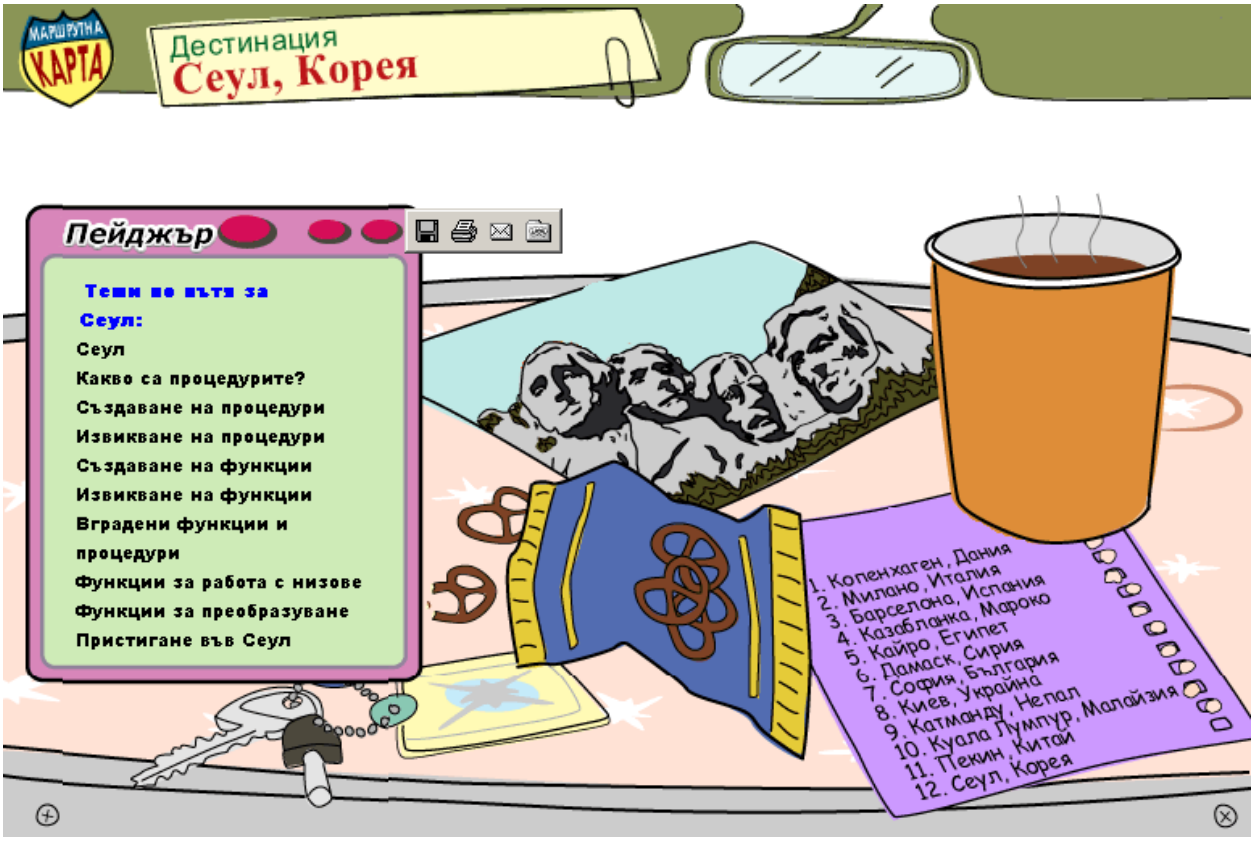
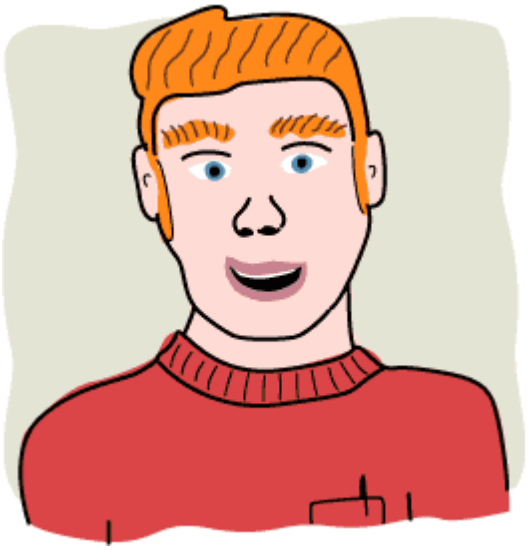

Маршрутна карта
Дестинация
Сеул, Корея
Пейджър
Теми по пътя за Сеул:
Сеул
Какво са процедурите?
Създаване на процедури
Извикване на процедури
Създаване на функции
Извикване на функции
Вградени функции и процедури
Функции за работа с низове
Функции за преобразуване
Пристигане във Сеул
1. Копенхаген, Дания
2. Милано, Италия
3. Барселона, Испания
4. Казабланка, Мароко
5. Кайро, Египет
6. Дамаск, Сирия
7. София, България
8. Киев, Украйна
9. Катманду, Непал
10. Куала Лумпур, Малайзия
11. Пекин, Китай
12. Сеул, Корея

Функции

Сеул, Корея

Полезен съвет

Някои учители накратко ще представят тази секция след секцията за изразите с присвояване или след секцията за аритметичните изрази, след което отново ще се върнат на нея в края на курса. Повечето



На този етап от курса вие ще приложите всичките си знания, които натрупахте до сега. Ще ви покажа как да пишете свои собствени процедури и функции, както и да използвате някои, които са вградени в Visual Studio 2010.

Това ще бъде една забавна част от пътуването ни! Предстои ни път до Сеул, Корея, така, че да потегляме!

Какво са процедурите?

Процедурите приличат на малки програми. Те се използват за реализиране на специфични задачи в приложенията. Когато пишете приложение, можете да го разделите да отделни части с определена функционалност, за всяка от които да създадете процедура. Приложението може да съдържа произволен брой процедури. Не трябва да се ограничавате в тяхното използване. Пишете толкова процедури, колкото са ви необходими, но се опитайте всяка една от тях да има специфична функционалност.

Защо използваме процедури? Процедурите правят кода ви по-лесен за четене и дебъгване, тъй като го разделят на малки парчета с определено предназначение. По-лесно е да пишете и дебъгвате няколко по-къси процедури, отколкото цяло приложение. Веднъж след като сте дебъгнали една процедура, тя е готова за използване. Тя ще работи винаги, когато решите да я използвате.

Процедурите спестяват време, когато кодът ви е сложен и трябва да бъде достъпен от различни части на програмата. Те могат да се използват многократно. Веднъж след като сте написали една процедура, тя може да бъде използвана от всяка точка на програмата ви, включително и от други процедури. Представяте ли си това!

Процедурите осигуряват начин за разделяне на работата върху голямо приложение. Отделните програмисти често пишат специфично множество от процедури, използвани в приложението. Те са отговорни за тяхното реализиране и дебъгване и сработването им с останалите части от кода.

Някои основни приложение на процедурите са свързани с реализиране на изчисления, форматиране и визуализиране на информация, настройка на потребителския интерфейс, запитване на потребителя за данни и осигуряване на вход и изход от програмата.

Създаване на процедури

Създаването а процедури е друга област от програмирането, която наистина предизвиква. Тя наподобява дизайна – изисква комбинация от творчество и програмни умения. Тъй като процедурите са като малки програми, те могат да съдържат всички разгледани до момента оператори!

Ще ви помогна да напишете собствена процедура и след това ще ви покажа как да я използвате в приложение. Ще започнем с изучаването на синтаксиса за дефиниране на процедури, след което ще ви покажа как да ги създавате. Синтаксисът е следния:

```
Private Sub SubName()
    code statement 1
    code statement 2
    code statement etc.
End Sub
```

Полезен съвет

Разгледайте програмата за игра на морски шах като пример за използване на

Забележете, че думите "Private", "Sub" и "End Sub" са ключови думи. Те са оцветени в синьо. Ключовата дума Private означава, че процедурата е достъпна само за останалия код от формата. SubName е името на процедурата. Можете да именувате процедурите както искате, но е препоръчително да използвате описателни имена, които подсказват предназначението им. Забележете кръглите скоби след името на процедурата. Когато процедурата получава информация, променливите, които я съхраняват, се поставят между тези скоби. Това можете да видите във всички процедури, обработващи събития, които използвахме в предишните секции на курса. Редовете от кода, който се изпълнява от процедурата, се поставя между ключовите думи Sub и End Sub. Изпълнението на редовете е последователно. Можете да използвате всякакви изрази.

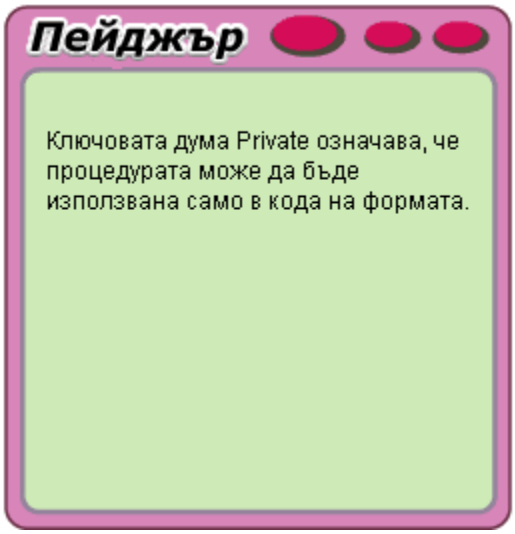
Пейджър

За да бъде ясно кой код принадлежи към тялото на процедурата, трябва да го отмествате навътре.

Сега ще ви покажа как да напишете своя собствена процедура и да я извикате от кода си. Създайте ново Windows приложение с име SimpleSub. Отворете прозореца с кода. Намерете ред със следния текст: "Windows Forms Designer generated code". На следващия ред добавете следния код:

```
Private Sub MyMessage()
    MessageBox.Show("Here's a message from Sub
MyMessage.")
End Sub
```

Както виждате кодът в процедурата MyMessage се изпълни, се визуализира диалогов прозорец с текст: "Here's a message from Sub MyMessage". Но как да изпълним този код?



Полезен съвет

Този начален курс не разглежда разработването на приложения с множество форми.

Извикване на процедури

За да накарате кода в процедурата ви да се изпълни, трябва да напишете ред в програмата, който "извиква" процедурата. Когато процедурата е "извикана", кодът от нея се изпълнява. Синтаксисът за извикване на процедура е прост. Просто напишете името на процедурата следвано от двойка кръгли скоби. Ето и синтаксисът:

```
SubName()
```

А сега да извикаме процедурата, която току що написахте.

В приложението SimpleSub, добавете бутон върху формата. Задайте стойност "MyMessage" за свойството Text на бутона. Щракнете двукратно върху него, за да създадете обработчик на събитието Button1_Click. Добавете следния код:

```
MyMessage()
```

Изградете и стартирайте проекта. Натиснете бутона "MyMessage". Показва се диалогов прозорец със съобщение "Here's a message from Sub MyMessage". От къде се появи той? Това е редът с код от процедурата MyMessage. Той се изпълнява при извикването на процедурата, която обработва генерираното при натискането на бутона събитие.

Сега ще ви дам друга процедура, която да добавите към приложението SimpleSub. Тя ще покаже друго съобщение. Отворете прозореца с кода и добавете следния код след оператора End Sub на процедурата MyMessage:

```
Private Sub YourMessage()
    MessageBox.Show("Here's a message from Sub
YourMessage."
End Sub
```

Този път нека да извикаме процедурата YourMessage от процедурата MyMessage вместо да използваме генерираното при натискане на бутон събитие. Редактирайте процедурата MyMessage по следния начин:

```
Private Sub MyMessage()
    MessageBox.Show("Here's a message from Sub
MyMessage.")
    YourMessage()
End Sub
```

Извикахме процедурата YourMessage от процедурата MyMessage. Изградете и стартирайте приложението. Натиснете бутона "MyMessage". Появяват се два диалогови прозореца. Първият показва "Here's a message from Sub MyMessage", а вторият – "Here's a message from Sub YourMessage". Как работи този код? Генерираното при натискане на бутона събитие извиква процедурата MyMessage. Тогава кодът в нея се изпълнява. Процедурата MyMessage притежава два реда. Първият от тях визуализира диалоговия прозорец със съобщението "Here's a message from Sub MyMessage". Вторият ред извиква процедурата YourMessage, която показва диалоговия прозорец със съобщението "Here's a message from Sub YourMessage".

Ясно ли е? Създадохте първите си две процедури и извикахте едната от тях при генериране на събитие. Втората процедура извикахте от първата!

Създаване и извикване на процедури с аргументи

Едно от полезните неща във Visual Basic и повечето съвременни езици за програмиране е, че процедурата може да получава информация. Това е много удобно, когато пишете процедури, които дават или действат в зависимост от подадената им информация. Частите от тази информация се наричат "аргументи". Когато пишете процедура, която получава аргументи, трябва да определите техния брой и тип.

Полезен съвет

Параметрите не са напълно ново понятие. Всички обработчици на събития имат параметри.

Ето синтаксиса на процедура, която изисква аргументи:

```
Private Sub SubName(ByVal argumentName1 As
argumentType1, ByVal argumentName2 As
argumentType2, ByVal argumentNameN As
argumentTypeN)
    code statement 1
    code statement 2
    code statement etc.
End Sub
```

Полезен съвет

Ключовата дума ByVal означава, че стойността на параметъра се подава на процедурата, но оригиналният параметър остава непроменен.

В случая кръглите скоби не са празни. Те съдържат списък от аргументи заедно с техните типове, които се подават на процедурата. Забележете, че думата "ByVal" е ключова дума. Тя е оцветена в синьо. Типът на аргументите е общовалиден като Integer, String и т.н. Аргументите се разделят един от друг със запетая.

Сигурен съм, че един реален пример ще направи нещата по-ясни. Отворете прозореца с код на приложението SimpleSub. Под оператора End Sub на процедурата YourMessage добавете следната процедура:


```
Private Sub
GeneralMessage(ByVal InMessage
As String)
    MessageBox.Show(InMessage)
End Sub
```

Тази процедура изисква аргумент от тип String. При извикването си тя визуализира диалогов прозорец, който показва стойността на подадения й аргумент.

А сега да извикаме процедурата GeneralMessage и да й подадем аргумент от тип String. Добавете втори бутон върху формата. Задайте стойност "AnyMessage" за свойството Text на бутона. Щракнете двукратно върху него, за да създадете обработчик на събитието Button2_Click. Добавете следните три реда код:

```
GeneralMessage("Whatever message.")
GeneralMessage("Some other message.")
GeneralMessage("A different message.")
```

Изградете и стартирайте проекта. Натиснете бутона "AnyMessage". Показват се три съобщения. Как е възможно това? Генерираното при натискане на бутона събитие извиква процедурата GeneralMessage три пъти. Всеки път като аргумент й подава различен символен низ. Първият път процедурата се извиква с аргумент "Whatever message", вторият път – с аргумент"Some other message.", а третия път – "A different message". Когато кодът в процедурата се изпълнява, символните низове се подават като аргументи и се използват в израза MessageBox.Show.

Да разгледаме още един пример. Този път ще ви покажа как да напишете процедура, която получава два аргумента от тип Integers, събира ги и показва резултата в диалогов прозорец. Готови ли сте? Добавете нова процедура с име "Adders" към проекта SimpleSub. Тя трябва да изглежда по следния начин:

```
Private Sub Adders(ByVal AddOne As Integer, ByVal
AddTwo As Integer)
Dim Total As Integer
    Total = AddOne + AddTwo
    MessageBox.Show(Total)
End Sub
```

Забележете, че при извикването си процедурата изисква два целочислени аргумента.

Върху формата добавете бутон. Задайте стойност "Adders". За свойството Text на бутона. Щракнете два пъти върху него, за да създадете обработчик на събитието Button3_Click. Добавете следния код:

```
Adders(34, 57)
```

Изградете и стартирайте приложението Натиснете бутона. "Adders". Визуализира се диалогов прозорец, който показва 91. Кодът от обработчика на събитието Button3_Click извива процедурата с аргументи 34 и 57. Процедурата Adders изпълнява три реда с код. Първият от тях декларира променлива с име Total. След това вторият ред установява в променливата Total сумата от аргументите AddOne (34) и AddTwo(57), които съхраняват подадените на процедурата стойности. Накрая третият ред показва стойността на променливата Total (91).

Пейджър

Променливите, които са декларирани в процедура с ключовата дума Dim, са локални променливи. Деклариране на променлива в процедура е аналогично на деклариране на променлива в обработчик на събитие, който също е процедура. Променливите са достъпни само при изпълнението на процедурата. След като процедурата приключи, променливите повече не са достъпни.

Сега вече можете да пишете сами процедури и да им подавате информация. Ще ви покажа и как се пишат функции. Подобно на процедурите, функциите също могат да получават информация посредством аргументи. За разлика от тях, обаче функциите могат и да предоставят информация.

Създаване на функции

Основната разлика между процедурите и функциите е, че функциите връщат информация. Тази информация се нарича върната стойност и е от определен тип. Когато пишете функции, трябва да определите типа на върнатата стойност. Нека да ви покажа синтаксиса на функция, която получава аргументи и връща стойност:

```
Private Function FunctionName(ByVal argumentName1 As
argType1, ByVal argumentName2 As argType2, ByVal
argumentNameN As argTypeN) As
ReturnType
    Code Statement 1
    Code Statement 2
    Code Statement etc.

FunctionName=returnValue
End Function
```

Полезен съвет

Забележете, че върнатата от функцията стойност се присвоява на името ѝ. Учениците използваха функции в предишните програми. Накарайте ги да извикат някои от тях повторно, като например Val.

Забележете, че думите "Private", "Function" и "End Function" са ключови думи. Те са оцветени в синьо. Името на функцията трябва да бъде описателно. Списъкът от аргументи заедно с техните типове се поставя в кръгли скоби след името на функцията. Аргументите се отделят със запетайка. Можете да дефинирате произволен брой аргументи, но върнатата стойност винаги е само една. Нейният тип също трябва да се определи (например, As Integer или As String). Последният ред от кода в тялото на функцията установява върнатата стойност. Забележете, че този ред присвоява на името на функцията стойността, която трябва да се върне като резултат.

Един реален пример ще ви помогне да схванете синтаксиса. Създайте ново Windows приложение с име FunctionJunction. В прозореца с кода открийте реда със следното съдържание "Windows Forms Designer generated code". На следващия ред добавете следния код:

```
Private Function Multipliers(ByVal MultOne As Integer,
ByVal MultTwo As Integer) As Integer
    Multipliers = MultOne * MultTwo
End Function
```

Тази функция е наречена Multipliers и приема два целочислени аргумента MultOne и MultTwo. Типа на върнатата стойност се задава след списъка с аргументи. В този случай той е Integer (As Integer). Функцията съдържа един ред с код:

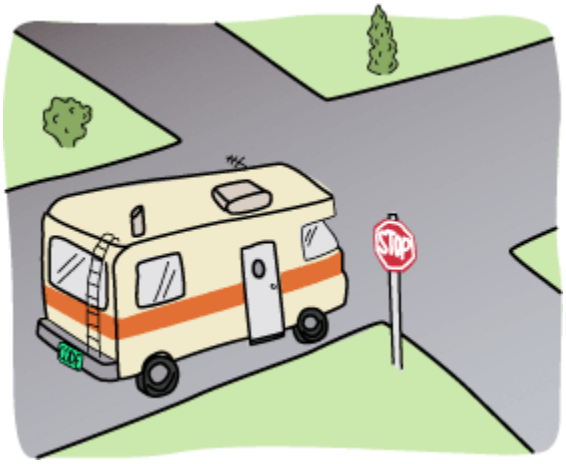
```
Multipliers = MultOne * MultTwo
```

Този ред с код умножава двата аргумента и установява върнатата стойност. Името на функцията – Multipliers, трябва да бъде използвано като име на върнатата стойност. В противен случай ще възникне грешка.

А сега да извикаме функцията и да използваме върнатата от нея стойност в програмата.

Извикване на функции

Един от начините за извикване на функция е да се използва променлива, която да присвои върнатата стойност. Променливата и върнатата стойност трябва да са от един и същ тип.



```
VariableOfReturnType =
FunctionName(argument1,
argument2, argumentN)
```

Нека да извикаме написаната от нас функция и да използваме върнатата от нея стойност в кода. Върху формата поставете бутон. Задайте стойност "Multiply" за свойството Text на бутона. Добавете следния код в обработчика на събитието Button1_Click:

```
Dim Product As Integer
Product = Multipliers(34, 57)
MessageBox.Show(Product)
```

Изградете и стартирайте проекта. Натиснете бутона "Multiply". Визуализира се диалогов прозорец, които показва 34x57, което е 1938. Как работи този код? В обработчика на събитието Button1_Click е декларирана променлива с име Product. Функцията Multipliers се извиква с аргументи 34 и 57. Тя умножава аргументите MultOne (34) и MultTwo (57) и присвоява върнатата от нея стойност на променливата Product. Върнатата стойност е от тип Integer. В обработчика на събитието Button1_Click целочислената променлива Product приема върнатата от функцията Multipliers стойност, след което резултатът се показва в диалогов прозорец.

Хей, имам идея! Да направим нещо по специално, като използваме функцията Multipliers. Върху формата добавете втори бутон. Задайте стойност "MultiplyAgain" за свойството Text на бутона. Добавете следния код в обработчика на събитието Button2_Click:

```
Dim Product As Integer
Product = Multipliers(Multipliers(2, 3),
Multipliers(5, 7))
MessageBox.Show(Product)
```

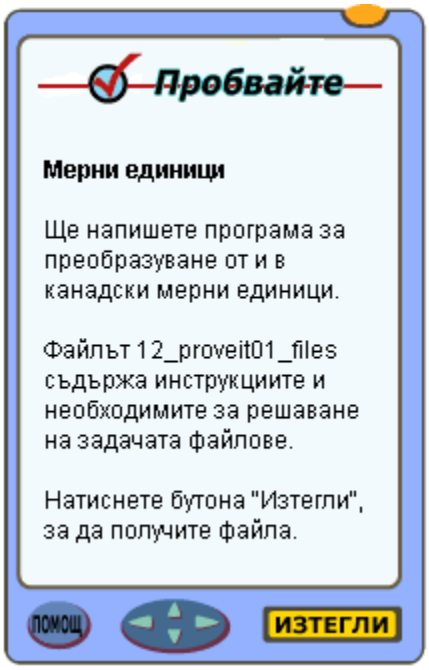
Изградете и стартирайте проекта. Натиснете бутона "MultiplyAgain". Показва се числото 210. Извикахме функцията Multipliers три пъти. Първия път й подадохме аргументи 2 и 3. Върнатата стойност (6) използваме като първи аргумент при третото извикване на функцията. Втория път й подадохме аргументи 5 и 7. Върнатата стойност (35) използваме като втори аргумент при третото извикване на функцията. Третия път извикахме функцията с аргументи 6 (върнатата стойност от първото извикване) и 35 (върнатата стойност от първото извикване).

Видяхте ли как да извикате една функция от друга функция (дори и същата)?

Пейджър

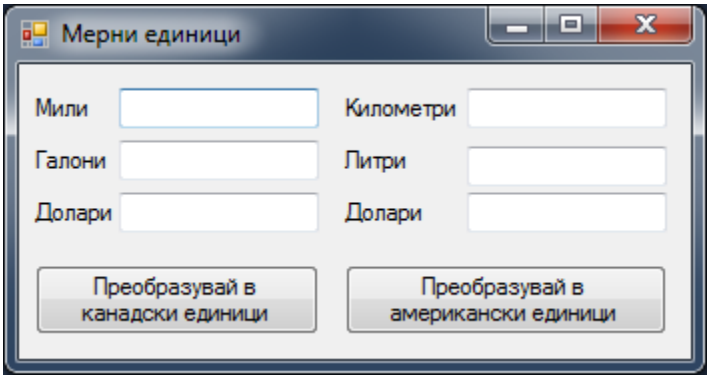
Променливите, които са декларирани във функция с ключовата дума Dim, са локални променливи. Те са достъпни само при изпълнението на функцията. След като функцията приключи, променливите вече не са достъпни.

12 Пробвайте 01 Мерни единици



Понякога бъркам ляво и дясно. В един от тези случаи направих ляв завой вместо десен и така се озовахме в Канада. Нека да напишем програма за преобразуване от американски в канадски мерни единици и обратно.

Формата, която трябва да създадете, изглежда по следния начин:



При натискането на бутона Convert to Canadian правим преобразуване от дясно на ляво. При натискането на бутона Convert to US правим преобразуване от дясно на ляво.

Използвайте следните съотношения:

1 Миля = 0.621 Километри
1 Галон = 3.79 Литри
1 Американски долар = 1.27 Канадски долар

Полезен съвет

Тази програма може да бъде написана и без използване на функции. Не позволявайте това. Проверете кода на учениците. Напомнете им, че проверяваме какво са научили, а не просто какъв резултат са получили.

Вече научихте как да дефинирате собствени функции и шаблони. В тази програма трябва да използвате функции, в противен случай няма да получите кредити за нея. За да ви улесним, написахме една от шестте функции. Вие трябва да напишете останалите пет. Предоставяме ви и кода за двата бутона.

Ако програмата работи правилно, я покажете на учителя си.

Вградени функции и процедури

Средата Visual Studio 2010 притежава много вградени функции и процедури за извършването на общи задачи. Те са дебъгвани и тествани и са достъпни за всички .NETезици за програмиране. Когато е възможно, вие би трябвало да ползвате вградените функции и процедури, вместо да пишете свои собствени. Това не само, че спестява време, но и уеднаквява вашия код, правейки го по-разбираем. Също така съществуват вградени функции и процедури, които не можете да напишете сами. Много от тях осъществяват достъп до скритите .NET и Windows API класове.

Ще ви покажа някои от най-често използваните вградени в Visual Studio 2010 средата функции и процедури Framework и приложението им във Visual Basic. Ще забележите, че някои от тях ви изглеждат познати, защото сте ги срещали в направените до сега задачи. Ние ви ги дадохме, за да ги използвате без да бъдат обяснени, така че сега ще разясним тяхното действие!

Функции за работа с низове

Средата .NET притежава много вградени функции, които се използват за обработване на символни низове. Те премахват определени символи, извличат низове, които са част от по-големи такива или преобразуват низове в малки или големи букви. Повечето от тях лесно се демонстрират с примери.

Всички функции за работа с низове имат подобен синтаксис. Те могат да бъдат извикани за всеки низ от символи. Някои от тях имат аргументи, а други – нямат. Повечето от тях връщат стойност от тип String, въпреки че функцията Length() връща стойност от тип Integer. Ето и синтаксиса на вградените във Visual Basic низови функции, където returnValue и someString са променливи:

```
returnValue =
someString.StringFunctionName(argument1, argument2,
argumentN)
```

Полезна функция е Length. Тя връща дължината на символен низ като Integer стойност. Ето пример:

```
MyText = "TextBox1"
myLength = MyText.Length ' returns 9, the length of
"TextBox1"
```

Функциите ToLower и ToUpper се използват за преобразуване на символния низ съответно в малки и големи букви. Разгледайте следните примери:

```
MyText = "TextBox1"
MyCaps = MyText.ToUpper 'returns "TEXTBOX1"

MyText = "TextBox1"
mySmalls = MyText.ToLower 'returns "textbox1"
```

Понякога ви е необходимо да знаете дали даден низ съдържа интервали в края или в началото си. Също така може би ще искате да ги премахнете. В такъв случай използвайте функцията Trim както е показано по-долу:

```
MyText = " TextBox1"
myClean = MyText.Trim 'returns
"TextBox1"

MyText = " TextBox1 "
MyCleaner = MyText.Trim
'returns "TextBox1"
```

Друга полезна функция е Substring. Тя връща част от символен низ (подниз). Когато я извиквате, трябва да й подадете два целочислени аргумента. Първият то тях задава началната позиция, а вторият – дължината на подниза. Забележете, че първият символ от низа е с индекс 0, а не с 1. Вторият символ е с индекс 1, а не с 2 и т.н.

```
MyText = "TextBox1"
MySub = MyText.Substring(0, 4) 'returns "Text",
starts at position 0 with length=4

MyText = "TextBox1"
MySub = MyText.Substring(1, 2) 'returns "ex",
starts at position 1 with length=2
```

Функцията Substring получава два аргумента от тип Integer и връща стойност от тип String. Първият аргумент определя стартовата позиция за търсене, а втория – дължината на низа, който ще се върне от нея. Запомнете, че индексите на символите в низовете започват от 0, а не от 1.

Знам, че нямате търпение да видите низовите функции в действие, така че да напишем малко приложение, за да покажем как работят! Създайте ново Windows приложение с име BuiltIn. Върху формата добавете две текстови полета. Задайте стойност "Yes" за свойството Multiline и стойност "Vertical" за свойството Scrollbars на второто текстово поле. Разпънете полето така, че височината му да стане почти колкото тази на формата. Добавете следния код в обработчика на събитието Button1_Click:

```
Dim MyText As String
Dim TempText As String = ""
MyText = TextBox1.Text
TempText = TempText & MyText
TempText = TempText & vbNewLine & MyText.ToLower
TempText = TempText & vbNewLine & MyText.ToUpper
TempText = TempText & vbNewLine & MyText.Trim
TempText = TempText & vbNewLine &
MyText.Substring(0, 4)
TempText = TempText & vbNewLine &
MyText.Substring(1, 2)
TempText = TempText & vbNewLine & MyText.Length
TextBox2.Text = TempText
```

Изградете и стартирайте приложението. Натиснете бутона. Второто текстово поле показва резултата от изпълнението на всяка една от низовите функции върху символния низ от първото текстово поле. Променете текста в първото текстово поле и натиснете отново бутона.

Полезен съвет

Кодът по-горе ще бъде по-значим, ако учениците го поставят в програма, която след това изпълнят стъпка по стъпка като използват средствата за дебъгване.

Генератори на случайни числа

Много от програмите се нуждаят от генериране на случайни числа. Това особено е в сила за приложения, например игри, които реализират случайни събития като хвърляне на монети или зарове. Средата Visual Studio 2010 поддържа клас System.Random, който притежава функции за генериране на случайни числа. Ще ви покажа код, който симулира хвърляне на зар. Върху формата в проекта BuiltIn добавете един бутон. Задайте стойност "RollEm" за свойството Text на бутона. Добавете следния код в обработчика на събитието Button2_Click:

```
Dim myRandomGenerator As System.Random
Dim myRandomInteger As Integer
myRandomGenerator = New System.Random
myRandomInteger = myRandomGenerator.Next(1, 7)
MessageBox.Show(myRandomInteger)
```

Изградете и стартирайте приложението. Натиснете бутона "RollEm". Визуализира се диалогов прозорец, който показва случайно число между 1 и 6. Натиснете отново бутона. Показва се друго случайно число между 1 и 6.

Как работи този код? Първо декларирахме променлива с име myRandomGenerator от тип System.Random. След това декларирахме променлива с име myRandomInteger от тип Integer. Променливата myRandomGenerator инициализирахме с нов System.Random обект като използвахме ключовата дума New. За да генерираме случайно число между 1 и 6, извикахме функцията Next() на обекта myRandomGenerator. с аргументи 1 и 6. Те дефинират минималната и максималната стойност на случайното число, което се генерира. Функцията Next() връща случайно число между 1 и 6, което се присвоява на променливата myRandomInteger. Последният ред от кода показва полученото случайно число в диалогов прозорец.

Функции за преобразуване

Visual Basic притежава две функции за преобразуване, които ще ви бъдат полезни, когато работите с числа. Това са функциите Val и Int. Първо да разгледаме функцията Val. Тази функция се използва за преобразуване на низ в число. Ако низът съдържа десетична запетая, върнатата от функцията стойност е от тип Double. В противен случай върнатата стойност е от тип Integer. Функцията Val често се използва за преобразуване на въведения в текстово поле низ в число, което може да бъде използвано за извършване на изчисления. Ето и някои примери:

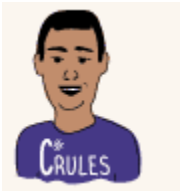
```
Dim MyInt As Integer
MyInt = Val("123") + 123 'returns 246

Dim MyDouble As Double
MyDouble = Val("123.22") + 123 'returns 246.22
```

Функцията Int връща цялата част на определено число. Това е частта вляво от десетичната запетая. Разгледайте следните примери,за да видите как работи:

```
Dim MyInt As Integer
MyInt = Int(123) 'returns 123
MyInt = Int(123.45) 'returns 123
```

Процедури и функции в C#

C# също ви позволява да създавате функции. Тези функции могат да получават аргументи и да връщат стойности. В C# няма еквивалент на процедурите. Те поддържат само функции. Също така при тях няма ключова дума "Function". Функциите се декларират посредством специален синтаксис и последователност от определени ключови думи. Да попитаме Дърк и Джин за функциите в тези езици за програмиране.



Както каза Клиф в C# не съществуват ключовите думи "Sub" или "Function". Във Visual Basic процедурите и функциите се различават. В C# между тях няма разлика. Да разгледаме кода по-долу, който декларира функция на C#, която приема два аргумента и връща целочислена стойност.

```
private static int Adders_C(int oneAdd, int twoAdd)
{
    int intSum=0;
    intSum = oneAdd + twoAdd;
    return intSum;
}
```

Забележете, че не използваме ключовата дума Function, както във Visual Basic.Ключовите думи "private" и "static" показват, че функцията може да бъде извикана само от кода на формата. Ключовата дума "int" (след static) определя типа на връщаната от функцията стойност (в случая Integer). Аргументите, които трябва да се подадат при извикването на функцията, са поставени в кръгли скоби след името на функцията Adders_C. И двата аргумента oneAdd и twoAdd са от тип int. Забележете ключовата дума "return". Тя определя стойността, която трябва да се върне от функцията. (в случая intSum).

Функцията Adders_C може да бъде извикана посредством следния код на C#. Той е подобен на кода, с който извикваме функции във Visual Basic:

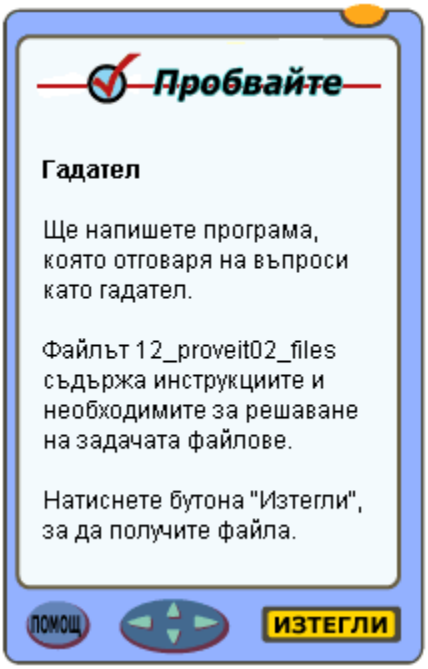
Пристигане във Сеул, Корея

Вече сме във Сеул, Корея! Със сигурност беше забавно! Ники и аз ви показахме с удоволствие как да пишете Visual Basic код, а Клиф and Джин използваха възможността да ви научат на това, което знаят за C#. Сега вече можете да определите някои прилики и разлики между тези езици за програмиране.

По пътя към Сеул ви показахме как да създавате и извиквате процедури и функции, както и да използвате някои, които са вградени в Visual Studio 2010 средата. Запомнете, че процедурите се използват, когато нямате върната стойност, а функциите, когато имате. При извикването им можете да им подавате аргументи. Тези аргументи се използват в кода от тялото на процедурите и функциите. Те не са необходими, ако променливите са декларирани глобално, т.е. ако са декларирани преди първата процедура.

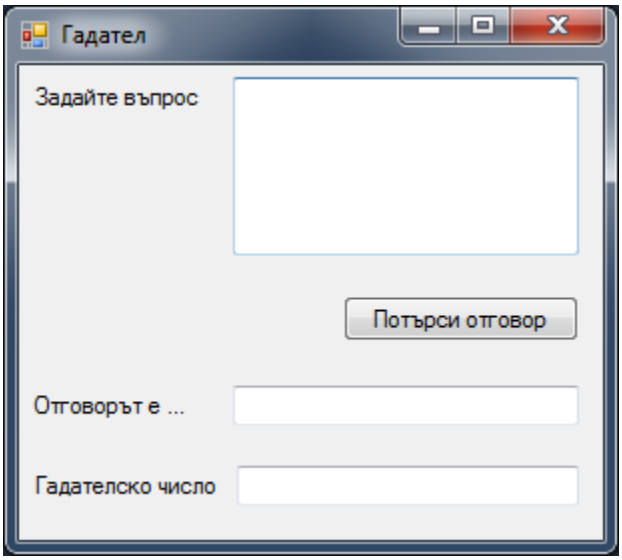
О, не мислите, че можете да си тръгнете без да сте направили още една задача?

12 Пробвайте 02 Гадател



Не вярвам, че можем да виждаме в бъдещето, но е забавно да се опитваме. Нека да напишем програма, която да отговаря на въпроси като гадател.

Формата, която трябва да създадете, изглежда по следния начин:



При натискането на бутона "Потърси отговор" използвайте метода SubString, за да преброите всяка една от гласните във въпроса (A, E, I, O, U). Не забравяйте малките и големите букви се броят поотделно. Проверете дали въпросът е въведен. Използвайте оператори за цикъл.

Запомнете, че методът Length преброява символите в низ. Позициите на символите в низ се номерират от 0 така, че ако броят им е 10, то позициите им ще имат номера от 0 до 9.

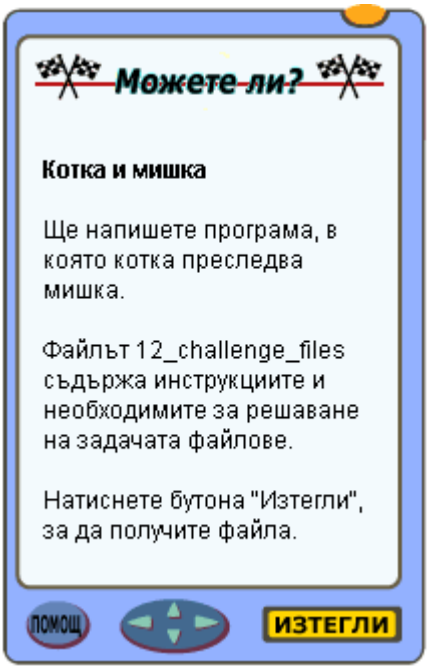
Като използвате броя на гласните букви във въпроса генерирайте отговор. Използвайте въображението си при създаване на отговорите. "ДА" и "НЕ" са подходящи, но отегчителни. Трябва да имате най-малко пет отговора. Препоръчително е да са повече.

Ако програмата работи коректно, я покажете на учителя си.

Комисията по състезанията остави за вас едно предизвикателство!

Полезен съвет

Както очаквате, тази програма е по-трудна от останалите в курса. Учениците не би трябвало да опитват да я направят наведнъж. Нека да направят една част, като движението на мишката. След това спирането на мишката на границата и т.н.



12 Можете ли? Котка и мишка

Време е за вашето предизвикателство. Играта на котка и мишка е доста по-трудна от тези, които вече реализирахме.

Създайте форма подобна на следната:

Приемете, че екранът е с ширина 600 пиксела и височина 300 пиксела.

Движете котката и мишката, като променяте местоположението на полетата, които съдържат техните изображения, добавяйки положителни и отрицателни стойности към текущите им X и Y координати.

Използвайте таймер, който отмерва 1/4 от секундата. В процедурата, обработваща събитието на таймера осигурете движението на котката и мишката. Това не може да стане директно. За да преместите полето, визуализиращо дадена картинка, трябва да преизчислите X и Y координатите му като използвате следния код:

```
Mouse.Location = New Point(X, Y)
```

След като реализирате преместването проверете следното:

- Мишката е избягала
- Котката е хванала мишката
- Мишката трябва да промени посоката на движението си, защото е твърде близо до котката
- Котката и мишката са до границите на екрана.

Котката е в поле за визуализиране на картинка с име Cat, а не PictureBox1.

Мишката е в поле за визуализиране на картинка с име Mouse, а не PictureBox2. Ако котката е на 20 пиксела от мишката (при изчислението се отчита горния ляв ъгъл на полетата за визуализация на картинките), то мишката е хваната. Ако мишката е на 20 пиксела от някое от двете черни текстови полета, представящи дупките ѝ, то мишката е избягала. В противен случай спрете таймера.

Ако мишката е на 40 пиксела от котката, то тя трябва да смени посоката на движението си.

Едната дупка на мишката е с координати (50, 150), а втората – (530, 150).

В шаблона има няколко неща, които ще ви помогнат да напишете програмата.

Функцията HowFar връща разстоянието между две точки с координати X и Y. Например,

```
Distance = HowFar(Cat.Location.X, Cat.Location.Y,
Mouse.Location.X,_ Mouse.Location.Y)
```

Функцията NewChg връща случайно число, което може да бъде положително или отрицателно. Числото представя разстоянието, на което се преместват мишката и котката на всеки ход. Например,

```
CatXChg = NewChg()
```

В процедурата, която обработва събитието Button_Click на бутона "Нова игра" инициализирайте със случайни стойности местоположението на котката и мишката. Активирайте таймера.

Съвет: Много програмисти считат, че е по-добре кодът да се пише на етапи, които завършват с тестване. Вие можете да напишете програмата на следните стъпки:

- Позициониране на котката и мишката
- Преместване на котката
- Проверка за достигане границата на екрана
- Преместване на мишката
- Проверка за достигане границата на екрана
- Проверка за избягала мишка
- Проверка за хваната мишка
- Проверка за близостта между котката и мишката

След всяка стъпка тествайте кода.

Ако програмата работи правилно, я покажете на учителя си.

Продължение / Обобщение

Съхранете посоката на разположението на изображенията. Променяйте тази посока при промяна в посоката на движение на котката или мишката. Използвайте следния код:

```
Cat.Image.RotateFlip(RotateFlipType.Rotate180FlipX)
```

Определете посоката, в която се движи мишката, и променяйте движението на котката, така че да следва това на мишката. Например, ако котката има X координата 450, а мишката – 250, то котката трябва да се движи в отрицателна посока.

Добавете куче, което да преследва котката.

Проверка на знанията

НАПРАВЕТЕ ТЕСТА ОТНОВО

1 Какво представляват аргументите на процедурите?

- A. Несъвместими стойности
- B. Информация, която се подава на процедурата за да я използва.
- C. Броят на редовете с код

2 Как ще получите буквите “BC” от низа X = “ABCD”?

- A. X.Substring(1, 2)
- B. X.Substring(2, 3)
- C. X.Substring(2, 2)

3 Как ще накарате една процедура да се изпълни?

- A. SubroutineName()
- B. Execute SubroutineName()
- C. Go To SubroutineName()

4 Как ще преобразувате низа от променливата X в число?

- A. X.ToNumber
- B. Convert(X)
- C. Val(X)