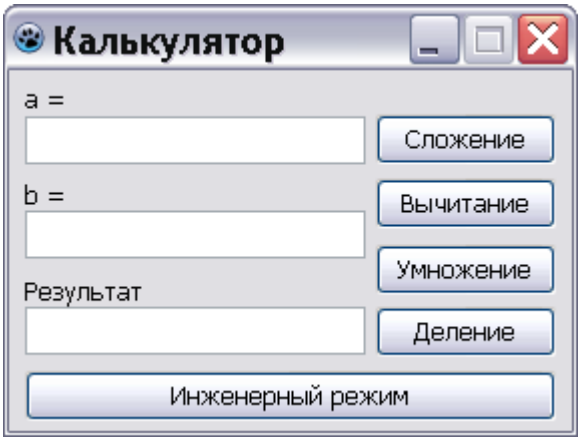
Лабораторная работа №3

Введение. Знакомство с интегрированной средой Lazarus. Изучение основных свойств формы. Обработка исключений.

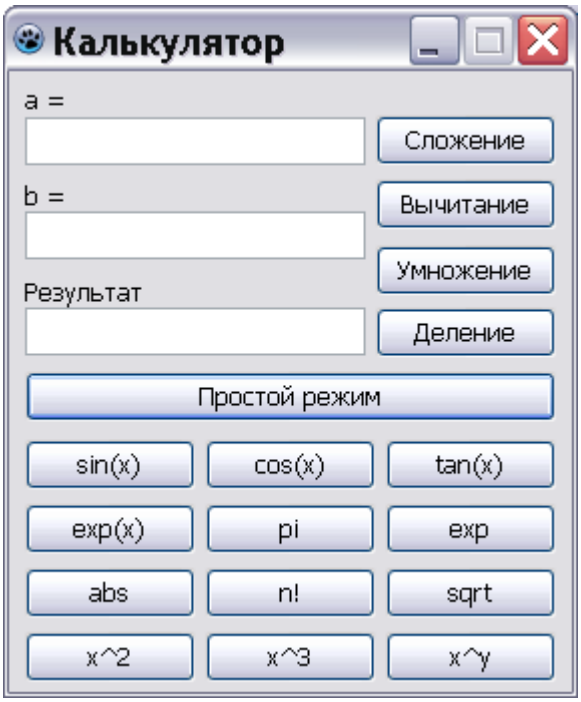
Цель: знакомство с интегрированной средой разработки программного обеспечения Lazarus. Изучение основных свойств формы. Разработка инженерного калькулятора. Знакомство с понятием обработки исключений.

Справочный материал.

Практическое задани: *Расширить функционал программы «Калькулятор» путем введения дополнительных операций (sin(x), cos(x), tg(x), ctg(x), exp(x), abs, x^2, x^3, x^y, sqrt, pi, n!). Интehatqc программы должен соответсвовать образцу:*



Простой вид калькулятора



Инженерный вид калькулятора

Выполнение задания

Выполнение задания начнем с подготовки макета формы и расположения на ней необходимых компонентов:

1. Изменим значение свойства **Height** формы на «310»;
2. Изменим значение свойства **BorderStyle** формы на «**bsSingle**»;
3. Добавим кнопку управления режимом калькулятора:
   - **a.** свойству **Caption** кнопки поставим в соответствие значение «Инженерный вид»
   - **b.** Изменим значение свойства **Width** кнопки на «265»;
   - **c.** Изменим значение свойства **Left** кнопки на «8»;
   - **d.** Изменим значение свойства **Top** кнопки на «150»;
4. Добавим остальные кнопки по образцу:
   - **a.** Изменим значение свойства **Width** кнопок на «85»;
   - **b.** Значение свойства **Left** первого столбца кнопок установить равным «8»;
   - **c.** Значение свойства **Left** второго столбца кнопок установить равным «98»;
   - **d.** Значение свойства **Left** третьего столбца кнопок установить равным «188»;
   - **e.** Значение свойства **Top** первого ряда кнопок установить равным «184»;
   - **f.** Значение свойства **Top** второго ряда кнопок установить равным «216»;
   - **g.** Значение свойства **Top** третьего ряда кнопок установить равным «248»;
   - **h.** Значение свойства **Top** четвертого ряда кнопок установить равным «280»;
   - **i.** Свойства **Caption** кнопок задаем согласно образцу;
5. После настройки свойств кнопок, установим значение свойства **Height** равным «180» - при этом с созданными нами кнопками ничего не случится, они проста останутся вне формы и их не будет видно.

Следующим шагом выполнения задания будет создание обработчиков событий новых кнопок:

1. Обработчик события нажатия кнопки изменения режимов калькулятора имеет следующий вид:

```
procedure TForm1.Button5Click(Sender: TObject);
begin
  if Button5.Caption = 'Инженерный режим' then
  begin
      Button5.Caption := 'Простой режим';
      Form1.Height := 310;
  end
```

```
  else if Button5.Caption = 'Простой режим' then
  begin
      Button5.Caption := 'Инженерный режим';
      Form1.Height := 180;
  end;
end;
```

В этом обработчике событий мы, впервые в этом проекте, используем условия. Оно необходимо нам для определения текущего режима работы калькулятора и переключения в другой режим. Флагом, сигнализирующим нам о том, в каком режиме работает калькулятор, является свойство Caption данной кнопки, а точнее его значение. Если Caption имеет значение «Инженерный режим», то это значит, что в данный момент калькулятор находится в простом режиме и при нажатии кнопки «Инженерный режим» мы переключимся в расширенный режим калькулятора. Если же Caption имеет значение «Простой режим», то это значит, что в данный момент калькулятор находится в инженерном режиме и при нажатии кнопки «Простой режим» мы переключимся из расширенного режима калькулятора в упрощенный.

Сама процедура переключения режима калькулятора из простого в инженерный, представляет собой ни что иное, как программное изменение размеров формы, путем установки свойства Height формы в значение «310» при переключении в расширенный режим и в значение «180», при переключении в упрощенный режим. При этом мы меняем и наш флаг – с «Инженерный режим» на «Простой режим», при переключении в расширенный режим и с «Простой режим» на «Инженерный режим», при переключении в упрощенный режим работы.

2. Далее создадим обработчики событий нажатия кнопок, выполняющих операции: *sin(x), cos(x), tan(x), exp(x), abs,* x^2, x^3, *sqrt*. В качестве примера рассмотрим листинг обработчика события нажатия кнопки «sin»:

```
procedure TForm1.Button5Click(Sender: TObject);
var a, c: real;
begin
    a := StrToFloat(Edit1.Text);
    c := sin(a);
    Edit3.Text := '';
    Edit3.Text := FloatToStr(c);
end;
```

Задание: самостоятельно напишите обработчики событий оставшихся кнопок, используя стандартные функции Lazarus: *sin(x), cos(x), exp(x), abs(x), sqr(x)* (для квадрата числа) и *sqrt*.

3. Теперь напишем обработчики событий оставшихся двух операций: x^y, *n!* . В качестве примера рассмотрим листинг обработчика события нажатия кнопки «x^y»:

```
procedure TForm1.Button16Click(Sender: TObject);
var a, b, c, i: longint;
begin
    a := StrToInt(Edit1.Text);
    b := StrToInt(Edit2.Text);
    c := 1;
    if b = 0 then c:=1
    else for i:=1 to b do c:=c * a;
    Edit3.Text := IntToStr(c);
end;
```

В данном обработчике событий мы также используем условие, но цель ее использования уже другая – известно, что любое число в степени 0 – это единица, поэтому мы и выполняем проверку на то, какое число введено в качестве степени, и если это 0, то результат в любом случае будет равен единице, в противном случае, т.е. если степень отлична от нуля, мы выполняем процедуру подсчета результата, используя для этого цикл с предусловием.

Тело цикла повторяется b раз, т.е. если нам необходимо посчитать 2 в степени 3, то мы должны выполнить умножения числа 2 на себя же ровно 3 раза: 2*2*2. Для выполнения этого действия мы используем следующую команду: *c := c * a;* Причем перед выполнением условия и цикла, мы присвоили переменной с значение единицы, так как в противном случае переменная с была бы равна нулю. Поясним это на нашем примере: 2^3, т.е. a = 2, b = 3.

I. Рассмотрим случай, когда мы не сделали бы присвоения *c :=1*:

Шаг 1. c = 0, c := c * a -> c:= 0*2 = 0;

Шаг 2. c = 0, c := c * a -> c:= 0*2 = 0;

И, наконец, третий шаг – т.к. b = 3, следовательно в цикле три итерации(повторения):

Шаг 3. c = 0, c := c * a -> c:= 0*2 = 0.

Результат c = 0, что конечно же является неверным, т.к. 2 в степени 3 равно 8.

II. Теперь рассмотрим случай, когда мы сделали присвоение *c :=1*:

Шаг 1. c = 1, c := c * a -> c:= 1*2 = 2;

Шаг 2. c = 2, c := c * a -> c:= 2*2 = 4;

И, наконец, третий шаг – т.к. b = 3, следовательно в цикле три итерации(повторения):

Шаг 3. c = 4, c := c * a -> c:= 4*2 = 8.

Результат c = 8, что является верным результатом.

III. Теперь рассмотрим случай, когда степень равна 0, т.е. b = 0 и мы не используем условие: *if b = 0 then c:=1.* В этом случае *с*разу же можно заметить, что цикл *for i:=1 to b do c:=c * a;* не будет выполнен, так как переменная i начинается со значения 1 и должна увеличиваться, а b = 0 меньше 1, т.е. увеличения переменной i не будет.

Задание: самостоятельно напишите обработчик события нажатия кнопки подсчета факториала «*n!*». Напомню, что факториалом числа n является значение: n! = 1*2*3*…*n, например 5! = 1*2*3*4*5=120, т.е. 5! = 120. Также напомню, что факториалом числа 0, является 1, т.е. 0! = 1 (для описания данного случая в обработчике событий Вам потребуется использовать условие наподобие того, которое использовалось в обработчике события нажатия кнопки «x^y»).

4. Теперь мы познакомимся с таким понятием как обработка исключений. Это есть ни что иное как обработка ошибок, которые могут возникнуть в ходе работы программы. Т.е. программист должен предусмотреть возможность обработки ошибок, в случае их возникновения. Например, если Вы введете в поле «a =» не число, а, скажем, строку «Привет», а в поле «b = 4» и нажмете кнопку «Сложение», то программа выдаст ошибку или вообще зависнет, так как инструкция (команда) a := StrToInt(Edit1.Text) не будет выполнена, т.к. «Привет» не является целочисленным значением и не может быть присвоено переменной a, тип которой integer. Следовательно, дальнейшая работа программы будет невозможна. Чтобы избежать этого, мы должны предусмотреть действия, которые не дадут нашей программе зависнуть в подобных случаях. Для обработки исключений используется конструкция:

Try ...Except ...End;

В блоке *Try ...Except* описываются команды, которые могут вызвать ошибку, а в блоке *except ...end* описываются команды, которые будут выполнены в случае ошибки, т.е. в данном блоке описываются команды, которые должны устранить ошибку.

Рассмотрим ее работу на примере обработки исключений в обработчике события нажатия кнопки «Сложение»:

```
procedure TForm1.Button1Click(Sender: TObject);
var a, b, c: integer;
begin
    try
    a := StrToInt(Edit1.Text);
    b := StrToInt(Edit2.Text);
    c := a + b;
    Edit3.Text := '';
    Edit3.Text := IntToStr(c);
    except
```

ShowMessage('Невозможно выполнить операцию! Проверьте введенные данные!');

end;

end;

В данном случае программа попытается (try – попытка, англ.) выполнить команды написанные до служебного слова except и если какая-либо из этих команд не сможет быть выполнена, то произойдет исключение и управление передастся командам, описанным между служебными словами except и end – в нашем случае это вывод окошка, сообщающего о том, что введенные данные некорректные. Таким образом, если мы теперь введем некорректные данные в поля ввода (например, текст или дробные числа, вместо целых чисел) и нажмем кнопку «Сложение», то нам будет выдано сообщение о некорректности введенных данных, и программа не зависнет.

Существует также следующая конструкция обработки исключения:

Try ...Finally ...End;

Она отличается от предыдущей следующим: в блоке *Try ...Except* описываются команды, которые могут вызвать ошибку, а в блоке *finally ...end* описываются команды, которые будут выполнены **даже** в случае ошибки. Т.е. эти команды не будут устранять ошибку, они будут передавать управление другим командам, которые будут выполнять устранение ошибки.

Эти две конструкции можно комбинировать:

```
Try
...
    Try
    ...
    Finally
    ...
    End;

Except
...
End;
```

или:

```
Try
...
    Try
    ...
    Except
    ...
    End;

Finally
...
End;
```

Задание: самостоятельно сделайте обработку исключений для всех обработчиков событий выполнения арифметических и тригонометрических операций, которые мы написали ранее.