Язык программирования Visual BASIC

Язык программирования Visual Basic относится к визуальным объектно ориентированным языкам. Первая версия языка бейсик была представлена в 1964 г. в Дартмутском колледже. Авторами нового алгоритмического языка были Джон Кемени и Томас Курте. Основной идеей заложенной разработчиками в язык Basic, стала простота взаимодействия человека и программы. Название BASIC является аббревиатурой английской фразы *«Beginner's All -purpose Symbolic Instruction Code»,* что в переводе означает «многоцелевой язык, символических команд для начинающих». Данный язык предназначался для учебных целей, но на сегодняшний день версия языка Visual Basic.Net является полноценным языком программирования, позволяющим решать широкий круг задач стоящих перед программистом. Вместе с тем язык, остался достаточно прост для изучения, и может применяться в обучении программированию.

Существует бесплатная Express версия программы, предназначенная для ознакомительных и учебных целей. Скачать данную версию, как и другие продукты фирмы Microsoft, можно с официального сайта разработчика, при этом от вас потребуется пройти несложную процедуру регистрации.

Следует заметить, что язык Visual Basic входит в пакет средств разработки Visual Studio и может быть скачан в рамках данного пакета.

Язык Visual Basic позволяет создавать приложения следующих типов:

1) Приложения Windows Form (программы с графическим интерфейсом)
2) Консольные приложения (интерфейс программы представлен командной строкой)
3) Приложения WPF (приложения с графическим интерфейсом, построенным с применением технологии DirectX)

Помимо Windows приложений язык Visual Basic позволяет создавать программные продукты для операционной системы Mac, а также заниматься разработкой web продуктов.

Интерфейс среды разработки

Внешний вид окна проекта Windows Form показан на рис. 5.

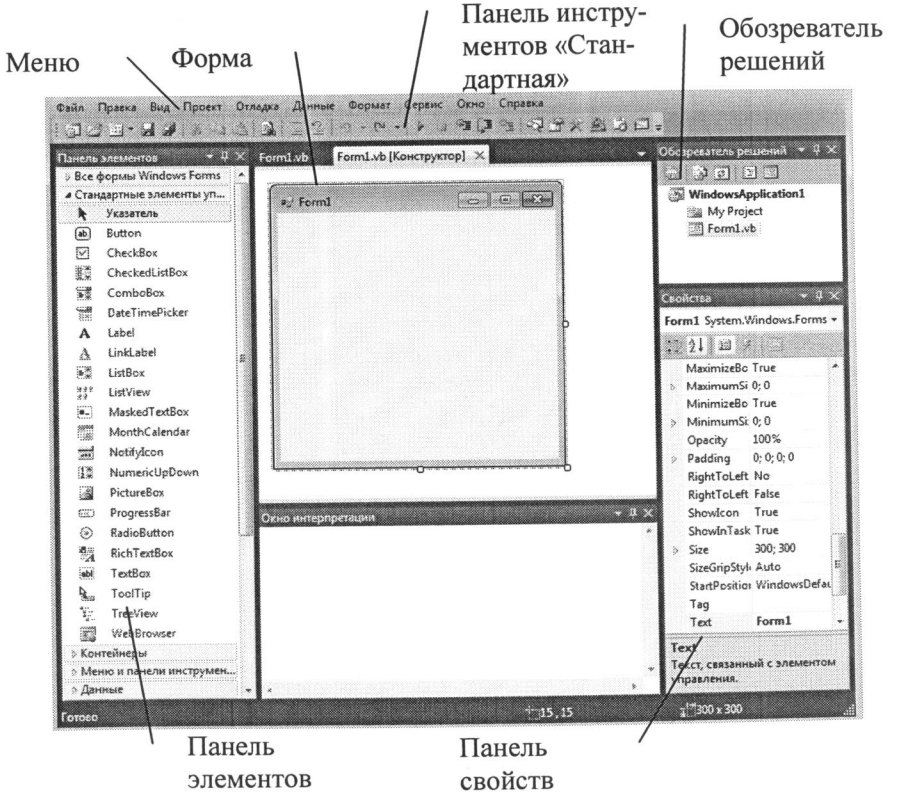

Рис. 5.

В центральной части окна находится форма, с помощью которой будет создаваться интерфейс бушующей программы. На форме могут быть расположены различные объекты, содержащиеся в панели элементов.

Свойства выделенных объектов отображаются в окне **Свойства**, расположенном в правой части экрана. На рис. 5 показаны свойства, единственного объекта в данном проекте – формы. Свойства позволяют изменять параметры объектов, находящихся на форме (цвета, шрифты, размеры, выравнивание текста, положение объекта на форме и т.д.). Для разных классов объектов набор свойств будет разный, но некоторые свойства являются общими для экземпляров разных классов. Список основных свойств, объектов приведен в таблице 2.

Таблица 2. Свойства объектов.

Название	**Описание**
Cursor	Задает внешний вид курсора при наведении мыши на элемент, результат виден только во время выполнения программы
Location	Задает координаты левого верхнего угла объекта. Начало координат находится в левом верхнем углу формы.
BackColor	Установка цвета фона
BorderColor	Устанавливает цвет границ элемента
BorderStyle	Устанавливает внешний вид границ элементов
Enabled	Определяет работоспособность элемента после запуска приложения
Font	Установка параметров шрифта (размер, гарнитура, начертание)
ForeColor	Установка цвета текста
Name	Указывает имя используемое в коде для идентификации объекта.
Size	Устанавливает высоту и ширину объекта. Значение свойства задается двумя числами, отделенными запятой. Первое число задает ширину, второе высоту объекта.

Text	Хранит значение текста, отображаемого внутри элемента.
Textalign	Устанавливает выравнивание текста в объекте.
Visible	Параметр, определяющий видимость элемента после запуска приложения

Перечень свойств в панели можно сгруппировать по категориям, нажав кнопку **По категориям**  или вывести свойства в алфовитном порядке нажав кнопку  .

В таблице 3 приведены некоторые типы объектов, которые будут использоваться в данном пособии.

Таблица 3.

Объект	Описание
Button	Командная кнопка, как правило используется для запуска выполнения процедур.
CheckBox	Позволяет пользователю выбрать или отменить выбор соответствующего параметра на форме.
DataGridView	Отображает столбцы и строки данных в сетке, пользователь может менять содержимое ячеек.
Label	Содержит в себе текстовую информацию, служащую пояснением к текстовым полям формы, может использоваться для вывода данных.
PictureBox	Используется для размещения изображений на форме.
ProgressBar	Показывает уровень выполнения операции.
RadioButton	Позволяет пользователю выбрать определённый параметр из группы в комбинации с другими флаговыми кнопками.
TextBox	Позволяет вводить текст. Используется

	для ввода и вывода данных.
TrackBar	Позволяет пользователю выбрать диапазон значений с помощью ползунка.

Создайте новый проект Windows Form, назовите его ***Первая работа.*** У открывшейся формы найдите свойство **Text** и вместо *Form1* введите фразу *работа 1*. Перенесите на форму объекты, показанные на рис. 6.

Выделите все текстовые поля (**textbox**), в окне свойств найдите строку **Size** и введите значение **80; 20**. Используя свойство **BackColor** измените цвет заднего фона для текстовых полей. В самом нижнем тестовом поле установите красный цвет шрифта – свойство **ForeColor** и полужирное начертание – свойство **Font**.

Установите размер кнопок **100;20**.

Выровняйте объекты с помощью линий отслеживания, при выравнивании по строкам используйте фиолетовые линии, по столбцам – синие.

Для изменения текста внутри объектов используйте свойство **Text**.

Сохраните проект.

Лексика языка

При записи текстов программ на языке Visual Basic разрешается использовать заглавные и строчные буквы латинского алфавита цифры знак подчеркивания и ограничители.

Ограничители представляют собой знаки пунктуации, знаки операций, разделители и служебные (зарезервированные) слова.

Назначение знаков пунктуации показано в таблице 4.

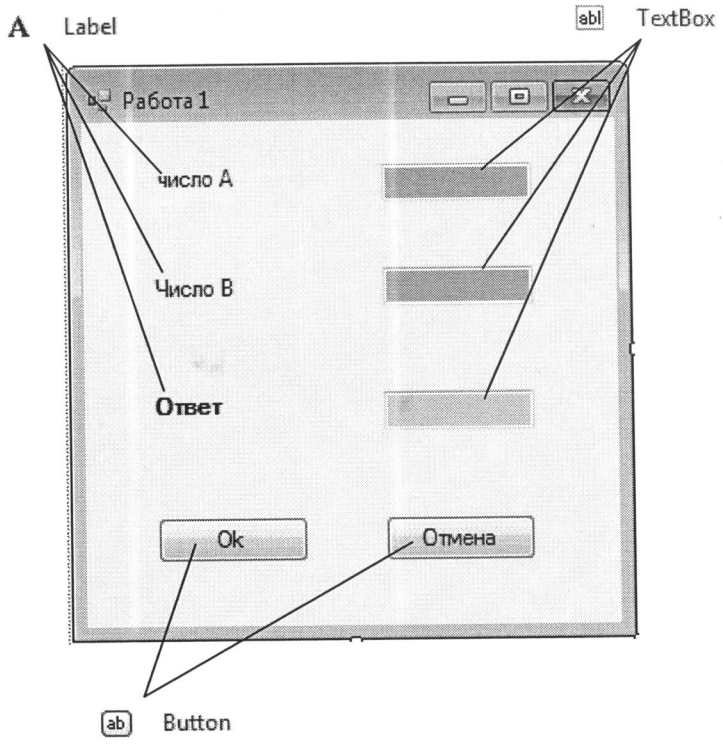

Рис. 6.

Таблица 4. Знаки пунктуации.

Знаки	Назначение
'	Апостроф — признак комментария. Текст после апострофа до конца строки поясняет алгоритм и не является его частью
,	Разделение списков значений, параметров процедуры и функции
( )	Задание индексов массива, выделение части выражения, задание списков параметров
?	Служебный знак символьного образца
*	Служебный знак символьного образца
[]	Ограничители множества значений для символьного

	образца
_	Признак переноса оператора на следующую строку
.	Десятичная точка, разделитель целой и дробной части
;	Разделитель выражений в операторах ввода-вывода
:	Разделитель операторов в одной строке

Таблица 5. Знаки операций:

Знаки	**Назначение**
+	Сумма
-	Разность
/	Деление
\	Деление нацело
*	Умножение
^	Возведение в степень
=	Равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно
<>	Неравно

В качестве разделителя выступает знак пробела.

Служебные слова могут быть использованы только по своему специальному назначению.

Таблица 6. Служебные слова.

And	**Boolean**	**Byte**	**Call**
Case	**Choose**	**Const**	**Date**
Dim	**Do**	**Double**	**Each**
Else	**End**	**Eqv**	**Exit**
Explicit	**For**	**Function**	**Get**
GoSub	**GoTo**	**If**	**Imp**
Integer	**Is**	**Let**	**Like**
Long	**Loop**	**Mod**	**Module**
New	**Next**	**Not**	**Object**

Option	Or	Private	Property
Public	ReDim	Return	Select
Set	Short	Single	Static
Stop	String	Sub	Switch
Then	Uinteger	Ulong	Ushort
Wend	While	With	Xor

Программный код является регистронезависимым и может быть записан как в верхнем, так и в нижнем регистре.

Переменные и типы данных

Переменные

Это именованная область оперативной памяти, хранящая в себе данные. Значения переменных может меняться в процессе выполнения программы. Объем памяти, занимаемой переменной зависит от выбранного типа данных, для хранимых значений.

В Visual Basic переменные именуются с помощью идентификаторов, длина которых может достигать 255 символов (они должны начинаться с буквы, за которой могут следовать другие буквы, цифры или символы подчеркивания). Регистр символов и наименований переменной значения не имеет.

Типы данных

В языке Visual Basic, существуют следующие типы данных:

Таблица 7. Типы данных

Название типа	Область изменения данных	Занимаемый размер в байтах	Коментарий
Целые типы данных			
Byte	0..255	1	Целое без знака
SByte	-128..127	1	Целое со знаком
Short	$-2^{15}..2^{15}-1$	2	короткое целое число со знаком

UShort	$0..2^{16}-1$	2	короткое целое число без знака
Integer	$-2^{31}..2^{31}-1$	4	Целое со знаком
UInteger	$0..2^{32}-1$	4	Целое без знака
Long	$-2^{63}…2^{63}-1$	8	длинное целое число со знаком
Ulong	$0..2^{64}-1$	8	длинное целое число без знака
Вещественные типы данных			
Single	–3,4028235E+38 до –1,401298E–45 для отрицательных значений; 1,401298E–45 до 3,4028235E+38 для положительных значений;	4	число одиночной точности с плавающей запятой
Double	–1,79769313486231570E+308 до –4.94065645841246544E–324 для отрицательных значений; от 4,94065645841246544E–324 до 1,79769313486231570E+308 для положительных значений	8	число двойной точности с плавающей запятой
Decimal	От 0 до +/–7,9... E + 28 без десятичной запятой; от 0 до +/–7,9..E-28 с десятичной запятой	16	
Прочие типы данных			
Boolean	**True** или **False**	2	Логический тип данных
Date	От 1 января 0001 года до 31 декабря 9999 года.	8	Формат даты и времени
Char	От 0 до 65535	2	Кодирует один символ
String	От 0 до ~ 2*109	Зависит от платформы	строка переменной длины
Object	Может хранить значения любого типа	4(32) 8(64)	

Целочисленные типы данных. В Visual Basic доступны 8 целочисленных типов данных, отличающихся диапазоном возможных значений и возможностью записи отрицательных чисел.

Вещественный тип данных. В Visual Basic определены три стандартных вещественных типа: Single, Double и Decimal.

Каждый тип имеет свой диапазон значений и точность.

Выбор конкретного типа для переменной связан с требуемой точностью вычислений.

Логический (булевский) тип данных. Данные логического типа (Boolean) в стандарте языка могут принимать одно из двух значений: True или False. Переменная или константа логического типа занимает в Visual Basic 2 байта, в которые записывается 0, если переменная или константа имеет значение false, и любое целое, отличное от 0, в противном случае.

Когда прочие числовые типы данных преобразуются в тип Boolean, значение 0 воспринимается как False, а любое другое значение становится значением True. Если значения типа Boolean преобразуются в значения других (числовых) типов, то значение False становится значением 0, а True становится -1.

Тип данных Дата. Переменные типа Дата представляют собой 8-байтовые представления в форме с плавающей точкой календарных дат в интервале от 1 января 100 года до 31 декабря 9999 года с составляющей времени в интервале от 0:00:00 до 23:59:59. Константы типа Дата должны справа и слева ограничиваться знаком «#», например, #3/15/1993#.

Тип данных String. Строковый тип данных позволяет хранить последовательности символов — строки. Строки могут быть переменной и фиксированной длины.

Теоретически такой тип данных позволяет хранить строковые переменные длиной до 2 миллиардов символов. Однако, на конкретном компьютере это число может быть гораздо меньше из-за ограниченных объемов оперативной памяти или ресурсов операционной системы.

Тип данных Object. Переменные этого тип данных содержат в себе не значения, а ссылки на объекты. Переменная **Object** может также ссылаться на данные любого типа значения. Использование такого типа данных, как Object, замедляет работу программы, так как требуется время и ресурсы для выполнения операций преобразования типов.

Объявление переменных

Назначение типа данных для переменных происходит в

процессе их объявления. В Visual Basic существуют следующие уровни определения переменной.

Уровень процедуры. Переменная является локальной и может использоваться только в рамках той процедуры, в которой ее объявили. Объявление процедуры происходит с применением функции **Dim**, внутри процедуры.

Dim x **As** Integer *(объявляет и размещает в памяти локальную переменную x с типом данных Integer)*
или

Dim a, b, c **As** String (*Объявляет три локальные переменные a, b и c строкового типа данных*).

Уровень модуля. Переменные могут использоваться во всех процедурах в рамках модуля, в котором их объявили. Применение в других модулях проекта невозможно. Для объявления таких переменных используют оператор Dim, расположенный в разделе описания модуля.

Dim x, y **As** Integer.

Уровень проекта. Глобальные переменные, которые могут использоваться во всех процедурах и модулях проекта. Описание переменной происходит при помощи оператора Public размещенного в разделе описания модуля.

Public a **As** String.

Идентификаторы типов данных

При объявлении переменных тип данных можно не указывать. Для того чтобы переменная была отнесена к определенному типу, можно использовать так называемые идентификаторы типов — специальные символы, добавляемые справа к идентификатору, задающему имя переменной.

Таблица 8. Идентификаторы типов данных.

Тип данных	**Знак**	**Пример**
Integer	%	Dim X%

Long	&	Dim M&
Single	!	Dim X!
Double	#	Dim X#
Decimal	@	Const Pi@ = 3.14
String	$	Dim W$ = “word”

Операции

Все операции в Visual Basic можно разделить на следующие типы:

- математические;
- логические;
- операции сравнения;
- операции с битами информации.

Приоритет выполнения операций

Если в выражении заданы операции более чем одной категории, то существует следующий порядок их выполнения: сначала выполняются арифметические операции, затем операции сравнения и последними выполняются логические операции.

Операции сравнения имеют один уровень приоритета и выполняются в порядке следования слева направо. Арифметические и логические операции выполняются в соответствии с порядком, приведенным в таблице:

Операция конкатенации строк (&) не относится к арифметическим операциям, но по приоритету выполнения находится как раз между арифметическими операциями и операциями сравнения (т.е. выполняется после арифметических операций, но перед операциями сравнения).

Таблица 9.

Арифметические операции	**Операции сравнения**	**Логические операции**
Возведение в степень (^)	Равенство (=)	Not
Признак отрицательного числа(-)	Неравенство(о)	And

Умножение и деление (*, /)	Меньше, чем (<)	Or
Деление нацело (\)	Больше, чем (>)	Xor
Остаток от деления (Mod)	Меньше либо равно (<=)	Eqv
Сложение и вычитание (+, -)	Больше либо равно (>=)	Imp
Конкатенация строк (&)	Like, Is	

Арифметические операции

Арифметические операции могут применяться только к операндам целых и вещественных типов.

Таблица 10.

Знак	Операция	Описание
^	Возведение в степень	x^y — возведение x в степень y. Значение x может быть отрицательным только при целом y
-	Признак отрицательного числа	Меняет значение операнда на значение противоположного знака
+	Сложение	Результат — сумма двух чисел
-	Вычитание	Результат — разность двух чисел
*	Умножение	Результат — произведение двух чисел
/	Деление	Результат — частное от деления двух чисел
\	Деление целых чисел	Результат — целая часть от деления целых чисел: 27\6 = 4
mod	Остаток от деления целых чисел	Результат — остаток от деления нацело: 27 mod 6 = 3 Если один из операндов — вещественный, перед выполнением операции происходит округление его до целого числа, например: 27 mod 6.7 = 4

В качестве операндов арифметических операций могут выступать стандартные арифметические (или тригонометрические) функции, т.е. функции с результатом целого или вещественного типа, аргументами которых являются выражения целого или вещественного типа. В таблице приведены характеристики некоторых арифметических и тригонометрических функций.

Таблица 11.

Функция	Назначение	Тип аргумента	Тип результата
Abs(X)	Модуль (абсолютная величина) аргумента	Целый Вещественный	Целый Вещественный
Atn(X)	Арктангенс аргумента	Вещественный	Вещественный (вычисляется в радианах)
Cos(X)	Косинус аргумента	Вещественный	Вещественный Abs(Cos(X))<=1
Exp(X)	Возведение числа e (основание натурального логарифма) в степень X	Вещественный, X<= 09.782712893; *e* считается равным 2.718282	Вещественный
Fix(X)	Целая часть числа. Для отрицательных чисел — минимальное целое число, большее или равное X	Целый или вещественный	Целый Например: Fix(-2.8) = -2 Fix(2.8)= 2
Int(X)	Целая часть числа. Для отрицательных чисел — минимальное целое число, меньшее или равное X	Целый или вещественный	Целый Например: Int(-2.8) = -3 Int(2.8) = 2
Log(X)	Натуральный логарифм аргумента	Целый или вещественный; e считается равным 2.718282	Вещественный
Rnd[(number)]	Генератор случайных чисел	Целый	Вещественный, 0= Rnd <1
Sgn(X)	Индикатор знака числа	Целый или вещественный	-1, если X <0; 0, если X = 0; 1,если X>0.
Sin(X)	Синус аргумента	Вещественный	Вещественный Abs(Sin(X))<=1
Tan(X)	Тангенс аргумента	Вещественный	Вещественный (в радианах)
Sqr(X)	Квадратный корень аргумента	Вещественный, X>=0	Вещественный
Pi	Число π		Вещественный

Для того чтобы получить последовательность случайных

чисел в заданном интервале [*нижняя_грана, верхняя_граница*], используют следующую формулу:

*Int((<верхняя_граница> – <нижняя_граница> + 1) * Rnd + <нижняя_граница>)*

Например, чтобы сгенерировать случайные числа в интервале от 20 до 100, необходимо записать следующую последовательность действий:

```
X = Int((100-20) * Rnd) + 20).
```

Операции сравнения

В результате выполнения операций сравнения получается значение логического типа: *true* или *false*. Приведенная ниже таблица показывает для каждой операции, выполняемой над операндом 1 (Xl) и операндом 2 (X2), значение результата в зависимости от значений операндов. Если хотя бы один из операндов при выполнении любой операции принимает стандартное значение Null, то результатом выполнения операции тоже будет величина, равная Null.

Таблица 12.

Операция	**Описание**	**True(истина), если**	**False(ложь), если**
<	Меньше	X1 <X2, например: 2 < 5 — истина	X1 >= X2, например: 5 < 5 — ложь
<=	Меньше либо равно	X1 <= X2, например: 2 <= 5 — истина	X1 > X2, например: 5 <= 2 — ложь
>	Больше	Xl > 02, например: 5 > 2 — истина	X1 <= 02, например: 5 > 5 — ложь
>=	Больше либо равно	X1 >= X2, например: 5 >= 2 — истина	01 <02, например: 2 >= 5 — ложь
=	Равно	X1 = X2, например: 5 = 5 — истина	X1 <> X2, например: 2 = 5 — ложь

<>	Не равно	X1 <> X2, например: 2 <> 5 — истина	X1 = X2, например: 5 <> 5 — ложь

При сравнении операндов следует учитывать, что правила сравнения чисел или символьных строк всегда корректно действуют только в том случае, если сравниваемые операнды однотипны, например, оба операнда — числовые значения или оба операнда — символьные строки. Если же операнды разного типа, то в силу вступят следующие правила выполнения операций сравнения:

- если один из операндов числового типа, а второй — Empty, то сравнение проводится по правилам сравнения чисел и значение Empty принимается равным 0;
- если один из операндов типа String, а второй — Empty, то сравнение проводится по правилам сравнения строк и значение Empty принимается равным пустой строке ("").

К операциям сравнения относят также и операции Is и Like, которые имеют специфические области применения.

Операция Is применяется для сравнения двух переменных A и B типа object. Если обе переменные ссылаются на один физический объект в памяти, то в результате операции A Is B получается значение True. В противном случае результат равен False.

Операция Like служит для выявления соответствия между значением переменной строкового типа и так называемым *образцом*. Под образцом здесь понимается строка, содержащая специальные символы, трактующиеся в соответствии со следующими правилами:

Таблица 13.

Символ в образце	**Соответствие в строке**
?	Любой символ, например: "aBa" Like "a?a" возвращает True
*	Любое количество символов (или ни одного), например: "BAT123456" Like "B?T*" возвращает True

#	Любая цифра (0-9), например: "год 2015" Like "год ####" возвращает True
[список_символов]	Любой символ из списка, например: "X" Like "[A-Z]" возвращает True "X" Like "[A-WY-Z]" возвращает False
[!список_символов]	Любой символ, не находящийся в списке, например: "9" Like "[!A-Z]"возвращает True "X" Like "[!A-WY-Z]" возвращает False

Логические операции

Логические операции применяются к операндам логического типа, т.е. в качестве операндов выступают выражения, при вычислении дающие результат логического типа. Результат выполнения логических операций тоже логического типа. Вычисление логических выражений происходит в соответствии с *таблицами истинности* логических операций. Таблицы истинности задают соответствие между значениями операндов и результатом выполнения операции.

Помимо значений True и False в таблицы истинности операций добавлено стандартное значение Null. Результат выполнения логической операции со значением Null в одном или в обоих операндах также может быть равным значению Null.

Примеры записи логических выражений:

```
(X>=0) and (X<=1)
((X>0) and (X<0.5)) or (X>3)
(N mod 2=0) imp (N>0)
```

При использовании логических выражений в качестве условий (например, в условных операторах и операторах цикла) значение логического выражения **Null** приравнивается к значению **False.**

Операции с битами информации

Операции побитового сравнения выполняются для операндов числового типа и в результате дают число, получающее-

ся путем побитового выполнения операции над операндами.

Рассмотрим примеры выполнения операций побитового сравнения. Пусть объявлены две переменных A и B типа Integer. В переменной A находится число 12, а в переменной B — число 9. Запишем значения переменных в двоичном побитовом представлении и применим операции побитового сравнения:

A=12	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
B = 9	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Not B = not 9 = -10	1	1	1	1	1	I	1	1	1	1	1	1	0	1	1	0
A and B = 12 and 9 = 8	0	0	0	0	0	0	0	0	0	0	0	0	I	0	0	0
A or B=12or 9=13	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
A eqv B= 12 eqv 9 = -6	1	1	1	1	1	I	1	1	1	I	1	1	I	0	1	0
A imp B = 12 imp 9 = -5	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
A xor B= 12 xor 9 = 5	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Линейные алгоритмы

Линейным принято называть вычислительный процесс, в котором операции выполняются последовательно, в порядке их записи. На схеме блоки, отображающие эти операции, располагаются в линейной последовательности.

На рис. 7 показан пример линейного алгоритма, определяющего процесс вычисления арифметического выражения $z = \left(2a^2 + \sqrt{b^3}\right) / \left(a^3 - \sin(b)\right)$ для различных значений переменных a и b. Алгоритм решения данной задачи может быть разбит на три этапа:

1) ввод исходных данных в программу;
2) выполнение операций вычисления;
3) вывод полученных результатов.

Данная последовательность действий будет актуальна для всех задач, рассматриваемых в данном пособии вне зависимости от используемых алгоритмических структур.

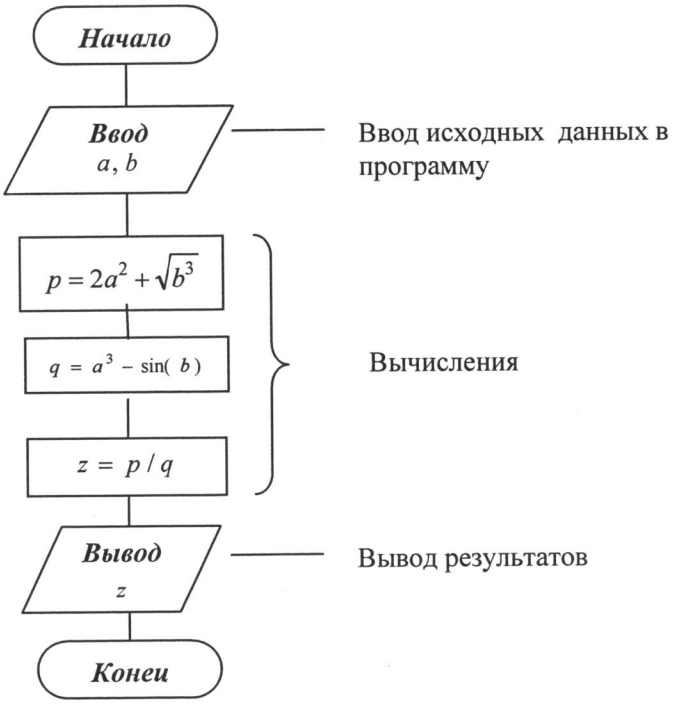

Рис. 7.

Ввод и вывод данных из программы подразумевает взаимодействие программы с человеком. Диалог человека с программой обеспечивает интерфейс приложения. В программах проектов Windows Form интерфейс представлен пользовательской формой с разработки которой, будут начинаться все наши программы. Форма должна быть такой, чтобы вы могли ввести в программу все исходные данные и вывести искомые результаты. Также на форме должны находиться управляющие элементы, реагирующие на события и запускающие соответствующие процедуры обработки, как правило, такими элементами являются объекты принадлежащие классу Button (кнопки).

Ввод и вывод данных может быть осуществлен через свойства объектов, расположенных на форме или через всплы-

вающие окна.

Ввод и вывод данных через свойства объектов.

Для ввода и вывода данных через свойства объектов обычно используют объекты класса TextBox или Label, при этом вводимое(выводимое) значение содержится в свойстве text соответствующего объекта.

Пример ввода переменной через текстовое поле будет иметь следующий вид:

```
a=TextBox1.text
  имя объекта  свойство
```

В приведенном примере переменной **a** присваивается значение, записанное в свойстве **text** объекта **Textbox1**.

Ввод данных с помощью надписи имеет следующий вид:

```
b= Label1.text
```

В этом примере значение переменной **b** берется из свойства **text** надписи **Label1**.

Вывод данных через свойства объектов имеет обратную структуру записи – значение переменной присваивается свойству объекта

```
TextBox1.text =a
  имя объекта  свойство
Label1.text = b
```

Рассмотрим реализацию линейного алгоритма с интерфейсом использующим объекты формы для ввода и вывода данных.

Раскройте файл проекта **Первая работа**, в котором хранится форма рис.6. Данная форма соответствует алгоритму, показанному на рис. 7

Для ввода данных будут использоваться текстовые поля:

для `a` – `TextBox1` (верхнее текстовое поле)

для `b` – `TextBox2` (среднее текстовое поле)

Результат вычисления должен быть помещен в текстовое поле TextBox3. Проверьте соответствие имен текстовых полей данной инструкции. Имена объектов можно узнать в окне свойств соответствующего объекта. Если имена объектов отличаются от указанных здесь, то в программный код будет необходимо внести соответствующие изменения.

Управляющими элементами на данной форме будут две кнопки **Ok** и **Отмена**. Первая кнопка будет запускать процедуру, обеспечивающую выполнение алгоритма обработки, вторая должна закрывать приложение.

Чтобы написать процедуру для кнопки **Ok** дважды щелкните по данной кнопке, при этом в окне проекта откроется вкладка Form1.vb*. В данной вкладке должен содержаться код обеспечивающий запуск процедуры.

При выполнении расчетов нам прийдется обращаться к математической функции SIN( ), чтобы это было возможно, в самом начале файла с исходным кодом добавьте строку

```
Imports System.Math
```

Остольные строки программы будут записаны между следующими строками:

```
Private Sub Button1Click(…)
```
имя объекта Событие

```
End Sub
```

В рамках алгоритма используются пять переменных a, b, p, q, z. Значения всех этих переменных должны принадлежать вещественному типу, поэтому присвоим тип данных Single.

```
Dim a, b, p, q, z As Single '–объявление переменных
   a = TextBox1.Text
   b = TextBox2.Text          '– Ввод исходных данных
   p = 2 * a ^ 2 + b ^ (2 / 3)
                              '– Вычисления
```

```
q = a ^ 3 - Sin(b)
z = p / q
TextBox3.Text = z '–Вывод результатов
```

Запишите преведенный выше код между строками `Private Sub` и `End Sub`.

Перейдите на вкладку с формой (*Form1.vb[Конструктор]**) и дважды щелкните по кнопке **Отмена**. Между появившихся строк `Private Sub Button2_Click()` и `End Sub` напишите команду `End`.

```
Private Sub Button2_Click(..)
      End
End Sub
```

Запустите программу нажав кнопку **Начать отладку**  или нажмите кнопку F5 на клавиатуре. Введите в два верхних текстовых поля числа и нажмите кнопку Ok. В результате выполнения программы в третьем текстовом поле должно появиться число.

Ввод и вывод данных через всплывающие окна

Всплывающие окна для ввода данных создаются с помощью оператора InputBox. Синтаксис данного оператора выглядит следующим образом:

```
a = InputBox("текст сообщения", "название
диалогового окна","значение по умолчанию")
```

Строке с кодом

```
a= InputBox("Введите число A", "Ввод данных", "1")
```

будет соответствовать диалоговое окно показанное на рис. 8.

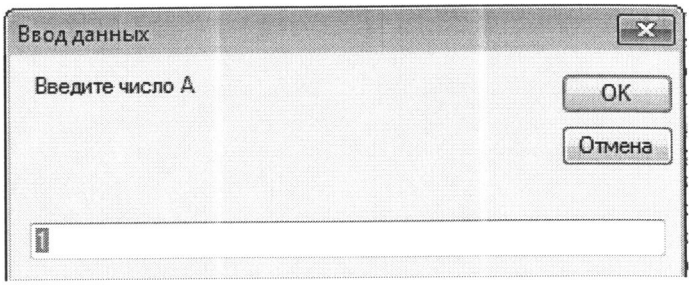

Рис. 8.

Если значение по умолчанию неизвестно, то последнюю пару кавычек можно оставить пустой.

Вывод данных посредством всплывающих окон осуществляется с помощью оператора `MsgBox(текст сообщения, стиль окна, “заголовок окна”)`.

Например, строке с приведенным ниже оператором ,будет соответствовать всплывающее окно, показанное на рис. 9.

```
MsgBox("z=" & z, , "ОТВЕТ")
```

Рис. 9.

В данном примере в тексте сообщения происходит объединение текста “z=” со значением переменной z в одну фразу с помощью оператора &.

Создайте новый файл проекта, назавите его **Работа 1.1**. Создайте форму, показанную на рис. 10.

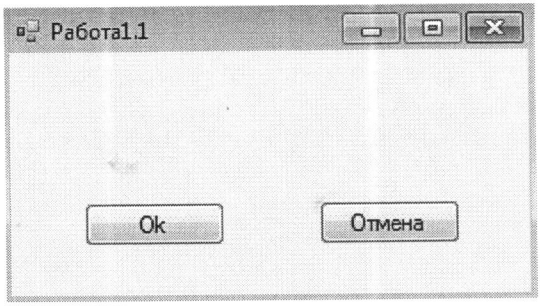

Рис. 10.

Для кнопки **Ok** запишите следующую процедуру.

```
Dim a, b, p, q, z As Single
a = InputBox("Введите число A","Ввод данных", "1")
b = InputBox("Введите число B", "Ввод данных", "")
p = 2 * a ^ 2 + b ^ (2 / 3)
q = a ^ 3 - Sin(b)
z = p / q
MsgBox("z=" & z, , "ОТВЕТ")
```

Для кнопки **Отмена**, введите команду `End`.

Не забудьте вставить в начало файла с кодом строку подключающую математические функции.

```
Imports System.Math
```

Разветвляющиеся алгоритмы

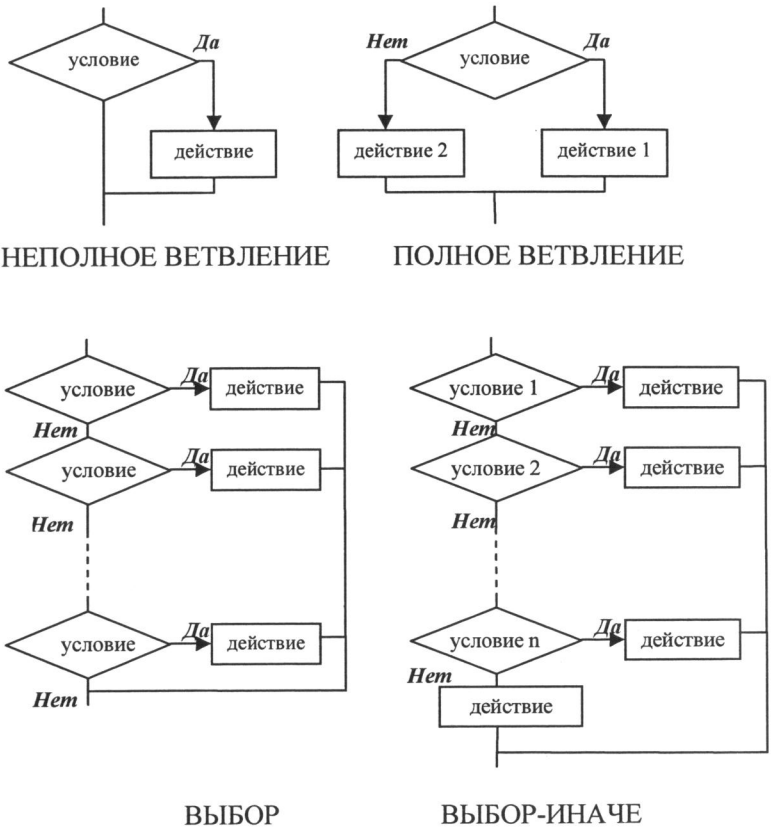

Рис. 11.

Вычислительный процесс называется *разветвляющимся*, если для его реализации предусмотрено несколько направлений (ветвей). Каждое конкретное направление процесса обработки данных является отдельной ветвью вычислений. Ветвление в программе – это выбор одной из нескольких последовательностей команд при выполнении программы. Выбор направления зависит от заранее определенного признака, который может от-

носиться к исходным данным, к промежуточным или конечным результатам. Признак характеризует свойство данных и имеет два или более значений.

Разветвляющийся процесс, включающий в себя две ветви, называется простым, более двух ветвей – сложным. Сложный разветвляющийся процесс можно представить с помощью комбинации простых разветвляющихся процессов. Направление ветвления выбирается логической проверкой, в результате которой возможны два ответа: «да» («+») – условие выполнено и «нет» («–») – условие не выполнено. Любая ветвь, по которой осуществляются вычисления, должна приводить к завершению вычислительного процесса.

Разветвляющиеся алгоритмические процессы существуют в следующих вариантах (см. рис.11): полное ветвление, неполное ветвление и выбор.

Полное и неполное ветвление.

В языке Visual Basic для реализации алгоритмов с полным и неполным ветвлением используется оператор условного перехода **If**.

Синтаксис данного оператора имеет следующий вид

```
If <условие> then
      <Действия ветви 1>
   Else
      <Действия ветви 2>
End If
```

Такая форма записи называется блочной и используется при выполнении достаточно большого количества действий в рамках ветвей. Например:

```
If X>0 then
      Y=X+5
      Z=X^2-y
      W=Cos(z)
   Else
      Y=X^2
      Z=Y-1
```

```
        W=Sin(z)
  End If
```

Следует заметить, что при модульном способе записи, в действиях ветви могут находиться не только линейные алгоритмические структуры, но и разветвляющиеся и циклические.

Если же количество действий небольшое, то оператор **If** может быть записан в строчном виде без использования оператора **End if**.

```
If<Условие>Then<Действия ветви 1>Else<Действия ветви 2>
```

Между разными операциями в рамках одной ветви в такой форме записи ставится знак двоеточие. Например:

```
If A=0 Then X=1:Y=0 Else X=Sin(A):Y=A+X
```

В случае с неполным ветвлением в алгоритме будет отсутствовать вторая альтернативная ветвь, и программный код будет иметь вид показанный ниже:

для модульной формы

```
If <условие> then
      <Действия>
End If
```

для строковой формы

```
If<Условие>Then<Действия>
```

Условие во всех алгоритмах ветвления может состоять из нескольких логических выражений объединенных с помощью логических операторов, например

```
If A=0 and x>0 then y=x
```

Рассмотрим пример программы выполняющей вычисление функции

$$z = \begin{cases} 2u^3 + \sqrt{v}, \ при \ \ x > 0, \\ w^3 - v, \quad при \ \ x \le 0. \end{cases}$$

Значение данной функции будет зависеть от знака переменной x, если $x>0$, то значение функции будет равно $2u^3 + \sqrt{v}$, иначе z будет определяться по формуле $w^3 - v$.

Алгоритм решения данной задачи показан на рис. 13.

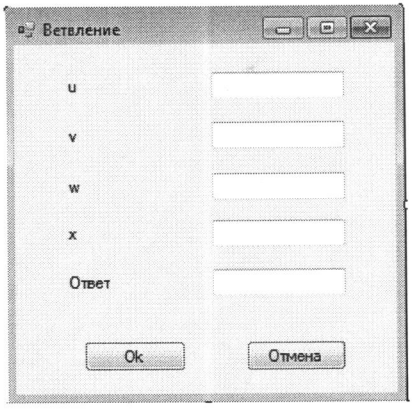

Рис. 12.

Создайте новый проект, назовите его **Ветвление**. В данном проекте создайте форму, показанную на рис. 12.

Для кнопки **Ok** запишите следующую процедуру.

```
Dim u, v, w, x, z As Single
u = TextBox1.Text
v = TextBox2.Text
w = TextBox3.Text
x = TextBox4.Text
If x>0 Then z=2*u^3+v^(1/2) Else z = w^3-v
TextBox5.Text = z
```

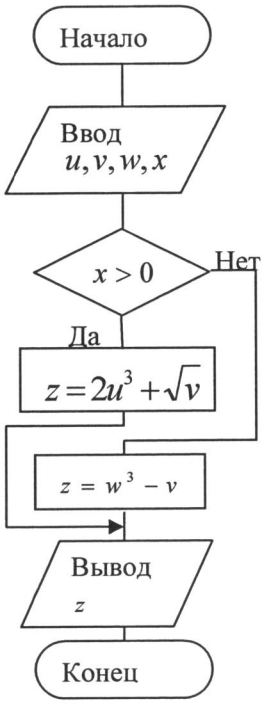

Рис. 13.

Проверьте работу программы, при разных значениях переменной *x*.

Выбор.

Алгоритмы типа выбор применяются в сложных разветвляющихся процессах, где возникает необходимость проверки большого количества условий.

В языке Visual Basic для реализации таких алгоритмов используется оператор **`Select Case`**, формат данной функции имеет следующий вид:

```
Select Case <Переменная>
```

```
        Case <Значение переменной 1>
              <Действие1>
        Case <Значение переменной 2>
              <Действие2>
        ….
        Case <Значение переменной N>
              <ДействиеN-1>
        Case Else
              <ДействиеN>
End Select
```

Ветвь **Case Else** может отсутствовать в конструкции выбор, если все возможные значения переменной были рассмотрены выше.

Значения переменной могут быть заданы следующим образом: одним элементом, списком, диапазоном, условием.

```
Case 2 '- значение задано одним элементом
Case 1,2,3,4,5 '- значение задано списком
Case 1 to 10 '- значение задано диапазоном от 1 до 10
Case Is >10 '- значение задано условием (переменная >10)
```

Рассмотрим пример реализации алгоритма определяющего фамилию студента по введенному номеру зачетной книжки.

Алгоритм работы программы приведен на рис. 15.

Создайте новый проект назовите его Выбор. Создайте форму, показанную на рис. 14. Для кнопки **Ok** напишите следующую процедуру обработки:

```
Dim n As UInteger
Dim fio As String
n = TextBox1.Text
Select Case n
    Case 123
        Fio = "Иванов А.А."
    Case 124
        Fio = "Петров Б.Б."
```

```
            Case 125
                Fio = "Сидоров В.В."
            Case 126
                Fio = "Кузнецов Г.Г."
            Case 127
                Fio = "Игнатьев Е.Е."
            Case Else
                Fio = "такой номер зачетки отсут-
ствует"
        End Select
        TextBox2.Text = Fio
```

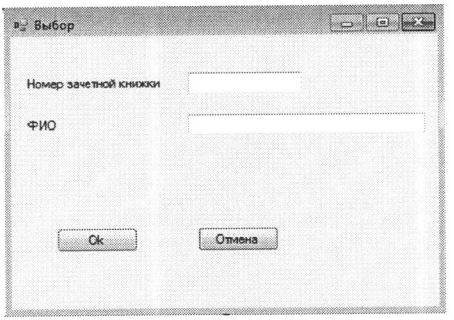

Рис. 14.

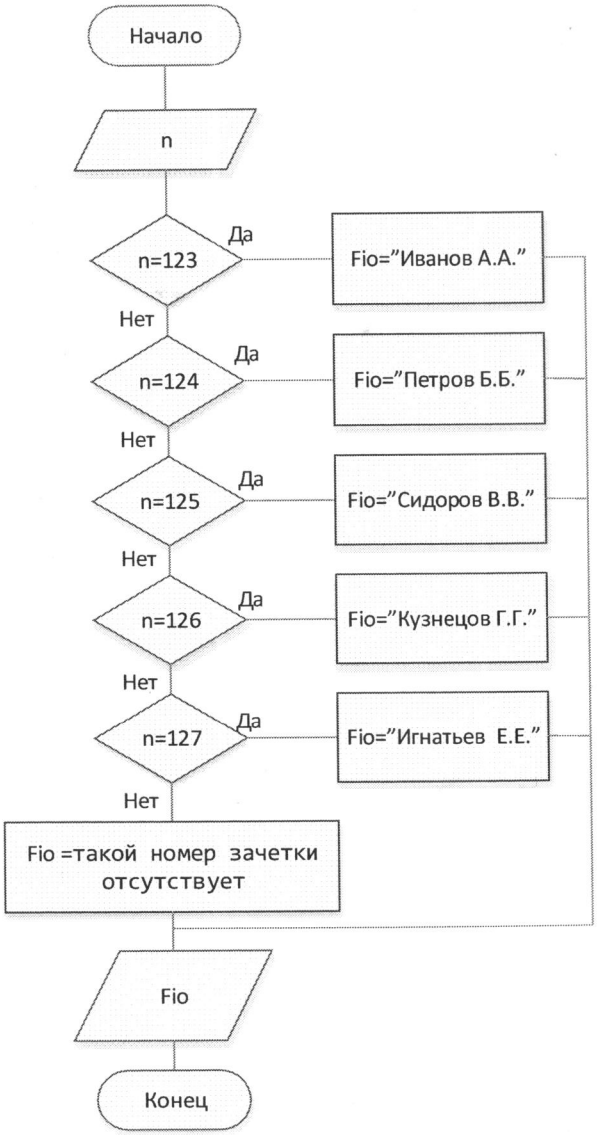
Начало
n
n=123
Да
Fio=”Иванов А.А.”
Нет
n=124
Да
Fio=”Петров Б.Б.”
Нет
n=125
Да
Fio=”Сидоров В.В.”
Нет
n=126
Да
Fio=”Кузнецов Г.Г.”
Нет
n=127
Да
Fio=”Игнатьев Е.Е.”
Нет
Fio =такой номер зачетки отсутствует
Fio
Конец

Рис. 15.

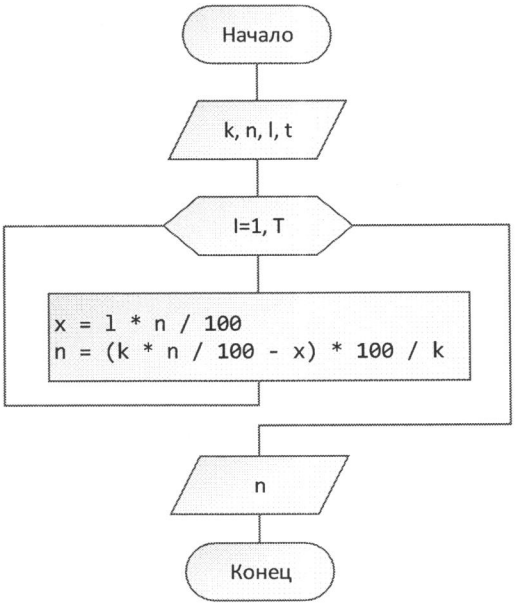
Начало
k, n, l, t
I=1, T
x = l * n / 100
n = (k * n / 100 - x) * 100 / k
n
Конец

Рис. 17.

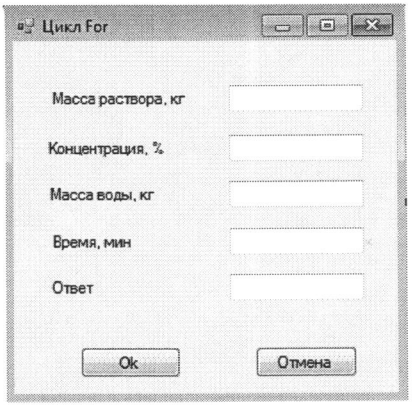
Цикл For
Масса раствора, кг
Концентрация, %
Масса воды, кг
Время, мин
Ответ
Ok
Отмена

Рис. 18.

Для кнопки **Ok** введите следующую процедуру.

```
Dim k, n, l, x As Single
Dim t As Integer
k = TextBox1.Text
n = TextBox2.Text
l = TextBox3.Text
t = TextBox4.Text
For i = 1 To t
    x = l * n / 100
    n = (k * n / 100 - x) * 100 / k
Next
TextBox5.Text = n
```

В данной программе переменная x показывает массу удаляемой щелочи (чистой).

Циклы с неизвестным числом повторений

Циклы с неизвестным числом повторений делятся на циклы с предусловием и пост условием см. рис. 16.

Для цикла с постусловием тело цикла выполняется как минимум один раз, так как сначала производятся вычисления, а затем проверяется условие выхода из цикла, данному типу циклов соответствует фраза «выполнять действия пока условие истинно (ложно)». В случае цикла с предусловием тело цикла может не выполниться ни разу в случае, если сразу соблюдается условие выхода данный тип циклов можно описать фразой: «Пока условие истинно (ложно) выполнять действия».

В зависимости от условия недетерминированные циклы можно разделит на цикл «ПОКА» и «ДО ТЕХ ПОР».

В циклах ПОКА используется условие на повторение цикла, т.е. пока условие выполняется, цикл будет повторятся, как только условие становится неверным, алгоритм выходит из цикла.

В циклах «ДО ТЕХ ПОР» используется условие на выход из цикла, т.е. цикл будет закончен, когда условие будет выполнено.

Для создания циклов ПОКА в Visual Basic используется

оператор **while**. Допускается применение данного оператора в следующих формах:

```
While <Условие на повторение>
<Действия>
End While
```

сокращенная форма цикла с предусловием

```
Do while <Условие на повторение>
<Действия>
Loop
```

Цикл с предусловием

```
Do
<Действия>
Loop While <Условие на повторение>
```

Цикл с постусловием

Для создания циклов ДО ТЕХ ПОР в Visual Basic используется оператор **Until**. Допускается применение данного оператора в следующих формах:

```
Do Until <Условие на выход>
<Действия>
Loop
```

Цикл с предусловием

```
Do
<Действия>
Loop Until <Условие на выход>
```

Цикл с постусловием

Рассмотрим пример задачи на цикл с неизвестным числом повторений

В емкости находится щелочной раствор массой **K** и концентрацией **N%**. Каждую минуту из емкости забирают **L** кг (**L<K**) раствора и заливают столько же воды. Определить через, сколько минут концентрация раствора уменьшится в два раза.

Алгоритм решения этой задачи показан на рис. 19.

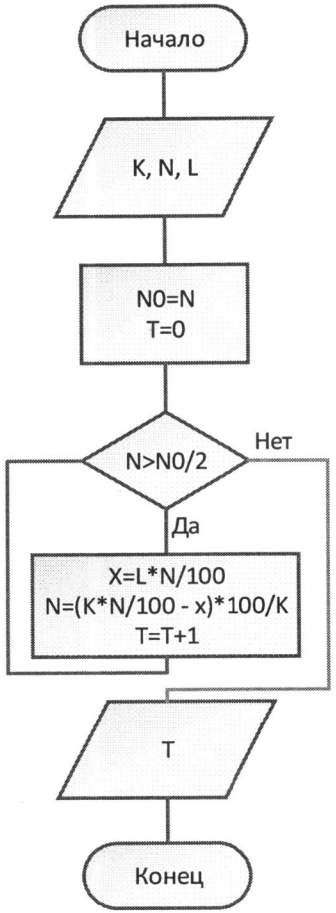

Рис. 19.

где
K – масса раствора;
N – процентное содержание щелочи в растворе;
L – масса выборки;
N0 – начальное процентное содержание щелочи;
X – масса щелочи в удаленной выборке.

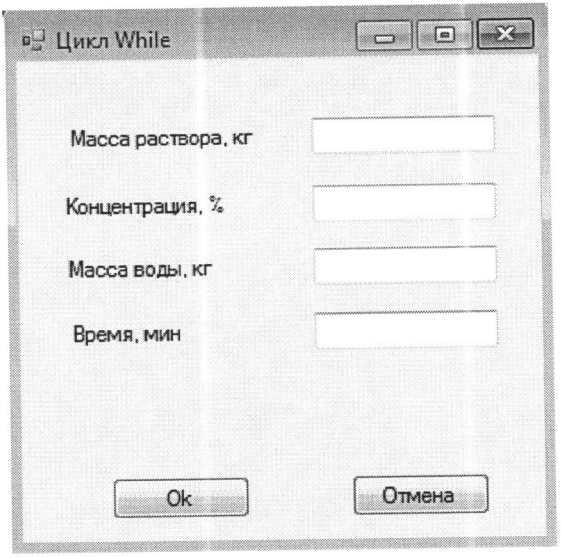

Рис. 20.

```
Private Sub Button1_Click(ByVal sender As Sys-
tem.Object, ByVal e As System.EventArgs) Handles
Button1.Click
        Dim k, n, l, n0, x As Single
        Dim t As Integer
        k = TextBox1.Text
        n = TextBox2.Text
        l = TextBox3.Text
        n0 = n
        t = 0
        Do While 2 * n > n0
            x = l * n / 100
            n = (k * n / 100 - x) * 100 / k
            t = t + 1
        Loop
        TextBox4.Text = t
End Sub
```

Вложенные циклы. В случае если цикл находится в теле другого цикла, то его называют *вложенным*. Вложенный цикл по отношению к циклу, в тело которого он вложен, будет именоваться *внутренним циклом*, и наоборот цикл, в теле которого существует вложенный цикл, будет именоваться *внешним* по отношению к вложенному. Внутри вложенного цикла в свою очередь может быть вложен еще один цикл, образуя следующий *уровень вложенности* и так далее. Количество уровней вложенности, как правило, не ограничивается.

На первом проходе, внешний цикл вызывает внутренний, который исполняется до своего завершения, после чего управление передается в тело внешнего цикла. На втором проходе внешний цикл опять вызывает внутренний. И так до тех пор, пока не завершится внешний цикл.

Примеры таких программ будут показаны в разделе массивы.

Массивы

Массив – набор однотипных элементов, расположенных в памяти непосредственно друг за другом, доступ к которым осуществляется по индексу (индексам).

Размерность массива – количество индексов, необходимых для однозначного доступа к элементу массива.

Нуль-мерный массив называется скаляром, одномерный – вектором, двумерный – матрицей.

a – ноль-мерный массив (скаляр),

$a_1, a_2, a_3 .. a_n$ – одномерный массив (вектор),

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \dots & & & & \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{pmatrix}$$ – двумерный массив (матрица) размерностью $m \times n$.

Встречаются также трехмерные, четырех мерные и т.д. массивы.

Динамическим называется массив, размер которого может меняться во время исполнения программы. Язык программирования, поддерживающий динамические массивы, должен предоставлять возможность для изменения размера массива. Динамические массивы делают работу с данными более гибкой, так как не требуют предварительного определения хранимых объёмов данных, а позволяют регулировать размер массива в соответствии с реальными потребностями. Обычные (не динамические) массивы называют ещё *статическими*.

При работе с массивами как правило применяют циклы, причем, если размерность массива равна двум или более то применяются кратные (вложенные) циклы. Максимальная глубина вложения равна размерности массива. Так для двумерного массива применяется один вложенный цикл.

Для создания статического массива указывается диапазон значений для каждого индекса и тип данных для элемента.

Переменная массива объявляется так же, как любая другая переменная, с помощью инструкции **Dim**. За именем переменой следует пара скобок, в которых указывается размерность массива. В случае вложенных массивов, количество пар скобок может отличаться от одной.

```
Dim a(10, 15) As Integer
```

Данная запись означает, что массив является двумерным и состоит из 10 строк и 15 столбцов. Элементы массива принад-

лежат целочисленному типу Integer.

Пример:

Написать программу, подсчитывающую сумму положительных элементов главной диагонали матрицы $A_{3\times3}$.

$$A=\begin{pmatrix}1 & 2 & -3\\ 3 & -2 & 6\\ 1 & 5 & 2\end{pmatrix}$$

Создайте форму, показанную на рис. 21. Измените свойство **name** текстовых полей, так чтобы в имени объекта присутствовал двойной индекс (строка-столбец). Например, текстовое поле TextBox1 переименуется в TextBox11, поле Textbox2 переименуется в TextBox12 и т.д.

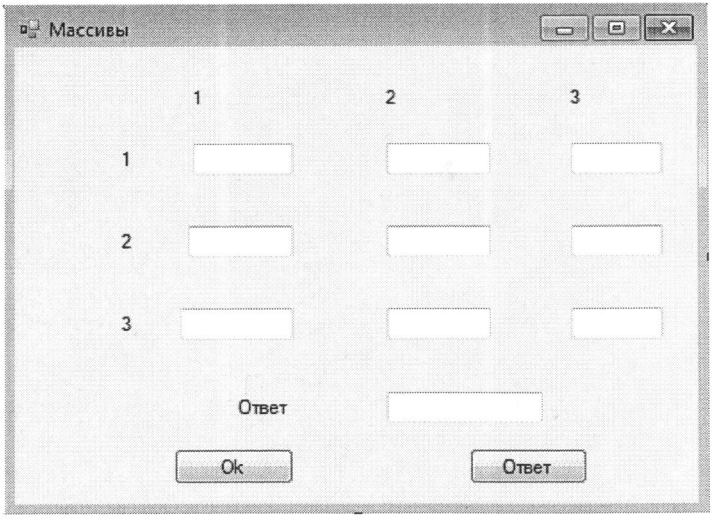

Рис. 21.

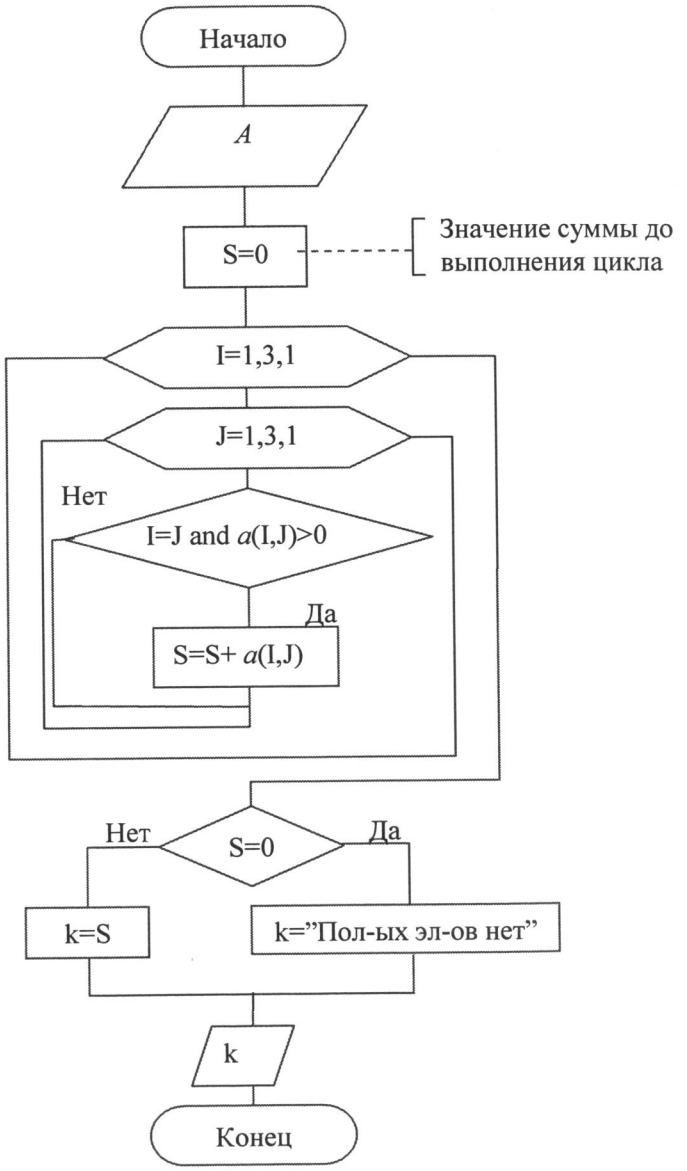
Начало
A
S=0
Значение суммы до выполнения цикла
I=1,3,1
J=1,3,1
Нет
I=J and a(I,J)>0
Да
S=S+ a(I,J)
Нет
S=0
Да
k=S
k="Пол-ых эл-ов нет"
k
Конец

Рис. 22.

Алгоритм решения поставленной задачи показан на рис. 22.

```
Dim a(3, 3) As Integer
Dim i, j As Byte
Dim k As String

a(1, 1) = TextBox11.Text
a(1, 2) = TextBox12.Text
a(1, 3) = TextBox13.Text
a(2, 1) = TextBox21.Text
a(2, 2) = TextBox22.Text
a(2, 3) = TextBox23.Text
a(3, 1) = TextBox31.Text
a(3, 2) = TextBox32.Text
a(3, 3) = TextBox33.Text
s = 0

For i = 1 To 3 Step 1 '- цикл обеспечивающий изме-
нение строк
    For j = 1 To 3 Step 1 '- вложенный цикл обес-
печивающий изменение столбцов
        If i = j And a(i, j) > 0 Then '- условие
отбора элементов главной диагонали
            s = s + a(i, j)
        End If
    Next
Next
    If s = 0 Then
         k = "Диагональ не содержит положительных
элементов"
        Else
         k = Str(s)
    End If
Otvet.Text = k
```

Строковые переменные

Для работы с текстом в языке Visual Basic предусмотрены операторы, обеспечивающие выполнение различных задач.

Список некоторых из них содержится в следующей таблице.

Таблица 14.

Оператор	**Описание**	**Пример**	**Результат**
`Asc (S)` `S- фраза`	Возвращает значение ASCII кода первого символа фразы **S**.	`Asc(Слово)`	209
`Chr(N)`	Возвращает знак, с кодом ASCII **N**.	`Chr(209)`	С
`InStr(S1,S2)`	Возвращает целое число, указывающее начальную позицию первого вхождения строки **S2** в строке **S1**.	`InStr("Слово", "о")`	3
`InStrRev(S1,S2)`	Возвращает целое число, указывающее позицию последнего вхождения строки S2 в строке S1.	`InStrRev("Слово" ,"о")`	5
`LCase(S)`	Преобразует строку S в нижний регистр	`LCase(СЛОВО)`	слово
`Left(S, N)`	Возвращает строку,	`Left("Слово", 3)`	Сло

	содержащую N первых знаков строки S.		
Len(S)	Возвращает количество символов в строке S.	Len(“Слово”)	5
LTrim(S)	Возвращает строку, содержащую копию строки S без пробелов в начале.	LTrim(“ Слово”)	“Слово”
Mid(S, N, K)	Возвращает строку, длинной K символов, вырезанную из строки S начиная с символа N.	Mid(“Слово”,3,2)	ов
Replace(S, S1, S2, N, K, P)	Заменяет в строке S подстроку S1 на подстроку S2. Поиск подстроки S1 начинается с позиции N. По умолчанию поиск начинается с первого символа. K – определяет количество замен. По умолчанию заменяются все найденные подстроки. P – Может принимать значения 0 или 1. По умолчанию – 0. Если 0, то замена производится только в случае полного совпадения. Если 1, то замена производится без учета регистра	Replace(“Слово”, “о”, “а”)	Слава

Right(S, N)	Возвращает строку, состоящую из N последних символов строки S.	Right(“Слово”,2)	во
RTrim(S)	Удаляет пробелы в конце фразы S.	Rtrim(“Слово ”)	“Слово”
Space(N)	Возвращает строку, состоящую из N пробелов.	Space(5)	“ ”
StrComp(S1, S2, [P])	Возвращает -1, 0 или 1 в зависимости от результата сравнения строк S1 и S2. Параметр P определяет способ сравнения, равный 0 (посимвольное сравнение) или 1 (без учета регистра символов). При S1<S2 возвращает -1 При S1=S2 возвращает 0 При S1>S2 возвращает 1	StrComp(“12”,”a”)	-1
StrDup(N, “S”)	Возвращает строку, состоящую из указанного знака S, повторенного N раз.	StrDup(3, "W")	www
StrReverse(S)	Возвращает строку,	StrRevers(“абв”)	вба

	содержащую те же знаки, что и в заданной строке, но в противоположном порядке.		
`Trim(S)`	Возвращает строку, содержащую копию указанной строки без пробелов в начале и конце.	`Trim(“ слово “)`	“слово”
`UCase(S)`	Возвращает строку или знак, содержащий указанную строку, преобразованную в верхний регистр.	`Ucase(“слово”)`	СЛОВО

Рассмотрим пример задачи обработки теста. Создать программу, определяющую фразы палиндромы.

Создайте форму, показанную на рис. 23.

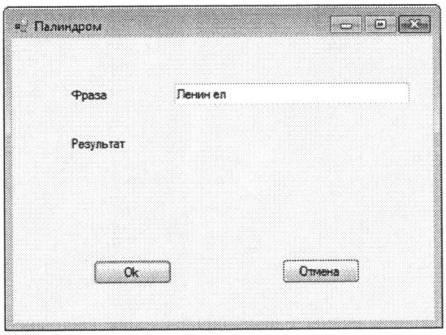

Рис. 23

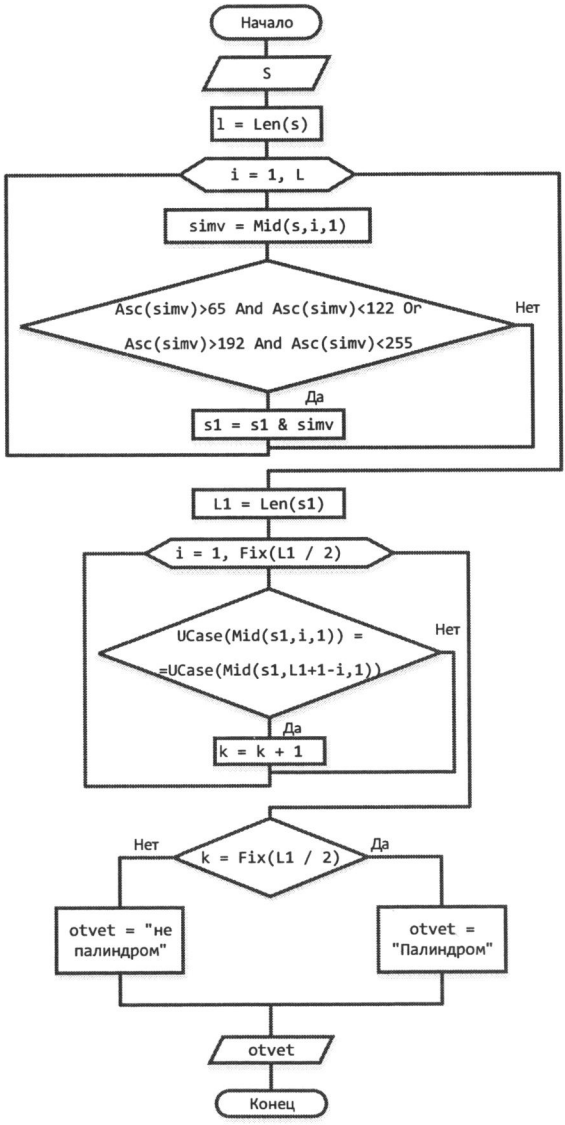
Начало
S
l = Len(s)
i = 1, L
simv = Mid(s,i,1)
Asc(simv)>65 And Asc(simv)<122 Or
Asc(simv)>192 And Asc(simv)<255
Нет
Да
s1 = s1 & simv
L1 = Len(s1)
i = 1, Fix(L1 / 2)
UCase(Mid(s1,i,1)) =
=UCase(Mid(s1,L1+1-i,1))
Нет
Да
k = k + 1
Нет
k = Fix(L1 / 2)
Да
otvet = "не палиндром"
otvet = "Палиндром"
otvet
Конец

Рис. 24.

Алгоритм решения данной задачи показан на рис. 24.

Программный код для кнопки **Ok** приведен ниже.

```
Dim s, s1, simv, otvet As String
Dim L, L1, k As Integer
s = TextBox1.Text
L = Len(s)
For i = 1 To L
    simv = Mid(s, i, 1)
    If Asc(simv)>65 And Asc(simv)<122 Or
Asc(simv)>192 And Asc(simv)<255 Then
        s1 = s1 & simv
    End If
Next
L1 = Len(s1)
For i = 1 To Fix(L1 / 2)
If UCase(Mid(s1,i,1))=UCase(Mid(s1,L1+1-i,1)) Then
        k = k + 1
End If
Next
If k = Fix(L1 / 2) Then
    otvet = "Палиндром"
ELse
    otvet = "не палиндром"
End If
LabeL2.Text = otvet
```

Подпрограммы

Подпрограммами называют именованные блоки программного кода, выполняющие определенное действие. Подпрограммы связаны с основной программой с помощью параметров, передаваемых из основной программы в подпрограмму. Совокупность основной программы и блоков подпрограмм дает решение общей задачи.