Лабораторная работа
ОСНОВЫ ПРОГРАММИРОВАНИЯ НА VISUAL BASIC 6.0

Цель работы. Знакомство с интегрированной средой разработки Visual Basic (VB) для операционной системы (ОС) Windows и приобретение навыков работы по созданию проектов приложения с **использованием переменных, функций и процедур** разных типов.

1. ОСНОВНЫЕ ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ

Программой называется конечная последовательность предписаний, сформулированных с помощью определенных синтаксических конструкций и служащая для выполнения каких-либо действий. Для перевода программы с алгоритмического языка в машинный код используются специальные программы-*трансляторы* (от англ. translator – переводчик): *интерпретаторы* (interpretation – истолкование) и *компиляторы* (compiler – составитель). В этих программах, своих для каждого алгоритмического языка, содержатся способы преобразования синтаксических конструкций в машинный язык. Программа-приложение составленная пользователем, представляет собой данные, подлежащие обработке транслятором.

Интерпретатор обрабатывает программу пользователя поэлементно: преобразует в машинный код и сразу же его исполняет. Таким образом, программа выполняется на ЭВМ только в среде интерпретатора.

С помощью *компилятора* создаются файлы-программы (exe-файлы), которые могут самостоятельно выполняться в среде операционной системы ЭВМ.

К основным алгоритмическим структурам программ относятся*: Присваивание. Следование. Цикл. Разветвление*.

1.1. ПРИСВАИВАНИЕ

Программы оперируют *переменными*. Каждая переменная в процессе действия программы в компьютере сохраняется в определенной ячейке памяти. Резервирование места для переменной в памяти реализуется путем указания имени и *типа* (описание переменной). Переменные создаются программистом по опреде-

ленным правилам (первым символом имени должна быть буква; остальные символы – буквы, цифры и знак "_"; прописные и строчные буквы различаются; нельзя использовать точку). Имя не должно совпадать с ключевым словом VB, т.е. не должно быть идентично уже зарезервированным словам и именам.

Значения переменных – это *данные*, которые обрабатываются компьютером. *Тип данных* определяет способ их хранения. Некоторые типы данных приведены в табл. 1:

Таблица 1. Основные типы данных

Тип	Содержание	Объем	Диапазон значений
`Byte`	Короткое неотрицательное число	1 байт	0...255
`Integer`	Целое число	2 байта	–32768…+32767
`Long`	Длинное целое число	4 байта	–2147483648… +2147483647
`Single`	Десятичное число обычной точности	4 байта	±(1.4011298e–45… 3.402823e+38)
`Double`	Десятичное число двойной точности	8 байт	±(1e–324…1e+308)
`String`	Набор символов (строка)	зависит от числа символов в строке	
`Boolean`	Логическая величина	2 байта	Истина (`True`) или Ложь (`False`)
`Date`	Дата	8 байт	Информация о дате
`Object`	Указатель на объект	4 байта	Значением является ссылка на объект
`Variant`	Произвольное значение	не менее 16 байт	Может быть переменной любого типа

Оператор объявления типа записывается в начале программы или процедуры в разделе объявлений (`Declarations`). Синтаксис записи:

`Dim` *Имя_переменной1* [`As` *тип1*][,]…

После слова `Dim` через запятую можно записывать несколько таких конструкций:

Dim X As Single, ЧислоЭлементов As Integer, Пи As Double, S As String.

После String может стоять знак* и указано число символов в строке (длина строки).

Оператор присваивания:

ИмяПеременной=ЗначениеПеременной

Наряду с переменными применяются встроенные и пользовательские *константы*. Все встроенные константы имеют префикс vb (например: vbWhite – это название константы обозначения цвета, т.е. ячейки памяти, где сохранено число 16777215).

Последовательность данных одного типа, расположенных в памяти последовательно, может образовывать *массив*. Массив обозначается именем с указанием размерности и типа данных:

Dim *ИмяМассива(размерность1, размерность2, ...)* As *тип.*

Пример: Dim arrB(1,9) As Integer.

Обращение к элементу массива осуществляется указанием его имени и индекса. По умолчанию индекс массива начинается с нуля. Для явного указания границ массива применяется служебное слово To:

Dim arrB(1 **To** 2,1 **To** 10) As Integer.

Возможно объявление *статических* и *динамических* массивов. Значения верхней и нижней границ области памяти для *статического* массива не могут быть изменены в программе. Если количество элементов массива неизвестно заранее и будет определяться по ходу выполнения программы, то следует объявить *динамический массив*. Для этого применяются операторы Dim, ReDim и Erase.

Пример:
```
Dim intI As Integer
Dim arrA() As integer
…
IntI=1000
…
ReDim arrA(intI)  /*для arrA выделяется память
                    из 2*1000 байт*/
…
Erase arrA /*память, выделенная для arrA: 0 байт*/
```

1.2. Следование

В программе следование реализуется последовательным выполнением операций в порядке извлечения команд и данных из памяти. Возможны специальные операции перехода к другим областям памяти.

1.3. Разветвление (операторы условного перехода)

Разветвление применяется в том случае, когда в зависимости от условия нужно выполнить одно или другое действие. Разновидности ветвления: обход и множественный выбор.

Два выражения, между которыми помещен знак сравнения, называются ***простыми условиями.*** Последовательность простых условий, соединенных знаками логических операций AND, OR, NOT называется ***сложным условием***.

Варианты условий:

A) Однострочная форма операторов условного перехода:

If *Условное Выражение* **Then** *Оператор1* [**Else** *Оператор2*]

B) Многострочная форма:

If *УсловноеВыражение*
Последовательностьоператоров1
[**Else**
Последовательностьоператоров2]
End If

C) С вложенными операторами условного перехода:

```
If              Then
Else
  If            Then
    ….
  End If
End If
```

D) Оператор **ElseIf** позволяет использовать другую запись подобной конструкции в программе:

```
If x >= -2 And x <= 5 Then
        a = x + 1.5 * x ^ 3 + e ^ (x + 1)
        strA = var1
   ElseIf x > 5 And x <= 7.5 Then
```

```
                a = 4.5 * x + 1.5
                strA = var2
        ElseIf x > 7.5 Then
                a = x + 1
                strA = var3
        Else: a = 0
End If
```

Е) Оператор множественного выбора **Select Case** удобнее применять, если требуется проверка нескольких условий:

Select Case *Переменная*
Case *Значение1*
Последовательность операторов1
...
Case *Значение(N–1)*
Последовательность операторов(N–1)
[**Case Else**
Последовательность операторовN]
End Select

Пример: проверка переменной *x*

```
Select Case x
    Case Is =3
    ‘Выполнить некоторые действия
    Case Is>17
    ‘Выполнить другие действия
    Case Else
    ‘Действия, которые следует выполнить, когда не верно
    ни одно из предыдущих условий
End Select
```

1.3. Циклы

Алгоритмические структуры циклов применяются в случае, если какие-либо операции требуется применять несколько раз, пока не выполнится некоторое условие. При этом многократная последовательность операций называется *телом* цикла, а переменная, подсчитывающая количество выполнений – *счетчиком*

или *индексом* цикла. Циклы могут быть вложенными. Существует несколько вариантов структур циклов:

Цикл со счетчиком

Синтаксис:

For *Имя=Значение1* **To** *Значение2* [**Step** *Значение3*]
ОператорыТелаЦикла
Next [*Имя*]

Имя – имя *счетчика* цикла.

Значение1 – начальное значение счетчика.

Значение2 – конечное значение счетчика.

Значение3 – величина, на которую изменяется значение счетчика за одно повторение.

ОператорыТелаЦикла – часть программы, которая должна повториться.

В цикле со счетчиком сначала осуществляется проверка на непротиворечивость, если есть противоречие, то работа цикла прекращается и у счетчика остается его начальное значение. Если противоречия нет, то повторяющиеся операторы будут выполнены при начальном значении счетчика, после чего его значение будет увеличено на шаг и будет снова проведена проверка на превышение конечного значения.

Циклы с условием

Если число повторений не известно заранее, то организуются циклы с условием. Синтаксис цикла имеет две формы в зависимости от местоположения условий.

Форма 1:	Форма 2:
Do *Условие* *ОператорыТелаЦикла* **Loop**	**Do** *ОператорыТелаЦикла* **Loop** *Условие*

В первой форме может случиться так, что операторы тела цикла не будут выполнены ни разу, а во второй форме операторы тела цикла всегда будут выполнены хотя бы один раз.

Условия также бывают двух типов: с ключевым словом `While` и с ключевым словом **`Until:`**

С ключевым словом **`While`** **(условие продолжения цикла)**	С ключевым словом **`Until`** **(условие завершения цикла)**
операторы тела цикла выполняются, если значение *Условие* есть Истина (`True`), иначе цикл завершается	операторы тела цикла выполняются, если значение *Условие* есть Ложь (`False`), иначе цикл завершается

1.4. Процедуры и функции

В случае многократного выполнения одной и той же последовательности операций целесообразно представить эту последовательность в виде процедуры или функции.

При вызове процедуры или функции в качестве параметров указываются имена тех переменных, значения которых необходимо передать в нее в данный момент и которые обрабатываются операторами вызываемой процедуры или функции.

Процедура – это подпрограмма, которая обычно используется, когда нужно в разных частях программы повторить один и тот же блок операторов. Она начинается оператором *Sub* и заканчивается оператором *End*, между ними помещается код процедуры:

`Sub` *Имя_процедуры([параметры])*
…(операторы)
`End Sub`

Вызов процедуры в программе осуществляется посредством указания имени и параметров:

`Call` *Имя_Процедуры[(параметры)]*

Пример: процедура SUM, суммирующая элементы заданного массива в программе вычисления среднего.

```
Sub SUM (z(1),k,S)'Начало описания процедуры SUM
S=0
    For I=1 to k
        S=S+z(I)
    Next I
End SUM          'Конец описания процедуры SUM
```

```
...
Dim B(3) As Double
B(1)=1 : B(2)=3 : B(3)=5
CALL SUM (B(),3,S) 'Обращение к процедуре SUM
S=S/3
...
End
```

В данной программе использовано обращение в процедуре к фактическому одномерному массиву B, состоящему из k элементов. При записи процедуры указывается размерность массива (z(1)), а при ее вызове – имя массива с пустыми скобками (B()).

Функция в отличие от процедуры возвращает в программу значение, полученное путем выполнения оператора присвоения вида: *Имя_функции=значение*. Функция, как и любая переменная, имеет определенный тип. При этом значение может быть указано внутри функции явно или получено в результате преобразований (вычислений или вызова другой функции).

Function *Имя_функции(параметры)* [As *тип*]
…(операторы)
Имя_функции=значение(выражение)
End Function

Функция вызывается указанием ее имени и параметров в правой части какого-либо оператора или в сложном выражении (т.е. там, где нужно получить ее значение):

x = ***Имя_Функции(параметры)***

Пример: функция, определяющая факториал числа.

```
'Начало описания функции Fact:
Function Fact(bytN As Byte) As Integer
Dim bytI As Byte
Dim intFact As Integer
For bytI=1 To bytN
    intFact=intFact*bytI
Next bytI
Fact=intFact
End Function          'Конец описания функции Fact
...
```

```
Dim Output As Integer
...
'Вызов функции Fact с параметром N:
Output=Fact(bytN))
```

Существует некоторое количество *встроенных* в VB функций (математические функции, функции выдачи сообщений и др.), которые не нужно определять в программе, и достаточно просто вызвать в нужном месте программного кода.

Примеры:

Арифметическое выражение – последовательность чисел, констант, переменных, функций, которые связаны знаками арифметических операций.

IIf (*УсловноеВыражение, Значение1, Значение2*) Эта функция возвращает *Значение1*, если условное выражение истинно и *Значение2*, если оно ложно.

При использовании процедур и функций, а также параметров в функциях или процедурах следует следить за *областью их видимости* (точнее – доступности) из разных разделов программы. Неправильное обращение к области видимости часто является причиной ошибок.

Существует несколько способов, уточняющих *область видимости* объявляемой переменной, процедуры или функции:

Private *Имя_1* [As *тип1*], *Имя_2* [As *тип2*],…

Static *Имя_1* [As *тип1*], *Имя_2* [As *тип2*],…

Public *Имя_1* [As *тип1*], *Имя_2* [As *тип2*],…

В том случае, когда переменная объявляется с ключевым словом Dim, областью ее использования будет только та процедура, в которой она была описана. Такие переменные называются *локальными (закрытыми)*. После выполнения процедуры их значения будут потеряны. В разных процедурах для локальных переменных можно использовать одинаковые имена (например, в счетчиках циклов). Если используется ключевое слово Static, то переменная тоже будет локальной, однако ее последнее значение сохраняется. Если требуется, чтобы переменные, функции и процедуры были доступны в нескольких разделах одного модуля,

следует использовать оператор `Private` и объявление переменных следует осуществлять в разделе `Declarations` текущего модуля. Когда нужно, чтобы переменная была доступна всем модулям, ее следует объявить в этом разделе оператором `Public`. Такие переменные называются *глобальными* (*открытыми*).

Следует также различать два способа передачи параметров в функции и процедуры: *по ссылке* и *по значению*.

При передаче параметров *по ссылке* (в VB по умолчанию) подпрограмма получает доступ к тем областям памяти, в которых расположены значения передаваемых ей параметров. Соответственно, значения этих переменных в ходе выполнения подпрограммы могут быть изменены.

При передаче параметров *по значению* (параметры – значения) у подпрограммы нет доступа к областям памяти, в которых они расположены. В этом случае будут изменяться только копии передаваемых переменных, для хранения которых в подпрограмме резервируются отдельные области памяти, сами же переменные после вызова подпрограммы останутся без изменения.

Пример передачи параметров по ссылке и по значению:

```
Private Sub Form_Load()
    Dim x, y
    x=3 : y=5
Call ByReference(x,y)        'результат x=9,y=25
Call ByValue(x,y)            'результат x=9,y=5
End Sub
...
'передача параметров по ссылке:
Sub ByReference(a,b)
    a=a*a
    b=b*b
End Sub
'передача параметров по значению:
Sub ByValue(a, ByVal b)
    a=a*a
    b=b*b    ‘вычисленное значение b=b*b можно использовать только в этой подпрограмме
    End Sub
```

После вызова процедуры ByValue изменится только значение переменной x, а переменная y останется без изменений.

Примеры определения областей видимости переменных:

А) Закрытые переменные локального типа

```
Sub Func1()As Integer
    Dim i, S As Integer
        For i=1 To 10
            S=S+i
        Next i
    Func1=S
End Function
```

В данном случае после выхода из подпрограммы Func1=55, а переменные i и S теряют свои значения (при входе в подпрограмму их начальные значения нулевые). Имена i и S могут использоваться в других частях программы для обозначения иных величин.

В) Закрытые переменные Static

```
Sub Func2()As Integer
    Dim i As Integer
    Static S As Integer
        For i=1 To 10
            S=S+i
        Next i
    Func2=S
End Function
```

В данном случае после выхода из подпрограммы Func2 переменная S сохраняет свое значение до следующего обращения к ней. После первого вызова значение функции Func2 будет 55, после второго – 110 и т.д., однако в этом случае другие части программы не могут изменить значение S.

С) Открытые переменные Private и Public

```
Private S As Integer
Sub Func3()As Integer
    Dim i As Integer
        For i=1 To 10
            S=S+i
        Next i
    Func3=S
End Function
```

В данном случае после первого выхода из подпрограммы-функции `Func3=55`, а переменная `S=55` и сохраняет свои значения и имеет один и тот смысл во всех частях программы. Использование `S` в других частях программы может изменять ее значение (в частности, повторный вызов этой функции приведет к изменению значения `S=110`). Замена объявления области доступности переменной `S` на `Public` позволит обращаться к ней из других программных модулей, например, из модуля другой экранной формы.

2. Понятие об объектно-ориентированном программировании (ООП)

Наборы процедур и функций целесообразно сохранять в отдельных файлах (модулях), чтобы они могли быть использованы в других программах. Современное обобщение этой тенденции приводит к тому, что основными операндами программы становятся не переменные, а *объекты*.

Объект – некоторая сущность, которая четко проявляет свое поведение и является представителем некоторого множества подобных себе сущностей. Каждый объект характеризуется определенными параметрами, которые условно можно разделить на методы, свойства и события.

Одна из ведущих идей при создании современных программ –формировать программный код таким образом, чтобы дать возможность другому разработчику воспользоваться уже готовыми продуктами (фрагментами программных кодов), которые сами предоставляют в распоряжение программиста набор определенных внутри них собственных свойств, действий, которые они могут выполнять, и внешних событий, на которые они могут реагировать. При этом внутреннее содержание этого продукта, реализующий его программный код, становится безразличным для постороннего пользователя, и нет необходимости вникать в детали его реализации. Для претворения этой идеи в жизнь в ряде языков программирования вводится понятие *класса*.

Класс – это обобщенная модель, служащая для описания объектов программ. При создании класса в нем описываются: параметры объектов, называемые *свойствами*; внешние и программные воздействия, называемые *событиями*, на которые реа-

гируют объекты; подпрограммы (процедуры и функции), называемые *методами,* которые устанавливают (изменяют) значения параметров или выполняются в ответ на события (подпрограммы – обработчики событий). Фактически это именованный набор компонентов, включающий структуры данных и связанные с ними функции их обработки.

Представление свойств и методов как неотъемлемых частей объекта носит название *инкапсуляции*. При этом внутреннее содержание объекта (т.е. как он непосредственно устроен) скрыто для пользователя и не имеет значения для работы с ним. К одному классу принадлежат объекты с одинаковым набором свойств, методов и событий. Класс может включать в себя объекты других классов.

С точки зрения ООП все операнды программы представляют собой *объекты*, каждый из которых принадлежит к какому либо *классу* (является его элементом).

Пример: класс телефонов. Звонок телефона – это событие. Мы реагируем на него методом «реакции на звонок» и поднимаем трубку. Чтобы позвонить, мы используем метод «набрать номер». Свойства определяют внешний вид аппарата и его характеристики.

Другой пример: объект – круг, нарисованный на экране монитора, – характеризуется радиусом, координатами центра, цветом. Параметры объекта (радиус, цвет и т.п.) называются его *свойствами*, а процедуры или функции, которые он выполняет в ответ на какой-либо запрос программы (поменять цвет, переместиться и т.п.) называются *методами*. Одна из характеристик объекта, описывающая внешние воздействия, на которые реагирует объект данного класса во время работы приложения, называется *событием* (например, щелчок мыши внутри круга).

Свойства отвечают за внешний вид и поведение объекта.

Методы – это рабочие операторы объекта, т.е. это некоторые действия, которое может выполнять VB над данным объектом. Эти действия выполняется особой, системной программой VB и в результате его в объекте происходят какие-то изменения. Например, метод *Move* позволяет переместить элемент управления в заданную позицию. Метод, который вызывается на конкретный запрос, определяется классом, экземпляром которого является данный объект.

Основное различие между методами и свойствами заключается в том, что со свойствами можно работать как во время разработки проекта, так и во время выполнения приложения, в то время как методы доступны только при выполнении программы.

<u>События</u> связаны с определенными действиями пользователя и могут вызвать код VB. Примеры событий: щелчок мышью, загрузка новой экранной формы, перемещение указателя мыши и т.п.

Стержень создания Windows-приложений – ориентировка на **события**. Концепция многозадачной ОС предполагает, что ни одно приложение не может работать само по себе, не взаимодействуя с другими приложениями и с самой системой. ОС (в нашем случае – Windows) должна воспринять событие, принять решение о том, как на него следует реагировать, затем обеспечить реакцию на событие, посылая его какому-либо приложению или системной подпрограмме, которые должны это событие обрабатывать. По этому же принципу VB работает «над Windows», перехватывая соответствующие сообщения и обеспечивая реакцию на события в рамках своих функциональных возможностей.

Главный вывод: для выполнения программного кода всегда необходимо ***событие***. Это одно из важнейших правил создания приложений в VB. Ни один код не выполняется без события, т.е. программы управляются событиями. В этом главное отличие от линейного программирования. В программах, управляемых событиями нет сплошного кода, который выполняется от начала до конца. В любой момент пользователь может активизировать другой код на выполнение, создав то или иное событие, на которое программа должна среагировать (например, нажать какую-либо кнопку в приложении, ввести текст в поле, прекратить работу и т.п.). Таким образом, после запуска программы у пользователя нет четко определенного пути, а есть ряд вариантов, определяемых многообразием предоставляемых в его распоряжение ОС или VB рядом различных событий. При наступлении одного из них начинается выполнение соответствующего этому событию программного кода.

Пример: программирование рисования кругов. Класс, к которому принадлежат все объекты-круги, называется `Round`. *Свойства* класса `Round`: **R** – радиус круга; **X, Y** – координаты

центра; **Color** – цвет круга. *Методы* класса `Round`: `Draw` – рисует круг с заданными параметрами; `Move` – перемещает круг на определенное расстояние в выбранном направлении; `ChangeColor` – изменяет цвет круга.

В этом случае для рисования некоторого количества кругов потребуется соответствующее количество объектов: `Cyrcl1, Cyrcl2,` … Все они принадлежат к классу `Round,` имеют одинаковые свойства и вызывают одинаковые методы в ответ на одни и те же запросы. Запись программы становится намного компактнее, и значительно облегчаются процессы ее редактирования и тестирования. Программа реализует и описывает элементы, которые могут использоваться в других приложениях.

Для дополнения свойств и методов объектов одного класса в ООП предусмотрен механизм *наследования. Наследованием* называется возможность доступа объектов, принадлежащих классу-потомку к методам и свойствам класса-предка. Оно служит средством детализации свойств объектов и реализуется путем создания *иерархической структуры* классов. Вначале создается класс, располагающийся в основании иерархической структуры, затем создается новый класс, для которого описывается некоторый новый метод или свойство, полагая, что все остальные свойства и методы остаются без изменения. В результате новый класс (потомок), наследует все свойства и методы родительского класса (предка). В нашем примере может быть определен класс `Round`. Объект этого класса может быть включен в класс `Figure`, в котором могут быть и другие объекты, например `Quadrate`.

Термин *полиморфизм* (греч. poly – много и morphos – форма) в программировании относится к таким переменным или параметрам, которые в процессе выполнения программы могут принимать значения разных типов. Полиморфизм в ООП означает ситуацию, когда методы с одни и тем же именем отвечают за одно действие, но реализуют его по-разному, в зависимости от того, к объекту какого класса следует применить данное действие.

3. Создание проекта в VB

Проектирование – в визуальной среде предполагает не только разработку программного кода, но и создание экранной формы

– *графического интерфейса пользователя* (Graphical User Interface – или GUI).

Интерфейс (в узком смысле) – совокупность средств, обеспечивающих физическое или логическое взаимодействие устройств и программ вычислительной системы. Центральное понятие интерфейса – *экранная форма. Экранная форма* это – графическое представление окна приложения Windows, включающее: 1)содержание этого окна, т.е. совокупность свойств самого окна с их значениями; 2) совокупность объектов, находящихся в окне; 3) совокупность свойств этих объектов с их значениями.

Приложения VB строятся по *модульному принципу*. Все компоненты или модули хранятся в памяти отдельно и независимо друг от друга, поэтому отдельные компоненты можно включать в разные проекты. Информация о связях между компонентами хранится в файле проекта, имеющем расширение *.VBP.

Основные компоненты проекта: файлы форм (*.FRM); файлы каждого модуля (*.BAS).

Среда разработки является *интегрированной (Integrated Development Environment – IDE)*, т.е. в ней можно выполнять все необходимые действия для создания программного продукта: проектирование и описание составных частей приложения, редактирование программного кода, отладка, компиляция все элементов приложения в выполняемый файл.

4. Задание для выполнения лабораторной работы:

4.1. Ознакомиться с основными элементами среды разработки проектов Visual Basic и изучить их свойства и применение для разработки проектов.

4.2. На примере выполнения конкретного задания познакомиться с основными приемами программирования в VB. Обратить внимание на применение в программе переменных, функций и процедур разных типов.

4.3. В соответствии с индивидуальным вариантом задания выполнить самостоятельно разработку, отладку и тестирование проекта.

5. Выполнение работы:

5.1. Ознакомление со средой проектирования VB

Открыть любым стандартным способом приложение VB (режим *Standard.exe\New*) и изучить вид *Главной панели* IDE проекта, ее

основные элементы и их назначение.

Основными элементами окна IDE являются: *Главное меню*, панели инструментов *Standard (Стандартная), Debug (Отладка), Edit (Редактирование)* и *Form Editor (Редактор форм)*, а также окна: *Экранной формы*, *Проводника проекта*, *Свойств объекта*, *Редактирования программного кода*, *Расположения форм* и *Просмотра характеристик классов*.

Главное меню с 13 элементами, выполняющее функции доступа ко всем опциям среды разработки. Для удобства работы некоторые пункты этого меню дублируются вызовом *Контекстного меню* (правая кнопка мыши при выделенном объекте).

Основные панели инструментов (*ToolBars*) IDE вызываются при помощи пункта главного меню (*View\ToolBars*).

Наиболее часто используемой является панель *Standard (Стандартная)*. Кроме пиктограмм основных элементов IDE на этой панели присутствует отображение текущих параметров выделенного объекта. Когда осуществляется редактирование формы, используется *графический режим*. В этом случае первая пара чисел в правой части панели *Standard* представляет координаты для левого верхнего угла выбранного объекта относительно левой верхней точки формы, а вторая пара чисел отображает его длину и ширину. По умолчанию в качестве координат в Visual Basic используются так называемые *твипы* (567 твипов = 1 сантиметр). При редактировании кода активизируется *Текстовый режим*, и тогда числа внизу показывают координаты курсора.

Кроме панелей инструментов в среде IDE возможно открытие следующих окон (пункт *View* Главного меню):

– ***Окно экранной формы*** (*…\Object*) служит для добавления или удаления из форм различных элементов управления.

– ***Окно проводника проекта*** (*…\Project Explore*) используется в случае необходимости работы с несколькими проектами одновременно. Содержимое этого окна отображает иерархические структуры, в которых вершинами всех иерархий являются открытые в данный момент проекты, а исходящие от них ветви – составными частями проектов (например, формы).

– ***Окно свойств объекта*** (*…\Properties Windows*) используется для изменения свойств выделенного объекта, в частности, элемента управления и состоит из двух списков. Верхний раскры-

вающийся список содержит имена всех объектов, помещенных программистом на экранную форму. Нижний список содержит перечень всех свойств указанного вверху объекта и состоит из двух столбцов, в которых указано наименование свойства (по категориям или по алфавиту) и его значение.

Пример: Объект *Текстовое поле (TextBox)* содержит более 50 различных свойств. Среди них в категории *Appearance* содержатся свойства: *ЦветФона (BackColor); ТипОбрамления (BorderStile); ЦветОбъектовОкна (ForeColor)*. В категории *Font* содержится только одно свойство *Font*.

– ***Окно редактирования программного кода*** (…*Code*) предназначено для создания или редактирования кода программы. Если необходима работа с несколькими формами, то имеется возможность открыть для каждого из них свое окно кода. Раскрытие окна осуществляется также при помощи двойного щелчка на объекте (форме или элементе управления), для которого предполагается изменить или создать программный код.

– ***Окно расположения формы*** (…*Form Layout*) служит, чтобы задать для формы место на экране, где она должна находиться при выполнении приложения. Для этого необходимо в окне размещения форм при помощи мыши переместить ее изображение в заданное место.

– ***Окно просмотра характеристик классов*** (...*Object Browser*) служит для получения информации о выделенном объекте приложения. В каталоге, открывающимся при помощи соответствующей кнопки на панели инструментов или из меню *View\Object Browser*, объекты сгруппированы по категориям, которые принято называть *библиотеками* объектов. В списке *Class* перечислены все объекты VB. После выбора объекта в списке *Members of <Имя класса>* выводятся все относящиеся к данному объекту свойства и методы. При написании кода целесообразно копировать название свойства или метода в буфер обмена и вставлять его в окно кода. Это выполняется при помощи клавиш Ctrl-C (поместить в буфер) и Ctrl-V (вставить из буфера).

– ***Панель элементов управления*** (…*General*). С помощью элементов управления создаются объекты управления на экранной форме, служащей интерфейсом между пользователем и вычисли-

тельной средой VB. Нужный образец (шаблон) элемента управления следует разместить на экранной форме, например, выделив его и указав его размеры при помощи мыши. Для облегчения размещения элементов экранная форма покрыта сеткой. Расстояние между соседними точками по умолчанию установлено равным 120 твипам.

5.2. Пример разработки программы

Задание: Разработать интерфейс и программу, позволяющую вычислять объем полого цилиндра и одновременно среднее значение толщины стенок цилиндра при разных исходных данных в одном сеансе работы. В проекте использовать две пользовательские формы: в первой разместить элементы управления для ввода исходных данных, запуска вычисления и окончания работы, а во второй – для вывода результатов. Вычисление объема оформить как подпрограмму-функцию, а вычисление среднего значения параметра – как процедуру. В программе использовать разные виды назначения области видимости переменных и передачу параметров по ссылке и по значению.

Поэтапное выполнение проекта

5.2.1. *Постановка задачи* – составление по возможности точного и понятного словесного описания того, как должно работать будущее приложение, и что должен делать пользователь в процессе работы. Должен быть составлен эскиз экранной формы и решено, в каком виде представляется информация.

Анализируя условие задачи, решаем, что на первой форме целесообразно разместить 5 меток с поясняющими текстами, 3 текстовых поля для ввода исходных данных (внешний и внутренний радиусы, высота), 1 рисунок полого цилиндра и две командные кнопки для запуска вычисления и окончания работы. На второй форме будут расположены три текстовых поля для вывода результатов (объем цилиндра, число обращений к программе расчета и среднее значение толщины стенок) и кнопка возврата к вводу исходных данных для повторных расчетов.

5.2.2. *Разработка интерфейса* – создание экранной формы (окна приложения) и задание свойств объектов.

5.2.3. Задаем название проекта (в окне *Project: (Name)=Cylinder* и его экранных форм (на панели *Properties: (Name)=F1_Cylinder* и заголовок *Caption=Form_Input*)). Добавля-

ем вторую форму (*Главное меню\Project\Add\Form\Открыть*) и также присваиваем ей название (*F2_Cylinder*) и заголовок (*Form_Output*). Сохраняем формы и проект *File\Save Form As…* и *File\Save Project As…*, дав им соответствующие имена.

5.2.4. Подготавливаем рисунок. При помощи одной из программ (Word, Paint или др.) выполняем рисунок полого цилиндра, который затем будет размещен на пользовательской форме.

5.2.5. Размещаем элементы управления на формах:

Метки (объекты Label): Двойным щелчком на соответствующем элементе управления или выделив элемент и указав его место и размеры на форме, создаем 5 меток с надписями. О выделении объекта можно судить по появлению на его границах стандартных для Windows манипуляторов размера. Задаем имена меток (*(Name)=…*) и необходимые по нашему мнению надписи на них (*Caption=…*). Для более выразительного изображения на форме изменяем свойства *Font* и *ForeColor*.

Текстовые поля (объекты TextBox): Создаем аналогично 3 текстовых поля, задав их имена (*TxtR1, TxtR2, TxtH*) и начальные значения (*Text=1, 0* и *1* соответственно).

Командные кнопки (объекты CommandButon): Аналогично размещаем на форме кнопки, присвоив им имена и задав соответствующие надписи на них.

Рисунок (объект PictureBox или Image): размещаем на форме соответствующий объект и при выделенном объекте через буфер обмена Windows вставляем в форму заготовленный заранее рисунок.

По мере необходимости изменяем размеры формы *Form_Input*, а также размеры и положение объектов на ней. Аналогично размещаем необходимые объекты управления на форме *Form_Output*. После этого при помощи окна *Form Layout* размещаем пользовательские формы на экране, в том виде как они должны выглядеть в работающей программе. Примерный вид интерфейса программы представлен на рис.1.

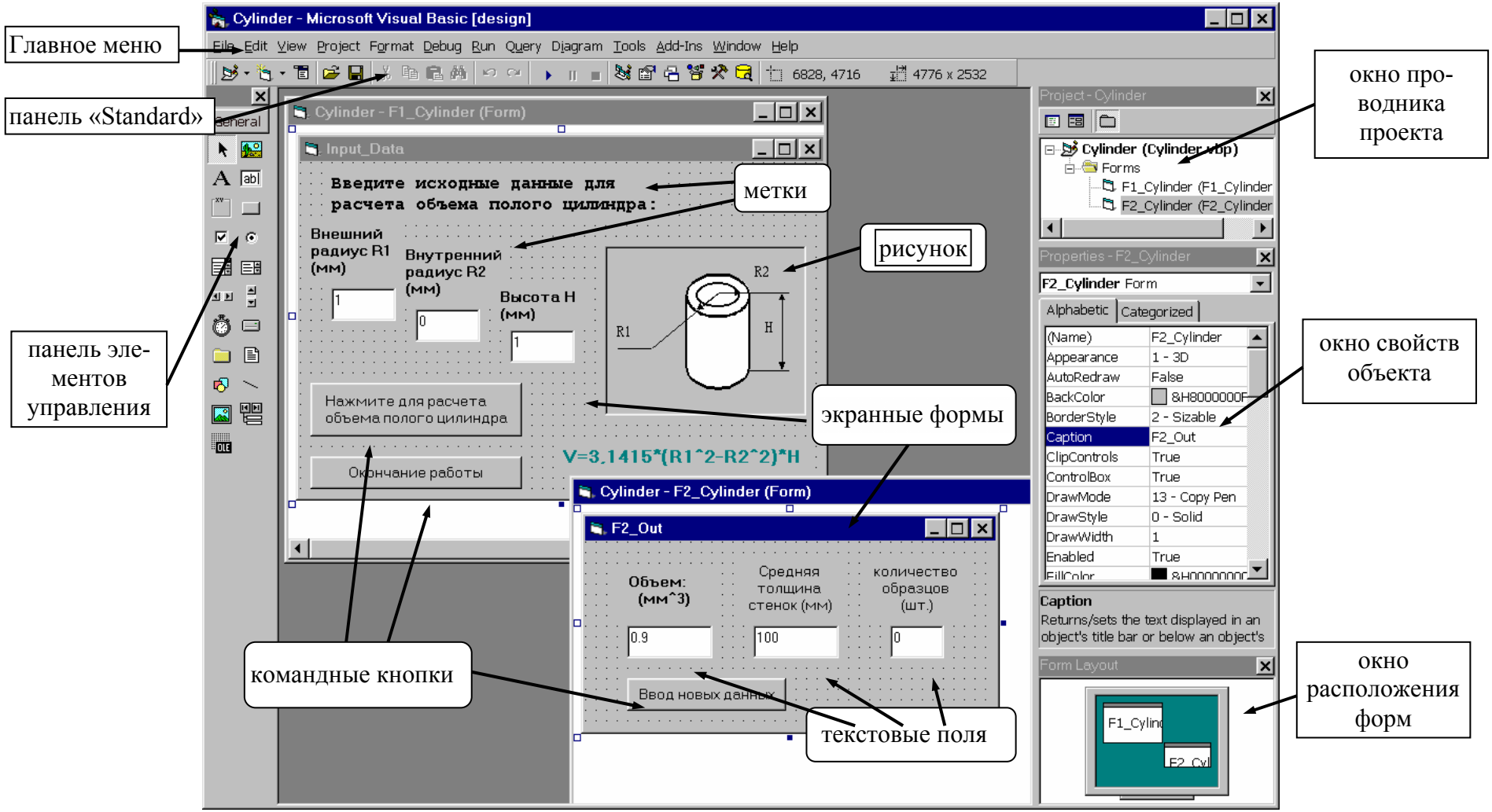

Рис. 1. Пример IDE с двумя экранными формами при выполнении лабораторной работы

Для дальнейшей работы над формами проекта следует воспользоваться уже освоенными, а также следующими приемами.

Выбор объекта, расположенного на форме, для редактирования его свойств можно выполнить из раскрывающегося списка в верхней части окна свойств формы.

Чтобы редактировать вид и расположение объектов на форме кроме мыши, можно пользоваться клавиатурой (клавиши *Ctrl* и *Shift* и стрелки), а также панелью *Form Editor*.

Установить значение свойства элемента управления (окно *Properties*) можно произвольно или из имеющегося набора при помощи двойного щелчка мыши на соответствующей строке и последовательного перелистывания все возможных значений или при помощи выбора пункта из раскрывающегося списка.

5.2.6. Для облегчения последующего редактирования и отчета по лабораторной работе и создаем описание элементов интерфейса и их измененных свойств (см. табл. 2).

Таблица 2. Измененные свойства элементов интерфейса

Элемент	Свойство	Значение
Форма 1	(Name)	***F1_Cylinder***
	Caption	***Input_Data***
	StartUpPosition	***0 – Manual***
Метка для заголовка	(Name)	*Label1*
	Caption	***Ведите исходные данные для расчета объема полого цилиндра:***
	Font	**MS San Serif, 10, жирный**
	ForeColor	***Desktop***
Метка для формулы	(Name)	***Label2***
	Caption	***S=3,1415*(R1^2-R2^2)*H***
	Font	***Times New, 16***
	ForeColor	***Desktop***
И т.д.		
…		

5.2.7. Собственно *программирование* – определение того, какие *события* будут происходить в процессе работы приложения, составление алгоритмов и процедур для этих событий и написание программы (кодов для этих процедур).

Для каждого объекта существует набор стандартных событий, которые могут возникнуть при работе приложения, и для любого события может быть написана процедура, которая его обрабатывает. В данном случае интересующее нас событие – это нажатие на кнопку мышью (событие *Click*). При написании кода обработки событий имена создаваемых процедур формируются из имени объекта (*Command1*), символа «_» и названия события (*Click*). Остальные события, относящиеся к объектам можно выбрать из списка в правой части окна *Code*. Режим окна для вывода процедур, относящихся к форме, задается функцией *Tool\Options* (флажок *Default to Full Module View*).

Для ввода кода используется встроенный в VB редактор, который позволяет оперативно проверять синтаксис языка. Выражение с ошибкой выделяется красным цветом. В среде разработки VB существует также возможность автоматизации редактирования кода, когда для текущей команды или слова отображается контекстный список всех возможных вариантов ее написания. Доступ к настройке автоматизации осуществляется меню *Tool\Options\Auto List Member* (нажатие клавиш **Ctrl-J**). После выбора необходимого элемента из списка его можно добавить в программу нажатием клавиши **Tab** или **Enter**.

Имеется также возможность автоматической подсказки синтаксиса вводимого оператора или функции, которая настраивается при помощи пункта меню *Tool\Options\Auto Quick Info* (клавиши **Ctrl-L**).

В нашем случае вычисления в программе должны начинаться после нажатия командной кнопки *Command1*. Программирование начинаем с задания программного кода обработки события метода нажатия на эту кнопку. Для вызова окна редактирования программного кода следует выполнить двойной щелчок на соответствующем элементе формы. В данном случае – на командной кнопке с надписью «Нажмите для расчета…» (другие способы:

через меню *View\Code* или через контекстное меню в проводнике проектов при выделенной форме).

Программный код для приведенного примера:

```
Option Explicit
Public a As Double
Public l As Integer
Const Pi As Double = 3.141593
'-------------------------------
Public Function Volume(R1, R2, h As
Double) As Double
Dim V As Double
V = Pi * (R1 ^ 2 - R2 ^ 2) * h
Volume = V
End Function
'-------------------------------
Public Sub Sum(ByVal Rr1 As Double, ByVal
Rr2 As Double)
Static k, summa As Integer
Rr1 = Rr1 - Rr2
k = k + 1
F2_Cylinder.txtOutN = Str(k)
summa = Rr1 + summa
F2_Cylinder.txtOutMed = Str(summa / k)
End Sub
'-------------------------------
Private Sub CommButSt_Click()
Dim Rad1, Rad2, Height As Double
Rad1 = Val(txtR1)
Rad2 = Val(txtR2)
Height = Val(txtH)
a = Volume(Rad1, Rad2, Height)
F2_Cylinder.Show
Call Sum(Rad1, Rad2)
F2_Cylinder.txtOutVal = Str(a)
F1_Cylinder.CommButSt.Visible = False
End Sub
'-------------------------------
Private Sub CommButOut_Click()
```

```
Unload F1_Cylinder
Unload F2_Cylinder
End
End Sub
'---------------------------------
Private Sub Command1_Click()
A = Val(TДлина.Text)
B = Val(TШирина.Text)
H = Val(TВысота.Text)
s = 2 * (A + B) * H
TПлощадь.Text = Str(s)
End Sub
'---------------------------------
Private Sub CommButRet_Click()
F1_Cylinder.Show
F1_Cylinder.CommButSt.Visible = True
End Sub
```

После создания программного кода перед запуском программы следует еще раз сохранить в своей папке форму и проект (пункт *File\Save…*, или кнопка **F2**).

5.2.8. *Отладка программы*. После написания программного кода и его проверки можно запустить программу на выполнение. Для этого следует нажать кнопку *Start* на стандартной панели инструментов (или выбрать пункт меню *Run\Start* или нажать клавишу **F5**).

Синтаксически правильно написанная программа может содержать логические ошибки, приводящие к ее неправильной работе. Наиболее часто встречаются следующие ошибки.

Аварийная остановка, например, после попытки деления на нуль. Сообщение об ошибке во время выполнения (*Run-time-error*). В ответ на это сообщение следует выбрать один из трех предложенных в окне вариантов действий *End (Конец)-Debug (Отладчик)-Help (Справка)*.

Зацикливание. В любом случае выполнение программы можно остановить, нажав клавиши **Ctrl+Break**. Это следует сделать, если программа работает подозрительно долго, то есть, возможно, выполняет бесконечный цикл.

Неправильные значения после выполнения (логическая ошибка). Эту ошибку труднее всего обнаружить, т.к. требуется тестирование всех возможных вариантов выполнения. В данном случае достаточно выполнить простейшее тестирование, задав несколько раз значения исходных параметров и сравнив выходные данные программы со значениями, полученными в результате заранее выполненного «ручного» расчета.

Для локализации ошибочного фрагмента кода иногда удобно использовать отладчик (*Debug*). Вызов отладчика – щелчок правой кнопкой мыши по линейке инструментов *Главной панели* и выбор опции *Debug*. Появится панель инструментов или меню отладчика, назначение пунктов которого следующее:

Toggle Breakpoint (установка точки останова или прерывания). Для задания точки останова следует в соответствующей строке кода щелкнуть кнопкой *Toggle Breakpoint* (нажать клавишу **F9**). Слева от выделенной строки появится темно-красный круг, изображающий контрольную точку. Выполнение программы останавливается в этой точке, а соответствующая строка выделяется желтым цветом. Отмена задания точек прерывания – *Debug\Clear All Breakpoints* (клавиши **Ctrl-Shift-F9**).

Quick Watch (просмотр текущих значений переменных в точках остановки). Для поиска ошибок полезно контролировать значения переменных при остановке в контрольных точках. Для этого следует установить окно слежения *Quick Watch*, в которое будут помещаться значения выбранных переменных в момент остановки. Окно настраивается из опции *Debug* командой *Add Watch*. В открывшемся окне следует указать имена переменных и установить необходимые флажки.

После локализации фрагмента программного кода, содержащего ошибку, иногда полезно воспользоваться методом пошагового выполнения программы. Настройка этой операции производится из меню *Debug\Step…* (клавиши **F8**, **Ctrl-Shift-F8**).

Просмотреть значения всех переменных, доступных в данной процедуре можно, воспользовавшись окном *Locals* (*View\Locals Wondows* или соответствующая кнопка на панели *Debug*). Для тестирования отдельных программных строк применяется окно

Immediate. Вызов этого окна – из пункта *View Главного меню*. Оно используется для проверки работы того или иного оператора непосредственно после остановки программы в точке останова, например, если в процедуре объявлена переменная Z, то ввод в этом окне кода **`Print (SIN(Z))`** выведет на экран значение синуса переменной **`Z`**. В окне *Immediate* можно также обращаться к методам объектов и выполнять любые действия, которые можно записать в одной строке.

В случае большого количества процедур, к которым происходит обращение в данной точке программы полезно воспользоваться окном *Call Stack (Debug\...)*, в котором по порядку вложения выводится список всех процедур, активных в данный момент.

5.3. *Сохранение проекта* и при необходимости – *компиляция*, т.е. превращение проекта в *исполняемое приложение*, способное работать самостоятельно вне среды программирования (сохранение файла с расширением *EXE,* пункт *File\Make* ...). После этого можно закрыть VB и проверить, что созданный файл работает правильно.

5.4. Результаты тестирования программы

Исходные данные: R1= …, R2=…, H= …

Результаты работы программы:

После ввода данных в соответствующие текстовые поля и нажатия на кнопку «`CommButSt_Click`» исчезает изображение кнопки, активизируется форма «*F2_Cylinder*» и выводятся значения:

Объем=

Количество обращений=

Среднее значение толщины стенок=

После нажатия на кнопку «`CommButRet_Click`» активизируется форма «*F1_Cylinder*» и возникает изображение кнопки «`CommButSt_Click`»

И т.д.

…

После нажатия на кнопку «`CommButOut_Click`» программа заканчивает работу и возвращается в среду разработки VB.

Задание по лабораторной работе

Разработать интерфейс и программу, позволяющую многократно вычислять некоторую физическую величину по заданной формуле (например: объем, центральный момент и т.п.) и одновременно вычислять среднее значение одного из параметров формулы (толщины стенок, высоты и т.п.). В проекте использовать две пользовательские формы. В первой разместить элементы управления для ввода исходных данных, запуска вычисления и окончания работы, а во второй – результаты расчета. Вычисления по формуле оформить как подпрограмму-функцию, а вычисление среднего значения параметра – как процедуру. В программе использовать разные виды назначения области видимости переменных и передачу параметров по ссылке и по значению.

Варианты заданий по лабораторной работе приведены в табл. 3.

Таблица 3. Варианты заданий по лабораторной работе

№	Наименование	Формула для расчета	Параметр
Объем геометрического тела			
1.	Тор	$V = 2\pi^2 R r^2$	r
2.	Бочка (h – высота, d и D– диаметры основания и средней части соответственно)	$V = \frac{\pi h}{15}\left(2D^2 + Dd + \frac{3}{4}d^2\right)$	h
3.	Усеченный круглый цилиндр	$V = \pi R^2 \frac{h_1 + h_2}{2}$	R
4.	Усеченный прямой конус	$V = \frac{\pi h}{3}(R^2 + r^2 + Rr)$	h
5.	Шаровой сектор	$V = \frac{2\pi R^2 h}{3}$	R
6.	Шаровой сегмент	$V = \frac{1}{3}\pi h^2(3R - h)$	h
7.	Шаровой слой (h – высота, 2a и 2b – диаметры окружностей в основаниях)	$V = \frac{1}{6}\pi h(3a^2 + 3b^2 + h^2)$	h

№	Наименование	Формула для расчета	Параметр
Центральный момент инерции тела, приведенный к единичной массе (при вычислениях задать $m=1$)			
8.	Прямоугольный параллелепипед	$J_z = \frac{m}{12}(a^2 + b^2 + c^2)$	c
9.	Полый прямой круглый цилиндр (Oz – ось цилиндра)	$J_z = \frac{m}{2}(R^2 + r^2)$	R–r
10.	Полый прямой круглый цилиндр (Oz – ось цилиндра)	$J_x = \frac{m}{12}(3R^2 + 3r^2 + H^2)$	R–r
11.	Полый шар	$J = \frac{2}{5} m \frac{R^5 - r^5}{R^3 - r^3}$	R–r
12.	Шаровой сектор	$J_z = \frac{mh}{5}(3R - h)$	h
13.	Шаровой сегмент	$J_z = \frac{mh}{20}\left(\frac{20R^2 - 15Rh + 3h^2}{3R - h}\right)$	h
14.	Усеченный прямой конус	$J_z = \frac{3}{10} m \frac{R^5 - r^5}{R^3 - r^3}$	h
15.	Тор	$J_z = m\left(R^2 + \frac{3}{4}r^2\right)$	r
16.	Тор	$J_x = J_y = \frac{m}{8}\left(4R^2 + 5r^2\right)$	r
Формулы для расчета физических величин			
17.	Импеданс последовательной R-L-C цепи	$Z_= = \sqrt{R^2 + \left(\omega L - \frac{1}{\omega C}\right)^2}$	R
18.	Импеданс параллельной R-L-C цепи	$Z_{//} = \frac{1}{\sqrt{\frac{1}{R^2} + \left(\frac{1}{\omega L} - \omega C\right)^2}}$	R
19.	Скорость тела при реактивном движении	$v = v_0 - v_{\text{газ}} \ln\left(\frac{m_0}{m}\right)$	m

№	Наименование	Формула для расчета	Параметр
20.	Емкость плоского конденсатора *(ε_0=8,85·10^{-12}Ф/м)*	$C = \frac{\varepsilon\varepsilon_0 S}{d}$	d
21.	Сила отталкивания двух электронов (e=1.6·10^{-29} *Кл*)	$F = \frac{1}{4\pi\varepsilon_0} \cdot \frac{e^2}{r^2}$	r
22.	Сила притяжения двух электронов (G=6.672·10^{-11} *Нм^2/кг^2* m_e=9.1·10^{-29} *кг*)	$F = G \cdot \frac{m_e^2}{r^2}$	r

6. *Оформление отчета о лабораторной работе*

Выполненной работа считается только в том случае, если она оформлена в виде текстового документа (предпочтительно в редакторе Word) или в отдельной тетради.

Отчет о выполненной работе должен содержать следующие пункты:

6.1. Формулировку задания с указанием номера варианта и входных данных.

6.2. Графическое изображение алгоритма решения задачи.

6.3. Описание экранных форм и элементов управления на них, примененных в программе (см. табл. 2) При этом обязательно указание свойств и методов объектов, изменяемых программистом при проектировании формы и в ходе работы программы.

6.4. Полный программный код с комментариями, выполняющий поставленную задачу.

6.5. Результаты тестирования программы или результаты работы программы.

Библиографический список

1. Глушаков С.В., Мельников И.В., Сурядный А.С. Программирование в среде Windows: Учебный курс / Харьков: Фолио; Ростов-н/Д: Феникс; Киев: Абрис, 2000. – 487 с.

2. Райтингер М., Муч Г. Visual Basic 6.0: пер. с нем. – К.: Издательская группа BHV, 2000. – 288 с.

3. Волченков Н.Г. Программирование на Visual Basic 6: В 3-х ч. М.: ИНФРА-М, 2000.