца:

Mas[m, n], где Mas – имя массива, n – номер строки, а m – номер столбца.

Описать матрицу в программе можно несколькими способами:

1) В разделе описания переменных:

Var Mas: Array[1..n, 1..m] of <тип элементов>;

2) При помощи одномерного массива, элементами которого являются одномерные массивы.

Пример:

```
Const
n = 5; m = 10;
Type
Arr1 = Array[1..m] of <тип элементов >;
Arr2 = Array[1..n] of arr1;
Var Mas: arr2;
```

Переменная Mas – матрица, состоящая из пяти строк, в каждую из которых включено по десять элементов.

3) Предыдущий способ можно упростить так:

```
Const n = 5; m = 10;
Type arr=Array[1..n] Of Array[1..m] of <тип элементов>;
Var Mas: arr;
```

4) И снова сократив запись, получим:

```
Const n = 5; m = 10;
Type arr = Array[1..n,1..m] of <тип элементов>;
Var Mas: arr;
```

Для обработки содержимого матрицы, удобно пользоваться вложенными циклами:

```
For i:= 1 To n Do
For j:= 1 To m Do
```

В следующей программе массив сначала заполняется числами с клавиатуры, а затем выводиться на экран.

```
program input_and_output_array;
uses crt;
const n=3; m=3;
var i, j: integer;
mas: array[1..n, 1..m] of integer;
begin
        {ввод массива}
        for i:=1 to n do
        for j:=1 to m do
        begin
                write(' Элемент ', i,' строки, ',j,' столбца = ');
                readln(mas[i, j]);
        end;
        writeln(' Получившаяся матрица: ');
        {вывод массива}
        for i:=1 to n do
        begin
                for j:=1 to m do
                begin
                        write(mas[i, j]:5);
                end;
                writeln;
        end;
end.
```

Стоит отметить, что для вывода двумерного массива в виде таблицы необходимо указать перевод строки после выхода из внутреннего цикла.

Количество элементов в массиве (его размерность) можно узнать, умножив количество строк на количество столбцов.

Вопросы для самоконтроля

1. Операторы отношения.
2. Оператор выбора.
3. Операторы перехода.
4. Операторы цикла.
5. Принудительный выход из цикла.

Лабораторное занятие 11. Реализация алгоритмов обработки текстовых данных.

Тема 6. Операторы отношения. Операторы цикла.

Цель: изучить алгоритмы работы с текстовыми данными.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

Обработка текстовых файлов в языке Free Pascal

При работе с текстовыми файлами следует учесть следующее:

1. Действие процедур reset, rewrite, close, rename, erase и функции eof аналогично их действию при работе с компонентными (типизированными) файлами.

2.Процедуры seek, truncate и функция filepos не работают с текстовыми файлами.

3.При работе с текстовыми файлами можно пользоваться процедурой append(f), где f – имя файловой переменной, которая служит для специального открытия файлов для дозаписи. Она применима только к уже физически существующим файлам, открывает и готовит их для добавления информации в конец файла.

4. Запись и чтение в текстовый файл осуществляются с помощью процедур write, writeln, read, readln следующей структуры:

read(f, x1, x2, x3, ..., xn); read(f, x);

readln(f, x1, x2, x3, ..., xn); readln(f, x);

write(f, x1, x2, x3, ..., xn); write(f, x);

writeln(f, x1, x2, x3, ..., xn); writeln(f, x);

В этих операторах f — файловая переменная. В операторах чтения (read, readln) x, x1, x2, x3, …, xn — переменные, в которые происходит чтение из файла. В операторах записи write, writeln x, x1, x2, x3, …, xn — переменные или константы, информация из которых записывается в файл.

Есть ряд особенностей при работе операторов write, writeln, read, readln с текстовыми файлами. Имена переменных могут быть целого, вещественного, символьного и строкового типа. Перед записью данных в текстовый файл с помощью процедуры write происходит их преобразование в тип string. Действие оператора writeln отличается тем, что записывает в текстовый файл символ конца строки.

При чтении данных из текстового файла с помощью процедур read, readln происходит преобразование из строкового типа к нужному типу данных. Если преобразование невозможно, то генерируется код ошибки, значение которого можно узнать, обратившись к функции IOResult. Компилятор FreePascal позволяет генерировать код программы в двух режимах: с проверкой корректности ввода-вывода и без нее.

В программу может быть включен ключ режима компиляции. Кроме того, предусмотрен перевод контроля ошибок ввода-вывода из одного состояния в другое: {$I+} – режим проверки ошибок ввода-вывода включен; {$I-} – режим проверки ошибок ввода-вывода отключен.

По умолчанию, как правило, действует режим {$I+}. Можно многократно включать и выключать режимы, создавая области с контролем ввода и без него. Все ключи компиляции описаны в приложении.

При включенном режиме проверки ошибка ввода-вывода будет фатальной, программа прервется, выдав номер ошибки.

Если убрать режим проверки, то при возникновении ошибки вво- да-вывода программа не будет останавливаться, а продолжит работу со следующего оператора. Результат операции ввода-вывода будет не определен.

Для опроса кода ошибки лучше пользоваться специальной функцией IOResult, но необходимо помнить, что опросить ее можно только один раз после каждой операции ввода или вывода: она обнуляет свое значение при каждом вызове. IOResult возвращает целое число, соответствующее коду последней ошибки ввода-вывода. Если IOResult=0, то при вводе-выводе ошибок не было, иначе IOResult возвращает код ошибки. Некоторые коды ошибок приведены в таблице.

Таблица. Коды ошибок

Код ошибки	**Описание**
2	файл не найден
3	путь не найден
4	слишком много открытых файлов
5	отказано в доступе
12	неверный режим доступа
15	неправильный номер диска
16	нельзя удалять текущую директорию
100	ошибка при чтении с диска
101	ошибка при записи на диск
102	не применена процедура Assign
103	файл не открыт
104	файл не открыт для ввода
105	файл не открыт для вывода
106	неверный номер
150	диск защищён от записи

Рассмотрим несколько практических примеров обработки ошибок ввода-вывода:

1. При открытии проверить, существует ли заданный файл и возможно ли чтение данных из него.

```
assign (f, 'abc.dat');
{$I-} reset(f); {$I+}
if IOResult<>0 then
writeln ('файл не найден или не читается') else
begin read(f,...);
```

```
…
close(f);
end;
```

2. Проверить, является ли вводимое с клавиатуры число целым. var i:integer;

```
begin {$I-} repeat
write('введите целое число i'); readln(i);
until (IOResult=0); {$I+}
```

{ Этот цикл повторяется до тех пор, пока не будет введено целое число} end.

При работе с текстовым файлом необходимо помнить специальные правила чтения значений переменных:

•если вводятся числовые значения, то два числа считаются разделенными, если между ними есть хотя бы один пробел, или символ табуляции, или символ конца строки;

•при вводе строк начало текущей строки идет сразу за последним введенным до этого символом. Вводится количество символов, равное объявленной длине строки. Если при чтении встретился символ «конец строки», то работа с этой строкой заканчивается. Сам символ конца строки является разделителем и в переменную никогда не считывается;

•процедура readln считывает значения текущей строки файла, курсор переводится в новую строку файла, и дальнейший ввод осуществляется с нее.

Считать из файла *input.txt* числа (числа записаны в столбик). Затем записать их сумму в файл *output.txt*

```
var p, x: integer;
f: text;
begin
    assign(f, 'input.txt');
    reset(f);
    p := 1;
    while not eof(f) do begin
        readln(f, x);
        p := p + x;
    end;
    close(f);
    assign(f, 'output.txt');
    rewrite(f);
    writeln(f, 'Сумма чисел ', p);
    close(f);
end.
```

Считать из файла *input.txt* числа (числа записаны в столбик). Затем записать их произведение в файл *output.txt*

```
var p, x: integer;
f: text;
begin
    assign(f, 'input.txt');
```

```
        reset(f);
        p := 1;
        while not eof(f) do begin
                readln(f, x);
                p := p * x;
        end;
        close(f);
        assign(f, 'output.txt');
        rewrite(f);
        writeln(f, 'Произведение чисел ', p);
        close(f);
end.
```

Вопросы для самоконтроля

1. Операторы отношения.
2. Оператор выбора.
3. Операторы перехода.
4. Операторы цикла.
5. Принудительный выход из цикла.

Лабораторное занятие 12. Реализация сложных алгоритмов поиска и ввода-вывода.

Тема 7. Массивы. Управляющие структуры. Подпрограммы.

Цель: изучить работу логического типа данных, ввод и вывод вещественных данных.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

Логический тип данных.

До сих пор мы писали линейные программы, то есть сначала выполняли первое действие, потом второе и т.д. Сегодня мы хотим поговорить с Вами о логическом типе данных.

Скажем у нас есть такая запись - x-y>10

Из этой следует условие, то есть если из x вычесть y то будет ли эта разность больше десяти. В этой записи нам нужно знать x и y - допустим x=11, а y=0, то x-y>10 -> 11-0>10 получается что условие верно - разность 11 и 0 больше 10. Об этом выражении можно сказать что оно булевое или логическое.

Название булевое произошло от имени Джорджа Буля - разработчика булевой логики. Переменная, которая может принимать одно из двух значений - true(истина) и false(ложь) - называется логической или булевой, но мы её будем называть так, как принято - логическая.

Для начала давайте рассмотрим пример простой программы:

```
Program logika;
uses crt;
var a, b: Boolean;
begin
	clrscr;
	a := true;
	b := false;
	write(a, ' ', b);
	readln;
end.
```

В этой программе мы объявили две переменные логического типа - логический тип в Pascal обозначается так - Boolean.

Потом переменным присвоили логические значения - true(истина) и false(ложь), после чего вывели их через пробел. Попробуйте скопировать к себе код этой программы и скомпилировав запустить его - на экране появятся две записи через пробел - TRUE FALSE.

Попробуем немного изменить код программы - пусть переменная a будет результатом сравнения двух чисел:

```
Program logika;
uses crt;
var a: Boolean;
begin
	clrscr;
	a := 2>4;
```

```
        write(a);
        readln;
end.
```

Мы проверили - больше ли двойка чем четыре, и после выполнения программы получим результат - FALSE - что значит ложь. Также можно проверить любые другие числа, например - 10 и 5, тогда результат будет TRUE, что значит истина. Также можно сравнивать и слова, но это пока нам не нужно.

Для последнего примера мы Вам предоставим код программы, которая считывает два числа с клавиатуры и проверяет их на все возможные сравнения - то есть больше, меньше, равно, и т.д.

И выводит на экран ряд сообщений типа - TRUE или FALSE и к ним приписывает какое сравнение было произведено:

```
Program logika;
uses crt;
var a: Boolean;
   num1, num2: Integer;
begin
        clrscr;
        write('Введите через пробел два числа - ');
        readln(num1, num2);
        a := num1 = num2;
        writeln('Первое число равно второму - ', a);
        a := num1 > num2;
        writeln('Первое число больше второго - ', a);
        a := num1 < num2;
        writeln('Первое число меньше второго - ', a);
        a := num1 <> num2;
        write('Первое число не равно второму - ', a);
        readln;
end.
```

Ввод-вывод вещественных чисел.

В pascal вещественное число представляется определённым образом, а именно с помощью десятичного признака и указания степени. Давайте создадим новую программу и в ней пропишем три переменных вещественного типа, где две переменных будем считывать с клавиатуры, и третей переменной присвоим произведение двух вещественных чисел, считанных с клавиатуры:

```
Program Real_Num;
uses crt;
var num1, num2, res: Real;
begin
        clrscr;
        write('Введите два вещественных числа через пробел - ');
        readln(num1, num2);
        res := num1 + num2;
        write('Сумма двух вещественных чисел - ', res);
```

```
    readln;
end.
```

Теперь выполним нашу программу и попробуем ввести два вещественных числа, например - 2.3212 5.2313 - и не стоит забывать что числа нужно вводить с точкой, а не с запятой.

И после получим результат - 1.21428935600000E+001. До знака плюс идёт десятичный признак, а после степень. Но такая запись нам не удобна. Для изменения вида в pascal используется специальная запись:

```
Program Real_Num;
uses crt;
var num1, num2, res: Real;
begin
    clrscr;
    write('Введите два вещественных числа через пробел - ');
    readln(num1, num2);
    res := num1 * num2;
    write('Произведение двух вещественных чисел - ', res:4:4);
    readln;
end.
```

В выводе переменной с произведением мы внесли некоторые изменения, а именно - после переменной res поставили двоеточие и кол-во знаков до запятой - у нас их четыре, потом поставили ещё двоеточие и кол-во знаков после запятой - у нас их столько же. Вот такая запись используется в pascal.

Если знаков до запятой меньше чем мы указали, то при выводе результата он у нас сместиться на недостающее кол-во цифр в право, то есть до видимого результата будут стоять невидимые нули, которые будут сдвигать ответ вправо.

Вопросы для самоконтроля

1. Массивы: определение, виды.
2. Объявление одномерного массива.
3. Ввод и вывод одномерных массивов.
4. Обработка одномерных и двумерных массивов.
5. Управляющие структуры.
6. Понятие потока.
7. Механизм буферизации.
8. Понятие подпрограммы.
9. Библиотеки среды разработки

Лабораторное занятие 13. Создание простейших классов.

Тема 8. Понятие класса и объекта. Общая форма определения класса.

Цель: изучить основы работы в Lazarus.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

1) Первая программа

Для создания графического интерфейса Lazarus предоставляет программисту палитру компонентов пользовательского интерфейса. Программисту требуется всего лишь выбрать на палитре нужные компоненты и с помощью мыши перенести их на форму.

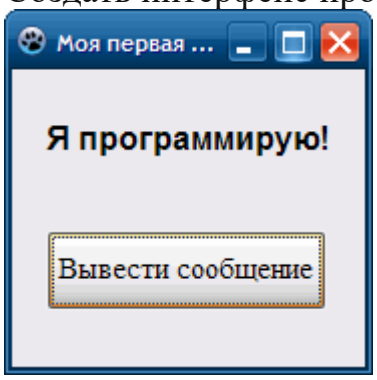
После того, как компонент размещен на форме, он становиться объектом, который имеет свои установленные по умолчанию свойства. Эти свойства можно просматривать и изменять с помощью окна Свойства.

В нашей первой программе при создании интерфейса пользователя будем использовать три компонента: TForm (Форма), Label (Надпись) и TButton (Командная кнопка).

Проект «Первая программа»

Задание. Создать проект, который после щелчка на кнопке выводит в поле надписи текст: «Я программирую!!!»

Создать интерфейс программы по образцу:



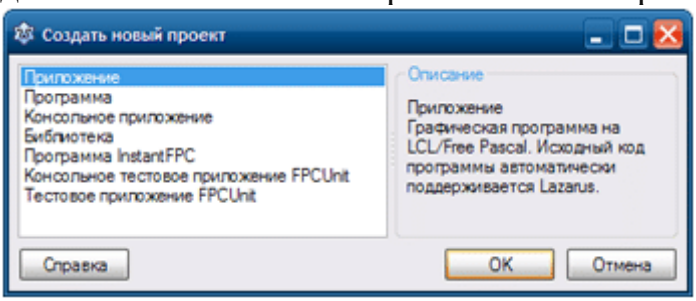
Разместить надпись и кнопку на форме и установите значения свойств, перечисленные в таблице. Когда вы это сделаете, форма примет такой вид , как на рисунке.

В следующих заданиях, когда Вы лучше освоите среду Lazarus, примеры форм будут сопровождаться только листингами исходного кода. От Вас ожидается, что вы , глядя на формы и листинги, сами догадаетесь, как должны быть установлены свойства компонентов.

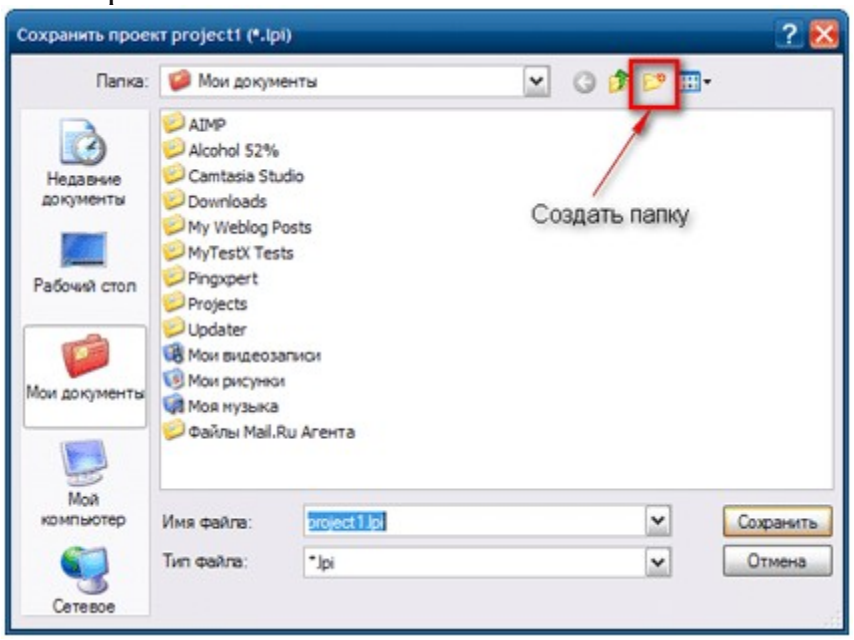
Ход выполнения проекта

1. Загрузите Lazarus. Создайте новый проект.Для этого:1) Выполнить команду Проект => Создать проект …2)В появившемся

диалоговом окне выбрать слово Приложение и нажать кнопку ОК.



2. Сохранить созданный проект.Для этого:
1) Выполнить команду *Проект — Сохранить проект как…* Откроется окно Сохранить проект.



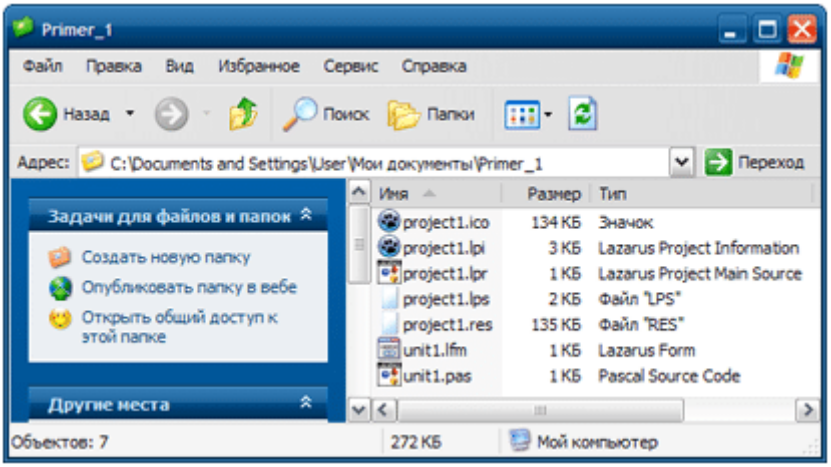
2) Не выходя из этого диалогового окна Создать новую папку Primer_1 для файлов вашего проекта (проект будет содержать несколько файлов), открыть ее и щелкнуть по кнопке Сохранить.

Тем самым мы сохраним файл Project1, содержащий сведения о проекте.

Сразу же откроется окно Сохранить Unit1 для сохранения программного кода проекта (файл Unit1.pas), в котором также необходимо щелкнуть по кнопке Сохранить.

Кроме этих двух файлов в папке проекта создается автоматически еще несколько файлов, в том числе – unit.lfm, который представляет собой файл с полными данными о проектировщике формы. Позиция, размер, расположенные компоненты и пр.

Папка проекта должна содержать следующие файлы:

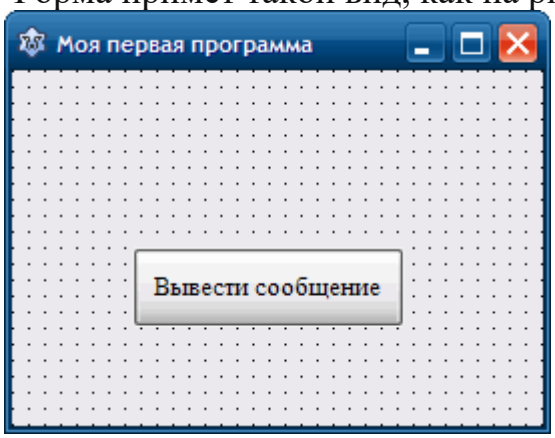


3. Разместите на форме компоненты Надпись (Label) и Кнопку (Button). Разместить компонент на форме можно одним из двух способов. Первый – дважды щелкнуть мышью на значке компонента, расположенного на палитре компонентов. Однако при этом компонент попадет не в то место, куда Вы хотите, а в левый верхний угол формы. Второй – щелкнуть на значке компонента (при этом он выделяется) и щелкнуть на форме. Таким образом компонент можно поместить в любое желаемое место на форме.

4. Установите новые значения для свойств, перечисленные в таблице.

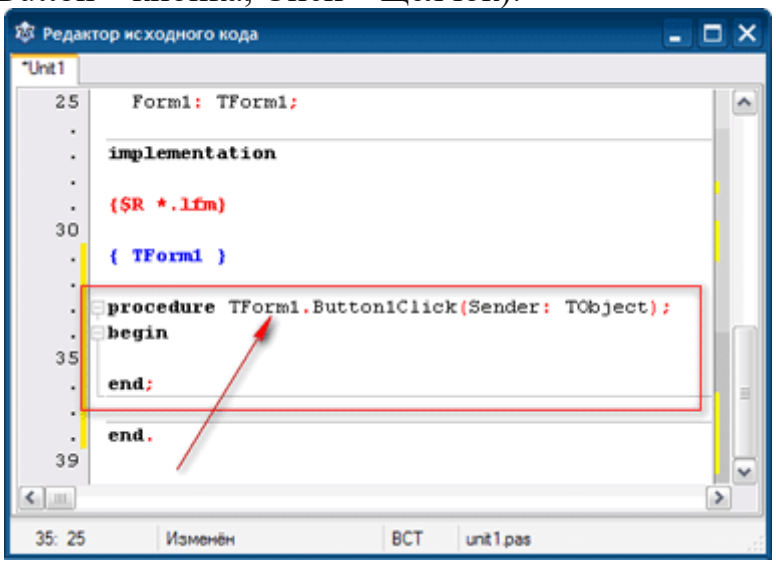
Компонент	**Свойство**	**Значение**
Форма	Caption	Моя первая программа
Надпись	Caption	Пустая строка
	Font NameSize	Arial20
	Style fsBold	True
Кнопка	Caption	Вывести сообщение

5. Форма примет такой вид, как на рисунке:

6. 

7. Напишите программный код для процедуры обработчика события щелчок на кнопке. Пока мы это не сделаем, кнопка не будет работать. При нажатии на кнопку ничего не будет происходить.Для этого:

1) Выполните двойной щелчок по кнопке. Откроется редактор исходного кода, в котором, после кода созданного автоматически, добавиться новая процедура -TForm1.Button1Click – обработчик события щелчок на кнопке (анг. Button – кнопка, Click – щелчок).

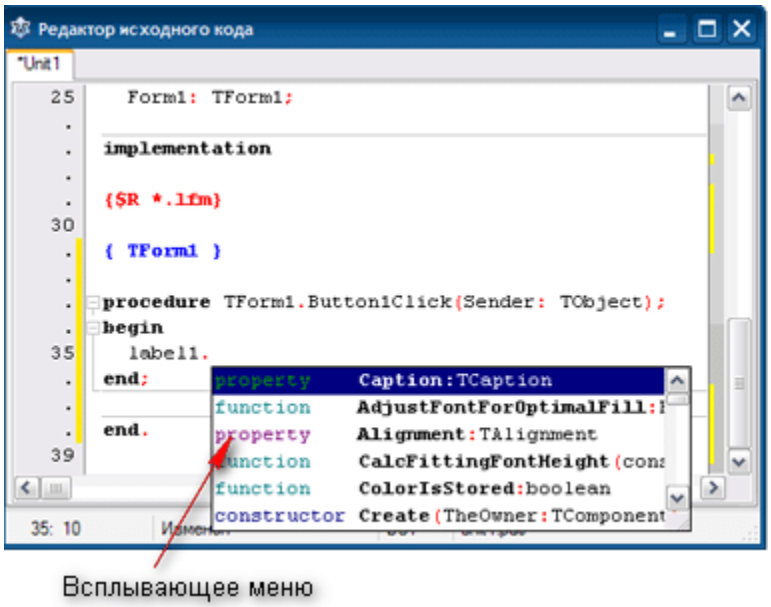


Сейчас процедура обработчика события пустая, при нажатии кнопки она ничего не делает.

2) Чтобы процедура выполнила необходимые действия, напишите соответствующий код между операторными скобками begin и end. В нашем случае это оператор присваивания, который изменяет свойство Caption (текст надписи) объекта Label1 на новое значение:

label1.Caption:='Я программирую!';

Вводя код, обратите внимание на подсказку, появившуюся после ввода точки, следующей за label1. Подсказка представляет собой всплывающее меню, в котором перечислены допустимые свойства и методы компонента label1

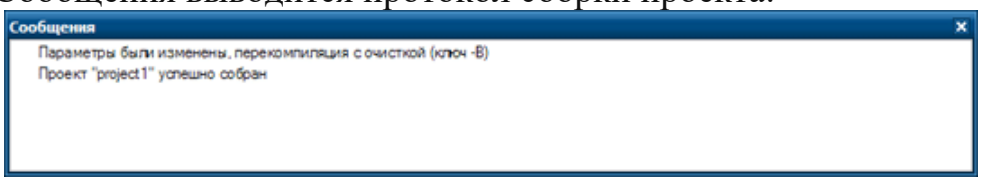


С помощью мыши вы можете выбрать из списка нужное свойство или метод.

Другой способ: вы можете начать вводить имя свойства, при этом Lazarus автоматически прокручивает список и находит имена, первые буквы которых совпадают с вводимыми буквами. Это поможет вам, если вы забыли

точное имя. Если теперь нажать <Space> или <Enter>, то Lazarus вместо вас автоматически завершит ввод имени.

8. Закончив вводить код, выполните программу. Это можно сделать одним из трех способов:
1) щелкнув по кнопке Run (Выполнить) на панели инструментов;
2) выбрав команду Run-Run в главном меню;
3) нажав клавишу <F9>.Происходит сравнительно недолгий процесс компиляции, в результате которого в папке проекта создается EXE файл. В окне Сообщения выводится протокол сборки проекта:



Далее этот файл, в случае успешного создания, запускается на выполнение.

В случае, если были допущены ошибки, сообщение об этом появляется в протоколе.

9. При успешной компиляции на экране появиться форма с кнопкой, однако пока что без надписи. Если теперь щелкнуть на кнопке, то на форме появиться надпись.

10. Таким образом, вы создали приложение, реагирующее на действия пользователя. Скомпилированная программа сохраниться в папке проекта в виде файла с расширением .EXE. Он может быть выполнен на компьютере без среды разработки Lazarus.

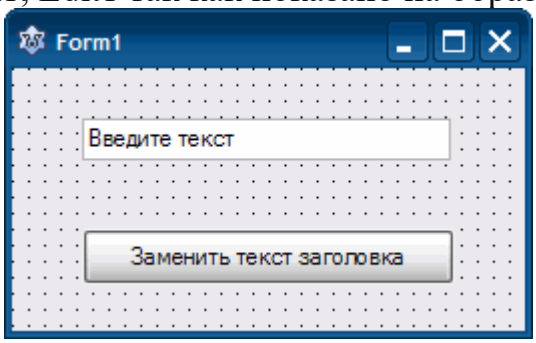
11. Сохраните все файлы проекта. Для этого выполните команду *Проект сохранить* или *Файл – Сохранить*

В предыдущем задании новые значения свойств для компонентов, размещенных на форме, были перечислены в таблице.

В следующих заданиях будут даны только текст задания и пример формы. От Вас ожидается, что вы, глядя на форму, сами догадаетесь, как должны быть установлены свойства компонентов.

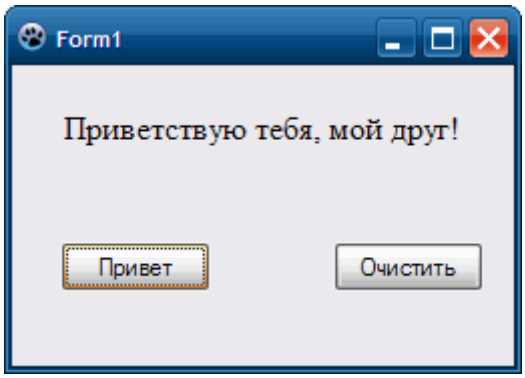
Задания для самостоятельного выполнения

Задание. Создайте приложение, разместите на форме компоненты: Buton1, Edit1 так как показано на образце.

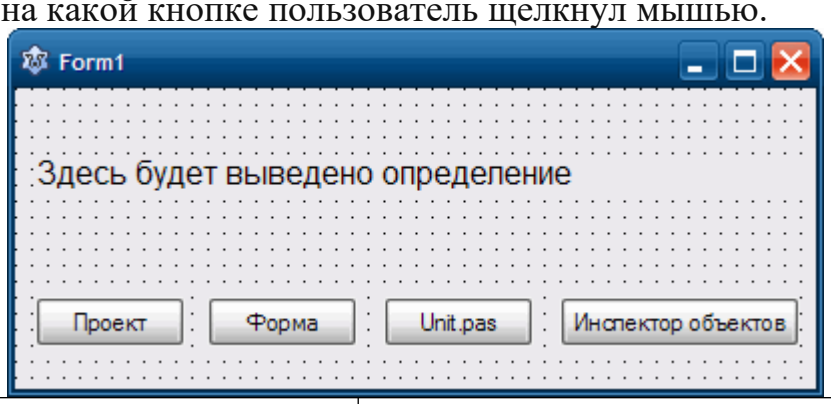


По щелчку на кнопке нужно заменить текст в заголовке окна на текст введенный пользователем в текстовое поле.

Задание. Создайте приложение, в результате работы которого при щелчке на кнопке привет в поле надписи выводится приветствие, при щелчке на кнопке Очистить сообщение исчезает.



Задание. Создайте приложение, в результате работы которого в поле надписи Label 1 выводится выводиться одно из сообщений, в зависимости от того, на какой кнопке пользователь щелкнул мышью.



Наименование кнопки	Отображаемый текст в надписи label1
Форма	Заготовка главного окна разрабатываемого приложения
Инспектор объектов	Окно, предназначенное для редактирования свойств объектов
Unit1.pas	Файл с программным кодом
Проект	Группа файлов, относящихся к данному приложению

Для объектов на форме установите следующие значения свойств:

Компонент	Свойство	Значение
Label1	Caption	Здесь будет выведено сообщение
	WordWrap	True
Form1	BorderIcons biMinimize biMaximize	False False

2) Компонент TImage

 Компонент Изображение (TImage) предназначен для вывода на форму графического рисунка. Он поддерживает многочисленные форматы графических файлов, включая .bmp, .ico, .jpg и т.д.

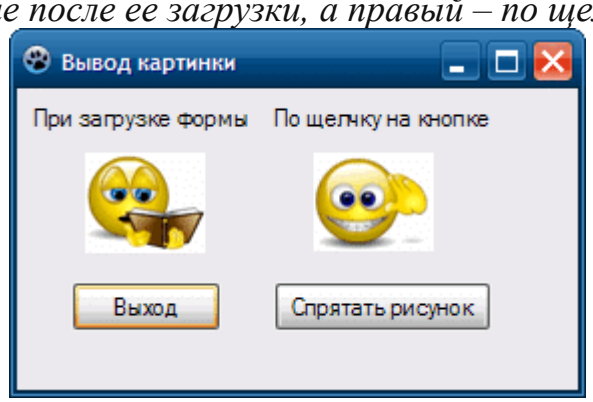
Для загрузки рисунка в поле Image на этапе разработки интерфейса приложения нужно присвоить свойству Picture файл, содержащий рисунок.

Для присвоения изображению файла с рисунком во время выполнения приложения используется метод LoadFromFile(), принадлежащий объекту Picture.

Например, для вывода в изображение imgIxample файла рисунка myPicture.jpg во время выполнения используется следующий оператор:

imgExample.Picture.LoadFromFile('myPicture.jpg')

Задание. *Создайте приложение «Вставка рисунка из файла», в результате работы которого левый рисунок будет отображаться на форме после ее загрузки, а правый – по щелчку на кнопке Вывести рисунок.*



Картинки для выполнения задания:

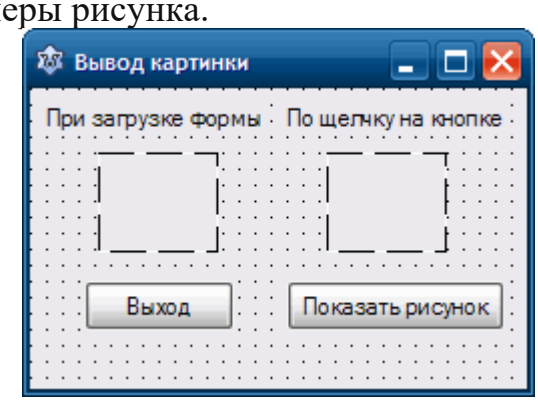
Сохранить их как файлы на вашем компьютере.

Ход выполнения

1. Создайте новое приложение *«Вставка рисунка из файла».*

Разместите на форме два изображения Image (вкладка Additional), две надписи (Label) и две кнопки (Button).

Значение свойства AutoSize для изображения установить True, чтобы размеры поля изображения автоматически изменялись, подстраивались под размеры рисунка.

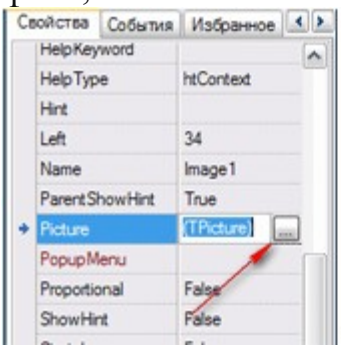


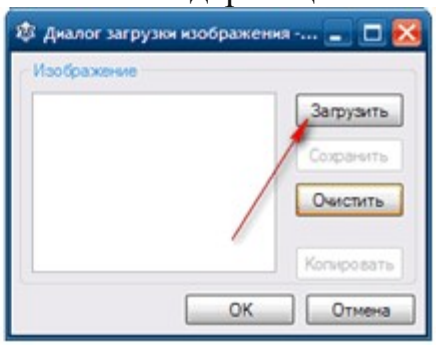
2. Скопируйте графические файлы с рисунками в папку проекта, иначе придется указывать полный путь к файлу.

3. Первый рисунок загрузите в Image1 на этапе проектирования интерфейса при настройке свойства Picture объекта Image1 в инспекторе объектов.

Для этого:

- Выделите элемент Image1 на форме, в окне Инспектор объектов.
- В строке Picture щелкните по кнопке с многоточием. Откроется окно *Диалог загрузки изображения.*
- В диалоговом окне щелкните по кнопке *Загрузить,* укажите файл, содержащий рисунок.





4. Второй рисунок загрузить в Image2 во время выполнения приложения с помощью метода LoadFromFile объекта Image2 после щелчка на кнопке Button2 (Показать рисунок).

Для этого:

- Откройте редактор кода двойным щелчком на элементе Button2. В программный код добавиться пустая процедура TForm1.Button2Click.
- Напишите программный код для TForm1.Button2Click.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  image2.Picture.LoadFromFile('pr7-2.gif');
end;
```

5. Проверьте работу приложения. Первая картинка должна отобразиться в поле Image1 сразу после загрузки приложения, вторая картинка – после щелчка на кнопке Показать рисунок.

6. Добавьте к предыдущему проекту кнопку Убрать рисунок (Button3). Наложите ее на кнопку Button2. Сделайте невидимой на момент загрузки приложения, для этого измените свойство Visible на False.

7. Измените программный код процедуры обработки щелчка на кнопке Button2 (Показать рисунок). Программный код выполняет следующие действия:

- Делает видимым компонент Image2;
- Загрузить рисунок в поле Image2;
- Делает невидимой кнопку Button2 (Показать рисунок);
- На ее месте делает видимой кнопку Button3 (Убрать рисунок).

Программный код процедуры:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  image2.Visible:=true;
  image2.Picture.LoadFromFile('pr7-2.gif');
  button2.Visible:=false;
  button3.Visible:=true;
```

end;

Для управления видимостью кнопок используйте свойство Visible.

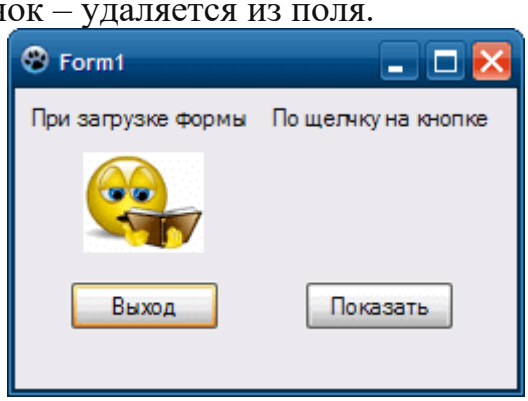
8. Напишите программный код процедуры обработки щелчка на кнопке Button3 (Убрать рисунок), который:

- прячет кнопку Button3 (Убрать рисунок);
- удаляет картинку из Image2;
- делает видимой кнопку Button2 (Показать рисунок).

Исходный текст процедуры:

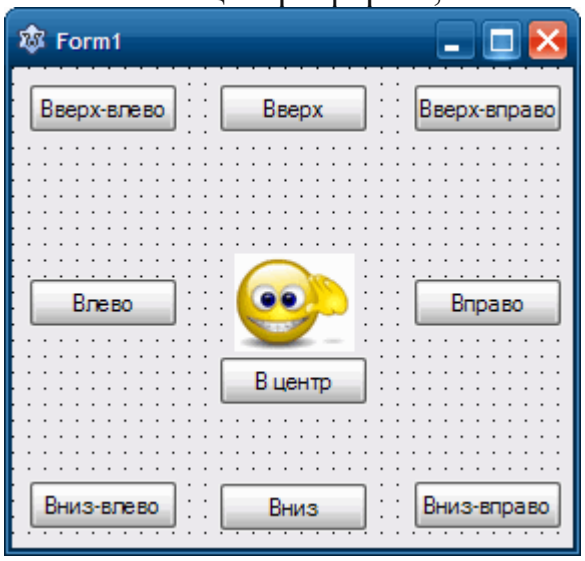
```
procedure TForm1.Button3Click(Sender: TObject);
begin
  button3.Visible:=false;
  button2.Visible:=true;
  image2.Visible:=false;
end;
```

8. Проверьте работу кнопок. После щелчка на кнопке Показать рисунок картина отобразится в поле Image2, а после щелчка на кнопке Убрать рисунок – удаляется из поля.



9. Сохраните проект.

Задание. Создайте приложение, которое над рисунком, расположенным в центре формы, выполняет действия, указанные на кнопках.



Ход выполнения

1. Создайте новое приложение, сохраните его в папке Перемещение картинки.

2. Установите на форме 9 кнопок (Button) и поле Image (изображение) как показано в образце.

3. Измените свойство Name кнопок в соответствии с надписями на них: — Vverh_Vlevo, vverch и т.д.

4. Написать программный код процедур обработки щелчков по кнопкам.

Напомню. Для того чтобы создать процедуру обработчик события, идущий по умолчанию, нужно выполнить двойной щелчок на элементе управления.

Примерный текст процедур обработчиков для кнопок Вверх-влево , Вверх-вправо и В центр приводится ниже:

```
procedure TForm1.btnVverh_VlevoClick(Sender: TObject);
begin
  image1.top:=10;
  image1.left:=10;
end;

procedure TForm1.btnVverh_VpravoClick(Sender: TObject);
begin
  image1.top:=10;
  image1.left:=form1.width-image1.width -10;
end;

procedure TForm1.btnCentrClick(Sender: TObject);
begin
  image1.top:=(form1.height-image1.height) div 2;
  image1.left:=(form1.width-image1.width) div 2;
end;
```

5. Программный код для остальных кнопок написать самостоятельно.

6. Проверить работу приложения.

3) Компоненты TBitBtn, TEdit и TMemo


Компонент TBitBtn (Кнопка с пиктограммой) находится на вкладке Addition, аналогичен компоненту TBbutton, но может содержать пиктограмму формата BMP или ICO.


Компонент TEdit (Поле ввода) – текстовое поле, которое в отличии от TLabel можно редактировать во время выполнения приложения. Текст, вводимый в поле ввода, хранится в свойстве Text.


Компонент TMemo предназначен для работы с многострочным текстом, который содержится в свойстве Lines.

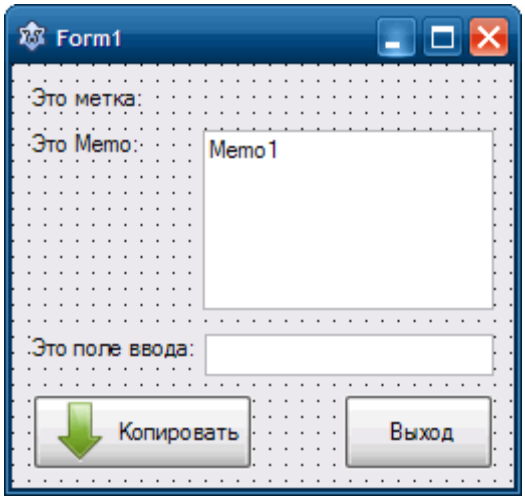
Компонент TBitBtn отличается от TButton тем, что на нем можно отображать пиктограммы. Кроме этого, этот компонент имеет и свои особые свойства.

Kind – задает тип кнопки. Имеются несколько предопределенных типов иконки с готовой пиктограммой и текстом.

Glyph – если вас не устраивают предлагаемые рисунки, вы можете выбрать другие. Будет открыто диалоговое окно, необходимо указать путь к этому рисунку.

Рассмотрим пример.

Задание. *Разработать приложение, в котором текст по щелчку на кнопке BitBtn1 будет копироваться из элемента Edit1 в метку Label1 и в Memo-поле.*



Сохранить рисунок стрелки, указанный ниже, как файл.

Ход выполнения

1. Загрузите среду программирования Lazarus, создайте приложение, сохраните во вновь созданную папку (Проект –Сохранить как …).

2. Скопируйте в папку проекта файл с изображением стрелки.

3. Создайте интерфейс по образцу. Разместите на форме 4 элемента TLabel, 1 элемент TMemo, 1 элемент TEdit, 2 элемента TBItBtn.

4. Настройте свойство элементов, которое отвечает за текст на поверхности элемента.

5. Чтобы поместить на кнопке BitBtn1 пиктограмму в виде стрелочки нужно:

1) Выделить элемент BitBtn1.

2) Загрузить картинку с помощью свойство Glyph.

Напишите программный код для процедуры TForm1.BitBtn1Click:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  label2.caption:= edit1.text;
  memo1.Text:=edit1.Text;
end;
```

6. Напишите программный код для процедуры TForm1.BitBtn2Click:

```
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
  close;
end;
```

7. Программа готова. Запустите приложение и проверьте его работу.

Введите в текстовое поле произвольный текст и нажмите кнопку Копировать. Введенный вами текст должен скопироваться в поле надписи и в memo-поле.

Нажмите на кнопку Выход – произойдет выход из программы.

8. Сохраните проект (Проект – Сохранить).

Вопросы для самоконтроля

1. Понятие класса и объекта.
2. Наследование.
3. Полиморфизм.
4. Инкапсуляция.

Лабораторное занятие 14. Создание простейших классов.

Тема 9. Метод: понятие, правила записи. Инкапсуляция, свойства класса, наследование.

Цель: изучить основы работы в Lazarus.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

1) Типы данных

Данные в языке Free Pascal

Для решения задач в любой программе выполняется обработка каких-либо данных. Данные хранятся в памяти компьютера и могут быть самых различных типов: целые и вещественные числа, символы, строки, массивы.

Типы данных определяют способ хранения чисел или символов в памяти компьютера. Они задают размеры ячейки, в которую будет записано то или иное значение, определяя тем самым его максимальную величину или точность задания.

Участок памяти (ячейка), в которой хранится значение определенного типа, называется переменной. У переменной есть имя (идентификатор), тип и значение.

Имя служит для обращения к области памяти, в которой хранится значение.

Во время выполнения программы значение переменной можно изменить.

В Lazarus каждая переменная перед использованием должна быть описана (объявлена). При объявлении переменной задается ее имя и тип. В оперативной памяти выделяется место для хранения переменной.

Для описание переменных используется служебного слова var.

Общий вид оператора:

Var имя_переменной: тип_переменной;

Например:

```
Var
a: integer; //Объявлена целочисленная переменная
b, c: real; //Объявлены две вещественные переменные.
```

Целочисленный тип данных

Целочисленные типы данных могут занимать в памяти компьютера один, два, четыре или 8 байтов.

Тип	**Диапазон**	**Размер (байт)**
Shortint	-128 .. 127	1
Integer	-32768 .. 32767	4
Longint	-2147483648 .. 2147483647	4
Byte	0 .. 255	1

Word	0 .. 65535	2

Вещественный тип данных

Внутренне представление вещественного числа в памяти компьютера отличается от представления целого числа. Вещественное число представлено в экспоненциальной форме mE±p, где m – мантисса (целое или дробное число с десятичной точкой), p – порядок (целое число).Чтобы перейти от экспоненциальной формы к обычному представлению числа, необходимо мантиссу умножить на десять в степени порядок.

Вещественное число может занимать от 4 до 10 байтов.

Вещественные типы	**Диапазон**	**Кол-во значащих цифр**	**Размер, байт**
Real	2.9e-39 ..1.7e+38	11-12	8
Single	1.5-e45 .. 3.4e+38	7-8	4
Double	5.0e-324.. 1.7e308	15-16	8
Extended	3.4e-4932 ..1.1e4932	19-20	8

Пример описания вещественных переменных:

```
Var
r1, r2: real; d: double;
```

Операции и выражения

Выражение задает порядок выполнения действий над данными и состоит из операндов (констант, переменных, обращений к функциям), круглых скобок и знаков операций.

Например a + b*sin(x).

В таблице представлены основные алгебраические операторы языка программирования Free Pascal.

Оператор	**Действие**
+	Сложение
—	Вычитание
*	Умножение
/	Деление
DIV	Целочисленное деление
MOD	Вычисление остатка от деления

Операторы целочисленной арифметики DIV и MOD применяются только к целочисленным операндам.

DIV позволяет получать целую часть результата деления одного числа на другое.

Например, 15 DIV 7 = 2.

Оператор MOD получает остаток от деления одного числа на другое.

Например, 15 MOD 7 = 1, Для задания нужного порядка выполнения операций в выражении можно использовать скобки.

Стандартные функции

В языке определены стандартные функции. Некоторые арифметические функции представлены в таблице ниже.

Обозначение	**Действие**
Abs(*n*)	*Абсолютное значение n.*
Sqrt(*n*)	*Квадратный корень из n.*
Sqr(n)	*Квадрат n.*
Exp(*n*)	*Экспонента n*.
Ln(n)	*Натуральный логарифм n.*
Random(n)	*Случайное целое число в диапазоне от 0 до n-1. (перед первым обращением к функции необходимо вызвать функцию* Randomize*, которая выполнит инициализацию программного генератора случайных чисел)*
Sin()	*Синус выраженного в радианах угла*
Cos()	*Косинус выраженного в радианах угла*
Arctan()	*Арктангенс выраженного в радианах угла*

Величина угла тригонометрических функций должна быть выражена в радианах. Для преобразования величины угла из градусов в радианы используется формула:

(α * 3.1415256) / 180,

где α – величина угла в градусах, 3.1415256 –число π.

Вместо константы 3.1415256 можно использовать стандартную именованную константу PI.

Функции преобразования

Функции преобразования типов часто используются при вводе и выводе информации

Например, для того чтобы вывести в поле вывода (компонент Label) диалогового окна значение переменной Real, необходимо преобразовать число в строку символов, изображающую данное число. Это можно сделать при помощи функции FloatToStr, которая возвращает строковое представление значения выражения, указанного в качестве параметра функции.

Основные функции преобразования типов

Обозначение	**Действие**

Chr(n)	*Символ, код которого равен n.*
IntToStr(k)	*Строка, являющаяся изображением целого k.*
FloatToStr(n)	*Строка, являющаяся изображением вещественного n.*
FloatToStrF(n,f,k,m)	*Строка, являющаяся изображением вещественного n. При вызове функции указывают:* f — *формат;* k – *точностью (общее количество цифр);* m — *количество цифр после десятичной точки.* Возможны следующие значения параметра Format: ffGeneral — общий числовой формат; ffFixed — фиксированный формат; ffCurrency — денежный формат.
StrToInt(s)	*Целое, изображением которого является строка* s.
StrToFloat(*s*)	*Вещественное, изображением, которого является строка* s.
Round(*n*)	*Целое, полученное путем округления* n *по известным правилам.*
Trunc(n)	*Целое, полученное путем отбрасывания дробной части* n.
Frac(n)	*Дробное, представляющее собой дробную часть вещественного* n.
Int(*n*)	*Дробное, представляющее собой целую часть вещественного* n.

Наиболее часто программа может получать исходные данные из окна ввода или из поля редактирования (компонент Edit). Для преобразования данных в числовой тип используют соответствующую функцию.

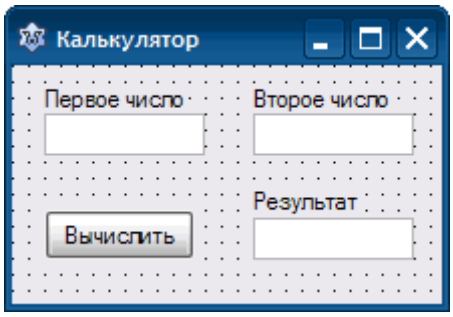
Например, для преобразования в вещественный тип используется оператор:

Funt:= StrToFloat(Edit1.Text);

Основные сведения об переменных, числовых типах данных и функциях преобразования типов мы повторили. Теперь переходим к практической части нашего занятия.

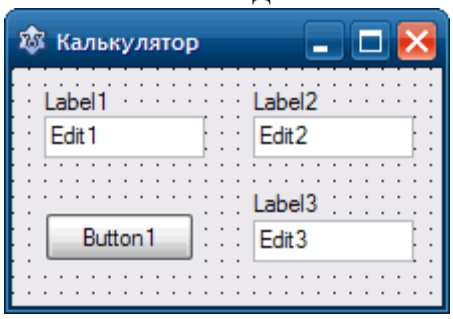
Проект «Калькулятор»

Задание. Создать приложение для вычисления суммы двух чисел. Окно программы должно выглядеть так, как показано на рисунке.



Ход выполнения

1. Создайте приложение. Сохраните проект в папке Калькулятор.
2. Создайте интерфейс по образцу.



3. Настройте свойства объектов в соответствии с таблицей.

Компонент	**Свойство**	**Значение**
Form1	Caption	Калькулятор
Label1	Caption	Первое число
Label2	Caption	Второе число
Label3	Caption	Результат
Edit1	TextName	ПустоEditNum1
Edit2	TextName	ПустоEditNum2
Edit3	TextName	ПустоEditResult
Button1	Coption	Вычислить

4. Написать процедуру обработки щелчка на кнопке Вычислить (btnMath). Для этого выполните двойной щелчок на кнопке Вычислить. Это приведет к созданию процедуры TForm1.Button1Clic в разделе implementation:

5. procedure TForm1.Button1Click(Sender: TObject);
6. begin

end;

Понятно, что созданная процедура не содержит ни одной команды. Ваша задача – заполнить шаблон операторами. Все команды, указанные в процедуре между begin и end, будут выполнены при щелчке на кнопке Выполнить.

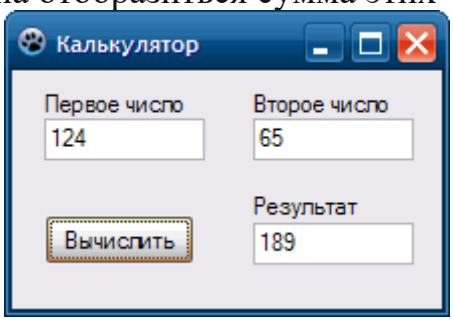
В нашем случае процедура TForm1 . But ton1Cl ick будет иметь вид:

```
procedure TForm1.Button1Click(Sender: TObject);
var num1, num2, result: integer;
```

```
begin
  num1:=StrToInt(editNum1.text);
  num2:=StrToInt(editNum2.text);
  result:=num1+num2;
  EditResult.text:=IntToStr(result);
end;
```

Обратите внимание, что были написаны всего 5 команд, предназначенных для решения поставленной задачи. Остальной текст в окне редактора создается автоматически

7. Сохранить проект. Проверьте работу приложения. Введите в первые два поля целые числа, нажмите кнопку Вычислить. В поле результат должна отобразиться сумма этих чисел.



2) Организация ввода и вывода данных

Другой способ организации ввода и вывода данных – использование встроенных диалоговых окон InputBox, ShowMessage. Эти диалоговые окна не устанавливаются программистом на форму во время разработки. Операторы их активации нужно вставлять в программный код.

Ввод данных

Функция InputBox() выводит на экран диалоговое окно, в котором можно ввести данные.

Аргументами этой функции являются три строки, а значением функции – строка введенная пользователем.

В общем виде строка программного кода с использованием функции InputBox выглядит так:

Переменная := InputBox(*'Заголовок'*, *'Подсказка'*, *'Значение по умолчанию'*);

где*:*

Переменная – переменная строкового типа, значение которой должно быть получено от пользователя;

Заголовок – текст заголовка окна;

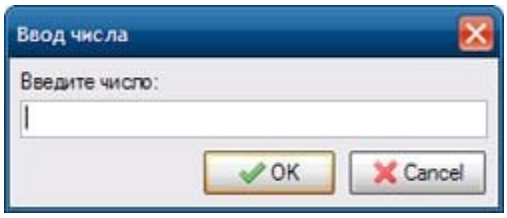
Подсказка – текст поясняющего сообщения;

Значение по умолчанию – текст, который будет находиться в поле ввода, когда окно появиться на экране.

Например,

n := InputBox(*'Ввод числа'*, *'Введите число:'*, *' '*);

Результат показан на рисунке:



Если пользователь щелкнет по кнопке OK, то значением функции станет строка, введенная пользователем в текстовое поле. Если пользователь щелкнет по кнопке Cancel, то значением функции станет строка «*Значение по умолчанию*».

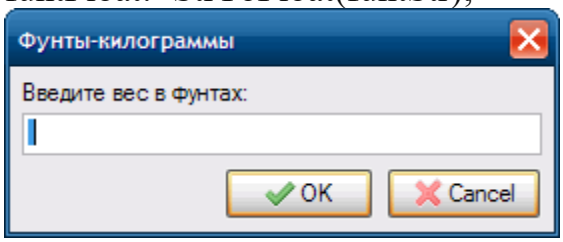
Значение функции InputBox всегда строкового типа (String), поэтому в случае, если нужно ввести число, то введенная строка должна быть преобразована в число при помощи соответствующей функции преобразования.

В качестве примера возьмем задачу пересчета веса из фунтов в килограммы

Ввод исходных данных из окна ввода и последующее преобразование данных может выглядеть так:

funtStr:= InputBox(‘Фунты-килограммы’,’Введите вес в фунтах:’,’ ‘);

funtFloat:=StrToFloat(funtStr);



Вывод данных

Результат работы программы чаще всего выводят в окно сообщения ShowMessage или в поле вывода (компонент Label).

Вывод в окно сообщения ShowMessage

Для вывода результата используется процедура ShowMessage(). Она выводит на экран диалоговое окно с текстом и командной кнопкой OK.

Общий вид инструкции вызова процедуры ShowMessage:

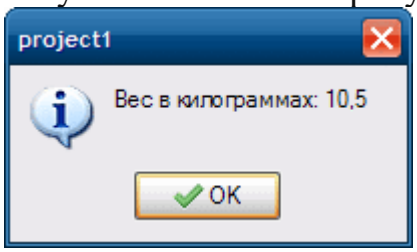
ShowMessage(‘*Сообщение’*);

где *Сообщение* – текст, который будет выведен в окне.

Например, для того чтобы вывести результат в программе пересчета веса из фунтов в килограммы, можно использовать следующую строку кода:

ShowMessage(‘Вес в килограммах: ‘+ FloatToStr(kg));

Результат показан на рисунке.



Вывод в поле вывода (Label)

Компонент TLabel(Поле вывода), в который будет осуществляться вывод, устанавливают на форме во время разработки. Содержание этого поля определяется значением свойства Caption.

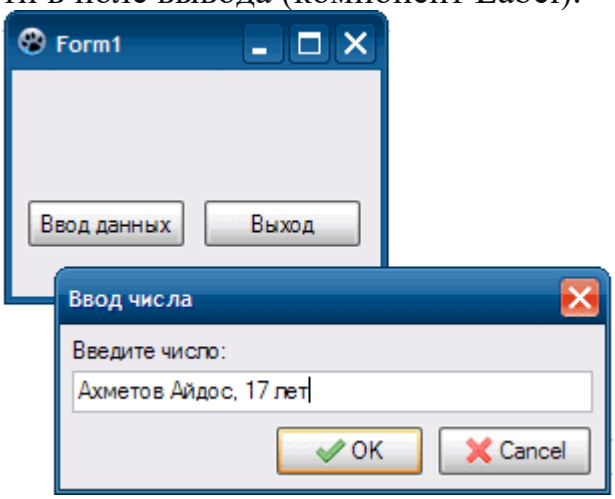
Для того чтобы вывести результаты в это поле, нужно в программном коде поместить оператор присваивания, который будет изменить значение свойства Caption на нужное вам значение.

Например, для того чтобы вывести результат в поле вывода Label1 в рассмотренной выше задаче, нужно использовать следующий оператор присваивания:

Label1.Caption:=FloatToStr(kg) + ‘кг‘;

Практическая работа

Задание. Поместить на форму две кнопки: Ввод данных и Выход. Пользователь должен ввести фамилию, имя и возраст. Для ввода данных использовать функцию InputBox. По окончании ввода анкетные данные вывести в поле вывода (компонент Label).



Ход выполнения

1. Создать новый проект. Папку проекта назвать Анкетные данные.

2. Разместить на форме две кнопки и надпись так, как показано на рисунке выше.

3. Написать программный код для кнопки Ввод данных.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
label1.Caption:=InputBox('Ввод анкетных данных', 'Введите фамилию, имя,
возраст:','');
end;
```

4. Написать программный код для кнопки Выход

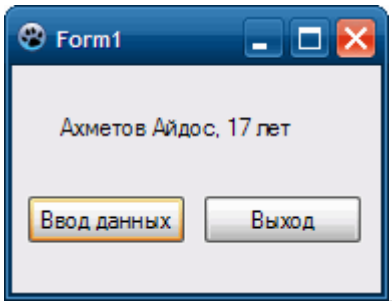
```
procedure TForm1.Button2Click(Sender: TObject);
begin
Close;
end;
```

5. Сохранить проект. Проверить работу приложения.

После щелчка на кнопке Ввод данных должно появиться диалоговое окно. Введите в него исходные данные.

После нажатия на клавишу Enter введенные данные должны отобразиться на форме.

Результат показан на рисунке.



В этом уроке мы рассмотрели способы организации ввода и вывода данных в среде Lazarus. Для ввода данных мы использовали диалоговое окно InputBox , для вывода – диалоговое окно ShowMessage и компонент формы TLabel (Поле вывода) **.**

Компонент TLabel устанавливается на форме во время разработки, диалоговые окна выводятся в отдельном окне во время выполнения приложения, не занимают место на форме. Для их вывода нужен соответствующий программный код.

Вопросы для самоконтроля

1. Метод: понятие, правила записи.
2. Инкапсуляция как управление доступом к данным.
3. Свойства класса: понятие, виды, правила записи.

Лабораторное занятие 15. Создание классов, иерархически связанных между собой.

Тема 10. Иерархия классов. Интерфейсы.

Цель: изучить основы работы в Lazarus.

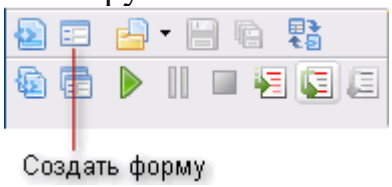
Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

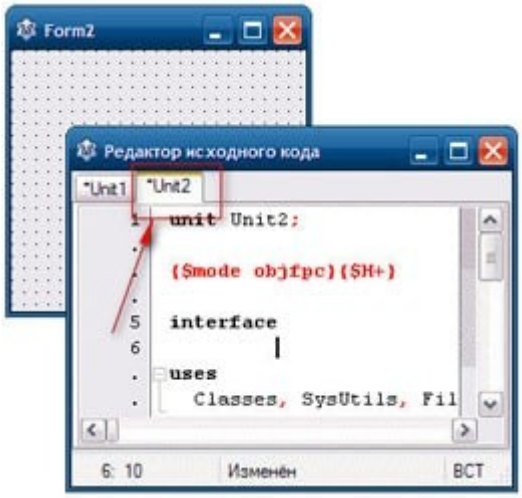
1) Компонент TRadioGroup. Приложение с несколькими формами

Добавление новой формы

Компонент форма является объектом, отсутствующим на палитре компонентов. Чтобы добавить новую форму в проект, нужно выбрать команду **Файл – Создать форму** или щелкнуть кнопку **Создать форму** на панели инструментов.



Появиться новая пустая форма. Называться она будет Form2, а соответствующий ей файл с исходными текстами добавиться в Редактор кода на новую вкладку Unit2.



После добавления новой формы, проект нужно сохранить.

Для показа форм можно использовать один из двух методов: Show или ShowModal.

Метод Show предназначен для показа формы в обычном окне, а ShowModal — для показа формы в модальном окне.

Различие между этими двумя видами окон состоит в том, что между обычными окнами можно перемещаться произвольным способом, а перейти в другое окно из модального окна можно только после его закрытия.

Показ формы как обычного окна

Чтобы вызвать форму в обычном окне используют ее метод Show. Он показывает форму, перемещает ее на передний план экрана и делает активной.

В примере показан вызов формы Form3 в обычном окне после щелчка на кнопке Button1.

procedure TForm1.Button1Click(Sender: TObject);

```
begin
  Form2.Show;
end;
```

Показ формы как модального окна

Чтобы вызвать форму в модальном окне, надо использовать метод ShowMoal. Переключиться из модального окна на другие окна не удастся, пока оно не будет закрыто.

Пример показа формы Form3 в модальном окне после щелчка на кнопке Button2.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.ShowModal;
end;
```

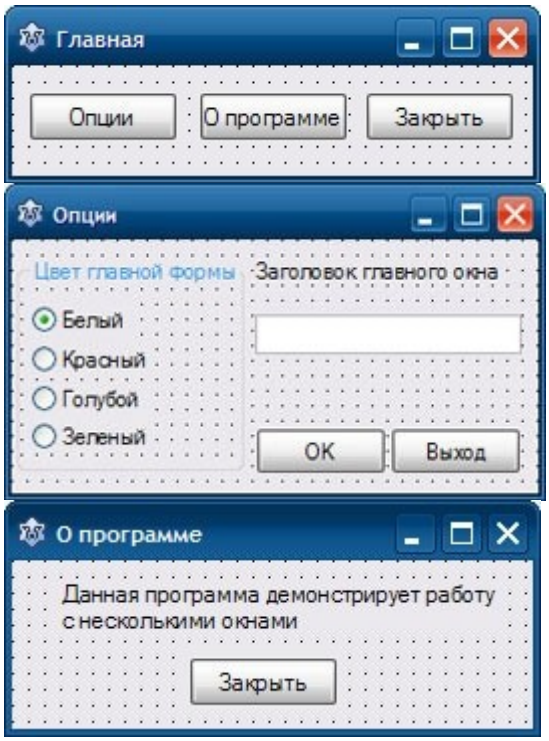
Модальные окна хорошо подходят для задания всевозможных настроек, выполнения ввода промежуточных значений, отображения результатов.

Теперь у нас есть все необходимое, для того чтобы создать свое первое приложение с несколькими формами.

Проект «Три формы»

Задание. Создать приложение с тремя формами: Главная, Опции и О программе. Форму Опции вызывать в обычном окне. Для вызова формы О программе использовать модальное окно.

На рисунке показаны главная форма и подформы нашего нового проекта.



Ход выполнения проекта

1. Создайте новое приложение, сохраните файлы проекта в папке «Три формы».

2. Разместите на форме Form1 3 кнопки, измените свойства объектов в соответствии с таблицей.

Компонент	Свойство	Значение

Форма	Name Caption	frmGeneral Главная
Кнопка 1	Caption	Опции
Кнопка 2	Caption	О программе
Кнопка 3	Caption	Закрыть

3. Создайте формуForm2. Для этого выберите в меню Файл команду Создать форму. На экране появиться новая форма Form2, в редакторе кода – новая вкладка Unit2.

4. Установите на форме компонент RadioGroup для выбора цвета, надпись, поле вывода, две кнопки. Настройте свойства объектов в соответствии с таблицей.

Компонент	Свойство	Значение
Форма	Name Caption	frmOptions Опции
RadioGroup	Caption Items	Цвет главной формы Ввести список
Label1	Caption	Заголовок главного окна
Edit1	Text	Пусто
Button1 Button2	CaptionCaption	ОК Выход

5. После размещения на форме компонента TradioGroup, входящие в него переключатели задаются перечислением их названий. Эти названия вводятся в свойство Items.

6. Так как требуется ввести не одну строку, а несколько, для их ввода предусмотрен специальный редактор, который вызывается щелчком на специальной кнопке  , расположенной справа в строке, описывающей свойство Items.

7. Большая текстовая область окна редактора предназначена для ввода названий переключателей по одному в каждой строке. Переход в начало следующей строки осуществляется при нажатии на клавиши Shift+Enter.

8. После окончания ввода списка, щелкните по кнопке ОК, и внешний вид объекта RadioGroup1 на форме сразу изменится.

9. Создайте еще одну форму — Form3, выбрав команду Файл-Создать форму. На экране появиться новая форма Form3 а в редакторе кода – новая вкладка Unit3.

10. Разместите на Form3 объекты Надпись и Кнопка. Настройте свойства объектов.

Компонент	Свойство	Значение
Форма	Name Caption	frmAbout О программе
Button1	Caption	Выход

11. Программный код для формы Главная (модуль Unit1) В модуле Unit1 в разделе Implementation необходимо записать директиву uses:

uses Unit2, Unit3;

Это необходимо для того чтобы модули Unit2, Unit3 форм Опции и О программе были видимы в главном модуле Unit1.

12. Написать обработчики событий для кнопок формы Главная. Первая кнопка формы Главная (кнопка Опции) вызывает форму Опции в обычном окне с помощью метода Show.

```
13.  procedure TfrmGeneral.Button1Click(Sender: TObject);
14.  begin
15.    frmOptions.Show;
end;
```

16. Первая кнопка формы Главная (кнопка О программе) вызывает форму О программе в модальном окне с помощью метода ShowModal.,/p>

```
17.  procedure TfrmGeneral.Button2Click(Sender: TObject);
18.  begin
19.    frmAbout.ShowModal;
end;
```

20. Третья кнопка формы Главная (Кнопка ОК) закрывает главное окно.

```
21.  procedure TfrmGeneral.Button3Click(Sender: TObject);
22.  begin
23.    Close;
end;
```

24. Открываем программный код формы Опции (модуль Unit2).В модуле Unit2 в разделе implementation записать директиву uses:

uses Unit1;

Это необходимо для того чтобы главный модуль Unit1 формы Главная был видим в этом модуле.

25. Создать обработчик загрузки формы Опции, в который записать программный код, передающий текст заголовка главной формы в поле Edit1.

```
26.  procedure TFrmOptions.FormCreate(Sender: TObject);
27.  begin
28.    frmOptions.Edit1.text:=frmGeneral.Caption;
end;
```

29. Кнопка ОК формы Опции. По щелчку на этой кнопке будет происходить изменение цвета главной формы.

```
30.  procedure TFrmOptions.Button1Click(Sender: TObject);
31.  begin
32.  if radioGroup1.ItemIndex=0 then frmGeneral.color:=clWhite;
33.  if radioGroup1.ItemIndex=1 then frmGeneral.color:=clRed;
```

```
34.        if radioGroup1.ItemIndex=2 then frmGeneral.color:=clBlue;
35.        if radioGroup1.ItemIndex=3 then frmGeneral.color:=clGreen;
end;
```

36. Кнопка Закрыть формы Опции. По щелчку на этой кнопке закрывается окно Опции.

```
37.        procedure TFrmOptions.Button2Click(Sender: TObject);
38.        begin
39.          close;
end;
```

40. Переходим в программный код формы О программе (модуль Unit3).В модуле Unit3 в разделе implementation записать директиву uses.

uses Unit1;

Модуль Unit1 формы Главная должен был видим в этом модуле.

41. Кнопка ОК формы О программе закрывает окно.

```
42.        procedure TfrmAbout.Button1Click(Sender: TObject);
43.        begin
44.          Close;
end;
```

45. Проект готов. Сохраните проект и проверьте его работу.

В этом уроке мы создали приложение состоящее из трех форм, использовали разные методы для вызова форм: простое окно и модальное окно.

В нашем приложении мы впервые применили компонент RadioGroup (Группа переключателей), использовали условный оператор для обработки выбора пользователя.

2) Операторы повторения

В программе цикл может быть реализован при помощи конструкций for, while и repeat.

Оператор цикла for … do

Оператор FOR используется в том случае, когда некоторую последовательность действий надо выполнить несколько раз, причем число повторений заранее известно.

Общий вид оператора:

For <счетчик> := <нач_знач> to <кон_знач> **do**

Begin

<тело цикла>

End;

где:

счетчик – переменная-счетчик числа повторений; нач_знач – выражение, определяющее начальное значение счетчика цикла; кон_знач – выражение, определяющее конечное значение счетчика цикла; тело цикла — операторы, которые будут повторяться.

Переменная-счетчик, выражения нач_знач и кон_знач должны быть целого типа.

Количество повторений цикла можно вычислить по формуле: кон_знач – нач_знач +1.

Если между begin и end находится только один оператор, то begin и end можно не писать.

Например, в результате выполнения следующего программного кода:

```
tab1:=' ';
for i:= 1 to 5 do
begin
tab1:= tab1 + IntToStr(i) + ' ' + IntToStr(i*i) + chr(13);
end;
```

переменная tabl будет содержать изображение таблицы квадратов чисел.

Если в операторе for вместо слова to записать downto, то после очередного выполнения тела цикла значение счетчика будет не увеличиваться, а уменьшаться.

Операторы цикла с условием while и repeat

Операторыwhile и repeat используются в том случае, когда некоторую последовательность действий надо выполнить несколько раз, причем необходимое число повторений во время разработки программы не известно и может быть определено только во время работы программы.

Оператор цикла с предусловием while .. do

Общий вид оператора:

```
While условие do
begin
<тело цикла>
end;
```

где *условие* – выражение логического типа, определяющее условие выполнения тела цикла. Операторы тела цикла выполняются, пока условие не станет ложным (False).

Оператор цикла с постусловием и repeat… until

Общий вид оператора:

```
repeat
  <тело цикла>
until условие
```

где *условие* – выражение логического типа, определяющее условие завершения цикла.

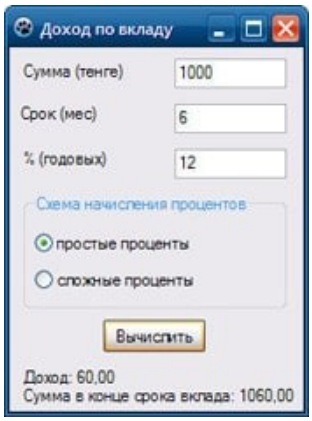
Операторы *тела цикла* выполняются до тех пор, пока *условие* не станет истинным (False).

Обратите внимание на различия. В операторе **while … do** выход из цикла происходит тогда, когда условие станет ложным, а в операторе **repeat … until –** когда условие станет истинным.

Более подробно об операторах цикла можно посмотреть в уроках по языку Паскаль.

Практическая работа

Задание. Напишите программу, определяющую доход по вкладу с учетом выбранных простых или сложных процентов. Простые проценты начисляются по окончании срока вклада, сложные проценты начисляются ежемесячно и прибавляется к сумме вклада.



Ход выполнения работы

1. Загрузите Lazarus, создайте новый проект. Сохраните файлы проекта в папке Доход по вкладу.

2. Создайте интерфейс по образцу.

3. Напишите обработчик события для кнопки Вычислить.

```
procedure TForm1.Button1Click(Sender: TObject);
var
sum:real; // сумма вклада
pr: real; // процентная ставка
period: integer; // срок вклада
profit: real; //доход по вкладу
sum2: real; //сумма при вычислении
//методом сложных процентов
i: integer;
begin
sum:=StrToFloat(Edit1.text);
pr:=StrToFloat(Edit2.text);
period:=StrToInt(Edit3.text);
if RadioGroup1.ItemIndex=0 then
// Выбран переключатель Простые проценты
profit:=sum*(pr/100/12)*period
else
// Выбран переключатель Сложные проценты
begin
sum2:=sum;
for i:=1 to period do
sum2:= sum2+sum2*(pr/100/12);
profit:=sum2-sum;
end;
sum:=sum+profit;
Label4.Caption:='Доход: '+FloatToStrF(profit,ffFixed,8,2)
+#13+'Сумма в конце срока вклада: '+FloatToStrF(sum, ffFixed,8,2);
end;
```

4. Проверьте работу приложения.

Вопросы для самоконтроля

1. Иерархия классов.
2. Интерфейсы.

Лабораторное занятие 16. Создание классов для обработки массива данных.

Тема 11. Модификаторы доступа к элементам класса. Методы классов.

Цель: изучить основы работы в Lazarus.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

1) Компонент Меню (TMainMenu)

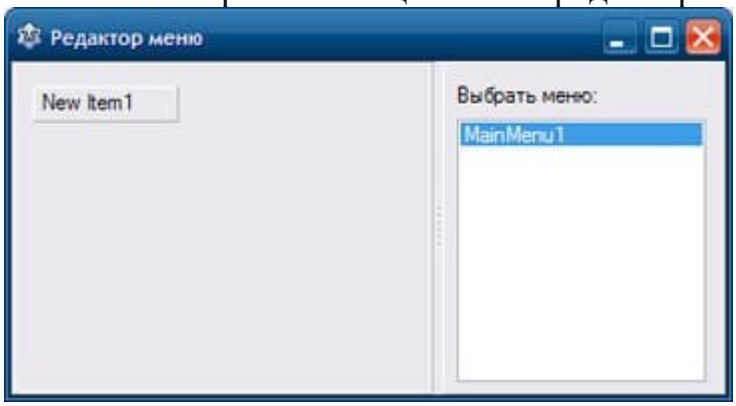
Компонент TMainMenu предназначен для добавления к программе главного меню, без которого не обходится практически ни одно из приложений Windows.

Чтобы добавить меню, надо выбрать на панели компонентов Standart (Стандартные) компонент TMainMenu и поместить его на форме в произвольном месте.

Компонент TMainMenu не визуальный, в отличии от визуальных компонентов TEdit и TLabel, в точности соответствующих своему внешнему виду в работающей программе.

Это означает, что хотя он виден на форме как небольшой значок, в окне созданной программы в таком виде он не появится. Представление его на форме в миниатюрном виде просто указывает на наличие в программе объекта, ответственного за меню.

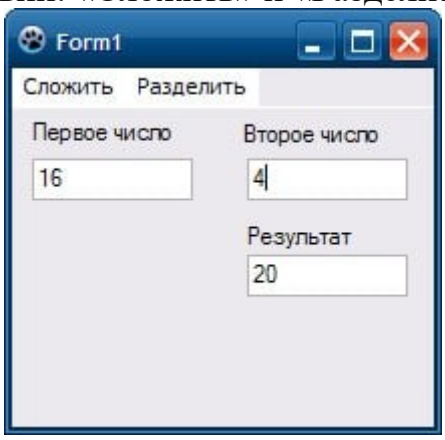
Пункты меню содержатся в свойстве Items. Чтобы начать формирование пунктов меню достаточно дважды щелкнуть по компоненту на форме или нажать на кнопку с многоточием в свойстве Items компонента в окне Свойства. Откроется специальный редактор меню.



Рассмотрим применение нового компонента на конкретном примере.

Практическая работа

Задание. Добавить главное меню в созданное Вами ранее приложение Калькулятор. В горизонтальное меню включить названия арифметических действий: «Сложить» и «Разделить».



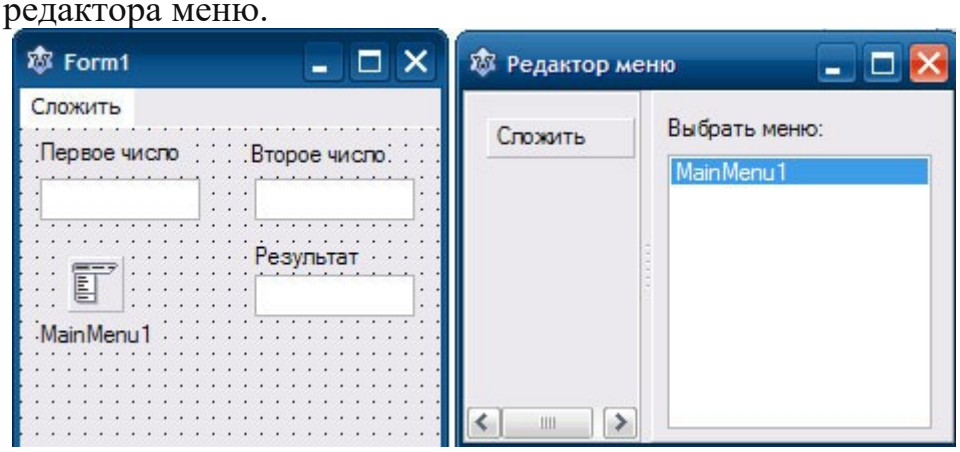
Ход выполнения проекта

1. Откройте в среде программирования Lazarus проект Калькулятор, созданный в 5 уроке.

2. Поместите компонент MainMenu на форму.

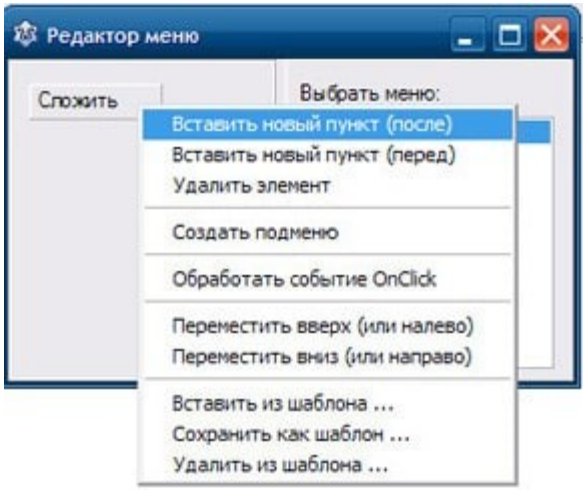
3. Чтобы начать формирование пунктов меню дважды щелкнуть по компоненту TMainMenu1 на форме или нажмите на кнопку с многоточием в свойстве Items компонента в инспекторе объектов.

На форме в горизонтальном меню появится первый пункт, который по умолчанию имеет имя New Item1 (Новый Элемент1). Откроется специальное окно редактора меню.



4. В инспекторе объектов в свойство Caption введите имя первого пункта меню Сложить.

5. Чтобы создать следующий пункт меню, установите курсор на текущий элемент меню и откройте контекстное меню правым щелчком. Выберите «Вставить новый пункт (после)».



На форме отобразится следующий пункт горизонтального меню.

6. В инспекторе объектов в свойство Caption введите название второго пункта меню: Разделить.

7. Теперь осталось написать программный код, который будет выполняться при выборе пунктов меню.

Напишем общую для всех пунктов меню процедуру Znak, параметром которой будет символ арифметической операции «+» или «/». Процедура будет получить числовые значения из полей EditNum1 и EditNum2, производить указанную в качестве параметра операцию и выводить полученный результат в поле EditResult.

Текст процедуры нужно поместить в раздел Implementation после директивы {$R *.lfm}.

```
procedure Znak(zn:char);
label metka;
var result, num1, num2:real;
begin
num1:=StrToFloat(form1.editNum1.text);
num2:=StrToFloat(form1.editNum2.text);
case zn of
'+': result:=num1 + num2;
'/': if num2<> 0 then result:=num1/num2;
else
begin
showMessage('Делить на 0 нельзя!');
form1.editNum2.text := '';
form1.editNum2.SetFocus;
form1.editResult.text := '';
goto metka;
end;
end; // casse
form1.EditResult.Text := FloatToStr(Result);
metka:
end;
```

8. Напишем обработчик для пункта меню Сложить. Для этого дважды щелкните на данном пункте меню.

Программный код будет иметь следующий вид:

```
procedure TForm1.MenuItem1Click(Sender: TObject);
begin
znak('+');
end;
```

В программном коде вызывается написанная нами процедура Znak, в которой в качестве фактического параметра передается знак арифметического действия.

9. Напишем обработчик для пункта меню Разделить.

```
procedure TForm1.MenuItem1Click(Sender: TObject);
begin
znak('/');
end;
```

10. Проект готов. Проверьте работу приложения. Для этого введите произвольные числа в поля ввода, выберите в горизонтальном меню нужное действие, проверьте результат.

2) Стандартные диалоги. Создание текстового редактора

Стандартные диалоги – это диалоговые окна, общие для большинства приложений Windows. Например, когда пользователь сохраняет файл в редакторе Word, он использует диалоговое окно Save As … (Сохранить как …). Когда он сохраняет рабочую книгу в Microsoft Excel, он также использует диалоговое окно Save As …

Операционная система Windows стандартизирует и значительно облегчает создание таких диалоговых окон. Программисту нет необходимости «изобретать колесо» каждый раз, когда понадобится диалоговое окно, в котором пользователь будет сохранять или открывать файл.

Вместо этого программист попросту устанавливает стандартный компонент на форму и задает ему необходимые свойства.

В составе Lazarus поставляется 14 диалоговых компонентов. Все они находятся на вкладке Dialogs. Как и MainMenu, в среде Lazarus диалоговые компоненты являются не визуальными, они присутствуют на форме в виде значков, однако во время выполнения они невидимы.

В этом уроке мы рассмотрим диалоговые компоненты: TOpenDialog, TSaveDialog, TFontDialog.

Для активизации диалогового окна используется метод Execute(). Он выполняет открытие соответствующего окна и возвращает значение True, если пользователь щелкнул по кнопке ОК.

Компоненты TOpenDialog и TSaveDialog

Компонент TOpenDialog предназначен для выбора файла с целью последующего открытия, а компонент TSaveDialog — для последующего сохранения файла.

свойства приведены в таблице.

Свойства	Описание
DefaultExt	Расширение файла по умолчанию Добавляется в конец имени файла, если расширение не указано явно.
Title	Заголовок диалогового окна
FileName	Выбранное пользователем имя файла вместе с полным путем поиска.
Filter	Список расширений файлов, в соответствии с которыми отбираются имена файлов для отображения в диалоговом окне при открытии файла . При сохранении файла выбранное из списка расширение добавляется к имени файла.
FilterIndex	Порядковый номер строки в списке расширений.

 Эти компоненты не предназначены для выполнения конкретных действий: загрузки файла, записи в файл. Они применяются только для получения от пользователя желаемых настроек, например ввода полного имени файла вместе с путем поиска.

Рассмотрим использование диалоговых окон на примере создания простейшего Текстового редактора.

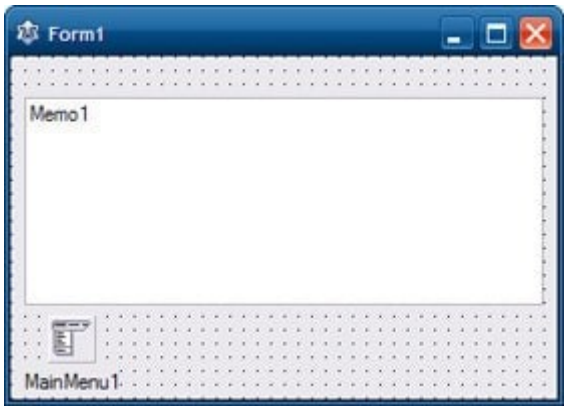
➡**Задание.** Создайте приложение Текстовый редактор. В приложении должно быть меню, содержащее пункты: Файл -Открыть, Файл-Сохранить, Шрифт. Приложение открывает текстовый файл на диске с использованием компонента TOpenDialog, записывает текст из файла в объект Memo на форме. После редактирования и форматирования текст необходимо сохранить. Для выбора файла для сохранения использовать компонент TSaveDialog, при форматировании шрифта — компонент TFontDialog.

 При работе с текстовыми файлами следует иметь в виду, что при выводе в TMemo файла с русским текстом, в Windows буквы могут отображаться некорректно. Это вызвано тем, что текстовые файлы в ОС Windows имеют кодировку CP-1251. Перед выводом текст необходимо преобразовать в кодировку UTF8.

Для преобразования символов из одной кодировки в другую можно использовать функции: SysToUTF8() и UTFToSys().

Ход выполнения проекта

1. Создайте форму и разместите на ней компоненты MainMenu, Memo1. Компонент MainMenu – невизуальный компонент, его значок можно поместить в произвольном месте.

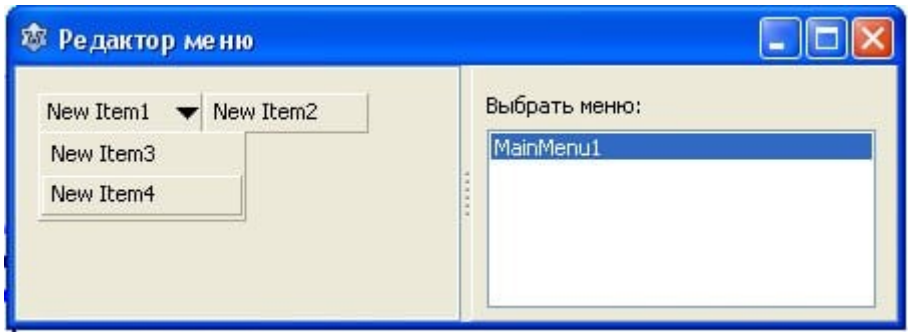


2. Чтобы начать формирование пунктов меню, дважды щелкните по компоненту TMainMenu1. Откроется специальное окно Редактор меню.

Определите пункты меню.

Меню первого уровне содержит два пункта: New Item1 и New Item2 . Первый пункт создается автоматически. Чтобы на этом уровне добавить второй пункт, откройте контекстное меню существующего пункта и выполните команду Вставить новый пункт (после). Появиться новый пункт New Item2.

Меню New Item1 содержит пункты : New Item3 и New Item4. Откройте контекстное меню пункта New Item1 и выполните команду Создать подменю. В подменю появиться пункт New Item3. Для него откройте контекстном меню и выполните команду Создать новый пункт (после). Появиться пункт New Item4.



3. Значения свойств установите в соответствии с таблицей.

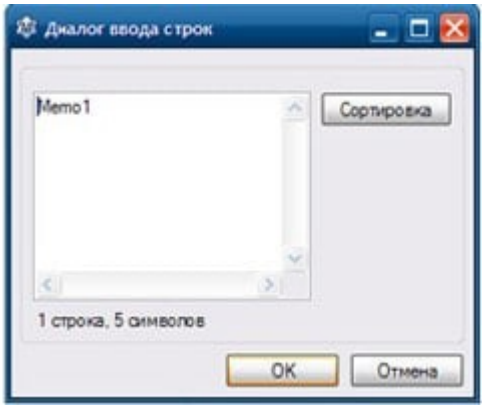
Компонент	Свойство	Значение
MenuItem1	Caption	Файл
MenuItem2	Caption	Шрифт
MenuItem3	Caption	Открыть
MenuItem4	Caption	Сохранить
Form1	Caption	Текстовый редактор
Memo1	ScrollBars	ssVertical

4. Удалите текст «Memo1» из окна Memo1.

Для этого:

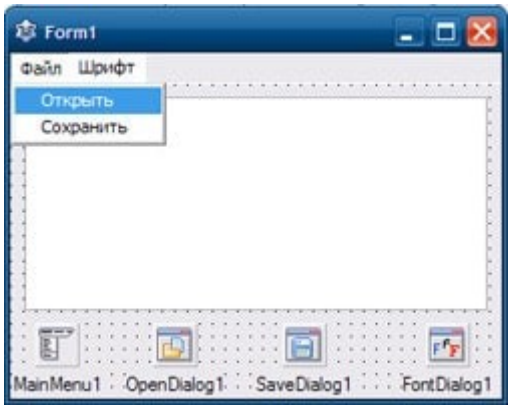
- выберите в окне Инспектор объектов объект Memo1;

- на странице Свойства в строке Lines дважды щелкните на значении String или на кнопке с многоточием для формирования и редактирования текста;
- в окне Диалог ввода строк удалите текст «Memo1» и щелкните мышью на кнопке ОК.



5. Выберите в палитре компонентов вкладку Dialogs и поместите на форму компоненты OpenDialog, SaveDialog, FontDialog.

Эти компоненты не визуальные, разместить их в нижней части формы рядом со значком TMainMenu.

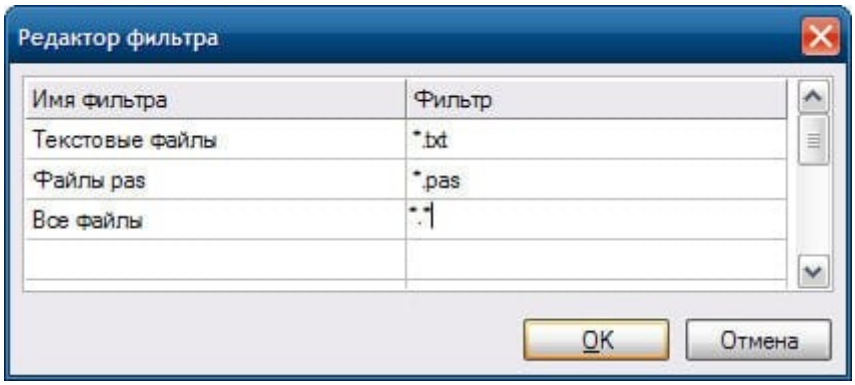


6. Настройте свойства компонентов OpenDialog, SaveDialog

Компонент	Свойство	Значение
OpenDialog1	Title	Открыть
SaveDialog1	Title	Сохранить

7. Чтобы реализовать выбор типа файла при открытии файла в окне диалога:

- выберите в окне Инспектор объектов объект OpenDialog1;
- на странице свойства дважды щелкните мышью по списку значений свойства Filter.
- в окне Редактор фильтра (Filter Edit) задайте фильтры для выбора типа и расширения файла.



8. Чтобы установить в качестве расширения файла первый вариант (.txt), задайте для свойства OpenDialog1.FilterIndex значение 1.

9. Для диалога SaveDialog задайте свойство DefaultExt = txt (чтобы расширение txt автоматически добавлялось к создаваемому файлу).

10. В разделе implementation после директивы { TForm1 } запишите программный код процедуры Ansi_Memo, реализующей загрузку текста из файла с системной кодировкой (кодировка Ansi) в memo-поле (кодировка UTF8), а также процедуры Memo_Ansi, сохраняющей текст из memo-поля в файле на диске.

```
procedure Ansi_Memo(File_Ansi: string);
// Загрузка текста из файла в memo-поле на форме
var
tfile: TStringList;
str: string;
begin
tfile:= TStringList.Create; // создать список строк
// загрузить текст из файла в список строк
tfile.LoadFromFile(File_Ansi);
str:= tfile.Text;
// загрузить текст из списка в memo-поле
Form1.Memo1.Lines.Add(str);
tfile.Free;
end;
procedure Memo_Ansi(File_Ansi: string);
// Сохранение текста из memo-поля в файле на диске
var
tfile: TStringList;
str: string;
begin
tfile:= TStringList.Create; // создать список строк
str:=Form1.Memo1.text;
// преобразовать текст в системную кодировку
str:= UTF8ToSys(str);
tfile.Add(str);
// сохранить в файле
tfile.SaveToFile(File_Ansi);
tfile.Free;
end;
```

11. Напишите программный код для процедуры обработчика щелчка на пункте меню Файл – Открыть. Для этого дважды щелкните на данном пункте меню.

```
procedure TForm1.MenuItem3Click(Sender: TObject);//Файл - открыть
var
File_Ansi:string;
begin
if OpenDialog1.Execute then
begin
File_Ansi:= OpenDialog1.FileName;
File_Ansi:= UTF8ToSys(File_Ansi);
Ansi_Memo(File_Ansi);
end;
end;
```

После выбора пользователем файла в свойстве OpenDialog1.FileName будет находиться имя файла вместе с путем к нему.

Обратите внимание на оператор:

fname:= UTF8ToSys(fname).

Если имя файла, а также путь содержит кириллицу, то необходимо строку с именем файла преобразовать в системную кодировку.

12. Напишите программный код процедуры обработчика щелчка на пункте меню Файл-Сохранить:

```
procedure TForm1.MenuItem4Click(Sender: TObject); //Файл - Сохранить
var
File_Ansi:string;
begin
if SaveDialog1.Execute then
begin
File_Ansi:=SaveDialog1.FileName;
File_Ansi:= UTF8ToSys(File_Ansi);
Memo_Ansi(File_Ansi);
end;
end;
```

Приложение открывает диалоговое окно «Сохранить», в котором задается имя файла. Имя файла из свойства SaveDialog1.FileName запоминается в переменной FName. В заключительной части процедуры оператор Memo1.Lines.SaveToFile(FName); используется для записи в файл содержимого свойства Lines объекта Memo1

13. Напишите программный код процедуры обработчика щелчка на пункте меню Шрифт:

```
procedure TForm1.MenuItem2Click(Sender: TObject);
begin
FontDialog1.Font:= memo1.Font;
if FontDialog1.execute=true then
begin
Form1.Memo1.Font := FontDialog1.Font;
end;
end;
```

14. Сохраните, откомпилируйте и запустите на выполнение созданное приложение.

15. Щелкните мышью на кнопке Открыть, в диалоговом окне Открыть текстовый файл выберите папку, задайте тип файла и выберите текстовый файл, после чего нажмите кнопку Открыть.

16. Отредактируйте текст в окне приложения и нажмите кнопку Сохранить.

После этого в диалоговом окне сохранить текстовый файл выберите в поле тип файла расширение для сохраняемого файла, задайте его имя и щелкните мышью по кнопке Сохранить.

17. Открыв в окне Проводника папку, в которой был сохранен файл, убедитесь, что файл с указанным вами именем в ней есть.

18. Измените размер шрифта и цвет в диалоговом окне Шрифт.

19. Завершите работу с приложением

Вопросы для самоконтроля

1. Переменные ссылочного типа.
2. Конструкторы.

Лабораторное занятие 17. Создание классов для вычисления математических выражений.

Тема 11. Модификаторы доступа к элементам класса. Методы классов.

Цель: изучить основы работы в Lazarus.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

1) Бегущая строка

Рассмотрим некоторые возможности работы с текстом на примере проекта «Бегущая строка».

Бегущая строка является одним из элементов, привлекающим внимание пользователя, а также для экономии места на экране. Бегущие строки часто помещают на Web-сайтах. Сегодня мы добавим бегущую строку в Lazarus-приложение.

Что из себя представляет бегущая строка? Эта область, где «проезжает» текст.

Компонент TTimer – таймер инициируем через определенные промежутки времени событие OnTimer. В нашем случае через определенные промежутки времени мы будем изменять положение текста в бегущей строке.

Компонент TTrackbar –индикатор текущего значения, который позволяет посредством перемещения мышью бегунка интерактивно изменять это значение. Мы будем использовать бегунок для выбора скорости движения бегущей строки.

Прежде чем приступить к выполнению проекта, познакомьтесь с описанием новых компонентов: TTimer (Таймер) и TTrackBar (Бегунок).

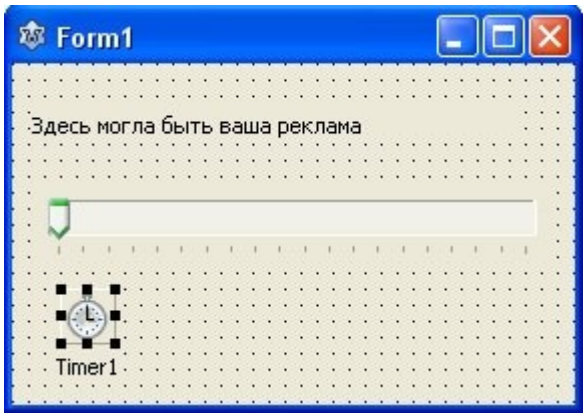
Задание. Создать бегущую строку, используя компонент TLabel (Надпись) и TTimer (Таймер). Б**егущая строка**, достигнув края, появляется с противоположной стороны. В качестве дополнительного буфера для хранения всего текста использовать строковую переменную (тип String).

На рисунке приведен пример формы с таймером, бегунком и компонентом Tlabel для отображения бегущей строки.

Ход выполнения проекта

1. Создайте новый проект, сохраните его в папке с названием «Бегущая строка».

2. На форме разместите компоненты: Tlabel (Надпись), TTimer (Таймер), TTrackBar (Бегунок) так, как показано в образце.



3. Настроим свойства компонента TrackBar

Свойства TrackBar	Назначение	Значение
Min	Минимальное значение – бегунок находится на левом краю полосы	10
Max	Максимальное значение – бегунок находится на правом краю полосы	200
Position	Текущее положение бегунка	10
Frequency	Частота засечек	10

4. Создайте обработчик события TrackBar1Change для компонента TTrackBar, выполнив на нем двойной щелчок.

При изменении положения движка должен включиться таймер, свойству Interval компонента TTimer присваивается значение, равное позиции движка. Через указанный интервал будет происходить событие OnTimer.

Вводим в созданный обработчик TForm1.TrackBar1Change код:

```
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
Timer1.Enabled:=true; //Таймер включить
//При перемещении движка Timer включается
// и его значение устанавливается равным позиции движка
Timer1.Interval:= TrackBar1.Position;
end;
```

5. Теперь займемся компонентом TLabel, в котором будет отображаться текст «Бегущей строки».

1) На TLabel, поместим нужный текст (свойство Caption).

2) Зададим размер (ширину) компонента в 250 пикселей (свойство Width).

6. Создадим обработчик события OnTimer для компонента TTimer. Для этого выполнив на нем двойной щелчок мышью.

Пусть направление текста будет справа-налево. Чтобы создать эффект движущегося текста, будем изменять текст следующим образом: вырезаем из него первый символ и помещаем его в конец строки.

Вводим в созданный обработчик события TForm1.Ttimer1Timer код, который будет примерно таким:

```
procedure TForm1.Timer1Timer(Sender: TObject);
var s:string[200];
begin
s:=Label1.Caption; //Запоминаем строку
s:=s + s[1]; //перемещаем первый символ в конец строки
Delete(s,1,1); //Удаляем первый символ
Label1.Caption:=s; //Отображаем "сдвинутую" строку
end;
```

7. Запустим программу и посмотрим, что получилось. Текст в поле надписи должен движется справа-налево.

2) Графические методы и процедуры

Основой графической системы Lazarus является класс TCanvas. Канва не является компонентом, но она входит в качестве свойства во многие другие компоненты, которые должны уметь нарисовать себя и отобразить какую-либо информацию.

Работа с графикой в Lasarus предполагает обращение к свойству Canvas компонентов, на которых вы хотите выводить изображение. Для программиста Canvas – это холст, который дает доступ к каждому пикселю.

При работе с графикой у вас в распоряжении находятся холст (свойство Canvas того компонента, на котором мы собираемся рисовать), кисть (свойство Brush объекта Canvas), перо (свойство Pen объекта Canvas). Кроме того, в нашем распоряжении будут и графические методы объекта Convas, которые позволяют рисовать не только пикселями, но и с помощью графических примитивов, что существенно упрощает работу с графикой в Lasarus.

Свойства объекта Pen (Перо)

<table>
<tr><td>Свойство</td><td>Описание</td></tr>
<tr><td>Color</td><td>Цвет линии</td></tr>
<tr><td>Width</td><td>Толщина линии(задается в пикселах)</td></tr>
<tr><td>Style</td><td>Стиль пера. Задается следующими именованными константами:
<table>
<tr><th>Значение</th><th>Описание</th><th>Вид линии</th></tr>
<tr><td>psSolid</td><td>Сплошная линия</td><td>_____</td></tr>
<tr><td>psDash</td><td>Штриховая линия</td><td>- - - - - - - - - - - -</td></tr>
<tr><td>psDot</td><td>Пунктирная линия</td><td>. . . . . .</td></tr>
<tr><td>psDashDot</td><td>Штрихпунктирная линия</td><td>_ . _ . _ . _ . _ .</td></tr>
<tr><td>psDashDotDot</td><td>Линия - штрих и два пунктира</td><td>_ . . _ . . _ . . _ . .</td></tr>
<tr><td>psClear</td><td>Отсутствие линии</td><td></td></tr>
</table>
</td></tr>
</table>

Свойства объекта Brush (Кисть)

Свойство	Описание
Color	Цвет закраски замкнутой области.

Style	Стиль закраски области:- сплошная заливка; штриховка:bsHorizontal — горизонтальная;bsVertical - вертикальная;bsFDiagonal -диагональная с наклоном линии вперед; bsBDiagonal -диагональная с наклоном линии назад; bsCross -диагональная клетка.

Основные свойства класса TFont

Свойство	Назначение
Charset	кодировка символов. Для шрифтов с русскими буквами — это PC-1251, КОИ-8
Name	Вид шрифта. Например *Arial*.
Size	Размер шрифта
Style	Начертание символов. Задается с помощью констант:fsBold (полужирный), fsItalic (курсив), fsUnderline (подчеркнутый), fsStrikeOut (перечеркнутый).Свойство Style является множеством, что позволяет комбинировать необходимые стили. Например, инструкция, которая устанавливает стиль «полужирный курсив» выглядит так:Font.Style := [fsBold, fsItalic]

Константы TColor

```
clBlack     -     черный   clAqua          -          бирюзовый
clPurple    -   фиолетовый  clOlive         -          оливковый
clWhite     -      белый     clFuchsia       -         сиреневый
clMaroon - темно-красный   clTeal        -         сине-зеленый
clRed       -      красный   clGray         -          темно-серый
clNavy    -   темно-синий   clLime         -         ярко-зеленый
clGreen     -     зеленый   clMoneyGreen  - цвет зеленых денег
clBrown    -   коричневый   clLtGray       -        светло-серый
clBlue      -       синий   clDkGray       -         темно-серый
clSkyBlue    -    голубой   clMedGray          -           серый
clYellow     -     желтый   clSilver     -          серебристый
clCream     -  кремовый
```

Свойства объекта Canvas

Свойство	Описание
Pen	Перо. Определяет цвет, стиль и ширину линии рисования.
Brush	Кисть. Определяет цвет, текстуру заполнения фигур или фона.
Font	Шрифт. Определяет вид и характеристики шрифта: цвет, размер, стиль и т.д.

Графические методы объекта Canvas

Метод	Описание
Arc (X1, Y1, X2, Y2,X3, Y3, X4, Y4)	Рисует дугу. Параметры X1,Y1, X2,Y2 задают эллипс, частью которого является дуга, параметры X3,Y3, X4,Y4 – начальную и конечную точку дуги.
Chord(X1, Y1, X2, Y2,X3, Y3, X4, Y4)	Рисует хорду и отсекаемую ею часть эллипса. Эллипс, начальная и конечная точки определяются, как в методе Arc.
Ellipse(X1, Y1, X2,Y2)	Рисует эллиспс (окружность). Параметры X1, *Y1* указывают координаты верхней левой точки, X*2*, Y*2* координаты нижней правой точки прямоугольника в который вписана окружность.
MoveTo(X, Y)	Перемещает текущее положение пера в точку (X,Y).
LineTo(X, Y)	Рисует линию из текущей точки в точку с координатами (X,Y).
Rectangle(X1, Y1, X2,Y2)	Рисует прямоугольник. Параметры X*1*, *Y1* указывают координаты верхней левой точки, а X*2*, Y*2* координаты нижней правой точки.
RoundRect(X1, Y1, X2,Y2,X3, Y3)	Рисует прямоугольник со скругленными углами. Параметры X*1*,Y*1* указывают координаты верхней левой точки, X*2*, Y*2* координаты нижней правой точки, а X*3*, Y*3* — радиус скругления.
FillRect(X1, Y1, X2,Y2);	Производит заливку прямоугольника (текущей кистью).
Draw(X, Y, Graphic);	Добавляет на холст рисунок, указанный в параметре Graphic, в место, определяемое координатами (X, Y).
FloodFill(X, Y, Color, FillStyle)	Производит заливку области текущей кистью. Процесс начинается с точки (X,Y). Если режим FillStyle равен fsSurface, то он продолжается до тех пор, пока есть соседние точки с цветом Color. В режиме fsBorder закрашивание, наоборот, прекращается при выходе на границу с цветом Color.
Polygon(Points);	Строит многоугольник, используя массив

	координат точек Points. При этом последняя точка соединяется с первой.Polygon ([Point(10,10), Point(30,30),Point(20,40)])
Polyline(Points);	Строит ломаную линию, используя массив координат точек Points.
TextOut(X, Y, Text)	Производит вывод строки Text начиная с точки (X,Y).

Вычерчивание графических примитивов на графической поверхности, например компонента Image (Вкладка Additional), выполняют соответствующие методы класса TCanvas.

Инструкция, обеспечивающая вычерчивание графического элемента на поверхности компонента , в общем виде выглядит так:

Объект.Canvas.Метод(Параметры);

Объект — это объект, на поверхности которого нужно нарисовать графический элемент. В качестве объекта можно указать компонент Image.

Метод – это имя метода, который обеспечивает рисование нужного графического элемента.

Параметры, в большинстве случаев, определяют положение графического элемента на графической поверхности и его размер.

Например в результате выполнения инструкции

Image1.Canvas.Rectangle(10,20,60,40);

в поле компонента Image1 будет нарисован прямоугольник шириной 50 и высотой 20 пикселей, левый верхний угол которого будет находиться в точке(10,20).

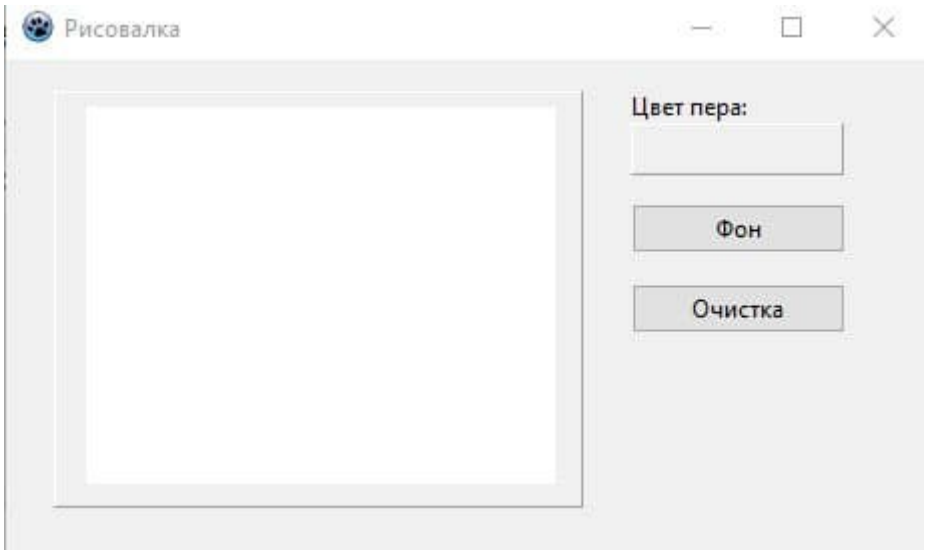
При записи инструкций, обеспечивающих вывод графики, удобно использовать инструкцию with, которая позволяет сократить количество набираемого кода. Например, вместо:

```
Image1.Canvas.Brush.Color := clGreen;
Image1.Canvas.Rectangle(20, 20, 46, 70);
```

Можно написать вот так:

```
With Image1.Canvas do
Begin
Brush.Color := clGreen;
Rectangle(20, 20, 46, 70);
End;
```

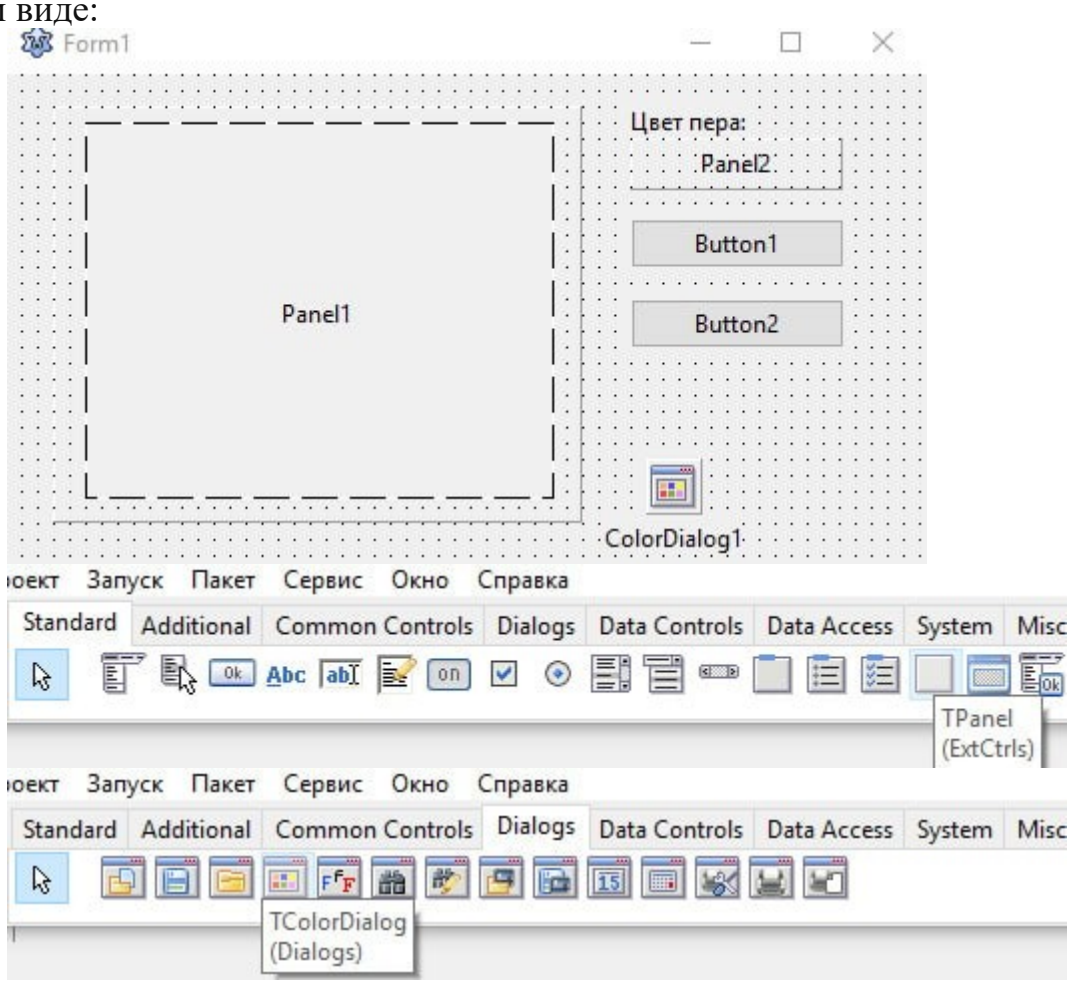
Задание. Создать проект, реализующий возможности простейшего графического редактора, позволяющего рисовать произвольные линии при нажатой клавише мыши.



Для создания простейшей программы для рисования нам потребуются следующие основные компоненты: TImage, TPanel, TColorDialog, TButton.

Ход выполнения проекта

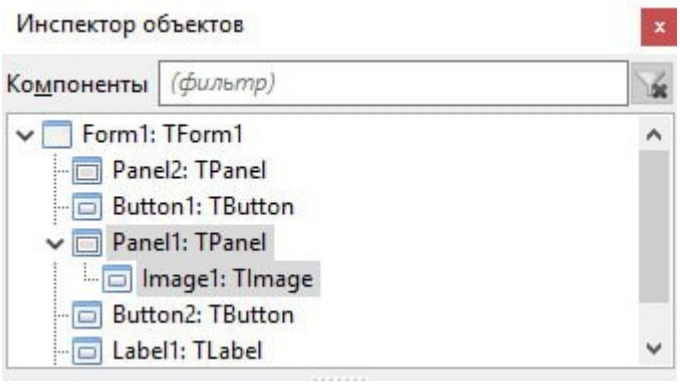
1. Для начала разместите на форме данные компоненты примерно в таком виде:



2. В инспекторе объектов выберите компонент Panel1 и установите значение bnLowered для свойства BevelOuter

Привяжите Image1 к Panel1, переместив его на Panel1 в инспекторе объектов.

При этом Panel1 должна быть немного больше чем Image1



3. Откройте вкладку “события” для Image1 в инспекторе объектов, найдите вкладки OnMouseDown и OnMouseMove, нажмите кнопки “…” напротив них, чтобы получились данные строки:

OnMouseDown	Image1MouseDown
OnMouseEnter	
OnMouseLeave	
OnMouseMove	Image1MouseMove

Проделайте ту же операцию с событием OnCreate для Form1

4. Теперь, в редакторе исходного кода найдите строку

procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

Вписываем данный код. Он нужен для установки позиции пера

```
procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
Image1.Canvas.MoveTo(x,y);
end;
```

5. Далее, найдите строку

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;X,Y: integer);

Впишите в эту процедуру данный код

```
procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;X,Y: integer);
begin
if ssLeft in shift then image1.Canvas.LineTo(x,y);
end;
```

Таким образом, будет рисоваться линия, если нажата левая кнопка мыши

6. После этого необходимо реализовать выбор цвета с помощью второй панели.

Для этого щелкните 2 раза по второй панели и в редакторе исходного кода

Пропишите данные строки

```
procedure TForm1.Panel2Click(Sender: TObject);
begin
//Устанавливаем цвет в диалоге "Цвет",
//согласно цвету пера
ColorDialog1.Color:=Image1.Canvas.Pen.Color;
```

```
        //Если цвет в диалоге "Цвет" выбран,
        //то присваиваем его перу и закрашиваем этим цветом панель
        If ColorDialog1.Execute then
        begin
        Panel2.Color:=ColorDialog1.Color;
        Image1.Canvas.Pen.Color:=ColorDialog1.Color;
        end;
        end;
```

7. Далее мы реализуем кнопку “Фон”. 2 раза щелкните по Button1 и впишите код:

```
        procedure TForm1.Button1Click(Sender: TObject);
        begin
        //Устанавливаем цвет в диалоге "Цвет",
        //Согласно цвету кисти
        ColorDialog1.Color:=Image1.Canvas.Brush.Color;
        //Если цвет в диалоге "Цвет" выбран,
        //то присваиваем его кисти
        //и закрашиваем канву выбранным в диалоге цветом
        if ColorDialog1.Execute then
        begin
        Image1.Canvas.Brush.Color:=ColorDialog1.Color;
        Image1.Canvas.Fillrect(Image1.Canvas.ClipRect);
        end;
        end;
```

8. Вторую кнопку мы используем для функции очищения изображения

```
        procedure TForm1.Button2Click(Sender: TObject);
        begin
        //Закрашиваем канву белым цветом
        Image1.Canvas.FillRect(Image1.Canvas.ClipRect);
        end;
```

Практически тот же самый код мы используем для очищения изображения при запуске программы

```
        procedure TForm1.FormCreate(Sender: TObject);
        begin
        Image1.Canvas.FillRect(Image1.Canvas.ClipRect);
        end;
```

9. Теперь, переходим к завершающему этапу программы – оформлению.

Установите данные значения в инспекторе объектов:

Компонент	Свойство	Значение
Form1	Caption	Рисовалка
Label1	Caption	Цвет пера:
Panel1	Caption	*пусто*
Panel2	Caption	*пусто*

Button1	Caption	Фон
Button2	Caption	Очистить

10. Графический редактор готов. Запустите программу, выберите цвет линий и цвет фона и можно рисовать.

Вопросы для самоконтроля

1. Переменные ссылочного типа.
2. Конструкторы.

Лабораторное занятие 18. Разработка проектов с обработкой событий.

Тема 12. Синтаксис наследования. Способы реализации интерфейсов. Обработка события.

Цель: изучить основы работы в Lazarus.

Оборудование: ПК, ОС Windows, MS Office, Pascal, Lazarus, методические указания по выполнению лабораторного занятия.

Ход работы:

1) Программа Светофор

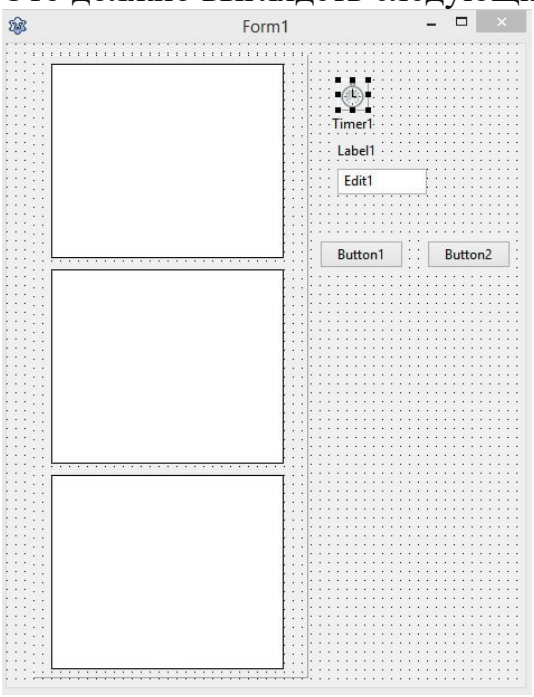
Задание. Смоделировать работу светофора. При запуске проекта панель светофора должна быть пустой. После нажатия на кнопку Пуск лампочки светофора начинают переключатся. После нажатия на кнопку Стоп – панель светофора опять пустая. С помощью таймера обеспечить смену сигнала светофора через равные промежутки времени. В поле Скорость вводится интервал таймера.

Ход выполнения проекта

1. Создайте новый проект. Сохраните его в отдельной папке, назовите ее «Светофор».

2. Разместите на форме панель (TPanel) с тремя фигурами (TShape), две кнопки (TButton), текстовое поле (TEdit), надпись (TLabel), таймер (TTimer) в соответствии с образцом:

Это должно выглядеть следующим образом:



2. Делаем оформление:

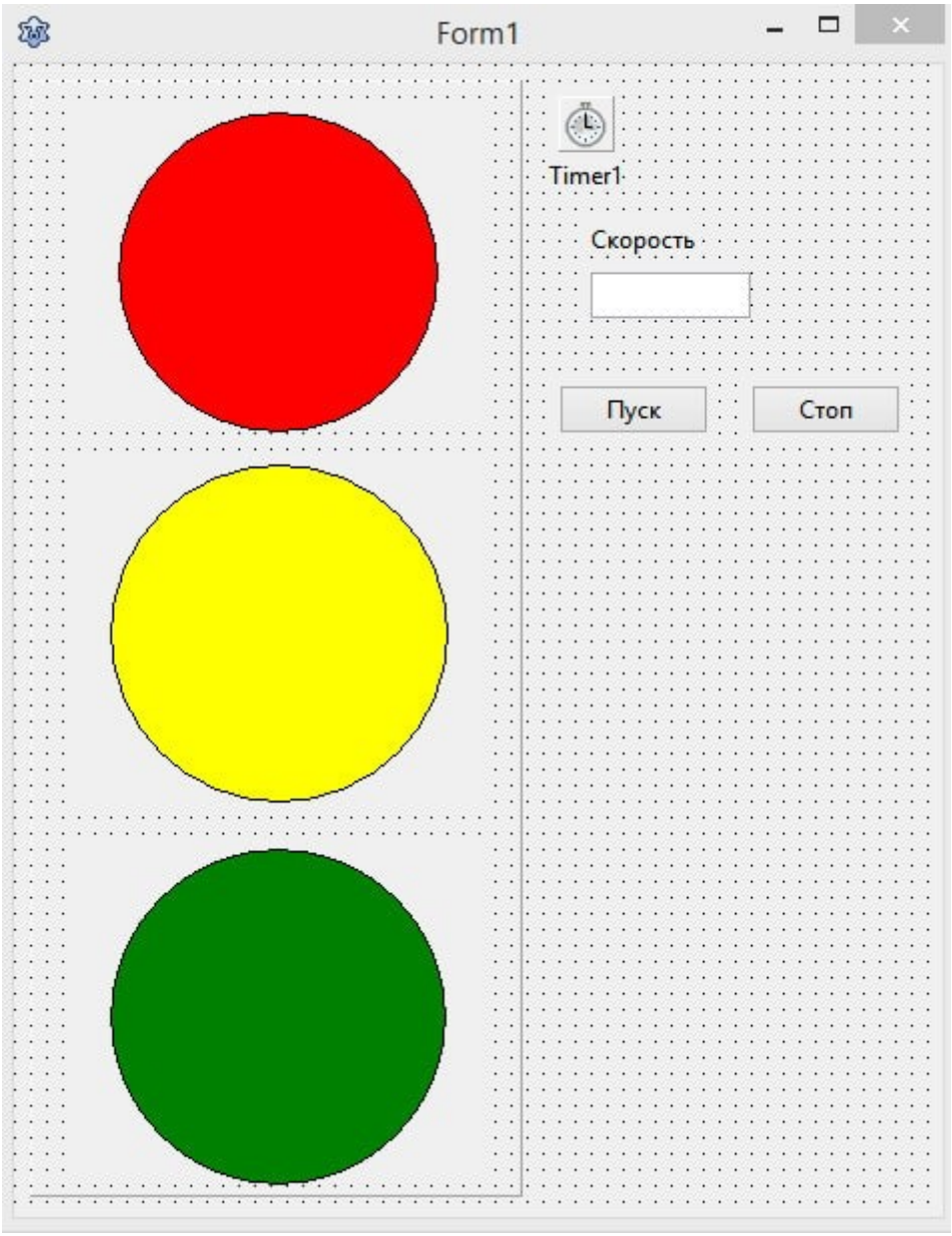
Установите данные значения свойств в инспекторе объектов:

Компонент	Свойство	Значение
Form1	Caption	Светофор
Panel1	Caption	*Пусто*
Shape1	Shape	stCircle
Shape2	Shape	stCircle
Shape3	Shape	stCircle
Label1	Caption	Скорость
Edit1	Text	*пусто*
Button1	Caption	Пуск
Button2	Caption	Стоп

3. Создаем событие для Form1 в разделе OnCreate – Нажать троеточие
Создаем событие для Timer1 в разделе OnTimer – Нажать троеточие
4. Задаем цвета фигурам:

Компонент	Свойство	Значение
Shape1	Brush(Нажать на стрелочку слева от свойства и выбрать color)	clRed
Shape2	Brush(Нажать на стрелочку слева от свойства и выбрать color)	clYellow
Shape3	Brush(Нажать на стрелочку слева от свойства и выбрать color)	clGreen

Финальный вид работы:



5. Во время загрузки формы таймер отключается, фигуры на панели становятся невидимыми.

Создаем обработчик события FormCreate (дважды щелкаем по компоненту Form1) и вставляем данный код:

```
var k:integer;
procedure TForm1.FormCreate(Sender: TObject);
begin
  Timer1.Enabled:=false;
  Shape1.Visible:=false;
  Shape2.Visible:=false;
  Shape3.Visible:=false;
end;
```

6. Чтобы переключались лампочки светофора, напишите программный код в обработчике события Timer1Timer. Код этот будет выполняться с интервалом, который пользователь введет в поле Скорость. По показаниям таймера определяется номер лампочки, которая должна включиться в данный момент.

Дважды щелкаем по компоненту Timer1 и вставляем данный код:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
   k:=k+1;
   //определяем, фигура с каким номером
   //в данный момент должна быть видимой
if k mod 3=1 then
begin
   Shape1.Visible:=true;
   Shape2.Visible:=false;
   Shape3.Visible:=false;
end
else
if k mod 3=2 then
begin
   Shape1.Visible:=false;
   Shape2.Visible:=true;
   Shape3.Visible:=false;
end
else
begin
   Shape1.Visible:=false;
   Shape2.Visible:=false;
   Shape3.Visible:=true;
end
end;
```

6. Напишите программный код для кнопки Пуск. По щелчку на кнопке из поля Скорость считывается интервал для таймера, зануляется показания таймера, таймер включается.

Дважды щелкаем по компоненту Button1 и вставляем код:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Timer1.Interval:=StrToInt(Edit1.text);
  k:=0;
  Timer1.Enabled:=true;
end;
```

7. Напишите программный код для кнопки Стоп. После щелчка на кнопке таймер должен отключиться, лампочки светофора снова становятся невидимыми.

Дважды щелкаем по компоненту Button2 и вставляем код:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Timer1.Enabled:=false;
  Shape1.Visible:=false;
  Shape2.Visible:=false;
  Shape3.Visible:=false;
end;
```

8. Запустите проект. В поле Скорость введите число 1000 (1000 мс=1с). Лампочки светофора начнут переключаться с интервалом в одну секунду.

2) Воспроизведение звука

Рассмотрим возможности воспроизведения звука в приложении Lazarus.

В самом простейшем случае приложение должно уметь выдавать звуковые сигналы или проигрывать небольшие звуковые сообщения при появлении каких-либо непредвиденных событий, когда нужно привлечь внимание пользователя.

Наиболее простой процедурой, управляющей звуком, является процедура Beep. Она не имеет параметров и воспроизводит стандартный звуковой сигнал, установленный в Windows.

Более серьезной функцией является функцию **sndPlaySound.**

Функция sndPlaySound воспроизводит указанный волновой файл формата .wav или определенный системой звук. Она объявлена в модуле mmSystem следующим образом:

BOOL sndPlaySound(LPCSTR lpszSoundFile, UINT wFlags);

Через параметр lpszSoundFile этой функции можно передать путь к wav-файлу.

Параметр wFlags определяет способ проигрывания звукового фрагмента. Используются следующие значения (некоторые из них можно комбинировать при помощи операции ИЛИ):

Значение	Описание
SND_SYNC	Синхронный режим работы. Функция sndPlaySound вернет управление только после завершения проигрывания звукового фрагмента
SND_ASYNC	Асинхронный режим работы. Функция вернет управление немедленно, проигрывание звукового фрагмента будет выполняться в фоновом режиме параллельно с работой приложения
SND_NODEFAULT	Если указанный файл не найден, функция «тихо» возвращает управление, не проигрывая никаких звуков. Если же этот флаг не указан, и файл не найден, будет проигран стандартный системный звук.А если и это невозможно, функция не будет ничего проигрывать и вернет значение FALSE
SND_MEMORY	Это значение используется для проигрывания

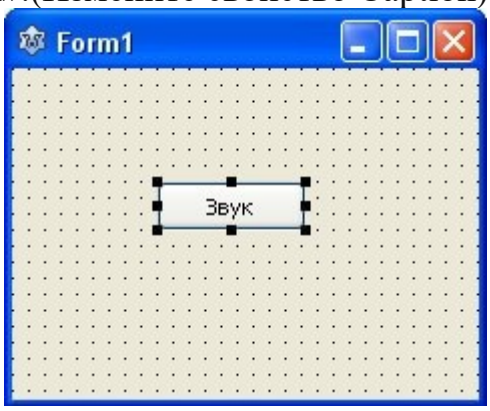
	звуковых файлов, загруженных в оперативную память, например, из ресурсов приложения
SND_LOOP	Если указано значение SND_ASYNC, проигрывание звукового фрагмента будет зациклено. Для того чтобы остановить проигрывание, необходимо вызвать функцию sndPlaySound, указав ей в качестве параметра lpszSoundFile значение NULL
SND_NOSTOP	При указании этого значения функция проверяет, выполняется ли в настоящий момент проигрывание фрагмента. Если да, функция возвращает значение FALSE

Во всех случаях, если не указан параметр SND_NOSTOP, функция sndPlaySound возвращает значение TRUE, если выполняется проигрывание, и FALSE — если нет. Учтите, что при использовании функций sndPlaySound есть ограничение на размер wav-файла — он должен целиком помещаться в физическую память. Поэтому этот пособы проигрывания звуковых фрагментов хороши только для относительно небольших файлов.

Задание. Создайте приложение, которое позволит прослушивать простейшие звуковые файлы формата wav.

1. Создайте новый проект, сохраните его в папке с названием «Звук».

2. Поместите на форму компонент TButton, измените надпись на ней на «Звук».(Измените свойство Caption).



3. Подготовьте wav-файл. (Скачайте файл CS2_AutoplayMusic.wav, поместите в папку проекта).

4. Создайте обработчик события для компонента TButton, выполнив на нем двойной щелчок, вводите в созданный обработчик код:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  sndPlaySound('CS2_AutoplayMusic.wav', snd_Async or snd_NoDefault);
end;
```

5. Так как в процедуре обработки нажатия кнопки используется вызов функции sndPlaySound, то следует включить в раздел описания uses вызов модуля mmSystem:

```
implementation
uses MMSystem;
{TForm1}
```

6. Запустим программу и посмотрим, что получилось. Щелкните на кнопке «Звук», звуковой файл должен начать проигрываться.

Вопросы для самоконтроля

1. Синтаксис наследования.
2. Обработка события.

Рекомендуемая литература

Основные источники:

1. Волобуева, Т. В. Информатика. Основы алгоритмизации: учебное пособие / Т. В. Волобуева. — Воронеж: Воронежский государственный технический университет, ЭБС АСВ, 2019. — 73 с. — ISBN 978-5-7731-0740-8. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: http://www.iprbookshop.ru/93316.html

2. Колокольникова, А.И. Практикум по информатике: основы алгоритмизации и программирования: [16+] / А.И. Колокольникова. – Москва; Берлин: Директ-Медиа, 2019. – 424 с.: ил., табл. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=560695

3. Тюльпинова, Н. В. Технология алгоритмизации и программирования на языке Pascal: учебное пособие / Н. В. Тюльпинова. — Саратов: Вузовское образование, 2019. — 244 c. — ISBN 978-5-4487-0471-0. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: http://www.iprbookshop.ru/80540.html

Дополнительные источники

1. Волобуева, Т. В. Информатика. Основы программирования на языке Pascal: учебное пособие / Т. В. Волобуева. — Воронеж: Воронежский государственный технический университет, ЭБС АСВ, 2019. — 93 с. — ISBN 978-5-7731-0756-9. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: http://www.iprbookshop.ru/93317.html

2. Нагаева, И.А. Алгоритмизация и программирование. Практикум: учебное пособие: [12+] / И.А. Нагаева, И.А. Кузнецов. – Москва; Берлин: Директ-Медиа, 2019. – 168 с.: ил., табл. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=570287

3. Тюльпинова, Н. В. Алгоритмизация и программирование: учебное пособие / Н. В. Тюльпинова. — Саратов: Вузовское образование, 2019. — 200 c. — ISBN 978-5-4487-0470-3. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: http://www.iprbookshop.ru/80539.html

Интернет источники:

1. http://delphiexpert.ru/ - уроки, видеокурсы по программированию в среде Free Pascal и Delphi.

2. https://www.sites.google.com/site/ifizmat/prog/lazarus - лабораторные занятия по Lazarus.

3. http://intuit.valrkl.ru/course-708/index.html#ID.1.lecture - Программирование на Free Pascal и Lazarus.