МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Транспортные и технологические машины»

ПАКЕТЫ ПРИКЛАДНЫХ МАТЕМАТИЧЕСКИХ ПРОГРАММ

Методические рекомендации к лабораторным работам для студентов направления подготовки 23.03.02 «Наземные транспортно-технологические комплексы» очной формы обучения

Могилев 2022

УДК 004.4
ББК 32.973
П13

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Транспортные и технологические машины» «29» марта 2022 г., протокол № 8

Составитель канд. техн. наук, доц. В. В. Береснев

Рецензент канд. техн. наук, доц. В. В. Кутузов

Методические рекомендации к лабораторным работам для студентов направления подготовки 23.03.02 «Наземные транспортно-технологические комплексы» очной формы обучения.

Учебно-методическое издание

ПАКЕТЫ ПРИКЛАДНЫХ МАТЕМАТИЧЕСКИХ ПРОГРАММ

Ответственный за выпуск	И. В. Лесковец
Корректор	А. А. Подошевко
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс. Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 36 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Программа курса «Пакеты прикладных математических программ» для студентов направления подготовки 23.03.02 «Наземные транспортно-технологические комплексы» предусматривает выполнение студентами лабораторных работ, целью которых является получение практических навыков программирования в среде Visual Studio.

Задание выдается преподавателем. Отчет представляется в виде файла с выполненным заданием.

1 Среда программирования Visual Studio. Создание простого приложения

Цель работы: ознакомиться с визуальной средой программирования VB.NET. Освоить основные приемы работы в VB.NET.

1.1 Основные сведения

VB.NET – это среда для разработки приложений на объектно-ориентированном языке программирования VB.NET .

Визуальная среда VB.NET представлена на рисунке 1.1.

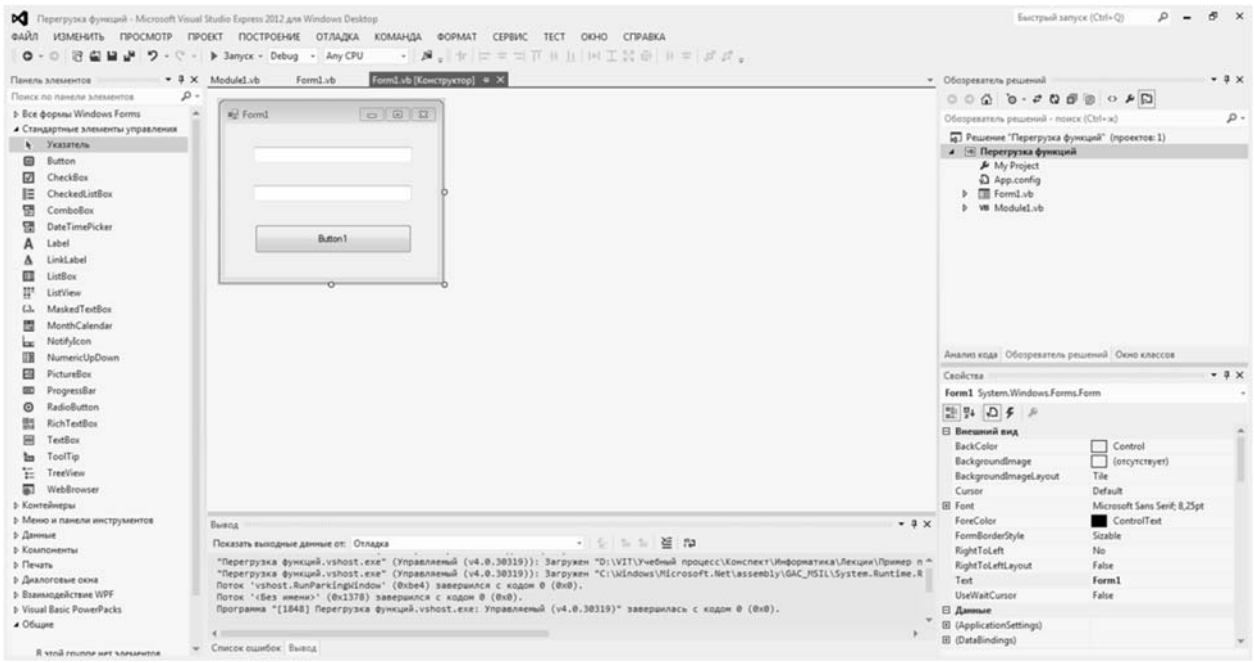

Рисунок 1.1 – Визуальная среда VB.NET

Главные составные части среды программирования VB.NET:

– Конструктор Форм;

– Окно Редактора Текста;

– Панель инструментов;

– Обозреватель решений;

– Панель свойств.

Программисты на VB.NET проводят большинство времени переключаясь между Дизайнером Форм и Окном Редактора (которое для краткости называют Редактор).

Панель инструментов представлена на рисунке 1.2. Панель инструментов позволяет Вам выбрать нужные объекты для размещения их на Дизайнере Форм. Для использования Панель инструментов просто первый раз щелкните мышью на один из объектов и потом второй раз – на Конструкторе Форм.

Выбранный Вами объект появится на проектируемом окне и им можно манипулировать с помощью мыши.

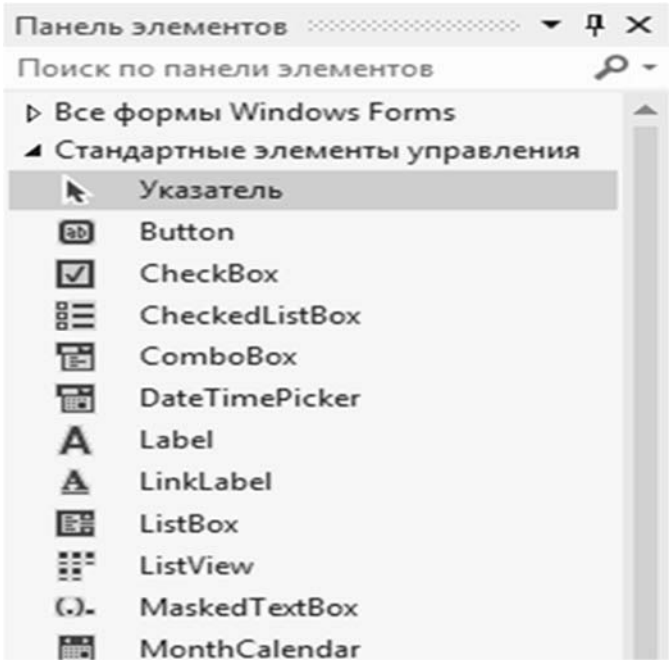

Рисунок 1.2 – Панель элементов VB.NET

Панель инструментов использует постраничную группировку объектов. Внизу Палитры находится набор закладок – «Стандартные элементы управления», «Компоненты» и т. д. Если Вы щелкнете мышью на одну из закладок, то сможете перейти на следующую страницу Панели инструментов.

Если поместить компонент TextBox на форму, его можно двигать с места на место. Можно использовать границу, прорисованную вокруг объекта для изменения его размеров. Большинством других компонент можно манипулировать тем же образом.

Панель свойств представлена на рисунке 1.3. Информация на Панели свойств меняется в зависимости от объекта, выбранного на форме. Важно понять, что каждый компонент является настоящим объектом и можно менять его вид и поведение с помощью Панели свойств.

Панель свойств состоит из двух страниц, каждую из которых можно использовать для определения поведения данного компонента. Первая страница – это список свойств, вторая – список событий. Если нужно изменить что-нибудь, связанное с определенным компонентом, то обычно делается это на Панели свойств. К примеру, можно изменить имя и размер компонента Label изменяя свойства Text, Left, Top, Height и Width.

Можно использовать закладки вверху Инспектора Объектов для переключения между страницами свойств и событий.

Наиболее часто используемые компоненты.

Label служит для отображения текста на экране. Вы можете изменить шрифт и цвет метки, если дважды щелкнете на свойство Font на Панели

свойств. Вы увидите, что это легко сделать и во время выполнения программы, написав всего одну строчку кода.

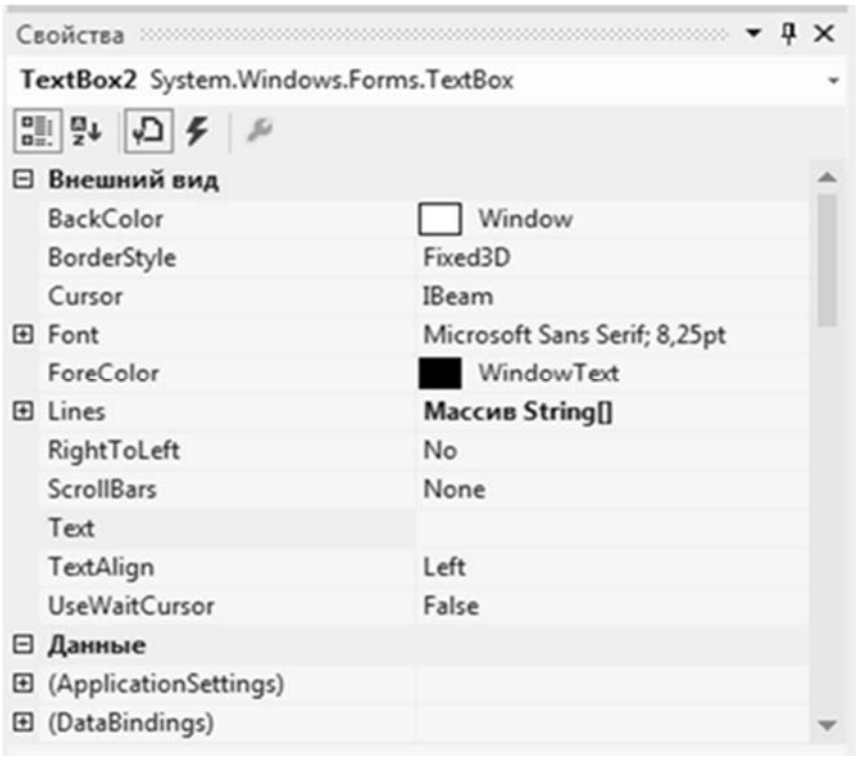

Рисунок 1.3 – Панель свойств VB.NET

TextBox – стандартный управляющий элемент Windows для ввода. Он может быть использован для отображения короткого фрагмента текста и позволяет пользователю вводить текст во время выполнения программы.

Button позволяет выполнить какие-либо действия при нажатии кнопки во время выполнения программы. В VB.NET все делается очень просто. Поместив Button на форму, Вы по двойному щелчку можете создать заготовку обработчика события нажатия кнопки. Далее нужно заполнить заготовку кодом.

CheckBox отображает строку текста с маленьким окошком рядом. В окошке можно поставить отметку, которая означает, что что-то выбрано. Например, если посмотреть окно диалога настроек компилятора (пункт меню Options | Project, страница Compiler), то можно увидеть, что оно состоит преимущественно из CheckBox’ов.

RadioButton позволяет выбрать только одну опцию из нескольких. Если Вы опять откроете диалог Options | Project и выберете страницу Linker Options, то можете видеть, что секции Map file и Link buffer file состоят из наборов RadioButton.

PictureBox – отображает на экране загруженные точечные рисунки и позволяет рисовать на своей поверхности с помощью графических методов.

Рассмотрим процесс создания простого приложения, которое вычисляет значение в соответствии с формулой $y = e^x \sin(x)$.

Создадим форму и разместим на ней в соответствии с рисунком 1.4:

– компонент Label со свойством Text «Вычислить уравнение»;

– компонент PictureBox и загрузим в свойство Image заранее подготовленный рисунок формулы;

– компонент Label со свойством Text «Введите X»;

– компонент TextBox;

– компонент Label со свойством Text «Результат»;

– компонент Button со свойством Text «Решение».

Дважды кликнем левой клавишей мыши по компоненту Button. В результате из конструктора перейдем на вкладку кода.

Создадим следующий код:

```
Public Class Form1
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim x, y As Double
    x = CDbl(Me.TextBox1.Text)
    y = Math.Exp(x) * Math.Sin(x)
    Me.TextBox2.Text = CStr(y)
End Sub
End Class
```

Запустим приложение на выполнение, напротив надписи «Введите X» в окне введем 3,5 и нажмем кнопку «Решение». Результат будет отображен в соответствии с рисунком 1.5.

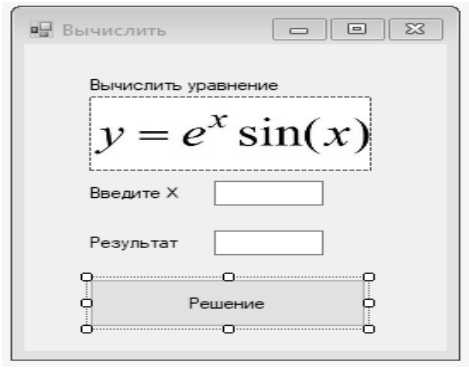

Рисунок 1.4 – Разработанная форма

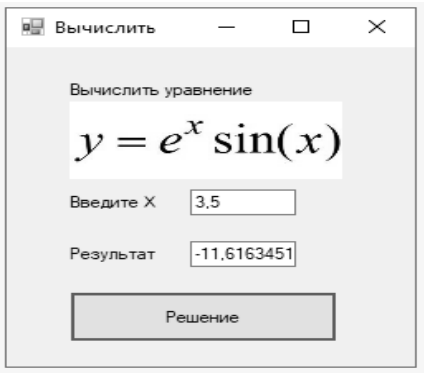

Рисунок 1.5 – Результат выполнения разработанного приложения

1.2 Порядок выполнения

1 Разработать блок-схему алгоритма в соответствии с заданием.

2 Создать форму приложения.

3 Набрать текст программы.

Контрольные вопросы

1 Свойства формы.
2 Основные компоненты, устанавливаемые на форму.
3 Использование математических функций.

2 Разработка приложений с использованием условных операторов

Цель работы: изучить условный оператор if и оператор выбора case.

2.1 Разработка приложений с использованием условного оператора if

Конструкция If ... Then проверяет истинность некоторого условия и зависимости от результатов проверки выполняет определенный код:

```
Dim num1 As Integer = 10
Dim num2 As Integer = 9
If (num1 > num2) Then
        Console.WriteLine(«Число {0} больше числа {1}», num1, num2)
End If
```

Здесь проверяем, больше ли число num1 числа num2, и если num1 больше чем num2, то выводим сообщение.

Но, допустим, мы захотим, чтобы в случае невыполнения этого условия тоже совершалось какое-нибудь действие. Тогда мы добавляем выражение Else и после него определяем те действия, которые будут совершаться, если программа не удовлетворяет условию в выражении If. Например, мы будем выводить сообщение, что первое число меньше второго:

```
Dim num1 As Integer = 10
Dim num2 As Integer = 9
  If (num1 > num2) Then
        Console.WriteLine(«Число {0} больше числа {1}», num1, num2)
  Else
        Console.WriteLine(«Число {0} меньше или равно числу {1}», num1,
num2)
End If
```

Разработаем приложение, определяющее в какой четверти лежит точка в соответствии с формой, представленной на рисунке 2.1, и следующим текстом программы:

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim X, Y As Double
    X = CDbl(Me.KX.Text)
    Y = CDbl(Me.KY.Text)
    If (X > 0) And (Y > 0) Then Me.Результат.Text = «Точка находится в I четверти»
    If (X < 0) And (Y > 0) Then Me.Результат.Text = «Точка находится во II четверти»
    If (X < 0) And (Y < 0) Then Me.Результат.Text = «Точка находится в III четверти»
    If (X > 0) And (Y < 0) Then Me.Результат.Text = «Точка находится в IV четверти»
End Sub
End Class
```

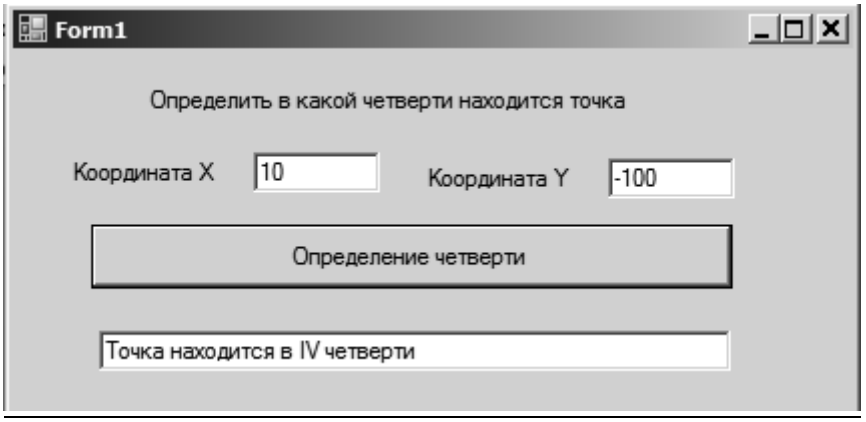

Рисунок 2.1 – Форма приложения, определяющего в какой четверти лежит точка

2.2 Разработка приложений с использованием оператора выбора case

Конструкция Select Case подобна в конструкции If ... Then, т. к. позволяет обрабатывать сразу несколько условий. После слов Select Case указывается сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после оператора Case. И в случае если значения совпали, то выполняется блок команд, помещенных после данного оператора Case. Конструкция завершается словами End Select. Если мы хотим определить действия, которые будут выполняться, если совпадений не выявлено, то мы можем использовать оператор Case Else, после которого помещаем блок действий по умолчанию. Блок Case Else необязателен и может не употребляться.

```
Dim num1 As Integer = 10
Select Case num1
   Case 1
      Console.WriteLine(«Число num1 равно 1»)
   Case 2
      Console.WriteLine(«Число num1 равно 2»)
   Case 3 To 25
      Console.WriteLine(«Число num1 находится на отрезке от 3 до 25»)
   Case Else
      Console.WriteLine(«Число num1 больше 25»)
End Select
```

Num1 должен быть равен одному их простых типов данных (Boolean, Byte, Char, Date, Double, Decimal, Integer, Long, Object, SByte, Short, Single, String, UInteger, ULong и UShort).

Разработаем приложение, определяющее по номеру название дня недели в соответствии с формой, представленной на рисунке 2.2, и следующим текстом программы:

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    Dim N As Integer
    Dim Naz As String
    N = CInt(Me.Номер.Text)
    Select Case N
    Case 1
    Naz = «Понедельник»
    Case 2
    Naz = «Вторник»
    Case 3
    Naz = «Среда»
    Case 4
    Naz = «Четверг»
    Case 5
    Naz = «Пятница»
    Case 6
    Naz = «Суббота»
    Case 7
    Naz = «Воскресенье»
    Case Else
    Naz = «Введите правильно номер дня недели»
    End Select
```

```
    Me.Название.Text = Naz
End Sub
End Class
```

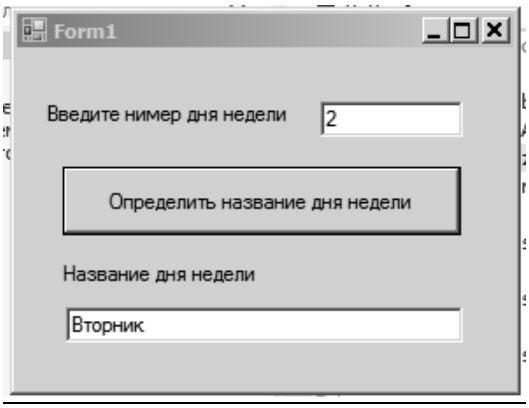

Рисунок 2.2 – Форма приложения, определяющего по номеру название дня недели

2.3 Порядок выполнения

1 Разработать блок-схему алгоритма в соответствии с заданием.
2 Создать форму приложения.
3 Набрать текст программы.

Контрольные вопросы

1 Конструкция оператора if.
2 Задание условий.
3 С какими типами работает оператор if.
4 Конструкция оператора Case.
5 Задание условий.
6 С какими типами работает оператор Case.

3 Создание приложения с использованием циклических алгоритмов

Цель работы: изучить основные операторы и конструкции языка VB.NET для программирования циклических алгоритмов.

3.1 Цикл For...Next

В цикл выполняется определенное число раз, причем это число задается счетчиком:

```
For i = 1 To 9
   Console.WriteLine(«Квадрат числа {0} равен {1}», i, i * i)
Next
```

Здесь переменная i выполняет роль счетчика. После слова To мы помещаем максимальное значение счетчика. При каждом цикле значение счетчика увеличивается на единицу. И это значение сравнивается со значением после To. Если эти два значения равны, то цикл прекращает свою работу.

При работе с циклами мы можем увеличивать значение счетчика при каждом проходе не только на единицу, но и вообще на любое число. Для этого нужно либо использовать ключевое слово Step и после него указать шаг цикла, на который будет увеличиваться значение счетчика, либо можно увеличивать счетчик непосредственно в цикле:

```
For i = 1 To -9 Step -1
   For j = 1 To 9
      Console.WriteLine(«Произведение чисел i и j равно {0}», i * j)
      j =j+1
   Next
Next
```

Обратите внимание, что в качестве шага в первом цикле выбрано отрицательное значение и значение счетчика с каждым проходом уменьшается на единицу. Во внутреннем цикле счетчик j при каждом проходе увеличивается на 2, т. к. он по умолчанию увеличивается на единицу, и еще мы явным образом увеличиваем его в цикле на единицу. В итоге внутренний цикл отрабатывает пять раз, а внешний девять, т. е. фактически получается 45 циклов.

Разработаем приложение, вычисляющее факториал числа в соответствии с формой, представленной на рисунке 3.1, и следующим текстом программы:

```
Public Class Form1
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
      Dim n, i, R As Integer
      n = CInt(Me.TextBox1.Text)
      R = 1
      For i = 1 To n
      R = R * i
      Next i
      Me.TextBox2.Text = CStr(R)
End Sub
End Class
```

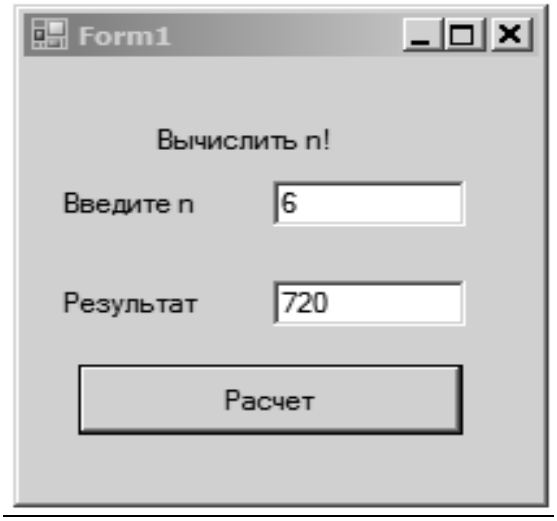

Рисунок 3.1 – Форма приложения, вычисляющего факториал

3.2 Цикл Do ... Loop

Цикл Do выполняется, пока соблюдается определенное условие. Однако он имеет разные формы. Так, в следующем примере сначала проверяется условие, а затем выполняется блок кода, определенный в цикле:

```
Dim j As Integer = 10
Do While j > 0
   Console.WriteLine(j)
   j -= 1
Loop
```

В данном случае цикл выполняется, пока значение j больше нуля. Если изначально условие, заданное в цикле, неверно, то цикл не будет работать.

Но мы можем определить проверку в конце цикла, и таким образом наш цикл как минимум один раз отработает:

```
Do
   Console.WriteLine(j)
   j -= 1
Loop While j > 0
```

Вместо ключевого слова While можно использовать Until.

При использовании ключевого слова While цикл повторяется до тех пор, пока условие равно False.

При использовании ключевого слова Until цикл повторяется до тех пор, пока условие равно True.

Разработаем приложение, вычисляющее сумму ряда в соответствии с формой, представленной на рисунке 3.2 и следующим текстом программы:

```
Public Class Form1

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    Dim S As Double = 0
    Dim S1 As Double = 0
    Dim n As Integer
    Dim k As Integer = 0
    Do
    S1 = S
    S = 0
    k = k + 1
    For n = 1 To k
    S = S + 20 / (5 * (n ^ 2) + 10 * n + 15)
    Next n
    Loop Until Math.Abs(S - S1) <= 0.001
    Me.TextBox1.Text = CStr(S)
    Me.TextBox2.Text = CStr(k)
End Sub
End Class
```

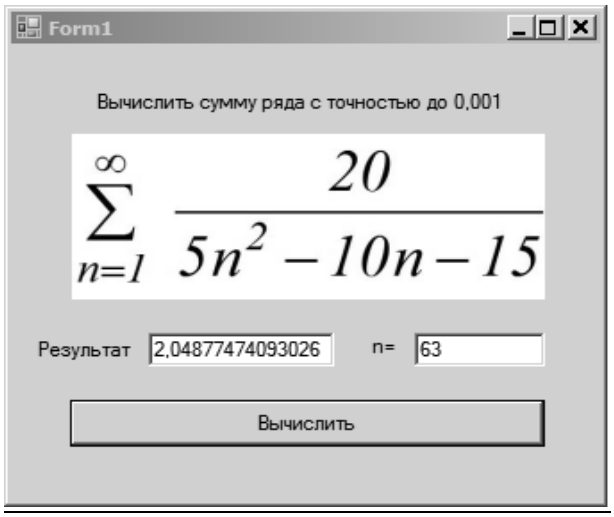

Рисунок 3.2 – Форма приложения, вычисляющего сумму ряда

3.3 Порядок выполнения

1 Разработать блок-схему алгоритма в соответствии с заданием.
2 Создать форму приложения.
3 Набрать текст программы.

Контрольные вопросы

1 Конструкция оператора for.
2 Конструкция оператора do.
3 С какими типами работают операторы for и do.

4 Создание приложения для вычисления численными методами определенного интеграла

Цель работы: изучить использование основных операторов и конструкций языка VB.NET для программирования циклических алгоритмов при решении задач численными методами.

4.1 Общие сведения

Многие инженерные задачи, задачи физики, геометрии и многих других областей человеческой деятельности приводят к необходимости вычислять определенный интеграл.

Пусть требуется вычислить с заданной точностью значение интеграла функции $f(x)$ на отрезке $[a, b]$:

$$\int_a^b f(x)dx .$$

Найти значение определенного интеграла – это значит найти площадь заштрихованной области (площадь криволинейной трапеции). Известно много способов вычисления подобных интегралов либо с помощью первообразной, либо используя приближенные методы. Вычисление интеграла будем производить, используя приближенный метод – метод прямоугольников.

При вычислении интеграла следует помнить, каков геометрический смысл определенного интеграла. Если $f(x) >= 0$ на отрезке $[a, b]$, то он численно равен площади фигуры, ограниченной графиком функции $y = f(x)$, отрезком оси абсцисс, прямой $x = a$ и прямой $x = b$ (рисунок 4.1) Таким образом, вычисление интеграла равносильно вычислению площади криволинейной трапеции.

Разделим отрезок $[a, b]$ на n равных частей, т.е. на n элементарных отрезков.

Тогда длина каждого элементарного отрезка

$$h = \frac{b-a}{n} .$$

Точки деления отрезка $[a, b]$ получим добавляя шаг h к нижнему пределу интегрирования (началу отрезка):

$$x_0 = a;$$

$$x_1 = a + h;$$

$$x_2 = a + 2h;$$

...

$$x_{n-1} = a + (n-1)h;$$

$$x_n = b.$$

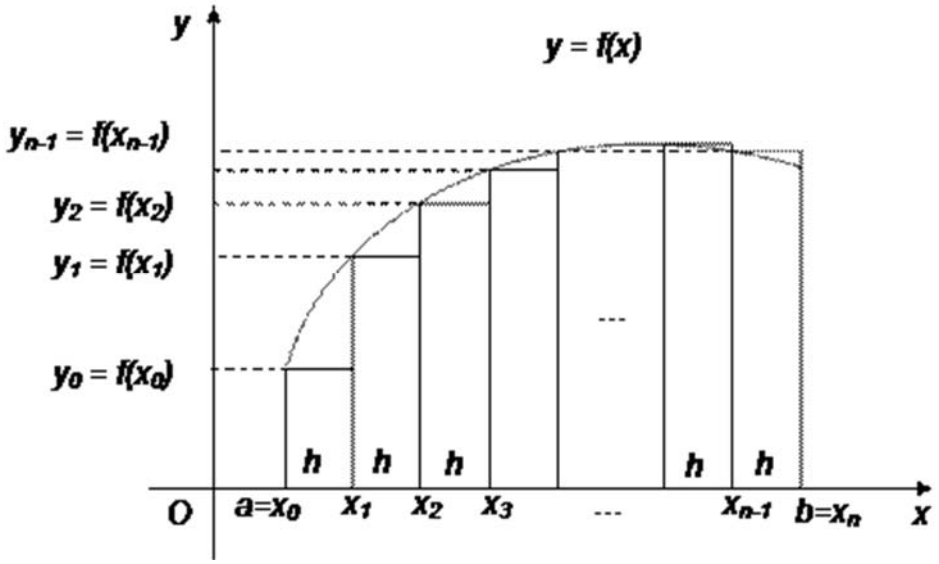

Рисунок 4.1 – График функции $f(x)$

Далее производим вычисление значений интегралов на каждом из элементарных участков как площадей прямоугольников (т. е. ширина прямоугольника равна h, а высота значению функции $f(x)$) при этом область фигуры, не попадающая в прямоугольник, является погрешностью вычисления и очевидно, что чем меньше ширина h прямоугольников и больше их количество, тем меньше погрешность – неучтенная площадь фигуры.

Зависимость для определения суммы площадей элементарных прямоугольников:

$$\int_a^b f(x)dx = \int_a^{x_1} f(x) + \int_{x_1}^{x_2} f(x) + ... + \int_{x_{n-1}}^{b} f(x) \approx \sum_{i=1}^{n} f(x_{i-1}) \cdot h \approx$$

$$\approx \sum_{i=0}^{n-1} f(x_{i+1}) \cdot h = \sum_{i=1}^{n} (f(a + (i-1) \cdot h) \cdot h.$$

Полученную формулу достаточно просто реализовать средствами среды программирования VB.NET.

Порядок решения задачи на ЭВМ следующий.

Как правило в первом приближении отрезок интегрирования разбивают на n частей (можно начать с двух), значение интеграла, вычисленное при n-разбиений обозначим через S_1.

Одно это приближение не позволяет оценить точность, с которой вычислено значение интеграла, необходимо найти второе приближение. Для этого увеличим n в 2 раза:

$$n = 2n .$$

Значение интеграла, вычисленное при n-разбиений обозначим через через S_2.

Далее чтобы установить достигли мы заданной точности ε или нет, проверим условие

$$| S_1 - S_2 | < \varepsilon .$$

Далее действуем по алгоритму:

1) если условие выполняется (true, истинно), то S_2 принимается за искомое значение интеграла;

2) если не выполняется (false, ложно), то последнее полученное значение S_2 считается предыдущим, т. е. $S_1 = S_2$, и удвоим число точек деления отрезка $[a, b]$, $n=2n$;

3) вычислим новое значение S_2;

4) процесс удвоения n и вычисления S_2 и замены его с S_1 будем продолжать до тех пор, пока условие не станет истинным.

Разработаем приложение, вычисляющее определенный интеграл в соответствии с формой, представленной на рисунке 4.2, и следующим текстом программы:

```
Public Class Form1
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    Dim y As Single, i As Integer
    Dim x, dx As Single
    Dim a As Single, b As Single, S As Single, N As Integer
    Dim S1 As Double
    a = CDbl(Me.НП.Text)
    b = CDbl(Me.ВП.Text)
    S = 0
    S1 = 0
    N = 1
    Do
    S1 = S
    S = 0
    x = a
```

```
        N = N * 2
        dx = (b - a) / N
        For i = 1 To N
        y = (x ^ 2) / (1 + x)
        S = S + y * dx
        x = x + dx
        Next i
        Loop Until Math.Abs(S1 - S) <= 0.0001
        Me.Результат.Text = CStr(S)
End Sub
End Class
```

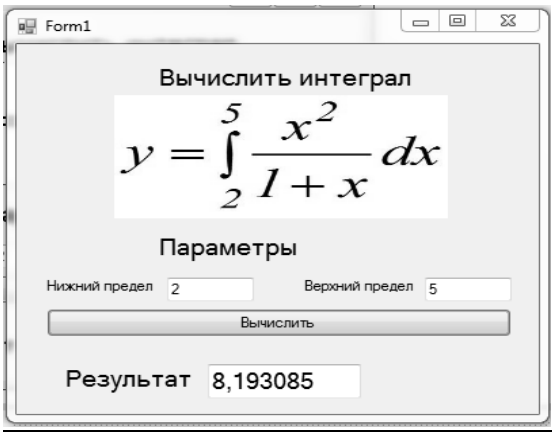

Рисунок 4.2 – Форма приложения, вычисляющего значение определенного оитеграла

4.2 Порядок выполнения

1 Разработать блок-схему алгоритма в соответствии с заданием.
2 Создать форму приложения.
3 Набрать текст программы.

Контрольные вопросы

1 Каков геометрический смысл определенного интеграла?
2 Порядок вычисления определенного интеграла на ЭВМ.
3 Понятие метода прямоугольников.

5 Создание приложения для создания и редактирования табличных данных

Цель работы: изучить работу с табличными данными, используя циклические алгоритмы.

5.1 Общие сведения

Объект **DataGridView** предназначен для отображения всей информации из таблиц, запросов или фильтров на форме в виде таблицы. Этот объект может быть создан как вручную (с последующим его подключением), так и перетаскиванием всего источника данных из окна **«Data Sources»**. Однако наиболее часто его создают перетаскиванием всей таблицы, запроса или фильтра из окна **«Data Sources»** на форму.

При перетаскивании этого объекта на форму, как и в случае с другими объектами, появляется панель навигации. Она выполняет функции: перемещение по записям, добавление, удаление и сохранение записей. После создания объекта **DataGridView** можно настраивать как свойства всего объекта, так и свойства отдельных столбцов. Начнём с настройки свойств всего объекта. Настройка данных свойств осуществляется в основном через меню действий. Возможны следующие настройки:

– **Choose Data Source** - источник данных, отображаемый в таблице;

– **Enable Adding** – добавлять записи;

– **Enable Deleting** – разрешается пользователям удалять записи;

– **Enable Editing** – разрешается пользователям изменять значения полей таблицы;

– **Enable Column Reordering** – разрешается пользователям изменять порядок столбцов, просто перетаскивая их мышью.

Также в меню действий возможны следующие действия с таблицей:

– **Dock in parent container** – вписать объект в форму;

– **Preview Data** – появляется окно с предварительным просмотром таблицы;

– **Add Query** – добавляет SQL-запрос, который выполняется на стороне клиента;

– **Add Column** – добавление нового столбца в таблицу;

– **Edit Columns** – настройка свойств отдельных столбцов таблицы.

Теперь перейдём к настройке отдельных столбцов таблицы.

Настройка свойств столбцов в DataGridView.

Если в меню действий выбрать пункт **«Edit Columns»**, то появляется окно, где можно добавлять, удалять и редактировать столбцы. Для этого в списке столбцов левой части окна выбираем столбец, а в правой – настраиваем его свойства. Наиболее часто настраиваются следующие свойства:

– **Name** – имя столбца;

– **AutoSizeMode** – подгонка ширины столбца по его содержимому;

– **ColumnType** – определяет внешний вид ячеек столбца (какой объект для отображения информации находится в ячейках столбца);
– **DataPropertyName** – имя, отображающего в столбце поля;
– **Frozen** – фиксация столбца (столбец не передвигается при прокручивании таблицы);
– **HeaderText** – текст заголовка столбца;
– **Width** – ширина поля;
– **MaxInputLength** – максимально вводимая длина текста;
– **MinimumWidth** – минимальная ширина столбца;
– **ReadOnly** – блокировка столбца для редактирования данных;
– **Resizable** – разрешает менять ширину столбца;
– **SortMode** – сортировка данных в таблице по этому столбцу;
– **ToolTipText** – всплывающая подсказка для столбца;
– **Visible** – делает столбец невидимым.

Для добавления новых столбцов в окне **«Edit Columns»** необходимо нажать кнопку **Add**, а для удаления кнопку **Remove**.

Если необходимо настроить внешний вид всех ячеек таблицы, то для этого необходимо выделить объект **DataGridView** и на панели свойств зайти в свойство **DefaultCellStyle**. Появится окно со свойствами всех ячеек таблицы.

В объекте **DataGridView** имеется возможность сортировки данных. Для этого используется метод Sort, имеющий следующий синтаксис:

DataGridView.Sort(<Имя столбца>, <Порядок сортировки>),

где DataGridView – это имя объекта;
<Имя столбца> – имя столбца (свойство **Name**), по которому происходит сортировка записей в таблице;
<Порядок сортировки> – определяет порядок сортировки и может принимать два значения:
1) System.ComponentModel.ListSortDirection.Ascending – сортировка по возрастанию;
2) System.ComponentModel.ListSortDirection.Descending – сортировка по убыванию.

Доступ к отдельным ячейкам таблицы можно получить через подобъект **Item**. Обращение к нему осуществляется следующим образом:

DataGridView.Item(i, j).<Свойство>.

Здесь DataGridView – это имя объекта, i – горизонтальная координата ячейки, а j – вертикальная, <Свойство> – это настраиваемое свойство ячейки.

Пример – В верхнюю левую ячейку таблицы записать слово «Привет» и сделать цвет текста в ячейке красным.

DataGridView.Item(0, 0).Value = «Привет»
DataGridView.Item(0, 0).Style.ForeColor = Color.Red

Здесь DataGridView – это имя объекта, свойство Value определяет содержимое ячейки таблицы, свойство Style.ForeColor определяет цвет текста в ячейке. Нумерация столбцов и строк в таблице начинается с нуля.

Разработаем приложение для вычисления элементов массива.

Поместим на форму компонент DataGridView, свойству Name присвоим значение «Таблица» и добавим три столбца с именами и подписями Стб1, Стб2 и Стб3. Поместим элементы Button и TextBox. Свойству TextBox Name присвоим «Сумма» в соответствии с формой, представленной на рисунке 5.1.

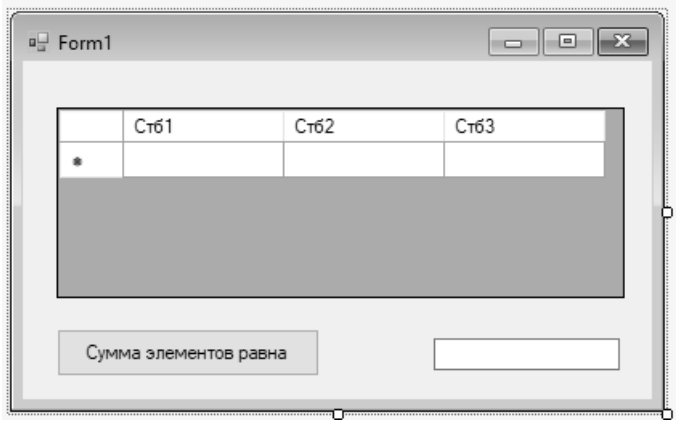

Рисунок 5.1 – Форма приложения для вычисления элементов массива

Создадим следующий код:

```
Public Class Form1
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim КолСтб, КолСтр, i, j As Integer
    Dim Сум As Double = 0
    КолСтб = Me.Таблица.ColumnCount
    КолСтр = Me.Таблица.RowCount - 1
    For i = 0 To КолСтб - 1
    For j = 0 To КолСтр - 1
    Сум = Сум + CDbl(Таблица(i, j).Value)
    Next j
    Next i
    Сумма.Text = CStr(Сум)
End Sub
End Class
```

Введем двухмерный массив. На рисунке 5.2 представлена форма для ввода двухмерного массива и вычисления суммы его элементов.

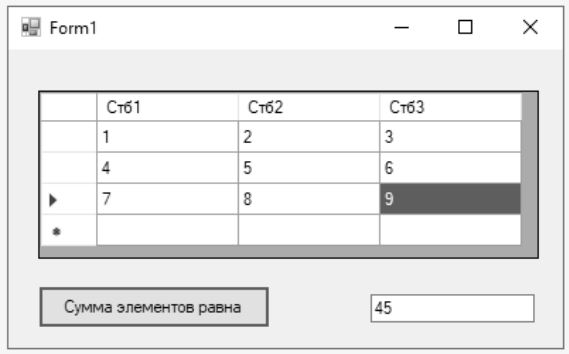

Рисунок 5.2 – Результат ввода и вычисления суммы элементов массива

5.2 Порядок выполнения

1 Разработать блок-схему алгоритма в соответствии с заданием.
2 Создать форму приложения.
3 Набрать текст программы.

Контрольные вопросы

1 Назначение объекта DataGridView.
2 Свойства объекта DataGridView.
3 Методы ввода текста в ячейки.

6 Создание приложений с использованием графики

Цель работы: ознакомиться с графическими возможностями визуальной средой программирования VB.NET.

6.1 Общие сведения

Рисовать можно с помощью методов класса Graphics. Рассмотрим следующие методы.

DrawLine(Pen, Point1, Point2) – проводит линию, соединяющую две структуры Point.

Point – структура. Представляет упорядоченную пару целых чисел – координат X и Y, определяющую точку на двумерной плоскости.

Параметр Pen – задает цвет линии.

Pens.Black – черный.

Pens.White – белый

DrawEllipse(Pen, Rectangle) – рисует эллипс, определяемый ограничивающей структурой Rectangle (рисунок 6.1).

Pen –задает цвет линии.

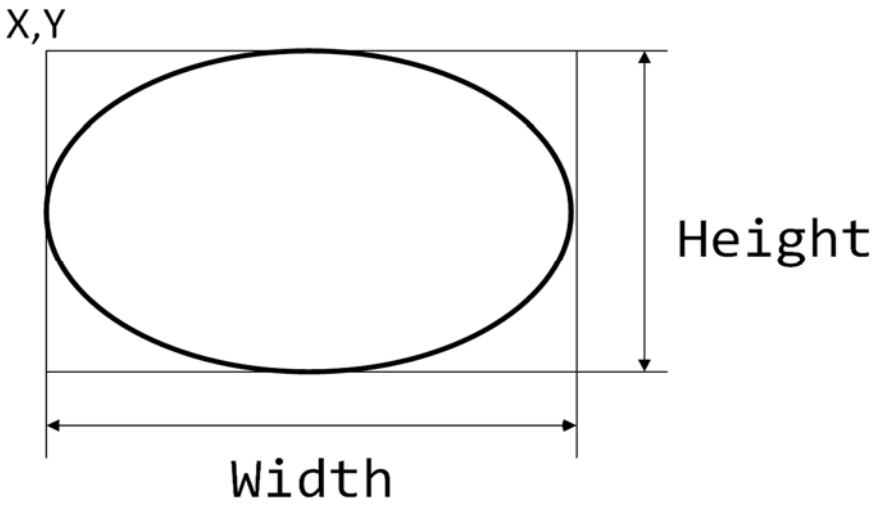

Рисунок 6.1 – Результат выполнения команды DrawEllipse

Rectangle – задает прямоугольник с координатами верхнего левого угла, высоты и ширины

```
Dim R As Rectangle
  R.X = P.X - 20
      R.Y = P.Y
      R.Height = 40
      R.Width = 40
```

DrawArc(Pen, Rectangle, Nangle, Angle).

Дуга является частью эллипса. Чтобы нарисовать дугу, вызовите DrawArc метод Graphics класса. Параметры DrawArc метода не отличаются от параметров DrawEllipse метода, за исключением того, что DrawArc требует начальный угол (Nangle) и угол поворота (Angle). В следующем примере рисуется дуга с начальным углом 0 град и угол поворота 225 град по часовой стрелке (рисунок 6.2).

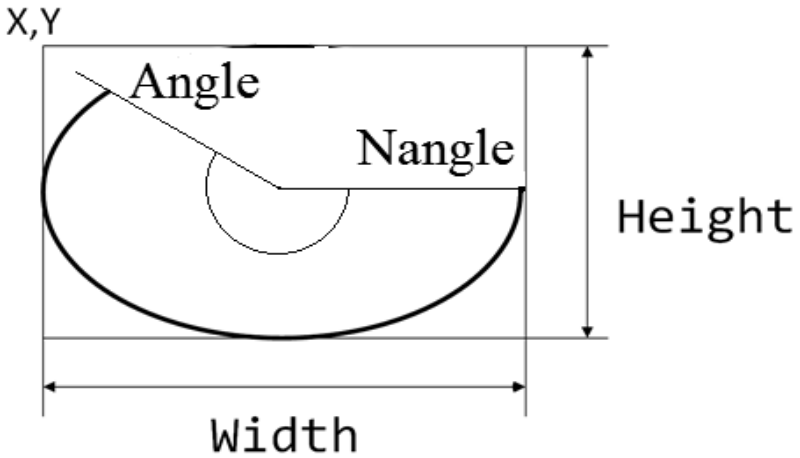

Рисунок 6.2 – Результат выполнения команды DrawArc

Разработаем приложение, которое выводит различные графические элементы в соответствии с формой, представленной на рисунке 6.3, и следующим текстом программы:

```
Public Class Form1
Private Sub Рисование_Click(sender As Object, e As EventArgs) Handles Рисование.Click
    Dim Рисунок As Graphics
    Dim Прямоугольник As Rectangle
    Dim N, K, S As Point
    Рисунок = Полотно.CreateGraphics
    Прямоугольник.X = 10
    Прямоугольник.Y = 10
    Прямоугольник.Width = 100
    Прямоугольник.Height = 100
    Рисунок.DrawRectangle(Pens.Black, Прямоугольник)
    Рисунок.DrawEllipse(Pens.Black, Прямоугольник)
    Прямоугольник.X = 130
    Прямоугольник.Y = 10
    Прямоугольник.Width = 100
    Прямоугольник.Height = 100
    Рисунок.DrawRectangle(Pens.Black, Прямоугольник)
    Рисунок.DrawArc(Pens.Black, Прямоугольник, 0, 225)
    N.X = 10
    N.Y = 130
    K.X = 110
    K.Y = 230
    Рисунок.DrawLine(Pens.Black, N, K)
    N.X = 180
    N.Y = 130
    K.X = 230
    K.Y = 230
    S.X = 130
    S.Y = 230
    Рисунок.DrawLine(Pens.Black, N, K)
    Рисунок.DrawLine(Pens.Black, S, K)
    Рисунок.DrawLine(Pens.Black, N, S)
End Sub
End Class
```

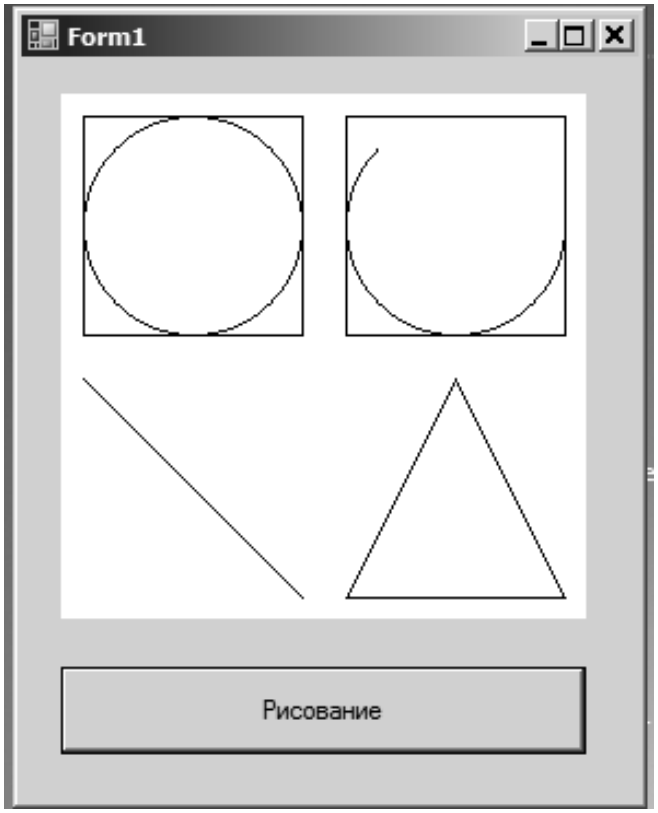

Рисунок 6.3 – Результат рисования графических элементов

6.2 Порядок выполнения

1 Разработать блок-схему алгоритма в соответствии с заданием.
2 Создать форму приложения.
3 Набрать текст программы.

Контрольные вопросы

1 Основные методы класса Graphics.
2 Понятие структуры Point.
3 Понятие структуры Rectangle.

7 Создание приложений с использованием объектов

Цель работы: получить навыки создания приложений с использованием объектно-ориентированного программирования.

7.1 Общие сведения

Visual Basic.NET является полноценным объектно-ориентированным языком, а это значит, что программа может быть представлена в виде взаимосвязанных объектов, которые взаимодействуют между собой. Описанием объекта является класс, в то время как объект – экземпляр этого класса. Класс определяется с помощью ключевого слова Class:

```
Class Book
Поля и методы
End Class
```

Всю функциональность класса обеспечивают его члены - поля, свойства, методы, конструкторы, события. Поля представляют обычные переменные, обычным образом также определяются процедуры и функции:

```
Public Class TLine
   Public Im As Graphics = Form1.Pol
   Public MoveTo As Point
   Public Sub RisLine(ByVal P As Point, ByVal Reg As Byte)
      If Reg = 1 Then
         Im.DrawLine(Pens.Black, MoveTo, P)
         MoveTo = P
      Else
         Im.DrawLine(Pens.White, MoveTo, P)
         MoveTo = P
      End If
   End Sub
End Class
```

Создание экземпляра производится следующей командой:

```
Dim Line  As New TLine()
```

Доступ к полям и методам осуществляется следующим образом:

Имя_экземпляра_класса.Имя_поля или
Имя_экземпляра_класса.Имя_метода

Например:

```
Line.MoveTo=Нач_Точка
Line. RisLine(P1, 1)
```

Одним из ключевых аспектов объектно-ориентированного программирования является наследование (inheritance). Сущность наследования заключается в том, что мы можем расширить функционал уже имеющихся классов с помощью создания наследников этого класса. Чтобы наследовать один класс от другого, нужно использовать ключевое слово Inherits:

```
Public Class TPrugina
   Inherits Tline              'Tprugina наследует класс Tline
   Protected P1 As Point
   Protected PruginaEnd As Point
   Private PointArr(0 To 10) As Point
   Public Sub Nach(ByVal P As Point)
      MoveTo.X = P.X – 10       'Использование MoveTo класса Tline
      MoveTo.Y = P.Y
      P1.X = P.X + 10
      P1.Y = P.Y
      RisLine(P1, 1) 'Использование RisLine класса Tline
      MoveTo = P
      P1.X = P.X
      P1.Y = P.Y + 10
      RisLine(P1, 1)
   End Sub
Public Sub PruginaRis(ByVal P As Point, ByVal dY As Integer, ByVal Reg As Byte)
      Dim i, os As Integer
      PointArr(1).X = P.X – 10
      PointArr(1).Y = P.Y + 10
      For i = 2 To 10
         os = i Mod 2
         If os = 0 Then
            PointArr(i).X = PointArr(i – 1).X + 20
         Else
            PointArr(i).X = PointArr(i – 1).X – 20
         End If
         PointArr(i).Y = PointArr(i – 1).Y + dY
      Next i
      Nach(P)
      For i = 1 To 10
         RisLine(PointArr(i), Reg)
      Next i
```

```
        P1.X = PointArr(10).X – 10
        P1.Y = PointArr(10).Y
        RisLine(P1, Reg)
        PruginaEnd.X = PointArr(10).X – 10
        PruginaEnd.Y = PointArr(10).Y + 10
        RisLine(PruginaEnd, Reg)
    End Sub
End Class
```

По умолчанию все классы могут наследоваться. Однако здесь есть ряд ограничений:

– во-первых, не поддерживается двойное наследование, класс может наследоваться только от одного класса. Хотя проблема множественного наследования реализуется с помощью концепции интерфейсов, о которых мы поговорим позже;

– во-вторых, при создании производного класса мы должны учитывать тип доступа к базовому классу – тип доступа к производному классу должен быть таким же, как и у базового класса, или более строгим. То есть если базовый класс у нас имеет тип доступа Friend, то производный класс может иметь тип доступа Friend или Private, но не Public.

Полиморфизм – расширение принципа наследования в объектно-ориентированном программировании.

Полиморфизм – наиболее туманный принцип ООП. Суть его состоит в том, что классы-потомки могут иметь методы с тем же самым названием, что и в родительском классе, но при этом выполнять другие действия, специфические для каждого конкретного класса.

Для этого нам надо в базовом классе пометить изменяемый метод модификатором Overridable. А уже в производном классе этот же метод использовать с модификатором Overrides:

```
Public Class TMehanizm1
    Inherits Tprugina          ‘Наследуем класс Tprugina
‘Разрешаем методу GruzRis перезапись
    Public Overridable Sub GruzRis(ByVal P As Point, ByVal Reg As Byte)
        P1.X = P.X – 20
        P1.Y = P.Y
        RisLine(P1, Reg)

        P1.X = P.X – 20
        P1.Y = P.Y + 20
        RisLine(P1, Reg)
```

```
            P1.X = P.X + 20
            P1.Y = P.Y + 20
            RisLine(P1, Reg)

            P1.X = P.X + 20
            P1.Y = P.Y
            RisLine(P1, Reg)
            RisLine(P, Reg)
        End Sub

    Public Class TMehanizm2
        Inherits TMehanizm1     'Наследуем класс TMehanizm1
    'Перезаписываем метод GruzRis, т.е. меняем его содержимое
        Public Overrides Sub GruzRis(ByVal P As Point, ByVal Reg As Byte)
            P1.X = P.X – 20
            P1.Y = P.Y + 20
            RisLine(P1, Reg)

            P1.X = P.X
            P1.Y = P.Y + 40
            RisLine(P1, Reg)

            P1.X = P.X + 20
            P1.Y = P.Y + 20
            RisLine(P1, Reg)

            RisLine(P, Reg)
        End Sub
    End Class

    Public Class Form1
        Dim Vst As Point
        Public Graf, Pol As Graphics
        Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
            Graf = Me.CreateGraphics          'Для рисования на форме
            Pol = PB.CreateGraphics
            Dim Mehanizm1 As TMehanizm1 = New TMehanizm1
            Dim Mehanizm2 As New TMehanizm2()
            Dim Mehanizm3 As New TMehanizm3
            Dim Mehanizm4 As New TMehanizm4
            Mehanizm4.Im = Graf                    'Для рисования на форме
            Pol.Clear(Color.White)
            Graf.Clear(Color.FromKnownColor(KnownColor.Control))
```

```
        Application.DoEvents()
        Vst.X = 50
        Vst.Y = 20
        Mehanizm1.Rabota(Vst)
        Vst.X = 100
        Vst.Y = 20
        Mehanizm2.Rabota(Vst)
        Vst.X = 150
        Vst.Y = 20
        Mehanizm3.Rabota(Vst)
        Vst.X = 200
        Vst.Y = 20
        Mehanizm4.Rabota(Vst)
    End Sub
End Class
```

Результат выполнения программы показан на рисунке 7.1.

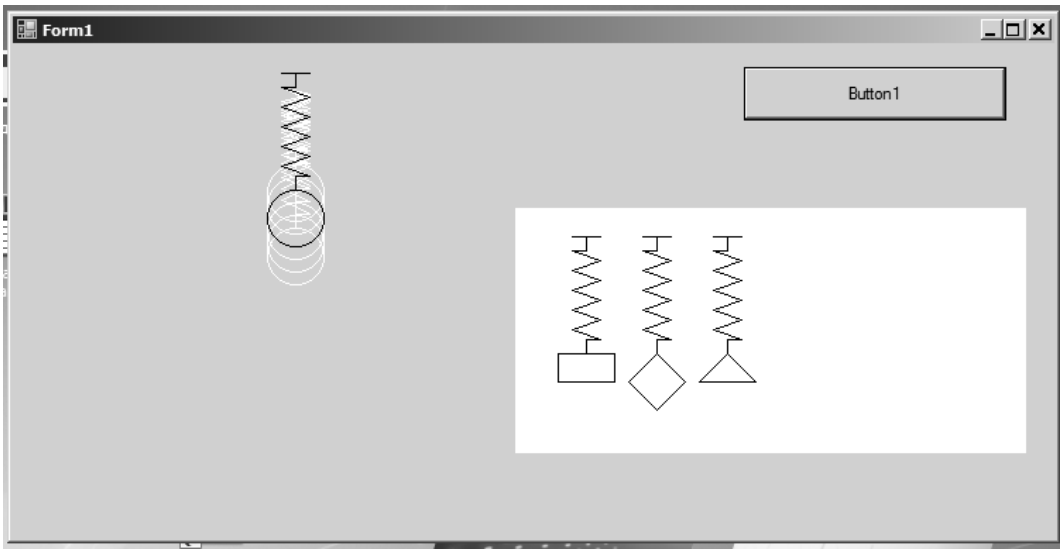

Рисунок 7.1 – Результат выполнения приложения, созданного с использованием методов объектно-ориентированного программирования

7.2 Порядок выполнения

1 Разработать блок-схему алгоритма в соответствии с заданием.
2 Создать форму приложения.
3 Набрать текст программы.

Контрольные вопросы

1 Понятие типа «класс».
2 Понятие инкапсуляции, наследования и полиморфизма.

Список литературы

1 **Шакин, В. Н.** Базовые средства программирования на Visual Basic в среде Visual Studio .NET : учебное пособие / В. Н. Шакин. – Москва : ФОРУМ; ИНФРА-М, 2019. – 304 с.

2 **Шакин, В. Н.** Базовые средства программирования на Visual Basic в среде Visual Studio.NET. Практикум : учебное пособие / В. Н. Шакин. – Москва: ФОРУМ; ИНФРА-М, 2021. – 287 с

3 **Хорев, П. Б.** Объектно-ориентированное программирование : учебное пособие / П. Б. Хорев. – 4-е изд., стер. – Москва : Академия, 2012. – 448 с.

4 **Майо, Д.** Самоучитель Microsoft Visual Studio 2010 / Д. Майо. – Санкт-Петербург: БХВ-Петербург, 2011. – 464 с.