

МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Транспортные и технологические машины»

ПАКЕТЫ ПРИКЛАДНЫХ МАТЕМАТИЧЕСКИХ ПРОГРАММ

*Методические рекомендации к практическим занятиям
для студентов направления подготовки
23.03.02 «Наземные транспортно-технологические комплексы»
очной формы обучения*



Могилев 2022

УДК 004.4
ББК 32.973-018
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 практическим занятиям 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. Подошевка
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.

© Белорусско-Российский
университет, 2022

Содержание

Введение.....	4
1 Среда программирования Microsoft Visual Studio	5
2 Основы языка VB.NET	8
3 Разработка программ на ПЭВМ.....	18
4 Программы, подпрограммы, программные модули	29
5 Графические возможности VB.NET.....	32
6 Объектно-ориентированное программирование	33
Список литературы	35

Введение

Целью преподавания дисциплины является формирование необходимых знаний для использования современных базовых компьютерных технологий в качестве инструмента решения практических задач в своей предметной области, а также изучение современных методов постановки, алгоритмизации, программирования и решения задач с применением средств вычислительной техники.

В результате изучения дисциплины студент узнает основы алгоритмизации задач; сумеет разрабатывать алгоритмы и программы на алгоритмическом языке VB.NET для разработки приложений, предназначенных для решения различных математических задач.

1 Среда программирования Microsoft Visual Studio

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.

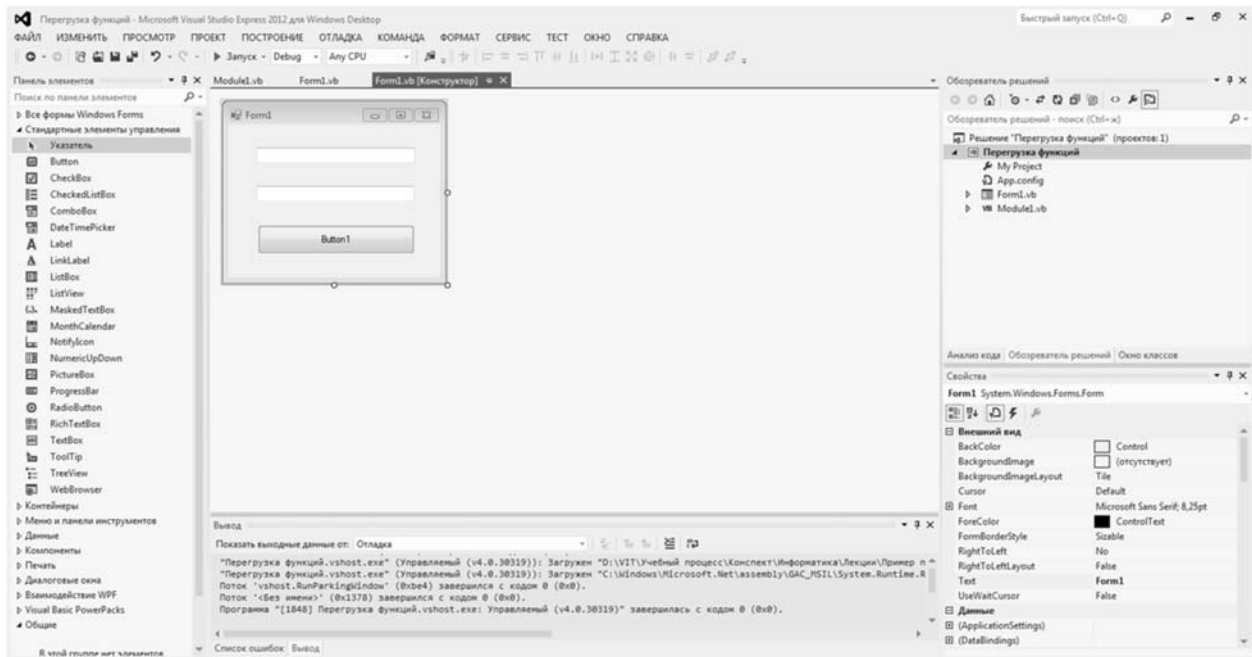


Рисунок 1.1 – Визуальная среда VB.NET

Главные составные части среды программирования VB.NET:

- Конструктор Форм;
- Окно Редактора Текста;
- Панель инструментов;
- Обзоратель 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 Панели инструментов просто первый раз щелкните мышкой 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.

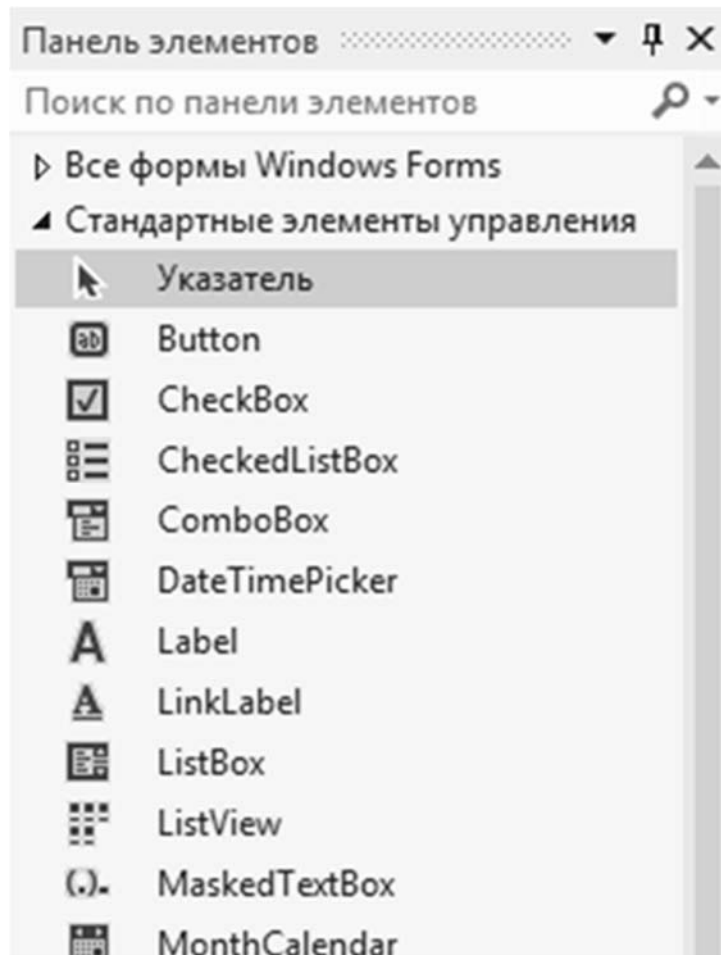


Рисунок 1.2 – Панель элементов VB.NET

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 Вы можете 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. Большинство 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.

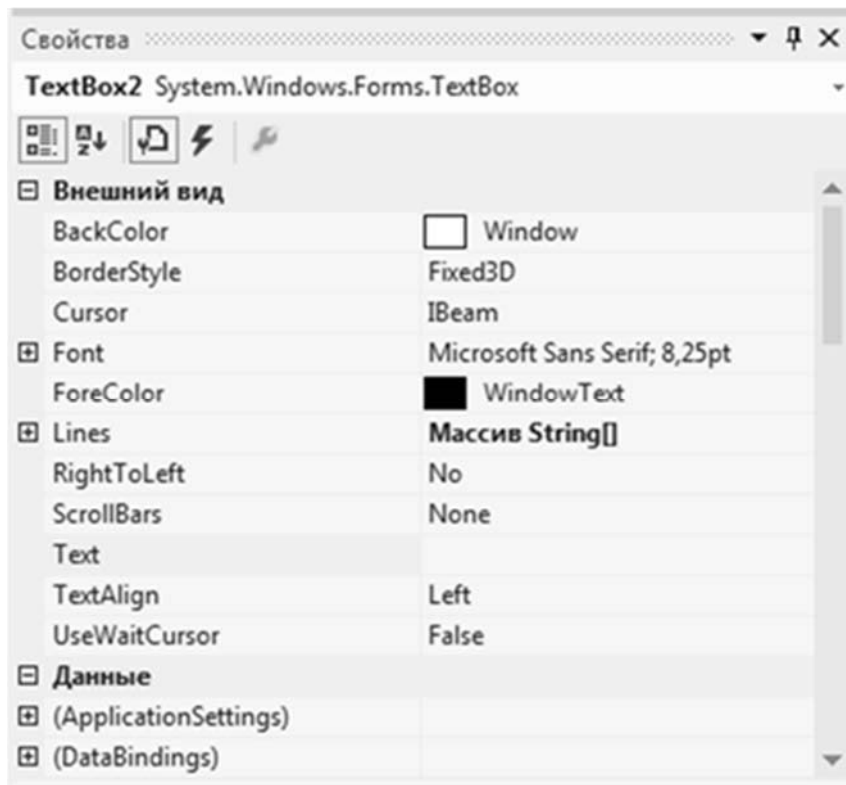


Рисунок 1.3 – Панель свойств VB.NET

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.

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 выполнять 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 Вы можете видеть, что секции `Map file` и `Link buffer file` состоят из набо-

ров RadioButton.

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

Контрольные вопросы

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

2 Основы языка VB.NET

2.1 Алфавит языка

Основными символами языка VB.NET являются:

- символы «+», «-», «*», «^»;
- 26 больших и 26 малых латинских букв A, B, ..., Y, Z, a, b, ..., y, z;
- 33 больших и 33 малых русских букв А, Б, ..., Ю, Я, а, б, ..., ю, я;
- 10 арабских цифр 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;
- специальные символы «*», «/», «=», «<», «>», «()», «[]», «{ }», «.», «,», «:», «;», «'», «#», «\$», «@».

Нет различий при использовании больших и малых букв в записи имен переменных, процедур, функций и меток. Их максимальная длина ограничена 126 символами.

2.2 Лексическая структура языка

В VB.NET различают следующие основные классы лексем.

1 Резервированные (служебные) слова. Этот класс состоит из слов, построенных только с помощью букв алфавита. Служебные слова можно использовать только по прямому назначению, т. е. так, как их назначение определил разработчик языка. Ни в каком другом виде, например в качестве имен переменных, их использовать нельзя.

Например: and, do, end, for, if, interface, not, sub, then, until, while, const и т. д.

2 Идентификаторы или имена предназначены для обозначения констант, переменных, типов, процедур, функций, меток. Они формируются из букв, цифр и символа «_» (подчеркивание). Длина имени может быть произвольной, однако компилятор учитывает имена по его первым 63 символам. Внутри имени не должно быть пробелов.

VB.NET в именах не различает больших и малых букв. Так, следующие имена будут идентичны:

SaveToFile, SAVETOFILE, savetofile, sAVeOfILE.

Тип константы определяется автоматически по виду ее значения.

Переменная отличается от константы или значения тем, что в процессе работы программы она может менять свое значение.

Переменная – это ячейка, в которой в каждый момент времени хранится одно значение или не хранится ничего. Переменная в любой момент времени может быть изменена программой. Всякая переменная, как правило, должна быть описана, т. е. должен быть явно указан ее тип.

Для объявления переменных мы используем ключевое слово Dim, либо один из модификаторов доступа: Private, Friend, Protected, Public или Static. Рассмотрим оператор Dim. После него требуется указать имя переменной. Требования к имени переменной следующие:

- имя должно содержать не более 255 символов;
- имя может содержать любые буквы, цифры и символ подчеркивания, при этом первый символ в имени должен быть буквой или символом подчеркивания;
- в имени должны отсутствовать пробелы и знаки пунктуации;
- имя не должно быть ключевым словом.

После имени переменной указывается ключевое слово As и затем идет тип данных переменной. После указания типа данных мы можем указать также и значение переменной с помощью знака «=».

Примеры объявления переменных:

```
Dim x As Integer
Dim y As Double = 4.0
Dim s As String = "Hello"
```

Это были примеры явного указания переменных, т. е. мы явно указывали их тип данных. Однако возможен и другой способ, когда компилятор выводит тип данных из ее значения, т. е. неявный способ определения переменных. Поэтому уместны и следующие объявления:

```
Dim x = 6
Dim y = 4.0
Dim s = "Hello"
```

Private – аналогично ключевому слову Dim, использование Private вместо Dim актуально в том случае, если требуется, что бы данное ключевое слово напоминало о том, что переменная доступна только в пределах данной процедуры или функции.

Public – позволяет сделать переменную VBA доступной из любой части проекта, однако это будет актуальным лишь в том случае, если объявление переменных в VBA происходит в разделе Declarations (Объявления) самого модуля. Если же прописать Public вместо Dim в теле процедуры, то видимость все равно будет доступна только в данной процедуре.

Static – данное ключевое слово актуально использовать вместо Dim тогда, когда нужно сохранять значение переменной в теле процедуры.

Все данные, используемые в Visual Basic.NET, описываются целой системой типов данных. Эта система определяет следующие примитивные типы данных:

- Boolean: представляет логическое значение. Может принимать True или False. Представлен системным типом System.Boolean;

- Integer: представляет целое число от $-2\,147\,483\,648$ до $2\,147\,483\,647$ и занимает 4 байта. Представлен системным типом System.Int32;

- Long: представляет целое число от $-9\,223\,372\,036\,854\,775\,808$ до $9\,223\,372\,036\,854\,775\,807$ и занимает 8 байт. Представлен системным типом System.Int64;

- Double: представляет числа с плавающей запятой двойной точности. Может принимать следующие значения: для отрицательных чисел от $-1,79769313486231570E+308$ до $-4,94065645841246544E-324$; для положительных чисел от $4,94065645841246544E-324$ до $1,79769313486231570E+308$. В памяти занимает 8 байт. Представлен системным типом System.Double;

- Decimal: хранит десятичное число с фиксированной запятой. Если употребляется без десятичной запятой, имеет значение от 0 до $\pm 79\,228\,162\,514\,264\,337\,593\,543\,950\,335$; если с запятой, то от 0 до $\pm 7,9228162514264337593543950335$ с 28 разрядами после запятой и занимает 16 байт. Представлен системным типом System.Decimal;

- Date: представляет дату от 0:00:00 1 января 0001 года до 23:59:59 31 декабря 9999 года и занимает 8 байт. Представлен системным типом System.DateTime;

- Char: хранит одиночный символ в кодировке Unicode и занимает 2 байта. Представлен системным типом System.Char;

- String: хранит набор символов Unicode. Представлен системным типом System.String;

- Object: может хранить значение любого типа данных и занимает 4 байта на 32-разрядной платформе и 8 байт на 64-разрядной платформе. Представлен системным типом System.Object, который является базовым для всех других типов и классов VB.NET.

Преобразования базовых типов данных.

В языке VB.NET применяются следующие методы для преобразований типов:

- метод CBool преобразует в тип Boolean любой числовой тип (включая Byte, SByte и типы перечисления), а также String, Object;

- метод CChar преобразует в Char объекты String, Object;

- метод CDate преобразует в Date объекты String, Object;

- метод CDb1 преобразует в Double любой числовой тип (включая Byte, SByte и типы перечисления), а также Boolean, String, Object;

- метод CDec преобразует в Decimal любой числовой тип (включая Byte, SByte и типы перечисления), а также Boolean, String, Object;
- метод CInt преобразует в Integer любой числовой тип (включая Byte, SByte и типы перечисления), а также Boolean, String, Object;
- метод CObj преобразует в Object любой тип данных;
- метод CStr преобразует в String любой числовой тип (включая Byte, SByte и типы перечисления), а также Boolean, Char, массив Char, Date, Object.

2.4 Операции языка Visual Basic.NET

Visual Basic.NET поддерживает большинство стандартных операций, принятых и в других языках программирования. Рассмотрим все виды операций.

Арифметические операции:

- «+» сложение двух чисел;
- «-» вычитание двух чисел;
- «*» умножение;
- «^» возведение в степень;
- «/» обычное деление;
- «\» целочисленное деление двух чисел;
- «Mod» получение остатка от деления двух чисел.

Примеры:

```
Dim x1 As Integer = 6 + 7 'Результат равен 13;
Dim x2 As Integer = 6 - 7 'Результат равен -1;
Dim x3 As Integer = 6 * 7 'Результат равен 42;
Dim x4 As Integer = 12 / 6 'Результат равен 2;
Dim x5 As Integer = 13 \ 6 'Результат равен 2, а остаток отбрасывается;
Dim x6 As Integer = 13 Mod 6 'Результат (он же остаток от деления 13 на 6)
равен 1;
Dim x7 As Integer = 6 ^ 2 'Результат равен 36.
```

2.5 Операции сравнения и логические операции

В Visual Basic.NET также имеются операторы сравнения:

- больше;
- >= больше или равно;
- < меньше;
- <= меньше или равно;
- = равно (в данном случае используется как знак сравнения на равенство двух значений);
- <> не равно.

Также в языке определены логические операторы, которые возвращают в

отличие от арифметических операций значение типа Boolean:

- And Логическое умножение (логическое И);
- Or Логическое сложение (логическое ИЛИ);
- Not отрицание.

Оператор And возвращает результат True только в том случае, если все выражения истинны. В остальных случаях он возвращает False. Оператор Or возвращает True, если хотя бы одно из выражений истинно. Он возвращает False тогда, когда неверны все выражения. Оператор Xor возвращает True, если одно из выражений истинно, а другое ложно. В остальных случаях, если оба выражения либо истинны, либо ложны, возвращается False. Оператор Not возвращает True, если выражение ложно, и False, если выражение истинно:

Dim x As Boolean = 6 > 2 And 2 < 4 'Результат True, т. к. и первое выражение и второе выражение истинны;

Dim y As Boolean = 6 > 2 And 2 > 4 'Результат False, т. к. только одно выражение истинно;

Dim x1 As Boolean = 6 > 2 Or 2 < 4 'Результат True, т. к. хотя бы одно выражение истинно;

Dim y1 As Boolean = 6 > 2 Or 2 > 4 'Результат True, т. к. опять же одно выражение истинно;

Dim y2 As Boolean = 6 < 2 Or 2 > 4 'Результат False, т. к. оба выражения ложны;

Dim x3 As Boolean = Not 2 < 4 'Результат False, т. к. выражение истинно;

Dim y3 As Boolean = Not 2 > 4 'Результат True, т. к. выражение ложно.

2.6 Массивы

Массив представляет собой набор данных одного типа. Например, объявим массив элементов типа Integer:

```
Dim nums(5) As Integer
nums(0) = 0
nums(1) = 1
nums(2) = 2
nums(3) = 3
```

Здесь объявлен массив из шести элементов типа Integer. По умолчанию всем шести элементам в массиве присваивается 0. Затем первым четверем элементам массива присваиваются некоторые значения. Обратите внимание, что индексация в массиве начинается с нуля. При этом нельзя выйти за рамки установленной длины массива в шесть элементов.

Следующий код вызовет исключение `ArrayIndexOutOfRangeException`, поскольку восьмого элемента в массиве не существует, в нем определено только шесть элементов:

```
Dim nums(5) As Integer
nums(7) = 7
```

В вышеприведенном примере неявно инициализировались члены массива, с размером в шесть элементов.

Но необязательно присваивать все значения массива после объявления. Можно все сделать уже при объявлении массива:

```
Dim nums2 As Integer() = New Integer(5) {0, 1, 2, 3, 4, 5}
```

Однако можно объявить массив, а количество элементов указать потом:

```
Dim nums() As Integer
```

В таком случае нам его еще предстоит инициализировать. Мы это можем сделать так:

```
Dim nums() As Integer
Dim mass() As Integer
nums = New Integer(5) {}
nums(0) = 0
nums(1) = 1
nums(2) = 2
nums(3) = 3
Dim mass() As Integer
ReDim mass(8)
```

Кроме размера массив характеризуется таким понятием как размерность (dimension). В предыдущих примерах мы использовали одномерные массивы. Но массивы бывают и многомерными. Например:

```
'Одномерный массив
Dim num1 As Integer(5)
Dim nums1 As Integer() = {0, 1, 2, 3, 4, 5}
'Двухмерный массив
Dim num2 As Integer(2,2)
Dim nums2 As Integer(,) = {{0, 1, 2}, {3, 4, 5}}
```

Здесь мы создали двухмерный массив, который можно представить в виде таблицы:

```
Одномерный массив nums1
0    1    2    3    4    5
Двухмерный массив nums2
```

0	1	2
3	4	5

2.7 Условные конструкции

В VB.NET имеются следующие условные структуры:

- If ... Then;
- Select Case.

Конструкция If ... Then проверяет истинность некоторого условия и в 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
```

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 из простых типов данных (Boolean, Byte, Char, Date, Double, Decimal, Integer, Long, Object, SByte, Short, Single, String, UInteger, ULong и UShort).

2.8 Циклы

Еще одним видом управляющих конструкций являются циклы. В VB.NET используется несколько видов циклов:

- For...Next;
- Do.

Цикл For...Next.

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

```

For i As Integer = 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 As Integer = 1 To -9 Step -1
    For j As Integer = 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 обрабатывает пять раз, а внешний девять, т. е. фактически получается 45 циклов.

Цикл Do.

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`.

3 Разработка программ на ПЭВМ

3.1 Процесс разработки новых программ для ЭВМ

Процесс разработки новых программ для ЭВМ включает в себя:

- 1) постановку задачи;
- 2) создание алгоритма ее решения;
- 3) реализацию алгоритма на ЭВМ в виде программы;
- 4) отладку программы.

Постановка задачи состоит в четком формулировании целей работы. Необходимо четко определить, что является исходными данными, а что требуется получить в качестве результата, каким должен быть интерфейс программы.

Алгоритм – описание последовательности операций, которые нужно выполнить для решения задачи.

Следующим шагом после создания алгоритма является написание реализующей его **программы**. Выполняемая программа представляет собой набор двоичных кодов – нулей и единиц. На языках высокого уровня она реализуется в виде текста. Перевести текст в набор цифр чрезвычайно сложно, в связи с этим в данный процесс вводится промежуточный этап – **разработка текста программы**.

К языкам высокого уровня относятся: фортран, бейсик, паскаль, си, алгол, алмир, ада, си++, delphi, java и сотни других. К языкам низкого уровня относятся ассемблер и автокод. Ассемблер, как язык низкого уровня, фактически состоит из набора команд данной машины, записанных в виде сокращений на английском языке. Автокод – вариант ассемблера на основе русского языка.

Программы трансляторы бывают двух типов:

- 1) **Интерпретаторы** транслируют текст программы и сразу же выполняют предписанные в нем действия, не создавая .exe-файл;
- 2) **Компиляторы** транслируют текст программы и создают готовую к исполнению программу в виде .exe-файла, который можно будет после запустить на исполнение.

Отладка программы – исправление в ней ошибок и тщательное ее тестирование.

При тестировании программы важно проверить ее работоспособность как можно в большем числе ситуаций, например при различных вариантах исходных данных. Бывает, что в 1000 случаях программа сработает нормально, а на 1001-й раз обнаружится ошибка. При написании серьезных программных продуктов для более полного их тестирования фирмы-разработчики часто распространяют их пробные версии (альфа- или бета-версии) среди как можно большего числа пользователей, которые сообщают в фирму об обнаруженных ошибках, что позволяет исправить их в окончательных версиях программных продуктов.

3.2 Понятие алгоритма и способы его описания

Алгоритм — точное описание последовательности действий, которые необходимо выполнить для решения задачи.

Название «**алгоритм**» произошло от латинской формы имени величайшего среднеазиатского математика **Мухаммеда ибн Муса ал-Хорезми** (Alhorithmi), жившего в 783–850 гг. В своей книге «Об индийском счете» он изложил правила записи натуральных чисел с помощью арабских цифр и правила действий над ними «столбиком».

Описание алгоритма:

- 1) текстовая инструкция;
- 2) графическая схема;
- 3) программа на одном из языков программирования.

Каждый алгоритм должен иметь:

- 1) название, отражающее суть решаемой задачи;
- 2) описание исходной информации;
- 3) описание последовательности действий;
- 4) описание выходной информации.

Свойства алгоритма.

Понятность для исполнителя – исполнитель алгоритма должен понимать, как его выполнять. Иными словами, имея алгоритм и произвольный вариант исходных данных, исполнитель должен знать, как надо действовать для выполнения этого алгоритма.

Дискретность (прерывность, раздельность) – алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определенных) шагов. Каждое действие, предусмотренное алгоритмом, исполняется только после того, как закончилось исполнение предыдущего.

Определенность (однозначность) – правила и порядок выполнения операций алгоритма не допускают неоднозначного толкования. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.

Результативность (конечность) – алгоритм должен приводить к решению задачи за конечное число шагов.

Массовость – алгоритм решения задачи разрабатывается в общем виде, т. е. он должен быть применим для некоторого класса задач, различающихся только исходными данными. При этом исходные данные могут выбираться из некоторой области, которая называется областью применимости алгоритма.

Виды алгоритмов как логико-математических средств отражают указанные компоненты человеческой деятельности и тенденции, а сами алгоритмы в зависимости от цели, начальных условий задачи, путей ее решения, определения действий исполнителя подразделяются следующим образом.

1 Механические алгоритмы, или иначе детерминированные, жесткие (например, алгоритм работы машины, двигателя и т. п.). Механический алгоритм задает определенные действия, обозначая их в единственной и досто-

верной последовательности, обеспечивая тем самым однозначный требуемый или искомый результат, если выполняются те условия процесса, задачи, для которых разработан алгоритм.

2 Гибкие алгоритмы, например стохастические, т. е. вероятностные и эвристические.

3 Вероятностный (стохастический) алгоритм дает программу решения задачи несколькими путями или способами, приводящими к вероятному достижению результата.

4 Эвристический алгоритм (от греческого слова «эврика») – это такой алгоритм, в котором достижение конечного результата программы действий однозначно не predetermined, так же как не обозначена вся последовательность действий, не выявлены все действия исполнителя. К эвристическим алгоритмам относят, например, инструкции и предписания. В этих алгоритмах используются универсальные логические процедуры и способы принятия решений, основанные на аналогиях, ассоциациях и прошлом опыте решения схожих задач.

3.3 Методы изображения алгоритмов

На практике наиболее распространены следующие формы представления алгоритмов:

- словесная (записи на естественном языке);
- псевдокоды (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
- графическая (изображения из графических символов);
- программная (тексты на языках программирования).

Словесное описание алгоритма.

Рассмотрим пример на алгоритме нахождения максимального из двух значений:

- определим форматы переменных X , Y , M , где X и Y – значения для сравнения, M – переменная для хранения максимального значения;
- получим два значения чисел X и Y для сравнения;
- сравним X и Y ;
- если X меньше Y , значит большее число Y ;
- поместим в переменную M значение Y ;
- если X не меньше (больше) Y , значит большее число X ;
- поместим в переменную M значение X .

Словесный способ не имеет широкого распространения по следующим причинам:

- такие описания строго не формализуемы;

- страдают многословностью записей;
- допускают неоднозначность толкования отдельных предписаний.

3.4 Понятие алгоритмического языка

Достаточно распространенным способом представления алгоритма является его запись на алгоритмическом языке, представляющем в общем случае систему обозначений и правил для единообразной и точной записи алгоритмов и исполнения их. Отметим, что между понятиями «алгоритмический язык» и «языки программирования» есть различие; прежде всего, под исполнителем в алгоритмическом языке может подразумеваться не только компьютер, но и устройство для работы «в обстановке». Программа, записанная на алгоритмическом языке, не обязательно предназначена компьютеру. Практическая же реализация алгоритмического языка – отдельный вопрос в каждом конкретном случае.

Как и каждый язык, алгоритмический язык имеет свой словарь. Основу этого словаря составляют слова, употребляемые для записи команд, входящих в систему команд исполнителя того или иного алгоритма. Такие команды называют простыми командами. В алгоритмическом языке используют слова, смысл и способ употребления которых задан раз и навсегда. Эти слова называют служебными. Использование служебных слов делает запись алгоритма более наглядной, а форму представления различных алгоритмов – единообразной.

АЛГ – название алгоритма;
НАЧ – серия команд алгоритма;
ЕСЛИ – начало условия;
ТО – серия команд 1;
ИНАЧЕ – серия команд 2;
ВСЕ – конец условия;
ПОКА – начало условного цикла;
НЦ, КЦ – серия команд;
КОН – конец условного цикла.

Алгоритмы представляются в виде:




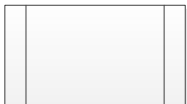


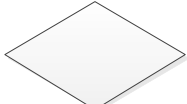
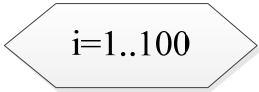
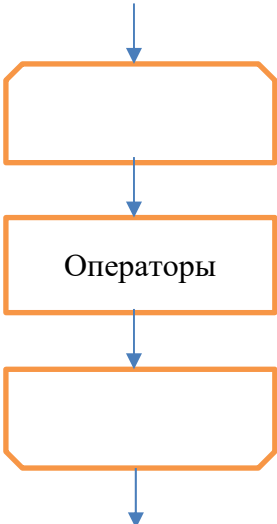
- звуковой информации (наставления, устная инструкция);
- текстовой информации (инструкции);
- графической информации (блок-схемы, фото и т. д.).

Для проектирования ПО применяются блок-схемы.

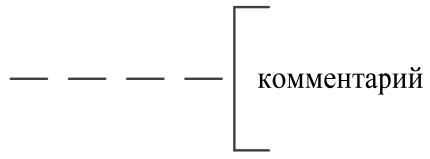


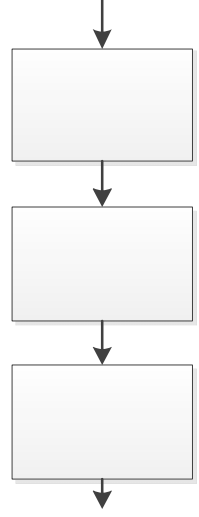
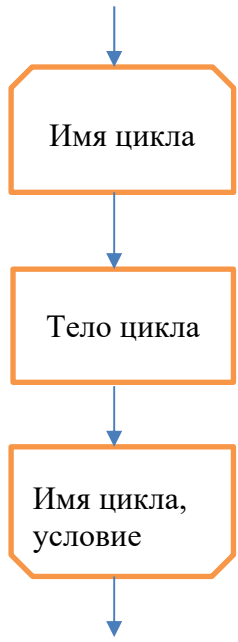
Элементы блок-схем стандартизованы и описаны в ГОСТ 19.701–90 *Схемы алгоритмов, программ данных и систем*.

Типовые действия, наиболее часто встречающиеся в схемах программ, представлены в таблице 3.1.

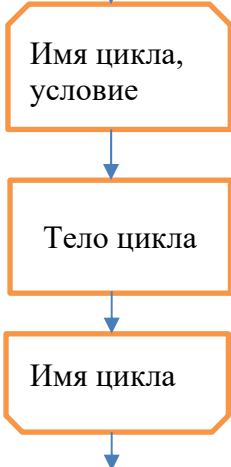
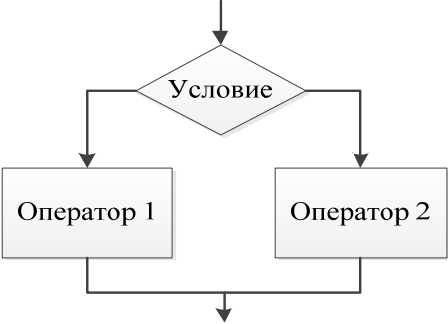
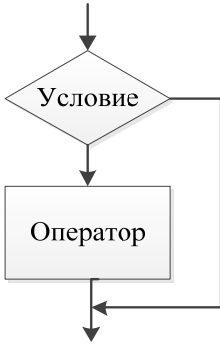
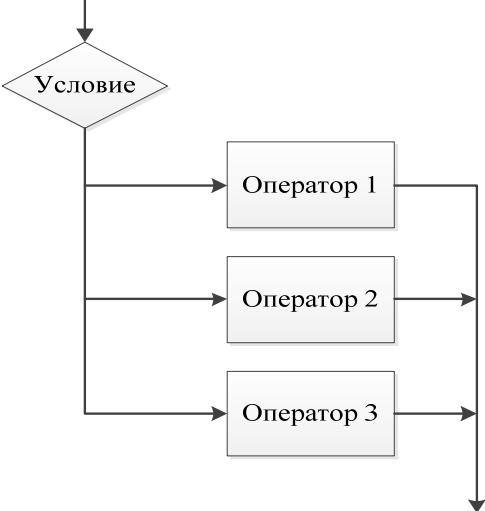
Таблица 3.1 – Обозначения элементов в соответствии с ГОСТ 19.701–90

Описание	Обозначение
Вычисление, функция обработки данных любого вида (символ «процесс»)	
Данные. Символ отображает данные, носитель данных не определен	
Документ. Символ отображает данные, представленные на носителе в удобочитаемой форме (машинограмма, документ для оптического или магнитного считывания, микрофильм, рулон ленты с итоговыми данными, бланки ввода данных)	
Вызов функции (символ «предопределенный процесс»)	
Начало или конец программы (символ «терминатор»)	
Дисплей. Символ отображает данные, представленные в форме, понятной для человека на носителе в виде отображающего устройства (экран для визуального наблюдения, индикаторы ввода информации)	
Проверка условия – функция переключательного типа, имеющая один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определённых внутри этого символа (символ «решение»)	
Символ «подготовка данных» в произвольной форме (в ГОСТ нет ни пояснений, ни примеров) задает входные значения. Используется обычно для задания циклов со счетчиком	
Вычисление в цикле (символ «граница цикла»). Символ состоит из двух частей, между которыми находится тело цикла. Условия для инициализации, приращения, завершения и т. д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие	

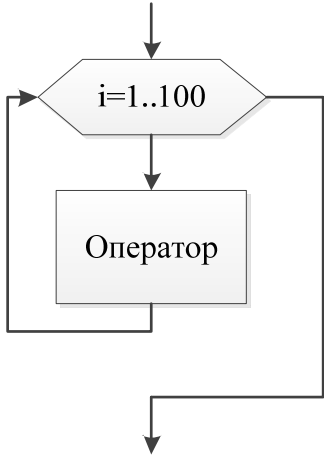
Продолжение таблицы 3.1

Описание	Обозначение
Комментарии к алгоритму (символ «комментарий»). Символ используют для добавления описательных комментариев или пояснительных записей в целях объяснения или примечания. Пунктирные линии связаны с соответствующим символом или могут обводить группу символов	
Обрыв линии, выход в часть схемы и вход из другой части этой схемы (символ «соединитель»)	
Поток данных (символ «линия»). Символ отображает поток данных. При необходимости или повышения удобочитаемости могут быть добавлены стрелки-указатели	
Следование – последовательное размещение символов (блоков) или группы символов	
Цикл с постусловием – цикл «do – loop». Цикл всегда выполняется хотя бы один раз. Проверка условия выхода из цикла происходит после того, как тело цикла выполнено. (Тело цикла – та последовательность действий которая выполняется многократно.)	

Продолжение таблицы 3.1

Описание	Обозначение
<p>Цикл с предусловием – цикл «do – loop». Проверка условия производится перед выполнением тела цикла. Если при первой проверке условие выхода выполняется, то само тело цикла не будет выполнено ни разу. Тело цикла выполняется до тех пор, пока не будет выполнено условие выхода из цикла</p>	
<p>Разветвление. Применяется, когда в зависимости от условия выполняется либо одно, либо другое действие. Действия могут содержать несколько этапов (символов)</p>	
<p>Обход – частный случай разветвления, когда одна ветвь не содержит никаких действий</p>	
<p>Множественный выбор – результат обобщения разветвления, когда в зависимости от значения некоторой переменной выполняется одно из нескольких действий</p>	

Окончание таблицы 3.1

Описание	Обозначение
Цикл с конечным числом повторов	

3.5 Методы программирования

При разработке модульных программ применяются два метода проектирования – нисходящее и восходящее.

При нисходящем проектировании разработка программного комплекса идет сверху вниз.

Метод нисходящего проектирования предполагает последовательное разложение общей функции обработки данных на простые функциональные элементы («сверху вниз»).

В результате строится иерархическая схема, отражающая состав и взаимоподчиненность отдельных функций, которая носит название функциональная структура алгоритма (ФСА) приложения.

Последовательность действий по разработке функциональной структуры алгоритма приложения:

- определяются цели автоматизации предметной области и их иерархия (цель-подцель);
- устанавливается состав приложений (задач обработки), обеспечивающих реализацию поставленных целей;
- уточняется характер взаимосвязи приложений и их основные характеристики (информация для решения задач, время и периодичность решения, условия выполнения и др.);
- определяются необходимые для решения задач функции обработки данных;
- выполняется декомпозиция функций обработки до необходимой структурной сложности, реализуемой предполагаемым инструментарием.

На первом этапе разработки кодируется, тестируется и отлаживается головной модуль, который отвечает за логику работы всего программного комплекса.

Остальные модули заменяются заглушками, имитирующими работу этих

модулей. Применение заглушек необходимо для того, чтобы на самом раннем этапе проектирования можно было проверить работоспособность головного модуля. На последних этапах проектирования все заглушки постепенно заменяются рабочими модулями.

При восходящем проектировании разработка идет снизу вверх. На первом этапе разрабатываются модули самого низкого уровня. На следующем этапе к ним подключаются модули более высокого уровня и проверяется их работоспособность. На завершающем этапе проектирования разрабатывается головной модуль, отвечающий за логику работы всего программного комплекса. Методы нисходящего и восходящего программирования имеют свои преимущества и недостатки.

Недостатки нисходящего проектирования:

- необходимость заглушек;
- до самого последнего этапа проектирования неясен размер программного комплекса и его эксплуатационные характеристики, за которые, как правило, отвечают модули самого низкого уровня.

Преимущество нисходящего проектирования – на самом начальном этапе проектирования отлаживается головной модуль (логика программы).

Преимущество восходящего программирования – не нужно писать заглушки.

Недостаток восходящего программирования – головной модуль разрабатывается на завершающем этапе проектирования, что порой приводит к необходимости дорабатывать модули более низких уровней.

На практике применяются оба метода. Метод нисходящего проектирования чаще всего применяется при разработке нового программного комплекса.

3.6 Модульное программирование

Модульное программирование основано на понятии модуля – логически взаимосвязанной совокупности функциональных элементов, оформленных в виде отдельных программных модулей, т. е. программа разбивается на относительно независимые составные части – программные модули. При этом каждый модуль может разрабатываться, программироваться, транслироваться и тестироваться независимо от других. Внутреннее строение модуля для функционирования всей программы, как правило, значения не имеет.

При модификации алгоритма, реализуемого модулем, структура программы не должна меняться.

Цели модульного программирования.

- 1 Улучшать читабельность программ.
- 2 Повышать эффективность и надежность программ (легко находить и корректировать ошибки).
- 3 Уменьшать время и стоимость программной разработки (уменьшается время отладки).

Модуль характеризуют:

- один вход и один выход – на входе программный модуль получает определенный набор исходных данных, выполняет обработку и возвращает один набор результатных данных;

- функциональная завершенность – модуль выполняет перечень регламентированных операций для реализации каждой отдельной функции;

- логическая независимость – результат работы программного модуля зависит только от исходных данных, но не зависит от работы других модулей, код модуля физически и логически отделен от кода других модулей, обмен информацией между модулями должен быть по возможности минимизирован.

Каждый модуль состоит из спецификации и тела. Спецификации определяют правила использования модуля, а тело – способ реализации процесса обработки. Следует избегать использования меток и оператора GOTO. Использовать только стандартные управляющие конструкции (условие, выбор, цикл, блок).

Принципы модульного программирования программных продуктов во многом сходны с принципами нисходящего проектирования. Сначала определяются состав и подчиненность функций, а затем – набор программных модулей, реализующих эти функции.

Однотипные функции реализуются одними и теми же модулями. Функция верхнего уровня обеспечивается главным модулем; он управляет выполнением нижестоящих функций, которым соответствуют подчиненные модули.

При определении набора модулей, реализующих функции конкретного алгоритма, необходимо учитывать следующее:

- каждый модуль вызывается на выполнение вышестоящим модулем и, закончив работу, возвращает управление вызвавшему его модулю;

- принятие основных решений в алгоритме выносится на максимально «высокий» по иерархии уровень;

- для использования одной и той же функции в разных местах алгоритма создается один модуль, который вызывается на выполнение по мере необходимости.

Модульность ведет к повышению производительности при создании проекта. Прежде всего, маленькие модули могут быть закодированы быстро и легко. Во-вторых, универсальные модули могут многократно использоваться, что приводит к более быстрому построению последующих программ. В-третьих, модули программы могут быть проверены независимо, что помогает уменьшить время, потраченное на отладку.

3.7 Функциональное программирование

Функциональное программирование – стиль программирования, в котором процесс вычисления трактуется как вычисление значений функций в математическом понимании (т. е. тех, чей единственный результат работы заключается в возвращаемом значении).

Противопоставляется стилю структурного программирования, в которой исполнителю программы *предписывается* последовательность выполняемых действий в то время, как в функциональном программировании способ решения задачи *описывается* при помощи зависимости функций друг от друга (в том числе возможны рекурсивные зависимости), но без указания последовательности шагов.

Функциональное программирование называется так, потому что программа полностью состоит из функций. Сама программа тоже является функцией, которая получает исходные данные в качестве аргумента, а выходные данные выдаёт как результат. Как правило, основная функция определена в терминах других функций, которые, в свою очередь, определены в терминах еще большего количества функций, вплоть до функций – примитивов языка на самом нижнем уровне.

Особенности и преимущества функционального программирования:

- функциональные программы не содержат операторов присваивания, а переменные, получив однажды значение, никогда не изменяются;
- обращение к функции не вызывает иного эффекта кроме вычисления результата, это устраняет главный источник ошибок, и делает порядок выполнения функций несущественным; т. к. побочные эффекты не могут изменять значение выражения, оно может быть вычислено в любое время.

3.8 Процедурное программирование

Процедурное программирование – это парадигма программирования, основанная на концепции вызова процедуры. Процедуры также известны как подпрограммы, методы или функции (это функции, подобные тем, которые используются в функциональном программировании). Процедуры просто содержат последовательность шагов для выполнения. В ходе выполнения программы любая процедура может быть вызвана из любой точки, включая саму данную процедуру.

Процедурное программирование – это лучший выбор, чем просто последовательное или неструктурированное программирование во многих ситуациях, которые требуют значительного упрощения поддержки.

Возможные выгоды:

- возможность повторного использования одного и того же кода из нескольких мест программы без его копирования;
- легче отследить поток выполнения программы, чем в случае использования инструкций GOTO или JUMP, которые могут сделать из большой, сложной программы так называемый «спагетти-код»;
- возможность поддержки модульности и структурности.

3.9 Структурное программирование

Структурное программирование основано на модульной структуре программного продукта и типовых управляющих структурах алгоритмов обработки данных различных программных модулей. Наиболее важное различие между структурным и неструктурным подходом в том, что структурные программы, разработаны модульным способом.

В любой типовой структуре блок, кроме условного, имеет только один вход и выход.

Логическая структура программы может быть выражена комбинацией трех базовых структур: следование, ветвление и цикл. Эти структуры могут комбинироваться одна с другой, как того требует программа. Используя эти структуры, можно писать программы без операторов GOTO. Но структурное программирование – это не просто программирование без GOTO. Это дисциплина программирования, которая объединяет несколько способов создания ясной, легкой для понимания программы. Вполне возможно писать структурированные программы, содержащие оператор GOTO, равно как и неструктурированные, не содержащие ни одного GOTO.

3.10 Объектно-ориентированное программирование

Объектно-ориентированное программирование – парадигма программирования, в которой предметная область представляется системой структур данных, каждая из которых представляет некий отдельный предмет (объект) с его внутренними свойствами и действиями над ним.

Первым объектно-ориентированным языком программирования была Симула, воплотившая главные черты того, что Алан Кэй, один из авторов языка Smalltalk, впоследствии назвал объектно-ориентированным программированием.

4 Программы, подпрограммы, программные модули

Подпрограмма – это законченная алгоритмическая единица, которая предназначена для выполнения некоторого ограниченного по отношению к использующей ее программе круга операций.

В языке VB.NET есть два вида подпрограмм – процедуры и функции. Структура всякой подпрограммы во многом напоминает структуру исходного модуля. Каждая такая подпрограмма перед ее использованием должна быть описана. Описанием подпрограммы называется ее исходный код, а обращением к подпрограмме является оператор или его часть, которые содержат код вызова такой подпрограммы. Таким образом, описание – это технология, а обращение – это действия по предписанной технологии.

Всякая подпрограмма может иметь локальные и глобальные по отношению

к ней параметры. Локальным является параметр, действие которого ограничено только подпрограммой, в которой он описан. Всякий другой параметр будет глобальным. Все глобальные параметры всегда описаны за пределами подпрограммы, в которой они используются.

4.1 Процедуры

Всякая процедура имеет заголовок и тело. Тело процедуры состоит из операторов, предназначенных для описания имен и действий над данными.

Синтаксис процедуры имеет вид:

```
Sub procedureName(параметры)
Локальные_переменные
Операторы
End Sub
```

Здесь `procedureName` – имя процедуры. Именем процедуры может быть любое имя, не совпадающее ни с каким другим описанным в том же, что и процедура, блоке, именем.

Локальные_переменные – внутренние описания.

Операторы – операторы тела процедуры.

Параметры – список формальных параметров может быть либо пуст (в этом случае ставятся пустые скобки), либо должен содержать последовательность входных и/или выходных величин.

4.2 Функции

В отличие от процедуры функция предназначена для вычисления одного значения любого типа. Тип функции указывается в конце ее заголовка. Тип возвращаемого значения отделяется от списка формальных параметров символом «:» (двоеточие). Кроме того, в теле функции, по крайней мере, один раз должно быть определено значение функции. Это значение присваивается имени функции.

Синтаксис функции имеет вид:

```
function имяфункции(параметры) as возвращаемый_тип
Локальные_переменные
Операторы
end
```

Здесь `имяфункции` – имя функции; `параметры`, `Локальные_переменные`, `Операторы` имеют тот же смысл, что и в процедуре; `возвращаемый_тип` – тип возвращаемого результата

Пример

```

Function Fact(n as integer) as integer ‘заголовок функции Fact
Dim I as integer, j as integer
j=1
if (n > 1) then
For i= 2 to n do j= j * i
Fact:=j ‘или Return j
Next I
End If
End Function ‘конец описания функции Fact
... {текст головной программы}
Dim r as Double
...
r= 3.4 * Fact(3) / 2.5 / (122 - Fact(5)) ‘обращение к функции Fact

```

В этом примере описана функция с именем Fact вычисления факториала $n!$ неотрицательного целого числа. Тип функции определен как integer. В теле функции размещен оператор Fact:=j, который определяет возвращаемый функцией результат. Способ обращения к функции демонстрирует последний оператор примера. Видно, что способ обращения к функции имеет существенное отличие от способа обращения к процедуре. Так, в этом операторе обращение к функции Fact производится дважды – один раз с фактическим параметром 3, другой – с параметром 5. Далее возвращенные результаты (соответственно, 6 и 120) будут подставлены в выражение правой части оператора, после чего последний будет вычислен и переменная r получит вещественное (Double) значение 4.08.

4.3 Формальные и фактические параметры

В VB.NET есть понятия формального и фактического параметров. Формальным называется параметр, который содержится в заголовке описания подпрограммы, а фактическим – параметр в обращении к подпрограмме. Так, в вышеприведенном примере написания функции в п. 4.2 параметр n является формальным, а значение 3 и 5 – фактическими.

При вызове подпрограмм необходимо иметь в виду следующее: фактические значения или константы должны быть совместимы по типу с объявленными формальными параметрами.

5 Графические возможности VB.NET

5.1 Основы рисования в VB.NET

Для создания рисунков необходимо создать на форме объект PictureBox (место для картинки / полотно) и задать в свойствах белый цвет. В примере имя объекта PictureBox РВ.

Компьютерная система координат представлена на рисунке 5.1.

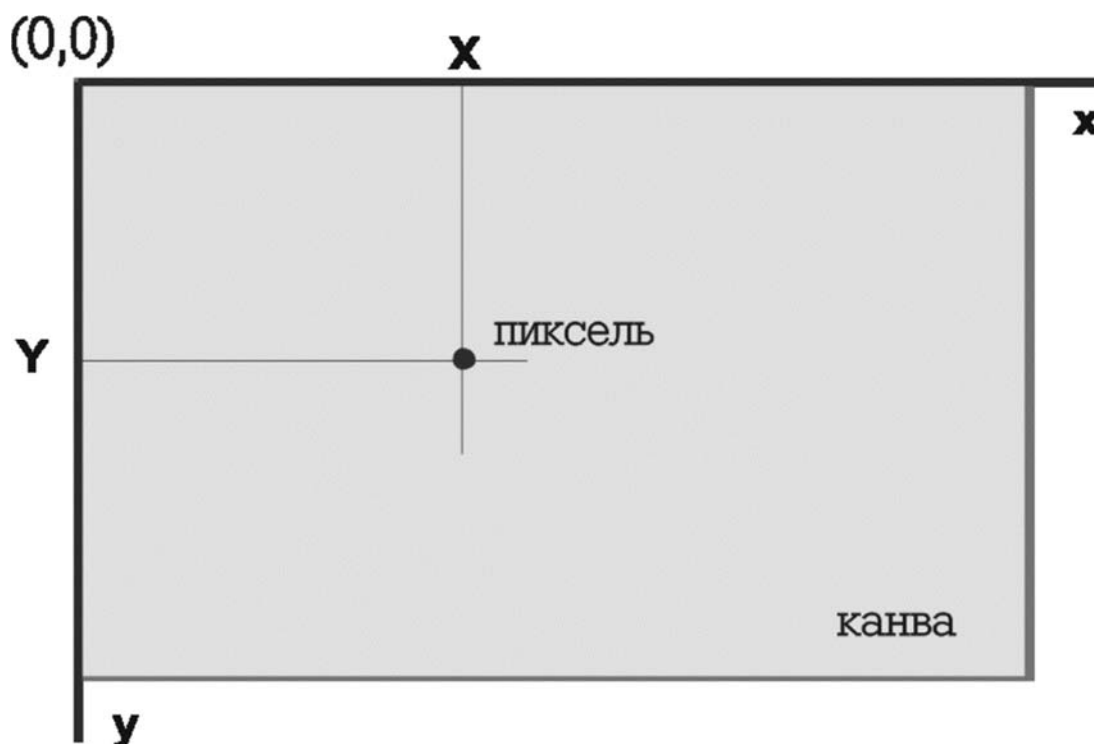


Рисунок 5.1 – Компьютерная система координат

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. Задаёт цвет линии:

Pens.Black – чёрный;

Pens.White – белый.

DrawEllipse(Pen, Rectangle). Рисует эллипс, определяемый ограничивающей структурой Rectangle (рисунок 5.2).

Pen. Задаёт цвет линии.

Rectangle. Задаёт 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

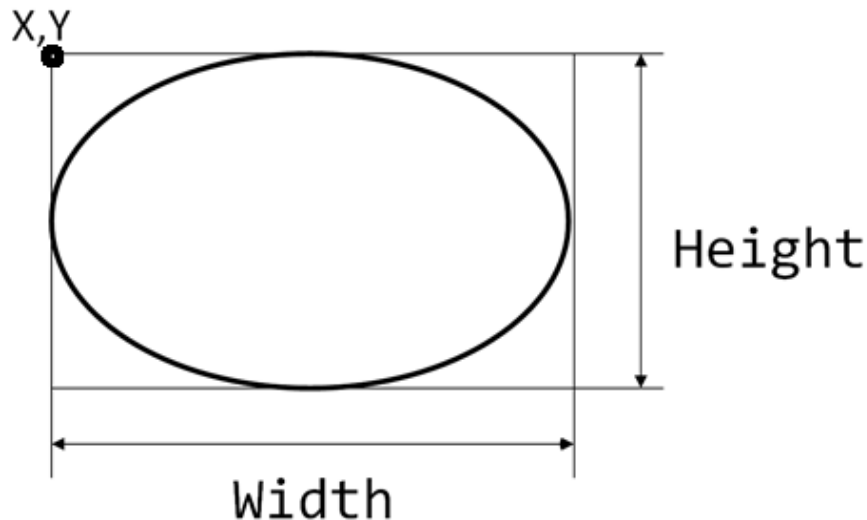


Рисунок 5.2 – Результат выполнения команды DrawEllipse

6 Объектно-ориентированное программирование

6.1 Классы и объекты

Инкапсуляция есть объединение в единое целое данных и алгоритмов обработки этих данных. В рамках объектно-ориентированного программирования (ООП) данные называются полями объекта, а алгоритмы – объектными методами.

Инкапсуляция позволяет в максимальной степени изолировать объект от внешнего окружения. Она существенно повышает надежность разрабатываемых программ, т. к. локализованные в объекте алгоритмы обмениваются с программой сравнительно небольшими объемами данных, причем количество и тип этих данных обычно тщательно контролируются. В результате замена или модификация алгоритмов и данных, инкапсулированных в объект, как правило, не влечет за собой плохо прослеживаемых последствий для программы в целом (в целях повышения защищенности программ в ООП почти не используются глобальные переменные).

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.

Создание экземпляра производится следующей командой:

```
Dim <переменная> As New <имя_класса>
```

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)
```

Наследование. 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.

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;

- 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.

Переопределение методов в производных классах. Полиморфизм.

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`.

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