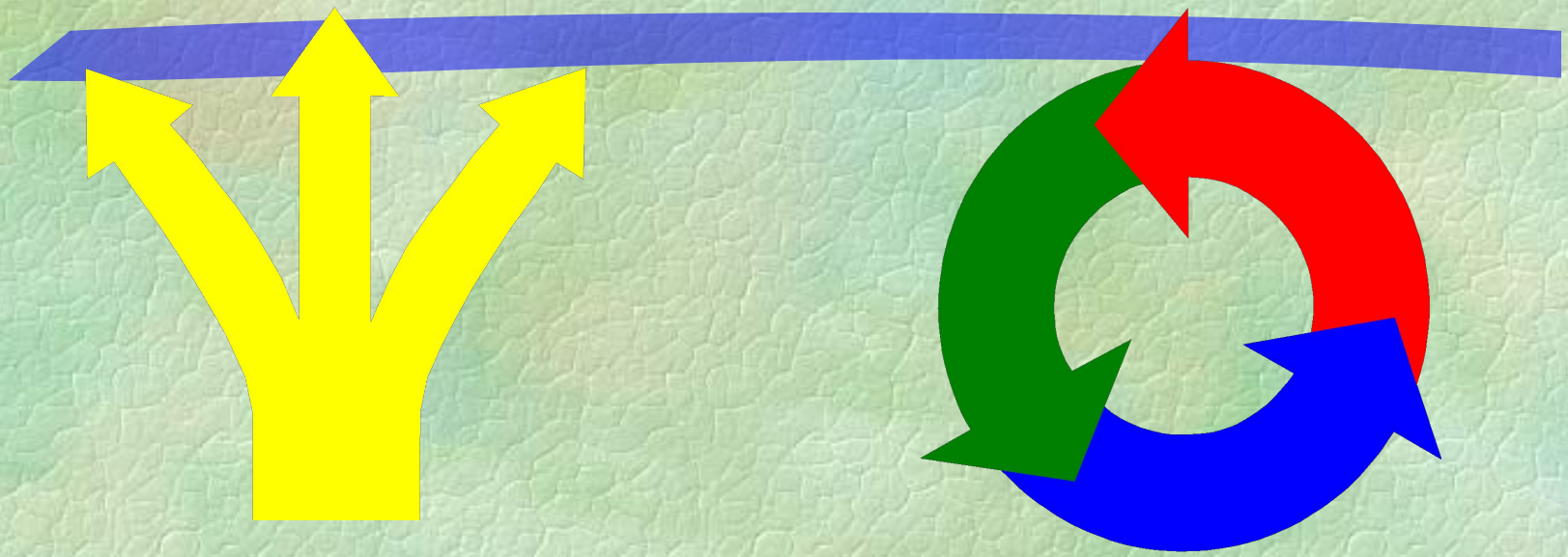
Программирование алгоритмов с ветвлениями и циклами



Delphi. Тема 2.

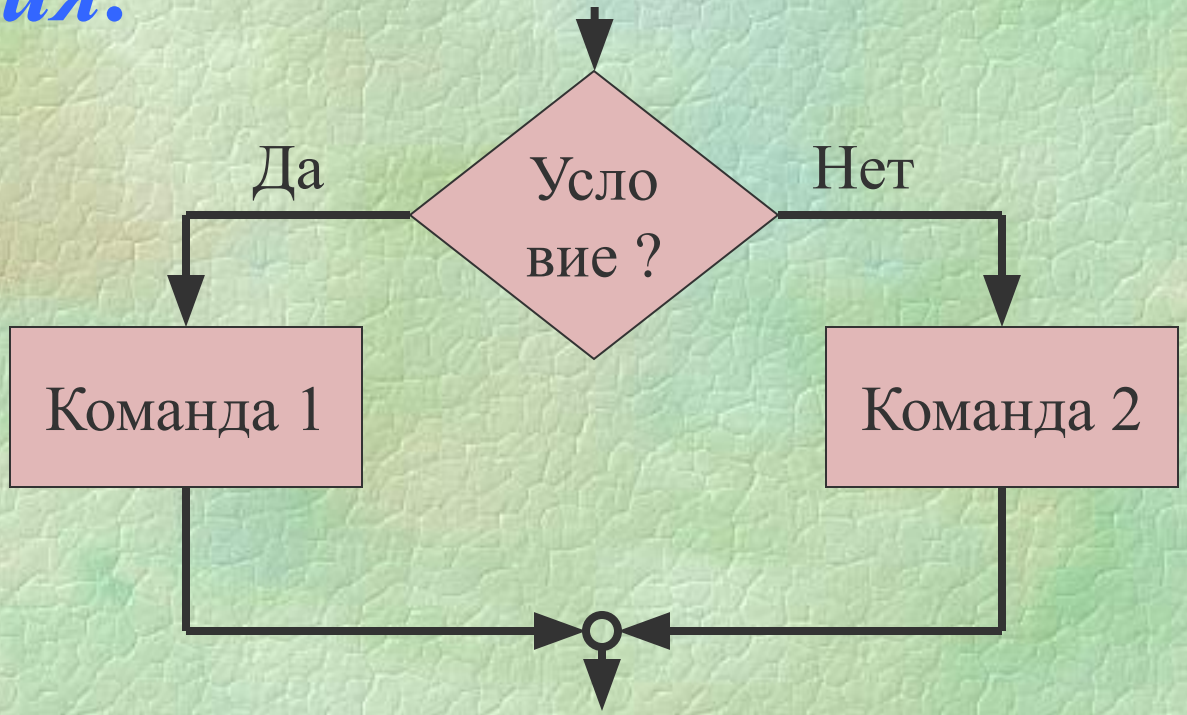
2. Программирование алгоритмов с ветвлениями и циклами.

План темы:

1. Понятие ветвления в алгоритме.
2. Условные операторы IF и CASE.
3. Понятие цикла в алгоритме.
4. Операторы повтора FOR, WHILE и REPEAT.
5. Компоненты CheckBox, RadioGroup, Memo.
6. Примеры программирования ветвлений и циклов.

1. Понятие ветвления в алгоритме.

Ветвление - выбор одного из двух предложенных вариантов действий, в зависимости от результата проверки ***условия:***



Операторы

Условные

Для записи ветвления в программе применяются *условные операторы.*

В Pascal имеются два условных оператора:
IF (Если) и CASE (Выбор).

Формат оператора IF:

IF <условие> THEN
<оператор1>
[ELSE <оператор2>];

Часть ELSE может отсутствовать. Перед ELSE точка с запятой не ставится.

Условие - **это выражение булевского (логического) типа. Результат проверки условия может быть ИСТИНО (TRUE) или ЛОЖНО (FALSE).**

Если значение условия ИСТИНО, то выполняется <оператор1>, иначе выполняется <оператор2>.

Операторы IF могут быть вложенными, т. е. внутри одного может содержаться другой.

Операторы

Условные

Оператор выбора CASE позволяет сделать выбор из произвольного числа имеющихся вариантов действий. Он состоит из выражения, называемого селектором, и списка операторов, каждому из которых предшествует список констант выбора (список констант может состоять из одной, или из нескольких констант, или указывать диапазон значений).

Формат:

```
CASE <выражение-селектор> OF
<значение 1>: <оператор 1>;
<значение 2>: <оператор 2 >;
. . .
<значение N>: <оператор N>
[ ELSE <оператор> ]
END;
```

Оператор CASE работает следующим образом. Сначала вычисляется значение выражения-селектора, затем обеспечивается реализация того оператора, константа выбора которого равна текущему значению селектора.

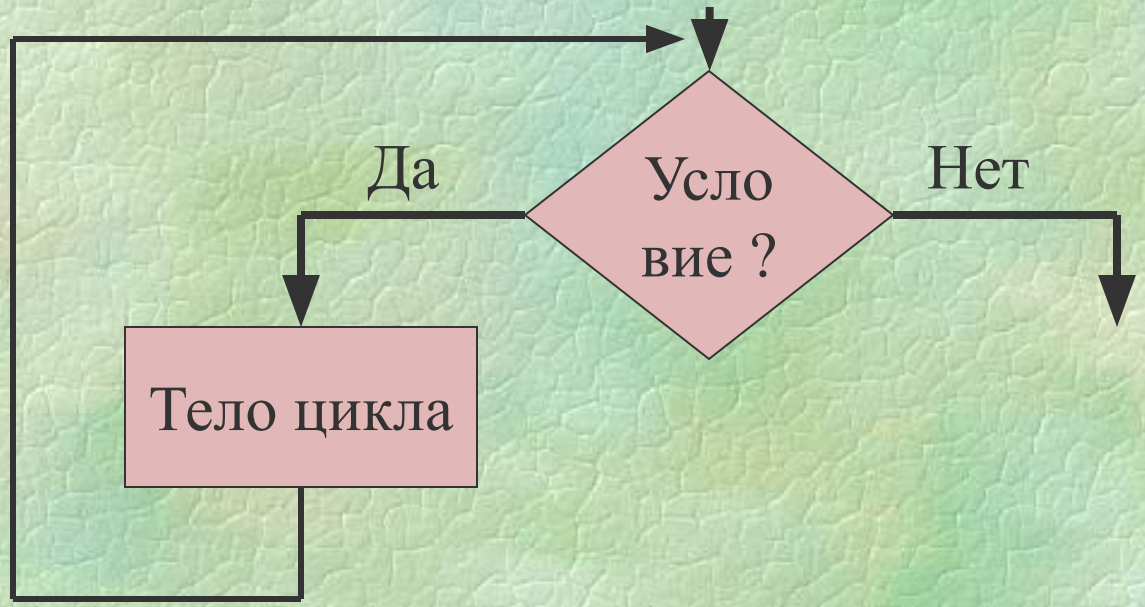
Пример программирования ветвления.

• *Определить положение точки с координатами (x, y) относительно прямой y = 2x + 1.*

```
var x, y: Real;
      s: string;
Begin
 …
 s:='Ответ: точка лежит ';
 If  y = 2*x + 1 then s:=s+'на '
    else If  y > 2 * x + 1 then s:=s+'выше '
                else s:=s+'ниже ';
 s:=s+' прямой.';
  …
End.
```

3. Понятие цикла в алгоритме.

Цикл - выполнение группы операторов *(тела цикла)* несколько раз:



Операторы повтора

Для организации циклов различных типов используются Операторы повтора (или цикла) **FOR, REPEAT, WHILE.**

Оператор цикла с параметром FOR состоит из заголовка и тела цикла. Он может быть представлен в двух форматах:

FOR <параметр цикла> := <S1> TO <S2> DO <оператор>;
FOR <параметр цикла> := <S1> DOWNTO <S2> DO <оператор>;

S1 и S2 - выражения, определяющие соответственно начальное и конечное значения параметра цикла. FOR ... DO - заголовок цикла, <оператор> - тело цикла. Тело цикла может быть простым или составным оператором. Оператор FOR обеспечивает выполнение тела цикла до тех пор, пока не будут перебраны все значения параметра цикла от начального до конечного с шагом изменения равным единице.

Примеры:

```
FOR I:= 1 TO 10 DO S:=S+I;      { вычисление суммы }
FOR I:= 10 DOWNTO 1 DO P:=P*I; { вычисление произведения }
```

Операторы

Повтора

Оператор цикла с постусловием REPEAT состоит из заголовка REPEAT, тела и условия окончания UNTIL.

Формат:

```
REPEAT
  <оператор;>
  ...
  <оператор>
UNTIL <условие>;
```

Операторы, заключенные между словами REPEAT и UNTIL, являются телом цикла. Вначале выполняется тело цикла, затем проверяется условие выхода из цикла. Если результат проверки условия равен False, то тело цикла выполняеся еще раз, если результат True - происходит выход из цикла. По крайней мере один из операторов тела цикла должен влиять на значение условия, иначе цикл будет выполняться бесконечно.

Операторы

Повтора

Оператор цикла с предусловием WHILE аналогичен оператору REPEAT, но проверка условия выполнения тела цикла производится в самом начале оператора.

Формат:

WHILE <условие> DO

<тело цикла>;

Условие - логическое выражение, тело цикла - простой или составной оператор.

Перед каждым выполнением тела цикла вычисляется значение выражения условия. Если результат равен True, тело цикла выполняется и снова вычисляется выражение условия. Если результат равен False, происходит выход из цикла и переход к первому после WHILE оператору.

Когда заранее неизвестно количество повторений цикла, то удобней применять операторы While или Repeat. Когда число повторений известно заранее, то удобней применять оператор For.

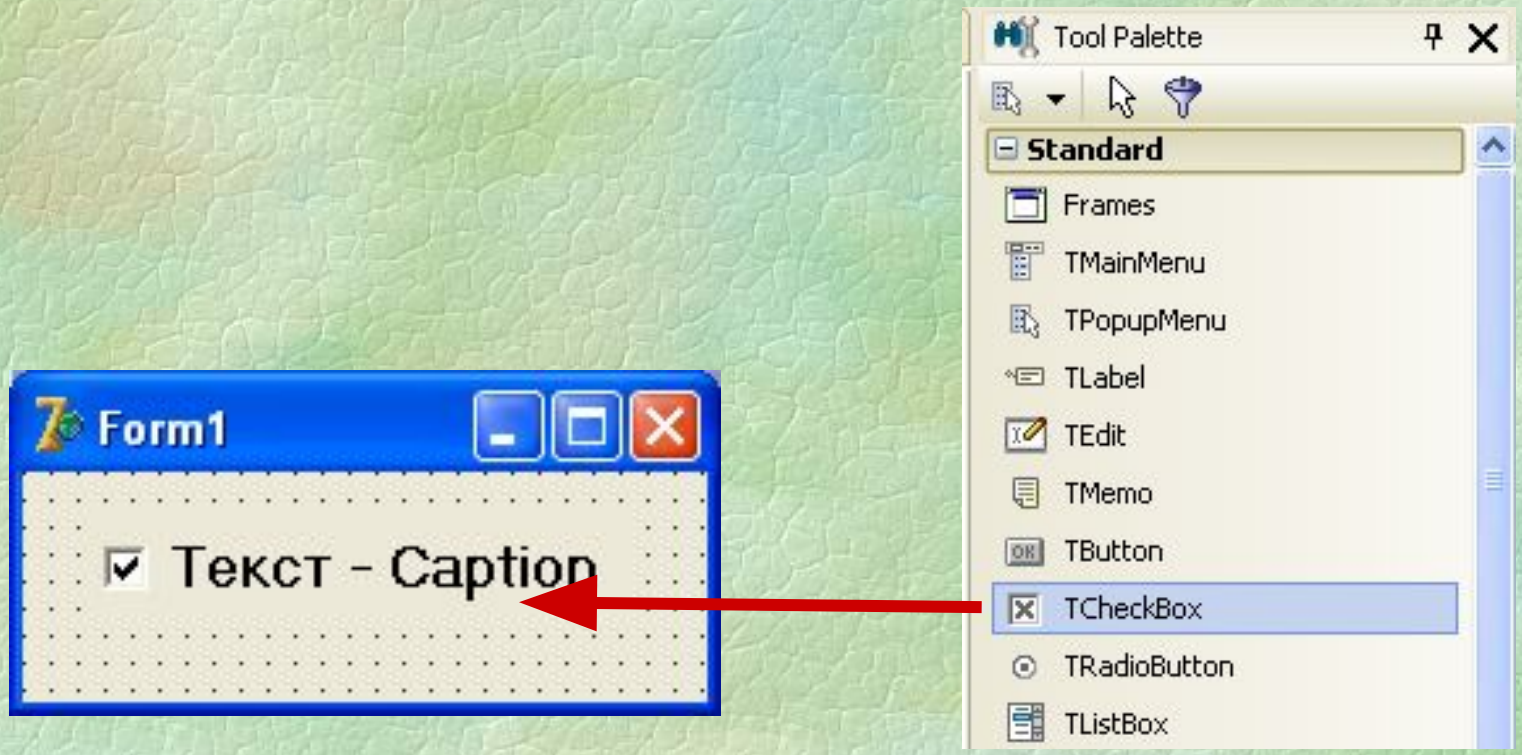
Пример программирования цикла.

• ***Вычислить значение суммы: 1 + 1/4 + 1/9 + 1/16 + … с точностью 0,001.***

```
var S, a: Real;
        i: Integer;
Begin
  S := 1;
   i := 2;
  Repeat
     a := 1/sqr( i );
     S := S + a;
      i := i + 1;
  Until a<=0.001;
  {Вывести значение S}
End.
```

5. Компоненты CheckBox, RadioGroup, Memo.

• **Независимые переключатели (check boxes) используются для установки параметров, характеризуемых двумя значениями — “Да” или “Нет” (True - False). Независимые переключатели создаются с помощью компонента *CheckBox.***



5. Компоненты CheckBox, RadioGroup, Memo.

- **Основные свойства компонента *CheckBox:***

Alignment	Определяет, с какой стороны от переключателя находится текст: taRightJustify - справа, taleftJustify - слева.
Caption	Текст рядом с переключателем.
Checked	Определяет, включен (True), или выключен (False) переключатель.

5. Компоненты CheckBox, RadioGroup, Memo.

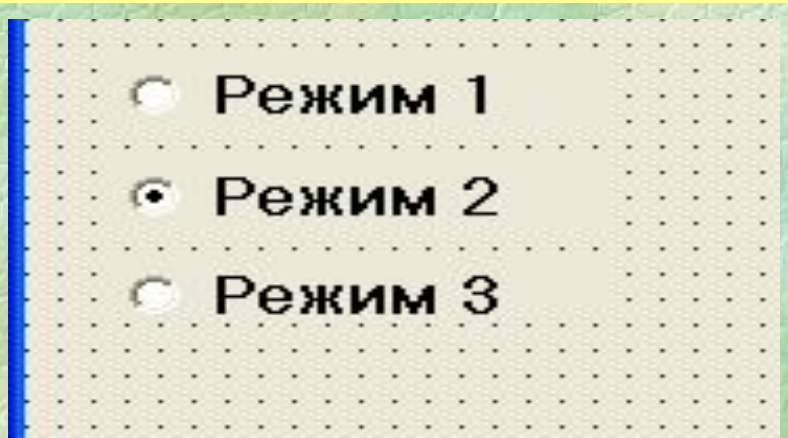
• Зависимые переключатели (radio buttons) служат для установки взаимоисключающих параметров. Они обычно объединяются в группы и позволяют пользователю выбрать одно значение из фиксированного множества альтернатив. При включении одного зависимого переключателя остальные переключатели этой же группы выключаются.

• В отдельности каждый зависимый переключатель представляется компонентом *RadioButton (раздел Standard).*

5. Компоненты CheckBox, RadioGroup, Memo.

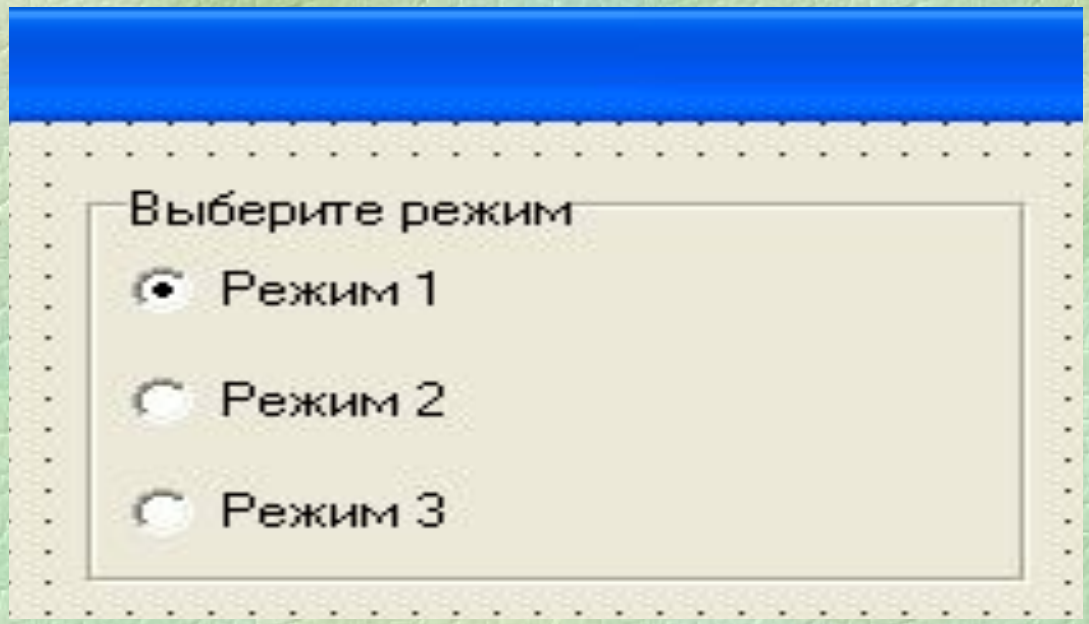
- **Основные свойства компонента *RadioButton:***

Alignment	Определяет, с какой стороны от переключателя находится текст: taRightJustify - справа, taleftJustify - слева.
Caption	Текст рядом с переключателем.
Checked	Определяет, включен (True), или выключен (False) переключатель.



5. Компоненты CheckBox, RadioGroup, Memo.

- **Зависимые переключатели как правило объединяются в группы. Для быстрой организации группы зависимых переключателей очень удобен компонент *RadioGroup (раздел Standard).***
- **Компонент *RadioGroup* удобен тем, что заменяет группу компонентов *RadioButton.***



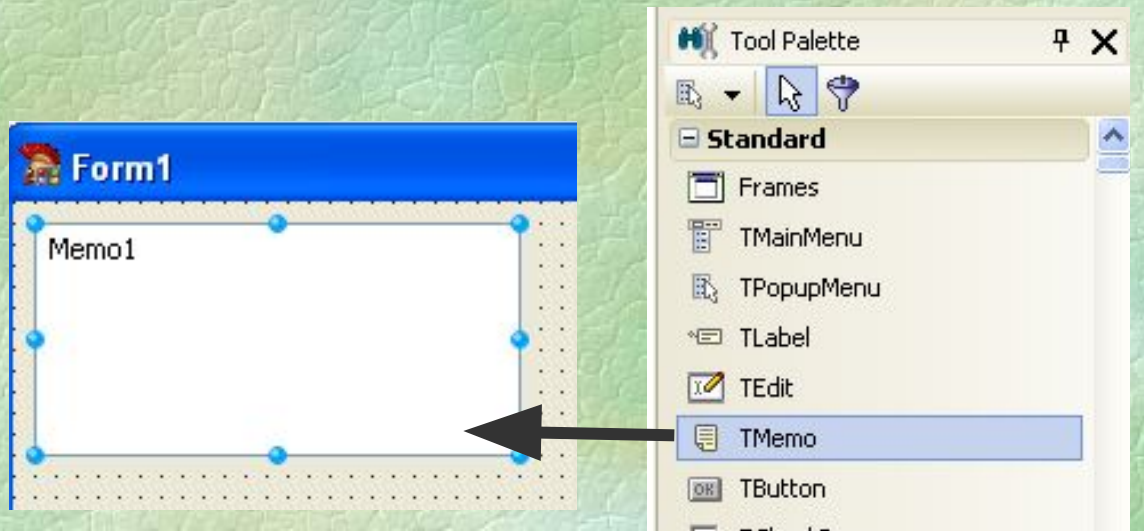
5. Компоненты CheckBox, RadioGroup, Memo.

- **Основные свойства компонента *RadioGroup:***

Alignment	Способ выравнивания группы зависимых переключателей в пределах владельца.
Caption	Подпись к группе зависимых переключателей.
Columns	Число колонок в группе зависимых переключателей.
ItemIndex	Номер выбранного элемента, начиная с нуля. Если все переключатели находятся в выключенном состоянии, то значение свойства равно -1.
Items	Массив переключателей и подписей к ним (нумерация начинается с нуля).

5. Компоненты CheckBox, RadioGroup, Memo.

- Компонент *Memo (раздел Standard)* похож на *Edit*, но в отличие от него хранит не одну строку текста, а множество строк.



5. Компоненты CheckBox, RadioGroup, Memo.

-Доступ к строкам обеспечивает свойство *Lines*, представляющее собой массив строк. Нумерация строк начинается с нуля. Пример:

Memo1.Lines[i]:=IntToStr(i)

- Свойство *Lines* доступно также из «Инспектора Объектов», поэтому на стадии проектирования можно заполнить компонент Memo некоторыми исходными данными.

5. Компоненты CheckBox, RadioGroup, Memo.

- Текущее количество строк в Memo содержится в свойстве *Lines.Count* (учитываются и пустые строки). Пример:

```
For i:=0 to Memo1.Lines.Count-1 do
    Memo1.Lines[i]:=IntToStr(i);
```

- В свойстве *ScrollBars* определяется наличие вертикальной и горизонтальной полос прокрутки в компоненте Memo.

5. Компоненты CheckBox, RadioGroup, Memo.

-Строки можно добавлять, вставлять, удалять при помощи соответствующих методов:

Memo1.Lines.Clear; **- очистить всё содержимое.**

Memo1.Lines.Delete(n); **- удалить строку с номером n.**

Memo1.Lines.Add(строка); **- добавить строку.**

Memo1.Lines.Insert(n,строка) **– вставить строку перед n-й строкой.**

Далее: Лабораторная работа №2. «Программирование алгоритмов с ветвлениями и циклами».