Основные средства языка программирования VB.NET

(1 лекция и 4 сдвоенных лабораторных занятия, контрольная работа, зачет)

Лектор: доцент Воробейчиков Леонид Александрович

Лабораторные занятия:

доцент Сосновиков Георгий Константинович

Кафедра информатики (комн. 523)

Темы лабораторных работ

1. **Основные средства языка программирования**

VB .NET. Типы данных (файл Тема 01–02)

2. Структура VB-программ и процедуры. Средства программирования алгоритмов линейной структуры (файл Тема 01–03)

3. Программирование алгоритмов разветвляющихся структур (файл Тема 01–04)

Контрольная работа №1

Программирование алгоритмов регулярных
циклических структур и циклических

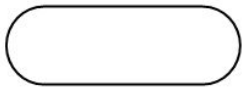
Методические указания и оргвопросы

1. Теоретические материалы и задания к лабораторным и контрольным работам содержатся в соответствующих файлах электронного пособия в компьютерных классах.
2. Отчеты по работам должны быть изготовлены в MS Word в формате А4 с титульным листом и представлены в бумажном виде. Содержание отчета указано в описаниях соответствующих работ.
3. В порядке исключения допускается сдача лабораторных работ без представления отчетов, с предъявлением работающей программы на кафедральном или личном компьютере. Контрольная работа представляется только в

Методические указания и оргвопросы

4. При сдаче лабораторной работы без представления отчета схема алгоритма программы должна быть нарисована от руки на отдельном листе.
5. Если лабораторная работа не выполнена в отведенные по расписанию часы, то она сдается на зачетном занятии с обязательным представлением бумажного отчета.
6. Ввиду ограниченного количества компьютеров в учебных классах приветствуется выполнение работ на личных компьютерах.
7. Зачет по дисциплине ставится по совокупности всех выполненных лабораторных работ и контрольной работы.

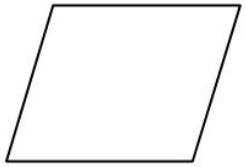
Изображения блоков на схемах алгоритмов



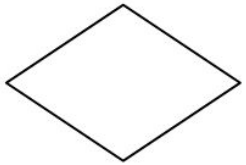
- начало или конец процесса обработки данных

- процесс (элементарная операция по обработке данных)

- предопределенный процесс (процедура)



- ввод-вывод данных



- решение (разветвление)

- начало цикла с параметром

Типы данных в VB

Тип	Значение	Байт	Диапазон
SByte	Целое число	1	от -128 до 127
Short	Целое число	2	от -32768 до 32767
Integer	Целое число	4	от -2147483648 до 2147483647
Long	Целое число	8	от -9223372036854775808 до 9223372036854775807
Single	Число с пл. точкой	4	от -3.4028235E38 до 3.4028235E38
Double	Число с пл. точкой	8	от -1.79769313486231E308 до 1.79769313486231E308
Char	Символ	2	любой символ Unicode в диапазоне 0 - 65535
String	Текст	2	От 0 до ≈ 2 миллиардов символов Unicode
Boolean	Логическое знач.	2	True (истина) или False (ложь)

Важно: арифметические данные **целых типов** хранят **точные** значения чисел,
а **вещественных типов** (с плавающей точкой) – **приближенные** значения

Объявление переменных

Переменные, используемые в программе, должны быть заранее объявлены. Для объявления переменных используется оператор **Dim**, имеющий следующий формат:

Dim *имена_переменных* **AS** *тип*, *имена_переменных* **AS** *тип*, …

Например, следующие операторы объявляют переменные разных типов, в дальнейшем используемые в некоторой программе:

```
Dim Name As String
Dim Кол As  Integer,  b  As  Single
Dim Ном, d  As  Integer
```

Инициализация переменных при объявлении. Объявление констант.

При объявлении переменные могут быть проинициализированы:

```
Dim Name As String = “Петров”
Dim Кол As  Integer = 0
Dim Eps  As  Single = 0.00001
```

Инициализация переменной возможна только в том случае, если она единственная в операторе объявления.

Объявление именованных констант производится с помощью следующего оператора:

Const *имя_константы* **AS** *тип = значение константы*

Например:

```
Const Pi AS Single = 3.14159
```

Оператор присваивания

Значения переменных в ходе выполнения программы обычно изменяются с помощью *оператора присваивания*, имеющего следующий формат:

идентификатор = выражение

Оператор заменяет значение переменной, стоящей слева от знака присваивания, новым значением – вычисленным значением выражения, стоящего справа от знака присваивания. Например:

```
a = 1
b = a
Name = “Иванов”
n = n + 1
c = a + b
```

Арифметические операции

Операция	Приоритет	Тип действий
()	1	Вычисление в круглых скобках
Функция	2	Вычисление значения функции
^	3	Возведение в степень
Унарные + -	4	Изменение знака в случае -
* /	5	Умножение и деление
\	6	Целочисленное деление без остатка
Mod	7	Остаток от деления
+ -	8	Сложение и вычитание

Встроенные функции

Функция (метод)	Возвращаемое значение
Abs(x)	Абсолютное значение x
Acos(x)	Значение арккосинуса x
Asin(x)	Значение арксинуса x
Atan(x)	Значение арктангенса x.
Ceiling(x)	Целое число, которое больше или равно аргументу
Cos(x)	Значение косинуса x
Exp(x)	Экспонента - e^x
Fix(x)	Целая часть числа (дробная часть отбрасывается)
Floor(x)	Целое число, которое меньше или равно аргументу
Int(x)	Значение целой части x (ближайшее меньшее)
Log(x[,n])	Логарифм по основанию n
Log10(x)	Десятичный логарифм
Max(x, y)	Максимальное из двух чисел
Min(x, y)	Минимальное из двух чисел
Pow(x, y)	Возведение x в степень y
Round(x, n)	Округление до n знаков после запятой
Sin(x)	Значение синуса x
Sqrt(x)	Квадратный корень из x
Tan(x)	Значение тангенса x

Неявные преобразования типов данных

При вычислении выражений операнды операций должны быть одинакового типа. Аналогичным образом, в операторе присваивания тип переменной в левой части оператора должен совпадать с типом выражения в правой части оператора. На практике эти условия обычно не выполняются, поэтому компилятор VB производит необходимые преобразования данных. Такие преобразования называются *неявными*. Преобразования могут быть *расширяющими* и *сужающими*.

Расширяющие преобразования не приводят к потерям данных. К ним относятся, например, преобразования:

Integer → Long, Integer → Single, Single → Double.

Сужающие преобразования могут приводить к потерям данных. К ним относятся, например, преобразования:

Long → Integer, Single → Integer, Double → Single.

Кроме того, к сужающим относятся преобразования строковых данных в числовые и обратно.

Использование неявных сужающих преобразований чревато случайными потерями данных из-за “недогляда” программиста. Этого можно избежать, запретив такие преобразования использованием установки

Option Strict On

Она разрешает все расширяющие преобразования, а все сужающие преобразования заставляет программиста делать осознанно, явно вызывая нижеописанные функции преобразования типов:

Функции явного преобразования типов данных

CDbl(x) преобразует значение параметра **x** к типу **Double**.

CInt(x) преобразует значения параметра **x** к целочисленному типу **Integer**. Если в качестве параметра **x** передано дробное число, то дробная часть просто округляется по правилам математики.

CLng(x) возвращает значение типа **Long**. Обработка параметра **x** производится аналогично функции **CInt(x)**.

CShort(x) преобразует значение параметра **x** к типу **Short**.

CSng(x) применяется для преобразования значения параметра **x** к типу **Single**.

CStr(x) используется для преобразования данных в строковый тип **String**.

Следует иметь в виду, что при использовании функций преобразования из строковых данных в числовые разделителем целой и дробной части в строке с числом должна быть запятая.

Кроме указанных функций для преобразования данных могут использоваться следующие функции преобразования данных VB:

Val(x) преобразует значение параметра **x** строкового типа **String** к числовому значению типа **Double**.

Str(x) преобразует значение числового параметра **x** в строку **String**.

Пример задания к лабораторной работе №1

$$z = \frac{0.002 - \ell^{x^y}}{(100 - y)(x + 2)}; \qquad k = z; \qquad m = [z],$$

где:

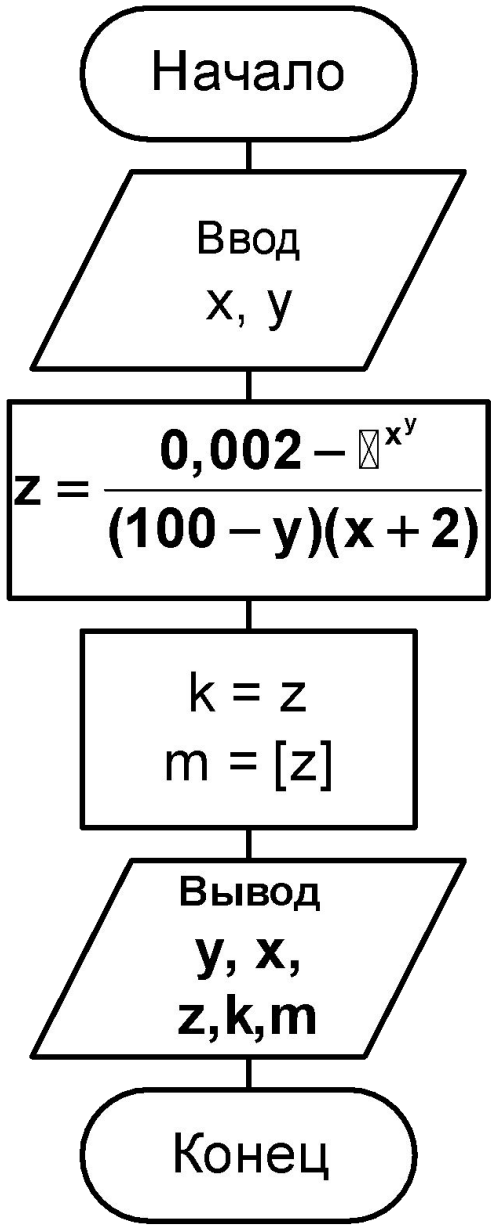
x, y и **z** – вещественные переменные типа **Double**

m – вещественная переменная типа **Single**

k – целая переменная типа **Integer**

[] – целая часть числа

Схема алгоритма



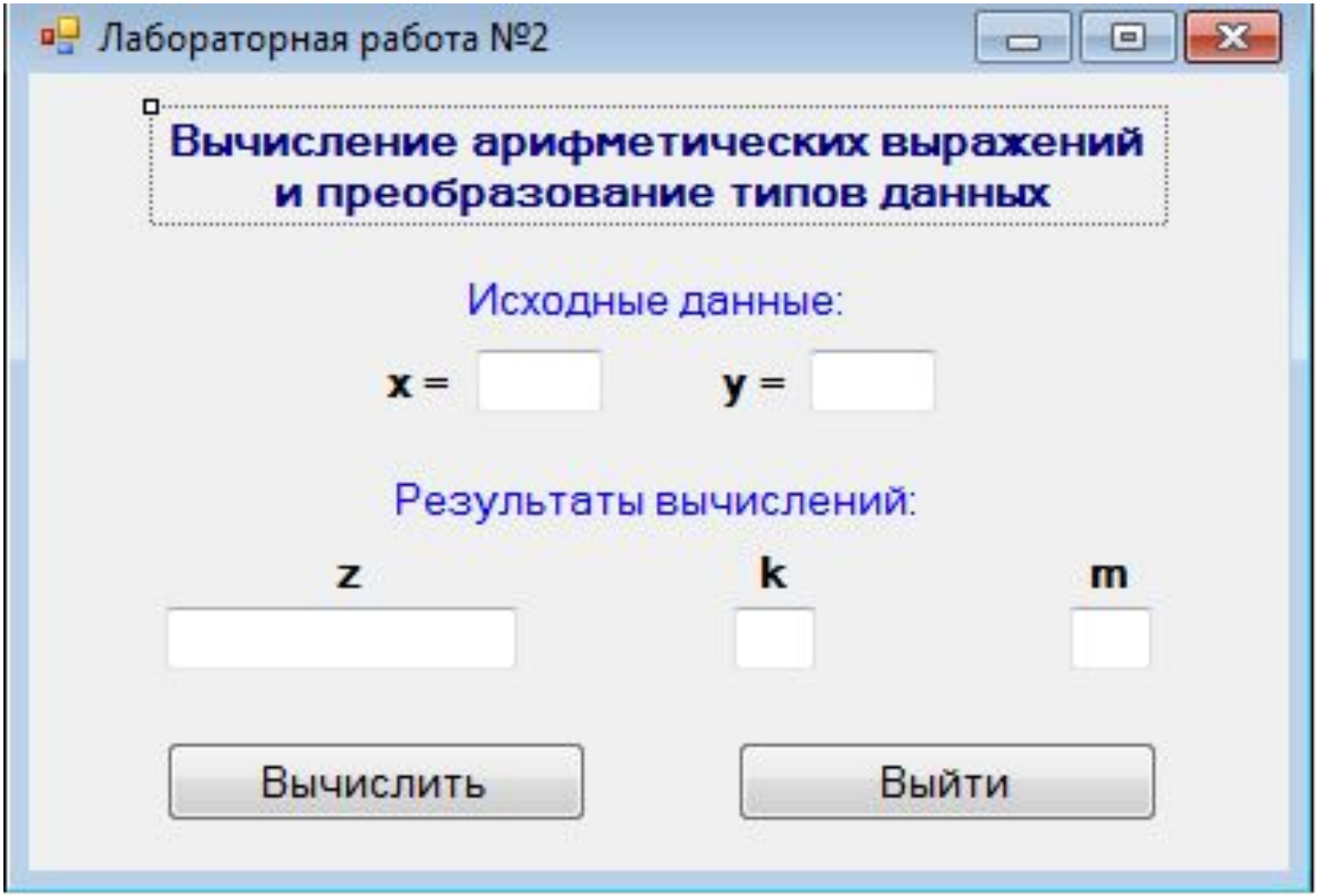
Запись вычислений на VB

z = (0.002 – Exp(x^y)) / ((100 – y)*(x + 2))

k = z

m = Fix(z)

Форма для организации интерфейса



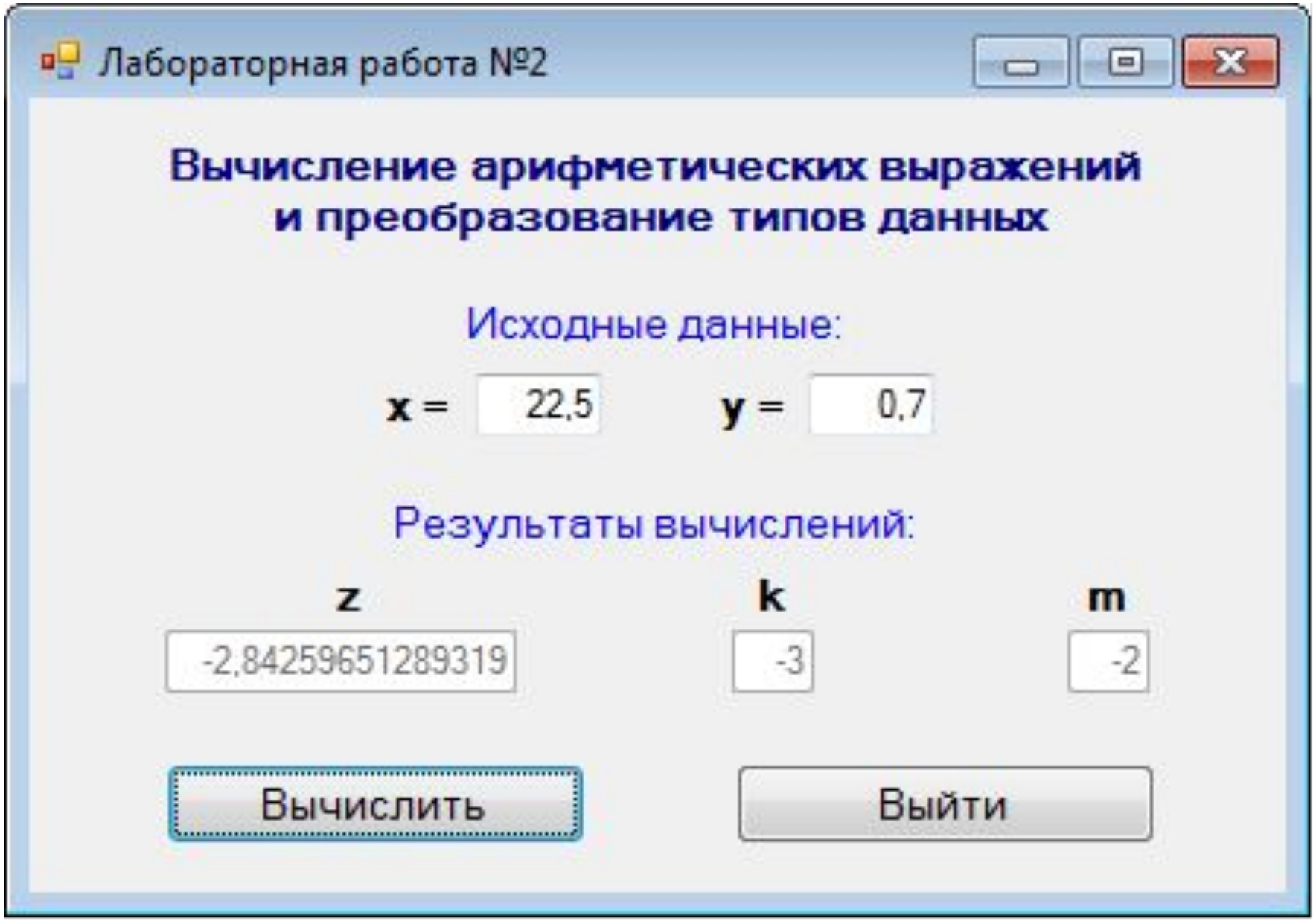
Программный код первого проекта

```
Imports System.Math      'подключение математического модуля
Public Class frmLab2
    Private Sub cmdCalc_Click(sender As Object, e As EventArgs)_
                                Handles cmdCalc.Click
        Dim x, y, z As Double, m As Single, k As Integer
        x = txtX.Text ‘неявное преобразование String в Double
        y = txtY.Text ‘неявное преобразование String в Double
        z = (0.002 - Exp(x ^ y)) / ((100 - y) * (x + 2))
        k = z          'неявное преобразование Double в Integer
        m = Fix(z)     ‘неявное преобразование Double в Single
        txtZ.Text = z ‘неявное преобразование Double в String
        txtK.Text = k ‘неявное преобразование Integer в String
        txtM.Text = m ‘неявное преобразование Single в String
     End Sub

    Private Sub cmdExit_Click(sender As Object, e As EventArgs) _
                                    Handles cmdExit.Click
         End
    End Sub
End Class
```

Программный код второго проекта

```
Option Strict On
Imports System.Math      'подключение математического модуля
Public Class frmLab2
    Private Sub cmdCalc_Click(sender As Object, e As EventArgs)_
                                Handles cmdCalc.Click
        Dim x, y, z As Double, m As Single, k As Integer
        x = CDbl(txtX.Text) 'преобразование типа String в Double
        y = CDbl(txtY.Text)
        z = (0.002 - Exp(x ^ y)) / ((100 - y) * (x + 2))
        k = CInt(z) 'преобразование типа Double в тип Integer
        m = CSng(Fix(z)) 'преобразование типа Double в тип Single
       'преобразование числовых типов в String
        txtZ.Text = CStr(z)
        txtK.Text = CStr(k)
        txtM.Text = CStr(m)
    End Sub

    Private Sub cmdExit_Click(sender As Object, e As EventArgs) _
                                    Handles cmdExit.Click
          End
    End Sub
EndClass
```

Результаты выполнения обоих проектов



Процедурное программирование

Процедурное программирование – это разбиение программного кода на отдельные, желательно небольшие, функциональные блоки с целью:

- упрощения кода для лучшего понимания алгоритма;
- повторного использования кода во избежание его многократного повторения в тексте программы;
- разделения кода на понятные специализированные блоки, намного более легкие в понимании и обслуживании, чем очень длинный монолитный код.

Повторяющиеся фрагменты программы (либо не повторяющиеся, но представляющие собой логически целостные вычислительные блоки) оформляются в виде так называемых ***процедур***. В этом случае в тексте основной программы, вместо помещённого в процедуру фрагмента, вставляется ***оператор вызова процедуры***. При выполнении такой инструкции выполняется вызванная процедура, после чего исполнение программы продолжается с инструкции, следующей за командой вызова процедуры.

Собственно же процедуры содержат последовательность операторов для выполнения конкретной функциональной задачи или ее фрагмента. При этом в ходе выполнения программы любая процедура может быть вызвана из любой точки произвольное число раз.

Процедурное программирование

Использование процедур предполагает, что они должны обязательно быть определены (описаны) и вызваны путем обращения по имени. Таким образом, каждой процедуре должно быть присвоено уникальное имя. Кроме того, для вызываемой процедуры может быть установлен перечень входных и выходных параметров.

Входные параметры процедуры – это переменные, значения которых при вызове этой процедуры определены (им присвоены допустимые значения) и которые участвуют в вычислительном процессе (реализации) процедуры.

Выходные параметры процедуры – это переменные, содержащие результаты ее выполнения.

В VB имеются два основных типа процедур:

Процедуры-функции или процедуры типа **Function** вызываются из процедур событий или других пользовательских процедур по имени. Они обычно используются для вычисления значений различных выражений, могут получать входные параметры и предназначены для возврата только одного значения.

Процедуры-подпрограммы или процедуры типа **Sub** также вызываются из процедур событий или других пользовательских процедур по имени. Они могут получать входные параметры и возвращать произвольное количество выходных параметров, полученных в результате работы процедуры.

Описание процедуры-функции

Описание процедуры-функции имеет следующий вид:

```
Function ИмяФункции (ФормальныеПараметры) As ТипВозЗначения
            …
    ОператорыФункции
            …
    Return значение    или    ИмяФункции = значение
            …
End Function
```

ИмяФункции – это уникальное имя создаваемой процедуры-функции.

ТипВозЗначения – это тип значения, возвращаемого функцией, то есть тип величины, которая является результатом работы функции.

ФормальныеПараметры – это список необязательных аргументов, разделенных запятыми и используемых в данной функции. Каждый аргумент должен быть объявлен с указанием конкретного типа данных. По умолчанию VB добавляет к каждому аргументу ключевое слово **ByVal** (***передача параметров по значению***), которое указывает на то, что в функцию через данный аргумент передается ***копия значения фактического параметра***, и все изменения значения этого аргумента не будут возвращены в вызывающую процедуру.

Описание процедуры-функции

ОператорыФункции – это блок операторов, который выполняет работу функции.

Return – это оператор, с помощью которого можно указать место, где в блоке кода функции требуется возвратить *значение* в вызывающую программу, и величину этого возвращаемого значения. После выполнения этого оператора происходит выход из процедуры-функции и управление передается в то место программы, откуда эта процедура-функция была вызвана.

Пример процедуры-функции

Например, следующая функция возвращает сумму квадратов двух величин типа **Double**:

```
Function SumD(ByVal a AS Double, ByVal b AS Double) AS Double
        Return a^2 + b^2
End Function
```

Вызов процедуры-функции производится аналогично вызову встроенных функций. При этом вместо формальных параметров “подставляются” соответствующие по порядку и типу значения фактических параметров.

Например:

```
……
Dim x, y, z AS Double
x = 2.5
y = -1.75
z = 2 * x * y + SumD(x, y)/2
……
```

Описание процедуры-подпрограммы

Описание процедуры-подпрограммы имеет следующий вид:

```
Sub ИмяПроцедуры (ФормальныеПараметры)
        …
    ОператорыПроцедуры
        …
End Sub
```

ИмяПроцедуры – это имя создаваемой процедуры-подпрограммы.

ФормальныеПараметры – это список необязательных аргументов, разделенных запятыми. Каждый аргумент должен быть объявлен с указанием конкретного типа данных.

Описание процедуры-подпрограммы

Формальные параметры в процедуре-подпрограмме могут быть как входными, так и выходными. К входным параметрам должно быть добавлено ключевое слово **ByVal** (значение по умолчанию), означающее, что в процедуру-подпрограмму через данный аргумент передается копия значения, и все изменения значения этого аргумента не будут возвращены в вызывающий код.

К выходным параметрам должно быть добавлено ключевое слово **ByRef** (***передача параметров по ссылке или адресу***), означающее, что в процедуру передается адрес соответствующего фактического параметра в вызывающей программе, по которому подпрограмма должна записать возвращаемый результат.

ОператорыПроцедуры – это блок операторов, который выполняет работу процедуры по заданному алгоритму.

Пример процедуры-подпрограммы

Например, следующая процедура-подпрограмма решает ту же задачу, что и процедура-функция в предыдущем примере:

```
Sub SumD(ByVal a AS Double, ByVal b AS Double, ByRef c AS Double)
        c = a^2 + b^2
End Sub
```

Вызов процедуры-подпрограммы производится отдельным оператором, содержащим имя вызываемой подпрограммы и заключенный в скобки список фактических параметров, соответствующий по количеству, порядку и типу со списком формальных параметров подпрограммы.

Например:

```
……
Dim x, y, z, r AS Double
x = 2.5
y = -1.75
SumD(x, y, r)
z = 2 * x * y + r/2
……
```

Использование процедур для ввода-вывода данных

Процедуры-функции удобно использовать для организации ввода данных, а процедуры-подпрограммы – для вывода результатов вычислений. Например, для ввода и преобразования в тип **Double** строкового значения, находящегося в объекте типа **TextBox** с заданным именем может быть использована такая процедура-функция:

```
Function GetDbl(ByVal T AS TextBox) AS Double
        Return Val(T.Text)
End Function
```

Для вывода и преобразования значения типа **Double** в объект типа **TextBox** с заданным именем может быть использована такая процедура-подпрограмма:

```
Sub PutDbl(ByVal Z As Double, ByVal T As TextBox)
        T.Text = CStr(Z)
End Sub
```

Области видимости переменных

При работе с процедурами важно знать так называемые ***области видимости переменных***. Оператор, объявляющий переменную, сообщает Visual Basic, не только *что* будет храниться в этой переменной, но и *где* эту переменную можно использовать. Область, где используется переменная, и называется областью видимости переменной. Если переменная доступна, то она существует, но не наоборот. Переменная может существовать в памяти и быть доступной для некоторых частей программного кода и при этом быть недоступной (невидимой) для других.

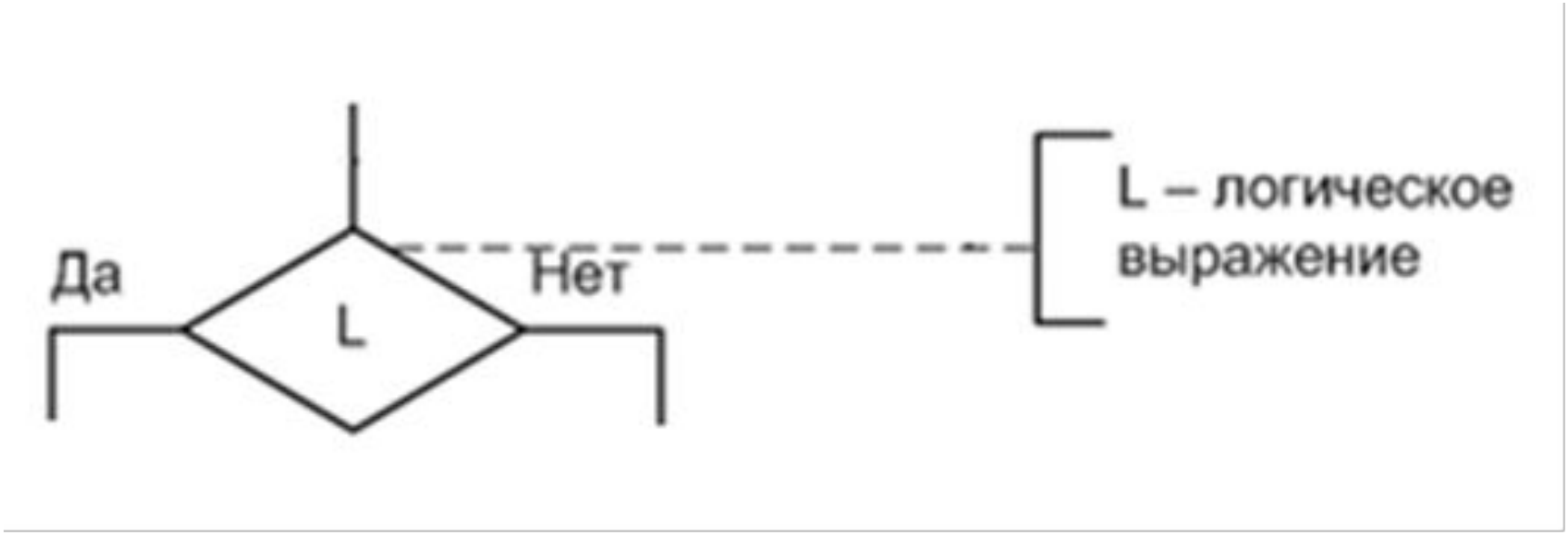
При работе с единственной формой проекта можно объявить переменную для работы с ней в пределах одной процедуры или в любой процедуре данной формы.

С помощью ключевого слова **Dim** объявляют ***локальные переменные***, которые существуют только во время вызова процедуры, где они объявлены. Но если переменная с помощью **Dim** или **Public** объявлена в разделе глобальных объявлений формы (до любого объявления событийной процедуры или процедуры пользователя), то она является ***глобальной*** и будет доступна во всех процедурах этой формы.

Современный стиль программирования рекомендует обходиться без глобальных переменных. В лабораторной работе №3 глобальные переменные используются с чисто ознакомительной целью.

Разветвляющиеся структуры

На практике при решении различных задач вычислительный процесс всегда разветвляется в зависимости от некоторых условий. Например, при вычислении действительных корней квадратного уравнения по известным вам формулам вычислительный процесс разветвляется на 3 направления в зависимости от значения дискриминанта уравнения D: при **D > 0** корни различны и вычисляются каждый по своей формуле; при **D = 0** корни одинаковы и вычисляются по общей формуле; при **D < 0** действительных корней не существует, о чем необходимо вывести соответствующее сообщение.



Логический тип данных и логические выражения

В VB определен логический тип данных **Boolean** (от фамилии английского математика *George Boole*, одного из основателей математической логики). Данные этого типа могут принимать одно из двух значений: **True** (истина) или **False** (ложь). Логические данные могут быть представлены в программе в форме констант **True** и **False**, переменных и выражений.

Логические переменные, как и переменные числовых типов, перед использованием в программе должны быть объявлены. При объявлении возможна инициализация переменной одной из двух логических констант. Неинициализированные переменные при объявлении получают значение **False**. Например:

```
Dim Flag As Boolean
Dim Check As Boolean = True
```

Логическое выражение может быть *простым* или *сложным*.

Простое логическое выражение – это два арифметических или строковых выражения, связанных *операцией отношения* (*сравнения*).

Операции отношения

Операции отношения	Значение
=	*Равно*
<	*Меньше, чем*
>	*Больше, чем*
<=	*Меньше или равно*
>=	*Больше или равно*
<>	*Не равно*

Сложное логическое выражение состоит из простых логических выражений, переменных и констант, связанных *логическими операциями*.

Логические операции

Логическая операция	Результат
Not (НЕТ, отрицание)	Преобразует значение **True** в **False** и наоборот – **False** в **True**
And (И,конъюнкция)	Результат операции **True**, если оба операнда имеют значение **True**
Or (ИЛИ,дизъюнкция)	Результат операции **True**, если хотя бы один из операндоа имеет значение **True**
Xor (исключающее ИЛИ)	Результат операции **True**, если только один из операндоа имеет значение **True**

Значения операндов		Not x	X And y	X Or y	x Xor y
X	Y		$(x \wedge y)$	$(x \vee y)$	$(x \oplus y)$
T	T	F	T	T	F
T	F	F	F	T	T
F	T	T	F	T	T
F	F	T	F	F	F

Однострочный оператор **If**

Разветвляющиеся алгоритмические структуры в языках программирования реализуются *операторами условного перехода***.**

Однострочный оператор **If** имеет следующий формат:

If L Then *оператор(ы)* [**Else** *оператор(ы)*]

Порядок выполнения оператора следующий: если значение логического выражения **L** равно **True**, то выполняется оператор (или операторы, отделенные друг от друга двоеточием), стоящий после **Then**, в противном случае выполняются операторы, стоящие после **Else**. Конструкция **Else** является необязательной и может отсутствовать (в формате она взята в квадратные скобки).

Особенностью однострочного оператора **If** является то, что он должен располагаться в одной строке программы. Например:

```
If x > 0 Then y = Sqrt(x) Else y = 1
If d >= 10 Then epsilon = 0.001

If lambda < 1 OR beta = 3 Then lambda = beta : x = 0
```

Блочный оператор **If**

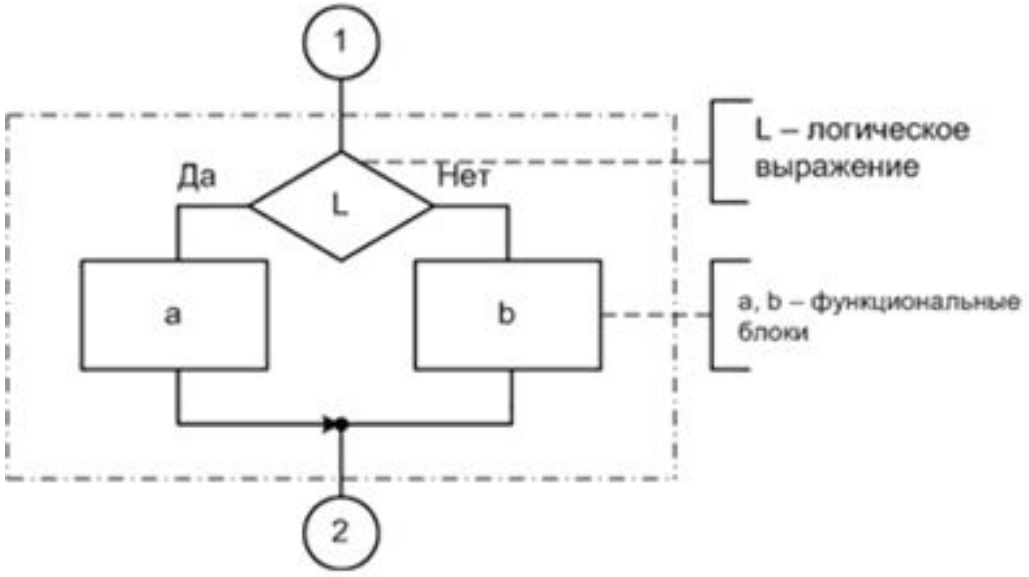
Блочный (многострочный) оператор **If** имеет следующий формат:

If L1 **Then**
блок_операторов_1
[ElseIf L2 **Then**
*блок_операторов_2***]**
……
[Else
*блок_операторов_n***]**
End If

Примеры блочного оператора **If**

```
If x < 10 Then
        y = 1
        z = 2
ElseIf x > 5 Then
        y = 2
        z = 1
Else
        y = 0
        z = 0
End If

If x>0  Then
        y = Sqrt(x)
        z = 1 + y
        p = 4
End If
```

Стандартное разветвление



If L Then

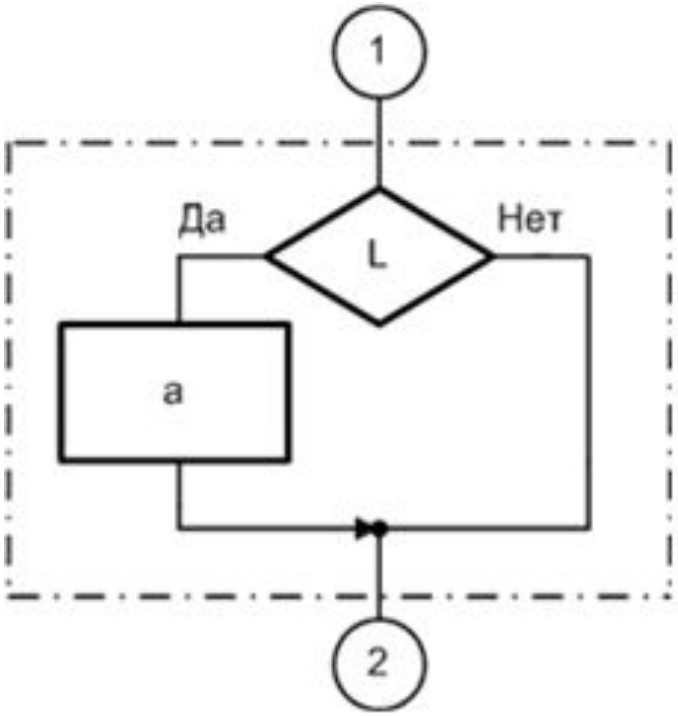
операторы блока a

Else

операторы блока b

End If

Усеченное разветвление



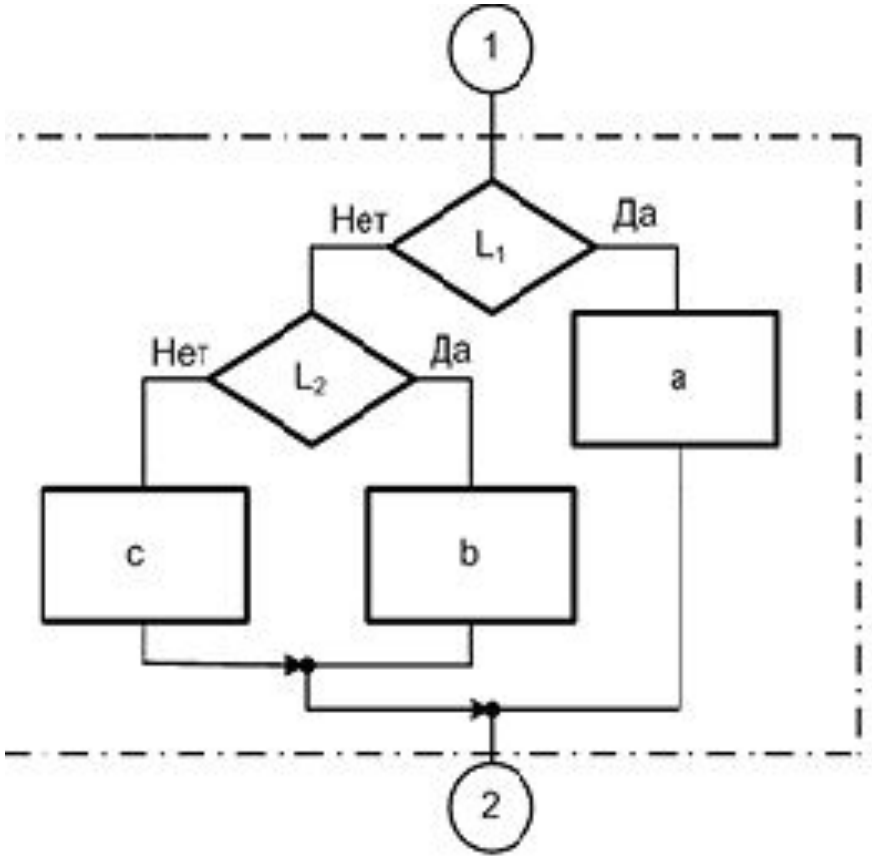
Усеченное разветвление тоже программируется блочным **If** без ветви **Else**:

If L Then
операторы блока a
End If

Однако, если блок *a* содержит единственный оператор, то удобнее использовать компактный однострочный **If**:

If L Then *оператор блока a*

Вложенное разветвление



If L_1 Then
операторы блока a
ElseIf L_2 Then
операторы блока b
Else
операторы блока c
End If

Циклические структуры и их программирование

Циклическими называются алгоритмические структуры (программы), в которых предусмотрена возможность многократного повторения выполнения участка алгоритма (программы). Этот участок называется *телом цикла*.

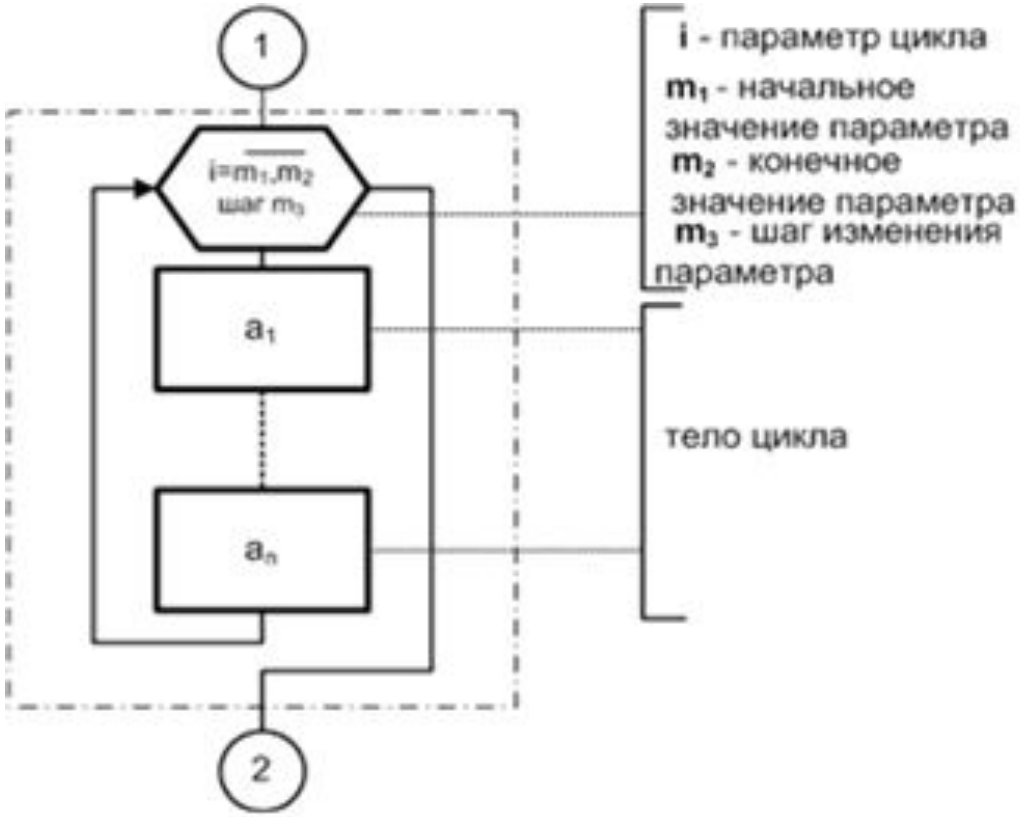
Различают циклы двух видов: с заранее известным и с заранее неизвестным числом повторений цикла.

Циклы, в которых число повторений цикла заранее известно или может быть определено до начала выполнения цикла, называются *регулярными*.

Циклы, в которых число повторений цикла заранее неизвестно, а определяется только в процессе выполнения алгоритма (программы), называются *итеративными*.

Циклы организуются с помощью базовых циклических структур и их программной реализации. В любой циклической структуре помимо тела цикла всегда присутствует оператор проверки условия и обеспечения выхода из цикла. Но это не оператор разветвления (**If** в VB), а один из специальных операторов, специфических для каждой базовой структуры.

Регулярный цикл с параметром



For *параметр = выражение1* **To** *выражение2* [**Step** *выражение3*]

Оператор_1

……/

[Exit For]

……

Оператор_n

Next *параметр*

Пример. Постановка задачи

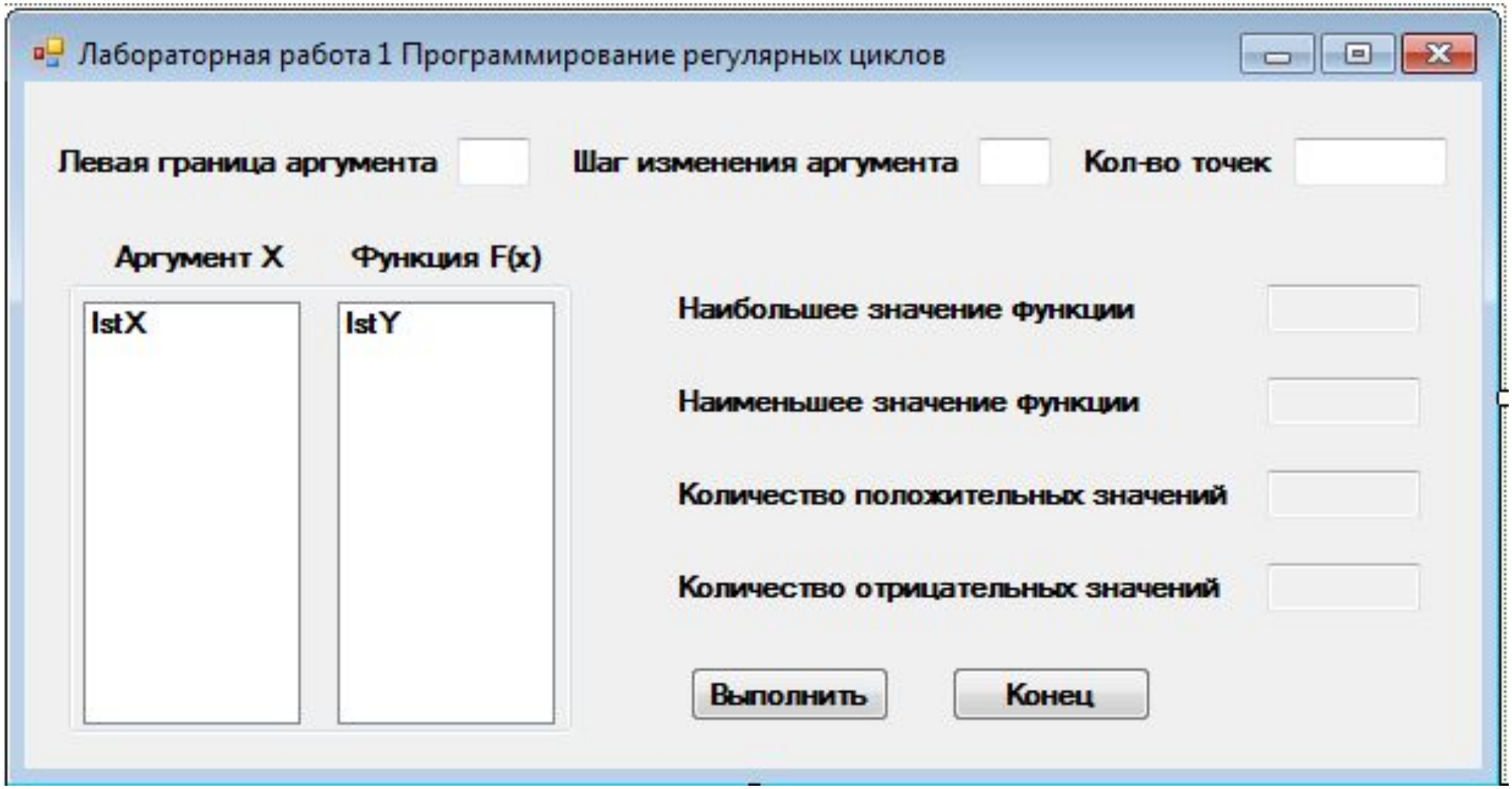
Построить таблицу значений функции

$$y = 12 \cdot \sin(x+2) + 15 \cdot \cos(4 \cdot x)$$

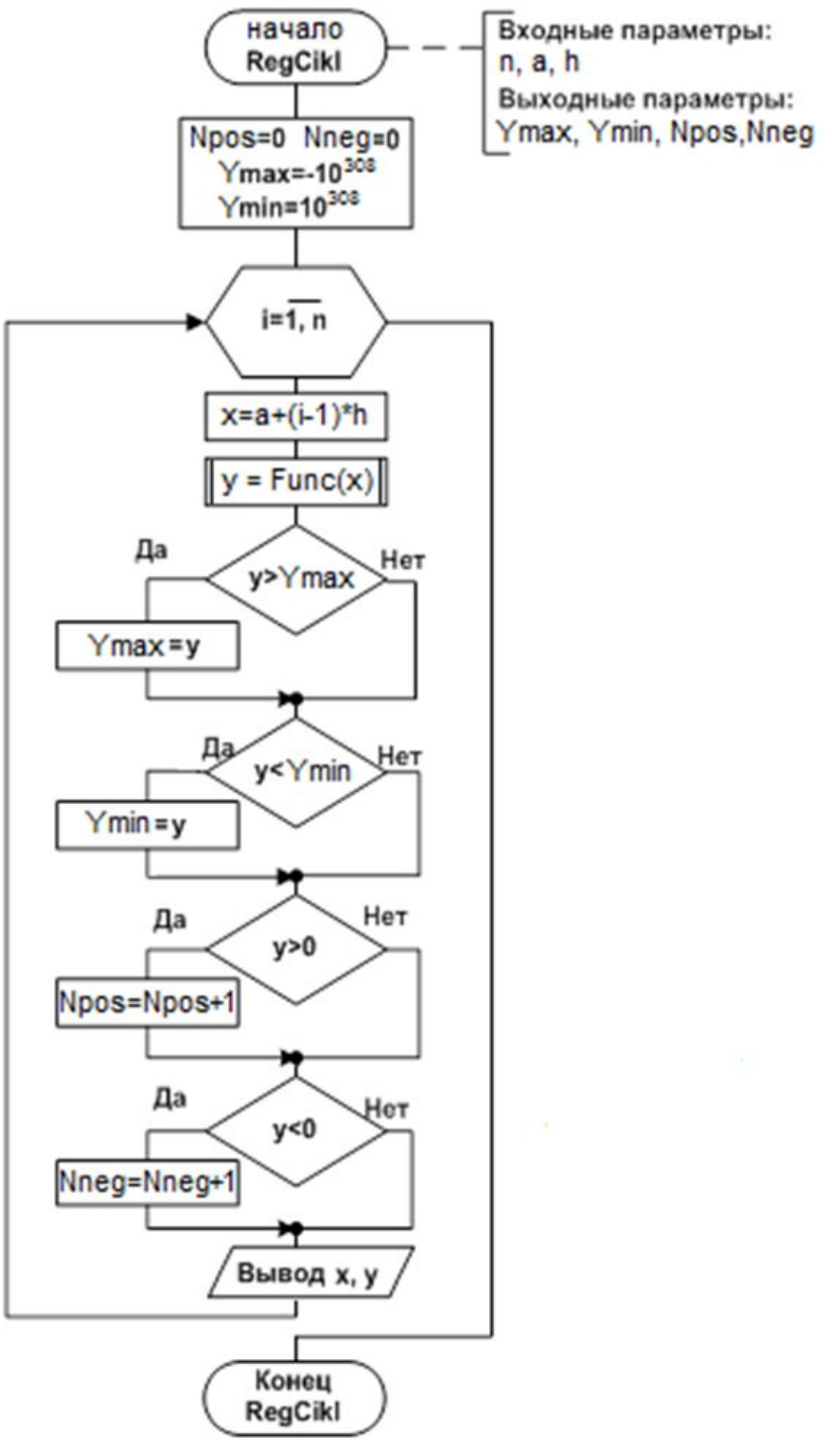
при изменении **x** на отрезке **[-1;1]** с шагом **0.1**.

Вычислить наибольшее и наименьшее значение функции на отрезке, а также количество положительных и количество отрицательных значений функции.

Пример. Интерфейс пользователя



Пример: Схема алгоритма процедуры **RegCikl**



Пример. Программный код

```
Option Strict On
Imports System.Math
Public Class Form1
    'Функция ввода вещественных исходных данных из TextBox
    Function GetDouble(ByVal T As TextBox) As Double
        Return Val(T.Text)
    End Function
    'Функция ввода целых исходных данных из TextBox
    Function GetInt(ByVal T As TextBox) As Integer
        Return CInt(T.Text)
    End Function 'Процедура вывода вещественного результата в TextBox
    Sub PutDouble(ByVal Z As Double, ByVal T As TextBox)
        T.Text = Format(Z, "00.0000")
    End Sub
    'Процедура вывода целого результата в TextBox
    Sub PutInt(ByVal K As Integer, ByVal T As TextBox)
        T.Text = CStr(K)
    End Sub
    'Процедура вывода вещественного результата  в ListBox
    Sub PutList(ByVal Z As Double, ByVal LB As ListBox)
        LB.Items.Add(Format(Z, "00.0000"))
    End Sub
    'Процедура-функция вычисления значения функции
    Function Func(ByVal x As Double) As Double
        Return Sin(x + 2) * 12 + Cos(x * 4) * 15
    End Function
```

Пример. Программный код (продолжение)

```
'Процедура решения задачи
Sub RegCikl(ByVal n As Integer, ByVal a As Double, ByVal h As Double, _
            ByRef Ymax As Double, ByRef Ymin As Double, _
            ByRef Npos As Integer, ByRef Nneg As Integer)
    Dim i As Integer, y As Double, x As Double
    Npos = 0 : Nneg = 0
    Ymax = Double.MinValue : Ymin = Double.MaxValue

    For i = 1 To n
        x = a + (i - 1) * h
        y = Func(x)
        If y > Ymax Then Ymax = y
        If y < Ymin Then Ymin = y
        If y > 0 Then Npos = Npos + 1
        If y < 0 Then Nneg = Nneg + 1
        PutList(x, lstX) : PutList(y, lstY)
    Next
End Sub
```

Пример. Программный код (окончание)

```
    Private Sub cmdExec_Click(ByVal sender As Object, ByVal e As EventAr
        Dim a As Double, h As Double
        Dim n, Pos, Neg As Integer
        Dim Fmax, Fmin As Double
        a = GetDouble(txtA)
        h = GetDouble(txtH)
        n = GetInt(txtN)
        RegCikl(n, a, h, Fmax, Fmin, Pos, Neg)
        PutDouble(Fmax, txtMax)
        PutDouble(Fmin, txtMin)
        PutInt(Pos, txtPos)
        PutInt(Neg, txtNeg)
    End Sub
    Private Sub cmdEnd_Click(ByVal sender As System.Object, ByVal e As S
        End
    End Sub
End Class
```

Пример. Результаты выполнения