## Тема 4.3. Структура VB-программ и процедуры.

## Средства программирования алгоритмов линейной структуры

[4.3.1.Структура простейших VB – программ](#_4.3.1._Структура_простейших)

[4.3.2. Создание и использование процедур пользователей](#_4.3.2._Создание_и)

[4.3.3. Передача аргументов по значению и по ссылке](#_4.3.3._Передача_параметров)

[4.3.4. Средства программирования алгоритмов линейной структуры и функции в/в данных](#_4.3.4._Средства_программирования)

[4.3.5. Перегрузка методов (процедур)](#_4.3.5.__Задачи)

[4.3.6. Задачи для самостоятельного решения по теме «Структура VB-программ и](#_4.3.6.__Задачи)

[процедуры. Средства программирования алгоритмов линейной структуры»](#_4.3.6.__Задачи)

[4.3.7. Тестовые задания по теме «Структура VB-программ и процедуры. Средства](#_4.3.7.__Тестовые)

[программирования алгоритмов линейной структуры»](#_4.3.7.__Тестовые)

### 4.3.1. Структура простейших VB-программ

При создании простых **Windows Приложений (Windows Application) *программа*** (***программный код проекта***), написанная на языке **VB,** может состоять из ***программного кода, генерируемого средой разработки, и программного кода, написанного или дописанного программистом.*** Причем базовая структура программы создается автоматически в тот момент, когда пользователь выбирает в окне **Создание проекта (New Project)** шаблон ***Windows Application* (Windows Приложение).**

Простые **Windows Приложения** могут содержать следующие составляющие:

* операторы Option, определяющие основные правила обработки программного кода, предупреждающие синтаксические и логические ошибки

Option Explicit,

Option Compare,

Option Strict;

* операторы Imports**…**, импортирующие в программу ***компоненты библиотек***, определяемые в импортируемом пространстве;
* модули форм

Public Class *Имя\_Формы*

**…**

End Class;

* модули классов

Public Class *Имя\_класса*

***…***

End Class;

* операторы описания типов данных с ключевыми словами Dim, Private и Public***,*** находящиеся внутри модулей или процедур;
* процедуры, генерируемые средой разработки внутри модулей формы;
* процедуры, написанные программистом, которые могут находиться как внутри модулей формы, так и внутри стандартных модулей (общие процедуры).

Программы **VB** хранятся в программных модулях в виде процедур. Эти программные модули могут быть двух видов:

* ***модуль формы;***
* ***модуль класса.***

***В начале обучения до темы 5.1. будут рассматриваться только* Windows Приложения*, которые состоят из одной формы, и содержат, как правило, только один модуль – модуль формы и имеют следующую структуру:***

Option Explicit On

Option Strict On

Imports System.Math

Imports…

Public Class Form1

…

End Class

В свою очередь модуль формы может содержать ***процедуры событий***, ***процедуры пользователей*** и ***операторы описания данных*** (простых переменных, массивов и констант) с ключевыми словами Dim, Private и Public.

Таким образом, программный код (доступный пользователю) простого приложения будет иметь структуру, приведенную на рис. 4.3.1-1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | Option Strict On  Option Strict On  Imports System.Math |   Private Sub Form1   |  | | --- | | Описание глобальных данных |  |  | | --- | | Процедура пользователя -1  (Общая процедура -1) |   **…**   |  | | --- | | Процедура пользователя - n  (Общая процедура – n) |  |  | | --- | | Процедура обработки событий - 1 |   **…**   |  | | --- | | Процедура обработки событий - m |   End Class |

Рис. 4.3.1-1

В дальнейшем, по мере усложнения приложения в нем могут создаваться модули классов, стандартные модули, а если программа достаточно большая, в ней будет несколько форм и большое число процедур событий, которые используют одни и те же переменные и процедуры. Причем некоторый программный код можно выделить в отдельный программный код, который будет являться общим для всех. Такой программный код может быть частью ***стандартного модуля***. Кроме того, ***стандартные модули***, так же, как и модули форм, могут содержать объявления переменных, констант и типов данных, описания общих процедур и процедур обработки событий.

По умолчанию переменные ***локальны*** для любой процедуры– они могут быть использованы и изменены только в той процедуре, в которой они ***созданы (объявлены)***. Также можно объявить переменные в верхней части программного кода формы, до описания процедур пользователей и процедур событий, и тем самым при­дать переменным более широкую ***область видимости***, распространяющу­юся на всю форму. Однако, при создании в проекте нескольких форм, переменные, объявленные в верхней части формы, будут доступны толь­ко в той форме, в которой они объявлены. Точно также, процедуры собы­тий по умолчанию объявляются как *частные* (Private***)*** и являются локальными для той формы, в которой они созданы. Например, нельзя из второй формы с именем *Form2* вызвать процедуру события Button1\_Click( ), если она объявлена в *Form1* как частная.

Чтобы сделать переменные и процедуры доступными для всех форм и процедур событий проекта, их следует объявить в одном или несколь­ких стандартных (общих) модулях, включенных в проект.

Изучение общих модулей и общих процедур пока отложим, а подробно изучим структуру и содержание модуля формы, а также правила написания процедур пользователя.

Объявление ***глобальной***, или ***открытой*** переменной в модуле формы достаточно просто – необходимо перед заголовками процедуры (сразу после Public Class Form1)ввести ключевое слово Public или Dim, за которыми следует ***имя*** переменной и объявление ее типа. После того, как переменная будет объявлена, читать ее содержимое, изменять его или отображать его на экране можно в любой процедуре программного кода модуля формы.

Например, операторы объявления, приведенные ниже, объявляют открытую переменную с именем RunTotal типа Integer:

|  |
| --- |
| **Public RunTotal As Integer**  **Dim RunTotal As Integer** |

***Обратите внимание на то, что в соответствии с технологией ООП все переменные, используемые внутри процедур, являются локальными, а число глобальных переменных должно быть минимально необходимо.***

### 4.3.2. Создание и использование процедур пользователей

Ранее мы познакомились со «встроенными» (т.е. готовыми к употреблению) процедурами (например, Sqrt( ) ), которые в **VB** являются методами класса. Мы также познакомились с процедурами обработки событий, которые являются частью пользовательского интерфейса и которые выполняются, только если происходит некоторое событие. В отличие от встроенных процедур, эти процедуры составляются программистом.

Что собой представляют процедуры?

Процедуры представляют собой отдельные блоки, из которых складывается код программы, причем каждая процедура должна выполнять какую-то законченную задачу.

Кроме процедур обработки событий, в программу можно включить процедуры, не связанные с событиями. Они выполняют отдельные действия и могут быть использованы (вызваны) неоднократно. Как известно, они называются общими или пользовательскими. Процедуры общего назначения вызываются на выполнение из кода программы. Использование процедур экономит время разработки приложения и позволяет избежать лишних ошибок.

Процедуры могут быть двух видов: процедуры - подпрограммы и процедуры – функции. Процедуры - функции отличаются от процедур - подпрограмм тем, что возвращают какое-то одно значение и поэтому могут участвовать в выражениях. Причем под подпрограммой или функцией понимается последовательность операций, которую нужно многократно выполнять в различных местах приложения. При этом, требуемый блок команд описывается в коде только один раз, после чего к нему можно обращаться из любой части программы и неоднократно.

Процедуры, которые будут рассмотрены ниже, не относятся ни к встроенным процедурам, ни к процедурам обработки событий – это общие процедуры или процедуры пользователей.

Общие процедуры, в отличие от процедур обработки событий, начинают выполняться не в ответ на какое-либо событие, а после их явного вызова. После выполнения такой процедуры происходит автоматический возврат в то место проекта, откуда процедура была вызвана.

Каждой вызываемой процедуре присваивается уникальное имя. Кроме того, для вызываемой процедуры может быть установлен перечень входных и выходных параметров.

***Входные параметры*** процедуры – это переменные, значения которых должны быть установлены до начала работы процедуры и которые участвуют в реализации процедуры. При описании процедуры перед ними ставиться ключевое слово **ByVal**.

***Выходные параметры*** процедуры – это переменные, значения которых вычисляются в результате работы процедуры. При описании процедуры перед ними ставиться ключевое слово **ByRef**.

В общем случае процедуры являются способом объединения набора связанных между собой выполнением одной задачи (конкретного алгоритма) операторов. **Visual Basic**включает два основных типа процедур:

1. ***Процедуры типа Function*** – ***процедура–функция*** или *процедура****-Function*** вызываются из процедур со­бытий или других процедур по имени. *Процедуры****-Function*** часто используются для вычислений, могут быть частью выражения, могут получать входные параметры и предназначены для возврата одного значения через свое имя или специальным оператором.
2. ***Процедуры типа Sub*** – ***процедура–подпрограмма*** или *процедура****-Sub*** вызываются из процедур со­бытий или других процедур по имени. Они могут получать входные параметры и возвращать измененные значения через список выходных параметров. Однако, в отличие от *процедуры****-Function***, *процедуры****-Sub*** не возвращают значения, свя­занного с их именами.

Процедуры-**Function** и процедуры-**Sub** могут определяться (описываться) в коде программы формы.

Процедуры общего назначения предоставляют следующие преимущества:

* они позволяют связать часто используемую группу операторов программы со  
  знакомым именем;
* они устраняют повторы строк кода, т.е. можно один раз определить процедуру  
  и заставить программу выполнять ее любое количество раз;
* они делают программы более простыми и легко читаемыми, программа, разделенная на небольшие части, легче воспринимается и понимается, чем программа, сделанная в виде одного большого фрагмента;
* они упрощают разработку программы (программы, разделенные на логические единицы, легче разрабатывать, писать и отлаживать; кроме того, если разрабатывают программу в группе разработчиков, то можно обмениваться процедурами и модулями, а не целыми программами);
* они могут быть повторно использованы в других проектах и решениях (можно легко встроить процедуры из стандартного модуля в другие программные проекты);
* они расширяют язык **VB** (процедуры часто могут выполнять задачи, которые не могут быть выполнены отдельными ключевыми словами **VB** или методами).

*Процедура****-Function*** – это группа операторов, расположенных между ключевыми словами Function и End Function*.* Операторы в функции выполняют осмысленную работу – в соответствии с заданным алгоритмом. Функции создают тогда, когда результатом выполнения процедуры является значение какой либо одной величины числового, строкового или логического типа. Чтобы ***выполнить*** (***вызвать*** или ***обратиться***)– функцию в программе, необходимо поместить в операторе (любое допустимое выражение) программы имя этой функции и все требуемые для нее **фактические** параметры.

***Аргументы (формальные параметры)*** процедуры**-Function** – это данные, используемые для работы функции, и они должны быть заключены в круглые скобки и разделены запятыми. В целом использование процедуры-**Function** в точности совпадает с использованием встроенных функций или методов. Функции, объявленные (описанные) в модуле формы, по умолчанию являются открытыми в пределах данной формы. В результате они могут использоваться (вызываться) из лю­бой процедуры события проекта или из другой процедуры.

***Описание*** *процедуры****-функции*** имеет следующий синтаксис:

Function *ИмяФункции* **(***ФормальныеПараметры***)** As *ТипВозвращаемогоЗначения*

***…***

*Операторы Функции*

***…***

Return *значение*

***…***

End Function

Важными являются следующие синтаксические элементы:

* ***ИмяФункции*** – это имя создаваемой функции;
* ***ТипВозвращаемогоЗначения*** – это тип значения, возвращаемого функцией, то есть тип величины, которая является результатом работы функции;
* ***Формальные параметры (аргумент)*** – это список необязательных аргументов, разделенных запятыми и используемых в данной функции. Каждый аргумент дол­жен быть объявлен с указанием конкретного типа данных. По умолчанию **Visual Basic** добавляет к каждому аргументу ключевое слово ByVal, которое указывает на то, что в функцию через данный аргу­мент передается копия значения фактического параметра и все изменения значения этого ар­гумента не будут возвращены в вызывающую процедуру;
* ***Операторы Функции*** – это блок операторов, который выполняет работу функции (первые операторы в этой функции обычно объявляют локальные переменные, которые будут использоваться внутри функции, а осталь­ные операторы выполняют работу функции).
* Return–это оператор, с помощью которого можно указать место, где в блоке кода функции требу­ется возвратить значение в вызывающую программу, и каково это возвращаемое значение. После выполнения этого оператора происходит выход из процедуры-функции и управление передается в то место программы, откуда эта процедура-функция была вызвана.

**Пример 4.3.2-1.Добавление в проект *Пример 4.1.7-1*** (***Счастливая семерка)* процедур и общих переменных.**

1. Форма нового проекта должна выглядеть как на рис. 4.3.2-1.



Рис. 4.3.2-1

1. Переместить точку ввода на пустую строку за оператором Public Class Form1 и ввести оператор объявления переменной Dim Wins As Short, а затем нажать на ***<Enter>.***

Этот оператор объявляет в программе открытую целочисленную переменную типа Short**.** Когда программа запускается, каждая процедура события этой программы получит доступ к этой переменной.

1. Щелкнуть дважды на кнопке формы ***Играть.***

В окне **Редактор кода (Code Editor)** появится процедура событияButton1\_Click( )для кнопки формы ***Играть***.

1. В этой процедуре события после оператора Веер( )следует ввести следующие операторы:

|  |
| --- |
| **Wins = Wins + 1**  **Label5.Text = "Побед:" & Wins** |

Эта часть кода программы увеличивает на единицу открытую переменную Wins, если при «вращении» появляется 7. Вторая строка использует оператор конкатенации (**&**), с помощью которого присваивает объек­ту *Label5.Text* строку в формате **Побед**: X, где X - это число выигрышей.

Надпись ***Побед: Х*** отслеживает выигрыши. Каждый раз, когда происходит выигрыш, она увеличивает значение переменной Wins на 1. После 10 вращений переменная Wins может принять значение, показанное на рис. 4.3.2-2.

1. Щелкнуть на ***Конец игры***, чтобы выйти из программы.

Глобальная переменная Wins полезна в процедуре события Button1\_Click( ) потому, что она сохраняла свое значение на протяжении нескольких вы­зовов. Если бы Wins была объявлена локально в процедуре события Button1\_Click( )*,* то эта переменная каждый раз сбрасывалась бы (обнулялась).

В этом примере использовалась открытая переменная для хранения числа выигрышей в программе игрового автомата. Открытые переменные отличаются тем, что они сохраняют свои значения во всей форме - другими словами, во всех компонентах, которые принадлежат к одному пространству имен проекта.

В проект примера 4.3.2-1 добавить функцию, которая будет вычислять долю выигрышей в игре, т. е., другими слова­ми, процентное выражение игр, в которых появляется хотя бы одна се­мерка. Чтобы это сделать, следует добавить в модуль формы процедуру - *Function* с именем HitRate( ) и открытую переменную Spins*,* а *з*атем вызывать процедуру - фун­кцию HitRate()каждый раз, когда производится щелчок на кнопке формы ***Играть.***

1. Отобразить форму программы ***Пример-4-3-2-1 (Счастливая семерка-2).***

Появится интерфейс пользователя программы ***Счастливая семерка2***.

1. С помощью элемента управления ***Label*** создать новую надпись ниже надписи ***Побед: 0*.**

Установить для этой надписи свойства, приведенные в таблице 4.3.2-1.

Таблица 4.3.2-1

|  |  |  |
| --- | --- | --- |
| **Объект** | **Свойство** | **Значение** |
| ***Label6*** | *Font* | *Arial, жирный, 12-point* |
| *ForeColor* | *красный* |
| *Name* | *Label6* |
| *Text* | *0.0%* |
| *TextAlign* | *MiddleCenter* |

Форма должна выглядеть аналогично той, которая изображена на рис. 4.3.2-1.

1. В окне **Обозреватель решений (Solution Explorer)** щелкнуть на команде ***Показать код (ViewCode)*.**

В окне **Редактора кода** (**CodeEditor)** появится программный код.

1. После оператора Public Wins As Short следует добавить следующее объявле­ние:

Public Spins As Short

Теперь модуль включает две открытые переменные, Wins и Spins, ко­торые доступны всем процедурам проекта. Причем Spins будет исполь­зоваться как счетчик для хранения числа сыгранных игр.

1. Далее добавить в модуль формы пустую строку, а затем следующее описание функции:

|  |
| --- |
| **Function HitRate(ByVal Hits As Short, ByVal Tries As Short) As String**  **Dim Percent As Single**  **Percent = Hits / Tries**  **Return Format(Percent, "0.0%")**  **End Function** |

После того как будет введена первая строка кода функции, **Visual Basic** автоматически добавит оператор End Function. Когда будет введена остальная часть кода функции, экран должен выглядеть ана­логично рис. 4.3.2-2.

Функция HitRate( ) определяет процентное соотношение выигрышей. Разделив аргумент Hits на аргумент Tries***,*** затем форматирует представ­ление результата с помощью встроенной функции Format( ). Функция HitRate( ) объявлена как имеющая тип строки, так как функция Format( ) возвра­щает строковое значение. Аргумент Hits и аргумент Tries - это места хранения двух коротких целых переменных, которые передаются в функцию при ее вызове. Функция HitRate( ) является функцией общего назначения и может быть использована с любыми короткими целыми числами или переменными, а не только с Wins и Spins.

1. За четвертой строкой процедуры события Button1\_Click( )

Label3.Text=CStr(Int(Rnd() \* 10))

следует ввести следующий оператор:

Spins = Spins + 1.

Этот оператор увеличивает на 1 переменную Spins каждый раз, ког­да пользователь щелкает на кнопке формы  ***Играть***, а в поля вывода чисел выводятся новые значения.

1. Прокрутить окно **Редактор кода (CodeEditor)** вниз, а затем в качестве последней стро­ки процедуры события Button1\_Click( ) между операторами End If и End Sub введите следующий оператор:

Label6.Text = HitRate(Wins, Spins)

При вводе функции HitRate( )обратите внимание, как **Visual Studio** ав­томатически отображает имена и типы аргументов только что создан­ной функции HitRate( ) (удобная подсказка).

Целью этого оператора является вызов функции HitRate( )с использо­ванием переменных Wins и Spins в качестве ее аргументов (фактических параметров). Возвра­щаемый результат – это процентное соотношение в строковом фор­мате, и это значение при каждой игре присваивается свойству *Text* надписи формы *Label6*.

1. Прокрутить процедуру события Form1\_Load( )в окне **Редактор кода (CodeEditor**) вниз  
   и удалите или закомментируйте (поместите перед ней символ комментария - апостроф) функцию Randomize.

Теперь при каждом запуске программы сгенерированные случайные числа будут соответствовать одному и тому же шаблону. Это помогает тестировать код, но по окончании тестирования не забудьте вернуть эту функцию обратно, чтобы результаты снова стали случайными.

|  |
| --- |
| **Option Strict On**  **Option Explicit On**  **Imports System.Math**  **Public Class Form1**  **Public Spins As Short**  **Public Wins As Short**  **Function HitRate(ByVal Hits As Short, \_**  **ByVal Tries As Short) As String**  **Dim Percent As Single**  **Percent = CSng(Hits / Tries)**  **Return Format(Percent, "0.0%")**  **End Function**  **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button2.Click**  **End**  **End Sub**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Igr()**  **End Sub**  **Sub Igr()**  **PictureBox1.Visible = False' скрыть картинку**  **Label1.Text = CStr(Int(Rnd() \* 10)) ' выбрать числа**  **Label2.Text = CStr(Int(Rnd() \* 10))**  **Label3.Text = CStr(Int(Rnd() \* 10))**  **Spins = CShort(Spins + 1)**  **If (Label1.Text = "7") Or (Label2.Text = "7") \_**  **Or (Label3.Text = "7") Then**  **PictureBox1.Visible = True'открыть рисунок**  **Beep()**  **Wins = CShort(Wins + 1)**  **Label5.Text = "Побед:"& Wins**  **End If**  **Label6.Text = HitRate(Wins, Spins)**  **End Sub**  **Private Sub Form1\_Load(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **HandlesMyBase.Load**  **Randomize()**  **End Sub**  **End Class** |

1. Чтобы запустить измененную программу ***Пример-4.3.2-1 (Счастливая семерка-3)следует*** щелкнуть на кнопке ***Start Debugging***.
2. Щелкнуть 10 раз на кнопке ***Играть***.

После 10 игр экран выглядит так, как показано на рис. 4.3.2-2.



Рис. 4.3.2-2

.

Если продолжить щелкать, то можно будет увидеть, что количество выигрышей упадет примерно до 28%. Функция HitRate( ) показывает, что в начале игры вы были слишком удачливы, но затем вам пришлось столкнуться с реальностью.

1. Чтобы закончить работать с программой, щелкнуть на кнопке ***Конец игры.***

Программа остановится и вернется в среду разработки. Чтобы увидеть, как программа работает «по-настоящему», верните в процедуру события Form1\_Load( ) функцию Randomize.

1. Чтобы сохранить изменения, щелкните на кнопке ***Сохранить все(Save All)***  на стандартной панели инструментов.

Процедуры-**Sub** похожи на процедуры-**Function**, за исключе­нием того, что процедура-**Sub** не возвращает значения, связанного с ее именем.

Проце­дуры-**Sub** также используются для обработки и обновления переменных, получаемых через список аргументов вызова процедуры, и передачи в вызывающую программу одного или несколько значений.

Базовый синтаксис ***описания процедуры*** Sub имеет вид

Sub *ИмяПроцедуры* ***(****ФормальныеПараметры****)***

***…***

*Операторы Процедуры*

***…***

End Sub

Отметим следующие синтаксические элементы:

* ***ИмяПроцедуры*** – это имя создаваемой процедуры-**Sub***.*
* ***Формальные параметры (аргументы )****-* это список необязательных аргументов, разделенных запятыми, если их больше одного, и используемых в процедуре-Sub***.*** Причем *к*аждый аргумент должен быть объявлен с указанием конкретного типа данных. По умолчанию **VS** добавляет к каждому ар­гументу ключевое слово ByVal, которое указывает на то, что в процедуру-подпрограмму через данный аргумент передается копия значения, и все изме­нения значения этого аргумента не будут возвращены в вызывающий код.
* ***ОператорыПроцедуры*** – это блок операторов, который выполняет работу процедуры по заданному алгоритму.

При вызове процедуры-**Sub** количество и типы аргументов, переда­ваемых в процедуру (они называются **фактическими** параметрами), должны совпадать с количеством и типом аргумен­тов, указанных при объявлении (описании) процедуры (это **формальные** параметры), и вся группа аргументов должна быть заключена в круглые скобки. Таким образом, количество и типы соответствующих друг другу формальных и фактических параметров должны совпадать. Если переменные, передава­емые в процедуру-Sub, изменяются при ее выполнении, то обновленные переменные не возвращаются, если процедура не объявляет эти аргу­менты с помощью ключевого слова ByRef. По умолчанию процедуры-Sub в модуле объявляются как открытые, так что они могут вызываться лю­бой процедурой проекта.

Необходимо отметить, что в **Visual Basic** все вызовы процедур-**Sub** после име­ни процедуры должны содержать круглые скобки. Если в про­цедуру не передается ни одного аргумента, то требуется нали­чие пустых круглых скобок(в нашем примере 4.3.2-1 таким образом вызывается процедура **Igr()**.

Процедуры Sub часто используются для обработки ввода данных в программу, когда информация поступает из двух или более источников, и требуется, чтобы она была в одном и том же формате. В следующем примере будет создана процедура-**Sub** с именем AddName( )*,* которая запрашивает у пользователя ввод данных и форматирует текст так, что его можно отобразить в нескольких строках в текстовом поле. Процедура сэкономит время, зат­рачиваемое вами на программирование, так как вы будете использовать ее в двух процедурах событий, и в каждой она будет связана с различ­ными текстовыми полями.

**Пример 4.3.2-2**. **Создание процедуры-Sub для ввода текстового поля.**

1. Создать новый проект с именем ***Пример-4-3-2-2.***

Будет создан новый проект, и в окне **Конструктор (Designer**) появится пустая форма.

1. Использовать элемент управления ***Textbox*** для создания двух текстовых полей, расположенных в центре формы рядом друг с другом.

Сегодня будем принимать решение о распределении группы студентов по подгруппам, и эти текстовые поля будут использоваться для хранения имен студентов, которые будут прикрепляться к двум преподавателям.

1. Использовать элемент управления ***Label*** и создать над этими текстовыми полями две надписи.

Эти надписи будут содержать имена преподавателей.

1. Использовать элемент управления ***Button*** и создать три кнопки: по одной под каждым текстовым полем и одну в нижней части формы.

Первые две кнопки будут использоваться для назначения студентов в подгруппы, а последняя кнопка – для выхода из программы.

1. Для этих объектов формы установить свойства, приведенные в таблице 4.3.2-2.

Так как текстовые поля будут содержать более одной строки, для них требуется установить свойства Multiline в значение True, а свойства Scrollbars *-* в значение Vertical. Эти установки обычно используются тогда, когда в текстовых полях отображается несколько строк текста. Также требуется установить их свойства TabStop в значение False, a их свойства ReadOnly в значение True так, чтобы информацию нельзя было бы изменить.

Таблица 4.3.2-2

|  |  |  |
| --- | --- | --- |
| **Объект** | **Свойство** | **Значение** |
| ***TextBox1*** | *Multiline* | *True* |
| *Name* | *TextBox1* |
| *ReadOnly* | *True* |
| *ScrollBars* | *Vertical (вертикальная)* |
| *TabStop* | *False* |
| ***TextBox2*** | *Multiline* | *True* |
| *Name* | *TextBox2* |
| *ReadOnly* | *True* |
| *ScrollBars* | *Vertical (вертикальная)* |
| *TabStop* | *False* |
| ***Label1*** | *Font* | *Bold (Полужирный)* |
| *Name* | *Label1* |
| *Text* | *"Доц. Скрыпникова М.И"* |
| ***Label2*** | *Font* | *Bold (Полужирный)* |
| *Name* | *Label2* |
| *Text* | *"Доц. Кувыкина М.И"* |
| ***Button1*** | *Name* | *Button1* |
| *Text* | *Добавить имя студента* |
| ***Button2*** | *Name* | *Button2* |
| *Text* | *Добавить имя студента* |
| ***Button3*** | *Name* | *Button3* |
| *Text* | *Выход* |
| ***Form1*** | *Text* | *Распределение студентов по подгруппам* |

1. Изменить размер и положение объектов так, чтобы форма выглядела как на рис. 4.3.2-3.

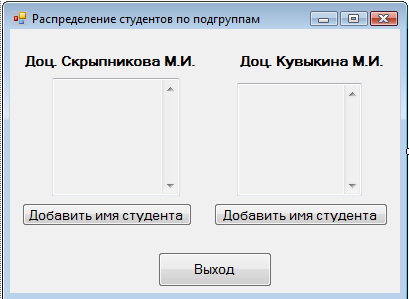


Рис. 4.3.2-3

Теперь требуется создать процедуру *Sub* общего назначения с именем AddName( ).

1. Ввести код процедуры ***AddName,*** показанный на рис.4.3.2-4.

|  |
| --- |
| **Sub AddName(ByVal Team As String, ByRef ReturnString As String)**  **Dim Prompt, Nm, WrapChar AsString**  **Prompt = "Ввести " & Team**  **Nm = InputBox(Prompt, "Ввод Фамилий студентов")**  **WrapChar = Chr(13) + Chr(10)**  **ReturnString = Nm & WrapChar**  **End Sub** |

Рис. 4.3.2-4

Эта процедура-**Sub** общего назначения для запроса имени студента использует функцию InputBox( ). При вызове про­цедуры она получает два аргумента: Team, являющийся строкой, со­держащей имя преподавателя подгруппы, и ReturnString***,*** который содержит пус­тую строковую переменную для отформатированного имени студента. ReturnString объявлена с ключевым словом ByRef*,* так что все изменения, сделанные в этом аргументе в процедуре, будут переданы через этот аргумент обратно в вызывающий код. Прежде чем имя студента будет возвращено, к строке добавляют­ся символы возврата каретки и перевода строки–chr(13)+chr(10) или vbCrLf, и, таким образом, каждое имя в текстовом поле будет отображаться на отдельной строке.

Эту методику можно использовать в любой строковой переменной, где требуется создать новую строку.

1. Снова отобразить форму, а затем дважды щелкнуть на первой из кно­пок ***Добавить имя*** формы (кнопке под текстовым полем ***Доц. Скрыпникова М.И.***). В проце­дуре события Button1\_Click() ввести следующие операторы:

|  |
| --- |
| **Dim Gr1Pos As String = ""**  **AddName("Студентов группы 1", Gr1Pos)**  **TextBox1.Text = TextBox1.Text & Gr1Pos** |

Вызов процедуры AddName( ) включает один аргумент, передаваемый по значению («**Студентов группы 1**»), и один аргумент, передаваемый по ссылке (**Gr1Pos**). Последняя строка использует аргумент, переданный по ссылке, для добавления текста в текстовое поле *TextBox1*. Оператор кон­катенации (&) добавляет новое имя в конец текста в этом текстовом поле.

1. В окне **Редактор кода (Code Editor)** щелкнуть на стрелке ***Class Name*** и в списке выбрать объект *Button2****.***

Затем щелкнуть на стрелке ***Method Name (имя метода)*** и выбрать событие ***Click***.

В окне **Редактор кода (Code Editor)**появится процедура события Button2\_Click( ).

1. В этой процедуре события ввести следующие операторы:

|  |
| --- |
| **Dim Gr2Pos As String= ""**  **AddName("Студентов группы 2", Gr2Pos)**  **TextBox2.Text = TextBox2.Text & Gr2Pos** |

Эта процедура события идентична Button1\_Click( ), за исключением того, что она посылает в процедуру AddName имя «**Студентов группы 2**» и об­новляет текстовое поле *TextBox2*.

1. Щелкнуть на стрелке ***Class Name*** и в списке выбрать объект *Button3.*

Затем щелкнуть на стрелке ***Method Name*** и выбрать событие ***Click.***

В окне **Редактор кода (Code Editor)** появится процедура события Button3\_С1ick( ).

1. В процедуре события Button3\_Click( ) ввеcти оператор End (рис. 4.3.2-5).

|  |
| --- |
| **Public Class Form1**  **Private Sub Button3\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button3.Click**  **End**  **End Sub**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button1.Click**  **Dim Gr1Pos As String = ""**  **AddName("Студентов группы 1", Gr1Pos)**  **TextBox1.Text = TextBox1.Text & Gr1Pos**  **End Sub**  **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button2.Click**  **Dim Gr2Pos As String = ""**  **AddName("Студентов группы 2", Gr2Pos)**  **TextBox2.Text = TextBox2.Text & Gr2Pos**  **End Sub**  **Sub AddName(ByVal Team As String, ByRef ReturnString As String)**  **Dim Prompt, Nm, WrapChar As String**  **Prompt = "Ввести "& Team**  **Nm = InputBox(Prompt, "ВводФамилийстудентов")**  **WrapChar = Chr(13) + Chr(10)**  **ReturnString = Nm & WrapChar**  **End Sub**  **End Class** |

Рис. 4.3.2-5

1. Чтобы сохранить изменения, щелкнуть на кнопке ***Сохранить все (Save All)*** на стан­дартной панели инструментов.
2. Чтобы запустить программу, щелкните на кнопке ***Начать Отладку*** (***Start Debugging)***, расположенной на стандартной панели инструментов.
3. Щелкните на кнопке формы ***Добавить имя студента*** под текстовым полем ***Доц.Скрыпникова М.И.,*** а затем введите в поле ввода ***Иванов Иван И.*** (Вы можете ввести и какое-ни­будь другое имя.)

Поле ввода должно выглядеть примерно так, как на рис. 4.3.2-6.



Рис. 4.3.2-6

1. Чтобы добавить это имя в текстовое поле ***Доц.Скрыпникова М.И.***, щелкните на кнопке **ОК**.

Имя появится в первом текстовом поле.

1. Щелкните на кнопке формы ***Добавить имя студента*** под текстовым полем ***Доц.Кувыкина М.И.***, вве­дите в поле ввода имя ***Петров Сидор В.***, а затем нажмите на **<Enter*>.***

Имя появится в текстовом поле подгруппы 2. Экран при этом будет выглядеть примерно так, как показано на рис . 4.3.2-7.

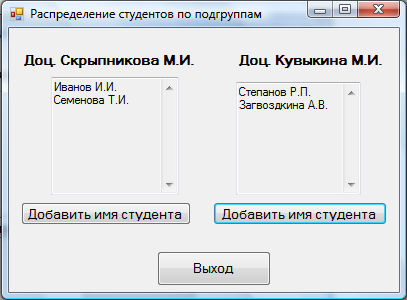


Рис. 4.3.2-7

1. В каждое из этих двух текстовых полей введите еще по нескольку имен.

Каждое имя в текстовых полях отображается в отдельной строке. Эти текстовые поля не прокручиваются автоматически, так что если вве­сти больше имен, чем помещается в текстовое поле, то увидеть все имена одновременно не удастся. Для доступа к невидимым именам следует использовать полосы прокрутки.

1. Когда закончите ввод, щелкните на кнопке ***Выход,*** чтобы остановить програм­му.

Пример 4.3.2-2 продемонстрировал, что одна процедура-**Sub** может выполнять задачи ввода для двух и более процедур событий. Используя эту базо­вую концепцию как отправную точку, можно создать более сложные программы, которые используют процедуры типов **Sub** и **Function** для выполнения общих задач.

### 4.3.3. Передача параметров по значению (ByVal) и по ссылке (ByRef)

Итак, каждой общей процедуре присваивается уникальное название – имя процедуры. Кроме того, для процедуры четко устанавливается перечень входных и выходных данных (входных и выходных параметров). Прежде чем определять процедуру, необходимо четко представить, **что** процедура делает, что она должна получать (**что** у нее на входе) и какие результаты возвращать (**что** на выходе). Параметры, перечисляемые в скобках при описании процедуры, называются **формальными** параметрами(перед такими параметрами обязательно ставится клю­чевое слово ByVal или ByRef и после имени параметра указывается его тип), а параметры, которые указываются при вызове процедуры, называются **фактическими**. Передача параметров в процедуру из вызывающей программы (т.е. от фактических параметров формальным) может осуществляться двумя способами: по значению(By Value) или по ссылке(By Reference). По значению передаются входные параметры, а по ссылке – выходные.

**Входной параметр процедуры** – это переменная, значение которой должно быть установлено до начала работы процедуры и которая участвует в работе процедуры. Перед таким параметром ставится клю­чевое слово ByVal (передача по значению) В этом случае в процедуру передается не сама переменная, а ее копия. В момент вызова процедуры для хранения формального параметра в ней создается локальная переменная, которая получает копию значения фактического параметра, а после окончания работы процедуры эта локальная переменная уничтожается. Поэтому изменение формального параметра в процедуре не затрагивает фактический параметр, так как изменяется не сама переменная, а ее копия и таким образом все изменения, сделанные в переменной, переданной по значению, не передаются об­ратно в вызывающую процедуру.

**Выходной параметр процедуры** – это переменная, которая получает свое значение в результате работы процедуры. Использование ключевого слова ByRef указывает, что переменные должны быть переданы в процедуру по ссылке. Это значит, что процедура не создает локальную переменную под формальный параметр, а получает доступ к области памяти, в которой хранится фактический параметр, т.е. получает его адрес. Поэтому все изменения, сделанные в рамках процедуры, будут производиться прямо на тех же ячейках памяти, в которых хранится фактический параметр, и следовательно переданы обратно в вызывающий код.

При выборе способа передачи параметра (по ссылке или по значению) решающий критерий – должен ли параметр изменяться в процедуре.

Вот некоторые рекомендации о том, когда следует использовать ByVal, а когда - ByRef:

* используйте ByVal тогда, когда требуется, чтобы процедура не изме­няла передаваемую в нее через аргумент переменную;
* используйте ByRef тогда, когда требуется позволить процедуре из­менять передаваемую в нее переменную;
* когда сомневаетесь, используйте ключевое слово ByVal, так как в этом случае фактический параметр гарантированно не будет изменен в ходе выполнения процедуры.

Однако, так как при передаче по значению происходит копирование объекта, данные большого объема (например, структуры или многомерные массивы) более эффективно передавать по ссылке.

Структуру программы, оператор присваивания и выражения мы уже разобрали. Для реализации алгоритмов линейной структуры необходимо знать, как может быть реализован в **VB** ввод исходных данных и вывод результатов.

### 4.3.4. Средства программирования алгоритмов линейной структуры и функции ввода и вывода

***Последовательными*** называются такие алгоритмические струк­туры, в которых функциональные элементарные блоки вы­полняются в том порядке, в котором они записаны или изо­бражены на схеме алгоритма. Такая структура может быть составлена из совокупности блоков «**Пуск/останов**», «**Ввод/вывод**», а также блоков «**Процесс**», используемых для проведения вычислений (рис. 2.3.2-1).

Для ***ввода информации*** пользователь может использовать ***объект текстового поля*** и его свойство *Text*. Другим способом ввода информации является использование функции **InputBox( )** для отображения на экране диалогового окна, и последующее использо­вание переменной для хранения текста, введенного пользователем.

Поскольку технология программирования предполагает ***максимальное использование процедур при разработке проектов***, студенты в своих работах для ввода исходных данных типа Double могут использовать процедуру-**Function**:

|  |
| --- |
| **'процедура-Function ввода данных типа Double из текстового поля TextBox**  **Function vvodDbl1(ByVal T As TextBox) As Double**  **Return (Val(T.Text))**  **End Function** |

Вызов такой процедуры можно осуществлять из любой точки программы, например, для ввода значения переменной x1 типа Double из текстового поля*TextBox1* формы:

|  |
| --- |
| **…**  **Dim x1 As Double**  **…**  **x1 = vvodDbl1(TextBox1)**  **…** |

Другой способ ввода информации можно реализовать с помощью функции VB InputBox( ). Функция InputBox( )вызывает окно **InputBox,** которое состоит из четырех элементов:

* строки заголовка (**title**);
* приглашение к вводу (**prompt**);
* поля ввода со значением, предлагаемым по умолчанию (**defaul**t);
* двух кнопок **(<OK>** и **<Отмена>**).

Формат функции InputBox( )следующий:

P=InputBox( *Приглашение* [,*Заголовок*] [,*ЗначениеПоУмолчанию*][,*x*][,*y*]),

где:*P* –возвращаемое значение функции;

*x, y* – координаты левого верхнего угла окна.

Все необязательные параметры указаны в квадратных скобках.

**Пример 4.3.4-1.Ввод данных с помощью функции *InputBox( ).***

1. Создать проект c именем ***Пример-4-3-4-1***.

В **IDE** откроется пустая форма проекта***Пример-4.3.4.1.***

Создадим форму этого проекта, которая содержит одну надпись(Label) и две кнопки(Button). Для получения вво­да от пользователя будет использоваться функция InputBox( ), а за­тем введенное значение необходимо отобразить в надписи формы (рис. 4.3.4-1).

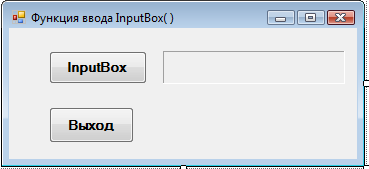


Рис. 4.3.4-1

1. Дважды щелкнуть мышью на кнопке ***InputBox****.*

В окне **Редактора кода** (**CodeEditor)** появится процедура события Button1\_Сliск()*.*

1. Чтобы объявить две переменные и использовать функцию InputBox( ), необходимо ввести операторы программы, показанные на рис. 4.3.4-2.

|  |
| --- |
| **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button1.Click**  **Dim Prompt, FullName As String**  **Prompt = "Введите Ваше имя"**  **FullName = InputBox(Prompt)**  **MsgBox(FullName, , "Введено имя")**  **Label1.Text = FullName**  **End Sub** |

Рис. 4.3.4-2

С помощью оператора Dim производится объявление сра­зу двух переменных: Prompt и FullName. Обе переменные объявля­ются с использованием типа String. (Вы можете объявить в одной стро­ке столько переменных, сколько хотите, но при условии, что они имеют один и тот же тип.)

Вторая строка процедуры события присваивает переменной Prompt текстовую строку. Это сообщение будет использовано как текстовый аргумент для функции InputBox( ). ***Аргумент (параметр)***– это значение или выражение, передаваемое в процедуру или функцию. Следующая строка вызывает функцию InputBox( ) и присваивает результат этого вызова (текстовую строку, введенную пользователем) переменной FullName. InputBox( ) – это специальная функция **VB**, которая отобра­жает на экране диалоговое окно и запрашивает пользовательский ввод. В дополнение к строке запроса функция InputBox( ) поддерживает другие аргументы.

После того как InputBox( ) возвратит в программу текстовую строку, чет­вертый оператор этой процедуры помещает имя пользователя в свой­ство *Text* объекта ***Label1*** и, таким образом, отображает его в форме.

1. Сохранить изменения.
2. Запустить программу.
3. Щелкнуть на кнопке ***InputBox*** формы.

**VB**  начнет выполнять процедуру события Button1\_Click( ), и на экране появится диалоговое окно ***InputBox***.

1. Ввести свое полное имя, а затем щелкнуть на ***ОК*** (рис. 4.3.4-3).

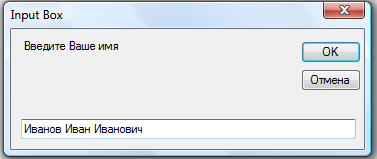


Рис. 4.3.4-3

Функция InputBox( ) возвращает введенное имя в программу и поме­щает его в переменную FullName. Затем программа использует эту переменную для отображения этого имени на форме, как показано на рис. 4.3.4-4.

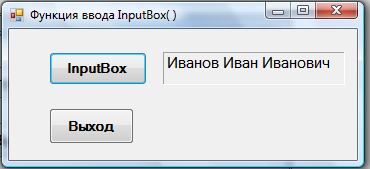


Рис. 4.3.4-4

Функцию InputBox( ) необходимо использовать в своих программах всегда, когда требуется запросить у пользователя какую-либо информацию. Вы мо­жете использовать эту функцию совместно с другими элементами управления ввода и управлять потоком данных, поступающих в программу и исходящих из нее.

1. Чтобы завершить выполнение программы, следует щелкнуть на кнопке формы ***Выход.*** Программа остановится, и вернется среда разработки.

InputBox( ) – это «встроенная» функция. ***Функция***- это оператор, который выполняет некую работу (такую, как запрос у пользователя информации или вычисление формулы), а затем возвращает резуль­тат в программу. Значение, возвращаемое функцией, может быть присвоено пе­ременной, как это сделано в программе примера 4.3.3-1, или присвоено свойству, или используется другим оператором или функцией. Функции **VB**  имеют один или более аргументов, определяющих их действия. В примере 4.3.4-1функция InputBox() использовала для отображения в диалоговом окне ин­струкций для пользователя одну переменную Prompt. Когда функция использует более одного аргумента, эти аргументы разделяются запятыми, а вся группа аргументов заключается в круглые скобки. Следующий оператор показывает вызов функции с двумя аргументами:

FullName = lnputBox(*Prompt*, *Title*).

Обратите внимание, что в этом описании синтаксиса используется курсив, ука­зывающий, что определенные элементы являются заполнителями (формальными параметрами) вместо реально указываемой информации.

Процедура-Function vvodDbl2( ) , использующая для ввода InputBox( ),может быть реализована следующим образом:

|  |
| --- |
| **'процедура-Function ввода данных типа Doubleиз функции InputBox( )**  **Function vvodDbl2(ByVal p As String, ByVal T As TextBox) As Double**  **T.Text = InputBox(P)**  **Return CDbl(Val(T.Text))**  **End Function** |

Вызов такой процедуры можно осуществлять из любой точки программы, например, для **ввода значения переменной x1 типа Double.**

|  |
| --- |
| **…**  **Dim x1 As Double**  **…**  **x1 = vvodDbl2("Введите имя", TextBox1)**  **…** |

Для вывода информации необходимо отобразить содержимое переменной на форме, что можно сделать, присвоив значение этой пе­ременной свойству (такому, как свойство *Text* объекта элемента управления формы), или пере­дав переменную как аргумент в функцию диалогового окна.

|  |
| --- |
| **'Процедура вывода данных типа Double в TextBox**  **Sub vivodDbl1(ByVal Z As Double, ByVal T As TextBox)**  **T.Text = CStr(Z)**  **End Sub** |

Вызов такой процедуры можно осуществлять из любой точки программы, например, для вывода значения переменной x1 типа Double в текстовое поле**:**

|  |
| --- |
| **Dim x1 As Double**  **…**  **vivodDbl1(x1, TextBox1)** |

Одной из полезных функций диалоговых окон для отображения вывода является функция MsgBox( ). При вызове функции MsgBox( )она отображает диало­говое окно, иногда называемое окном сообщения, используя при этом различные дополнительные опции. Аналогично InputBox( ),она принима­ет на входе один или несколько аргументов, а результат вызова этой функции может быть присвоен переменной. Синтаксис функции MsgBox( ) имеет вид:

[*B* = ]MsgBox(*Prompt*[*, Buttons*] [*, Title*]),

где *Prompt*–это текст, отображаемый в окне сообщения, *Buttons*–это число, определяющее кнопки, пиктограммы и другие опции отображе­ния окна сообщения, a *Title*–это текст, отображаемый в строке заголов­ка окна сообщения. Переменной B присваивается результат, возвращаемый этой функцией и указывающий, на какой кнопке диало­гового окна щелкнул пользователь.

Если с помощью функции MsgBox( ) просто отображается сообщение, то переменная B*,* оператор присвоения (**=**), аргумент *Buttons* и аргумент *Title* необязательны. В следующем примере будет использоватьcя только аргумент *Title.*

**Пример 4.3.4-2. Вывод сообщения с помощью функции MsgBox( )*.***

Если окно **Редактор кода (Code Editor)** из предыдущего примера не отображается в **IDE**, следует сделать двойной щелчок мышью на кнопке ***InputBox*** формы *Функция ввода InputBox* (рис. 4.3.4-4)

1. *.*

В окне **Редактор кода (Code Editor)** появится процедура события Button1\_Click(). (Это код, который вы ввели в предыдущем примере.)

1. Выделить в процедуре события оператор (последняя строка):

Label1.Text = FullName

Это оператор, который отображает в надписи содержимое перемен­ной FullName***.***

1. Нажать клавишу <**Delete>***,* чтобы удалить эту строку.  
   Оператор будет удален.
2. Вместо него в процедуре события ввести следующую строку:

MsgBox(FullName, , "Введено имя")

Этот новый оператор вызывает функцию MsgBox( ), отображает в диа­логовом окне содержимое переменной FullName и помещает в его стро­ку заголовка слова **Введено имя***.* (Необязательный аргумент *Buttons* и переменная *B* опущены.) Процедура события должна выглядеть, как на рис. 4.3.4-5. Обратите внимание: если аргумент *Buttons* опущен, то между *Promp t*и *Title* в синтаксисе функции ***MsgBox( )*** должны быть две запятые.

|  |
| --- |
| **Public Class Form1**  **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button2.Click**  **End**  **End Sub**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button1.Click**  **Dim Prompt, FullName AsString**  **Prompt = "Введите Ваше имя"**  **FullName = InputBox(Prompt)**  **MsgBox(FullName, , "Введено имя")**  **‘Label1.Text = FullName**  **End Sub**  **End Class** |

Рис. 4.3.4-5

1. Щелкнуть на кнопке ***Начать отладку (StartDebugging)*** на стандартной панели инструментов.
2. Щелкнуть на кнопке ***InputBox*** формы, а затем ввести в поле ввода имя и щелкнуть на ***ОК.***

***Visual Basic*** сохранит введенное значение в переменной FullName*,* а за­тем отобразит его в окне сообщения. Экран будет выглядеть, как на рис. 4.3.4-6.



Рис. 4.3.4-6

1. Чтобы закрыть окно сообщения, щелкнуть на ***ОК***. Затем щелкнуть на кнопке ***Выход*** формы***,*** чтобы закрыть программу.

Процедура-Sub vivodDbl2( ), использующая для вывода MsgBox( )может быть реализована следующим образом:

|  |  |
| --- | --- |
| |  | | --- | | **'Процедура вывода данных типа Double функцией MsgBox**  **Sub vivodDbl2(ByVal Z As Double, \_**  **ByVal S As String,ByVal T As TextBox)**  **T.Text = MsgBox(CStr(Z), S)**  **End Sub** | |

Вызов такой процедуры можно осуществлять из любой точки программы, например, для вывода значения переменной x1 типа Double в текстовое поле**:**

|  |
| --- |
| **…**  **Dim z As Double**  **…**  **vivodDbl2(z, "Вывод значения z", TextBox1)**  **…** |

**Пример 4.3.4-3.Вычисление периметра и площади треугольника с использованием процедур для ввода исходных данных и вывода результатов.**

Переработаем программный код **Проект-4-1-Лаб,** используя процедуры vvodSng3( )

|  |
| --- |
| **Option Strict On**  **Option Explicit On**  **Imports System.Math**  **Public Class Form1**  **Function vvodSng3(ByVal T As TextBox) As Single**  **Return CSng(Val(T.Text))**  **End Function**  **Sub vivodSng3(ByVal Z As Single, ByVal T As TextBox)**  **T.Text = CStr(Z)**  **End Sub**  **Sub PS(ByRef P As Single, ByRef S As Single)**  **P = (A + B + C) : Pp = P / 2**  **S = CSng(Sqrt(Pp \* (Pp - A) \* (Pp - B) \* (Pp - C)))**  **End Sub**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button1.Click**  **Dim P As Single, S As Single**  **Dim A, B, C, Pp As Single**  **A=vvodSng3(TextBox1): B=vvodSng3(TextBox2):C=vvodSng3(TextBox3)**  **PS(A, B, C, P, S)**  **vivodSng3(P, TextBox4)**  **vivodSng3(S, TextBox5)**  **End Sub**  **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button2.Click**  **End**  **End Sub**  **End Class** |

### 4.3.5. Перегрузка методов (процедур)

***Перегрузка*** – это создание более чем одной процедуры в классе с одинаковыми именами, но с различными типами аргументов.

***Перегрузка*** методов представляет собой один из частных случаев **полиморфизма**. Напомним, что ***полиморфизм*** в переводе с греческого языка означает «много форм». В отношении к методам класса полиморфизм позволяет с помощью одного имени представлять различный программный код, то есть различное поведение.

Перегрузка особенно полезна, когда в объектной модели нужно использовать одинаковые имена процедур, работающих с различными типами данных. Например, в примерах из предыдущего параграфа для вывода данных были использованы процедуры:

|  |
| --- |
| **'Процедура вывода данных типа Double в TextBox**  **Sub vivodDbl1(ByVal Z As Double, ByVal T As TextBox)**  **T.Text = CStr(Z)**  **End Sub** |

|  |  |
| --- | --- |
| |  | | --- | | **'Процедура вывода данных типа Double функцией MsgBox**  **Sub vivodDbl2(ByVal Z As Double, \_**  **ByVal S As String,ByVal T As TextBox)**  **T.Text = MsgBox(CStr(Z), S)**  **End Sub** | |

|  |
| --- |
| **Sub vivodSng3(ByVal Z As Single, ByVal T As TextBox)**  **T.Text = CStr(Z)**  **End Sub** |

Для вызова этих процедур использовались следующие операторы:

|  |
| --- |
| **vivodDbl1(x1, TextBox1)**  **…**  **vivodDbl2(z, "Вывод значения z", TextBox1)**  **…**  **vivodSng3(S, TextBox5)** |

При описании эти методы можно перегрузить, то есть присвоить им одно имя, Для этого необходимо использовать модификатор процедуры **Overloads** перед описанием процедур. Например,

|  |
| --- |
| **'Процедура вывода данных типа Double в TextBox**  **Overloads Sub vivod (ByVal Z As Double, ByVal T As TextBox)**  **T.Text = CStr(Z)**  **End Sub** |

|  |  |
| --- | --- |
| |  | | --- | | **'Процедура вывода данных типа Double функцией MsgBox**  **Overloads Sub vivod (ByVal Z As Double, \_**  **ByVal S As String,ByVal T As TextBox)**  **T.Text = MsgBox(CStr(Z), S)**  **End Sub** | |

|  |
| --- |
| **Overloads Sub vivod (ByVal Z As Single, ByVal T As TextBox)**  **T.Text = CStr(Z)**  **End Sub** |

Для вызова этих процедур использовались следующие операторы:

|  |
| --- |
| **vivod(x1, TextBox1)**  **…**  **vivod(z, "Вывод значения z", TextBox1)**  **…**  **vivod(S, TextBox5)** |

Перегруженный элемент класса создается путем добавления двух или более методов с тем же именем. Все перегруженные элементы должны иметь разные списки параметров. При перегрузке процедуры следующие элементы не могут быть использованы как средства определения.

* Модификаторы, такие как  **ByVal** или **ByRef**, которые применимы к элементу, или параметрам элемента.
* Имена параметров
* Возвращаемые типы процедур

Ключевое слово **Overloads**  не обязательно при перегрузке, но, если некоторый перегруженный элемент использует ключевое слово  **Overloads**, то все остальные перегруженные элементы с этим же именем также должны быть определены этим ключевым словом.

Таким образом, при перегрузке методов, сигнатура метода – это комбинация его имени, списка параметров, служебных слов, определяющих доступность метода, пространства имен, к которому относится имя метода.

### 4.3.6. Задачи для самостоятельного решения по теме

### «Структура VB-программ и процедуры.

### Средства программирования алгоритмов линейной структуры»

Написать программный код процедуры (процедуры-функции или процедуры-подпрограммы) с соответствующими входными и выходными формальными параметрами для решения задач из п. 4.2.5 Темы 4.2. Причем в процедуре не должно быть ввода исходных данных и вывода результатов.

Написать также программный код вызывающей процедуры, в которой описываются исходные данные, происходит инициализация этих данных, задаются конкретные значения переменным, которые являются фактическими параметрами вызывающей процедуры, и вызывается процедура поставленной задачи.

### 4.3.7. Тестовые задания по теме

### «Структура VB-программ и процедуры.

### Средства программирования алгоритмов линейной структуры»

1. **Алгоритм – это**
2. перечень правил выполнения определенных действий
3. ориентированный граф, указывающий порядок исполнения некоторого набора команд
4. понятное и точное предписание исполнителю совершить последовательность действий, направленных на достижение поставленной цели
5. протокол компьютерной сети
6. **Алгоритм называется линейным, если**
7. его выполнение предполагает многократное повторение одних и тех же действий
8. ход его выполнения зависит от истинности тех или иных условий
9. его команды выполняются в порядке их естественного следования независимо от каких-либо условий
10. он включает в себя вспомогательный алгоритм
11. **Программа – это**
12. протокол взаимодействия компонентов компьютерной сети
13. понятное и точное предписание человеку совершить последовательность действий,

направленных на достижение поставленных целей

1. ориентированный граф, указывающий порядок исполнения команд компьютера
2. алгоритм, записанный на языке программирования
3. набор команд операционной системы компьютера
4. **Поиск и устранение ошибок в программе происходит на этапе**
5. выбора методики решения
6. постановки задачи
7. алгоритмизации
8. решения задачи с различными исходными данными
9. отладки программы
10. программирования
11. **Процедура – это**
12. последовательность логически связанных фрагментов программы, оформленная как отдельная часть программы специальным способом
13. последовательность отдельных фрагментов программы
14. последовательность только циклических фрагментов программы
15. последовательность только разветвляющихся фрагментов программы
16. **Глобальные переменные – это**
17. переменные, доступные во всех других процедурах и модулях без механизма передачи параметров
18. все переменные одной отдельной процедуры
19. все переменные всех используемых процедур
20. все переменные, описанные в модуле
21. в списке нет правильного ответа
22. **Локальные переменные – это**
23. все переменные с одинаковыми именами во всех процедурах
24. переменные только одной процедуры
25. переменные всех процедур
26. переменные, область видимости которых ограничена процедурой, в которой они

описаны

1. **Формальными параметрами называются**
2. список передаваемых параметров, указанных в описании процедуры
3. список параметров, указанных в модуле программе
4. все переменные, используемые в процедуре
5. список передаваемых параметров, указанных при вызове процедуры
6. **Формальные параметры являются для данной процедуры**
7. глобальными
8. фактическими
9. локальными
10. формализованными
11. **Формальные параметры определяют**
12. имя и тип переменных
13. только имена переменных
14. только типы массивов
15. в списке нет правильного ответа
16. **При вызове процедуры происходит замена формальных параметров**
17. локальными
18. глобальными
19. статическими
20. фактическими
21. **Фактические параметры – это**
22. конкретные значения переменных, на которые заменяются формальные параметры при обращении к процедуре
23. глобальные переменные
24. локальные переменные
25. статические переменные
26. **Параметры могут передаваться в процедуру**
27. только по значению
28. по значению или по ссылке
29. только по ссылке
30. любым образом
31. **При передаче параметра по значению в процедуру передается**
32. копия передаваемого параметра
33. адрес передаваемого параметра
34. адрес и значение передаваемого параметра
35. ничего не передается
36. **По значению могут передаваться**
37. входные параметры процедуры
38. выходные параметры процедуры
39. как входные, так и выходные параметры процедуры
40. только глобальные переменные
41. **По ссылке могут передаваться**
42. как входные, так и выходные параметры процедуры
43. только входные параметры процедуры
44. только выходные параметры процедуры
45. в списке нет правильного ответа
46. **Изменение в процедуре значения параметра, передаваемого по значению**
47. не приводит к изменению значения этого параметра в вызывающей процедуре
48. приводит к изменению значения этого параметра в вызывающей процедуре
49. приводит к изменению значений всех параметров вызывающей процедуры
50. в списке нет правильного ответа
51. **К средствам процедурного программирования в VB относятся**
52. процедуры-функции и процедуры-подпрограммы
53. процедуры-функции
54. процедуры-подпрограммы
55. процедуры-функции, процедуры-подпрограммы и событийные процедуры
56. **Результатом выполнения функции является**
57. скалярная величина
58. несколько скалярных величин
59. коллекция данных
60. коллекция данных и скалярные величины
61. **Переменные, описанные внутри функций, являются**
62. глобальными
63. локальными
64. статическими
65. фактическими
66. **При вызове функций и процедур контролируется**
67. количество и тип аргументов, их порядок следования
68. количество аргументов, их порядок следования
69. порядок следования аргументов
70. количество и тип аргументов
71. **Передача фактических параметров-выражений в процедуру и в процедуру-функцию производится**
72. по значению
73. по ссылке
74. как по ссылке, так и по значению
75. в процедуру - по ссылке, в процедуру-функцию – по значению