

ვიზუალური დაპროგრამების
საფუძვლები

VISUAL BASIC

ზ. ლურწკაია მ. მესხია



თბილისი 2012

VISUAL STUDIO

ვიზუალური დაპროგრამების
საფუძვლები

VISUAL BASIC

ზ. ლურწყაია მ. მესხია

თბილისი 2012 წ.

რეცენზენტები:

თენგიზ მაჭარაძე,
ტექნიკის მეცნიერებათა დოქტორი,
სრული პროფესორი

ნოდარ ჯიბლაძე
ტექნიკის მეცნიერებათა დოქტორი,
სრული პროფესორი

ნიგნი განკუთვნილია დამწყები პროგრამისტებისათვის, ვისაც სურს დაეუფლოს დაპროგრამებას და დაიწყოს საკუთარი პროგრამების შექმნა უმოკლეს ვადებში. განხილულია დაპროგრამების საფუძვლები, დაპროგრამების ენის Visual Basic.Net-ის მაგალითზე. პროექტების შესაქმნელად გამოყენებულია Visual Studio 2008-ის გარემო. შესაძლებელია პროგრამის შემდგომი ვერსიების გამოყენებაც (Visual Studio 2010).

ახალ საუკუნეში Visual Basic-ის Net ვერსიების გამოჩენის შემდეგ, ეს ენა ითვლება მძლავრ და იმავდროულად შესასწავლად მარტივ დაპროგრამების საშუალებად. მისი გამოყენებით შესაძლებელია დაინეროს ნებისმიერი სირთულის სამომხმარებლო პროგრამები, გააჩნია ინტერნეტში დაპროგრამების მძლავრი საშუალებები. Visual Basic-ის სიმარტივე ითვლება მის ერთ-ერთ მნიშვნელოვან ღირსებად, რის გამოც დღეისათვის პროგრამისტების უმრავლესობა დამწყებთათვის შესასწავლად სწორედ ამ ენას უწევს რეკომენდაციას. ნიგნში განხილულია Visual Basic.Net-ის ძირითადი კონსტრუქციები, ვიზუალური და ობიექტებზე ორიენტირებული დაპროგრამების მეთოდები.

თეორიული მასალის გამყარება შესაძლებელია მრავალი კონკრეტული პროექტის საფუძველზე. განხილულია სახალისო პროექტებიც: მარტივი თამაშების შექმნა და სხვ.

ნიგნს თან ერთვის CD დისკი, რომელშიც ჩანერილია მასში განხილული ყველა პროექტი.

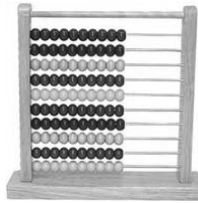
ზვიად ლურწკაია

სარჩევი

კომპიუტერის და დაპროგრამების მოკლე ისტორია	6
დაპროგრამების ენები	14
VISUAL BASIC.NET	14
ჩემი პირველი პროექტი	29
ახალი პროექტის შექმნა	33
ვიზუალური დაპროგრამების საფუძვლები	36
ახალი პროექტის შექმნა	37
პროექტის შენახვა	40
დანართის შესრულება	
მოვლენების დამუშავება	41
მოქმედებები ობიექტებზე	43
მოქმედებები ფორმაზე	46
დაპროგრამების საფუძვლები	55
მუშაობა მონაცემებთან, ცვლადები და კონსტანტები	
პროექტი "დროის ფორმატი	70
მასივები	77
პროგრამული კოდის გაფორმება	81
პროგრამული მოდულები	82
პროცედურები	81
მართვადი კონსტრუქციები და ციკლები	90
შედარების ოპერატორები	90
პირობითი კონსტრუქცია if ...Then, If ... Then ...Else	92
პირობითი კონსტრუქცია Select Case	96
პროექტი "ფერები"	98
პროექტი "შუქნიშანი"	105
პროექტი "ევკლიდეს ალგორითმი"	112
პროექტი "კვადრატული განტოლება"	116
პროექტი "კალკულატორი"	118
ციკლები	131
ციკლი For . . . Next	131
ციკლი Do . . . Loop	135
პროექტი "ევკლიდეს ალგორითმი" (ციკლის გამოყენებით)	137
ოპერატორი Exit	141
ოპერატორი Continue	142
კონსტრუქცია With ...End With	142
მუშაობა მაუსთან და კლავიატურასთან	145

პროექტი “კლავიშების კოდები”	147
პროექტი “ბრძანებები კლავიშებიდან”	148
პროექტი “მაუსის ლილაკები”	152
შემთხვევითი რიცხვები	155
თამაში “ჩაფიქრებული რიცხვის გამოცნობა”	156
თამაში “კამათელი”	161
მუშაობა გრაფიკასთან	169
პროექტი “მშვილდოსანი”	177
ობიექტებზე ორიენტირებული დაპროგრამება	185
კლასები და ობიექტები	186
კლასის ობიექტების მასივი	195
ინკაფსულაცია	198
მემკვიდრეობითობა	198
პოლიმორფიზმი	202
ვიზუალური კლასის შექმნა	205
კლასების დიაგრამა	211
თამაში “დაჭერობანა”	212
მონაცემთა ბაზები	231
მონაცემთა ბაზა “სტუდენტები”	233
რეპორტი	246
Visual Basic და ინტერნეტი	255
საკუთარი ბრაუზერი	256
ვებ-გვერდის შექმნა	260
ჩემი პირველი ვებ-გვერდი	261
თამაში “ჩაფიქრებული რიცხვის გამოცნობა” ინტერნეტში	266
დინამიური ვებ-გვერდი, მონაცემთა ბაზები	267
პროგრამის საინსტალაციო პაკეტის შექმნა	274
რეკომენდირებული ლიტერატურა	292

კომპიუტერის და დაპროგრამების მოკლე ისტორია



ადამიანი უხსოვარი დროიდან ოცნებობდა შეექმნა მანქანა, რომელიც ავტომატურად შეასრულებდა სასურველ გამოთვლებს. ინფორმაციები პირველი კომპიუტერის შესახებ ურთიერთგამომრიცხავია. მასზე პრეტენზიას სხვადასხვა გამომთვლელი საშუალებები აცხადებენ.

ავტომატური გამომთვლელი მანქანები შეიძლება დავყოთ ორ ჯგუფად, არაპროგრამირებადი გამომთვლელი მანქანები (ასრულებენ მხოლოდ მარტივ მათემატიკურ ოპერაციებს, მაგ: კალკულატორები) და პროგრამირებადი გამომთვლელი საშუალებები. პირველი კომპიუტერები მხოლოდ გარკვეულ არითმეტიკულ ოპერაციებს ასრულებდნენ და მექანიკურ მანქანებს წარმოადგენდნენ (მაგ. პასკალის ამჯამავე).

პირველ, შესაძლებელია ითქვას პროგრამირებად მექანიკურ გამომთვლელ საშუალებად, შეიძლება ჩაითვალოს ფრანგი გამომგონებლის ჟოზეფ მარი ჟაკარის (Joseph Marie Jacquard, 1801) გამომთვლელი მანქანა.

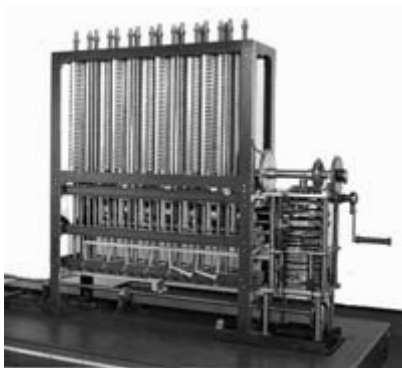


პასკალის ამჯამავე მანქანა.

ინგლისელი მათემატიკოსის, ჩარლზ ბებიჯის (Charles Babbage, 1837) მექანიკურ გამომთვლელ მანქანას, შეეძლო გარკვეული განტოლებების ამოხსნა და იყო სრულად პროგრამირებადი.



ჟოზეფ მარი შაკარის გამომთვლელი მანქანა.



ჩარლზ ბაბეჯის მექანიკური გამომთვლელი მანქანა.

პირველი რეალურად მომუშავე ელექტრონული კომპიუტერის შექმნა დაკავშირებულია კონრად ცუზეს სახელთან. მან შექმნა ელექტრონული მანქანა, რომელზეც შესაძლებელი იყო დაპროგრამება. გერმანელ მეცნიერს ჯერ კიდევ სტუდენტობის პერიოდში გაუჩნდა იდეა შეექმნა პროგრამირებადი გამომთვლელი საშუალება.



კონრად ცუზე (1910–1995)

ცუზემ ჩაატარა მრავალი ექსპერიმენტი თვლის ათობითი სისტემის გამოყენებით და საბოლოოდ მივიდა დასკვნამდე, რომ ამ მიზნისათვის ოპტიმალური თვლის ორობითი სისტემა იყო. 1938 წელს გაჩნდა მისი რეალურად მომუშავე გამომთვლელი მანქანა Z1, ეს იყო ორობითი გამომთვლელი შეზღუდული შესაძლებლობებით. მონაცემების შეყვანა შეიძლებოდა კლავიატურის საშუალებით. შედეგები აისახებოდა ნათურებიან პანელზე. ეს იყო ექსპერიმენტალური მოდელი და პრაქტიკული მიზნებისათვის არ გამოიყენებოდა.

ცუზემ 1940 წელს შექმნა მოდელი Z2, რომელიც დაფუძნებული იყო სატელეფონო რელეებზე, ამ მოწყობილობას შეეძლო ინფორმაციის წაკითხვა პერფორირებული 35 მმ-იანი კონოფირიდან. ეს უკანასკნელიც იყო სადემონსტრაციო ვარიანტი და არ გამოიყენებოდა პრაქტიკული მიზნებისათვის.

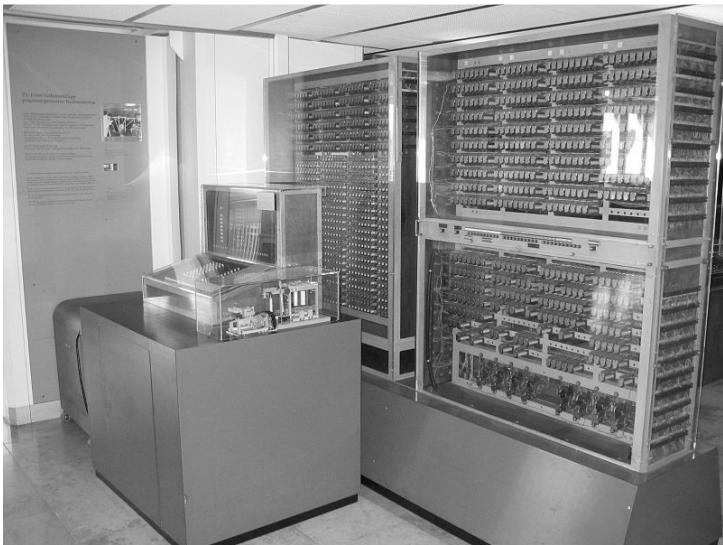
1941 წელს მან შექმნა მოდელი Z3, რომელსაც დღეს მეცნიერთა უმრავლესობა თვლის პირველ, რეალურად მომუშავე პროგრამირებად კომპიუტერად. თუმცა მასშიც დაპროგრამების შესაძლებლობები შეზღუდული იყო, არ არსებობდა პირობითი გადასვლები და ციკლები. Z3-მა ნახა პრაქტიკული გამოყენება

სამხედრო მიზნებისათვის. სამივე მოდელი განადგურდა ბერლინის დაბომბვებისას 1944 წელს.

ცუხემ ასევე შექმნა დაპროგრამების პირველი მაღალი დონის ენა და მას უწოდა Plankalkülam (გერმანულად გეგმების გამოთვლა).

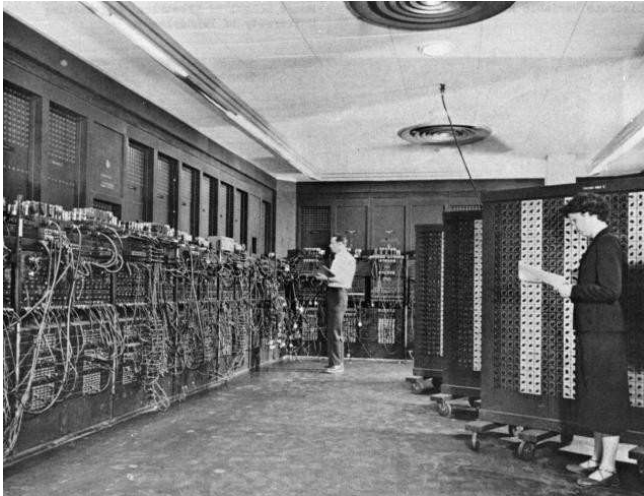
1950 წელს შექმნა მისი კომპიუტერი Z4, ეს იყო მსოფლიოში პირველი კომპიუტერი რომელიც გაიყიდა. ცუხეს კომპანიის მიერ შემდგომში შეიქმნა მრავალი კომპიუტერი. ყოველი მათგანის დასახელება იწყებოდა Z აბრევიატურით. ველაზე მეტად ცნობილია Z11 და Z22.

დღეისათვის ფუნქციონერებადი აღდგენილი Z1 მოდელი ინახება “ბერლინის ტექნოლოგიური მუზეუმში”. ხოლო Z3 “გერმანულ მუზეუმში” მიუხეხნში.

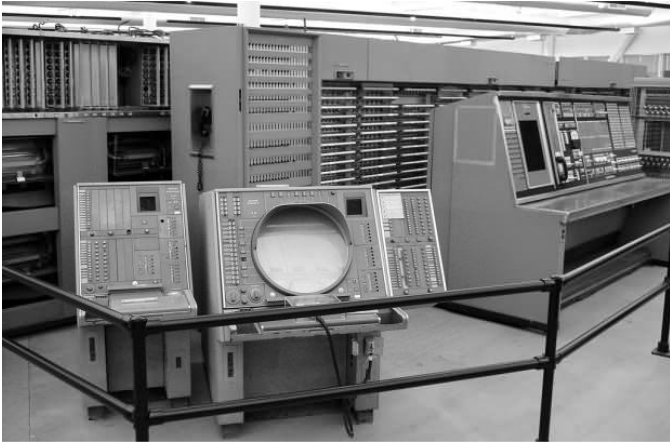


კომრად ცუხეს ცომპიუტერი Z3.

1944 წელს მოლჩიმ და ეკერტმა გადაწყვიტეს შეექმნათ ახალი კომპიუტერი, რომელსაც ექნებოდა პროგრამის საკუთარ მეხსიერებაში შენახვის საშუალება. 1945 წელს მათ შეუერთდა ცნობილი მათემატიკოსი ჯონ ფონ ნეიმანი. ნეიმანმა გააკეთა მოხსენება ასეთი კომპიუტერის ფუნქციონირების ძირითადი პრინციპების შესახებ, რამაც დიდი პოპულარობა მოუტანა. დღევანდელი კომპიუტერების უმრავლესობა სწორედ ამ პრინციპებზეა დაფუძნებული.



ერთ-ერთი პირველი კომპიუტერი ENIAC 44 (lectronic Numerical Integrator and Computer). აშშ-ში ექსპლუატაციაში შევიდა 1946 წელს. გამოიყენებოდა სამხედრო მიზნებისათვის ბალისტიკური ტრაექტორიის გამოსათვლელად. მოხმარებული ძაბვა 150 კBT, მასსა 27 ტონა.



SAGE, 1954. IBM's SAGE (Semi-Automatic Ground Environment). ის გამოიყენებოდა აშშ-ში ცივი ომის პერიოდში. საბჭოთა კავშირის რაკეტებისაგან თავდასაცავად. მასსა 300 ტონა, ღირებულება 10 მილიარდი დოლარი. მასზე მუშაობდა დაახლოებით 200 პროგრამისტი.

40-50-იან წლებში კომპიუტერები იქმნებოდა ელექტრული მილაკების საფუძველზე. ამიტომ მათი ზომები ძალიან დიდი იყო, რამოდენიმე დარბაზი ეკავათ, ღირებულება კი რამოდენიმე მილიონით განისაზღვრებოდა.

1948 წელს ტრანზისტორის გამოგონებამ სიტუაცია მნიშვნელოვნად შეცვალა. ტრანზისტორების ფასი დაბალი იყო და ზომებიც მინიატურული. მათმა გამოყენებამ კომპიუტერების ზომები ათეულჯერ შეამცირა, ასევე შეამცირა მათი ფასი და საიმედოობა. პირველი კომპიუტერები ტრანზისტორებზე გაჩნდა 50-იანი წლების ბოლოს. 1965 წელს კი კომპანია Digital Equipment გამოუშვა პირველი მინი კომპიუტერი, რომლის ზომები დაახლოებით საყოფაცხოვრებო მაცივრის ზომებს შეესაბამებოდა და ღირებულება მხოლოდ 20 ათას დოლარს შეადგენდა.

1959 წელს რობერტ ნოისმა (შემდგომში კომპანია Intel-ის დამაარსებელი) გამოიგონა მეთოდი, რომელიც საშუალებას იძლეოდა ერთ ფირფიტაზე მოთავსებულიყო ტრანზისტორები და მათ შორის ყველა აუცილებელი შეერთებები. ასეთ სქემებს

ინტეგრალური სქემები ან ჩიპები ეწოდათ. 1968 წელს კომპანია Burroughs-მა გამოუშვა პირველი კომპიუტერი ინტეგრალურ სქემებზე. 1970 წელს Intel-მა დაიწყო ინტეგრალური სქემების გაყიდვა. შემდგომ წლებში თანდათანობით იზრდებოდა ფართობის ერთეულზე მოთავსებული ტრანზისტორების რაოდენობა.

ინტეგრალური სქემების ზომები ძალიან მცირე იყო. მათმა გამოყენებამ მნიშვნელოვნად შეამცირა კომპიუტერების ზომები, შეამცირა ღირებულება და გაზარდა საიმედოობის ხარისხი.

1974 წელს რამოდენიმე კომპანიამ გამოაცხადა Intel-8008 მიკროპროცესორის ბაზაზე პერსონალური კომპიუტერის შექმნის შესახებ (ანუ კომპიუტერი რომელიც გათვლილი იყო ერთ მომხმარებელზე).

1975 წელს გაჩნდა პირველი კომერციული პერსონალური კომპიუტერი "ალტაირ 8800", რომელიც დახლოებით 500 დოლარი ღირდა. რატემაუნდა მისი შესაძლებლობები ძალიან შეზღუდული იყო, ოპერაციული მეხსიერება მხოლოდ 256 ბაიტს შეადგენდა, არ გააჩნდა კლავიატურა და ეკრანი.



ალტაირ 8800 (Altair 8800)-ის სარეკლამო განცხადება.

1975 წლის ბოლოს პოლ ალენმა და ბილ გეიტსმა (შემდგომში კომპანია Microsoft-ის დამაარსებლები) შექმნეს ამ



ბილ გეიტსი და პოლ ალენი.

კომპიუტერისათვის Basic-ის ინტერპრეტატორი. რამაც გაამარტივა პროგრამების დაწერის პროცედურა. “ალტაირ 8800“-ის წარმატებამ გამოიწვია ის, რომ მრავალი კომპანია დაინტერესდა პერსონალური კომპიუტერების წარმოებით, მალე გაჩნდა კლავიატურა და დისკლეი.

1980 წლის ბოლოს პატარა ჯგუფმა Entry Systems Division, რომელიც შეიქმნა კომპანია IBM-ში (თავიდან შტატი მხოლოდ 12 თანამშრომლისგან შედგებოდა) მიიღო დავალება შეექმნათ IBM-ის პირველი პერსონალური კომპიუტერი. ისინი თვლიდნენ, რომ აქამდე შექმნილი გამომთვლელი მანქანები არ წარმოადგენდნენ ნამდვილ კომპიუტერებს.

IBM-მა გააფორმა კონტრაქტი, ოპერაციული სისტემების შექმნაზე, იმ დროისათვის პატარა კომპანია Microsoft-თან.

IBM-ის პირველი პერსონალური კომპიუტერი გამოვიდა 1981 წელს. მას შემდეგ კომპანიამ დაიკავა წამყვანი ადგილი კომპიუტერების ინდუსტრიაში.

დაპროგრამების ენები



50-იანი წლებიდან მოყოლებული, მას შემდეგ რაც პირველი მაღალი დონის დაპროგრამების ენები შეიქმნა, განუწყვეტილად იხვეწება ამ უკანასკნელთა დამუშავებისა და რეალიზაციის მეთოდები.

პირველი ვერსიები ენებისა FORTRAN და LISP 50-იან წლებში გაჩნდა. ისტორია ენებისა C, Pascal, Prolog და Smalltalk 70-იანი წლებიდან იწყება. 80-იან წლებში გამოჩნდა ისეთი ენები, როგორებიცაა C++ , ML, Perl, Postscript. Java მათ შორის ყველაზე ახალგაზრდაა, იგი 90-იან წლებში შეიქმნა. 60-70-იან წლებში ახალი ენები ხშირად დიდი პროექტების დასამუშავებლად ჩნდებოდა.

დაპროგრამების ენების რიცხვი ძალიან დიდია. 70-იან წლებში Ada-ს დამუშავების პროექტის ფარგლებში, ამერიკის თავდაცვის სამინისტროს მიერ განხორციელდა ანგარიში იმ დროისათვის გამოყენებული დაპროგრამების ენების შესახებ, გაირკვა, რომ სხვადასხვა თავდაცვით პროექტებში 500-ზე მეტი ენა გამოიყენებოდა.

ენები რიცხვითი გამოთვლებისათვის

კომპიუტერული ტექნოლოგიის განვითარების პირველი ეტაპი მიეკუთვნება პერიოდს, რომელიც დაიწყო მეორე მსოფლიო ომამდე და გაგრძელდა 40-ანი წლების დასაწყისამდე. მეორე მსოფლიო ომის დროს, კომპიუტერების მთავარი ამოცანა

(მათ ენოდებოდა ელექტრონული გამოთვლითი მოწყობილობები), იყო ის, რომ დიფერენციალური განტოლებების ამოხსნით განესაზღვრათ ჭურვების ბალისტიკური ტრაექტორია.

პირველ კომპიუტერებთან ერთად გაჩნდა მოთხოვნილება კომპიუტერულ პროგრამებზე, რომლებიც ამ კომპიუტერებზე იმუშავებდნენ. კომპიუტერი პროგრამას ასრულებს როცა ის მას მიენოდება მანქანურ ენაზე (ორობით კოდში). ორობით, რვაობით და თექვსმეტობით კოდებში პროგრამის დაწერა რთულია. სწორედ ამიტომ, გაჩნდა მოთხოვნილება შექმნილიყო სპეციალური დაპროგრამების ენები, რომლებზეც პროგრამის დაწერა შეიძლება უფრო მარტივად და შემდეგ ეს პროგრამა კომპილატორით გადაიყვანება მანქანურ ენაზე. პირველი კომპიუტერული პროგრამები წარმოადგენდნენ სწორედ დაპროგრამების ენებს.

პირველი ცნობილი დაპროგრამების ენები გაჩნდა 50-ანი წლების დასაწყისში. გრეის ჰუპერი (Grace Hooper), რომელიც ხელმძღვანელობდა კომპანია Univac ჯგუფს, დაამუშავა ენა A-O. ხოლო ჯონ ბეკუსმა (John Backus) IBM 701-მანქანისათვის შექმნა ენა Speedcoding. ამ ორი ენის დანიშნულება იყო მარტივი არითმეტიკული გამოსახულებების გარდაქმნა.

ნამდვილი გარღვევა მოხდა 1957 წელს, როცა IBM-ის თანამშრომლებმა ჯონ ბეკუსის ხელმძღვანელობით, შექმნეს ალგორითმული ენა FORTRAN (FORmula TRANslator-ფორმულების გარდამქმნელი).

თავდაპირველად FORTRAN ორინტირებული იყი რიცხვით გამოთვლებზე, ხოლო საბოლოოდ მივიღეთ ბოლომდე დასრულებული დაპროგრამების ენა, რომელიც მოიცავს მართვის სტრუქტურას, პირობით და შეტანა-გამოტანის ოპერატორებს. მან კონკურენცია გაუწია ასამბლერს.

FORTRAN გამოდგა “ილბლიანი” ენა და დომინირებდა თითქმის 70-ათიან წლებამდე. ის გამოიყენებოდა სამეცნიერო ტექნიკური და საინჟინრო ამოცანების გადასაწყვეტად. 1958 წელს გამოვიდა ახალი ვერსია FORTRAN II, რამოდენიმე წლის შემდეგ გამოჩნდა FORTRAN IV.

ეგმ-ის (ელექტრონული გამოთვლელი მანქანა) ყოველი მწარმოებელი, თავისი კომპიუტერებისთვის ენის საკუთარ

ვერსიას ქმნიდა, ამიტომ წარმოიქმნა ქაოსი—გაჩნდა სტანდარტიზაციის აუცილებლობა.

1966 წელს FORTRAN IV სტანდარტული გახდა, მას ეწოდებოდა FORTRAN 66. ორჯერ გადახედვის შემდეგ გაჩნდა სტანდარტები FORTRAN 77 და FORTRAN 90.

ნინა ვერსიის ენებზე დაწერილი უამრვი პროგრამის არსებობა, იყო იმის მიზეზი, რომ შესაქმნელ ტრანსლატორებს “უკუთავსდებადობის” მოთხოვნა უნდა დაეკმაყოფილებინათ, ეს კი ხელს უშლიდა ენაში ახალი იდეების და კონცეპციების დანერგვას.

რადგანაც FORTRAN გახდა წარმატებული ენა, ევროპაში გაჩნდა იმის შიში, რომ IBM კომპიუტერულ სფეროში დომინანტი გახდებოდა. გამოყენებითი მათემატიკის გერმანულმა საზოგადოებამ (German society applied mathematics-GAMM) შექმნა უნივერსალური ენის დამუშავების კომიტეტი. იმავე დროს მსგავსი კომიტეტი შექმნა აშშ-ში (Association for Computing Machinery).

მიუხედავად იმისა, რომ ევროპელებს ჰქონდათ ამერიკელების ბატონობის შიში, ეს ორი კომიტეტი მაინც გაერთიანდა. პიტერ ნაურის (Peter Naur) ხელმძღვანელობით ამ კომიტეტმა შექმნა ენა IAL (International Algorithmic Language). მაგრამ მას შემდეგ ოფიციალური სახელი IAL შემდგომში ALGOL 58-ით იქნა შეცვლილი. შემდგომი ვერსია ALGOL 60 (შექმნა 1962 წელს) 60-იანი წლებიდან 70-ანების დასაწყისამდე იყო სტანდარტული აკადემიური დაპროგრამების ენა. ALGOR 60 მრავალი ენისათვის კონცეპტუალური საფუძველი გახდა.

მიუხედავად იმისა, რომ ALGOL-ს ევროპაში გარკვეული წარმატება ქონდა, მან ამერიკაში კომერციულ წარმატებას ვერ მიაღწია, მაგრამ მისი გავლენა სხვა ენებზე საკმარისად დიდი იყო. მაგალითად მოვიყვანოთ შვარცის (Jules Schwarts) მიერ შექმნილი IAL-JOVIAL ენის ვერსია, System Development Corporation (CDS)-ში დამუშავებული ეს ენა გამოიყენება აშშ-ის სამხედრო-საჰაერო ძალებში (გამოყენებითი ამოცანების ამოსახსნელად).

კომპანია Burroughs და კომპანია Sperry Univac შერწყმისას შექმნილმა კომპანიამ Unisys-მა პოლონელი მათემატიკოსის,

ლუკაშევიჩის (Lukasiewicz) ნამუშევრები აღმოაჩინა. მათემატიკოსს დამუშავებული ქონდა მეთოდიკა, რომელსაც შეუძლია არითმეტიკული გამოსახულების ჩანერა ფჩხილების გარეშე. ამ მეთოდიკამ კომპილატორების დამუშავების თეორიაზე დიდი გავლენა მოახდინა. მისი გამოყენებით კომპანია Burroughs-მა, რეალიზება გაუკეთა ALGOL-ის კომპილატორს, რომელსაც იმ დროს არსებულ FORTRAN-ის კომპილატორზე ბევრად მეტი სიჩქარე ქონდა.

ამის შემდეგ სიტუაცია იცვლება. 60-ან წლებში გაჩნდა სამომხმარებლო ტიპის ამოცანების გადაწყვეტის მოთხოვნა, რაც არც FORTRAN-ში და არც ALGOL-ში არ იყო შესაძლებელი.

1963 წელს IBM-მა თავის ლაბორატორიაში, ახალი ენა NPL (New Programmig Language) შექმნა. შემდეგ მას გადაერქვა სახელი MPPL (Multy Purpose Programming Language). ეს სახელწოდება კვლავ შეიცვალა PL/I-ით. ენა PL/I-მა გააერთიანა FORTRAN-ის გამოთვლითი და COBOL ენის ბიზნეს-პროგრამირების შესაძლებლობები (საქმიანი ინფორმაციის დამუშავება). 70-იან წლებში PL/I ენა პოპულარული იყო, დღეს ის თითქმის დავინწყებულია. PL/I შეცვალეს ისეთმა ენებმა როგორცაა C, C++ და Ada.

დაპროგრამების ენა C შეიქმნა 1972 წელს დენის რიჩის (Dennis Ritchie) და კენ ტომპსონის მიერ (Ken Thompson). სტილით ის გავს ALGOL-ს და Pascal-ს. ასევე იყენებს PL/I-ის თვისებებს. მიუხედავად იმისა, რომ C წარმოადგენს დაპროგრამების უნივერსალურ ენას, კომპაქტურმა სინტაქსმა და მასზე დაწერილი პროგრამების შესრულების ეფექტურობამ, ის ასევე აქცია პოპულარულ, სისტემური დაპროგრამების ენად.

ენა Simula-67-მ, რომელიც შექმნა ნორვეგელმა ჰაიგარდმა (Nugaard) და დალომ (Dahl), Seitana კლასის კონცეპცია ALGOL-ში. 80 წლებში ამან უბიძგა სტრაუსტრუსს (Stroustrup) რომ შეექმნა C-ის გაფართოება C++, რომელშიც დამატებული იყო კლასები. 60 წლებში ვირტ-მა (Wirth) დაამუშავა ALGOL-ის გაფართოება ALGOL-W, რომელმაც ნაკლებ წარმატებას მიღწია. მიუხედავად ამისა 70-ან წლებში მანვე შექმნა დაპროგრამების ენა Pascal, რომელიც იმ წლებში სამეცნიერო დაპროგრამების ენა გახდა.

მეორე კომიტეტმა, რომელიც ორიენტირებული იყო ALGOL 60-ის წარმატებაზე, ენა ALGOL 68 დაამუშავა, რომელიც ძალიან რთულად გასაგები გამოდგა.

70-იან წლებში შეიქმნა დაპროგრამების ენა BASIC. მისი შექმნის მიზეზი იყო ის, რომ დაპროგრამება ხელმისაწვდომი გაეხადათ უფრო ფართო საზოგადოებისათვის (გამოთვლებისათვის რომელსაც კავშირი არ ქონდა მეცნიერებასთან), მაგრამ შემდგომში მისი შესაძლებლობები მნიშვნელოვნად გაიზარდა.

სახელწოდება Basic წარმოდგება აბრევიატურისაგან: Beginner's All-Purpose Symbolic Instruction Code (უნივერსალური სიმბოლური კოდი დამწყებთათვის). ეს ენა შემუშავებულ იქნა სასწავლო მიზნით. მართალია ის იდეალური იყო დაპროგრამების სწრაფი შესწავლისათვის, მაგრამ პროფესიონალ პროგრამისტებს შორის ვერ მოიპოვა პოპულარობა, რადგანაც არ იყო სწრაფი და მოხერხებული. პროგრამისტები, რომლებიც მუშაობდნენ C ან FORTRAN-ზე, ამ პროგრამას „მავნე ბავშვურ სათამაშოს“ უწოდებდნენ, რადგანაც ის არ ეხმარებოდა დამწყებ პროგრამისტებს გამოემუშავებინათ პროგრამის სტრუქტურის შემუშავების თვისებები.

მაგრამ მას შემდეგ Basic მნიშვნელოვნად შეიცვალა დაწყებული Microsoft Quick Basic-იდან, ის გადაიქცა მარტივი ენიდან, რომელიც გამოსადეგი იყო მხოლოდ სტუდენტებისათვის და მოყვარულებისათვის, მძლავრ, მოხერხებულ და ეფექტურ დაპროგრამების საშუალებად, რომლითაც შესაძლებელია სრულფასოვანი კომერციული და სამეცნიერო სირთულის პროექტების შექმნა. ამასთან ერთად მან შეინარჩუნა თავისი სიმარტივე და „მომხიბვლელობა“.

ასეთია თანამედროვე Basic რომელიც საფუძვლად უდევს პროგრამირების გარემოს „Visual Basic“. ახალი საუკუნის დასაწყისში, Visual Basic.NET-ის შექმნის შემდეგ, ის წარმოადგენს მძლავრ, ობიექტებზე ორიენტირებულ დაპროგრამების ენას.

საქმიანი ინფორმაციის დამუშავების ენები

1955 წელს Univac-ის თანამშრომელთა ჯგუფმა, რომელსაც ხელმძღვანელობდა გრეის ჰუპერი (Grace Hooper), დაამუშავა ენა FLOWMATIC, მიზანი იყო შეექმნათ დანართი საქმიანი ინფორმაციის დამუშავებისათვის, სადაც გამოყენებული იქნებოდა ინგლისურის ენის მსგავსი ტექსტი.

1969 წელს შეიქმნა ახალი ენა COBOL (Common Business Oriented Language).

შემდეგ მოხდა მისი გადახედვა და სტანდარტიზაცია (1974 და 1984 წლებში მასში კვლავ შეიტანეს ცვლილებები).

ხელოვნური ინტელექტის ენები

ამ ტიპის დაპროგრამების ენებისადმი ინტერესი გაჩნდა 50-იან წლებში, როდესაც კომპანიამ Rand Corporation შექმნა ენა IPL (Information Processing Language). ვერსია IPL-V გახდა ცნობადი, მაგრამ მისი გამოყენება შეზღუდული იყო, რადგან IPL-V არ წარმოადგენდა მაღალი დონის ენას.

ჯონ მაკ-კარტის (John McCarthy) და მასაჩუსეტის ტექნოლოგიური ინსტიტუტის (MIT) თანამშრომლების მიერ, წინ გადადგმული, დიდი ნაბიჯი იყო LISP-ის შექმნა (LISt Processing) კომპიუტერისათვის IBM 704. LISP-ის განვითარება დღემდე მიმდინარეობს.

LISP იქმნებოდა, როგორც სიების დამუშავების ფუნქციონალური ენა. პროგრამას, რომელიც დანერილია LISP-ზე შეუძლია შექმნას “ხე”, რომელსაც აქვს შესაძლებლობები განსაზღვროს მოძრაობის მიმართულებება და ამ ხეზე მოძრაობით ოპტიმალურ სტრატეგიას ეძებს.

ხელოვნური ინტელექტის ენები გამოიყენება თამაშების შექმნისათვის, ტექსტის სათარგმნ პროგრამებში, საექსპერტო სისტემებში და სხვ.

ასეთი ენების კონცეფცია განსხვავდება ალგორითმული ენების კონცეფციისაგან. ალგორითმი მკაცრად განსაზღვრული პროცესია. რადგან ინტელექტი ასეთი პროცესი არ არის,

ხელოვნური ინტელექტის ენებს უწევთ განსაკუთრებული მიდგომების გამოყენება. პროგრამებში სადაც გამოყენებულია ხელოვნური ინტელექტი, იქმნება შთაბეჭდილება, რომ ის შესრულების პროცესში ფიქრობს როგორც ადამიანი და როგორც ადამიანი რათქმაუნდა შეცდომებსაც უშვებს. მაგ: თამაშები სადაც მოთამაშე ეთამაშება კომპიუტერს, თუ კომპიუტერმა შეცდომა არასოდეს არ დაუშვა თამაში აზრს დაკარგავს, მოთამაშე მუდამ დამარცხებული იქნება. ასევე ვისაც შეხება ქონია ტექსტის სათარგმნ პროგრამებთან იცის, რომ მათ მიერ ნათარგმნი ტექსტი ყოველთვის კორექტული არ არის. ამ შემთხვევაში კორექტულობა აუცილებელია, მაგრამ ჯერჯერობით ხელოვნური ინტელექტის ოპტიმალური რეალიზაცია მიღწეული არ არის.

რამდენი დაპროგრამების ენა უნდა ვისწავლოთ?

არიან პროგრამისტები, რომლებიც რამოდენიმე დაპროგრამების ენას იყენებენ, ზოგიერთი პროგრამისტი კი, ერთი ან ორი ენით შემოიფარგლება. რა სარგებელს ღებულობს პროგრამისტი, თუ მან ისწავლა მრავალი სხვადასხვა ენა, რომელიც შეიძლება არც კი გამოიყენოს?

ამ ყველაფერს აზრი იმ შემთხვევაში აქვს, როცა ენების შესაძლებლობებს ზედაპირულად არ განვიხილავთ, არამედ ვეცდებით ჩანვდეთ მათ კონცეფციას. ამ შემთხვევაში მყისიერად ჩნდება ექვსი ძირითადი მოტივი:

1. შეგიძლიათ დაამუშავოთ უფრო ეფექტური ალგორითმი.

ენების უმრავლესობა, სწორად გამოყენების შემთხვევაში, პროგრამისტს შესაძლებლობას აძლევს მიიღოს სარგებელი, ხოლო არასწორად გამოყენებამ, შეიძლება მიიყვანოს კომპიუტერული დროის დიდ დანაკარგამდე ან პროგრამაში ლოგიკურ შეცდომამდე, რომლის გამოსწორება ძალებისა და დროის დიდ ხარჯებთან იქნება დაკავშირებული.

ის პროგრამისტიც კი, რომელიც განსაზღვრულ ენას მრავალი წლის განმავლობაში იყენებს, შეიძლება მის ყველა შესაძლებლობას ვერ ჩანვდეს. ლიტერატურაში პროგრამირების ახალი მეთოდების აღწერა მუდმივად მიმდინარეობს. იმისათვის, რომ საუკეთესოდ გამოვიყენოთ ობიექტებზე ორიენტირებული, ლოგიკური, ან პარალელური დაპროგრამების კონცეფციები, საჭიროა იმ კონკრეტული ენების ცოდნა, სადაც ეს კონცეპციები ხორციელდება.

ახალმა ტექნოლოგიებმა, როგორცაა ინტერნეტი და გლობალური ქსელი (**World Wide Web**), ძირფესვიანად შეცვალეს დაპროგრამების ბუნება. დაპროგრამების იმ მეთოდების შექმნა, რომელიც ამ ახალ პირობებს ოპტიმალურად უპასუხებს, ენების არსის ღრმა ცოდნას მოითხოვს.

2. თქვენ შეგიძლიათ უფრო ეფექტურად გამოიყენოთ ის დაპროგრამების ენა, რომლითაც ჩვეულებრივ სარგებლობთ.

უფრო ეფექტური პროგრამა შეგიძლიათ დაწეროთ მაშინ, როცა გაიგებთ, თუ როგორ არის რეალიზებული ესა თუ ის შესაძლებლობა თქვენ მიერ გამოყენებულ ენებში. მაგალითად, იმის გაგება თუ როგორ იქმნება მასივები, სტრიქონები, სიები ან ჩანაწერები და როგორ მიმდინარეობს მოცემულ ენაში მათი დამუშავება. რეკურსიების რეალიზაციის კარგად ცოდნა, ან იმის გაგება თუ როგორ უნდა ავაგოთ ობიექტების კლასები, საშუალებას მოგცემთ, ამ კომპონენტების დახმარებით უფრო ეფექტური პროგრამები მიიღოთ.

2. თქვენ შეავსებთ სასარგებლო პროგრამული კონსტრუქციების ნაკრებს.

ენების როლი აზროვნებაში ორმაგია, რადგანაც ენა ერთდროულად ეხმარება აზროვნებას და იმავდროულად ზღუდავს კიდეც მას. საერთოდ ადამიანები ენებს აზრის გამოსახატავად იყენებენ, მაგრამ ენა აზროვნების საშუალებას მხოლოდ იმ დონემდე იძლევა, სანამდეც აზრების სიტყვიერი გამოსატყვის საშუალება არსებობს. პროგრამირების მხოლოდ ერთი ენის ცოდნა მსგავს შეზღუდვას იწვევს. ზოგიერთი

ამოცანის ამოხსნისათვის საჭირო მეთოდებისა და ხერხების ძიებისას, ადამიანი მხოლოდ იმ დონემდე აზროვნებს, სანამდეც მისთვის ცნობილ ენაში შეუძლია რეალიზება.

დაპროგრამების სხვადასხვა ენაში არსებული კონსტრუქციების შესწავლით, პროგრამისტი იმდიდრებს ლექსიკონს. განსაკუთრებით მნიშვნელოვანია კონსტრუქციების სხვადასხვა ენაში განახორციელების გაგება. იმისათვის, რომ გამოყენებულ იქნას ზოგიერთი ისეთი კონსტრუქცია, რომელიც მოცემულ ენაში არ არის წარმოდგენილი, პროგრამისტს მოუწევს მოახდინოს მისი რეალიზაცია ამ ენის ბაზისური ელემენტების ტერმინებით.

4. თქვენ შეგიძლიათ კონკრეტული პროექტის რეალიზაციისათვის მაქსიმალურად მისაღები დაპროგრამების ენა აირჩიოთ.

ასე შეამცირებთ შესასრულებელი სამუშაოს მოცულობას. პროგრამები სადაც დიდი მოცულობით რიცხვების გაანგარიშება სრულდება, მარტივად შეიძლება დამუშავდეს ისეთ ენებზე როგორცაა **C, Pascal**.

პროგრამები, რომლებიც გადანყვეტილებების მიღებისათვის გამოიყენება, მაგ. ხელოვნური ინტელექტის შემთხვევაში, უკეთესია დაინეროს ენებზე: **Lisp, ML** ან **PROLOG**.

ინტერნეტ-დანართებისთვის უფრო **Perl, Java, VB.NET** გამოიყენება.

ენების მთავარი განსაკუთრებულობების: მათი ღირსებისა და ნაკლოვანებების ცოდნა, პროგრამისტს ფართო არჩევანის საშუალებას აძლევს.

5. თქვენთვის ახალი ენების შესწავლა უფრო მარტივია იქნება.

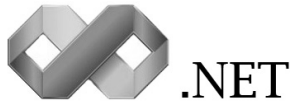
ლინგვისტს, რომელსაც ღრმად ესმის სამეტყველო ენის სტრუქტურის საფუძველი, დამწყებთან შედარებით სწრაფად შეუძლია შეისწავლოს ახალი უცხო ენა.

ასევე პროგრამისტიც, რომელიც იცნობს დაპროგრამების ენის გამოყენების ხერხებს, სწრაფად შეისწავლის ახალ დაპროგრამების ენას.

6. თქვენ შეძლებთ ახალი დაპროგრამების ენის შექმნას!

პროგრამისტების უმრავლესობა, არც კი გაიფიქრებს ახალი ენის შექმნაზე, თუნდაც მათ მიერ შესაქმნელი პროგრამა, სინამდვილეში წარმოადგენდეს დაპროგრამების ენის რალაც ფორმას. დიდი პროგრამების ინტერფეისის შემქმნელები, ისეთებისა, როგორებიცაა: ტექსტური რედაქტორი, ოპერაციული სისტემა ან გრაფიკული პაკეტი, აუცილებლად აწყდებიან მრავალ ისეთ პრობლემას, რომლებიც ზოგადი დანიშნულების პროგრამირების ენის დამუშავების დროსაც გვხვდება. ამოცანა მარტივდება თუ პროგრამისტი კარგად იცნობს სხვადასხვა კონსტრუქციებსა და მათი რეალიზაციის ხერხებს.

სინამდვილეში ენების შესაძლებლობების მსგავსება მაცდურია. ორი სხვადასხვა ენის ერთი და იგივე შესაძლებლობა შეიძლება აბსოლუტურად განსხვავებული ხერხით განხორციელდეს და თითოეულის გამოყენების ღირებულება ძალიან განსხვავდებოდეს. მაგალითად თითქმის ყველა ენაში შეკრების ოპერაცია რეალიზებულია, როგორც ერთ-ერთი ძირითადი ოპერაცია, მაგრამ სხვადასხვა ენაში, შეკრების ოპერაციის კონსტრუქცია განსხვავდება (მაგ. Basic-ში $a+b$ Lisp-ში იქნება $+(a,b)$).



აბრევიატურა **RAD** ძირითადად გამოიყენება როგორც შემოკლებული ვარიანტი სიტყვის **radical**, მაგრამ, დღეისათვის როდესაც ხდება ინფორმაციული ტექნოლოგიების რევოლუციური ცვლილებები, მაშინ როდესაც ჩაფიქრებული პროექტი უნდა იყოს მზად უკვე “გუშინ”, **RAD** აღნიშნავს **rapid application development** (პროექტის სწრაფი შექმნა).

Visual Basic იყო პირველი **RAD** ინსტრუმენტი ოპერაციულ სისტემა **Windows**-ისათვის. დღეისათვის ეს პროგრამა საყოველთაოდ აღიარებულა თუ საუბარი ეხება პროექტის სწრაფად შექმნას.

Visual Basic არის ვიზუალური დაპროგრამების გარემო. ვიზუალური დაპროგრამება ნიშნავს ვიზუალურ ინტერფეისს პლიუს პროგრამული კოდი. **Visual Basic** არის ვიზუალური ინტერფეისი პლიუს **Basic** კოდი. **Visual Basic** იყო პირველი ვიზუალური დაპროგრამების ენა.

Visual Basic.NET-ეს არის ახალი და მძლავრი დაპროგრამების ენა შექმნილი **Microsoft**-ის მიერ ათასწლეულის დასაწყისში. მას შენარჩუნებული აქვს წინამორბედის **Visual Basic 6.0**-ის მრავალი უპირატესობა და სიმარტივე. ის წარმოადგენს ობიექტებზე ორიენტირებულ დაპროგრამების ენას. **Visual Basic.Net** არის მძლავრი ენა ტრადიციული დაპროგრამების სფეროში, მაგრამ დროის გამოძახილში მან შეიძინა ახალი მიმართულება—პროგრამირება ქსელში. ის საშუალებას გვაძლევს დაინეროს ნებისმიერი პროგრამა, როგორც პერსონალური კომპიუტერებისათვის, ასევე ქსელისათვის, მობილური ტელეფონებისათვის და სხვ.

Visual Basic.NET არის საიმედო იარაღი პროფესიონალის ხელში, არის მოსახერხებელი და სიმძლავრის მიუხედავად საკმაოდ მარტივი. ის ყველა ენაზე უფრო მოსახერხებელია დაპროგრამების შესწავლისათვის.

ვისთვის არის განკუთვნილი ეს წიგნი?

წიგნი განკუთვნილია დამწყები პროგრამისტებისათვის, მეტიც ისეთებისათვის რომლებსაც არასდროს არ ქონიათ შეხება დაპროგრამებასთან. საკმარისია კომპიუტერის მინიმალური ცოდნა. ასევე მათთვის, ვისაც სურს გაიღრმავოს თავისი ცოდნა Visual Basic.NET-ში.

.NET Framework

Visual Basic .NET არ წარმოადგენს ცალკეულ პროექტს. ის შექმნილია კომპანია Microsoft-ის ახალი იდეოლოგიის ჩარჩოებში. ამ იდეოლოგიამ მიიღო სახელწოდება NET. მისი არსი მდგომარეობს იმაში, რომ დაპროგრამება ნელ-ნელა გადაადგილდება პერსონალური კომპიუტერიდან—ქსელში (Net ქსელი).

ამიტომ საჭიროა მეტი ყურადღება დაეთმოს ინტერნეტში დაპროგრამებას. კომპანიამ შექმნა პროგრამული უზრუნველყოფის შესაბამისი კომპლექსი, რომელსაც ეწოდება Net platform. Net platform-ის მნიშვნელოვან ნაწილს წარმოადგენს პროგრამული კომპლექსი .net Framework. თვენ ვერ განახორციელებთ დაპროგრამებას VB.Net გარემოში თუ კომპიუტერში ჩანერილი არ არის ეს პლატფორმა. მისი ძირითადი ნაწილებია:

კლასების ბიბლიოთეკა .NET Framework

კლასები წარმოადგენს პროგრამის “საშენ მასალას”. ისინი მრავლად არის ბიბლიოთეკაში (საკმარისია ნებისმიერი სირთულის პროგრამის შექმნისათვის).

Common Language Runtime (CLR)

ეს არის .NET Framework-ის ნაწილი, რომელიც მართავს თქვენი პროგრამის შესრულებას და უზრუნველყოფს მისი შესრულების საიმედოებას და უსაფრთხოებას. თქვენ ვერ შეასრულებთ პროგრამას სხვა კომპიუტერზე თუ მასზე არ არის დაყენებული CLR. ის შედის Windows-ის ბოლო ვერსიებში, ამიტომ მისი ჩაწერის აუცილებლობა არ არსებობს.

ინსტრუმენტები

.NET პლატფორმაზე პროგრამირებისათვის აუცილებელია ინსტრუმენტები— დაპროგრამების ენები:

- **Visual Basic .NET**
- **Visual C++ .NET**
- **Visual C# .NET**
- **Visual J# .NET**

ზემოთმოყვანილი დაპროგრამების ენები შედის ერთ პროგრამაში, რომელსაც ეწოდება **Visual Studio.NET, Visual Studio 2003, 2005, 2008, 2010** (ჩვენ პროექტებში გამოვიყენებთ Visual Studio 2008-ს). თუ თქვენ ჩაწერთ კომპიუტერში ერთ-ერთ ამ პაკეტს, თქვენ შეგიძლიათ დაპროგრამება ერთ-ერთ ზემოთ მოყვანილ ენაზე, მეტიც, თქვენ შეგიძლიათ დაპროგრამება ყველა მათგანზე ერთდროულად.

რატომ Visual Basic და არა C++?

რატომ Visual Basic? ეს შეკითხვა ყოვლთვის აქვთ ახალბედა პროგრამისტებს, რომლებმაც არ იციან რომელი დაპროგრამების ენა შეისწავლონ თავდაპირველად. აქ მოვიყვანთ ვარიანტების მოკლე მიმოხილვას:

დღეისათვის ყველაზე ცნობილი დაპროგრამების ენებია, **C, C++ java, Pascal, Basic.NET**. ისინი განსხვავდებიან თავიანთი წინამორბედებისაგან, მიუხედავად იმისა რომ წარმოადგენენ მათ

პირდაპირ შთამომავლებს, რადგან იყენებენ **NET Framework**-ის მექანიზმებს.

წინამორბედისაგან ყველაზე მეტად განსხვავდება **Visual basic.NET**. რაც შეეხება განსხვავებებს ენებს შორის, ისინი შენარჩუნებულია მათ **NET** ვერსიებშიც.

Java ახალგაზრდა ენაა და შექმნილია ინტერნეტში დაპროგრამებისათვის, ძირითადად სწორედ ამ მიზნით გამოიყენება. თუ ჩვენ გვინდა დაპროგრამება არა მხოლოდ ინტერნეტში, მაშინ ჯობია ეს ენა დროებით გადავდოთ.

რაც შეეხება **C++**-ს, ეს არის მაღალპროფესიონალური დაპროგრამების ენა და ის პროგრამისტებს შორის ფართოდაა გავრცელებული. ამ ენაზე დაწერილ პროგრამებს აქვთ დიდი სწრაფმოქმედება, მაგრამ ის ძალიან რთულად აღსაქმელია დაწვეები პროგრამისტებისათვის, სწორედ ამიტომ მისგან დაპროგრამების სწავლების დანყება არ არის რეკომენდირებული. თუ გვინდა შევისწავლოთ ეს ენა, ჯობია დავიწყოთ **Visual basic**-ით და მერე გადავიდეთ **C++**-ზე. ასეთი მიდგომა გავგიადვილებს მის ათვისებას.

პროფესიონალების აზრით ყველაზე მარტივი და იოლად გამოსაყენებადი სწორედ **Visual Basic**-ია. პროგრამისტებს ის უყვართ, რადგან მასზე პროექტის შექმნაზე იხარჯება გაცილებით ნაკლები დრო ვიდრე **C**-ზე და **C++**-ზე.

Visual Basic-ის ნაკლს წარმოადგენს ის, რომ მასზე დაწერილი პროგრამები მუშაობს შედარებით ნელა ვიდრე **C++**-ზე, მაგრამ დაწვეები პროგრამისტი ამას ვერც კი შეამჩნევს, ამას ასევე ხელს უწყობს თანამედროვე პერსონალური კომპიუტერების სწრაფმოქმედება.

Pascal-ს უკავია შუამდებარე პოზიცია **C**-ისა და **Basic**-ს შორის. დღეისათვის მასზე დაპროგრამება წარმოებს **Delphi**-ს გარემოში. პასკალი არ შედის **Visual studio**-ს დაპროგრამების ენებში. **Pascal**-ის გამოყენება ეფექტურია სამეცნიერო-გამოთვლითი პროექტების შესაქმნელად.

საბოლოო არჩევანი მკითხველის პრეროგატივაა.

რატომ **Visual Basic.NET** და არა ძველი **Visual Basic**?

რატომ არის რეკომენდირებული შევისწავლოთ **Visual Basic.NET** და არა მისი სხვა ძველი ვერსია, მაგალითად ყველაზე პოპულარული და NET-ის წინამორბედი **Visual Basic 6.0**?

- იმიტომ რომ, ეს არის **Visual basic**-ის ყველაზე მძლავრი ვერსია, ის გვაძლევს მრავალ ისეთ შესაძლებლობებს, რომლებიც არ იყო ხელმისაწვდომი წინა ვერსიებში.
- რადგან ეს არის **visual basic**-ის პირველი ობიექტებზე ორიენტირებული ვერსია. ამხრივ ის ეწევა Delph-ს და C++-ს.
- ძველი **Visual basic** მოხმარებაში იქნება მხოლოდ რამოდენიმე წელიწადს, შემდეგ ის გზას დაუთმობს NET ვერსიებს.

ჩემი პირველი პროექტი



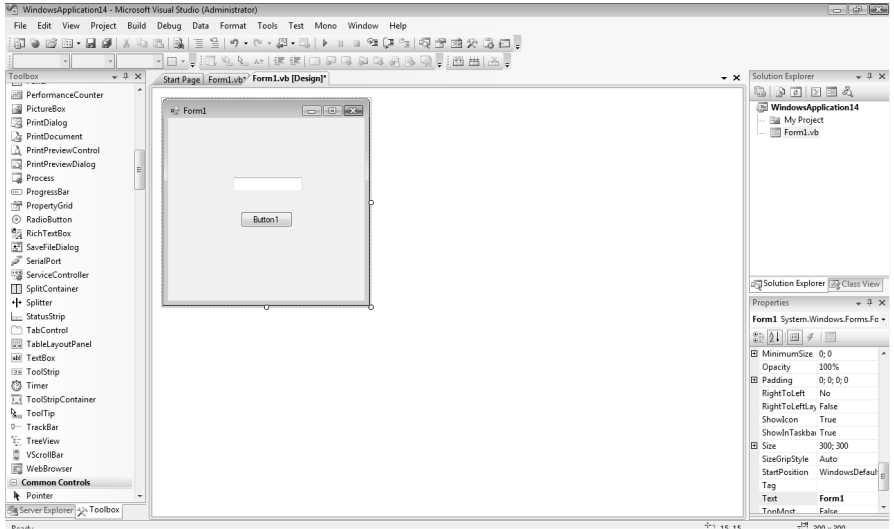
აღბათ ვერ ითმენთ, პირდაპირ გსურთ პრაქტიკაზე გადასვლა და სასურველი პროგრამების დაწერა. აუცილებელი თეორიული ცოდნის გარეშე ამის გაკეთება ძალიან რთულია, მაგრამ მოდით ყოველგვარი თეორიული ნაწილის გარეშე შევექმნათ პირველი უმარტივესი პროექტი, რომელიც მართალია ზედაპირულად, მაგრამ მაინც შეგიქმნით წარმოდგენას ამ მძლავრი დაპროგრამების საშუალების შესაძლებლობებზე და შედარებით სიმარტივეზე.

Visual Basic-ში პროექტი ეწოდება ფაილების ერთობლიობას, რომელიც შედის დანართში (პროგრამაში) და ინახავს ინფორმაციას მის კომპონენტებზე. ანუ ჩვენს მიერ შესაქმნელი პროგრამა არის ჩვენს მიერ შესაქმნელი პროექტი.

შევექმნათ ახალი პროექტი:

- გაუშვით პროგრამა **Visual Basic 2008** (ან 2010).
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project**.
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანექით ღილაკს **New Project**.
- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Windows Forms Application (Windows დანართი)**. დავრწმუნდეთ რომ გრაფაში **Project Types** მონიშნულია **Visual Basic**.
- შემდეგ დააჭირეთ **OK** ღილაკს.

მთავარ ფანჯარაში გაიხსნება ახალი პროექტი, რომელიც თავდაპირველად შეიცავს ცარიელ ფორმას. უკვე შეიძლება მუშაობის დაწყება: უჩუმრად დაყენებული თვისებების შეცვლა, მასში მართვის ელემენტების მოთავსება და სხვ.



ცარიელ ფორმაზე ელემენტთა პანელიდან **Toolbox** (განლაგებულია ეკრანის მარცხენა მხარეს) გადმოვიტანოთ ელემენტები: ტექსტური ბლოკი **TextBox** და ლილაკი **Button**. თუ ვერ ვხედავთ ელემენტთა პანელს, მისი გამოტანად ავირჩიოთ ბმული **Toolbox**, მენიუთა სტრიქონიდან **View**.

დავაკლიკოთ ორჯერ ლილაკზე. გაიხსნება კოდის შეყვანის ფანჯარა. აქ უკვე დაგვხვდება კოდის ნაწილი. იქ სადაც ციმციმებს კურსორი ჩავწეროთ შემდეგი კოდი:

`TextBox1.Text = "ჩემი პირველი პროექტი"`

```
Start Page Form1.vb* Form1.vb [Design]*
Button1 Click
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        TextBox1.Text = "ჩემი პირველი პროექტი"
    End Sub
End Class
```

კლავიატურაზე დავაჭიროთ ღილაკს <F5>. ჩვენი პირველი პროექტი გაეშვება. დავაჭიროთ ღილაკს გაშვებული პროგრამის ფორმაზე. ტექსტურ ბლოკში განდება ნარწერა “ჩემი პირველი პროექტი” 😊.



იმაზე თუ რა მოხდა და როგორ, დაწვრილებით მოგვიანებით ვისაუბრებთ. ახლა კი მცირედი დრო აუცილებელ თეორიულ ნაწილს დავუთმობთ.

ვიზუალური დაპროგრამების საფუძვლები

სანამ უშუალოდ გადავიდოდეთ პრაქტიკაზე, გარკვეული დრო დავუთმობთ ყველაზე აუცილებელ თეორიულ ნაწილს, რომლის გარეშეც საკუთარი პროგრამის დანერგა (კარგი პროექტის შექმნა) შეუძლებელი იქნება. ამ თავში განიხილება **Visual Basic**-ის ძირითადი ელემენტები, რომელსაც იყენებენ პროექტის შექმნისას. მაშ ასე:

რას ნიშნავს ვიზუალური ინტერფეისი?

ვიზუალური ინტერფეისი არის პროგრამის სახე, ანუ თუ როგორ გამოიყურება ვიზუალურად პროგრამა მისი გაშვებისას, მაგ: საყოველთაო ცნობილი ტექსტური რედაქტორი Word, ან ცხრილების გარემო Exel და სხვ.

პროგრამის გაშვებისას მათ აქვთ გარკვეული ვიზუალური სახე (შემდგომში ინტერფეისი), რომელიც დამახასიათებელია მხოლოდ ამ პროგრამებისთვის.

პროგრამისტები მივიდნენ იმ დასკვნამდე, რომ მართალია სხვადასხვა პროგრამა მნიშვნელოვნად განსხვავდება ერთმანეთისაგან ინტერფეისით, მაგრამ მათ ყველას გააჩნია საერთო ელემენტები. მაგ: ინტერფეისით ძალიან განსხვავებულია ტექსტური რედაქტორი Word და 3D Studio Max (სამგანზომილებიანი გრაფიკისა და ანიმაციის პროგრამა), ასევე Photo Shop (გრაფიკული რედაქტორი), მაგრამ მათ აქვთ მსგავსი ელემენტები. მაგ: ლილაკები, მენიუთა პანელი, ტექსტური ბლოკები და სხვ. აქედან გამომდინარე დაისვა საკითხი ვიზუალური დაპროგრამების გარემოს შექმნის შესახებ.

დღეისათვის Visual Basic-ის და სხვა ვიზუალური დაპროგრამების გარემოს საშუალებით შესაძლებელია შევქმნათ პროგრამის ინტერფეისი ისე, რომ არ დავწეროთ მისი კოდი (ინტერფეისის კოდი). ელემენტთა პანელში არსებული მართვის ელემენტები შეიძლება გადმოვიტანოთ პროექტის ფორმაზე, შევქმნათ პროექტის ინტერფეისი და შემდეგ შევუდგეთ

პროგრამის კოდის წერას (ანუ თუ რა უნდა შეასრულოს პროგრამის ამა თუ იმ ელემენტმა).

ასეთი გარემოს არსებობამდე პროგრამისტებს უნევდათ ძალიან დიდი კოდის დაწერა იმისათვის, რომ შეექმნათ პროგრამის ინტერფეისი. ვიზუალური დაპროგრამების გარემოდღეს ამას ჩვენს მაგივრად გააკეთებს.

ახალი პროექტის შექმნა

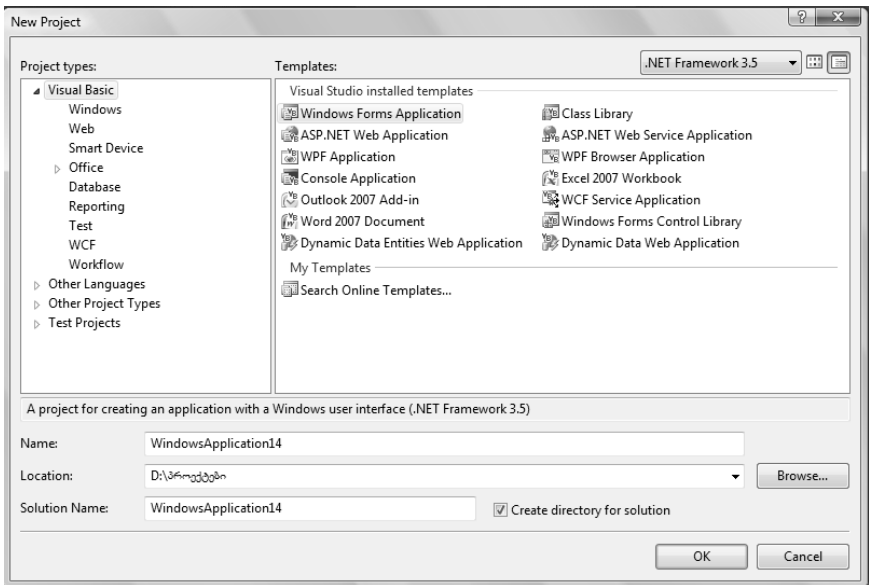
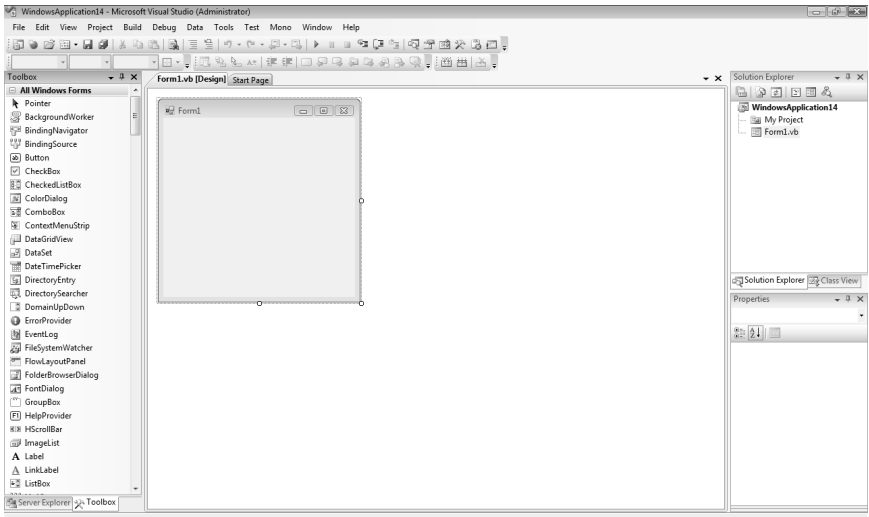
Visual Basic-ში პროექტი ეწოდება ფაილების ერთობლიობას, რომელიც შედის დანართში (პროგრამაში) და ინახავს ინფორმაციას მის კომპონენტებზე. ანუ ჩვენს მიერ შესაქმნელი პროგრამა არის ჩვენს მიერ შესაქმნელი პროექტი.

შევექმნათ ახალი პროექტი:

- გაუშვით პროგრამა **Visual Basic 2008** (ან 2010).
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project**.
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანექით ლილაკს **New Project**.
- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Windows Forms Application (Windows დანართი)**. დავრწმუნდეთ რომ გრაფაში **Project Types** მონიშნულია **Visual Basic**.
- ველში **Name** ჩაწერეთ შესაქმნელი პროექტის სახელი.
- შეგვიძლია ჩამოვშალოთ სია, სადაც ჩამოთვლილია **NET Framework**-ის ვერსიები და მივუთითოთ სასურველი ვერსია.
- შემდეგ დააჭირეთ **OK** ლილაკს.

მთავარ ფანჯარაში გაიხსნება ახალი პროექტი, რომელიც თავდაპირველად შეიცავს ცარიელ ფორმას. უკვე შეიძლება

მუშაობის დაწყება: უჩუმრად დაყენებული თვისებების შეცვლა, მასში მართვის ელემენტების მოთავსება და სხვ.



ფორმის შექმნა

როცა შევექმნით **Visual Basic**-ის ახალ პროექტს, შეიქმნება ახალი ცარიელი ფორმა, ამის შემდეგ იწყება დანართის პროექტირება.



Visual Basic-ის პროექტებში ფორმა შედგება ობიექტებისაგან, რომლებსაც მართვის ელემენტებს უწოდებენ (მაგ: ლილაკი, ტექსტური ბლოკი და სხვ.). ყველა მართვის ელემენტს გააჩნია თავისი დამახასიათებელი თვისება (მაგ: ფერი, ზომა, წარწერა და სხვ.).

ნებისმიერ ობიექტს შეიძლება მივანიჭოთ მოქმედება, რომელსაც ასრულებს პროგრამა მაშინ, როდესაც მოხდება გარკვეული მოვლენა—ანუ პროგრამა რეაგირებს მოვლენებზე.

ფორმის შექმნის პროცესი მოიცავს ფორმაში ობიექტების განთავსებას, მათთვის განსაზღვრული თვისებების მინიჭებას (ვიზუალური ინტერფეისის შექმნა) და მათში პროგრამული კოდის ჩანერას, რომელიც შესრულდება კონკრეტული მოვლენის არსებობისას (მოვლენა შეიძლება იყოს: ფორმის ლილაკზე დაკლიკება, ფორმის ლილაკზე თავგის მარჯვენა ლილაკით დაკლიკება, ტექსტის ცვლილება და სხვ.).

ფორმაში ობიექტის განთავსებისათვის გამოიყენება მართვის ელემენტების პანელი **Toolbox** (ეკრანის მარცხენა მხარეს).

თუ ეკრანზე **Toolbox** გამოტანილი არ არის, შეასრულეთ შემდეგი მოქმედებებიდან ერთ-ერთი:

- მენიუდან **View** აირჩიეთ ბრძანება **Toolbox**.
- დააჭირეთ ლილაკს **Toolbox** ინსტრუმენტების სტანდარტულ პანელზე.

ფორმაში მართვის ელემენტების განთავსება ხორციელდება შემდეგი სახით:

- მონიშნეთ საჭირო მართვის ელემენტი (რომელიც განთავსებულია პანელზე **Toolbox**), დააჭირეთ თავის მარცხენა ლილაკს.
- არ აუშვათ მაუსის ლილაკს, გადმოიტანეთ მართვის ელემენტი ფორმაზე და მოხაზეთ მისი საჭირო ზომა (ოთხკუთხედი).
- აუშვით ხელი მაუსის ლილაკს.

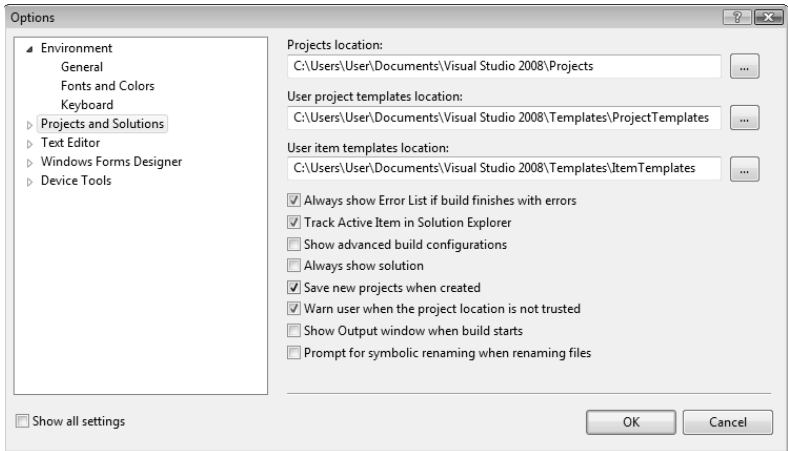
პროექტის შენახვა

Visual Basic 2008-ში პროექტის შენახვა ხდება დიალოგის ფანჯარის **Save Project** დახმარებით, რომელიც იხსნება ბრძანებით **Save All**, მენიუდან **File**, ან სტანდარტული ინსტრუმენტების პანელიდან შესაბამისი ლილაკით.

იმისათვის, რომ შემდგომ ობიექტს შევუცვალოთ სახელი, საჭიროა დიალოგის ფანჯარაში **Solution Explorer** ამოვარჩიოთ შესაბამისი ფაილი ან პროექტი. შემდეგ თვისებათა ფანჯარაში **Properties** დავარქვათ ახალი სახელი **File Name**, **Project Name**.

ასევე შეიძლება პროექტის შენახვა მისი შექმნისთანავე. ამისათვის აუცილებელია შეიცვალოს შესაბამისი რეგულირება,

მენიუ **Tools**-ის ფანჯარაში ავირჩიოთ **Options**. შემდეგ განყოფილებაში **Projects and Solutions** მიუთითეთ მონიშვნა **Save new projects when created** (ახალი პროექტის შენახვა შექმნისთანავე).




თუ პროექტის შენახვა ხდება მისი შექმნისთანავე, მაშინ დიოლოგის ფანჯარაში **New Project** გაჩნდება დამატებითი ველი **Location**, სადაც მითითებულია გზა პროექტისაკენ და მისი სახელი.

დანართის შესრულება

Visual Basic 2008-ში დანართის შესასრულებლად (პროგრამის გასაშვებად) არსებობს რამოდენიმე ხერხი. გამოვიყენოთ ნებისმიერი მათგანი.

- ამოიჩიეთ მენიუ **Debug**-დან ბრძანება **Start Debugging**.

- სტანდარტული ინსტრუმენტების პანელზე დააჭირეთ ღილაკს **Start Debugging** .
- დააჭირეთ კლავიატურაზე ღილაკს <F5>.

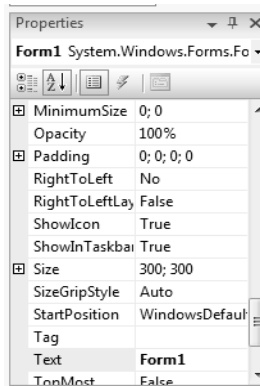
მართვის ობიექტები

Visual Basic-ში, ყველა მართვის ელემენტი (ობიექტი) განლაგებულია ფორმაზე. ფორმა, ისევე როგორც მართვის ელემენტები, ხასიათება თვისებებით, რომელიც შეიძლება შევცვალოთ როგორც პროგრამის გაშვებამდე, ასევე მისი მუშაობის პროცესში (პროგრამულად).

ფორმაზე მოთავსებული ობიექტის თვისებების დათვალიერებისა და რედაქტირებისათვის, მონიშნეთ ის, ხოლო შემდეგ შეასრულეთ ერთ-ერთი შემდეგი მოქმედება:

- მენიუ **View**-დან ამოირჩიეთ ბრძანება **Properties Window**.
- დააჭირეთ მაუსის მარჯვენა ღილაკს და კონტექსტური მენიუდან ამოირჩიეთ **Properties**.
- დააჭირეთ ღილაკს <F4>.

ამის შემდეგ ეკრანის მარჯვენა მხარეს გაიხსნება თვისებათა პანელი **Properties**.



ზედა ნაწილში ის შეიცავს ჩამოსაშლელ სიას, სადაც არის ფორმის ობიექტების ჩამონათვალი. მისი გამოყენება შეიძლება ობიექტის ასარჩევად (გამოიტანს მის თვისებებს).

სიების ქვეშ განლაგებულია ლილაკი **Categorized**, რომელიც დაალაგებს ობიექტების თვისებებს კატეგორიების მიხედვით, ხოლო ლილაკი **Alphabetical**–ანბანის მიხედვით.

დიალოგიური ფანჯარის **Properties** ქვედა ნაწილში მოთავსებულია სიაში ამორჩიული თვისების მოკლე ახსნა.

რომელიმე ობიექტის თვისების შესაცვლელად, აუცილებელია გაიხსნას **Properties** ფანჯარა და იმ სტრიქონზე გადასვლა, რომელიც შეიცავს მოცემულ თვისებას. თვისებების მნიშვნელობა განლაგებულია თვისებათა დასახელების მარჯნივ. ზოგიერთი თვისება მოყვანილია ცხრილში.

<i>თვისებათა კატეგორია</i>	<i>თვისებები</i>
Apperance	ამ კატეგორიაში განლაგებული თვისებები, განსაზღვრავს ობიექტის გარეგნულ სახეს. მაგალითად: ფორმის თვისება Text იძლევა ტექსტის შექმნის საშუალებას, რომელიც განთავსებულია ობიექტზე, ხოლო თვისება BorderStyle განსაზღვრავს ობიექტის ჩარჩოს სტილს.
Behavior	ამ კატეგორიაში განლაგებული თვისებები, განსაზღვრავენ ობიექტის ქცევას. მაგალითად, თუ ობიექტის თვისება Visible ღებულობს მნიშვნელობას False , მაშინ პროგრამის შესრულებისას ობიექტი უხილავია. მნიშვნელობა False , რომელიც მინიჭებული აქვს თვისებას Enabled , ბლოკირებს უკითხვს ობიექტს.
Data	ეს კატეგორია საშუალებას იძლევა განისაზღვროს გამოყენებული მონაცემები. მაგალითად, თვისება Table , მიუთითებს გამოყენებული ცხრილის სახელს, თვისება DataSource მონაცემების წყაროს.
Focus	მოცემული კატეგორია შეიცავს თვისებას CausesValidation . ის განსაზღვრავს, გამოიძახება თუ არა მართვის ელემენტების მოვლენა Validating და Validated .

	რომელიც განკუთვნილია შეყვანის სისწორის შესამოწმებლად და მის დასრულების შემდეგ.
Layout	ამ კატეგორიის თვისებები საშუალებას იძლევა მივუთითოთ ობიექტის მდებარეობა ფორმის მარცხენა ზედა კუთხის მიმართ, ასევე მისი ზომები.
Misc	ამ კატეგორიაში შედის სხვადასხვა სახის თვისებები. მაგალითად, თვისება AcceptButton და CancelButton რომელთა დახმარებით შეიძლება ღილაკს მიანიჭო თვისება რომ, ამუშავდეს <Enter> ან <Esc> ღილაკზე დაჭერისას.
Windows Style	ამ კატეგორიის თვისების დახმარებით შეიძლება შეიქმნას ფორმის დანართის სახე. მაგალითად, თვისებას Opacity შეუძლია ფორმას მიანიჭოს გამჭვირვალობა.

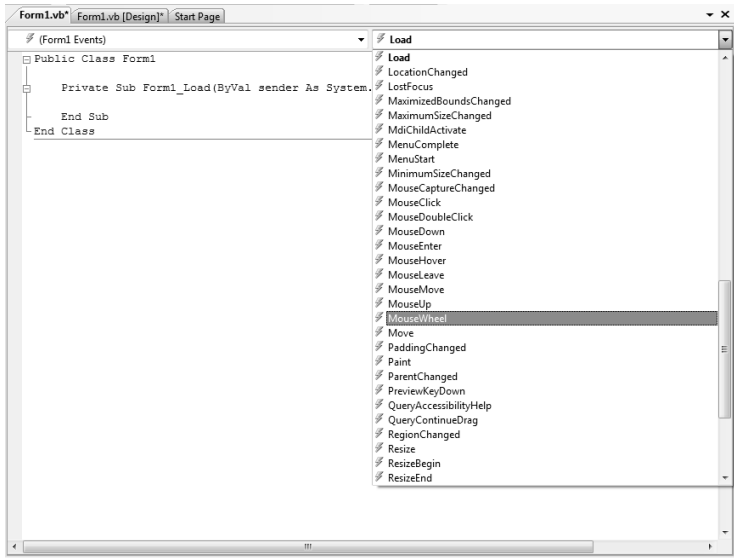
მოვლენების დამუშავება

Visual Basic.NET წარმოადგენს ობიექტებზე ორიენტირებულ დაპროგრამების ენას. თვისებათა გარდა, ობიექტისათვის შეიძლება შეიქმნას პროგრამული კოდი, დაწერილი **Basic**-ის ენაზე, რომელიც შესრულდება მაშინ, როდესაც მოხდება მასთან დადაკავშირებული მოვლენა. მაგალითად, ღილაკზე დაჭერისას, წარმოიქმნება მოვლენა **Click**. პროგრამული კოდის ჩასანერად, უნდა გავხსნათ შესაბამისი ფანჯარა, ამისათვის შევასრულოთ ერთ-ერთი შემდეგი მოქმედებებიდან:

- მაუსით დააკლიკეთ ორჯერ იმ ობიექტზე, რომლისთვისაც გინდათ შექმნათ პროგრამული კოდი.
- მენიუდან **View** ამოირჩიეთ ბრძანება **Code**.
- დააჭირეთ ღილაკს <F7>.

ნებისმიერი ამ მოქმედების შესრულების შემდეგ, გაიხსნება კოდის ფანჯარა.

ფანჯარის ზედა ნაწილში განლაგებულია ორი ჩამოსაშლელი სია: **Class Name** (კლასის სახელი) და **Method Name** (მეთოდის სახელი). მარცხენა სია **Class Name** შეიცავს ყველა ობიექტს, ფორმის ჩათვლით.



სიაში **Method Name** ჩამოთვლილია მოცემულ ობიექტთან დაკავშირებული შესაძლო მოვლენები, როდესაც სიიდან ამოირჩევა მნიშვნელობა, შეიქმნება შესაბამისი პროცედურა. პროცედურის ტექსტი თავსდება **Sub** და **End Sub** ოპერატორებს შორის.

პროცედურის შესაქმნელად უნდა შესრულდეს შემდეგი მოქმედება:

- კოდის ფანჯარის გახსნა ნებისმიერი მარტივი ხერხით.
- ჩამოსაშლელი სიიდან **Class Name** ამოირჩიეთ ობიექტი, რომლისთვისაც იქმნება პროცედურა.

- ჩამოსაშლელი სიის **Method Name** გამოყენებით, აირჩიეთ სასურველი მოვლენა.
- **Sub** და **End Sub** ოპერატორებს შორის ჩანერეთ პროცედურის ტექსტი (პროგრამული კოდი).

მოქმედებები ობიექტებზე

ფორმის შექმნის პროცესში, შეიძლება ობიექტის გადაადგილება, ნაშლა ან მისი ზომების შეცვლა.

ობიექტის ფორმის მონიშვნა

ობიექტის მართვისათვის თავდაპირველად ის უნდა მონიშნოს. ერთი ობიექტის მოსანიშნად საკმარისია დავაკლიკოთ მასზე. რამოდენიმე ობიექტის ერთდროულად მოსანიშნად შევასრულოთ ერთ-ერთი შემდეგი მოქმედებებიდან:

- დავაჭიროთ კლავიშს **<Shift>** და ხელის აულებლად მოვნიშნოთ ყველა ობიექტი.
- გამოიყენეთ ბრძანება **Select All**. მენიუდან **Edit** ან **<Ctrl>+<A>**.

ობიექტების ფორმის გათანაბრება

გარეგნული სახის გაუმჯობესებისათვის, ფორმის ობიექტებს ათანაბრებენ ერთმანეთის მიმართ. ობიექტის გათანაბრებისათვის გამოიყენება პუნქტი **Align** მენიუდან **Format**, რომელიც შეიცავს ცხრილში მოცემულ ბრძანებებს.

<i>ბრძანება</i>	<i>დანიშნულება</i>
Lefts	მონიშნული ობიექტების გათანაბრება მარცხენა მხარეს, ყველაზე მარცხნივ მდებარე ობიექტის მიმართ.
Centers	მონიშნული ობიექტების გათანაბრება,

	ვერტიკალური ღერძის მიმართ.
Rights	მონიშნული ობიექტების გათანაბრება მარჯვენა მხარეს, ყველაზე მარჯვნივ მდებარე ობიექტის მიმართ.
Tops	მონიშნული ობიექტების გათანაბრება ყველაზე ზევით მდებარე ობიექტის მიმართ.
Middles	მონიშნული ობიექტების გათანაბრება, ჰორიზონტალური ღერძის მიმართ.
Bottoms	მონიშნული ობიექტების გათანაბრება ქვედა მხარეს, ყველაზე ქვევით მდებარე ობიექტის მიმართ.
To Grid	მონიშნული ობიექტების გათანაბრება ბადის წრფის მიმართ.

მენიუ **Format** ასევე შეიცავს პუნქტს **Make Same Size**, რომლის ბრძანებებია **Width**, **Height**, **Both** და **Size to Grid**, რომლებიც საშუალებას იძლევა ამორჩეული ობიექტებისათვის დაწესდეს ერთნაირი სიგანე, სიმაღლე ან ერთდროულად ორივე ზომა. ასევე ობიექტების ზომების დაყვანა ბადის უჯრედის ზომამდე.

ამორჩეული ობიექტებს შორის მანძილის მართვისათვის ჰორიზონტალური და ვერტიკალური მიმართულებით, გამოიყენება ბრძანებები **Horizontal Spacing** და **Vertical Spacing**.

<i>ბრძანება</i>	<i>მოქმედება</i>
Make Equal	ამორჩეულ ობიექტებს შორის ერთნაირი მანძილის დაყენება
Increase	ამორჩეულ ობიექტებს შორის ზრდის მანძილს.
Decrease	ამორჩეულ ობიექტებს შორის ამცირებს მანძილს.
Remove	ამორჩეულ ობიექტებს შორის ანულებს მანძილს.

მენიუ **Format** ასევე შეიცავს ბრძანებებს, რომელიც მართავს ობიექტის ასახვას ფორმაზე.

<i>ბრძანება</i>	<i>შესრულება</i>
Bring to Front	ამორჩიულ ობიექტს მოათავსებს ფორმის ყველაზე ზედა ფენაზე.
Send to Back	ამორჩიულ ობიექტს მოათავსებს ფორმის ყველაზე ქვედა ფენაზე.

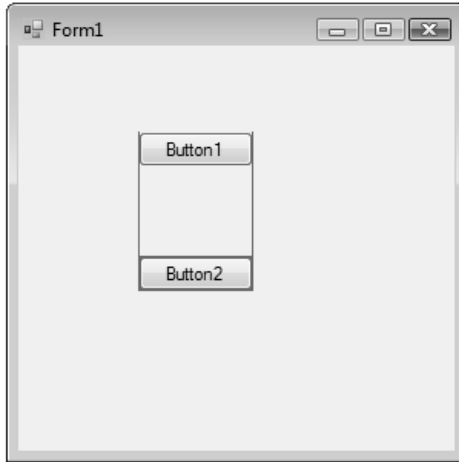
ფორმაზე ობიექტების პოზიციონირება

Visual basic 2008-ში ფორმაზე ობიექტის ზუსტი პოზიციონირებისათვის გამოიყენება ორი საშუალება: ბადის წრფე და საყრდენი წრფე.

მენიუდან **Tools** ავირჩიოთ ბრძანება **Options**, შემდეგ გავხსნათ დიალოგის ფანჯარა **Windows Forms Designer**. პარამეტრების თვისებები მოცემულია ცხრილში.

<i>თვისებები</i>	<i>მოქმედებები</i>
GridSize	იცვლება ბადის უჯრედის სიგანე და სიგრძე.
LayoutMode	შეიცავს SnapLines და SnapToGrid მნიშვნელობებს, რომელიც განსაზღვრავს საყრდენი წრფის ან ბადის წრფის გამოყენებას ფორმაზე მართვის ელემენტების ერთმანეთთან გასათანაბრებლად.
ShowGrid	მნიშვნელობა True ფორმაზე გამოიტანს ბადეს, False გააქრობს მას.
SnapToGrid	მნიშვნელობა True კრძალავს ობიექტების განლაგებას ფორმის ნებისმიერ ადგილას. ობიექტის განლაგების ყველა ოპერაცია, მათი გადაადგილება და ზომების შეცვლა, შესრულებული იქნება ბადის უჯრედის ზომის გათვალისწინებით.

სურათზე ნაჩვენებია, თუ როგორ შეიძლება საყრდენი ნრფის დახმარებით მართვის ელემენტების გათანაბრება.



მოქმედებები ფორმაზე

ფორმის შექმნის პროცესი შეიძლება დაიყოს სამ ეტაპად:

- ფორმის პარამეტრების აწყობა.
- ფორმაზე ობიექტების განლაგება.
- ობიექტებისათვის თვისებების მინიჭება.

ფორმას, როგორც ობიექტებს, რომლებიც მასზეა განლაგებული, გააჩნია თვისებები. მათი შეცვლით მას შეიძლება მიეცეს ზომა, ზედა მარცხენა კუთხის კოორდინატები, ჩარჩოს ფარგლების სტილი, სათაური, ფერი და ა.შ.

გარდა ამისა ფორმა ხასიათდება მოვლენებით და მეთოდებით.

ფორმის პარამეტრების აწყობა ხორციელდება ფანჯარაში **Properties**, მის გასახსნელად დააყენეთ კურსორი ფორმის

ობიექტისაგან თავისუფალ ზედაპირზე, შემდეგ შეასრულეთ ერთ-ერთი შემდეგი მოქმედება:

- მენიუ **View**-დან ამოირჩიეთ ბრძანება **Properties Window**.
- დააჭირეთ მაუსის მარჯვენა ღილაკს და კონტექსტური მენიუდან ამოირჩიეთ **Properties**.
- დააჭირეთ ღილაკს **<F4>**.

ფორმის მდებარეობა

ფორმის მდებარეობა ეკრანზე განისაზღვრება **X** და **Y** თვისებებით ან თვისებით **Location**, რომელიც მიუთითებს მანძილს მარცხენა ზედა კუთხიდან.

გარდა ამისა ფორმის განლაგებისათვის, გამოიყენება თვისება **StartPosition**, რომელსაც შუძლია მიიღოს ცხრილში მოყვანილი მნიშვნელობები.

<i>StartPosition თვისების მნიშვნელობა</i>	<i>ფორმის მდებარეობა</i>
Manual	ფორმის მდებარეობა განისაზღვრება თვისებით Location .
CenterScreen	ფორმა განლაგდება ეკრანის ცენტრში.
WindowsDefaultLocation	ფორმის მდებარეობა განისაზღვრება Windows სისტემით, რომელიც გამომდინარეობს გახსნილი ფანჯარების რაოდენობიდან და მათი მდებარეობიდან, ხოლო ზომა განისაზღვრება თვისებით Size .
WindowaDefaultBounds	ფორმის მდებარეობა და მისი ზომა განისაზღვრება Windows სისტემით, უჩუმრად.
CenterParent	ფორმა მდებარეობს მშობლიური ფორმის ცენტრში.

ფორმის ზომების შესაცვლელად გამოიყენება თვისება **Height** და **Width**, რომლებიც განსაზღვრავენ ფორმის სიმაღლეს და სიგანეს.

თვისებების **MinimumSize** და **MaximumSize** დახმარებით შეიძლება განისაზღვროს ფორმის ზომის მინიმუმი და მაქსიმუმი.

თვისება **WindowState** განისაზღვრავს ფორმის ზომას პროგრამის გაშვებისთანავე და მას შეუძლია მიიღოს ცხრილში მოცემული მნიშვნელობებიდან ერთ-ერთი.

<i>WindowState</i> თვისების მნიშვნელობა	აღწერა
Normal	ფორმას აქვს ზომა, განსაზღვრული მისი თვისებებით.
Minimized	ფორმა ჩაიხურება.
Maximized	ფორმა გაიშლება მთელს ეკრანზე.

ფორმის სათაური

ფორმისათვის სათაურის მისანიჭებლად (განლაგებულია ფორმის ზედა ნაწილში), განკუთვნილია მისი თვისება **Text** (არ უნდა აგვერიოს თვისებაში **Name**, რომელიც წარმოადგენს ფორმის სახელს, რითაც მას მიმართავს პროგრამა).

იმისათვის, რომ ფორმა საერთოდ არ შეიცავდეს სათაურს, საჭიროა თვისება **Text**-ის მარჯვენა სვეტში ინფორმაციის ნაშლა.

შესაძლებელია ასევე შეიცვალოს სურათი, რომელიც ფორმის სათაურის მარცხენა მხარესაა განლაგებული. ამისათვის გამოიყენება თვისება **Icon**.

ფორმის ჩარჩოში ჩასმის სტილი

ფორმის ჩარჩოში ჩასმის სტილი შეიქმნება **FormBorderStyle** თვისებით. მას შეუძლია მიიღოს მნიშვნელობები, რომელიც მოცემულია ცხრილში.

<i>თვისება FormBorderStyle</i>	
<i>თვისების მნიშვნელობა</i>	<i>აღწერა</i>
None	ფორმას არ გააჩნია ჩარჩო, სათაურის არე, სისტემური მენიუს გამოძახების ლილაკი, ფანჯრის მართვის ლილაკი.
FixedSingle	უცვლელი ერთმაგი ჩარჩო. სათაურის არეში განლაგებულია სისტემური მენიუს გამოძახების ლილაკი ნიშნის სახით. ფორმის სათაური და ფანჯრის მართვის ლილაკი.
Fixed3D	უცვლელი მოცულობითი ჩარჩო. სათაურის არეში განლაგებულია სისტემური მენიუს გამოძახების ლილაკი ნიშნის სახით. ფორმის სათაური და ფანჯრის მართვის ლილაკი.
FixedDialog	უცვლელი ერთმაგი ჩარჩო. სათაურის არეში განლაგებულია ფორმის სათაური და ფანჯრის მართვის ლილაკი.
Sizable	ცვალებადი ჩარჩო (ფორმის ზომების შეცვლა შესრულების დროს). სათაურის არეში განლაგებულია სისტემური მენიუს გამოძახების ლილაკი, ფორმის სათაური და ფანჯრის მართვის ლილაკი.
FixedToolWindow	უცვლელი ერთმაგი ჩარჩო. სათაურის არეში განლაგებულია ფორმის სათაური და ფორმის დასახური ლილაკი.
SizableToolWindow	უცვლელი ერთმაგი ჩარჩო. სათაურის არეში განლაგებულია ფორმის სათაური და ფორმის დასახური ლილაკი.

ფორმის ფონი

ფორმის ფონის ფერის მისაცემად გამოიყენება თვისება **BackColor**.

ფონისთვის ასევე შეიძლება გამოყენებულ იქნას სურათი. ამისათვის გამოიყენება თვისება **BackgroundImage**. მისი მნიშვნელობის შესაცვლელად, თვისებათა პანელზე მის გვერით, დაკლიკებისას გამოჩნდება ღილაკი სამი ნერტილით, მასზე დაჭერა გამოიძახებს დიალოგის ფანჯარას **Select Resource**. ამ დიალოგიური ფანჯარის გამოყენებით შეიძლება ამოვირჩიოთ საჭირო ნახატი.

ფონად სურათის გამოყენებისას, მისი განლაგების განსაზღვრისათვის გამოიყენება თვისება **BackgroundImageLayout**, მისი მნიშვნელობები მოცემულია ცხრილში.

BackgroundImageLayout თვისების მნიშვნელობები	
თვისების მნიშვნელობა	ფორმის მდებარეობა
None	ნახატი განლაგდება ფორმის მარცხენა ზედა კუთხეში და ინარჩუნებს თავის ზომას.
Tile	ნახატი განლაგდება ფორმაზე მოზაიკის სახით.
Center	ნახატი განლაგდება ფორმის ცენტრში.
Stretch	ნახატი გაიჭიმება ფორმის ზომამდე, პროპორციების შენარჩუნების გარეშე.
Zoom	ნახატი გაიჭიმება ფორმის ზომამდე, ინარჩუნებს პროპორციებს.

ფორმის მოვლენები

ფორმებს, ისევე როგორც მასზე განთავსებულ ობიექტებს, შეუძლიათ უპასუხონ მოვლენებს და შეასრულონ ბრძანებები. ცხრილში მოყვანილია ფორმის რამოდენიმე მოვლენა.

<i>ფორმის მოვლენები</i>	
მოვლენა	წარმოშობა
Activated	იმ მომენტში, როდესაც ფორმა ხდება აქტიური ანუ როცა აისახება ეკრანზე.
Deactivate	როდესაც ფორმა ხდება არააქტიური. მაგალითად, როდესაც ხდება მეორე ფორმის გააქტიურება.
Load	მახსოვრობაში ფორმის ჩატვირთვის მომენტში (ფორმის გაშვებისას), სანამ ის გამოჩნდება ეკრანზე.
Paint	ფორმაზე გამოსახულების ხატვის დროს.
Resize	ფორმის ზომების შეცვლის დროს.

მენიუთა სტრიქონი

Visual Basic-ში მარტივად შეიძლება შეიქმნას ფორმის მენიუთა სტრიქონი. მენიუს პროექტირებისას, უნდა ვიხელმძღვანელოთ გარკვეულ პრინციპებით. მათგან მთავარია "სტანდარტი".

რეკომენდირებულია მენიუს ბრძანებათა დასახელება და მათი განლაგება შეესაბამებოდეს მიღებულ სტანდარტებს. მაგალითად: ფაილებთან მუშაობის მენიუს პუნქტს, რეკომენდირებულია მიენიჭოს სახელი **Fail**. ასეთ შემთხვევაში თქვენ მიერ შექმნილი პროგრამა უფრო იოლად გასაგები იქნება მომხმარებლისათვის.

მენიუს რედაქტორი **Menu Editor**

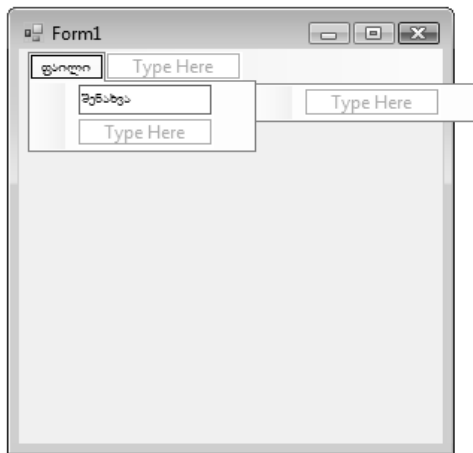
ყველა სახის მენიუს პროექტირებისათვის, გამოიყენება მენიუს რედაქტორი **Menu Editor**. მენიუს რედაქტორთან მუშაობისთვის, აუცილებელია ფორმაზე განლაგდეს შემდეგი მარჯეთის ელემენტებიდან ერთ-ერთი:

- **MenuStrip**-მართვის ელემენტი, რომელიც განკუთვნილია დანართის მთავარი მენიუს შესაქმნელად.
- **ContextMenuStrip** მართვის ელემენტი, რომელიც გამოიყენება კონტექსტური მენიუს შესაქმნელად.

MenuStrip

თუ ფორმაზე გადავიტანთ მართვის ელემენტს **MenuStrip**, ფორმის ზევით, სათაურის ქვეშ, გამოჩნდება რედაქტირების მენიუ. ის წარმოადგენს ელემენტს **Type Here**, რომელიც განკუთვნილია მენიუს პუნქტის ჩასაწერად. ტექსტის შეყვანის დროს მენიუს პუნქტში გამოჩნდება დამატებითი ელემენტი **Type Here** ქვევით და მარჯვნივ.

შესაბამის პუნქტებში ტექსტის შეყვანის შემდეგ ჩვენ შეგვიძლია მივიღოთ ნებისმიერი სირთულის მენიუთა სტრიქონი, სხვადასხვა პუნქტებით და გადასვლებით. ვფიქრობთ, რომ თქვენ მასში მარტივად გაერკვევით. რაც შეეხება მენიუს თითოეულ პუნქტში კოდის ჩანერას, ეს ისევე ხდება როგორც ფორმის ან მართვის ობიექტის შემთხვევაში. საჭიროა ორჯერ დავაკლიკოთ შესაბამის პუნქტზე და გავხსნათ პროგრამული კოდის ფანჯარა.

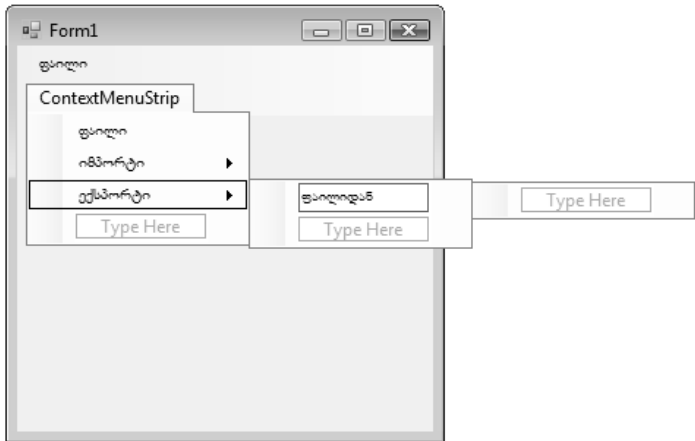


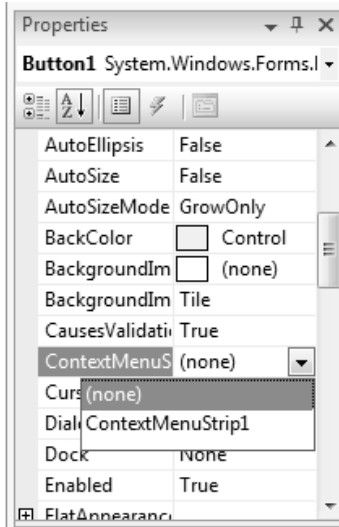
ContextMenuStrip

თუ ფორმაზე გადავიტანთ მართვის ელემენტს **ContextMenuStrip**, ფორმის ზევით, სათაურის ქვეშ, გამოჩნდება რედაქტირების მენიუ.

ის წარმოადგენს ელემენტს **Type Here**, რომელიც განკუთვნილია მენიუს პუნქტის ჩასაწერად. ტექსტის შეყვანის დროს მენიუს პუნქტში გამოჩნდება დამატებითი ელემენტი **Type Here** ქვევით და მარჯნივ (ისევე როგორც **MenuStrip**-ის შემთხვევაში).

ContextMenuStrip უნდა გამოჩნდეს მაუსის მარჯვენა ღილაკზე, ფორმაზე ან რომელიმე ელემენტზე დაკლიკებისას. შეიძლება გვექონდეს რამოდენიმე ასეთი ელემენტი. ფორმის ობიექტებზე მათ მისაბმელად უნდა მოვნიშნოთ სასურველი ობიექტი (მაგ: **Button**) და შემდეგ თვისებათა ფანჯარაში მოვნახოთ მისი თვისება **ContextMenuStrip**. მის მარჯვენა მხარეს ჩამოსაშლელ სიაში ავირჩიოთ სასურველი კონტექსტური მენიუს დასახელება.





დაპროგრამების საფუძვლები

როგორც ავლნიშნეთ ვიზუალური დაპროგრამება წარმოადგენს ვიზუალურ ინტერფეისს პლიუს პროგრამული კოდი. მას შემდეგ, რაც შექმნილია ვიზუალური ინტერფეისი უნდა ჩავწეროთ კოდი, რომელიც მას “გააცოცხლებს” ანუ ის შეასრულებს რაიმე მოქმედებას წინასწარ განსაზღვრული ალგორითმით. ალგორითმის განმარტება არის შემდეგი: ალგორითმი წარმოადგენს ლოგიკური მოქმედებების სასრულ მიმდევრობას, რომელიც საშუალებას გვაძლევს ამოვხსნათ მოცემული ამოცანა. ანუ პროგრამის დაწერამდე აუცილებელია მისი ალგორითმის განსაზღვრა. კომპიუტერით ამოცანის გადაჭრაში განასხვავებენ შემდეგ ეტაპებს, რომელიც პროგრამის დაწერის პროცესში აუცილებლად უნდა გავიაროთ”

- ამოცანის დასმა.
- ალგორითმის განსაზღვრა.
- პროგრამის დაწერა.
- პროგრამის გამართვა.

პროგრამის დაწერისას უმნიშვნელოვანესი და ყველაზე შრომატევადი პროცესია მისი ალგორითმის განსაზღვრა. თუ ალგორითმი განსაზღვრულია და კორექტულია, მისი რეალიზაცია დაპროგრამების ენის საშუალებით ძნელი არ არის.

რომელ ენაზე დაინერება პროგრამა, ამას მისი მომხმარებლისათვის ხშირად მნიშვნელობა არა აქვს, რადგანაც ის საბოლოოდ გადაიყვანება მანქანურ ენაზე და სრულიად კარგავს კავშირს იმ დაპროგრამების ენასთან, რომელზეც შეიქმნა—ანუ ბოლოს ყველა პროგრამა მუშაობს მანქანურ ენაზე.

მუშაობა მონაცემებთან, ცვლადები და კონსტანტები

ყველა პროგრამა უნდა მუშაობდეს გარკვეულ მონაცემებთან. იმის მიუხედავად ეს მონაცემები წარმოადგენს ტექსტს, გრაფიკულ გამოსახულებას თუ ციფრების ნაკრებს.

პროგრამის შესრულებისას ეს მონაცემები უნდა განთავსდეს კომპიუტერის მეხსიერებაში.

პროგრამა შესრულებისას ინახავს მონაცემებს ცვლადებისა და კონსტანტების სახით. ცვლადები (სახელიდან გამომდინარე), არის ინფორმაციის ერთეული, რომელიც შესაძლებელია იცვლებოდეს, კონსტანტები კი უცვლელი რჩება.

Visual Basic-ს გააჩნია ცვლადებისა და კონსტანტების ფართო არჩევანი, რაც დააკმაყოფილებს ნებისმიერ მოთხოვნას. ცვლადები Visual Basic-ში შესაძლებელია დაიყოს სამ კატეგორიად. რიცხვითი ცვლადები, რომლის ფუნქციაა რიცხვების შენახვა, ტექსტური ცვლადები-ტექსტური ინფორმაციის შესანახად და მესამე ტიპს მიეკუთვნება ცვლადები, რომლებიც არ შეიძლება მივაკუთვნოთ არც ტექსტურ და არც რიცხვით ცვლადებს. ყოველი კატეგორია შემდგომში იყოფა ქვეკატეგორიებად.

ცვლადები

ცვლადი წარმოადგენს კომპიუტერის ოპერატიულ მეხსიერებაში გამოყოფილ ადგილს, მონაცემების დროებითი შენახვისათვის. თითოეულ ცვლადს აქვს თავისი საკუთარი სახელი (მაგ: x, y, “სახელი”, “ასაკი”) მისი მნიშვნელობა კი შეიძლება შეიცვალოს პროგრამის მუშაობის პროცესში. მაგ x ცვლადს მიენიჭოს მნიშვნელობა “5”, y-ს “7”, სახელ-ს “გიორგი”, ასაკი-ს “15 წლის”. ცვლადის მნიშვნელობა შეიძლება შეიცვალოს მომხმარებლის მიერ ან ავტომატურად.

მას შემდეგ რაც ცვლადს მიენიჭება მნიშვნელობა, თქვენ შეგიძლიათ პროგრამაში ამ მნიშვნელობის ნაცვლად გამოიყენოთ ცვლადის სახელი.

ცვლადების სახელები

იმისათვის რომ თქვენი ცვლადი წასაკითხად უფრო თვალსაჩინო და მარტივი გახდეს, რეკომენდირებულია მას მიანიჭოთ სახელი, რომელსაც აქვს გარკვეული აზრობრივი

მნიშვნელობა. არსებობს ცვლადისათვის სახელის დარქმევის წესები:

- ცვლადის სახელი უნდა შეიცავდეს არაუმეტეს 255 სიმბოლოს.
- ცვლადის სახელი შეიძლება შეიცავდეს ნებისმიერ ასოს, ციფრს და ხაზგასმით სიმბოლოს.
- პირველი სიმბოლო ცვლადის სახელში უნდა იყოს ასო ან ხაზგასმითი სიმბოლო.
- ცვლადის სახელი არ უნდა შეიცავდეს ინტერვალს და პუნქტუაციის ნიშანს.
- სახელი უნდა იყოს უნიკალური ხედვის შიდა არეში (არ შეიძლება ხედვის შიდა არეში ორ ცვლადს დავარქვათ ერთიდაიგივე სახელი).
- სახელი არ უნდა წარმოადგენდეს **Basic**-ის საკვანძო სიტყვას, მაგალითად **Print**.

შენიშვნა:

*შეზღუდვის სია ძალიან დიდია, მისი ცოდნა ზეპირად რთულია, მაგრამ თქვენ ყოველთვის დაგეხმარებათ **Visual Studio**-ს პროგრამის სინტაქსური შესწორება.*

შეზღუდვა სახელის აზრობრივად სწორ დარქმევაში არ არსებობს, მაგრამ უმჯობესია სახელები იყოს ინფორმატიული. ეს ხელს შეუწყობს კოდის უკეთ აღქმას.

ზოგიერთი პროგრამისტი თვლის, რომ სახელი უნდა იყოს არა მარტო ინფორმატიული, მასში უნდა იყოს ინფორმაცია ცვლადების ტიპის შესახებ. ასეთ სისტემას მიეკუთვნება აღნიშვნების “უნგრული სისტემა”. მასში გამოიყენება სახელის წინ გარკვეული აღნიშვნები, მაგალითად თუ ცვლადი მიეკუთვნება ტიპს Integer, მის წინ უნდა დაინეროს აღნიშვნა i, თუ ცვლადს დავრქვით სახელი „Books” და აღნიშნავს წიგნების რაოდენობას ბიბლიოთეკაში, მაშინ ის უნდა ჩაინეროს ასე “iBooks”.

მაგრამ Visual Basic-ში “უნგრული სისტემის” გამოყენება არამიზანშეწონილია, რადგანაც Visual Basic-ის რედაქტორი საშუალებას იძლევა მყისიერად ამოიცნოს ცვლადის ნებისმიერი ტიპი. ამისათვის საკმარისია მივიყვანოთ კურსორი ცვლადის სახელთან, გამოჩნდება მენიუ სადაც არის ინფორმაცია ამ ცვლადის ტიპის შესახებ.

როცა გვაქვს ასეთი ეფექტური მეთოდი იმისათვის, რომ გავიგოთ ცვლადის ტიპი, “უნგრული სისტემის” გამოყენება მნიშვნელობას კარგავს.

ცვლადების ტიპები

Visual Basic.NET-ის ცვლადების ძირითადი ტიპები მოცემულია ცხრილში.

<i>მონაცემთა ტიპი</i>	<i>ადგილი მეხსიერებაში</i>	<i>მნიშვნელობა</i>
Boolean	სხვადასხვა	True, False
Byte	1 ბაიტი	0 დან 255
Char	2 ბაიტი	ერთი სიმბოლო კოდირებული Unicod -ით
Data	8 ბაიტი	0001 წლის 1 იანვრიდან 9999 წლის 31 დეკემბრამდე. დრო 0:00:00 დან 23:59:59სთ-მდე
Decimal	16 ბაიტი	-79228162514264337593543950335-დან +79228162514264337593543950335-მდე -7,9228162514264337593543950335-დან +7,9228162514264337593543950335-მდე
Double	8 ბაიტი	-1,79769313486231570308-დან 4,94065645841246544-324-მდე 4,94065645841246544-324-დან -1,79769313486231570308-მდე
Integer	4 ბაიტი	-2147483648-დან 2147483647-მდე

Long	8 ბაიტი	-9223372036854775808 დან 9223372036854775808 მდე
Object	სხვადასხვა	ნებისმიერი მნიშვნელობა
SByte	1 ბაიტი	-128-დან 127-მდე
Short	2 ბაიტი	-32768-დან 32768-მდე
Single	4 ბაიტი	-3,402823538 დან -1,401298-45-მდე 1,401298-45-დან 3,402823538-მდე
String	სხვადასხვა	დაახლოებით 2 მილიარდი სიმბოლო კოდირებული Unicode -ით
UInteger	4 ბაიტი	0-დან 4294967295-მდე
ULong	8 ბაიტი	0-დან 18446744073709551615-მდე
UShort	2 ბაიტი	0-დან 65535-მდე

განვმარტოთ ზოგიერთი ტიპის ცვლადი:

ლოგიკური ცვლადი

Boolean ტიპის ცვლადს, შეუძლია მიიღოს მხოლოდ ორი მნიშვნელობა: **True** და **False** (ლოგიკური ცვლადი). როდესაც რიცხვითი ცვლადი გადაიყვანება ლოგიკურში, მაშინ 0 ხდება **False**, ხოლო დანარჩენი მნიშვნელობა—**True**. როდესაც ლოგიკური ცვლადი გადადის რიცხვითში **False** ხდება 0, ხოლო **True**-1.

უჩუმრად (თავდაპირველად) **Boolean** ცვლადს მიენიჭებული აქვს მნიშვნელობა **False**.

რიცხვითი ცვლადები

დაპროგრამებაში, რიცხვებთან მუშაობა, ყველაზე ფართოდ გავრცელებული ოპერაციაა, ამდენად **Visual Basic.NET**-ს გააჩნია რიცხვითი ცვლადების მრავალი ტიპი.

რა საჭიროა რიცხვითი ცვლადების ამდენი ტიპი? რატომ არ ჩავნეროთ ყველა რიცხვითი ცვლადი ერთი ტიპის სახით? პასუხი მარტივია—საქმე ეხება ეფექტურობას.

ციფრი 5 იკავებს რა თქმა უნდა კომპიუტერის მეხსიერებაში უფო მცირე ადგილს და მუშავდება უფრო სწრაფად, ვიდრე 12324838673409568390,7878. თუ ყველა რიცხვითი მონაცემი მიეკუთვნება გარკვეულ ტიპს, ამით გაიზრდება პროგრამის ეფექტურობა და სწრაფმოქმედება.

რიცხვები იყოფა ორ ნაწილად: მთელი რიცხვები, რომლებსაც არ გააჩნია წილადი ნაწილი და რიცხვები მცურავი მძიმით (რომლებსაც შეიძლება გააჩნდეთ წილადი ნაწილი), არსებობს ცვლადები მთელი რიცხვების და ასევე რიცხვების მცურავი მძიმით წარმოსადგენად.

ყოველ მათგანს გააჩნია თავიანთი დიაპაზონი (მაქსიმალური და მინიმალური მნიშვნელობა, რომელიც შეიძლება მათ მიენიჭოთ). ამის გარდა მონაცემები მცურავი მძიმით ხასიათდებიან სიზუსტის თვისებით, ანუ რამდენი ციფრია მძიმის შემდეგ. Visual Basic.NET-ის რიცხვითი ცვლადების ტიპებს შეუძლიათ ადექვატურად ასახონ ნებისმიერი რეალური სიტუაცია.

ორობითი რიცხვის შესანახად გამოიყენება Byte ტიპის ცვლადი.

იმისათვის, რომ შევინახოთ რიცხვის მთელი ნაწილის მნიშვნელობა, ნიშნის რიცხისათვის (რომლებიც ლებულობენ როგორც დადებით ასევე უარყოფით მნიშვნელობას) გამოიყენება ცვლადები Short, Integer და Long, ხოლო უნიშნო ცვლადებისათვის გამოიყენება Ushort, UInteger და Ulong.

რიცხვებისათვის მთელი და წილადური ნაწილით, განკუთვნილია მონაცემთა ტიპები Double და Single, რომლებიც ინახავენ რიცხვს მცურავი მძიმით. მაგალითად 4,57, რაც ნიშნავს $4,5 \cdot 10^7$ ან 45 000 000. რიცხვს მცურავი მძიმით შეიძლება ქონდეს 10-ის უარყოფითი ხარისხის მაჩვენებელი, მაგალითად 4,5-4 რაც ნიშნავს $4,5 \cdot 10^{-4}$ ან 0,00045. ეს იმას ნიშნავს, რომ ასეთი ცვლადები გამოიყენება როგორც ძალიან პატარა, ისევე ძალიან დიდი სიდიდეების შესანახად.

ცვლადებს, რომლებიც გამოცხადებულია როგორც Decimal, მცურავი მძიმით რიცხვებისაგან განსხვავებით, არ გააჩნიათ ნამრავლი "ათი ხარისხად". ეს საშუალებას იძლევა აცილებულ იქნას შეცდომები დამრგვალებისას.

ტექსტური ცვლადი

ტექსტური ინფორმაციისათვის განკუთვნილია Char და Strind ტიპის ცვლადები. პირველი მათგანი ინახავს ერთ სიმბოლოს Unicode კოდირებაში, ხოლო მეორე სტრიქონს 0-დან 2 მილიარდ სიმბოლომდე (სტრიქონი ეწოდება ბრჭყალებით დაბოლოებულ სიმბოლოთა მიმდევრობას). String ტიპის ცვლადი წარმოადგენს ბმულს სტრიქონიზე.

Date ტიპის ცვლადი

Date ტიპის ცვლადი ინახავს დროისა და თარიღის მნიშვნელობას. თარიღის მნიშვნელობა უნდა იყოს მოთავსებული ნიშან “#”-ს შორის და უნდა იყოს ფორმატში “თვე/დღე/წელი/”, მაგალითად #5/31/1993#.

უჩუმრად Date ტიპის ცვლადს მინიჭებული აქვს მნიშვნელობა 12:00 1 იანვარი, 00001 წელი.

Object ტიპის ცვლადი

Object ტიპის ცვლადს შეუძლია შეინახოს ნებისმიერი ტიპის მონაცემები და პროგრამის შესრულების მომენტში ცვალოს მათი ტიპი.

ცვლადების გარდაქმნა ერთი ტიპიდან მეორეში

ცვლადების გარდაქმნა ერთი ტიპიდან მეორეში შეიძლება იყოს აშკარა და არააშკარა. გარდაქმნა სრულდება ავტომატურად როგორც კი ცვლადს მიენიჭება განსაზღვრული მნიშვნელობა.

ხოლო იმ შემთხვევაში, როცა ხდება აშკარა გარდაქმნა, მაშინ გამოიყენება **System.Convert** კლასის მეთოდი.

ცვლადის გამოცხადება

Visual Basic.NET-ში არსებობს ცვლადის ცხადი და არაცხადი გამოცხადება.

ცხადი გამოცხადება წარმოადგენს ცვლადის სახელისა და ტიპის მითითებას მისი გამოყენების წინ. ის ხორციელდება **Dim, Private, Static, public**, ოპერატორებით, რომლებსაც აქვს შემდეგი სინტაქსი:

Dim ცვლადის სახელი *As* მონაცემის ტიპი = მნიშვნელობა
Private ცვლადის სახელი *As* მონაცემის ტიპი = მნიშვნელობა
Static ცვლადის სახელი *As* მონაცემის ტიპი = მნიშვნელობა
public ცვლადის სახელი *As* მონაცემის ტიპი = მნიშვნელობა

ცვლადი, რომელიც გამოცხადებულია **Dim** ოპერატორით, მისაწვდომია პროგრამის ნებისმიერი ადგილიდან მხედველობის არის ფარგლებში, რომელიც შეიცავს ოპერატორ **Dim**-ს. მაგალითად თუ ის გამოცხადებულია მოდულის შიგნით არა ნებისმიერ პროცედურაში, მაშინ ასეთი ცვლადი მისაწვდომია ამ მოდულის ნებისმიერი ადგილიდან. თუ ცვლადი გამოცხადებულია პროცედურის შიგნით, მაშინ ის მისაწვდომია მხოლოდ ამ პროცედურის საზღვრებში. ასეთ ცვლადს ეწოდება “ლოკალური”.

ცვლადის ხელმისაწვდომობის უფრო დეტალურად განსაზღვრისათვის, გამოიყენება ოპერატორები **Private** და **Public**.

Public ოპერატორის გამოყენება ნიშნავს, რომ ცვლადს აქვს საერთო ხელმისაწვდომობა—ხელმისაწვდომობა ნებისმიერი შეზღუდვის გარეშე. **public** ცვლადი პროცედურის შიგნით არ შეიძლება იყოს გამოცხადებული.

ცვლადი რომელიც არის გამოცხადებული საკვანძო სიტყვით **Private**, მისაწვდომია მხოლოდ კონტექსტის ფარგლებში, რომელშიც არის გამოცხადებული (პროცედურების ჩათვლით). **Private** ცვლადი შეიძლება გამოვაცხადოთ მოდულის შიგნით, კლასში ან სტრუქტურაში, მაგრამ პროცედურის შიგნით არა.

თუ ცვლადის გამოცხადების დროს მითითებულია როგორც **Static**, მაშინ ის აგრძელებს არსებობას მახსოვრობაში და ინახავს თავის ბოლო მნიშვნელობას იმ პროცედურის შესრულების შემდეგ, სადაც იყო გამოცხადებული. **Static-**

ცვლადი არ შეიძლება იყოს გამოცხადებული პროცედურის გარეთ.

ერთი ოპერატორის დახმარებით შეიძლება გამოვაცხადოთ რამოდენიმე ცვლადი, რომელებიც ერთმანეთისაგან უნდა გამოიყოს მძიმით. ცვლადის გამოცხადების მაგალითები:

dim y as byte

Dim LastName, firstName As String, dblSum As Double

Private x As Boolean

ცვლადის გამოცხადების “*As მონაცემის ტიპი*” ნაწილი არ არის აუცილებელი, მაგრამ თუ მონაცემთა ტიპი არ არის მითითებული, მაშინ **Visual Basic.NET** მიაანიჭებს მას შესაბამის მნიშვნელობას.

თუ მონაცემთა ტიპი არ არის მითითებული და ცვლადი არ არის ინიცირებული არავითარი საწყისი მნიშვნელობით, მაშინ **Visual Basic.NET** მას მიაანიჭებს მონაცემთა ტიპს **Object**.

უჩუმრად **Visual Basic.NET**-ში გააქტივებულია ცვლადის გამოცხადების აშკარა რეჟიმი (თუ ცვლადს წინასწარ არ გამოვაცხადებთ პროგრამა მისი შეყვანისას გამოგვიტანს შეცდომას). იმისათვის რომ ეს შეცვალოთ, საჭიროა შეასრულოთ ერთ-ერთი შემდეგი მოქმედებებიდან:

- პროგრამის კოდის დასაწყისში მიუთითეთ ოპცია **Option Explicit off**.
- ფანჯარა **Solution Explorer**-ში მონიშნეთ შესაბამისი პროექტი და მის კონსტექსტურ მენიუში აირჩიეთ **Properties**. გადადით პუნქტზე **Compile** და სიაში **Option explicit** ამოირჩიეთ სასურველი მნიშვნელობა.

როგორ მივიღოთ გადაწყვეტილება იმის შესახებ თუ ცვლადის რომელი ტიპი გამოვიყენოთ კონკრეტულ სიტუაციებში?

უნდა გამოვიდეთ მინიმალური საკმარისობის პრინციპიდან!

მთელი რიცხვებისათვის ტიპი Integer უფრო მისაღებია ვიდრე Long, მაგრამ მხოლოდ იმ შემთხვევაში თუ დარწმუნებული ვართ რიცხვთა მნიშვნელობა არ გავა Integer-ის დასაშვები მნიშვნელობების საზღვრებიდან.

შეზღუდული მნიშვნელობის გამო, ტიპი Byte შესაძლებელია გამოყენებულ იქნას მხოლოდ სპეციალურ სიტუაციებში (მაგრამ შეიძლება ის გარკვეულ პირობებში ძალიან საჭირო იყოს).

აუცილებელია თუ არა ცვლადის გამოცხადება?

პასუხი არის ერთმნიშვნელოვანი—ცვლადი უმჯობესია ყოველთვის გამოცხადდეს. მეტიც, უჩუმრად **Visual Basic**-ში გააქტივებულია ცვლადის გამოცხადების აშკარა რეჟიმი (ანუ გამოცხადების გარეშე ცვლადი აღიქმება როგორც შეცდომა).

რატომ უნდა გამოვაცხადოთ ცვლადები?

რა თქმა უნდა ცვლადის გამოყენება მისი წინასწარი გამოცხადების გარეშე, გამოიყურება უფრო მარტივად, მაგრამ ამის გაკეთება რეკომენდირებული არ არის შემდეგი გარემოებების გამო:

ცვლადის გამოცხადების გარეშე მისი გამოყენებისას, მას მიენიჭება მნიშვნელობა **object** და დაიკავებს მეხსიერების უფრო დიდ ნაწილს ვიდრე ჩვენ მოცემულ შემთხვევაში შეიძლება გვჭირდებოდეს. თანამედროვე კომპიუტერების პირობებში ეს შეიძლება ვერ შევიგრძნოთ, მაგრამ როცა საქმე დიდ პროექტებს ეხება, ყოველივე ზემოთქმულის გათვალისწინებას შეიძლება კრიტიკული მნიშვნელობა ქონდეს.

ასევე მეორე მომენტი: დაფუძვით, რომ თქვენ შექმენით ცვლადი და არ გამოაცხადეთ ის. შემდეგ პროგრამის სტრუქტურებში კი დაუშვით შეცდომა მის სახელში. ასეთ შემთხვევაში პროგრამა აღიქვამს ცვლადს, როგორც ახალს, რაც

მოგვცემს სერიოზულ შეცდომებს, რაც ხშირად ძნელად გამოსასწორებელი ხდება.

თუ ცვლადი გამოცხადებულია, მაშინ პროგრამა მოგვცემს ინფორმაციას მისი სახელის შეცდომით შეყვანის შესახებ.

ცვლადისათვის მნიშვნელობის მინიჭება

მანამ, სანამ პროგრამაში გამოვიყენებთ ცვლადს, მას უნდა მივანიჭოთ მნიშვნელობა. მინიჭების ყველაზე მარტივი ხერხი მდგომარეობს იმაში, რომ გამოყენებულ იქნეს მინიჭების ოპერატორი “=”, რომელსაც აქვს შემდეგი სინტაქსი:

ცვლადი = გამოსახულება

მაგალითად:

$x = 10$

Lastname = “პეტრიაშვილი”

ტოლობის ნიშნის მარჯვნივ შეიძლება იყოს არა მარტო კონსტანტა, არამედ უფო რთული გამოსახულება. მაგალითად:

Result = $x + 255$

Name = “პეტრიაშვილი” & “: & “მანუჩარი”.

$D = b^2 - 4 * a * c$

ცვლადის ნულოვანი მნიშვნელობა

ზოგჯერ ცვლადთან მუშაობის დროს, იქმნება სიტუაცია, როდესაც აუცილებელია რომ მას არ ქონდეს გარკვეული მნიშვნელობა. მაგალითად, მონაცემთა ბაზის ველთან მუშაობის დროს მისი შევსება აუცილებელი არ იყოს. ცვლადი მიიღებს ან განსაზღვრულ მნიშვნელობას, ან არ ექნება არანაირი მნიშვნელობა.

ასეთი შემთხვევებში გამოიყენება სტრუქტურა **Nullable**. შემდეგი სტრიქონი საშუალებას იძლევა განვსაზღვროთ ცვლადის ტიპი **Boolean**, რომელსაც შეუძლია მიიღოს ნულოვანი მნიშვნელობა:

Dim HasChildren As Nullable (of Boolean)

Nullable სტრუქტურაში ყველაზე უფრო მნიშვნელოვან თვისებას წარმოადგენს **HasValue** და **Value**. იმისათვის რომ გავარკვიოთ, შეიცავს თუ არა ცვლადი განსაზღვრულ მნიშვნელობას, გამოიყენება თვისება **HasValue**. თუ ეს თვისება იღებს მნიშვნელობას **True**, მაშინ ცვლადის მნიშვნელობის მიღება შეიძლება **Value** თვისების დახმარებით.

უჩუმრად, **Nullable** ცვლადის ტიპის გამოცხადებისას, თვისება **HasValue** იღებს მნიშვნელობას **False**.

კონსტანტები


კონსტანტა ეწოდება ელემენტს, რომლის მნიშვნელობა პროგრამის შესრულების პროცესში არ იცვლება. მოვიყვანოთ რამოდენიმე მაგალითი:

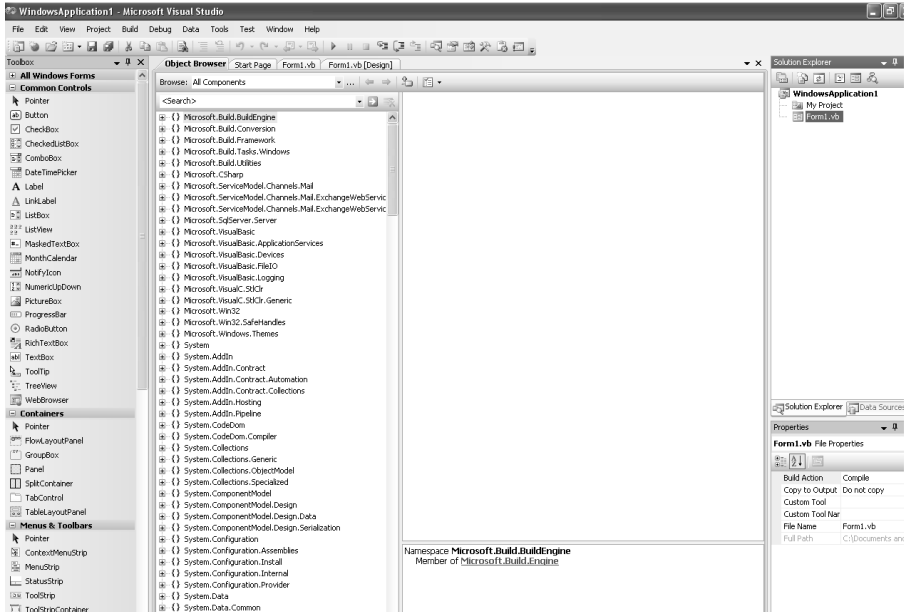
<i>75.074</i>	<i>რიცხვითი კონსტანტა</i>
“ჩემი ხატია სამშობლო, სახატე მთელი ქვეყანა”	სიმბოლური კონსტანტა
8/12/2004	დროითი კონსტანტა
False	ლოგიკური კონსტანტა

Visual Basic-ის საკუთარი კონსტანტები

Visual Basic.Net შეიცავს საკუთარი კონსტანტების უდიდეს რაოდენობას, პრაქტიკულად ყველა შესაძლო

შემთხვევისათვის. მაგ: ცვლადი—ფერი, კონსტანტა—ნითელი, მწვანე და ასშ.

გარკვეული კატეგორიის კონსტანტის ძებნისათვის გამოიყენეთ ობიექტების ბრაუზერი, რომელიც იხსნება **Object Browser** (ობიექტის დათვალიერება) ღილაკზე დაჭერისას  (ის განთავსებულია სტანდარტული ინსტრუმენტების პანელზე).



Object Browser-ის დახმარებით ინტეგრირებული კონსტანტის ძებნა.

კონსტანტის გამოცხადება

კონსტანტის გამოცხადება ცვლადის გამოცხადების ანალოგიურია. კონსტანტის გამოცხადება ხდება მოდულისა და პროცედურის დონეზე. ამავე დროს მისი მოქმედების არე განისაზღვრება იმავე ნუსით, როგორც ცვლადის შემთხვევაში.

პროცედურის დონეზე კონსტანტის გამოცხადებისათვის გამოიყენება ოპერატორი **Const**, რომელსაც აქვს შემდეგი სინტაქსი:

Const კონსტანტის სახელი **As** მონაცემთა ტიპი = გამოსახულება

მაგალითად:

Const X As String = “ჩემი ხატია სამშობლო, სახატე მთელი ქვეყანა”

მოდულის დონეზე კონსტანტის გამოცხადებისათვის დამატებით შეიძლება მიეთითოს მისი მოქმედების არე. ამ შემთხვევაში ოპერატორს **Const** აქვს შემდეგი სინტაქსი:

Private Const კონსტანტის სახელი **As** მონაცემთა ტიპი = გამოსახულება

შემდეგ მოყვანილ მაგალითში კონსტანტა **X** გამოცხადებულია გლობალურად:

public Const X As String = “ჩემი ხატია სამშობლო, სახატე მთელი ქვეყანა”

შენიშვნა

უჩუმრად **Visual Basic.2008**-ში გააქტივებულია რეჟიმი “კონსტანტის არააშკარა გამოცხადება”. იმისათვის, რომ ეს შევცვალოთ, პროგრამის კოდს დასაწყისში უნდა მივუთითოთ **option Strict On**.

ჩამონათვალი

ჩამონათვალი წარმოადგენს ურთიერთდაკავშირებული კონსტანტების ნაკრებს. მაგ: შეიძლება მათი გამოყენება დღეების და თვეების წარმოსადგენად.

ჩამონათვალის გამოცხადებისათვის გამოიყენება ოპერატორი Enum, რომელსაც აქვს შემდეგი სინტაქსი:

მხედველობის არე Enum ცვლადის სახელი As მონაცემთა ტიპი ჩამონათვალის ნევრი

End Enum

ჩამონათვალი შეიძლება იყოს მხოლოდ მთელრიცხვიანი მონაცემთა ტიპის (**Byte, Integer, Long, Sbyte, Short, UInteger, Ulong, Ushort**). თუ მონაცემთა ტიპი მითითებული არ არის, მაშინ ინიციალიზირებული მნიშვნელობის საფუძველზე, კომპილატორი ქმნის ტიპს, თუ არ არის მითითებული ტიპი და ელემენტების ჩამონათვალს არ მიენიჭება კონკრეტული მნიშვნელობა, მაშინ უჩუმრად გამოიყენება ტიპი **Integer**, ხოლო ჩამონათვალის ელემენტები ღებულობს მნიშვნელობას ნულიდან—ელემენტთა რაოდენობას მინუს ერთამდე ($n-1$).

ჩამონათვლის გამოცხადების მაგალითი:

Public Enum seasons

winter = 1

spring

summer

autumn

End Enum

ამ მაგალითებში ელემენტების ჩამონათვალს **spring**, **summer** და **autumn**, მიენიჭება შესაბამისად 2, 3 და 4 მნიშვნელობა.

შენიშვნა

ჩამონათვალი შეიძლება გამოცხადებული იყოს მხოლოდ კლასის ან მოდულის გამოცხადების ნაწილში. მისი გამოცხადება არ შეიძლება პროცედურაში.

პროექტი “ღოს ფორმატი”

მოდით აქ ცოტახნით შევისვენოთ თეორიისაგან და გადავიდეთ პრაქტიკაზე. შევექმნათ პროექტი, რომელიც საშუალებას მოგვცემს გამოვიტანოთ ამჟამინდელი თარიღი და დრო, ასევე შევცვალოთ ამ ინფორმაციის ფორმატი.

თავდაპირველად ჩამოვაყალიბოთ ამოცანა. ის თუ რის გაკეთებას ვთხოვთ ჩვენს მიერ შექმნილ პროგრამას (პროექტს). ამჯერად გეტყვით, რომ ჩვენი მიზანია ღილაკზე დაჭერისას ეკრანზე გამოვიდეს დღევანდელი თარიღი და დრო. შემდეგ კი გვექონდეს საშუალება ვცვალოთ ამ ინფორმაციის ფორმატი.

ამოცანის ჩამოყალიბების შემდეგ უნდა მოვიფიქროთ მისი შესრულების ალგორითმი.

მოცემული პროექტის ალგორითმი მარტივია, რადგან ის არ შეიცავს პირობით გადასვლებს. ალგორითმი იქნება წრფივი სტრუქტურის. უბრალოდ ღილაკზე **Button** დაჭერით შევასრულებთ ბრძანებას. **RadioButton**-ებზე დაჭერით სხვა ბრძანებებს.

გავიხსენოთ, რომ ვიზუალური დაპროგრამება ეს არის ვიზუალური ინტერფეისი, პლიუს პროგრამული კოდი. ე.ი. პირველ რიგში უნდა შევექმნათ ვიზუალური ინტერფეისი.

ცარიელ ფორმაზე ელემენტთა პანელიდან Tolboxes გადმოვიტანოთ ორი ლილაკი Button. ერთი ტექსტური ბლოკი TextBox და სამი ელემენტი RadioButton. მოვნიშნოთ ელემენტი ლილაკი (Button) და თვისებათა ფანჯარაში მას შევუცვალოთ წარწერა. ამისათვის თვისებათა ფანჯარაში მოვნახოთ თვისება Text და მის გვერდით არსებული ტექსტის ნაცვლად ჩავწეროთ: "amJamindeli dro". *ტექსტი ქართულია, მაგრამ შემთხვევით არ არის ლათინური სიმბოლოებით.*

თუ გვინდა რომ ლილაკზე ან სხვა ელემენტზე გვქონდეს ქართული წარწერა, ამისათვის არსებობს 2 საშუალება:

- ჩავწეროთ ქართული ტექსტი ლათინური სიმბოლოებით და შემდეგ თვისებათა ფანჯარაში მოვნახოთ ელემენტის თვისება **Font**, მის გვერდზე დაჭერით გაიხსნება შესაბამისი ფანჯარა, სადაც ჩვენ შეგვიძლია შევცვალოთ ფონტი (ავირჩიოთ ქართული) შევცვალოთ მისი ზომა (**Size**), სისქე (**Bold**), დახრილობა (**Italic**) და სხვ. ქართული ფონტის არჩევის შემდეგ, წარწერა ელემენტზე Button გახდება ქართული☺.
- არსებობს მეორე და უფრო მარტივი ვარიანტი. ტექსტი ავკრიფოთ ქართული **Unicod**-ის სიმბოლოებით. **Visual studio**-ს აქვს **Unicod**-ის მხარდაჭერა.

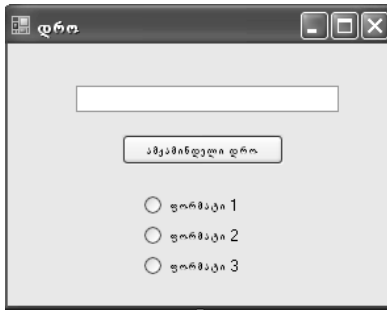
მოვნიშნოთ ელემენტი **TextBox** და თვისებათა ფანჯარაში შევცვალოთ მისი თვისება **Name** —დავარქვათ **Display**.

*ერთმანეთისაგან არ უნდა შეგვეშალოს თვისება **Name** და თვისება **Text**. თავდაპირველად (უჩუმრად) ელემენტების ეს თვისებები ერთნაირია. თვისება **Text** წარმოადგენს წარწერას ელემენტზე. თვისება **Name** კი არის ელემენტის სახელი, რომლითაც მიმართავს მას პროგრამა.*

ასევე შევცვალოთ **RadioButton**-ების თვისება **Name**. **Format1**, **Format2** და **Format3** შესაბამისად.

შეგვიძლია ელემენტების ეს თვისება ხელუხლებელი დავტოვოთ. ამ შემთხვევაში პროგრამამ მათ უნდა მიმართოს მათი სახელებით, რომელიც მინიჭებული აქვთ უჩუმრად. თუ გვაქვს 2 ან მეტი ერთნაირი ელემენტი, პროგრამა მათ მიანიჭებს სახელებს რომლებიც განსხვავებულია “ინდექსით”. მაგ: TextBox1, TextBox2, TextBox3 და ასშ.

ასევე მონიშნეთ სათითაოდ ელემენტები RadioButton და შევცვალოთ მათი თვისება Text. ფორმატი1, ფორმატი 2 და ფორმატი 3 შესაბამისად. საბოლოოდ ფორმა მიიღებს შემდეგ სახეს:



ახლა მოდით გავაცოცხლოთ ჩვენი პროგრამა (ჯერ-ჯერობით ის ვერაფერს გააკეთებს). ამისათვის საჭიროა ჩავწეროთ პროგრამული კოდი **Basic**-ის ენაზე. რასაკვირველია კოდი სადღაც უნდა ჩაინეროს. | ამოცანა იქნება ის თუ სად ჩავწეროთ კოდი. გვახსოვდეს პროგრამირების ერთ-ერთი ძირითადი კანონი: *პროგრამა რეაგირებს მოვლენაზე*. ე.ი. პროგრამული კოდი გაეშვება რაღაც მოვლენასთან მიმართებაში. რა შეიძლება იყოს მოვლენა? ერთ შემთხვევაში შეიძლება იყოს ლილაკზე დაჭერა, მეორე შემთხვევაში ლილაკზე აშვება, მესამე შემთხვევაში პროგრამის გაშვება, მეოთხე შემთხვევაში ტექსტის ცვლილება და ასშ.

მოცემულ შემთხვევაში ამოცანას გაგიმარტივებთ და გეტყვით, რომ ჩვენი პროექტისათვის უნდა ჩაინეროს კოდი, რომელიც გაეშვება ლილაკზე **Button** დაჭერისას, რადგანაც სწორედ მასზე დაჭერისას უნდა გამოვიდეს ინფორმაცია

ამჟამინდელი დროის შესახებ. დროის ფორმატი ასევე უნდა შეიცვალოს ელემენტებზე **RadioButton** დაჭერისას.

დავაჭიროთ სწრაფად ორჯერ ღილაკს წარწერით “ამჟამინდელი დრო”. გაიხსნება პროგრამული კოდის ფანჯარა. დავინახავთ შემდეგ კოდს:

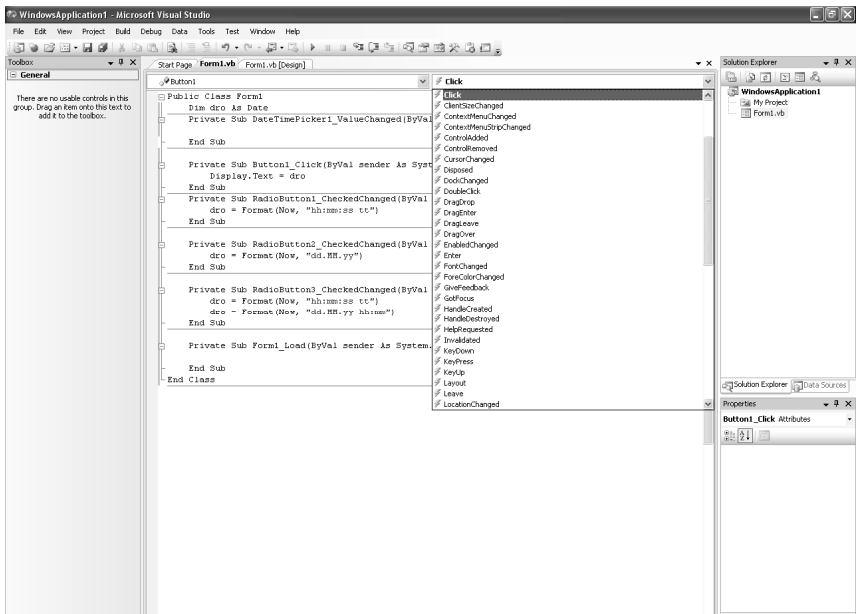
```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As

End Sub

End Class
```

კურსორი ციმციმებს იმ უბანში სადაც ჩვენ უნდა ჩავწეროთ კოდი, თუ გვინდა რომ ის გაეშვას ღილაკზე დაჭერისას.



მის ზემოთ წარწერა `Private Sub Button1_Click` აღნიშნავს, რომ ბრძანება შესრულდება ღილაკზე `Button1`-ზე დაჭერისას. პროგრამა რეაგირებს მოვლენაზე “დაჭერა” ანუ `Click`. მოვლენა ყოველთვის მთავრდება ბრძანებით `End Sub` (გამოდის ავტომატურად). მთელი კოდი კი იწყება `Public Class`-ით და მთავრდება `End Class`-ით. როგორც მიხვდით მთელი კოდის ხელით შეყვანა არ მოგიწევთ, მის ნაწილს **visual studio** თქვენს მაგივრად გააკეთებს.

ეკრანის მარჯვენა მხარეს შეგვიძლია ჩამოვშალოთ მოვლენათა სია. აქ ჩამოთვლილია ყველა მოვლენა რომელზეც შესაძლებელია პროგრამა რეაგირებდეს. შემდგომში ჩვენ ამ საკითხს კიდევ დავუბრუნდებით. მანამდე კი თქვენ შეგიძლიათ გაეცნოთ ამ მოვლენებს და ჩაატაროთ დამოუკიდებელი ექსპერიმენტები. ღილაკის კოდს უჩუმრად დაყენებული აქვს მოვლენა `Click`, ანუ ღილაკის კოდი რეაგირებს ღილაკზე დაჭერაზე (შესრულდება ღილაკზე დაჭერისას).

რაც შეეხება იმ უბანს, სადაც ციმციმებს კურსორი—კოდი ანუ ბრძანება თუ რა უნდა მოხდეს მოცემულ ღილაკზე დაჭერისას ჩვენ უნდა ჩავწეროთ.

შევიყვანოთ შემდეგი კოდი:

```
Display.Text = dro
```

როცა ავკრეფთ კოდს **Display** (ასე დავარქვით ჩვენს **TextBox**-ს, და ავკრეფთ სიმბოლოს ”.” (წერტილი). ჩამოიშლება მოცემული ელემენტის თვისებათა სია, რომელიც ჩვენ შეგვიძლია შევცვალოთ პროგრამულად (ანუ ისინი წარმოადგენენ ცვლადებს). ამჯერად ჩვენ გვინდა რომ ელემენტზე შეიცვალოს ტექსტი, ანუ გამოჩნდეს მასზე ამჟამინდელი დრო. ამიტომ ჩამონათვალიდან ავირჩევთ თვისებას **Text** და მივანიჭებთ მას ცვლადი **dro**-ის მნიშვნელობა. ანუ რა მნიშვნელობაც ექნება ცვლადს დრო, ის ჩაინერება **TextBox**-ში.

ახლა კი გამოვაცხადოთ ცვლადი **dro** და მივანიჭოთ მას ტიპი **Date**. ამისათვის **Public Class** `Form1`-ის ქვემოთ ავკრიფოთ:

```
Dim dro As Date
```

ფორმაზე ორჯერ დავაკლიკოთ ელემენტს RadioButton რომელსაც მივანიჭეთ სახელი **Format1** და იმ ადგილზე სადაც ციმციმებს კურსორი ჩავწეროთ:

```
dro = Format(Now, "hh:mm:ss tt")
```

შემდეგ 2-ჯერ დავაჭიროთ **Format2**-ს და ჩავწეროთ კოდი:

```
dro = Format(Now, "dd.MM.yy")
```

Format3 ში კი ჩავწეროთ:

```
dro = Format(Now, "dd.MM.yy")
```

საბოლოოდ მთლიანი კოდი მიიღებს სახეს:

```
Public Class Form1
    Dim dro As Date
    Private Sub Button1_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
        Display.Text = dro
    End Sub
    Private Sub RadioButton1_CheckedChanged(ByVal
sender As System.Object, ByVal e As System.EventArgs)
Handles Format1.CheckedChanged
        dro = Format(Now, "hh:mm:ss tt")
    End Sub

    Private Sub RadioButton2_CheckedChanged(ByVal
sender As System.Object, ByVal e As System.EventArgs)
Handles Format2.CheckedChanged
        dro = Format(Now, "dd.MM.yy")
    End Sub

    Private Sub RadioButton3_CheckedChanged(ByVal
sender As System.Object, ByVal e As System.EventArgs)
Handles Format3.CheckedChanged
        dro = Format(Now, "dd.MM.yy hh:mm")
    End Sub
End Class
```

End Sub

ახლა კი დადგა დრო გამოვცადოთ ჩვენი პროგრამა. ამისათვის უნდა გავუშვათ ის. (ერთდროულად ხდება მისი მანქანურ ენაზე გადაყვანა). პროგრამის გაშვება ხდება შემდეგნაირად:

- მენიუთა სტრიქონში ავირჩიოთ **Debug** და ჩამოშლილ მენიუში ბრძანება **Start Debugging**.
- ან დავაჭიროთ სამკუთხა ფორმის ლილაკს ლილაკების სტრიქონში.

პროგრამა გაეშვება. თუ ავირჩევთ დროის ფორმატს ელემენტებზე **RadioButton** დაჭერით და შემდეგ დავაჭერთ ლილაკს “ამჟამინდელი დრო”, ელემენტზე **TextBox** მივიღებთ შესაბამის ინფორმაციას. დროის ფორმატის შემდგომი შეცვლისას, კვლავ დავაჭერთ ლილაკს და დრო უკვე სხვა ფორმატში გადავა☺.

სიდიდეების შესანახად, გარდა მარტივი ცვლადებისა, შეიძლება გამოყენებულ იქნას მასივები. მასივი წარმოადგენს ერთი ტიპის ცვადის მონესრიგებულ ერთობლიობას. მასივის ცვლადებს გააჩნიათ ერთიდაიგივე დასახელება (გაერთიანებული არიან ერთ სახელ ქვეშ) და ერთმანეთისაგან განსხვავდებიან ინდექსებით.

სიტყვა მასივის გაგონებისას უპირველესი ასოციაცია შეიძლება დაკავშირებულ იქნას საცხოვრებელი კორპუსების მასივთან. დიახ სწორედ ეს წარმოადგენს მასივის ერთ-ერთ მაგალითს. ასეთი მასივის ელემენტებია საცხოვრებელი კორპუსები, რომლებიც ერთმანეთისაგან ნომრებით განსხვავდებიან (ინდექსები).

ელემენტების ინდექსი მითითებულია მასივის სახელის შემდეგ მრგვალ ფრჩხილებში. მაგალითად: $X(1)$, $X(2)$... $X(10)$. კორპუსი(1), კორპუსი(2) ... კორპუსი (10). აღნიშნული ცვლადები წარმოადგენენ მასივის ელემენტებს, რომელთა სახელია ერთ შემთხვევაში X , მეორე შემთხვევაში “კორპუსი”.

თითოეულ ასეთი ცვლადს ეწოდება “*მასივის ელემენტი*”. მასივში ელემენტების რაოდენობას ეწოდება “*მასივის განზომილება*”. მასივის განზომილება შეზღუდულია ოპერატიული მეხსიერების მოცულობით და მასივის ელემენტების ტიპით.

მასივის გამოცხადება

Visual Basic-ში არსებობს ფიქსირებული და დინამიური განზომილების მასივები. ფიქსირებული განზომილების მასივს აქვს უცვლელი განზომილება. დინამიურ მასივებს შეუძლიათ განზომილების შეცვლა პროგრამის მსვლელობის პროცესში.

ფიქსირებული ზომის მასივის გამოცხადება.

ფიქსირებული ზომის მასივის გამოცხადება დამოკიდებულია ხილვადობის არეზე და ხორციელდება შემდეგი სახით:

- **Public** ოპერატორის დახმარებით—გლობალური მასივი.
- **Private** ოპერატორის დახმარებით—მოდულის დონის მასივი.
- **Dim** ოპერატორის დახმარებით—ლოკალური მასივი.

მასივის გამოცხადების დროს, მისი სახელის შემდეგ მრგვალ ფრჩხილებში მითითებულია მასივის ზედა ზღვარი. მასივის ქვედა ზღვარი უჩუმრად ყოველთვის 0-ის ტოლია. მაგალითად, შემდეგი კოდით გამოცხადებულია მასივი, რომელიც შედგება 21 ელემენტისაგან. მასივის ელემენტების ინდექსი იცვლება 0-დან 20-მდე:

Dim x (20) As Integer

იგივე ზომის გლობალური მასივის შესაქმნელად აუცილებელია გამოყენებული იქნას შემდეგი კოდი:

public x (20) As Integer

Visual Basic.NET საშუალებას იძლევა, რომ გამოყენებულ იქნას მრავალგანზომილებიანი მასივებიც. მაგალითად, შემდეგ კოდში გამოცხადდება ორგანზომილებიანი მასივი 21×21:

Dim x (20 , 20) As Integer

დინამიური მასივის გამოცხადება

Visual Basic იძლევა პროგრამის მუშაობის პროცესში მასივის განზომილების შეცვლის საშუალებას. დინამიური

მასივის გამოყენება უზრუნველყოფს მეხსიერების ეფექტიურ მართვას, დროებით დაიკავებს დიდი მასივისათვის მეხსიერებას (როცა ეს მასივი გამოიყენება), ხოლო შემდეგ ათავისუფლებს მას.

დინამიური მასივის შექმნა ხორციელდება განზომილების მაგივრად ცარიელი ბრჩხილების მითითებით. მაგ:

Dim x () As Integer

ოპერატორის **ReDim** დახმარებით მიეთითება მასივის განზომილება. ოპერატორს **ReDim** გააჩნია სინტაქსი, რომელიც ფიქსირებული მასივის ზომის გამოცხადების ანალოგიურია. მაგალითად, მასივის ზომა შეიძლება იყოს მოცემული ნებისმიერი შემდეგი ხერხით:

ReDim x (50)

ოპერატორ **ReDim** შესრულების დროს, იკარგება მონაცემები, რომელიც ადრე იყო განთავსებული მასივში. ეს მოსახერხებელია იმ შემთხვევაში, როდესაც მონაცემები უკვე საჭირო არ არის, თქვენ გინდათ შეცვალოთ მასივის ზომა და მოამზადოთ ის ახალი მონაცემების განსათავსებლად.

თუ თქვენ გინდათ მასივის ზომის შეცვლა მონაცემების დანაკარგის გარეშე, აუცვლებელია გამოიყენოთ ოპერატორი **ReDim** საკვანძო სიტყვით **Preserve**. მაგალითად შემდეგი პროგრამული კოდი ზრდის მასივის ზომას 1-ით, მასივში შენახული მონაცემების დაკარგვის გარეშე:

ReDim Preserve y (x + 1)

მასივის ინიციალიზაცია

მასივის ინიციალიზაცია ხორციელდება მინიჭების ოპერატორის დახმარებით, მაგრამ ეს შეიძლება გაკეთდეს მისი გამოცხადების დროსაც, როდესაც მასივის მნიშვნელობას მოათავსებენ ფრჩხილებში:

Dim Names () As String = (“ირაკლი” , “ დავითი “ ,
“გიორგი”)

ამ შემთხვევაში მასივის ინიციალიზაციისას მისი საზღვრები მითითებული არ არის.

მასივებთან მუშაობა

ცვლადების ყველა მასივი იქმნება **Array** კლასის საფუძველზე (კლასებზე შემდგომში გვექნება საუბარი). ცხრილში მოყვანილია ამ კლასის რამოდენიმე სასარგებლო მეთოდი.

<i>Array</i> კლასის მეთოდები	
<i>მეთოდი</i>	<i>აღწერა</i>
BinarySearch	<p>იძლევა გადარჩეული მასივების ძებნის საშუალებას. თუ ელემენტი მონახულია, მაშინ ბრუნდება მისი ინდექსი, სხვაგვარად—უარყოფითი რიცხვი. მაგალითად:</p> <pre>Dim Names () As String = (“ირაკლი” , “ დავითი “ , “გიორგი”)</pre> <pre>Dim searchNames As String = “დავითი”</pre> <pre>Dim i As Integer = Array.BinarySearch (Names, searchNames)</pre>
GetLowerBound	საზღვრავს მასივის მინიმალურ ინდექსს.
GetUpperBound	<p>საზღვრავს მასივის მაქსიმალურ ინდექსს.</p> <pre>Dim strNames () As String = (“ირაკლი” , “ დავითი “ , “გიორგი”).</pre> <pre>Dim i As Integer = strNames.GetUpperBound (0)</pre>
Reverse	ცვლის მასივის ელემენტების მიმდევრობას უკუმიმართულებით.

Sort	<p>მასივის ელემენტების სორტირება.</p> <p>Dim strNames () As String = (“ირაკლი”, “ დავითი “,“გიორგი”)</p> <p>Array.Sort (strNames)</p>
-------------	---

პროგრამული კოდის გაფორმება

კომენტარი

კომენტარი ეს არის პროგრამაში კოდის განმარტება, რომელიც მომხმარებელს ეხმარება მის უკეთ გარკვევაში.

იმისათვის რომ ჩაერთოს კომენტარი, საჭიროა დაიწეროს ' სიმბოლო, რომელიც შეიძლება იყოს პირველი სიმბოლო სტრიქონში ან შეიძლება მდებარეობდეს მის ნებისმიერ ადგილზე. ეს სიმბოლო ნიშნავს კომენტარის დაწყებას. ნებისმიერი ტექსტი, რომელიც განლაგებულია სტრიქონში მოცემული სიმბოლოს გაყოფაზე, იქნება აღქმული როგორც კომენტარი (არ აღიქმება როგორც პროგრამული კოდი), **Visual Basic** ამ ტექსტის ტრანსლირებას არ გაუკეთებს.

მაგალითად:

Dim Name as string ‘ გამოვაცხადეთ ცვლადი და მივანიჭეთ მას ტიპი.

“გამოვაცხადეთ ცვლადი და მივანიჭეთ მას ტიპი”— ეს ჩვენს მიერ გაკეთებული კომენტარია, რომელიც შემდგომში შეიძლება გამოგვადგეს.

ოპერატორის განლაგება რამოდენიმე სტრიქონად

იმ შემთხვევაში თუ ოპერატორი (ერთ სტრიქონზე განლაგებული პროგრამული კოდი) დიდი სიგრძისაა, ის შეიძლება დაიყოს რამოდენიმე სტრიქონად, სტრიქონის გაგრძელების სიმბოლოს გამოყენებით, რომელიც წარმოადგენს პრობელს და შემდეგ ხასგასმის სიმბოლოს (_).

მაგალითად, განვათავსოთ ოპერატორი ორ სტრიქონად, რომელიც აერთიანებს გვარს, სახელს და მამის სახელს:

Name = Lastname & Firstname & Secondname

მივიღებთ :

*Name = Lastname _
& Firstname & Secondname*

რამოდენიმე ოპერატორის განლაგება ერთ სტრიქონში

როგორც წესი პროგრამის დანერისას, ოპერატორები განლაგდებიან სხვადასხვა სტრიქონში. თუ ოპერატორები მცირე ზომისაა, ზოგჯერ მოსახერხებელია მათი ერთ სტრიქონში გაერთიანება.

Visual Basic საშუალებას იძლევა “:” სიმბოლოთი გამოყოფით, ისინი განთავდეს ერთ სტრიქონში. მაგალითად:

Lastname = “პეტრიაშვილი” : Firstname = “დავითი”

პროგრამული მოდულები

Visual Basic.NET-ის პროგრამები ინახება პროგრამულ მოდულებში, რომლებიც შეიძლება იყოს სამი სახის: *ფორმის მოდული*, *სტანდარტული მოდული* და *კლასის მოდული*.

მარტივი დანართები, შემდგარი ერთი ფორმისაგან, როგორც ნესი შეიცავს მხოლოდ ფორმის მოდულს.

იმის მიხედვით თუ როგორ გართულდება პროგრამა, განმეორებადი ფუნქციები, რომლებიც სრულდება რამოდენიმე ფორმის მოდულში, შეიძლება გამოვყოთ ცალკე პროგრამულ კოდად, რომელიც ყველასათვის საერთო იქნება. ასეთ პროგრამულ კოდს უწოდებენ *სტანდარტულ მოდულს*.

ფორმის მოდული შეიძლება შეიცავდეს ცვლადების და კონსტანტების გამოცხადებას, გარე პროცედურებს, რომლებიც გამოიყენება მოდულის დონეზე.

სტანდარტული მოდული შეიძლება შეიცავდეს გლობალური და ლოკალური ცვლადების და კონსტანტების გამოცხადებას, შიდა და საერთო ხასიათის პროცედურებს, რომლებიც ხელმისაწვდომია მოცემული პროგრამის სხვა მოდულებისათვის.

ობიექტებზე ორიენტირებული დაპროგრამების გამოყენებისას, Visual Basic.NET-ში იქმნება *კლასის მოდულები*.

პროცედურები

დაპროგრამებისას ხშირად იყენებენ “პროცედურებს” — პროგრამული კოდის ლოგიკურად დასრულებული ბლოკები. პროცედურამ შეიძლება მიიღოს რომელიმე სანყისი მნიშვნელობა და დააბრუნოს შედეგი. პროცედურის გამოყენება ამცირებს პროგრამის მოცულობას, ამარტივებს მის სტრუქტურას, აიოლებს პროგრამის გამართვას. თავისთავად პროცედურები შესაძლოა გამოყენებულ იქნას სხვა პროცედურების შესაქმნელად.

Visual Basic-ში არსებობს შემდეგი სახის პროცედურები:

- **Sub**
- **Function**

- **Property**

პროცედურა Sub

პროცედურა Sub არ აბრუნებს მნიშვნელობას და ხშირად გამოიყენება მასთან დაკავშირებული მოვლენის დამუშავებისათვის. ის შეიძლება განათავსოთ სტანდარტულ მოდულში, კლასისა და ფორმის მოდულში. მას აქვს შემდეგი სინტაქსი:

ხელმისაწვდომობის დონე Sub პროცედურის სახელი
(*არგუმენტები*)

ოპერატორები

End Sub

პარამეტრით “ხელმისაწვდომობის დონე” მიეთითება, პროგრამის სხვა ნაწილებისათვის მისაწვდომია თუ არა პროცედურა. მას შეუძლია მიიღოს შემდეგი სახე:

- **Public**—პროექტში პროცედურა საყოველთაოდ ხელმისაწვდომია (სადაც ის განსაზღვრულია).
- **Private**—პროცედურა ხელმისაწვდომია მხოლოდ იმ კლასში ან მოდულში სადაც ის განსაზღვრულია.
- **Protected**—დაცული პროცედურები ხელმისაწვდომია კლასის შიგნით, სადაც ისინი არიან გამოცხადებულნი, ასევე ამ კლასის წარმოებულებში.
- **Friend**—მეგობრული პროცედურები, ხელმისაწვდომია მხოლოდ იმ შიდა კონსტრუქციისათვის, სადაც ის არის

გამოცხადებული. კონსტრუქცია—დანართის სრულიად დამოუკიდებელი ერთეულია.

- **protected Friend** – პროცედურის ხელმისაწვდომობა ფართოვდება პროგრამის ანყობისა და წარმოებული კლასებისათვის.

საკვანძო სიტყვებს **Sub** და **End Sub** შორის, პროცედურაში განთავსდებიან მისი გამოცხადებისას შესასრულებელი პროგრამული კოდის ოპერატორები. **Sub** პროცედურა იყოფა საერთო და მოვლენების პროცედურებად.

მოვლენების პროცედურა

მოვლენების დამუშავების პროცედურა დაკავშირებულია ობიექტებთან, რომლებიც განთავსებულია **Visual Basic**-ის ფორმაზე და სრულდება იმ მოვლენების გაჩენისთანავე, რომლებთანაც ისინი არიან მიბმულნი. ანუ პროგრამა რეაგირებს მოვლენებზე.

მაგალითად: მოვლენას შეიძლება წარმოადგენდეს ლილაკზე მაუსით დაკლიკება (პროგრამა დაიწყებს შესრულებას ლილაკზე დაკლიკებისას ანუ რეაგირებს ამ მოვლენაზე). ლილაკზე მაუსის კურსორის დაყენება (პროგრამა დაიწყებს შესრულებას ლილაკზე მაუსის კურსორის დაყენებისას), ტექსტურ ბლოკში ტექსტის ცვლილება (პროგრამა დაიწყებს შესრულებას ტექსტის ცვლილებისას) და ასე.

მოვლენისათვის, რომელიც დაკავშირებულია ფორმასთან, მოვლენების დამუშავების პროცედურას **Sub** აქვს შემდეგი სინტაქსი:

Private Sub ფორმის სახელი_მოვლენის სახელი (არგუმენტები)
Handles მოვლენის სახელი
ოპერატორები

End Sub

“სახელი_მოვლენის” სახელი შეიძლება გამოიყურებოდეს შემდეგნაირად **Form1_Load** ეს ნიშნავს რომ ფორმის სახელია **Form1** და პროგრამა გაეშვება მოვლენაზე **Load**, ანუ ფორმის გაშვებისას.

მოვლენისათვის, რომელიც დაკავშირებულია მართვის ობიექტებთან, მოვლენების დამუშავების პროცედურა **Sub**-ს აქვს შემდეგი სინტაქსი:

```
Private Sub ელემენტის სახელი_მოვლენის სახელი  
(არგუმენტები) Handles მოვლენის სახელი  
    ოპერატორები  
End Sub
```

“სახელი_მოვლენის” შეიძლება გამოიყურებოდეს შემდეგნაირად **Button1_Click** ეს ნიშნავს რომ ელემენტის სახელია **Button1** (ლილაკი) და პროგრამა გაეშვება მოვლენაზე **Click** ანუ ლილაკზე დაკლიკებისას.

Visual Basic ამსუბუქებს პროგრამისტის ამოცანას. შესაბამის ელემენტზე ორჯერ დაკლიკებისას გაიხსნება პროგრამული კოდის ფანჯარა, სადაც უკვე ჩანერილია პროცედურა. პროგრამისტი კი კოდს ჩანერს **Sub** და **End Sub** შორის.

ასევე ჩამოსაშლელი სიით შეუძლია შეარჩიოს ის მოვლენები, რომელთა მოხდენისას შესრულდება პროგრამული კოდი.

საერთო პროცედურები

საერთო პროცედურები—ეს არის **Visual Basic**-ის ოპერატორების სერია, რომელიც მოთავსებულია **Sub** და **End Sub** საკვანძო სიტყვებს შორის. პროცედურის ყოველი გამოძახებისას, ეს ოპერატორები სრულდება, დანყებული პირველი

ოპერატორიდან და მთავრდება End Sub, Exit Sub ან Return-თან შეხვედრისას.

Sub პროცედურა ასრულებს გარკვეულ მოქმედებას, მაგრამ არ აბრუნებს მნიშვნელობას.

Sub პროცედურა შეიძლება იქნეს განსაზღვრული მოდულის, კლასის ან სტრუქტურის შიგნით. უჩუმრად ის საყოველთაოდ მისაწვდომია—თუ მითითებული არ არის Private პარამეტრი ან საკვანძო სიტყვა Static, პროცედურას შეიძლება მივმართოთ პროგრამის ნებისმიერი ადგილიდან.

პროცედურების გამოძახება

Sub პროცედურის გამოძახება ხორციელდება შემდეგი სინტაქსის დახმარებით:

Call პროცედურეს სახელი (არგუმენტი 1, არგუმენტი 2, . . . არგუმენტი N)

საკვანძო სიტყვა Call არა არის აუცილებელი

პროგრამის სხვა მოდულიდან პროცედურის გამოძახების შემთხვევაში, აუცილებელია მიუთითოთ მოდულის სახელი, რომელიც შეიცავს ამ პროცედურას.

მაგალითად პროცედურის გამოსაძახებლად, რომელიც იმყოფება Form1 ფორმის მოდულში, ოპერატორს უნდა ქონდეს შემდეგი სახე:

*Call Form1.პროცედურეს სახელი (არგუმენტი 1, _
არგუმენტი 2, . . . არგუმენტი N)*

პროცედურები **Function**

პროცედურებს **Function**, **Sub**-ისგან განსხვავებით, შეუძლიათ მნიშვნელობის დაბრუნება მათ გამომწვევ პროცედურაში. მის სინტაქსს აქვს შემდეგი სახე:

მისაწვდომობის დონე Function პროცედურეს სახელი_
(არგუმენტი) As type

ოპერატორები

End Function

მისაწვდომობის დონედ შეიძლება მითითებულ იქნას **Public**, **Protected**, **Friend**, **Protected Friend** ან **Private**.

Function პროცედურებს, როგორც ცვლადებს, გააჩნიათ ტიპი, გამოცხადება ხდება საკვანძო სიტყვის **As** დახმარებით. თუ პროცედურის ტიპი მოცემული არ არის, მაშინ უჩუმრად მას ენიჭება ტიპი **Object**.

პროცედურის ტიპი, განსაზღვრავს მის მიერ დაბრუნებული მნიშვნელობის ტიპს. *დაბრუნებული მნიშვნელობა* ეწოდება მნიშვნელობას, რომელსაც ფუნქცია უკან დაუბრუნებს მის მიერ გამოძახებულ პროგრამას.

ფუნქციას შეუძლია მნიშვნელობა დააბრუნოს ორი ხერხით:

- მნიშვნელობა არ გადაეცემა პროგრამას, რომელმაც გამოიძახა მოცემული ფუნქცია მანამ, სანამ არ შესრულდება **End function** ან **Exit Function**.
- **Return** ოპერატორის გამოყენებით შეგვიძლია განვსაზღვროთ ფუნქციის მნიშვნელობა და მართვა

მაშინათვე გადავცეთ პროგრამას, რომელმაც გამოიძახა მოცემული ფუნქცია.

პირველი ხერხის უპირატესობა არის ის, რომ ფუნქციის სახელს შეიძლება წინასწარ მივანიჭოთ მნიშვნელობა, რომელიც შემდეგ პროცედურის შესრულებისას მარტივად შეიძლება შეიცვალოს.

სტანდარტული ფუნქცია ეს არის რალაც დაფარული პროგრამა, რომელიც მიიღებს თავის პარამეტრებს სანყისი მონაცემების სახით (არგუმენტები), მოახდენს მათ გარდაქმნას და მიიღებს ერთ სიდიდეს, რომელსაც ფუნქციის მნიშვნელობა ეწოდება

როცა ჩვენ ვხედავთ ოპერატორს

$$b = a * (Len(w) - Abs(c+200))$$

ვამბობთ, რომ მისი შესრულებისას კომპიუტერი მიმართავს ფუნქციებს *Len* და *Abs*. ამ ფუნქციების გამოყენებას კი მათზე მიმართვა ეწოდება.

ახლა კი გადავიდეთ მომხმარებლის ფუნქციებზე და შევიგრძნოთ თუ როგორ მოხერხებულია მათი პროგრამაში გამოყენება. მათ თქვენ ქმნით მაშინ, როცა .NET Framework-ის კლასების ბიბლიოთეკაში საჭირო ფუნქცია არ არის. მაგ: გინდათ ფუნქცია, რომლის არგუმენტები იქნება მართკუთხედის გვერდები, მნიშვნელობა კი ამ მართკუთხედის პერიმეტრი.

ფუნქციის გამოყენება განსაკუთრებით მნიშვნელოვანია მაშინ, როცა გვიხდება პროგრამაში ერთიდაიმავე ოპერატორის რამოდენიმეჯერ განმეორება. მაგ. როცა პროგრამამ უნდა გამოთვალოს რამოდენიმე კვადრატის ფართობი. შესაბამისი ფუნქციის გამოყენებით არ მოგვიწევს ფორმულის რამოდენიმეჯერ ჩანერა.

რათქმაუნდა მის ეფექტურობას უკეთ შევიგრძნობთ, როცა საქმე გვექნება უფრო რთულ ამოცანებთან.

მართვადი კონსტრუქციები და ციკლები

როგორც უკვე იცით, პროგრამა სრულდება იმავე თანმიმდევრობით, რა თანმიმდევრობითაც არის ჩანერილი პროგრამული კოდი, ზემოდან—ქვემოთ. მაგრამ ხშირად საჭიროა შეიცვალოს პროგრამის შესრულების თანმიმდევრობა, იმის მიხედვით, თუ როგორაა ის დამოკიდებული გარკვეულ პირობებზე—საწყის ან შუალედურ მონაცემებზე (განშტოებული სტრუქტურის ალგორითმები).

Visual Basic-ში, ისევე როგორც დაპროგრამების სხვა ენებში, არსებობს მართვადი კონსტრუქციები (პირობითი კონსტრუქციები), რომლებიც განკუთვნილია ბრძანებების შესრულების თანმიმდევრობის სამართავად.

გამოყოფენ მართვადი ოპერატორების (პირობითი ოპერატორების) შემდეგ ძირითად ტიპებს:

- **If** – “თუ” შეუძლია მიიღოს ორი მნიშვნელობა: True/False.
- **Select Case** – “თუ” განმსაზღვრელი პირობა არის გამოსახულება, რომელსაც შეუძლია მიიღოს ორზე მეტი მნიშვნელობა (მაგალითად, კლავიატურიდან მიწოდებული სიმბოლო შეიძლება იყოს ასო, ციფრი, პუნქტუაციის ნიშანი).
- **Try catch** – გამოიყენება გამონაკლისების დამუშავებისათვის. საშუალებას გვაძლევს პროგრამის შესრულების პროცესში გამონაკლისის წარმოქმნისას, შეასრულოს გარკვეული ოპერატორები.

შედარების ოპერატორები

მართვად კონსტრუქციაში გადაწყვეტილების მიღების საფუძველს წარმოადგენს პირობითი გამოსახულება.

პირობითი გამოსახულება—ეს ისეთი გამოსახულებაა, რომელიც აბრუნებს ერთ-ერთს ორი მნიშვნელობიდან True და False. თუ სრულდება პირობა—დააბრუნებს **True**-ს, თუ არა **False**-ს.

პირობით გამოსახულებაში გამოიყენება შედარების ოპერატორები, რომლებიც მოყვანილია ცხრილში.

ცხრილი: შედარების ოპერატორები.

ოპერატორი	აღწერა
=	ტოლია
>	მეტია
<	ნაკლებია
<>	არ უდრის
>=	მეტია ან ტოლია
<=	ნაკლებია ან ტოლია

პირობით გამოსახულებაზე, შეიძლება შესრულდეს მათემატიკური და ლოგიკური მოქმედებები (ლოგიკური ოპერაციები).

- **AND** (და)—აბრუნებს მნიშვნელობას True, თუ ოპერაციაში მონაწილე ყველა გამოსახულებას აქვს მნიშვნელობა True. სხვა შემთხვევაში ბრუნდება მნიშვნელობა False.
- **OR** (ან)—აბრუნებს მნიშვნელობას True, თუ ოპერაციაში მონაწილე ერთ გამოსახულებას მაინც აქვს მნიშვნელობა True. წინააღმდეგ შემთხვევაში აბრუნებს False-ს.

- XOR—აბრუნებს True-ს, თუ ოპერაციაში მონაწილე მხოლოდ ერთ გამოსახულებას აქვს მნიშვნელობა True. სხვა შემთხვევაში ბრუნდება მნიშვნელობა False.
- NOT (არა)—უარყოფის ოპერაცია. თუ გამოსახულება უდრის True-ს, მაშინ აბრუნებს False-ს, და პირიქით, თუ გამოსახულების მნიშვნელობა უდრის False-ს მაშინ ბრუნდება მნიშვნელობა True.

ლოგიკური ოპერაციის სინტაქსი იგივეა, რაც არითმეტიკულის, მაგალითად:

(გამოსახულება1 AND გამოსახულება2 AND გამოსახულება3) OR (გამოსახულება4). ფრჩხილები ლოგიკურ გამოსახულებაში მოქმედებენ ისევე, როგორც არითმეტიკულში. პირველად სრულდება ის მოქმედება, რომელიც მოთავსებულია ფრჩხილებში.

პირობითი კონსტრუქცია **if ...Then, If ...Then ...Else**

კონსტრუქცია if . . . Then

კონსტრუქცია **if . . . Then** გამოიყენება იმ შემთხვევაში, როდესაც აუცილებელია გარკვეული პირობის (პირობების) არსებობისას, შესრულდეს ოპერატორი (ბრძანება) ან ოპერატორთა ჯგუფი. ოპერატორები შესრულდება თუ მოცემული პირობის მნიშვნელობა უდრის True-ს.

არსობს **if . . . Then** ოპერატორის ორი ნაირსახეობა: ერთსტრიქონიანი და მრავალსტრიქონიანი. ერთსტრიქონიან ოპერატორს აქვს შემდეგი სინტაქსი:

If პირობა Then ოპერატორები

ამ ოპერატორში პირობა და პირობის შესრულებისას განსახორციელებელი მოქმედება განლაგებულია ერთ სტრიქონში. ერთ სტრიქონში ასევე შეიძლება ჩაინეროს რამოდენიმე ოპერატორი, რომლებიც ერთმანეთისაგან გამოიყოფა ორწერტილით.

if $A > 10$ Then $A = A + 1 : B = B + A : C = C + B$

იმ შემთხვევისთვის, თუ პირობის შესრულებისას საჭიროა შესრულდეს ოპერატორების ბლოკი (არაერთი ოპერატორი), უმჯობესია გამოვიყენოთ მრავალსტრიქონიანი პირობითი ოპერატორი, რომელსაც აქვს შემდეგი სინტაქსი:

if პირობა Then

ოპერატორები
.....

End if

შემდეგში მოყვანილი პროგრამული კოდის ფრაგმენტები ასრულებენ ერთიდაიმავე მოქმედებას:

ერთსტრიქონიანი ოპერატორი

if $y > 10$ Then $y = 2$

მრავალსტრიქონიანი ოპერატორი

if $y > 10$ Then

$y = 2$

End if

ერთსტრიქონიანი ოპერატორი

If $X > 0$ then Text = “რიცხვი დადებითია”

მრავალსტრიქონიანი ოპერატორი

If $X > 0$ then

Text = “რიცხვი დადებითია”

Text2 = “რიცხვი არ არის უარყოფითი”

End If

If კონსტრუქციას მოსდევს ლოგიკური გამოსახულება, რომელიც შეიცავს პირობას. პირობისათვის გამოიყენება ლოგიკური გამოსახულება.

საკვანძო სიტყვა End If აღნიშნავს მრავალსტრიქონიანი კონსტრუქციის დასასრულს და მისი არსებობა ამ შემთხვევაში აუცილებელია. თუ მითითებული პირობა სრულდება, ანუ შემონმების შედეგი უდრის True-ს, მაშინ Visual Basic შეასრულებს ოპერატორებს (ბრძანებებს) რომელიც მოსდევს Then ოპერატორს.

თუ პირობა არ სრულდება, მაშინ Visual Basic შეასრულებს End If-ის შემდეგ მდგომ ოპერატორებს.

კონსტრუქცია if... Then ... Else

კონსტრუქცია if... Then ... Else, if... Then კონსტრუქციის ანოლოგიურია, მაგრამ იძლევა საშუალებას შესრულდეს გარკვეული ოპერატორები პირობის შესრულებისას და გარკვეული ოპერატორები მისი შეუსრულებლობის შემთხვევაში. If(თუ), Else (თუ არა).

კონსტრუქციას აქვს შემდეგი სინტაქსი:

if პირობა Then

ოპერატორები (რომლებიც შესრულდება თუ პირობა სრულდება)

Else

ოპერატორები (რომლებიც შესრულდება თუ პირობა არ სრულდება)

End if

საკვანძო სიტყვას *if* და *End if* აქვთ ისეთივე აზრი, რაც *If . . . Then* კონსტრუქციისას.

თუ მოცემულ კონსტრუქციაში პირობა არ სრულდება (შემომნების შედეგი არის *False*) *Visual Basic* შეასრულებს ბრძანებებს, რომლებიც განლაგებულია *Else*-ს შემდეგ.

მაგალითად:

if x >= 0 Then

Label1.Text = "რიცხვი მეტია ან უდრის 0-ს"

Else

Label1.Text = "რიცხვი ნაკლებია 0-ზე"

End if

If ბრძანებას შეუძლია შეამონმოს მხოლოდ ერთი პირობა. თუ თქვენ მოგიწევთ განახორციელოთ გადასვლა რამოდენიმე პირობის შემომნების შედეგების გათვალისწინებით, ასეთი შესაძლებლობაც არსებობს. დამატებითი პირობა შეიძლება მიეცეს ოპერატორ *ElseIf*-ის დახმარებით და ის შემომნდება იმ შემთხვევაში, როცა წინა პირობა მცდარია. მაგალითად:

If x >= 0 Then

Label1.Text = "რიცხვი დადებითია"

ElseIf x = 0 Then

Label1.Text = "რიცხვი უდრის ნულს"

Else

Label1.Text = "რიცხვი უარყოფითია"

End If

საკვანძო სიტყვა ElseIf შეიძლება გამოყენებულ იქნას რამოდენიმეჯერ, მაგრამ საკვანძო სიტყვა Else-მდე.

ოპერატორები If . . . Then, If . . . Then, . . . Else შეიძლება მოთავსებული იყოს ერთმანეთში.

პირობითი კონსტრუქცია Select Case

კონსტრუქცია Select Case იძლევა რამოდენიმე პირობის დამუშავების საშუალებას. ის არის If . . . Then . . . Else კონსტრუქციის ანალოგიური. ეს კონსტრუქცია შედგება გასაანალიზირებელი გამოსახულებისაგან და Case ოპერატორების ნაკრებისაგან.

მოცემული კონსტრუქცია მუშაობს შემდეგნაირად: თავდაპირველად Visual Basic გამოითვლის კონსტრუქციაში მოცემული გამოსახულების მნიშვნელობას. შემდეგ მიღებულ მნიშვნელობას ადარებს მნიშვნელობებს, რომლებიც მოცემულია ოპერატორ Case-ს კონსტრუქციებში. თუ მოინახება საძებნი მნიშვნელობა, სრულდება ბრძანება, რომელიც მიკუთვნილია მოცემულ Case ოპერატორზე. კონსტრუქციის შესრულების დამთავრების შემდეგ, მართვა გადაეცემა კონსტრუქციას, რომელიც საკვანძო სიტყვა End Select-ის შემდეგა მოდის.

Select Case კონსტრუქციის სინტაქსი შემდეგია:

```
Select Case შესადარებელი მნიშვნელობა (ცვლადი, კონსტანტა)
    Case მნიშვნელობა 1
        ოპერატორი 1
    Case მნიშვნელობა 2
        ოპერატორი 2
```

```

. . . . .
Case Else
    ოპერატორი N
End Select

```

კონსტრუქციის დასაწყისში განთავსებულია საკვანძო სიტყვა *Select Case*, რომელიც მიუთითებს, რომ მის გვერდით მდგომი პარამეტრიც (შესადარებელი მნიშვნელობა), შემონმდება რამოდენიმე პირობაზე.

შემდეგ კონსტრუქციაში განთავსებულია ბრძანებების ჯგუფი, რომელიც იწყება საკვანძო სიტყვა *Case*-ით. თუ პარამეტრი *შესადარებელი მნიშვნელობა* უდრის მნიშვნელობას, რომელიც მითითებულია ოპერატორ *Case*-ში, მაშინ შესრულდება ოპერატორები, რომლებიც განლაგებულია მას და შემდეგ საკვანძო სიტყვა *Case*-ს შორის.

კონსტრუქცია შეიძლება შეიცავდეს საკვანძო სიტყვა *Case*-ს ნებისმიერ რაოდენობას, შესაბამისი ოპერატორების ბლოკით.

თუ არცერთი პირობა არ შესრულდა, შესრულდება ოპერატორები, რომლებიც საკვანძო სიტყვა *Case Else*-ს შემდეგ მოდის. საკვანძო სიტყვა *Case Else*-ს არსებობა აუცილებელი არ არის.

მაგალითი:

```

Select Case x
Case 1 To 9
    Label1.Text = "მნიშვნელობა მეტია ნულზე"
Case 0
    Label1.Text = "მნიშვნელობა უდრის ნულს"
Case -1 To -9
    Label1.Text = "მნიშვნელობა ნაკლებია ნულზე"
End Select

```


მოდით აქ ცოტახნით შევისვენოთ თეორიისაგან და გადავიდეთ პრაქტიკაზე. შევექმნათ პროექტი, რომელიც საშუალებას მოგვცემს შევცვალოთ პროგრამის ეკრანის ფერი. ამისათვის გამოვიყენოთ პირობითი ოპერატორები.

თავდაპირველად ჩამოვაცალიბოთ ამოცანა. ის თუ რის გაკეთებას ვთხოვთ ჩვენს მიერ შექმნილ პროგრამას (პროექტს). ტექსტურ ბლოკში ჩვენ ქართულად ჩავწერთ ფერების დასახელებას (ნიტელი, ყვითელი, შავი, ლურჯი დას ხვ.), პროგრამა კი შესაბამისი ფერით შევლებავს ფორმას.

ამოცანის ჩამოყალიბების შემდეგ უნდა მოვიფიქროთ მისი შესრულების ალგორითმი. ჩვენ ამისათვის აუცილებლად უნდა გამოვიყენოთ რომელიმე პირობითი ოპერატორი.

ალგორითმი სიტყვიერი ფორმით იქნება შემდეგი: თუ ტექსტურ ბლოკში ჩავწერთ სიტყვას “ნითელი” ფორმამ მიიღოს ნითელი ფერი, თუ ჩავწერთ “მწვანე” ფორმამ მიიღოს მწვანე ფერი და ასშ.

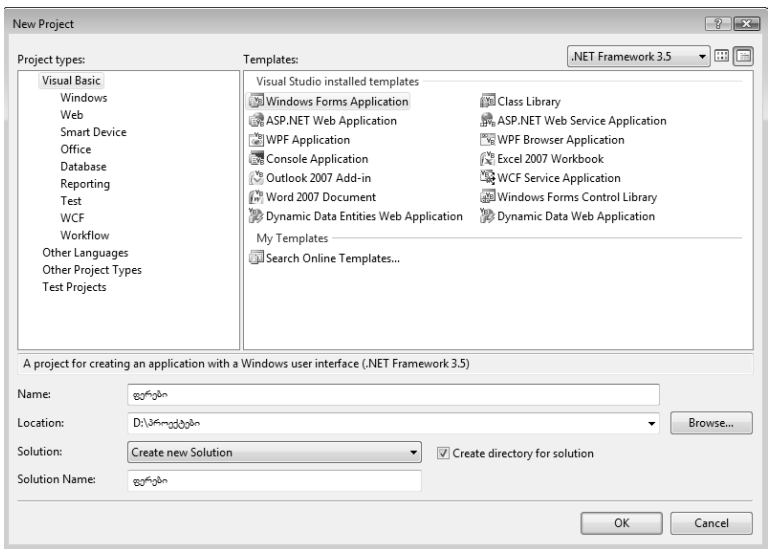
- გაუშვით პროგრამა **Visual Basic 2008** (ან 2010).
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project** (პროექტის შექმნა).
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანექით ლილაკს **New Project** .
- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Windows Forms Application (Windows** დანართი).
- ველში **Name** ჩაწერეთ შესაქმნელი პროექტის სახელი.
- შემდეგ დააჭირეთ **OK** ლილაკს.

*Visual Basic 2008–ში პროექტის შენახვა ხდება დიალოგის ფანჯარის **Save Project** დახმარებით, რომელიც იხსნება*

ბრძანებით **Save All** მენიუდან **File** ან სტანდარტული ინსტრუმენტების პანელიდან შესაბამისი ლილაკით.

შეიძლება პროექტის შენახვა მისი შექმნისთანავე. ამისათვის აუცილებელია შეიცვალოს შესაბამისი რეგულირება, განყოფილებაში **projects and Solutions** მენიუ **Tools**-ის ფანჯარაში **Options**. მიუთითეთ მონიშვნა **Save new projects when created** (ახალი პროექტის შენახვა შექმნისთანავე).

თუ პროექტის შენახვა ხდება მისი შექმნის თანავე, მაშინ დიოლოგის ფანჯარაში **New Project** გარჩდება დამატებითი ველი **Location**, სადაც მითითებულია გზა პტოექტისაკენ და მისი სახელი.



გაიხსნება ცარიელი ფორმა.

გავიხსენოთ, რომ ვიზუალური პროგრამირება ეს არის ვიზუალური ინტერფეისის პლიუს პროგრამული კოდი. ე.ი. პირველ რიგში უნდა შევქმნათ ვიზუალური ინტერფეისი.

ცარიელ ფორმაზე ელემენტთა პანელიდან გადმოვიტანოთ ტექსტური ბლოკი **TextBox** და **Label**. დავაკლიკოთ ფორმაზე თვისებათა ფანჯარაში მოვნახოთ თვისება **Text** და მის

გვერდით არსებული ტექსტის ნაცვლად ჩავწეროთ: “ფერები”. ასევე მოვნიშნოთ ელემენტი აბელ და მის თვისებაში Text შევიყვანოთ “შეიყვანეთ ფერი”.

თუ გვინდა რომ ფორმაზე ან სხვა ელემენტზე გვექონდეს ქართული წარწერა, ამისათვის არსებობს 2 საშუალება.

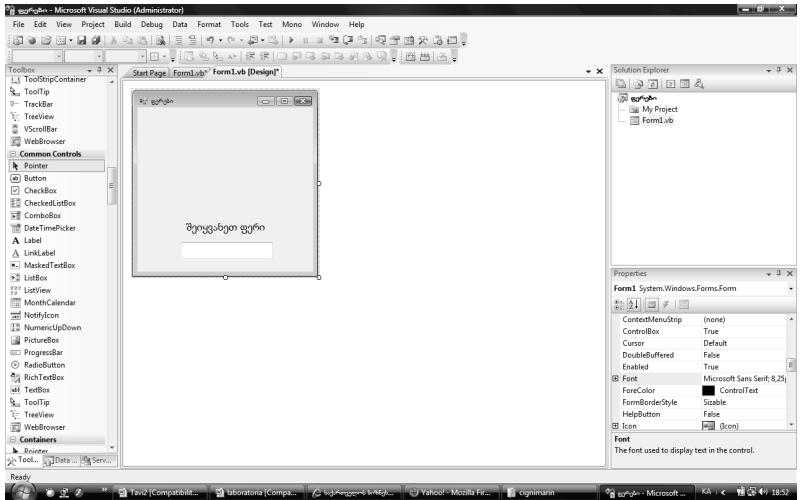
- ჩავწეროთ ქართული ტექსტი ლათინური სიმბოლოებით და შემდეგ თვისებათა ფანჯარაში მოვნახოთ ელემენტის თვისება **Font**, მის გვერდზე დაჭერით გაიხსნება შესაბამისი ფანჯარა, სადაც ჩვენ შეგვიძლია შევცვალოთ ფონტი (ავირჩიოთ ქართული) შევცვალოთ მისი ზომა (**Size**), სისქე (**Bold**), დახრილობა (**Italic**) და სხვ. ქართული ფონტის არჩევის შემდეგ, წარწერა ელემენტზე **Button** გახდება ქართული☺.
- არსებობს მეორე და უფრო მარტივი ვარიანტი. ტექსტი ავკრიფოთ ქართული

Unicod-ის სიმბოლოებით. **Visual studio**-ს გააჩნია **Unicod**-ის მხარდაჭერა (ჩვენ წიგნში გამოვიყენებთ ამ უკანასკნელ მეთოდს).

*ერთმანეთისაგან არ უნდა შეგვეშალოს თვისება **Name** და თვისება **Text**. თავდაპირველად (უჩუმრად) ელემენტების ეს თვისებები ერთნაირია. თვისება **Text** წარმოადგენს წარწერას ელემენტზე. თვისება **Name** კი არის ელემენტის სახელი, რომლითაც მიმართავს მას პროგრამა.*

ახლა მოდით გავაცოცხლოთ ჩვენი პროგრამა (ჯერ-ჯერობით ის ვერაფერს გააკეთებს). ამისათვის საჭიროა ჩავწეროთ პროგრამული კოდი **Basic**-ის ენაზე. რასაკვირველია კოდი სადღაც უნდა ჩაიწეროს. პირველი ამოცანა იქნება ის თუ სად ჩავწეროთ კოდი. გვახსოვდეს პროგრამირების ერთ-ერთი ძირითადი კანონი: პროგრამა რეაგირებს მოვლენაზე. ე.ი. პროგრამული კოდი გაეშვება რაღაც მოვლენასთან მიმართებაში. რა შეიძლება იყოს მოვლენა? ერთ შემთხვევაში შეიძლება იყოს ღილაკზე დაჭერა, მეორე შემთხვევაში ღილაკზე აშვება, მესამე შემთხვევაში პროგრამის გაშვება, მეოთხე შემთხვევაში ტექსტის ცვლილება და ასე შემდეგ.

მოცემულ შემთხვევაში ჩვენი პროექტისათვის უნდა ჩაინეროს კოდი, რომელიც გაეშვება ღილაკზე **TextBox**-ში ტექსტის ცვლილებისას, რადგანაც სწორედ მისი შეცვლისას უნდა შეიცვალოს ფორმის ფერები.



დავაჭიროთ სწრაფად ორჯერ **TextBox**-ს გაიხსნება პროგრამული კოდის ფანჯარა. დავინახავთ შემდეგ კოდს:

```
Public Class Form1
```

```
Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles TextBox1.TextChanged
```

```
End Sub
```

```
End Class
```

კურსორი ციმციმებს იმ უბანში სადაც ჩვენ უნდა ჩავწეროთ კოდი, თუ გვინდა რომ ის გაეშვას ტექსტის ცვლილებისას.

მის ზემოთ ნარწერა `Private Sub TextBox1_TextChanged` აღნიშნავს, რომ ბრძანება შესრულება ტექსტის `TextBox1` ცვლილებისას. პროგრამა რეაგირებს მოვლენაზე “ცვლილება” ანუ `Changed`. მოვლენა ყოველთვის მთავრდება ბრძანებით `End Sub` (გამოდის ავტომატურად). მთელი კოდი კი იწყება `Public Class`-ით და მთავრდება `End Class`-ით. როგორც მიხვდით მთელი კოდის ხელით შეყვანა არ მოგიწევთ, მის ნაწილს **visual studio** თქვენს მაგივრად გააკეთებს.

ჩვენერთ შემდეგი კოდი იქ სადაც კურსორი ციმციმებს (ქართული სიტყვები აკრიფეთ უნიკოდით):

```
If TextBox1.Text = "წითელი" Then BackColor = _
Color.Red
If TextBox1.Text = "მწვანე" Then BackColor = _
Color.Green
If TextBox1.Text = "ლურჯი" Then BackColor = _
Color.Blue
If TextBox1.Text = "შავი" Then BackColor = _
Color.Black
If TextBox1.Text = "ყვითელი" Then BackColor=_
Color.Yellow
```

საბოლოო კოდი მიიღებს სახეს:

```
Public Class Form1

    Private Sub TextBox1_TextChanged(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
TextBox1.TextChanged
        If TextBox1.Text = "წითელი" Then BackColor = _
Color.Red
        If TextBox1.Text = "მწვანე" Then BackColor = _
Color.Green
        If TextBox1.Text = "ლურჯი" Then BackColor = _
Color.Blue
    End Sub
End Class
```

```

        If TextBox1.Text = "შავი" Then BackColor = _
Color.Black
        If TextBox1.Text = "ყვითელი" Then BackColor = _
Color.Yellow
    End Sub
End Class

```

ახლა კი დადგა დრო გამოვცადოთ ჩვენი პროგრამა. ამისათვის უნდა გაფუშვავთ ის. (ერთდროულად ხდება მისი მანქანურ ენაზე გადაყვანა). პროგრამის გაშვება ხდება შემდეგნაირად:

- მენიუთა სტრიქონში ავირჩიოთ **Debug** და ჩამოშლილ მენიუში **Start Debugging**.
- ან დავაჭიროთ სამკუთხა ფორმის ლილაკს ლილაკების სტრიქონში.

პროგრამა გაეშვება. შევიყვანოთ ფერის დასახელება ტექსტურ ბლოკში (ქართულად, უნიკოდით) იქმნება შთაბეჭდილება, რომ პროგრამამ ქართული იცის 😊.

ახლა მოდით შევქმნათ იგივე პროექტი პირობითი ოპერატორის **Select Case** გამოყენებით.

ნინა კოდის ნაცვლად ჩავენეროთ შემდეგი კოდი:

```

Select TextBox1.Text

Case "წითელი"
    BackColor = Color.Red
Case "მწვანე"
    BackColor = Color.Green
Case "ლურჯი"
    BackColor = Color.Blue

Case "შავი"
    BackColor = Color.Black
Case "ყვითელი"

```

```
        BackColor = Color.Yellow  
  
    End Select
```

საბოლოო კოდი მიიღებს სახეს:

```
Public Class Form1  
  
    Private Sub TextBox1_TextChanged(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
TextBox1.TextChanged  
        Select TextBox1.Text  
            Case "წითელი"  
                BackColor = Color.Red  
            Case "მწვანე"  
                BackColor = Color.Green  
            Case "ლურჯი"  
                BackColor = Color.Blue  
  
            Case "შავი"  
                BackColor = Color.Black  
            Case "ყვითელი"  
                BackColor = Color.Yellow  
        End Select  
  
    End Sub  
End Class
```

გავუშვათ პროგრამა, ჩვენ ვნახავთ რომ მიუხედავად კოდის შეცვლისა პროგრამა მაინც გამართულად მუშაობს. ანუ შესაძლებელია ორივე პირობითი კონსტრუქციის გამოყენება. უნდა ავირჩიოთ ის კონსტრუქცია, რომელიც მოცემულ შემთხვევაში უფრო მოსახერხებელია. ცონსტრუქცია **Select Case** უფრო მოსახერხებელია როცა მგვაქვს მრავალი პირობა.

პროექტი “შუქნიშანი”

შევქნათ პროექტი “შუქნიშანი”. თავდაპირველად ჩამოვყალიბოთ ამოცანა. ის თუ რის გაკეთებას ვთხოვთ ჩვენს მიერ შექმნილ პროგრამას (პროექტს). ეკრანზე უნდა გვექონდეს შუქნიშანის სურათი და ის უნდა იყოს რეალური ანუ იცვლიდეს ფერებს საჭირო თანმიმდევრობით და საჭირო დროის ინტერვალით.

ამოცანის ჩამოყალიბების შემდეგ უნდა მოვიფიქროთ მისი შესრულების ალგორითმი. ჩვენ ამისათვის აუცილებლად უნდა გამოვიყენოთ რომელიმე პირობითი ოპერატორი, მაგრამ როგორ მოვიქცეთ დროის ინტერვალთან მიმართებაში? ამისათვის **Visual Basic**-ს გააჩნია მართვის ელემენტი **Timer**. რომელსაც შეუძლია მოგვცეს ჩვენთვის სასურველი დროის ინტერვალები.

ალგორითმი სიტყვიერი ფორმით იქნება შემდეგი: ტექსტურ ბლოკში თავდაპირველად ჩაინერება ციფრი “0”. ელემენტი **Timer** უზრუნველყოფს იმას, რომ მას დროის გარკვეულ ინტერვალში დაემატება 1 (გახდება 2,3 და ასე, ანუ ვიყენებთ პირდაპირ მთვლელს). როცა ტექსტურ ბლოკში წერია “1” გამოჩნდება შუქნიშანზე ნითელი ფერი, სხვა ფერები ჩაქრება, როცა ბლოკში წერია “2” გამოჩნდება ყვითელი ფერი, სხვა ფერები ჩაქრება. როცა ბლოკში წერია “3” გამოჩნდება მყვანე ფერი, სხვა ფერები ჩაქრება. ბოლოს კი ჩავწერთ ალგორითმს: თუ ტექსტურ ბლოკში წერია 4 შემდეგ ის შეიცვლება და ჩაენერება 1 ანუ ყველაფერი დაინყება თავიდან.

რაც შეეხება შუქნიშანს ჩვენ გამოვიყენებთ მის სურათს, რომელიც ჩანერილია **CD** დისკზე. აქვე გავეცნობით ახალ მართვის ელემენტს **Picture Box**. შუქნიშანის ფერებად გამოვიყენებთ ტექსტურ ბლოკებს. შუქნიშანი რომ

რეალური იყოს და იქმნებოდეს ილუზია რომ ფერი ნათურა მართლა აინთო ჩვენ გამოვიყენეთ ერთმანეთზე მოთავსებული 2 ტექსტური ბლოკი. ორივე შეფერილია ერთიდაიმავე ფერში ოღონდ ერთის ფერი უფრო მუქია ვიდრე მეორესი. როცა შუქნიშნის მოცემული ფერის “ანთებისას” ერთი ტექსტური ბლოკი გაქრება და მეორე გამოჩნდება, შეიქმნება შუქნიშნის ფერის ანთების ილუზია.

- გაუშვით პროგრამა **Visual Basic 2008** (ან 2010).
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project** (პროექტის შექმნა).
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანეკით ლილაკს **New Project**.
- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Windows Forms Application (Windows** დანართი).
- ველში **Name** ჩანერეთ შესაქმნელი პროექტის სახელი.
- შემდეგ დააჭირეთ **OK** ლილაკს.

გაიხსნება ცარიელი ფორმა.

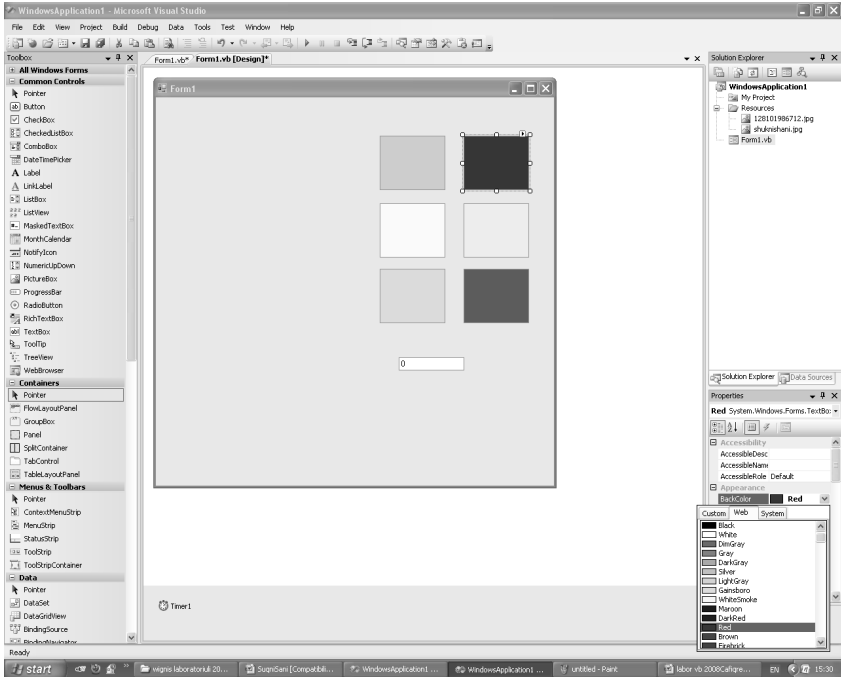
Properties (თვისებათა ფანჯარა) -დან ფორმას

მივანიჭოთ სახელი – შუქნიშანი , ვიყენებთ თვისებას **Text**.

ელემენტთა პანელიდან ფორმაზე მოვათავსოთ მართვის 7 ელემენტი **Textbox** და დავარქვათ მათ შესაბამისი სახელები, ამისათვის თვისებათა ფანჯარიდან–დან ვიყენებთ თვისებას (**Name**) სადაც თითოეულ **Textbox**-ს მივანიჭებთ სახელებს: **Red, Yellow, green, pink, light yellow, pale green** და **date**.

ელემენტ **date**-ში **Properties**-დან თვისება **Text**-ში ჩავწეროთ 0.

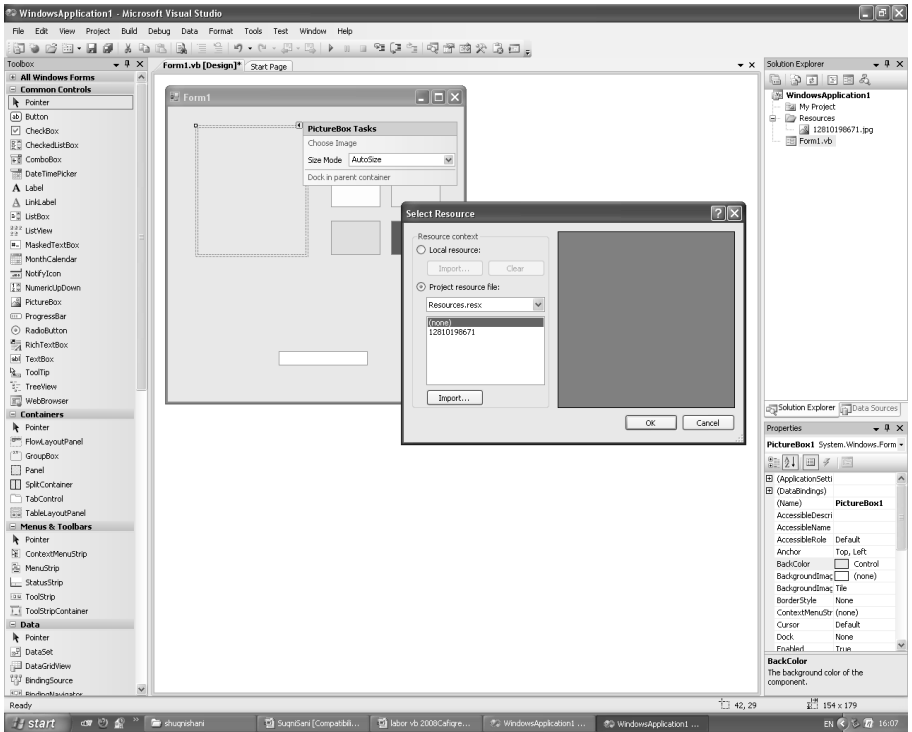
აგრეთვე თვისება BackColor-დან ელემენტებს Rad, Yellow, green, pink, light yellow, pale green მივანიჭოთ შესაბამისი ფერები (მათი სახელების შესაბამისი).

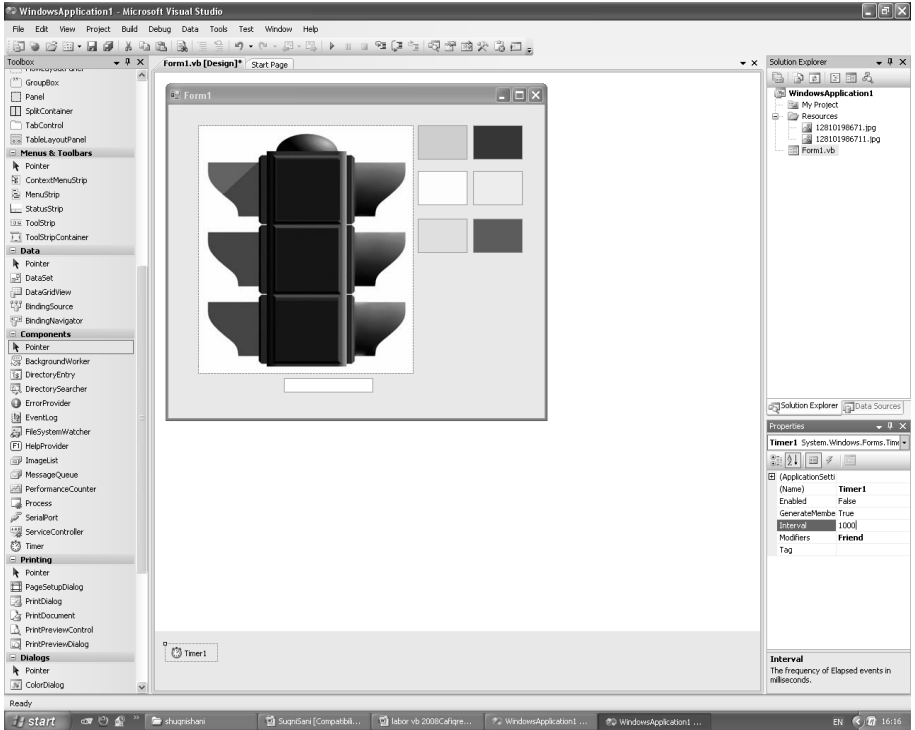


ჩავსვათ შუენიშნის სურათი, ამისათვის ელემენტთა პანელიდან **Toolbox** ავირჩიოთ ელემენტი **PictureBox**. და გადმოვიტანოთ ფორმაზე. მოვნიშნოთ ის და დავაჭიროთ პატარა სამკუთხედს მის ზედა მარჯვენა კიდეზე, გაიხსნება მენიუ სადაც ავირჩიოთ **Choose Image**. შემდეგ დავაჭიროთ ლილაკს **Import**, მოვნახოთ სურათი რომელიც მდებარეობს CD დისკზე და დავაჭიროთ ლილაკს **OK**.

Size Mode-ს მივანიჭოთ **AutoSize**.

შემდეგ **Toolbox** ელემენტთა პანელიდან ავირჩიოთ ელემენტი **Timer**. და გადმოვიტანოთ ფორმაზე. თვისებათა ფანჯარა-დან **Timer**-ის **interval**-ს მივანიჭოთ 5000 (რაც მეტია ინტერვალი მით უფრო იშვიათად იცვლება მისი მოვლენა **Timer Tick**). თუ ქვინდა რომ ტაიმერი ჩაირთოს ფორმის გაშვებისთანავე მის თვისებას **Enabled** მივანიჭოთ თვისება **True**.





Textbox-ები დავალაგოთ შუენიშანზე შემდეგი თანმიდევრობით: pink, light yellow, pale green ზემოდან დავალაგოთ Rad, Yellow, green.

ახლა კი ჩაწეროთ პროგრამული კოდი. ტაიმერი გაეშვება პროგრამის გაშვებისთანავე და მას მოვლენას Tick უნდა დაეუკავშიროთ შემდეგი კოდი (დავაკლიკოთ ტაიმერზე ორჯერ და ჩაწეროთ კოდი):

```
date.Text = date.Text + 1
```

ასე შევქმნით პირდაპირ მთვლელს.

ახლა კო როცა ჩვენს ტექსტურ ბლოკში (**date**) რიცხვები იცვლება ჩვენთვის სასურველი თანმიმდევრობით, შეგვიძლია პროგრამული კოდი დავუკავშიროთ მის მოვლენას **date_TextChanged**.

დავაკლიკოთ ელემენტ **date** –ზე და ჩავწეროთ საჭირო კოდი

```
If date.Text = 1 Then
    Rad.Visible = True
    Yellow.Visible = False
    green.Visible = False
End If
If date.Text = 2 Then
    Rad.Visible = False
    Yellow.Visible = True
    green.Visible = False
End If
If date.Text = 3 Then
    Rad.Visible = False
    Yellow.Visible = False
    green.Visible = True
End If
If date.Text = 4 Then date.Text = 0
```

საბოლოო კოდი მიიღებს შემდეგ სახეს:

```
Public Class Form1

    Private Sub Timer1_Tick(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Timer1.Tick
        date.Text = date.Text + 1
    End Sub

    Private Sub date_TextChanged(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles TextBox4.TextChanged
        If date.Text = 1 Then
            Rad.Visible = True
            Yellow.Visible = False
```

```

        green.Visible = False
    End If
    If date.Text = 2 Then
        Rad.Visible = False
        Yellow.Visible = True
        green.Visible = False
    End If
    If date.Text = 3 Then
        Rad.Visible = False
        Yellow.Visible = False
        green.Visible = True
    End If
    If date.Text = 4 Then date.Text = 0
End Sub

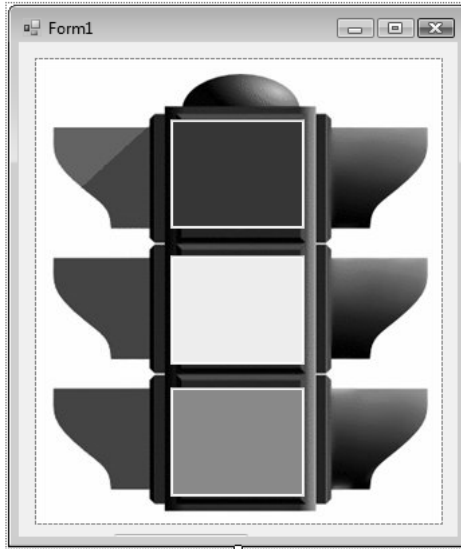
```

ვფიქრობთ რომ თუ როგორ მუშაობს კოდში პირობითი ოპერატორები თქვენით მარტივად გაერკვევით.

რაც შეეხება ელემენტის თვისება **Visible** ის გააჩნია უმრავლეს მართვის ელემენტს. **Visible** წარმოადგენს **Boolean** ტიპის ცვლადს, ანუ გააჩნია ორი მნიშვნელობა **True** და **False**. თუ ელემენტის ამ თვისებას გააჩნია მნიშვნელობა **False** ის არ ჩანს პროგრამის მუშაობისას თუ **True** მაშინ მას ვხედავთ.

შეამცირეთ ფორმის ზომები შუქნიშნის ზომებამდე და ტექსტური ბლოკის თვისებას **Visible** მიანიჭეთ მნიშვნელობა **false**. პროგრამის მუშაობისას ტექსტური ბლოკი არ გამოჩნდება, მაგრამ თავის ფუნქციას შეასრულებს.

გაუშვით პროგრამა. შუქნიშანი ჩაირთვება. შეგვიძლია ვანარმოოთ ექსპერიმენტები **Timer**-ის თვისებაზე **Interval**. მისი ცვლილებით შუქნიშნის ფერთა ცვლილების სიხშირე შეიძლება გავზარდოთ ან შევამციროთ.



პროექტი “ევკლიდეს ალგორითმი”



შევქნათ პროექტი “ევკლიდეს ალგორითმი”.
თავდაპირველად ჩამოვყალიბოთ ამოცანა. ეკრანზე უნდა

გვეკონდეს ორი ტექსტური ბლოკი, მათში შევიყვანოთ რიცხვებს ლილაკზე დაჭერით კი უნდა მივიღოთ ამ ორი რიცხვის უდიდესი საერთო გამყოფი.

ამოცანის ჩამოყალიბების შემდეგ უნდა მოვიფიქროთ მისი შესრულების ალგორითმი. რაც შეეხება ამ ამოცანის ალგორითმს ის დიდი ხნის წინ მოგვანოდა ევკლიდემ. ორი რიცხვის უდიდესი საერთო გამყოფის მოსაძებნად უდიდესს უნდა გამოვაკლოთ უმცირესი და უდიდესს უნდა მივანიჭოთ ნაშთის მნიშვნელობა. შემდეგ მიღებულ რიცხვებზე კვლავ გავიმეოროთ იგივე მოქმედება მანამ, სანამ ორივე რიცხვი ერთმანეთის ტოლი არ გახდება. სწორედ ეს რიცხვი იქნება უდიდესი საერთო გამყოფი (უსგ).

ყველაზე თვალსაჩინოდ ეს პროცესი გამოჩნდება რიცხვებზე 95 და 60. მოგიყვანოთ მაგალითს ცხრილის სახით:

a	b
95	60
35	60
35	25
10	25
10	15
10	5
5	5

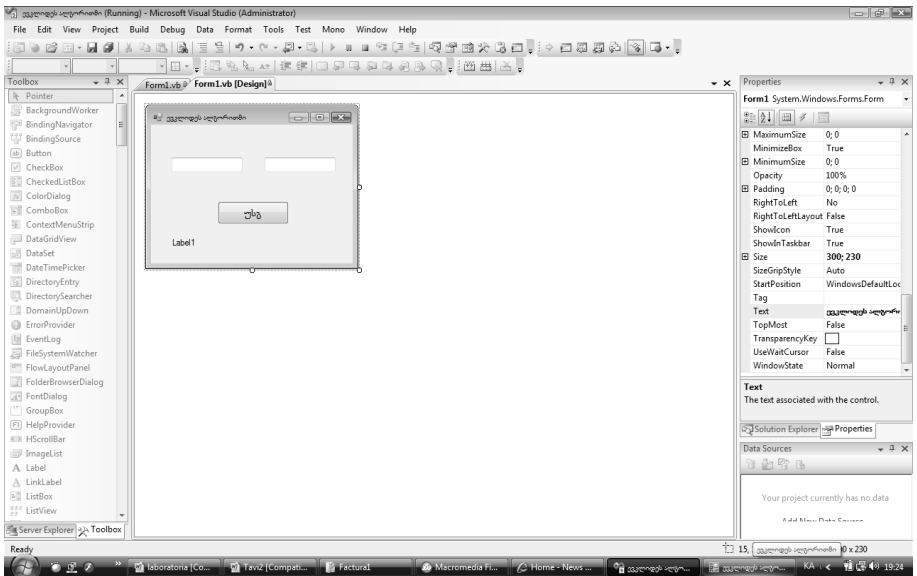
ჩვენი პროგრამა გამოთვლის უდიდეს საერთო გამყოფს და ასევე გვარჩვენებს ყველა ბიჯს, ბოლოს კი გამოიტანს წარწერას "ამ ორი რიცხვის უდიდესი საერთო გამყოფია" და ამ რიცხვს. ლილაკზე უნდა დავაჭიროთ მანამ სანამ შედეგს არ მივიღებთ.

- გაუშვით პროგრამა **Visual Basic 2008**.
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project** (პროექტის შექმნა).
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანექით ლილაკს **New Project**.

- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Windows Forms Application (Windows დანართი)**.
- ველში **Name** ჩანერეთ შესაქმნელი პროექტის სახელი.
- შემდეგ დააჭირეთ **OK** ლილასს.

გაიხსნება ცარიელი ფორმა.

Properties (თვისებათა ფანჯარა) -დან ფორმას მივანიჭოთ სახელი – “ევკლიდეს ალგორითმი”, ვიყენებთ თვისებას **Text**. ელემენტთა პანელიდან ფორმაზე მოვათავსოთ მართვის 2 ელემენტი **Textbox**, ერთი ელემენტი **Button** და ერთი ელემენტი **Label**. მოვნიშნოთ ელემენტი **Button** და თვისებათა ფანჯარაში შევცვალოთ მისი თვისება **Text**. დავანეროთ “უსგ.



ორჯერ დავაკლიკოთ ელემენტზე **Button** და ჩაწეროთ შემდეგი კოდი:

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button1.Click
        If Val(TextBox1.Text) > Val(TextBox2.Text) Then
            TextBox1.Text = Val(TextBox1.Text) - _
Val(TextBox2.Text)
        End If

        If Val(TextBox1.Text) < Val(TextBox2.Text) Then
            TextBox2.Text = Val(TextBox2.Text) - _
Val(TextBox1.Text)
        End If

        If TextBox1.Text = TextBox2.Text Then
            Label1.Text = "ამ ორი რიცხვის უდიდესი _
საერთო გამყოფია " & TextBox1.Text
        End If
    End Sub
End Class
```

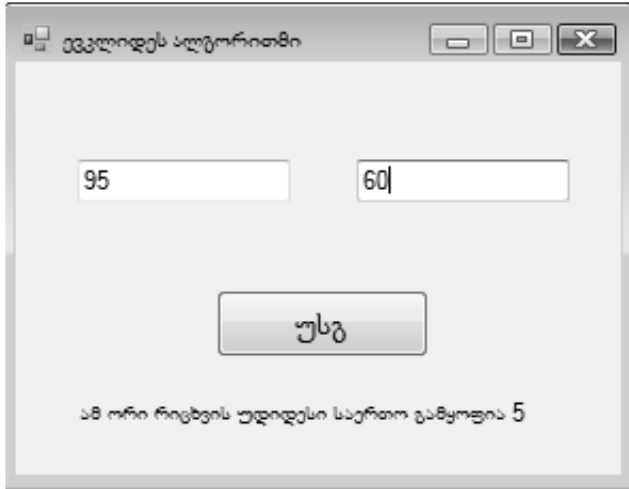
ვფიქრობთ კოდში მარტივად გაერკვევით. რაც შეეხება ოპერატორს **Val**, ის დაგვჭირდა იმისათვის, რომ პროგრამამ ტექსტურ ბლოკში ჩანერილი რიცხვები როგორც რიცხვითი ცვლადები ისე აღიქვას (და არა როგორც ტექსტური ცვლადი).

დავაკვირდეთ ჩვენს მიერ ჩანერილ ოპერატორს:

```
TextBox1.Text = Val(TextBox1.Text) - Val(TextBox2.Text)
```

ის ანალოგიურია კონსტრუქციის $a=a+b$. აღსანიშნავია რომ მათემატიკაში თუ **b** ნულის ტოლი არ არის ეს ტოლობა არაკორექტულია. დაპროგრამებაში კი ის ხშირად გამოიყენება. ის ნიშნავს რომ **a**-ს მიენიჭა ახალი მნიშვნელობა $a+b$. დაპროგრამებაში “=” ნარმოადგენს მინიჭების ოპერატორს.

გავუშვათ პროგრამა. ჩვენერთ რიცხვები ტექსტურ ბლოკებში და დავაჭიროთ ლილაკს “უსგ” მანამ, სანამ არ მივიღებთ რიცხვების უდიდეს საერთო გამყოფს.



პროექტი “კვადრატული განტოლება”

შევქმნათ პროექტი “კვადრატული განტოლება”. პროგრამამ უნდა ამოხსნას კვადრატული განტოლება, დაგვინეროს დისკრიმინანტი და ფესვები. თუ განტოლებას ამონახსნი არა აქვს მოგვცეს შესაბამისი ინფორმაცია.

რაც შეეხება განტოლების ამოხსნის ალგორითმს ის ცნობილია სკოლის კურსიდან. ჩვენ უნდა შევიყვანოთ კომპიუტერში **a, b** და **c** ცვლადები.

$$x_+ = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad x_- = \frac{-b - \sqrt{b^2 - 4ac}}{2a},$$

შეიძლება ალგორითმი დავყოთ შემდეგ ბიჯებად:

1) მოცემული განტოლებისთვის **a, b** და **c** კოეფიციენტის შეყვანა.

2) დისკრიმინანტის გამოთვლა $D = b^2 - 4ac$;

3) დისკრიმინანტის ნიშნის ანალიზი, კვადრატული განტოლების ფესვის ამოხსნა და მისი გამოტანა ფორმაზე (თუ $D > 0$).

4) თუ $D < 0$ მაშინ გამოიტანს შეტყობინებას: " განტოლებას არა აქვს ამონახსნი".

- გაუშვით პროგრამა **Visual Basic 2008** (ან 2010).
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project** (პროექტის შექმნა).
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანექით ლილაკს **New Project**.
- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Windows Forms Application (Windows დანართი)**.
- ველში **Name** ჩანერეთ შესაქმნელი პროექტის სახელი.
- შემდეგ დააჭირეთ **OK** ლილაკს.

გაიხსნება ცარიელი ფორმა.

ელემენტთა პანელიდან გადმოვიტანოთ ფორმაზე 6 ელემენტი **TextBox**, 6 ელემენტი **Label** და ერთი ელემენტი **Button**.

განვათავსოთ ისინი ფორმაზე თანმიმდევრობით ისე როგორც სურათზეა ნაჩვენები. იმისათვის რომ მომხმარებელი მიხვდეს თუ სად უნდა შეიყვანოს ცვლადები და სად გამოიტანს პროგრამა ამონახსნს, განვათავსოთ ტექსტური ბლოკების გვერდით ელემენტები **Label** და შევცვალოთ მათი თვისება **Text** ისე როგორც სურათზეა ნაჩვენები.

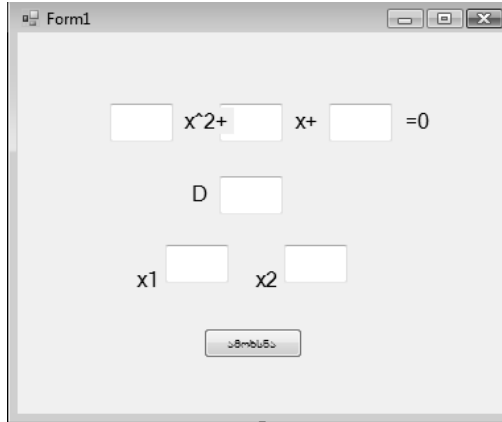
ღილაკს გაუკეთოთ ნარწერა "ამოხსნა".

პროგრამული კოდის დაწერისათვის დაგეჭირდება ცვლადების გამოცხადება და ასევე მათემატიკური ოპერატორები.

ელემენტარული მათემატიკური მოქმედებები **Visual basic**-ში შემდეგნაირად გამოიყურება:

+	პლიუსი
-	მინუსი
/	გაყოფა
*	გამრავლება
^	ახარისხება

გამოვიყენებთ კლასს **Math**. კვადრატულ ფესვს ამოვიღებთ ბრძანებით **Math.Sqrt**.



გავხსნათ პროგრამული კოდის ფანჯარა და გამოვაცხადოთ ცვლადები (ისინი ჩავწეროთ **Public Class Form1**-ის ქვემოთ).

```
Public Class Form1

    Dim a as Decimal
    Dim b as Decimal
    Dim c as Decimal
    Dim D as Decimal
```

დავაკლიკოთ ორჯერ **Button1**-ზე და ჩავწეროთ:

```
Private Sub Button1_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button1.Click
    a = TextBox1.Text
    b = TextBox2.Text
    c = TextBox3.Text

    TextBox4.Text = b ^ 2 - 4 * a * c

    D = TextBox4.Text

    If D >= 0 Then
```

```

        TextBox5.Text = (-b - Math.Sqrt(D)) / 2 * a
        TextBox6.Text = (-b + Math.Sqrt(D)) / 2 * a
    End If

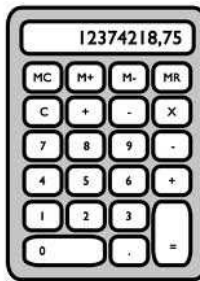
    If D < 0 Then
        Label7.text = "განტოლებას ამონახსნი არა აქვს"
    End If

End Sub
End Class

```

გავუშვათ პროგრამა. ჩვენერთ ნებისმიერი კვადრატული განტოლება. დავაჭიროთ ლილაკს ამოხსნა. მივიღებთ შედეგს. პროგრამა ამოხსნის ნებისმიერ კვადრატულ განტოლებას ანუ უნივერსალურია. უნივერსალობა ალგორითმის ერთ-ერთი აუცილებელი თვისებაა და გულისხმობს იმას, რომ ალგორითმი გამოსადეგი იყოს მსგავსი ტიპის ნებისმიერი ამოცანის ამოსახსნელად.

პროექტი “კალკულატორი”



შევქმნათ პროექტი “კალკულატორი”. ჩვენი კალკულატორი იქნება დაახლოებით ისეთივე როგორც Windows-ის კალკულატორი. რათქმაუნდა შეგვიძლია მივცეთ მას ჩვენთვის სასურველი დიზაინი, დავამატოთ ფუნქციები და სხვ.

- გაუშვით პროგრამა **Visual Basic 2008** (ან 2010).
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project**.
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანექით ლილაკს **New Project**.
- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Windows Forms Application (Windows დანართი)**.
- ველში **Name** ჩაწერეთ შესაქმნელი პროექტის სახელი.
- შემდეგ დააჭირეთ **OK** ლილაკს.

გაიხსნება ცარიელი ფორმა.

გავიხსენოთ, რომ ვიზუალური პროგრამირება ეს არის ვიზუალური ინტერფეისი პლიუს პროგრამული კოდი. ე.ი. პირველ რიგში უნდა შევქმნათ ვიზუალური ინტერფეისი.

ცარიელ ფორმაზე ელემენტთა პანელიდან გადმოვიტანოთ ტექსტური ბლოკი **TextBox** და **17 Button**. დავაკლიკოთ ფორმაზე თვისებათა ფანჯარაში მოვნახოთ თვისება **Text** და მის გვერდით არსებული ტექსტის ნაცვლად ჩავწეროთ: **“CALCULATOR”**.

ასევე მოვნიშნოთ ელემენტი **Button**, შევცვალოთ მათი თვისება **Text** და განვალაგოთ ისინი ისე როგორც კალკულატორის სურათზეა ნაჩვენები (დავანეროთ ციფრები, მოქმედებები და სხვ.). რაც შეეხება ლილაკებს რომლებზეც ციფრებია, **Button1**-ს დავანეროთ 1, **Button2**-ს 2 ბუთონს 3-ს 3 და ასე. **Button10**-ს კი 0.

მოვნიშნოთ ტექსტური ბლოკი **TextBox**. თვისებათა ფანჯარაში შევუცვალოთ მას სახელი **Name** და დავარქვათ **Display**. მისი თვისება **Text** ნავშალოთ და დავტოვოთ ცარიელი.

შევცვალოთ თვისება **BackColor** და შევარჩიოთ შავი ფერი (ჩამოვშალოთ ფერთა პალიტრა მის მარჯვენა მხარეს და

შევარჩიოთ შავი ფერი). შევცვალოთ ასევე თვისება **ForeColor**—შევარჩიოთ ღია მწვანე ფერი (ჩამოვშალოთ ფერთა პალიტრა მის მარჯვენა მხარეს და შევარჩიოთ მწვანე ფერი) ეკრანი გახდება შავი ხოლო მასზე ციფრები ღია მწვანე ფერის იქნება, ამნის ეკრანზე ციფრების ნათების ილუზიას.

შევცვალოთ ასევე **Font**. გავზარდოთ მისი ზომა, მივანიჭოთ მას მნიშვნელობა 18. ამისათვის დავაჭიროთ თვისება **font**-ის მარჯვნივ ღილაკს და გავხსნათ ფანჯარა **Font**.



ამ ეტაპზე ჩვენ გვაქვს შექმნილი კალკულატორის ინტერფეისი. თუ გავუშვებთ პროგრამას ვნახავთ, რომ კალკულატორის ღილაკებს შეგვიძლია დავაჭიროთ, მაგრამ ისინი არანაირ ფუნქციას არ შეასრულებენ. საჭიროა პროგრამული კოდის ჩანერა, რომელიც კალკულატორს აამუშავებს.

ალგორითმი იქნება შემდეგი: ღილაკებზე დაჭერისას ეკრანზე გამოჩნდება ციფრები და რიცხვები. როცა დავაჭერთ გარკვეულ მოქმედებას ეკრანზე მყოფი რიცხვი “გადავარდება”

მეხსიერებაში, შემდეგ დავაჭერთ მეორე ციფრს (ან ციფრთა კომბინაციას), რომელიც გამოჩნდება ეკრანზე. ანუ მოცემულ მომენტში გვაქვს 2 რიცხვი ერთი ეკრანზე და ერთი მეხსიერებაში. ასევე გვჭირდება მეხსიერებაში ინფორმაცია იმის შესახებ თუ რომელი მოქმედების ლილაკს დააჭირა მომხმარებელმა.

ყველა მოქმედებას შეასრულებს ლილაკი “=” რომელიც ამ ორ რიცხვს შორის შეასრულებს იმ მოქმედებას რომლის შესახებ ინფორმაცია მეხსიერებაშია შენახული და შედეგს გამოიტანს ეკრანზე.

პირველ რიგში საჭიროა რომ ეკრანზე გამოვიდეს ციფრები, რომლებსაც დავაჭერთ და მათგან შეიქმნას რიცხვი.

დავაკლიკოთ ორჯერ ლილაკზე წარწერით “1” და შევიყვანოთ შემდეგი კოდი:

```
display.Text = display.Text & Button1.text
```

& გამოიწვევს იმას, რომ ბოლოს შეყვანილი ციფრი დადგება მის წინ შეყვანილი ციფრის გვერდით და შექმნის რიცხვს.

ასევე გავხსნათ სხვა ლილაკების კოდი და ჩავწეროთ მათში იგივე კოდი, ოღონდ შევცვალოთ ლილაკის დასახელება (თუ გვაქვს ლილაკი წარწერით 7 ჩავწეროთ `display.Text = display.Text & Button7.text`).

გამოვაცხადოთ ცვლადები (ისინი ჩავწეროთ **Public Class Form1**-ის ქვემოთ):

```
Dim მეხსიერება As Decimal  
Dim მიმატება As Boolean  
Dim გამოკლება As Boolean  
Dim გაყოფა As Boolean  
Dim გამრავლება As Boolean
```

ახლა ჩავწეროთ კოდი მოქმედებების ღილაკებში:
დავაკლიკოთ ღილაკზე მიმატება და ჩავწეროთ შემდეგი კოდი:

```
მეხსიერება = display.Text  
display.Text = ""  
მიმატება = True  
გამოკლება = False  
გაყოფა = False  
გამრავლება = False
```

პირველი სტრიქონი მეხსიერებაში გადაიტანს დისპლეიზე არსებულ რიცხვს. მეორე სტრიქონი კი დისპლეის გაასუფთავებს.

შემდეგ უნდა დავიმახსოვროთ თუ რომელ მოქმედებას დააჭირა მომხმარებელმა. ამისათვის თუ ვწერთ კოდს ღილაკში მიმატება **Boolean** ტიპის ცვლადი **mimateba** უნდა გახდეს **True**, სხვა მოქმედებების ცვლადები კი **False**. მოცემულ მომენტში მნიშვნელობა **True** ექნება მხოლოდ ერთ ცვლადს (რომლის შესაბამის მოქმედებასაც დააჭირა მომხმარებელმა).

ასეთივე კოდი ჩავწეროთ სხვა მოქმედებების ღილაკებში, ოღონდ მნიშვნელობა **True** უნდა ჰქონდეს მხოლოდ მის შესაბამის ცვლადს, სხვა ცვლადები უნდა გახდეს **False**.

ახლა ჩავწეროთ კოდი ღილაკში “=”. მან უნდა შეასრულოს მოქმედებები, თანაც უნდა გამოიცნოს თუ რომელი მოქმედებაა შესასრულებელი. ამისათვის დაგვჭირდება პირობითი კონსტრუქციის გამოყენება.

დავაკლიკოთ ორჯერ ღილაკზე “=” და ჩავწეროთ შემდეგი კოდი:

```
If მიმატება = True Then display.Text = _  
display.Text + მეხსიერება  
If გამოკლება = True Then display.Text = _  
მეხსიერება - display.Text  
If გამრავლება = True Then display.Text = _  
display.Text * მეხსიერება  
If გაყოფა=True Then display.Text = მეხსიერება / _  
display.Text
```

ვფიქრობთ ამ კოდში თქვენ დამოუკიდებლად გაერკვევით.

ახლა ორჯერ დავაკლიკოთ ღილაკზე “C” და ჩავნეროთ შემდეგი კოდი:

```
Display.Text = ""
```

ცარიელი ბრჭყალები ნიშნავს, რომ ტექსტური ბლოკი გასუფთავდება.

მთლიანი კოდი გამოიყურება შემდეგნაირად:

```
Dim მეხსიერება As Decimal  
Dim მიმატება As Boolean  
Dim გამოკლება As Boolean  
Dim გამრავლება As Boolean  
Dim გაყოფა As Boolean
```

ღილაკი “1”

```
Private Sub Button1_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button1.Click  
    Display.Text = Display.Text & Button1.Text  
End Sub
```

ღილაკი “2”

```
Private Sub Button2_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button2.Click  
    Display.Text = Display.Text & Button2.Text  
End Sub
```

ღილაკი “3”

```
Private Sub Button3_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button3.Click  
    Display.Text = Display.Text & Button3.Text  
End Sub
```

ლილაკი “4”

```
Private Sub Button4_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button4.Click  
    Display.Text = Display.Text & Button4.Text  
End Sub
```

ლილაკი “5”

```
Private Sub Button5_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button5.Click  
    Display.Text = Display.Text & Button5.Text  
End Sub
```

ლილაკი “6”

```
Private Sub Button6_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button6.Click  
    Display.Text = Display.Text & Button6.Text  
End Sub
```

ლილაკი “7”

```
Private Sub Button7_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button7.Click  
    Display.Text = Display.Text & Button7.Text  
End Sub
```

ლილაკი “8”

```
Private Sub Button8_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button8.Click  
    Display.Text = Display.Text & Button8.Text  
End Sub
```

ლილაკი “9”

```
Private Sub Button9_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button9.Click  
    Display.Text = Display.Text & Button9.Text  
End Sub
```

ლილაკი “0”

```
Private Sub Button10_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button10.Click  
    Display.Text = Display.Text & Button10.Text  
End Sub
```

ლილაკი “”

```
Private Sub Button19_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button19.Click  
    Display.Text = Display.Text & Button19.Text  
End Sub
```

ლილაკი “C”

```
Private Sub Button15_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button15.Click  
    Display.Text = ""  
End Sub
```

ლილაკი “+”

```

Private Sub Button14_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button14.Click
    მებსიერება = Display.Text
    Display.Text = ""

    მიმატება = True
    გამოკლება = False
    გაყოფა = False
    გამრავლება = False

End Sub

```

ლილაკი “-”

```

Private Sub Button13_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button13.Click
    მებსიერება = Display.Text
    Display.Text = ""

    მიმატება = False
    გამოკლება = True
    გაყოფა = False
    გამრავლება = False

End Sub

```

ლილაკი “:”

```

Private Sub Button11_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button11.Click
    მებსიერება = Display.Text
    Display.Text = ""

    მიმატება = False
    გამოკლება = False

```

```
გაყოფა = True
გამრავლება = False
```

```
End Sub
```

ლილაკი "*"

```
Private Sub Button12_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button12.Click
    მესიერება = Display.Text
    Display.Text = ""

    მიმატება = False
    გამოკლება = False
    გაყოფა = False
    გამრავლება = True

End Sub
```

ლილაკი "="

```
Private Sub Button18_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button18.Click
    On Error Resume Next

    If მიმატება = True Then
        Display.Text = მესიერება +
Decimal.Parse(Display.Text)
    End If

    If გამოკლება = True Then
        Display.Text = მესიერება -
Decimal.Parse(Display.Text)
    End If

    If გაყოფა = True Then
```



```

        Display.Text = მესიერება /
Decimal.Parse(Display.Text)
    End If

    If გამრავლება = True Then
        Display.Text = მესიერება *
Decimal.Parse(Display.Text)
    End If

End Sub

End Class

```

გავუშვათ პროგრამა. ჩვენი კალკულატორი უკვე მზადაა გამოყენებისათვის.

რა დაგვრჩა?

ყველაზე მარტივ კალკულატორსაც კი უნდა ჰქონდეს უარყოფითი რიცხვების შეყვანის და პროცენტის ფუნქცია. ვფიქრობთ რომ ამ ამოცანას თქვენით დამოუკიდებლად გადანყვეთ. დამოუკიდებელი გადანყვეტილებების (თუნდაც მარტივი) გარეშე დაპროგრამების შესწავლა ვფიქრობთ შეუძლებელია. აქამდე თქვენს მიერ მიღებული ცოდნა ამ ლილაკების დამატებისათვის სავსებით საკმარისია. ასევე შეგიძლიათ ჩაატაროთ ექსპერიმენტები მის დიზაინზე.

ციკლები

მოდით ცოტახნით დავეუბრუნდეთ თეორიას და შევისწავლოთ დაპროგრამების ერთ-ერთი უმნიშვნელოვანესი ელემენტი—ციკლები.

დაპროგრამებაში განმეორებადი მოქმედებების შესასრულებლად გამოიყენება ციკლი. Visual Basic-ში ის შეიძლება იყოს შემდეგი ტიპის:

- *For . . . Next.*
- *For Each . . . Next.*
- *Do . . . Loop.*

განვიხილოთ ისინი სათითაოდ:

ციკლი For . . . Next

კონსტრუქცია For . . . Next ასრულებს ბრძანებებს განსაზღვრულ რიცხვჯერ, ასეთ კონსტრუქციას უწოდებენ ციკლს, ხოლო მის მიერ შესრულებულ პროგრამულ კოდს—ციკლის სხეულს. For . . . Next ციკლში განმეორებათა რაოდენობა წინასწარ არის ცნობილი. ასეთ ციკლს “არითმეტიკულ ციკლს” უწოდებენ. ციკლს რომელშიც განმეორებათა რაოდენობა წინასწარ ცნობილი არ არის “იტერაციული” ეწოდება (იტერაციული ციკლი მოგვიანებით იქნება განხილული).

For . . . Next კონსტრუქციის სინტაქსი შემდეგია:

For მთვლელო *As* მონაცემთა ტიპი=საწყის მნიშვნელობა *To*
საბოლოო მნიშვნელობა *Step* ბიჯი

ოპერატორები

Next მთვლელო

მაგალითად: შემდეგი ციკლი მოახდენს რიცხვების
აჯამებას 0 დან 10 მდე

For x *As* Integer = 0 *To* 10
 $x=x+1$
Next x

კონსტრუქციის პირველი არგუმენტი—*მთვლელო* განსაზღვრავს ცვლადის სახელს, რომელიც “დაითვლის” ციკლის გამეორებათა რაოდენობას. ეს ცვლადი შეიძლება გამოცხადდეს პირდაპირ კონსტრუქციაში.

პარამეტრი *საწყისი მნიშვნელობა* მიუთითებს რიცხვით მნიშვნელობას, რომელიც მიენიჭება ცვლად-მრიცხველს თავდაპირველად (ციკლის დაწყებამდე). ციკლი სრულდება მანამ, სანამ მთვლელის მნიშვნელობა არ აღემატება საბოლოო მნიშვნელობას, რომელიც მითითებულია საკვანძო სიტყვა *To*-ს შემდეგ.

მთვლელის მნიშვნელობა იცვლება *ბიჯი* სიდიდით, რომელიც მითითებულია საკვანძო სიტყვა *Step*-ის შემდეგ.

საკვანძო სიტყვა *Next* აღნიშნაქვს ციკლის სხეულის დასასრულს და წარმოადგენს აუცილებელს.

ციკლის ყოველი გამეორების შემდეგ *Visual Basic* ადარებს *მთვლელის* და არგუმენტ საბოლოო მნიშვნელობის მნიშვნელობებს.

ცვლადი *მთვლელო* უნდა იყოს რიცხვითი ტიპის და შეასრულოს ოპერაცია მეტობა (>), ნაკლებობა (<), და ჯამი(+).

რეკომენდირებულია ცვლადი *მთვლელის* მითითება საკვანძო სიტყვა *Next*-ის შემდეგ. განსაკუთრებით მაშინ, როდესაც რამოდენიმე ციკლი განთავსებულია ერთმანეთში. მაგალითად:

```
Dim n (10 , 10 ) As Integer
For I As Integer = 1 To 10
    For j As Integer = 1 To 1
        n ( I , j ) = I + j
    Next j
Next i
```

მთვლელის ცვლილების ბიჯი შეიძლება იყოს უარყოფითიც. მაგალითად:

```
For nCounter = 100 To 1 Step -10
    nDecades (nCounter) = nCounter * 2
Next
```

ამ შემთხვევაში ციკლი შესრულდება მანამ, სანამ *nCounter* მეტია 1-ზე.

საკვანძო სიტყვა *Step* შეიძლება არ მივუთითოთ. ამ შემთხვევაში ბიჯის მნიშვნელობა გაუტოლდება 1-ს.

შეიძლება შეგვხვდეს ისეთი სიტუაციები, როდესაც ციკლის შესრულება შეუძლებელია ან პირიქით, მისი შესრულება უსასრულო ხდება. მაგალითად:

შეუსრულებადი ციკლი: დადებითი ბიჯის არსებობისას, მრიცხველის სანყისი მნიშვნელობა მეტია საბოლოოზე.

```
For nCounter = 100 To 1
    nDecades (nCounter) = nCounter
Next
```

უსასრულო ციკლი: მთვლელის მნიშვნელობა არასდროს არ აღემატება 10-ს

```
For nCounter = 1 To 10
    nCounter = 1
Next
```

ციკლი Do . . . Loop

ციკლი, რომელიც მოცემულია კონსტრუქციით Do . . . Loop, სრულდება მანამ, სანამ მასში მოცემული პირობა ჭეშმარიტია. ციკლში განმეორებათა რაოდენობა წინასწარ არ არის ცნობილი. ასეთ ციკლს “იტერაციული” ეწოდება.

Do . . . Loop კონსტრუქციას აქვს შემდეგი სახე:

```
Do While პირობა
    ოპერატორები
Loop
```

კონსტრუქციის არგუმენტი *პირობა* წარმოადგენს ლოგიკურ გამოსახულებას, რომლის მნიშვნელობა მოწმდება ციკლის ყოველი განმეორების შემდეგ. თუ ეს მნიშვნელობა უდრის True-ს, მაშინ სრულდება ბრძანებათა მიმდევრობა, რომელიც განთავსებულია Do While და საკვანძო სიტყვა Loop-ს შორის. ეს კონსტრუქციები ქმნიან *ციკლის სხეულს*.

თუ ციკლის მორიგი განმეორების შემდეგ *პირობა* გახდება False, მაშინ ხდება ციკლიდან გამოსვლა და მართვა გადაეცემა Loop-ის შემდეგ მდგომ კონსტრუქციას.

ზოგჯერ გვხვდება ისეთი სიტუაცია, როდესაც ციკლის ოპერატორი ერთხელაც არ შესრულდება. ეს მოდება იმ შემთხვევაში, როცა პირველივე შემოწმებისას პირობა მცდარი იქნება.

Visual Basic-ში არსებობს Do . . . Loop კონსტრუქციის ციკლის კიდევ ერთი სახეობა. თუ წინა კონსტრუქციაში, ციკლიდან გამოსვლის პირობა განთავსებულია სათაურში, აქ პირობა განთავსებულია ციკლის ბოლოს:

Do
ოპერატორები
Loop While პირობა

ამ ფორმის ოპერატორის გამოყენებისას, ციკლის სხეული ერთხელ მაინც სრულდება, რის შემდეგ ხდება მოცემული პირობის შემოწმება.

Visual Basic-ში არსებობს Do . . . Loop კონსტრუქციის კიდევ ერთი ციკლი. ის განსხვავებულია *Do While...Loop* ციკლისაგან, ციკლიდან გამოსვლის პირობით. განსხვავებულია იმაში, რომ ციკლი სრულდება მანამ, სანამ ციკლიდან გამოსვლის პირობა არა ჭეშმარიტი, არამედ მცდარია. მას გააჩნია შემდეგი სინტაქსი:

Do Until პირობა
ოპერატორები
Loop

და

Do
ოპერატორები
Loop Until პირობა

მაგალითი:

Dim nDecades (10) As Integer , nCounter As Integer = 2

Do While nCounter < 10
*nDecades (nCounter) = (nCounter) * 2*

Loop

იგივე ციკლი შეიძლება ჩაინეროს შემდეგნაირად:

Dim nDecades (10) As Integer , nCounter As Integer = 2

Do Until nCounter > 10

*nDecades (nCounter) = (nCounter) * 2*

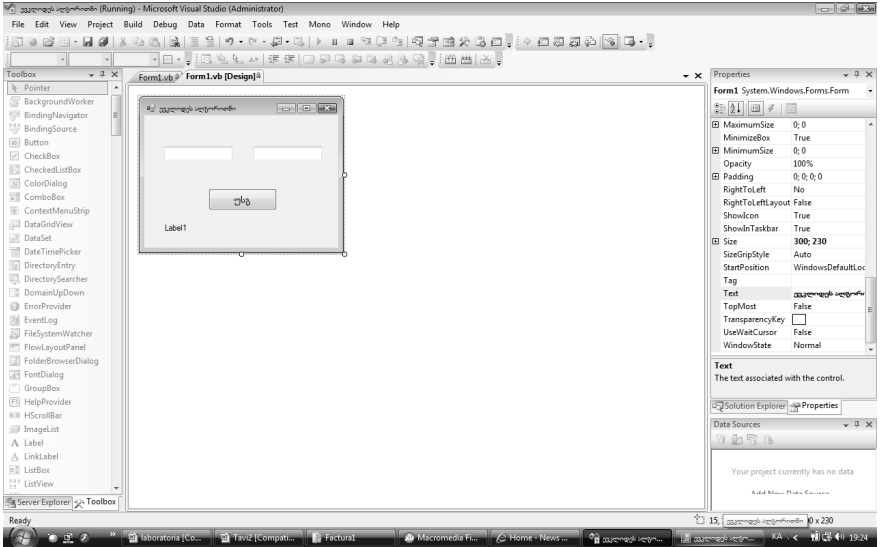
Loop

პროექტი “ევკლიდეს ალგორითმი” (ციკლის გამოყენებით)

ჩვენ უკვე შევქმენით პროექტი “ევკლიდეს ალგორითმი”. ახლა გადავაკეთოთ ჩვენი პროექტი და გამოვიყენოთ ციკლი. ჩვენი პროექტის შესრულებისას ფაქტიურად საქმე გვაქვს ციკლთან. რივხვები ერთმანეთს აკლდება მანამ სანამ ისინი ერთმანეთს არ გაუტოლდება და ეს მოქმედებები მეორდება რამოდენიმეჯერ. მაგრამ ამისათვის ჩვენ გვინევს ღილაკზე დაჭერა რამოდენიმეჯერ.

რაც შეეხება ამ ამოცანის ორი რიცხვის უდიდესი საერთო გამყოფის მოსაძებნად უდიდესს უნდა გამოვაკლოთ უმცირესი და უდიდესს უნდა მივანიჭოთ ნაშთის მნიშვნელობა. შემდეგ მიღებულ რიცხვებზე კვლავ გავიმეოროთ იგივე მოქმედება მანამ, სანამ ორივე რიცხვი ერთმანეთის ტოლი არ გახდება. სწორედ ეს რიცხვი იქნება უდიდესი საერთო გამყოფი (უსგ).

თუ გამოვიყენებთ პროგრამულ კოდში ციკლს, უდიდეს საერთო გამყოფს დავიანგარიშებთ ღილაკზე ერთხელ დაჭერით (პასუხი გამოვა ისე რომ ვერ ვნახავთ ბიჯებს. მოქმედებები რიცხვებზე შესრულდება მანამ, სანამ ისინი ერთმანეთს არ გაუტოლდება და ეს ისე სწრაფად მოხდება, რომ ჩვენ ბიჯებს ვერ დავინახავთ.



დავაჭიროთ ორჯერ ღილაკს და კოდი რომელიც ჩანერილია მოვაქვეით ციკლში **Do While** (ან ციკლში **Do Until** თუ მოვაქცევთ ციკლში **Do Until** ნიშანი "<>" (არ უდრის) უნდა შევცვალოთ ნიშნით "="):

```
Public Class Form1
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
```

```
        Do While TextBox1.Text <> TextBox2.Text
            If Val(TextBox1.Text) > Val(TextBox2.Text)
                Then
                    TextBox1.Text = Val(TextBox1.Text) - _
                    Val(TextBox2.Text)
                End If
```

```
            If Val(TextBox1.Text) < Val(TextBox2.Text)
                Then
```

```

        TextBox2.Text = Val(TextBox2.Text) - _
Val(TextBox1.Text)
    End If

    If TextBox1.Text = TextBox2.Text Then
        Label1.Text = "ამ ორი რიცხვის უდიდესი _
საერთო გამყოფია " & TextBox1.Text
    End If
Loop
End Sub
End Class

```



```

Public Class Form1

    Private Sub Button1_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button1.Click

```

```

        If Val(TextBox1.Text) > Val(TextBox2.Text) Then
            TextBox1.Text = Val(TextBox1.Text) - _
Val(TextBox2.Text)
        End If

        If Val(TextBox1.Text) < Val(TextBox2.Text) Then
            TextBox2.Text = Val(TextBox2.Text) - _
Val(TextBox1.Text)
        End If

        If TextBox1.Text = TextBox2.Text Then
            Label1.Text = "ამ ორი რიცხვის უდიდესი _
საერთო გამყოფია " & TextBox1.Text
        End If
    End Sub
End Class

```

გავუშვათ პროგრამა. ჩავწეროთ რიცხვები ტექსტურ ბლოკებში და დავაჭიროთ ლილაკს “უსგ” პროგრამა პირდაპირ გამოიტანს ორი რიცხვის უდიდეს საერთო გამყოფს☺.

ოპერატორი Exit

რიგ შემთხვევებში, აუცილებელია ციკლის დასრულებამდე მისი შეწყვეტა. ეს შეიძლება შესრულდეს Exit ბრძანების საშუალებით.

Exit ბრძანება ასრულებს ციკლის მუშაობას და მართვას გადასცემს ციკლის შემდგომ კონსტრუქციას. ამ ოპერატორს For ციკლში აქვს სახე - Exit For, ხოლო Do ციკლში - Exit Do.

```
For მთვლელი As მონაცემთატიპი = საწყისი მნიშვნელობა To  
საბოლოო მნიშვნელობა Step ბიჯი  
ოპერატორები  
Exit For  
ოპერატორები  
Next მთვლელი
```

```
Do {While/Until} პირობა  
ოპერატორები  
Exit Do  
ოპერატორები  
Loop
```

მაგალითად:

```
For nCounter As Integer = 100 To 1 Step -10  
nDecades (nCounter) = (nCounter) * 2  
If nDecades (nCounter) > 20 Then Exit For  
Next
```

ოპერატორი **Exit** შეიძლება ციკლის შიგნით შეგვხვდეს რამოდენიმეჯერ, ნებისმიერი საჭირო რაოდენობით, მაგალითად:

```
Do Until Y = -1
  If x < 0 Then Exit Do
  x = Sqrt(x)
  If Y < 0 Then Exit Do
  Y = Y + 3
  If z < 0 Then Exit Do
  z = x / z
Loop
```

Exit ოპერატორი ასევე შეიძლება გამოვიყენოთ **Sub** და **Function** პროცედურებიდან გამოსასვლელად. ამ შემთხვევაში მას აქვს შემდეგი სახე **Exit Sub** და **Exit Function**.

ოპერატორი Continue

ოპერატორი **Continue** იძლევა ციკლის შემდეგ იტერაციაზე სწრაფად გადასვლის საშუალებას.

ოპერატორი **Continue**-ს დახმარებით შეიძლება შესრულდეს გადასვლა ერთი იტერაციიდან მეორეზე. განვიხილოთ მაგალითი:

```
Dim I As Integer
For I = 1 To 4
    If i = 2 Then Continue For
    Console.WriteLine(i)
Next
```

პროგრამის შესრულების შემდეგ კონსოლზე გამოისახება რიცხვები 1, 3, 4. ანუ გამოტოვებს 2-ს.

კონსტრუქცია With ...End With

თუ არსებობს ოპერატორების მიმდევრობა, რომლებიც მუშაობენ ერთი და იმავე მართვის ობიექტთან, მაშინ შეიძლება გამოვიყენოთ ოპერატორი **With . . . End With**, რომელიც ახორციელებს ყველა ოპერატორისათვის ობიექტის ერთჯერად მითითებას. ამ ოპერაციის დახმარებით ჩქარდება პროცედურის შესრულება და საჭირო არ ხდება ზედმეტი ტექსტის შეყვანა.

კონსტრუქციას **With . . . End With** აქვს შემდეგი სინტაქსი:

With ობიექტი
ოპერატორები
End With

კონსტრუქცია *With . . . End With* საშუალებას იძლევა საგრძნობლად გაამარტივდეს მრავალჯერადი მიმართვა ობიექტის თვისებებზე და მეთოდებზე. მაგალითად, შემდეგი კოდის დახმარებით შეიძლება შევცვალოთ ობიექტ *Label*-ის (სახელად *Label1*) თვისებები.

With Label1
 . Text = “დილა მშვიდობისა!”
 . ForColor = System.Drawing.Color.Green
 . Font = New Font (. Font , FontStyle.Bold)
End With

ობიექტის თვისებების წინ დაიწერება წერტილის ნიშანი.



Visual Basic საშუალებას გვაძლევს ჩვენს პროექტებში გამოყენებული იყოს მაუსი და კლავიატურა არა მხოლოდ ტექსტური ინფორმაციის შესაყვანად, არამედ ნებისმიერი ბრძანების შესასრულებლად (ისევე როგორც ტექსტურ რედაქტორებში, თამაშებში), ანუ მაგ: შესრულდეს კონკრეტული ბრძანება კლავიატურის კონკრეტულ ღილაკზე დაჭერით ან მაუსის კონკრეტულ ღილაკზე დაკლიკებით.

ამის განსახორციელებლად დაგვჭირდება გავერკვეთ მართვის ელემენტების “ფოკუსის” ცნებასთან.

რა არის ფოკუსი? ამის გასარკვევად პროექტის ფორმაზე გადმოვიტანოთ 2 ელემენტი **TextBox** და 2 ელემენტი **Button**. გავუშვათ პროექტი. დავაჭიროთ ერთ ღილაკს (**Button1**) შემდეგ მეორეს. ვნახავთ ვიზუალურ განსხვავებას პირველ ღილაკს და ბოლოს დაჭერილ ღილაკს შორის. სემდეგ მოვინსნოთ ერთი ტექსტური ბლოკი, შემდეგ მეორე, ასევე ვნახავთ რომ ბოლოს მონიშნულ ტექსტურ ბლოკში ციმციმებს კურსორი.

პროგრამა გვიჩვენებს რომელ ელემენტს დავაჭირეთ ბოლოს (რომელზე მოვახდინეთ ფოკუსირება). ამბობენ რომ ობიექტს გააჩნია “ფოკუსი”.

ობიექტებს, რომლებსაც უნარი აქვთ გააჩნდეთ ფოკუსი აქვთ 2 მოვლენა **Enter**, რომელიც ხდება მაშინ, როცა ის მიიღებს ფოკუსს და **Leave** როცა ის დაკარგავს ფოკუსს.

ობიექტისათვის ფოკუსის მინიჭება შეიძლება შემდეგი კოდის გამოყენებით:

```
TextBox1.Focus  
TextBox2.Focus
```

ჩავნეროთ ეს კოდი რომელიმე ლილაკის კოდში.

თუ გავუშვებთ ამ პროექტს და დავაჭერთ ლილაკს, ვნახავთ თუ როგორ გადავა ფოკუსი სასურველ ელემენტებზე. ფოკუსი საჭიროა იმისათვის, რომ კომპიუტერმა იცოდეს რომელმა ელემენტმა უნდა მოახდინოს რეაგირება კლავიატურაზე. თუ თქვენ კლავიატურაზე აკრეფთ ტექსტს, ის შევა მხოლოდ იმ ტექსტურ ბლოკში, რომელსაც აქვს ფოკუსი. თუ ფოკუსი არის ლილაკზე, თქვენ შეგიძლიათ დააჭიროთ მას არა მარტო მაუსით, არამედ კლავიატურიდანაც “Enter” ლილაკის საშუალებით.

ფოკუსის გადატანა ერთი ელემენტიდან, მეორეზე თქვენ შეგიძლიათ ლილაკით **Tab**.

ფოკუსის გადასვლის თანმიმდევრობას განსაზღვრავს მართვის ელემენტების თვისება **TabIndex**. თქვენ მისი ნახვა შეგიძლიათ ელემენტთა თვისებათა ფანჯარაში. **TabIndex**-ის მნიშვნელობა 0-ს მიიღებს პირველი ობიექტი, რომელიც გაჩნდა ფორმაზე, მეორე 1-ს და ასშ. ამიტომ პროექტის გაშვებისას ყოვლთვის მონიშნულია პირველი ობიექტი რომელიც მოთავსდა ფორმაზე.

თქვენ შეგიძლიათ თვისებათა ფანჯარაში შეცვალოთ ელემენტთა თვისება **TabIndex**.

თქვენ ასევე შეგიძლიათ აკრძალოთ ფოკუსირება, რომელიმე მართვის ობიექტზე, მის თვისებას **TabStop** მიანიჭეთ მნიშვნელობა **False**.

გავცნოთ კლავიატურასთან დაკავშირებულ მოვლენებს. ასეთი სამია: **KeyDown** (კლავიში დაჭერილია), **KeyUp** (კლავიში აშვებულია), **KeyPress** (კლავიატურიდან შეიყვანეს სიმბოლო).

კლავიატურის ყოველ კლავიშს გააჩნია თავისი კოდი, რომლითაც შეგიძლია მივმართოთ. ან შეიძლება პირდაპირ მივუთითოთ მოცემულ კლავიშზე. ამისათვის გამოვიყენებთ ერთ-

ერთ ზემოთ ჩამოთვლილ მოვლენას და კონსტრუქციას e.KeyCode.

როგორ გავიგოთ კლავიატურის კლავიშების კოდი? მაგ: რა კოდი აქვს კლავიშს A.

ამის ათვის არსებობს სპეციალური ცხრილი, რომელიც ჩვენ გვაქვს მოყვანილი. მეორე მეთოდია კოდის გარკვევა სპეციალური პროგრამის დაწერით. დაწერეთ პროგრამას და გამოვიყენებთ მას როცა დაგვჭირდება.

პროექტი “კლავიშების კოდები”

შევქმნათ ახალი პროექტი, დავარქვათ მას სახელი, მივუთითოთ შენახვის მისამართი.

შეიქმნება ცარიელი ფორმა. ამჯერად არ გამოვიყენებთ მართვის ელემენტებს. დავაკლიკოთ ორჯერ ფორმაზე, გაიხსნება პროექტის კოდი და მოვლენა Form1_Load.

ჩვენ დაგვჭირდება მოვლენა Form1_KeyDown, რომელიც უნდა ვიპოვოთ მოვლენათა ჩამონათვალში ეკრანის მარჯვენა მხარეს.

ჩავწეროთ ქვემოთ მოყვანილი კოდი:

```
Private Sub Form1_KeyDown(ByVal sender As Object, ByVal _  
e As System.Windows.Forms.KeyEventArgs) Handles _  
Me.KeyDown
```

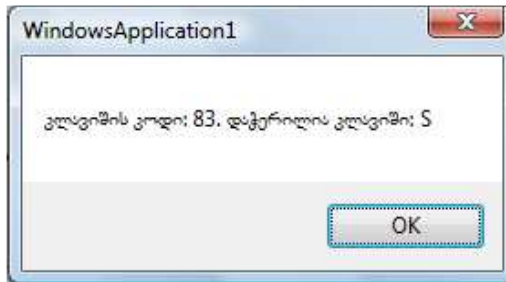
```
    MsgBox("კლავიშის კოდი: " & e.KeyCode & ". _  
დაჭერილია კლავიში: " & Chr(e.KeyCode))
```

```
End Sub
```

ჩვენს კოდში Chr(e.KeyCode) - განსაზღვრავს კლავიშის დასახელებას კლავიატურაზე.

e.KeyCode - კლავიშის კოდს.

გავუშვათ პროგრამა. კლავიატურის კლავიშზე დაჭერისას მივიღებთ შეტყობინებას MsgBox-ით. სადაც იქნება კლავიშის დასახელება და მისი კოდი. მაგ S კლავიშზე დაჭერისას მივიღებთ შემდეგ შეტყობინებას:



ახლა მოდით ჩვენს პროექტში გადმოვიტანოთ ელემენტი **Button**. და კვლავ გავუშვათ პროგრამა. ახლა კლავიშზე დაჭერისას ჩვენი პროექტი არ იმუშავებს. საქმე იმაშია რომ მოვლენა `Form1_KeyDown` სრულდება მაშინ თუ ფოკუსი არის ფორმაზე. ჩვენს შემთხვევაში კი ფოკუსი იქნება ღილაკზე.

ამ შემთხვევაში საჭიროა ფოკუსი გადმოვიტანოთ ფორმაზე ან გამოვიყენოთ მოვლენა `Button1_KeyDown`.

არსებობს სხვა გადაწყვეტილებებიც. ექსპერიმენტები მკითხველისთვის მიგვინდია. ფოკუსი საჭიროა იმისათვის, რომ კომპიუტერმა იცოდეს, რომელმა ელემენტმა უნდა მოახდინოს რეაგირება კლავიატურაზე.

პროექტი “ბრძანებები კლავიშებიდან”

შევქმნათ პროექტი რომელიც მოგვცემს საშუალებას კლავიშებიდან შევასრულოთ ბრძანებები—ვცვალოთ ფორმის ფერები.

მაგ: **R** კლავიშზე დაჭერისას ფორმის ფონი გახდეს წითელი, **Y**-ზე დაჭერისას—ყვითელი, **G**-ზე—მწვანე და **B**-ზე—ცისფერი.

შევქმნათ ახალი პროექტი, დავარქვათ მას სახელი, მივუთითოთ შენახვის მისამართი.

შეიქმნება ცარიელი ფორმა. ამჯერად არ გამოვიყენებთ მართვის ელემენტებს. დავაკლიკოთ ორჯერ ფორმაზე, გაიხსნება პროექტის კოდი და მოვლენა Form1_Load.

ჩვენ დაგვჭირდება მოვლენა Form1_KeyDown, რომელიც უნდა ვიპოვოთ მოვლენათა ჩამონათვალში ეკრანის მარჯვენა მხარეს.

ჩავწეროთ ქვემოთ მოყვანილი კოდი:

```
Private Sub Form1_KeyDown(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyEventArgs) Handles Me.KeyDown
    If e.KeyCode = Keys.R Then BackColor = _
Color.Red
    If e.KeyCode = Keys.B Then BackColor = _
Color.Blue
    If e.KeyCode = Keys.Y Then BackColor = _
Color.Yellow
    If e.KeyCode = Keys.G Then BackColor = _
Color.Green
End Sub
```

Keys-ის შემდეგ ნერტილის დასმის შემდეგ ჩამოიშლება სია სადაც შეგვიძლია მივუთითოთ კლავიშის დასახელება. ანუ არ გვინევს კლავიშის კოდის მითითება. თუმცა შეგვიძლია Keys-ის ნაცვლად მივუთითოთ კლავიშის კოდი. მაგ: ქვემოთ მოყვანილი კოდი შეასრულებს იგივე ფუნქციას რასაც ზემოთ მოყვანილი (კოდები შეიძლება ვიპოვოთ ცხრილში ან ჩვენი წინა პროექტის გამოყენებით):

```
Private Sub Form1_KeyDown(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyEventArgs) Handles _
Me.KeyDown
    If e.KeyCode = 82 Then BackColor = Color.Red
    If e.KeyCode = 66 Then BackColor = Color.Blue
    If e.KeyCode = 89 Then BackColor = Color.Yellow
    If e.KeyCode = 71 Then BackColor = Color.Green
End Sub
```

გავუშვათ პროექტი. დავაჭიროთ კლავიატურის ღილაკებს R,Y,G,B. ფორმა შესაბამისად შეიცვლის ფერებს. ანუ ჩვენ შეგვიძლია ჩვენს პროექტში ჩავრთოთ კლავიატურის ნებისმიერი კლავიში და შევასრულებინოთ მას გარკვეული ბრძანება. თქვენ იცით რომ ეს ფართოდ გამოიყენება სხვადასხვა პროგრამებში (გრაფიკულ პროგრამებში, თამაშებში და სხვ).

კოდი	კლავიში	კოდი	კლავიში
		048	0
000	სპეც. NOP	049	1
001	სპეც. SOH	050	2
002	სპეც. STX	051	3
003	სპეც. ETX	052	4
004	სპეც. EOT	053	5
005	სპეც. ENQ	054	6
006	სპეც. ACK	055	7
007	სპეც. BEL	056	8

008	სპეც. BS	057	9
009	სპეც. Tab	058	:
010	სპეც. LF	059	;
011	სპეც. VT	060	<
012	სპეც. FF	061	=
013	სპეც. CR	062	>
014	სპეც. SO	063	?
015	სპეც. SI	064	@
016	სპეც. DLE	065	A
017	სპეც. DC1	066	B
018	სპეც. DC2	067	C
019	სპეც. DC3	068	D
020	სპეც. DC4	069	E
021	სპეც. NAK	070	F
022	სპეც. SYN	071	G
023	სპეც. ETB	072	H
024	სპეც. CAN	073	I
025	სპეც. EM	074	J
026	სპეც. SUB	075	K
027	სპეც. ESC	076	L
028	სპეც. FS	077	M
029	სპეც. GS	078	N
030	სპეც. RS	079	O
031	სპეც. US	080	P
032	სპეც. SP (პრობელი)	081	Q
033	!	082	R
034	"	083	S
035	#	084	T
036	\$	085	U
037	%	086	V
038	&	087	W
039	'	088	X
040	(089	Y
041)	090	Z

042	*	091	[
043	+	092	\
044	,	093]
045	-	094	^
046	.	095	_
047	/	096	`
097	a	112	p
098	b	113	q
099	c	114	r
100	d	115	s
101	e	116	t
102	f	117	u
103	g	118	v
104	h	119	w
105	i	120	x
106	j	121	y
107	k	122	z
108	l	123	{
109	m	124	
110	n	125	}
111	o	126	~

პროექტი “მაუსის ღილაკები”

შევქმნათ ახალი პროექტი, დავარქვათ მას სახელი, მივუთითოთ შენახვის მისამართი. შეიქმნება ცარიელი ფორმა. ელემენტთა პანელიდან გადმოვიტანოთ მასზე მართვის

ელემენტი **TextBox**. გავხსნათ კოდი და ვნახოთ მოვლენათა ჩამონათვალში მაუსთან დაკავშირებული მოვლენები.

Click (კლიკი), **DoubleClick** (ორმაგი კლიკი), **MouseDown** (დაჭერილია მაუსის კლავიში), **MouseUp** (აშვებულია მაუსის კლავიში), **MouseEnter** (მაუსი გაჩნდა ფორმაზე), **MouseLeave** (მაუსმა დატოვა ფორმა), **MouseMove** (მაუსი გადაადგილეს).

ასევე შესაძლებელია კონკრეტული ოპერატორების (ბრძანებების) დაკავშირება მაუსის ლილაკებზე დაჭერის მოვლენასთან (მაუსის მარცხენა, მარჯვენა და შუა ლილაკი).

ამისათვის უნდა გამოვიყენოთ პირობითი ოპერატორი და `e.Button` კონსტრუქცია:

```
e.Button = MouseButton.Left  
e.Button = MouseButton.Right  
e.Button = MouseButton.Middle
```

ჩავნეროთ შემდეგი პროგრამული კოდი:

```
Private Sub Form1_Click(ByVal sender As Object, ByVal e_ _  
As System.EventArgs) Handles MyBase.Click  
    TextBox1.Text = ("ამოქმედდა მოვლენა Click")  
End Sub  
  
Private Sub Form1_DoubleClick(ByVal sender As _  
Object, ByVal e As EventArgs) Handles _  
MyBase.DoubleClick  
    TextBox1.Text = ("ამოქმედდა მოვლენა  
DoubleClick")  
End Sub  
  
Private Sub Form1_MouseDown(ByVal sender As Object, _  
ByVal e As MouseEventArgs) _  
Handles MyBase.MouseDown  
    If e.Button = MouseButton.Left Then  
        TextBox1.Text = ("დაჭერილია მაუსის მარცხენა  
ლილაკი")  
    End If
```



```

        If e.Button = MouseButton.Right Then
            TextBox1.Text = ("დაჭერილია მაუსის მარჯვენა _
ლილაკი")
        End If
        If e.Button = MouseButton.Middle Then
            TextBox1.Text = ("დაჭერილია მაუსის შუა _
ლილაკი")
        End If
    End Sub

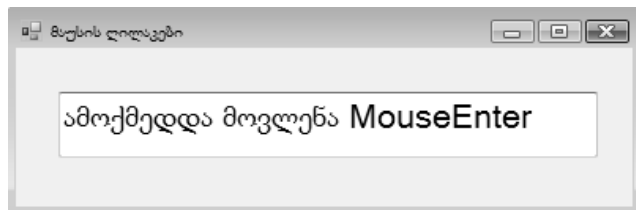
    Private Sub Form1_MouseUp(ByVal sender As Object, _
ByVal e As MouseEventArgs) Handles MyBase.MouseUp
        TextBox1.Text = ("ამოქმედდა მოვლენა MouseUp")
    End Sub

    Private Sub Form1_MouseEnter(ByVal sender As _
Object, ByVal e As EventArgs) Handles MyBase.MouseEnter
        TextBox1.Text = ("ამოქმედდა მოვლენა _
MouseEnter")
    End Sub

    Private Sub Form1_MouseLeave(ByVal sender As _
Object, ByVal e As EventArgs) Handles MyBase.MouseLeave
        TextBox1.Text = ("ამოქმედდა მოვლენა MouseLeave")
    End Sub

```

გავუშვათ პროგრამა. დავაკირდეთ მაუსის სხვადასხვა მოქმედებებზე თუ რა ნარწერა გამოვა ტექსტურ ბლოკში.



შემთხვევითი რიცხვები



შემთხვევითი რიცხვები ფართოდ გვხვდება ჩვენს ყოველდღიურ ცხოვრებაში. მათ დიდი გამოყენება აქვთ კომპიუტერულ თამაშებში, სხვადასხვა გათამაშებებში (ლატარეა) და ასევე მეცნიერების სხვადასხვა დარგში. შემთხვევითი რიცხვები ეს არის რიცხვების თანმიმდევრობა, რომლებსაც ერთმანეთთან არანაირი კავშირი არა აქვთ. ისინი ამოირჩევიან შემთხვევითი წესით და არა რაიმე კანონზომიერებით. მათი წინასწარმეტყველება ფაქტიურად შეუძლებელია. შემთხვევითი რიცხვების მარტივი მაგალითია კამათელის გაგორება ან ჟეტონის აგდება. პირველი მოქვცემს 6 შემთხვევითი რიცხვიდან ერთს. მეორე ორიდან ერთს.

თითქმის ყველა კომპიუტერული თამაშისათვის აუცილებელია შემთხვევითი რიცხვების გამოყენება. თუ ასეთი რამ არ მოხდება, თამაში ყოვლთვის იქნება ერთნაირი თანმიმდევრობით, და მალე ბოსაბეზრებელი გახდება.

ხშირად გამოიყენებენ შემთხვევით რიცხვებს სამეცნიერო კვლევებისათვის. მაგალითად მედიცინაში. მაგ: როცა საჭიროა წამლის ეფექტის დადგენა, კვლევა რომ მეტად სარწმუნო იყოს საჭიროა რომ პაციენტები საკვლევ ჯგუფებში გადანაწილდნენ შემთხვევითი წესით—რანდომიზირებულად, ამისათვის მათ დანომრავენ და გამოიყენებენ რომელიმე შემთხვევითი რიცხვების გენერატორს (მაგ: დახურული კონვერტები,

რომლებსაც აიღებენ პაციენტები ან კომპიუტერული პროგრამა, რომელიც მათ ჯგუფებში გადაანაწილებს).

Visual Basic-ს აქვს ფუნქცია, რომელიც მოქცევს შემთხვევით რიცხვებს ჩვენთვის საჭირო დიაპაზონში. ამისათვის გამოიყენება ოპერატორი Rnd. Rnd გვაძლევს შემთხვევით რიცხვებს. თუ გნივდა განვსაზღვროთ მათი დიაპაზონი მაშინ უნდა გავამრავლოთ დიაპაზონის ამსახველ რიცხვზე. მაგ: თუ გვინდა შემთხვევითი რიცხვები 0 დან 99-მდე უნდა გამოვიყენოთ შემდეგი კოდი $Rnd * 100$.

აღსანიშნავია რომ Rnd გვაძლევს არა მთელ რიცხვებს. იმისათვის რომ მივიღოთ მთელი რიცხვები უნდა გამოვიყენოთ ოპერატორი Int. ოპერატორი Int მოაცილებს რიცხვს წილადურ ნაწილს (არ ახდენს დამრგვალებას).

მაშ ასე თუ გვინდა მივიღოთ შემთხვევითი რიცხვები 0 დან 99-მდე უნდა გამოვიყენოთ შემდეგი კოდი $Int (Rnd * 100)$.

აღსანიშნავია, რომ ამ კოდით პროგრამა მოგვცემს შემთხვევით რიცხვებს, მაგრამ მათ მოგვცემს მუდამ ერთნაირი თანმიმდევრობით. ეს იმას ნიშნავს რომ პროგრამის თავიდან გაშვებისას იგივე რიცხვებს მივიღებთ. ეს ასე რომ არ იყოს ამისათვის გამოიყენება ოპერატორი

Randomize, რომელიც უნდა დაინეროს ზემოთ მოყვანილი კოდის თავზე. Randomize მუშაობს როგორც “ბანქოს აჩეხვის” პრინციპი.

საბოლოო კოდი მიიღებს სახეს:

```
Randomize  
Int (Rnd * 100).
```

თამაში “ჩაფიქრებული რიცხვის გამოცნობა”

შევქნათ პროექტი თამაში “ჩაფიქრებული რიცხვის გამოცნობა”. ალბათ ეს თამაში ყველამ კარგად იცის.

პროგრამის გაშვებისას ტექსტურ ბლოკში ჩაენერება შემთხვევითი რიცხვი, მაგრამ მას მოთამაშე ვერ დაინახავს. შემდეგ მოთამაშე იწყებს თამაშს და მეორე ტექსტურ ბლოკში ჩანერს სავარაუდო რიცხვს, პროგრამა კი დაუნერს ჩაფიქრებული რიცხვი (რიცხვი რომელიც ჩანერილია პირველ ტექსტურ ბლოკში და რომელიც არ ჩანს) მეტია შეყვანილ რიცხვზე თუ ნაკლები. როცა მოთამაშე გამოიცნობს ჩაფიქრებულ რიცხვს პროგრამა გამოიტანს შესაბამის შეტყობინებას. ასევე გვექნება მესამე ტექსტური ბლოკი სადაც დაითვლება მოთამაშის მცდელობათა რაოდენობა (პირდაპირი მთვლელი).

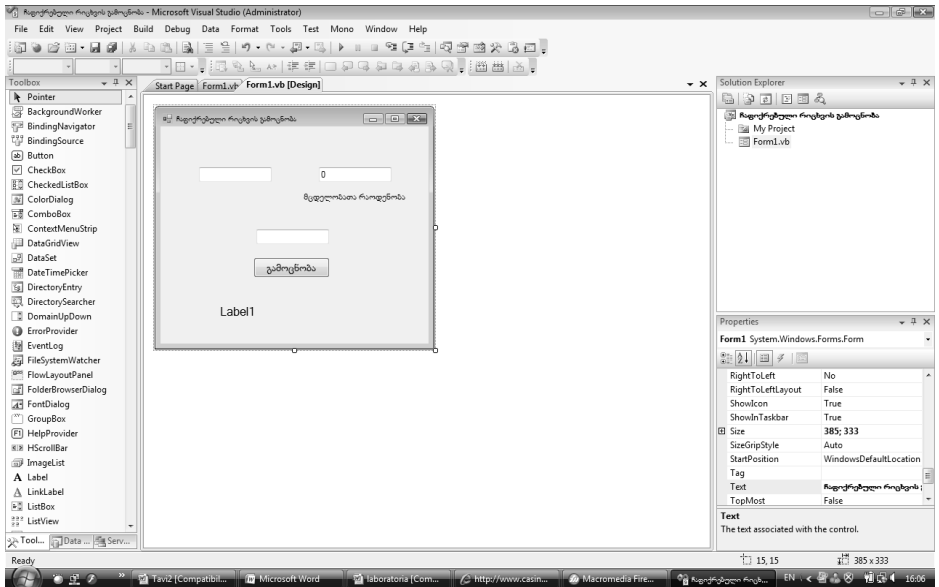
- გაუშვით პროგრამა **Visual Basic 2008** (ან 2010).
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project**.
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანექით ლილაკს **New Project**.
- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Windows Forms Application (Windows დანართი)**.
- ველში **Name** ჩაწერეთ შესაქმნელი პროექტის სახელი.
- შემდეგ დააჭირეთ **OK** ლილაკს.

გაიხსნება ცარიელი ფორმა.

ფორმას მივანიჭოთ სახელი “ჩაფიქრებული რიცხვის გამოცნობა”, ვიყენებთ ფორმის თვისებას **Text**.

ელემენტთა პანელიდან ფორმაზე მოვათავსოთ 3 ელემენტი **TextBox**, 2 ელემენტი **Label** და ერთი ელემენტი **Button**. ელემენტი **Button**-ს თვისებათა ფანჯარიდან შევუცვალოთ ნარწერა და დავანეროთ “გამოცნობა”. ფორმის ზედა მარჯვენა მხარეს განლაგდება **TextBox1**, მის გვერდით **TextBox2** და ქვემოთ **TextBox3**. **TextBox2**-ის ქვემოთ მოვათავსოთ ელემენტი **Label** ნარწერით “მცდელობათა რაოდენობა”. მეორე ელემენტი **Label** კი ლილაკის ქვეშ. მისი თვისება **Text** ნავშალოთ და დავტოვოთ ცარიელი.

Textbox3-ში ჩვენერთ 0, იმისათვის რომ მოვლელის ათვლა ნულიდან დაიწყოს.



ინტერფეისი შექმნილია. დავაკლიკოთ ფორმაზე ორჯერ და გახსნილ ფანჯარაში ჩავწეროთ შემდეგი კოდი:

```
Randomize()
TextBox1.Text = Int(Rnd() * 100)
```

დავაკლიკოთ Button1-ზე ორჯერ და გახსნილ ფანჯარაში ჩავწეროთ შემდეგი კოდი:

```
If Val(TextBox2.Text) > Val(TextBox1.Text) Then
Label1.Text = "ჩაფიქრებული რიცხვი ნაკლებია"
```

```

        If Val(TextBox2.Text) < Val(TextBox1.Text) Then
Label1.Text = "ჩაფიქრებული რიცხვი მეტია"

        If Val(TextBox2.Text) = Val(TextBox1.Text) Then
            Label1.Text = "თქვენ გამოიგანით ჩაფიქრებული
რიცხვი"
            TextBox1.Visible = True
        End If

        TextBox3.Text = TextBox3.Text + 1
    End Sub

```

მთლიანი კოდი გამოიყურება შემდეგნაირად:

```

Public Class Form1

    Private Sub Form1_Load(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
MyBase.Load
        Randomize()
        TextBox1.Text = Int(Rnd() * 100)

    End Sub

    Private Sub Button1_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button1.Click
        If Val(TextBox2.Text) > Val(TextBox1.Text) Then
Label1.Text = "ჩაფიქრებული რიცხვი ნაკლებია"
        If Val(TextBox2.Text) < Val(TextBox1.Text) Then
Label1.Text = "ჩაფიქრებული რიცხვი მეტია"

        If Val(TextBox2.Text) = Val(TextBox1.Text) Then
            Label1.Text = "თქვენ გამოიგანით ჩაფიქრებული
რიცხვი"
            TextBox1.Visible = True
        End If

```

```
    TextBox3.Text = TextBox3.Text + 1
End Sub
```

```
End Class
```

და ბოლოს რადგანაც საჭიროა რომ მოთამაშემ არ უნდა დაინახოს ჩაფიქრებული რიცხვი მოვნიშნოთ **TextBox1** (სადაც ჩაენერება ჩაფიქრებული რიცხვი) და მის თვისებას **Visible** მივანიჭოთ მნიშვნელობა **False**).

დააკვირით კოდის ნაწილს:

```
If Val(TextBox2.Text) = Val(TextBox1.Text) Then
    Label1.Text = "თქვენ გამოიცანით ჩაფიქრებული
რიცხვი"
    TextBox1.Visible = True
End If
```

აქ ჩვენ გამოვიყენეთ მრავალსტრიქონიანი ოპერეტორი **If ...Then**. იმისათვის რომ თუ პირობა სრულდება, შესრულდეს ორი ბრძანება. პირველი: გამოვიდეს წარწერა "თქვენ გამოიცანით ჩაფიქრებული რიცხვი" და მეორე: გამოჩნდეს ჩაფიქრებული რიცხვი.

ჩვენი პროექტი დასრულებულია. გავუშვათ პროგრამა. დავინყოთ თამაში ☺.



თამაში “კამათელი”

შევქნათ პროექტი თამაში “კამათელი”.

ლილაკზე “გაგორება” დაჭერისას შემთხვევითი წესით უნდა მივიღოთ კამათლების მხარეები.

- გაუშვით პროგრამა **Visual Basic 2008** (ან 2010).
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project**.
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანეჭით ლილაკს **New Project**.
- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Windows Forms Application (Windows** დანართი).
- ველში **Name** ჩაწერეთ შესაქმნელი პროექტის სახელი.
- შემდეგ დააჭირეთ **OK** ლილაკს.

გაიხსნება ცარიელი ფორმა.
ფორმას მივანიჭოთ სახელი “კამათელი”, ვიყენებთ ფორმის თვისებას **Text**.

ელემენტთა პანელიდან **Toolbox** ფორმაზე მოვათავსოთ 12 ელემენტი **PictureBox**, 2 ელემენტი **TextBox** და ერთი ელემენტი **Button**. ელემენტი **Button**-ს თვისებათა ფანჯარიდან შევუცვალოთ ნარნერა და დავანეროთ “გაგორება”.

მოვნიშნოთ ერთი ელემენტი **PictureBox1** და დავაჭიროთ პატარა სამკუთხედს მის ზედა მარჯვენა კიდეზე, გაიხსნება მენიუ სადაც ავირჩიოთ **Choose Image**. შემდეგ დავაჭიროთ ღილაკს **Import**, მოვნახოთ სურათი “კამათელი1” რომელიც მდებარეობს CD დისკზე და დავაჭიროთ ღილაკს **OK**.

Size Mode-ს მივანიჭოთ **AutoSize**.

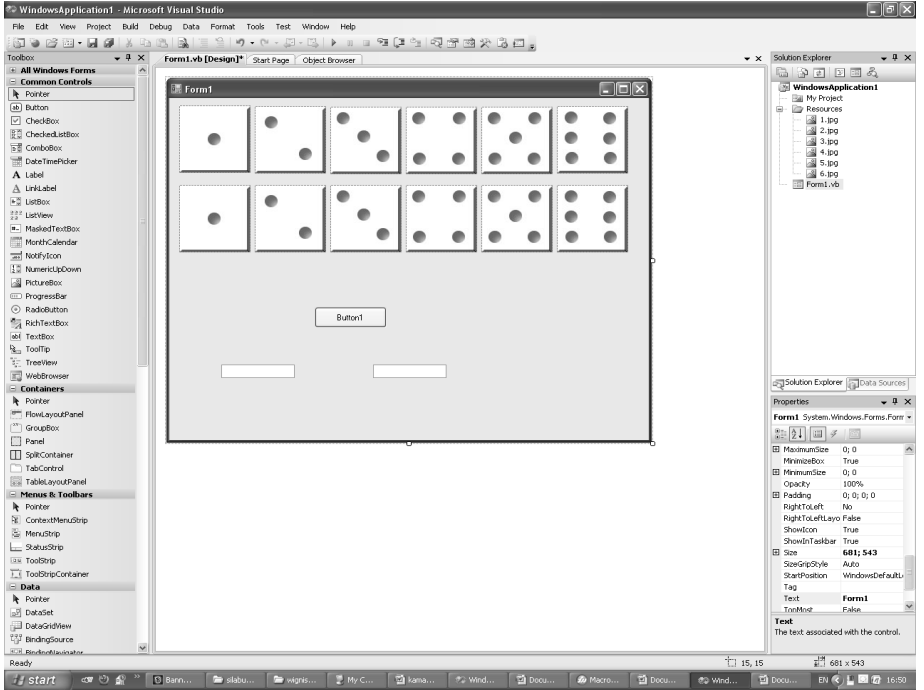
ასევე მოვიქცეთ სხვა 5 **PictureBox**-ისთვისაც. მათ შევურჩიოთ კამათლის სხვა მხარის სურათები ისე, რომ ისინი არ მეორდებოდნენ.

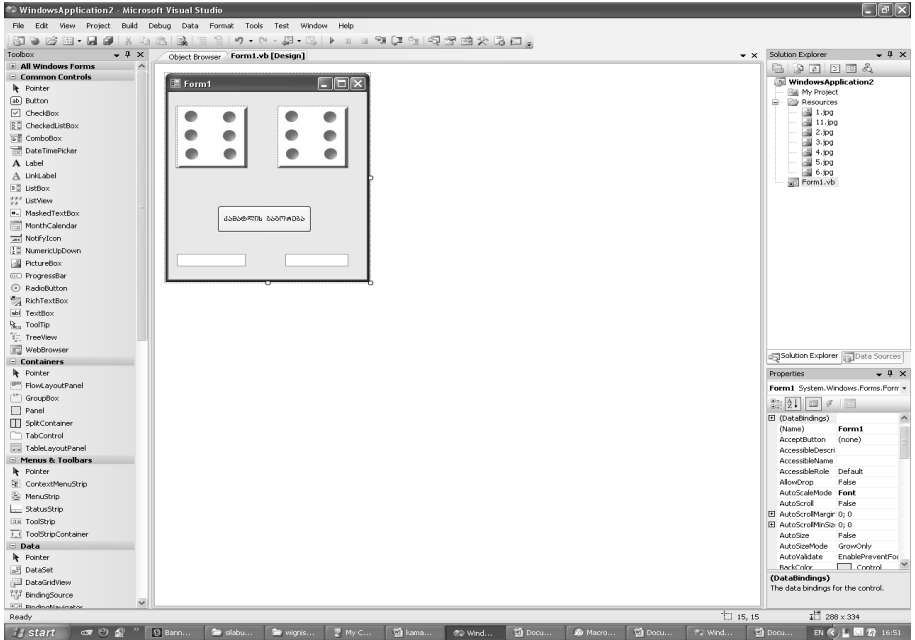
ეს რაც შეეხება პირველი კამათლის 6 მხარეს. ახლა კი იგივე გავაკეთოთ შემდეგი 6 **PictureBox**-ისთვისაც. მივიღებთ მეორე კამათლის 6 მხარესაც (თითო სურათი მეორდება 2-ჯერ). სურთი . .

პირველი 6 სურათი დავალაგოთ ერთმანეთზე, ისე რომ ერთმანეთს ფარავდნენ (ერთი კამათელი), შემდეგ კი მეორე 6 სურათი დავალაგოთ ერთმანეთზე.

ახლა ავამოძრაოთ ჩვენი კამათლები, ამისათვის დავაჭიროებთ შემთხვევითი რიცხვები ერთიდან ექვსამდე. პირველი კამათლის შემთხვევითი რიცხვები უნდა ჩაინეროს პირველ **TextBox**-ში. მეორესი კი მეორე **TextBox**-ში.

როცა გვექნება შემთხვევითი რიცხვები მათ დაუკავშირებთ ჩვენს **PictureBox**-ებს. მაგ თუ **TextBox**-ში წერია 3, გამოჩნდება კამათლის მხარე სადაც არის სამი წერტილი, სხვა სურათები კი გაქრება, თუ **TextBox**-ში წერია 6, გამოჩნდება კამათლის მხარე სადაც არის ექვსი წერტილი, სხვა სურათები კი გაქრება და მოცემულ მომენტში გვექნება მხოლოდ ერთი სურათი.





ორჯერ დავაკლიკოთ ელემენტი Button-ზე და ჩავენეროთ შემდეგი კოდი:

```
Randomize()
TextBox1.Text = Int(Rnd() * 6) + 1
Randomize()
TextBox2.Text = Int(Rnd() * 6) + 1
```

ეს კოდი ორივე ტექსტურ ბლოკში გამოიყვანს განსხვავებულ შემთხვევით რიცხვებს. + 1 დავჭირდა იმისათვის, რომ შემთხვევითი რიცხვები იყოს არა 0 დან 5-მდე არამედ 1 დან 6-მდე.

შემდეგ გავაგრძელოთ:

```
If TextBox1.Text = 1 Then
    PictureBox1.Visible = True
```

```
Else
    PictureBox1.Visible = False
End If
If TextBox1.Text = 2 Then
    PictureBox2.Visible = True
Else
    PictureBox2.Visible = False
End If
If TextBox1.Text = 3 Then
    PictureBox3.Visible = True
Else
    PictureBox3.Visible = False
End If
If TextBox1.Text = 4 Then
    PictureBox4.Visible = True
Else
    PictureBox4.Visible = False
End If
If TextBox1.Text = 5 Then
    PictureBox5.Visible = True
Else
    PictureBox5.Visible = False
End If
If TextBox1.Text = 6 Then
    PictureBox6.Visible = True
Else
    PictureBox6.Visible = False
End If
If TextBox2.Text = 1 Then
    PictureBox7.Visible = True
Else
    PictureBox7.Visible = False
End If
If TextBox2.Text = 2 Then
    PictureBox8.Visible = True
Else
    PictureBox8.Visible = False
End If
If TextBox2.Text = 3 Then
    PictureBox9.Visible = True
Else
```

```

        PictureBox9.Visible = False
    End If
    If TextBox2.Text = 4 Then
        PictureBox10.Visible = True
    Else
        PictureBox10.Visible = False
    End If
    If TextBox2.Text = 5 Then
        PictureBox11.Visible = True
    Else
        PictureBox11.Visible = False
    End If
    If TextBox2.Text = 6 Then
        PictureBox12.Visible = True
    Else
        PictureBox12.Visible = False
    End If

```

ვფიქრობთ კოდის ამ ნაწილში იოლად გაერკვევით. თითოეული სურათი უნდა გამოჩნდეს მხოლოდ მასინ როცა ტექსტურ ბლოკში მისი შესაბამისი ციფრი წერია. ეს გამოიწვევს იმას, რომ მოცემულ მომენტში გამოჩნდება მხოლოდ ერთი სურათი.

მთლიანი კოდი გამოიყურება შემდეგნაირად:

```

Private Sub Button1_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button1.Click
    Randomize()
    TextBox1.Text = Int(Rnd() * 6) + 1
    Randomize()
    TextBox2.Text = Int(Rnd() * 6) + 1

    If TextBox1.Text = 1 Then
        PictureBox1.Visible = True
    Else
        PictureBox1.Visible = False
    End If

```

```
End If
If TextBox1.Text = 2 Then
    PictureBox2.Visible = True
Else
    PictureBox2.Visible = False
End If
If TextBox1.Text = 3 Then
    PictureBox3.Visible = True
Else
    PictureBox3.Visible = False
End If
If TextBox1.Text = 4 Then
    PictureBox4.Visible = True
Else
    PictureBox4.Visible = False
End If
If TextBox1.Text = 5 Then
    PictureBox5.Visible = True
Else
    PictureBox5.Visible = False
End If
If TextBox1.Text = 6 Then
    PictureBox6.Visible = True
Else
    PictureBox6.Visible = False
End If
If TextBox2.Text = 1 Then
    PictureBox7.Visible = True
Else
    PictureBox7.Visible = False
End If
If TextBox2.Text = 2 Then
    PictureBox8.Visible = True
Else
    PictureBox8.Visible = False
End If
If TextBox2.Text = 3 Then
    PictureBox9.Visible = True
Else
    PictureBox9.Visible = False
End If
```

```

If TextBox2.Text = 4 Then
    PictureBox10.Visible = True
Else
    PictureBox10.Visible = False
End If
If TextBox2.Text = 5 Then
    PictureBox11.Visible = True
Else
    PictureBox11.Visible = False
End If
If TextBox2.Text = 6 Then
    PictureBox12.Visible = True
Else
    PictureBox12.Visible = False
End If

End Sub

```

და ბოლოს, რადგანაც აუცილებელი არაა რომ ვხედავდეთ ტექსტური ბლოკებს სადაც შემთხვევითი რიცხვები ჩნდება მათ თვისებას **Visible** მივანიჭოთ მნიშვნელობა **False**.

ჩვენი პროექტი დასრულებულია. გავუმვათ პროგრამა. გავაგოროთ კამათელი ☺.





გამოსახულების დატანა ფორმაზე და მართვის ელემენტებზე უ მეთოდით შეიძლება. შეიძლება მათზე გრაფიკული გამოსახულებების დატანა გრაფიკული ფაილებიდან. შეიძლება გამოვიყენოთ მართვის ობიექტები Visual Basic Power Packs. ასევე შეიძლება პროგრამულად დავხატოთ ჩვენთვის სასურველი ფიგურები და სხვ.

კლასი Graphics

რაზე შეგვიძლია ვხატოთ? ფორმაზე, ღილაკზე, ტექსტურ ბლოკზე, გრაფიკულ ელემენტზე *PictureBox*, სხვა მართვის ელემენტებზე. მაგრამ ფორმის კლასებში კოდში ჩვენ არ გაგვაჩნია ასეთი პროგრამები. სახატავად V-ში არის სპეციალური კლასი **Graphics** რომელიც განლაგებულია სახელების სივრცეში **System.Drawing**. მის კოდს შეიცავენ პროგრამები, რომლებიც განკუთვნილია გრაფიკასთან სამუშაოდ.

თუ ჩვენ კოდის ფანჯარაში ავკრეფთ კოდს **Graphics** და მის შემდეგ ნერტილს დავსვამთ არაფერს არ მივიღებთ. ასე მარტივად მის მეთოდებს ვერ გამოვიყენებთ.

ჩვენ შეგვიძლია მათი გამოყენება მხოლოდ მაშინ თუ კლასიდან შევქმნით ობიექტს. ობიექტი როგორც თქვენ უკვე

იცით კლასის ეკზემპლარია. ის კლასის კოპიოს წარმოადგენს და გააჩნია მისი ყველა შესაძლებლობები.

ჩვენს ამოცანას წარმოადგენს დავხაზოთ ლურჯი ფერის მონაკვეთი ორ მოცემულ წერტილს შორის პირველი წერტილის კოორდინატებია $x=50$, $y=20$. მეორე წერტილის კოორდინატებია $x=200$, $y=100$.

შექმენით ახალი პროექტი. მოათავსეთ ფორმაზე ელემენტი ლილაკი და მის კოდში ჩაწერეთ შემდეგი კოდი:

```
Public Class Form1

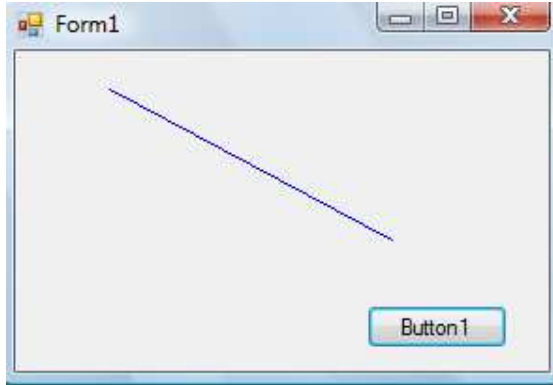
    Private Sub Button1_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button1.Click
        Dim გრაფიკა As Graphics
        გრაფიკა = Me.CreateGraphics
        გრაფიკა.DrawLine(Pens.Blue, 50, 20, 200,
100)
    End Sub
End Class
```

თავიდან გამოვაცხადეთ კლასის ობიექტი. შემდეგ შევექმენით ეს ობიექტი. ჩვენ ვიცით რომ ობიექტები ცხადდება ოპერატორით **New**, მაგრამ ამ კლასის შემთხვევაში ჩვენ სხვა მეთოდს გამოვიყენებთ. ფორმას და მართვის ელემენტებს გააჩნია სპეციალური მეთოდი **CreateGraphics**, რომლის საშუალებით ისინი ქმნიან თავიანთ საკუთარ ობიექტს კლასიდან **Graphics**. ობიექტი შეიქმნა და რადგანაც ის შეიქმნა მეთოდით **CreateGraphics**, რომელიც ეკუთვნის ფორმას, მას შეუძლია დახატვა მხოლოდ ფორმაზე.

ახლა შეგვიძლია დახატვა. ამისათვის გამოვიყენებთ მეთოდს **DrawLine**. ოთხი რიცხვი წარმოადგენს მის საწყის დასაბოლოო კოორდინატებს.

რაც შეეხება **Pens.Blue**, **Blue** განსაზღვრავს ფერს. **Pens** კი წარმოადგენს კლასს, რომელიც მიუთითებს რომ

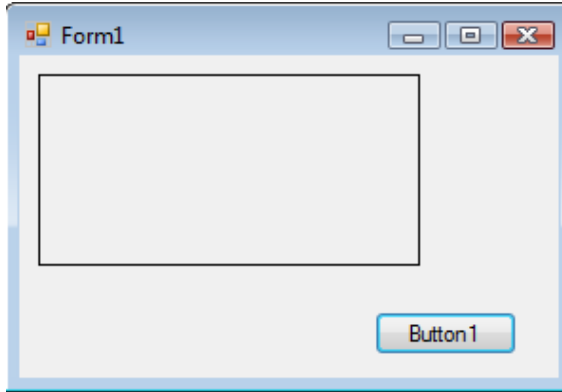
დახატვა ხდება კალმით. წარმოიდგინეთ ყუთი სადაც ჩალაგებულია სხვადასხვა ფერის კალმები, ჩვენ კი ამოვიღეთ ცისფერი კალამი.



შეგვიძლია ფორმაზე დავამატოთ მეორე ღილაკიც და მას მივცეთ ნითელი მონაკვეთის დახატვის ფუნქცია.

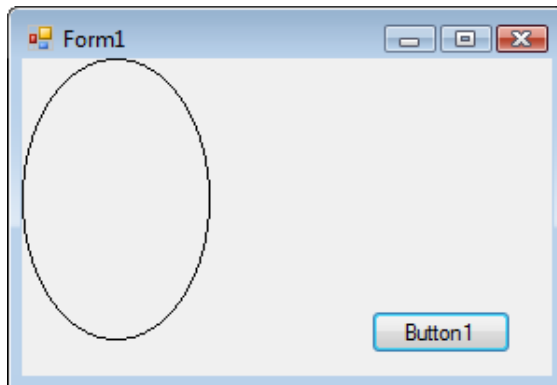
```
Private Sub Button1_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button1.Click  
    Dim გრაფიკა As Graphics  
    გრაფიკა = Me.CreateGraphics  
    გრაფიკა.DrawRectangle(Pens.Black, 10, 10, 200,  
100)  
End Sub
```

ახლა ანალოგიურად შეგვიძლია დავხატოთ მართკუთხედი. გამოვიყენებთ მეთოდს **DrawRectangle**. მასში თქვენ მარტივად გაერკვევით. 10 და 10 წარმოადგენს ზედა მარცხენა წერტილის კოორდინატს. 200 სიგანე, 100 სიმაღლე.



ანალოგიურად შეგვიძლია დავხატოთ ელიფსი, ან წრე. წრე იქნება ელიფსი თანაბარი სიმაღლით და სიგანით.

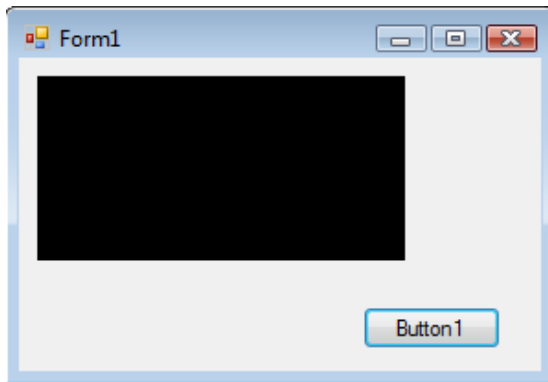
```
Private Sub Button1_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button1.Click  
    Dim გრაფიკა As Graphics  
    გრაფიკა = Me.CreateGraphics  
    გრაფიკა.DrawEllipse(Pens.Black, 0, 0, 100, 150)  
End Sub
```



გავაფერადოთ ფიგურები

მარტივად შეგვიძლია გავაფერადოთ ჩვენს მიერ შექმნილი ფიგურები. მართკუთხედის დასახატად და გასაფერადებლად გამოვიყენებთ მეთოდს `FillRectangle` და კლასს `Brushes`. შეგვიძლია ასევე შევარჩიოთ ფერები.

```
Private Sub Button1_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button1.Click  
    Dim გრაფიკა As Graphics  
    გრაფიკა = Me.CreateGraphics  
    გრაფიკა.FillRectangle(Brushes.Black, 10, 10, _  
200, 100)  
End Sub
```



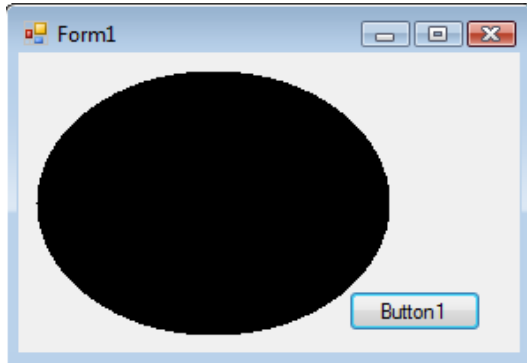
ელიფსის დასახატად და გასაფერადებლად გამოვიყენებთ მეთოდს `FillRectangle` და კლასს `Brushes`.

```
Private Sub Button1_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button1.Click  
    Dim გრაფიკა As Graphics  
    გრაფიკა = Me.CreateGraphics
```

```

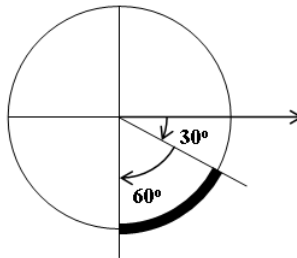
გრაფიკა.FillEllipse(Brushes.Black, 10, 10, 200, _
150)
End Sub

```



დავხატოთ რკალები და სექტორები

ახლა კი დავხატოთ რკალი. ამისათვის გამოვიყენებთ მეთოდს DrawArc. პირველი ოთხი რიცხვი ისეთივე მნიშვნელობისაა როგორც ელიფსის შემთხვევაში. რაც შეეხება ბოლო ორს 30 აღნიშნავს რკალის დაწყების გრადუსს, 60 კი იმას რომ რკალი 60 გრადუსიანია.



```

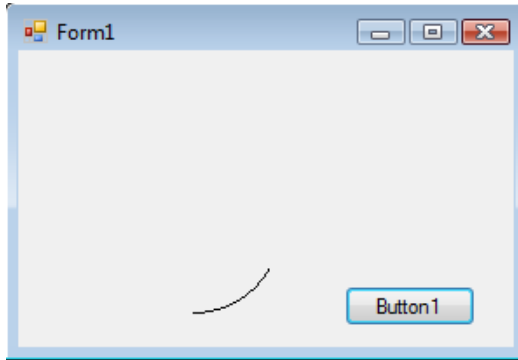
Private Sub Button1_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button1.Click
    Dim გრაფიკა As Graphics
    გრაფიკა = Me.CreateGraphics

```

```

        გრაფიკა.DrawArc(Pens.Black, 50, 50, 100, 100, _
30, 60)
End Sub

```

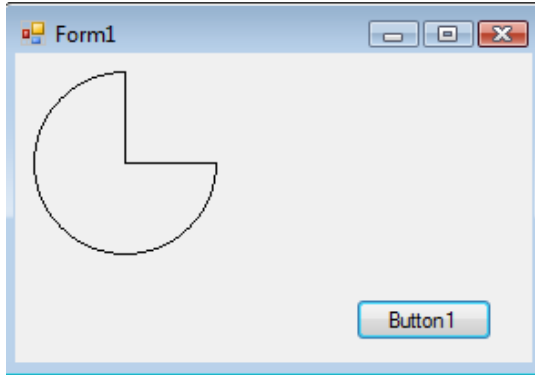


ახლა კი დავხატოთ სექტორი. ამისათვის გამოვიყენებთ მეთოდს DrawPie. პირველი ოთხი რიცხვი ისეთივე მნიშვნელობისაა როგორც ელიფსის შემთხვევაში. რაც შეეხება ბოლო ორს 0 აღნიშნავს სექტორის დანყების გრადუსს, 270 კი იმას რომ სექტორი 270გრადუსიანია.

```

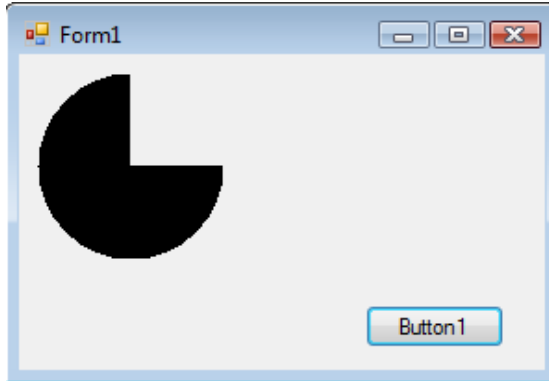
Private Sub Button1_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button1.Click
    Dim გრაფიკა As Graphics
    გრაფიკა = Me.CreateGraphics
    გრაფიკა.DrawPie(Pens.Black, 10, 10, 100, 100, _
0, 270)
End Sub

```



ასევე შეგვიძლია დავხატოთ და გავაფერადოთ სექტორი.
ამისათვის გამოვიყენებთ შემდეგ კოდს.

```
Private Sub Button1_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button1.Click  
    Dim გრაფიკა As Graphics  
    გრაფიკა = Me.CreateGraphics  
    გრაფიკა.FillPie(Brushes.Black, 10, 10, 100,  
100, 0, 270)  
End Sub
```



პროექტი “მშვილდოსანი”

შევქნათ პროექტი თამაში “მშვილდოსანი”. ღილაკზე დაჭერით უნდა გავისროლოთ ისარი. ისრის გასროლით კი, უნდა გავხიოთ რაც შეიძლება მეტი ბუშტი, რომლებიც ერთმანეთის პარალელურად მოძრაობენ.

- გაუშვით პროგრამა **Visual Basic 2008** (ან 2010).
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project**.
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანეკით ღილაკს **New Project**.
- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Windows Forms Application** (**Windows** დანართი).

- ველში **Name** ჩანერეთ შესაქმნელი პროექტის სახელი.
- შემდეგ დააჭირეთ **OK** ღილაკს.

გაიხსნება ცარიელი ფორმა.

ფორმას მივანიჭოთ სახელი “მშვილდოსანი” (ვიყენებთ ფორმის თვისებას **Text**).

ელემენტთა პანელიდან **Toolbox**, ფორმაზე მოვათავსოთ 7 ელემენტი **PictureBox**, 2 ელემენტი **TextBox**, 2 ელემენტი **Timer** და 3 ელემენტი **Button**. ელემენტებს **Button**-ს თვისებათა ფანჯარიდან შევუცვალოთ ნარწერა და დავანეროთ “გასროლა”, “დანყება” და “ახალი თამაში”.

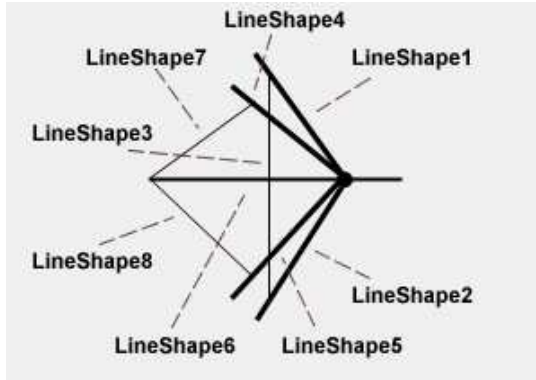
მოვნიშნოთ ერთი ელემენტი **PictureBox1** და დავაჭიროთ პატარა სამკუთხედს მის ზედა მაჯვენა კიდეზე, გაიხსნება მენიუ სადაც ავირჩიოთ **Choose Image**. შემდეგ დავაჭიროთ ღილაკს **Import**, მოვნახოთ სურათი “ბუშტი1”, რომელიც მდებარეობს **CD** დისკზე და დავაჭიროთ ღილაკს **OK**.

Size Mode-ს მივანიჭოთ **AutoSize**.

ასევე შევფურჩიოთ დანარჩენ ელემენტებს **PictureBox** სხვადასხვა ფერის ბუშტების სურათები (**CD** დისკიდან).

ასევე დავაჭიროებთ ელემენტი **LineShape**, რომლის საშუალებით დავხატავთ მშვილდ-ისარს. თუ თქვენი პროგრამის პაკეტში არ შედის **Visual Basic Power Packs**, მაშინ ამ ელემენტს მართვის ელემენტების პანელზე ვერ იპოვით. ამ შემთხვევაში მოგიწევთ **Visual Basic Power Packs**-ის რომელიმე ვერსიის ინტერნეტიდან გადმოწერა.

ელემენტებისაგან **LineShape** დახატეთ მშვილდ-ისარი ისე, როგორც ნაჩვენებია სურათზე. უნდა დავხატოთ ის ორ მდგომარეობაში (მოზიდვის და გასროლის).



გავხსნათ კოდის ფანჯარა. და ჩავწეროთ შემდეგი კოდი:

```
Public Class Form1
    Dim a As Byte
    Dim b As Byte, c As Byte, d As Byte, k As Byte, f As
    Byte, m As Byte
```

ბუშტებს გააცოცხლებს Timer1-ში ჩანერილი კოდი:

```
Private Sub Timer1_Tick(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Timer1.Tick
    PictureBox1.Top = PictureBox1.Top - 1
    PictureBox2.Top = PictureBox2.Top - 1
    PictureBox3.Top = PictureBox3.Top - 1
    PictureBox4.Top = PictureBox4.Top - 1
    PictureBox5.Top = PictureBox5.Top - 1
    PictureBox6.Top = PictureBox6.Top - 1
    PictureBox7.Top = PictureBox7.Top - 1
```

ისრის გასროლა:

```
Private Sub Timer2_Tick(ByVal sender As System.Object, _ByVal_
e As System.EventArgs) Handles Timer2.Tick
    LineShape6.X1 = LineShape6.X1 + 10
    LineShape6.X2 = LineShape6.X2 + 10
End Sub
```

```
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
    Timer2.Enabled = True
    LineShape4.Visible = False
    LineShape5.Visible = False
    LineShape7.Visible = False
    LineShape8.Visible = False
    LineShape1.Visible = True
    LineShape2.Visible = True
    LineShape3.Visible = True

    TextBox1.Text = TextBox1.Text + 1
End Sub
```

```
Private Sub Button2_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button2.Click
    Timer1.Enabled = True
End Sub
```

ბუშტის გახეთქვის (გაქრობის) პირობის განსაზღვრისათვის ჩვენ გამოვიყენებთ ისრისა და ბუშტების კოორდინატებს, თუ ისინი ერთმანეთს დაემთხვევა (გარკვეულ დიაპაზონში) მაშინ ბუშტი გაქრება (თვისება Visible მიიღებს

მნიშვნელობას True). ჩვენს შემთხვევაში ისრის წვერის კოორდინატებს წარმოადგენს LineShape6.X2 და LineShape6.Y2. X2 კოორდინატი იცვლება ისრის გადაადგილებისას. ბუშტი უნდა გაქრეს მაშინ როცა ისრის წვერის კოორდინატები იქნება ნახატის “შიგნით” და არ გასცდება მას. ქვემოთ მოყვანილი კოდი ახდენს სწორედ ამის რეალიზებას. 74 არის ნახატის სიგანე (PictureBox-ის თვისება With), 93 ნახატის სიგრძე (PictureBox-ის თვისება Heigh).

```

    If LineShape6.X2 > PictureBox1.Location.X And _
LineShape6.X2 < (PictureBox1.Location.X + 74) And _
LineShape6.Y2 > PictureBox1.Location.Y And LineShape6.Y2 < _
(PictureBox1.Location.Y + 93) Then
        PictureBox1.Visible = False
    End If

    If LineShape6.X2 > PictureBox2.Location.X And _
LineShape6.X2 < (PictureBox2.Location.X + 74) And _
LineShape6.Y2 > PictureBox2.Location.Y And LineShape6.Y2 < _
(PictureBox2.Location.Y + 93) Then
        PictureBox2.Visible = False
    End If

    If LineShape6.X2 > PictureBox3.Location.X And _
LineShape6.X2 < (PictureBox3.Location.X + 74) And _
LineShape6.Y2 > PictureBox3.Location.Y And LineShape6.Y2 < _
(PictureBox3.Location.Y + 93) Then
        PictureBox3.Visible = False
    End If

    If LineShape6.X2 > PictureBox4.Location.X And _
LineShape6.X2 < (PictureBox4.Location.X + 74) And _
LineShape6.Y2 > PictureBox4.Location.Y And LineShape6.Y2 < _
(PictureBox4.Location.Y + 93) Then
        PictureBox4.Visible = False
    End If

    If LineShape6.X2 > PictureBox5.Location.X And _
LineShape6.X2 < (PictureBox5.Location.X + 74) And _
LineShape6.Y2 > PictureBox5.Location.Y And LineShape6.Y2 < _
(PictureBox5.Location.Y + 93) Then
        PictureBox5.Visible = False
    End If

```

```

        If LineShape6.X2 > PictureBox6.Location.X And _
LineShape6.X2 < (PictureBox6.Location.X + 74) And _
LineShape6.Y2 > PictureBox6.Location.Y And LineShape6.Y2 < _
(PictureBox6.Location.Y + 93) Then
            PictureBox6.Visible = False
        End If

        If LineShape6.X2 > PictureBox7.Location.X And _
LineShape6.X2 < (PictureBox7.Location.X + 74) And _
LineShape6.Y2 > PictureBox7.Location.Y And LineShape6.Y2 < _
(PictureBox7.Location.Y + 93) Then
            PictureBox7.Visible = False
        End If

    If LineShape6.X2 > 1450 Then
        LineShape6.X1 = 70
        LineShape6.X2 = 211
        Timer2.Enabled = False

        LineShape4.Visible = True
        LineShape5.Visible = True
        LineShape7.Visible = True
        LineShape8.Visible = True
        LineShape1.Visible = False
        LineShape2.Visible = False
        LineShape3.Visible = False
    End If

    If PictureBox1.Top < 0 Then PictureBox1.Top = 638
    If PictureBox2.Top < 0 Then PictureBox2.Top = 638
    If PictureBox3.Top < 0 Then PictureBox3.Top = 638
    If PictureBox4.Top < 0 Then PictureBox4.Top = 638
    If PictureBox5.Top < 0 Then PictureBox5.Top = 638
    If PictureBox6.Top < 0 Then PictureBox6.Top = 638
    If PictureBox7.Top < 0 Then PictureBox7.Top = 638

```

მოხვედრების (გამსკლარი ბუშტების) რაოდენობის განსაზღვრა:

```

    If PictureBox1.Visible = False Then
        a = 1
    Else
        a = 0
    End If

```

```

End If
If PictureBox2.Visible = False Then
    b = 1
Else
    b = 0
End If
If PictureBox3.Visible = False Then
    c = 1
Else
    c = 0
End If

If PictureBox4.Visible = False Then
    k = 1
Else
    k = 0
End If

If PictureBox5.Visible = False Then
    f = 1
Else
    f = 0
End If

If PictureBox6.Visible = False Then
    m = 1
Else
    m = 0
End If

If PictureBox7.Visible = False Then
    d = 1
Else
    d = 0
End If

    TextBox2.Text = a + b + c + m + k + f + d
End Sub

```

თავიდან დანწყობა:

```

Private Sub Button3_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button3.Click
    Timer1.Enabled = False
    Timer2.Enabled = False

```

```
TextBox1.Text = 0
TextBox2.Text = 0

PictureBox1.Top = 638
PictureBox2.Top = 638
PictureBox3.Top = 638
PictureBox4.Top = 638
PictureBox5.Top = 638
PictureBox6.Top = 638
PictureBox7.Top = 638

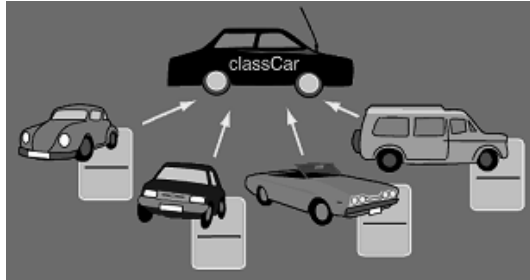
LineShape6.X1 = 70
LineShape6.X2 = 211

LineShape4.Visible = True
LineShape5.Visible = True
LineShape7.Visible = True
LineShape8.Visible = True
LineShape1.Visible = False
LineShape2.Visible = False
LineShape3.Visible = False

PictureBox1.Visible = True
PictureBox2.Visible = True
PictureBox3.Visible = True
PictureBox4.Visible = True
PictureBox5.Visible = True
PictureBox6.Visible = True
PictureBox7.Visible = True

End Sub
End Class
```

ობიექტებზე ორიენტირებული დაპროგრამება



დაპროგრამების ენა Visual Basic 2008 წარმოადგენს ობიექტ-ორიენტირებულ დაპროგრამების ენას (NET ვერსიების წინამორბედი ვერსიები (მაგ: Visual Basic 6) არ წარმოადგენდნენ ობიექტ-ორიენტირებულ დაპროგრამების ენებს).

ეს იმას ნიშნავს, რომ დანართის ყველა ფუნქციონალური ნაწილები განიხილება როგორც ობიექტები, რომლებიც შეიცავენ თვისებებს, შეუძლიათ შეასრულონ გარკვეული მეთოდები და შეუძლიათ მოვლენების გენერაცია.

დამწყები პროგრამისტისათვის აუცილებელია განისაზღვროს განსხვავება კლასსა და ობიექტს შორის. კლასი წარმოადგენს ობიექტის აღწერას, მაშინ როცა ობიექტი წარმოადგენს ამ კლასის კონკრეტულ წარმომადგენელს. მაგ: ავტომობილი ასახავს ობიექტების მთელ კლასს, რომლებსაც აქვთ საჭე ბორბლები მოტორი და შეუძლიათ გადაადგილება. თითოეული კონკრეტული მოდელის ავტომობილი კი წარმოადგენს ამ კლასის ობიექტს, რომელსაც გააჩნია გარკვეული ზომის ბორბლები, გარკვეული ძალის მოტორი და გარკვეული ფორმის საჭე.

ობიექტ-ორიენტირებულ დაპროგრამების ძირითად ცნებებს მიეკუთვნება: ინკაფსულაცია, მემკვიდრეობითობა და პოლიმორფიზმი. მათზე დეტალურად ქვემოთ გვექნება საუბარი.

კლასები და ობიექტები

კლასი წარმოადგენს ობიექტ-ორიენტირებული დაპროგრამების საფუძველს. აქამდე ჩვენ მუდმივად გვქონდა საქმე კლასებთან, მაგრამ ეს იყო კლასები გამზადებული სახით .NET Framework-კლასების ბიბლიოთეკიდან. ახლა ჩვენ სვექმნით ჩვენს საკუთარ კლასებს. მათ სარგებლობას ცვენ მალე სვევიგრზნობთ, ახლა კი უბრალოდ შევისწავლოთ მათი შექმნა მათი ზემოქმედება პროგრამის სხვა მოდულებთან.

.NET Framework-ის კლასებს ჩვენ ვიყენებდით 2 სახით ობიექტის შექმნის გარეშე და მისი შექმნით. პირველ შემთხვევაში ჩვენ უბრალოს ვწერდით კლასის სახელს, სემდეგ წერტილს და სემდეგ ვირჩევდით მის რომელიმე თვისებას. მაგ კლასი *Math* ატემატიკური ოპერაციების შესრულებისათვის. მეორე შემთხვევაში კლასის გამოყენებისათვის ცვენ ჯერ უნდა სეგვექმნა ობიექტი, რომელიც შეიცავდა ამ კლასს. ჯერ დაწერდით ობიექტის სახელს, სემდეგ წერტილს და ასე შემდეგ.

კლასის შექმნისას ცვენ შეგვიძლია მივუთითოთ თუ რომელი ზემოთ მოყვანილი 2 მეთოდით მოხდება მისი გამოყენება.

შვექმნათ კლასი, ამისათვის შევექმნათ ახალი პროექტი → **Project** → **Add New Item** → ფანჯარაში *Add New Item* ავირჩიოთ *Class* → **Open**.

Solution Explorer ში გამოჩნდება ნიშანი *Class1.vb*, პროექტს დაემატება ფანჯარა შესაბამისი კოდით:

```
Public Class Class1  
  
End Class
```

თქვენს წინაშეა თქვენი საკუთარი კლასი.

```
Public Class კლასი
```

```

Public Sub პროცედურა()
    Form1.BackColor = Color.Red
End Sub
End Class

```

ახლა შევექმნათ ობიექტი და გამოვიყენოთ ჩვენს მიერ შექმნილი კლასი. გადმოვიტანოთ ფორმაზე ღილაკი და მასში ჩავწეროთ შემდეგი კოდი:

```

Public Class Form1

    Private Sub Button1_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button1.Click
        Dim ობიექტი As New კლასი
        ობიექტი.პროცედურა()
    End Sub

End Class

```

გავუშვათ პროექტი. როცა დავაჭერთ ღილაკს. ფორმა მიიღებს წითელ ფერს.

კოდის პირველი სტრიქონი ქმნის ობიექტს ომელიც წარმოადგენს ცვენს მიერ შექმნილი კლასის ეგზემპლარს. შექმნილი ობიექტის შემდეგ ნერტილის დასმისას გამოჩნდება სია სადაც არის ცვენს მიერ შექმნილი პროცედურაც.

ჯერ-ჯერობით კლასის შექმნის სარგებლობაზე წარმოდგენა მაინც ბუნდოვანი იქნება, ცოტაც მოვითმინოთ.

კლასის სტრუქტურა

ყოველი კლასი შეიცავს ველებს, მეთოდებს, თვისებების და მოვლენების ნაკრებს (მათ კლასის წევრებს უწოდებენ). მოკლედ განვიხილოთ ყოველი მათგანი:

Visual Basic-ის გრამატიკის თვალსაზრისით მონაცემები რომლებიც ჩანს კლასის გარედან იყოფა თვისებებად და ველებად.

ველები (*field*)—ეს არის ცვლადები, რომელიც ეკუთვნის ამ კლასს ან კლასის ეგზემპლარს და გამოცხადებულია ამ კლასში *Public* და *Friend* მოდიფიკატორებით. მაგ:

Public D As Integer = 100

მეთოდები—წარმოადგენს კლასის პროცედურებს და ფუნქციებს.

თვისებები (*Property*) —საშუალებას გვაძლევს ფორმაში გამოვიძახოთ ფუნქციები. მაგ: გამოვაცხადოთ თვისება ასაკი და მასში უარყოფითი მნიშვნელობის შეტანისას გამოგვიტანოს იფორმაცია შეცდომის შესახებ.

მოვლენები—პროგრამა რეაგირებს მოვლენებზე. მაგ: ლილაკის დაჭერის მოვლენაზე, ყოველი დაჭერისას გამოიძახებს გარკვეულ მეთოდს.

ყოველ კლასს გააჩნია თავისი ხედვის არე, რომელიც განისაზღვრება ხედვის მოდიფიკატორებით. მოდიფიკატორები შეიძლება იყოს შემდეგი ტიპის:

Public, Private, Friend, Protected, Protected Friend.

შევემნათ და გამოვიყენოთ ერთი კლასის ორი ობიექტი

გთხოვთ ყურადღებით წაიკითხოთ ეს თავი. ამ პროექტში კარგად გარკვევა იქნება თქვენთვის ობიექტ-ორიენტირებად დაპროგრამებაში გარკვევის საფუძველი.

წარმოიდგინეთ, რომ თქვენ ხართ გარკვეული სამეურნეო ნაკვეთების მმართველი. ჯერჯერობით თქვენს განკარგულებაში მხოლოდ 2 ნაკვეთია, მაგრამ თქვენ უკვე გადაწყვიტეთ

გამოიყენოთ კომპიუტერი მათ სამართვად. როგორც პროგრამისტმა თქვენ გადანიჭვით შექმნათ კლასი “ნაკვეთი”, სადაც გექნებათ მისი მახასიათებელი ყველა აუცილებელი პარამეტრი (მაგ: ზომები, ფართობები).

- გაუშვით პროგრამა **Visual Basic 2008** (ან 2010).
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project**.
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანექით ლილაკს **New Project** .
- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Windows Forms Application (Windows** დანართი).
- ველში **Name** ჩანერეთ შესაქმნელი პროექტის სახელი.
- შემდეგ დააჭირეთ **OK** ლილაკს.

გაიხსნება ცარიელი ფორმა. ფორმას დავანეროთ სატაური “ნაკვეთები” (თვისება **Text**).

მოვათავსოთ ფორმაზე 5 ტექსტური ბლოკი, 3 ელემენტი ლილაკი და 6 ელემენტი **Label**. შევცვალოთ ელემენტების **Label** თვისება **Text**, **Label6** კი გავასუფთავოთ.

პროექტი შედგება ფორმისაგან და კლასისაგან (ფორმაც თავის მხრივ წარმოადგენს კლასს).

მომხმარებელს ტექსტურ ბლოკებში შეჰყავს პირველი ნაკვეთის შესახებ შემდეგი ინფორმაცია:

- *TextBox1* ნაკვეთის მფლობელი.
- *TextBox2* ნაკვეთის სიგრძე.
- *TextBox3* ნაკვეთის სიგანე.
- *TextBox4* ღობის სიმაღლე.
- *TextBox5* საღებავის ხარჯი ღობის 1 მ²-ზე.

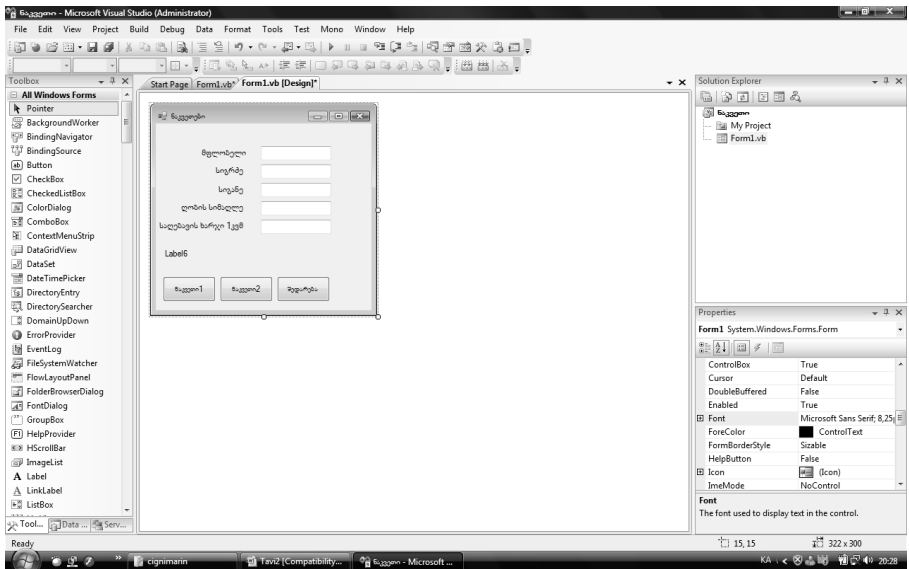
ამის შემდეგ მომხმარებელი დააჭერს ლილაკს და კლასიდან “ნაკვეთი” შეიქმნება ობიექტი “ნაკვეთი 1”, რომელიც

მიიღებს მონაცემებს ტექსტური ბლოკებიდან თავისი ველების სახით.

შემდეგ მომხმარებელს კვლავ შეჰყავს მონაცემები (ამჯერად უკვე მეორე ნაკვეთის) ტექსტურ ბლოკებში და დააჭერს მეორე ლილაკს. კლასიდან “ნაკვეთი” შეიქმნება ობიექტი “ნაკვეთი 2”, რომელიც მიიღებს მონაცემებს ტექსტური ბლოკებიდან თავისი ველების სახით.

ამის შემდეგ მებსიერებაში გვაქვს 2 ობიექტი—კლასის (ნაკვეთი) ექვემპლარები. პროგრამისტს უკვე შეუძლია ისინი გამოიყენოს საკუთარი მიზნებისათვის.

მაგ: მესამე ლილაკზე დაჭერისას და გამოვიდეს ინფორმაცია იმის შესახებ თუ რომელი ნაკვეთის ღობის შესაღებად არის საჭირო საღებავის მეტი დანახარჯი.



```
Public Class Form1
```

```
Dim ნაკვეთი1, ნაკვეთი2 As ნაკვეთი
```

```

Private Sub Button1_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button1.Click
    ნაკვეთი1 = New ნაკვეთი
' კლასიდან იქმნება ობიექტი
    ნაკვეთი1.მფლობელი = TextBox1.Text
    ნაკვეთი1.სიგრძე = TextBox2.Text
    ნაკვეთი1.სიგანე = TextBox3.Text
    ნაკვეთი1.ღობის_სიმაღლე = TextBox4.Text
    ნაკვეთი1.საღებავის_ხარჯი_კვ_მ_ზე = TextBox5.Text
End Sub

```

```

Private Sub Button2_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button2.Click
    ნაკვეთი2 = New ნაკვეთი
' კლასიდან იქმნება ობიექტი
    ნაკვეთი2.მფლობელი = TextBox1.Text
    ნაკვეთი2.სიგრძე = TextBox2.Text
    ნაკვეთი2.სიგანე = TextBox3.Text
    ნაკვეთი2.ღობის_სიმაღლე = TextBox4.Text
    ნაკვეთი2.საღებავის_ხარჯი_კვ_მ_ზე = TextBox5.Text
End Sub

```

```

Private Sub Button3_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button3.Click
    If ნაკვეთი1.საღებავის_ხარჯი_ღობეზე > _
ნაკვეთი2.საღებავის_ხარჯი_ღობეზე Then
        Label6.Text = ("პირველ ნაკვეთს სჭირდება მეტი_
საღებავი, ვიდრე მეორეს")
    End If

    If ნაკვეთი1.საღებავის_ხარჯი_ღობეზე < _
ნაკვეთი2.საღებავის_ხარჯი_ღობეზე Then
        Label6.Text = ("მეორე ნაკვეთს სჭირდება მეტი_
საღებავი ვიდრე პირველს")
    End If

```

```
End If
End Sub
```

კლასი:

```
Public Class ნაკვეთი
    Public მფლობელი As String
    Public სიგრძე, სიგანე As Integer
    Public ღობის_სიმაღლე As Integer
    Public საღებავის_ხარჯი_კვ_მ_ზე As Integer
    Private პერიმეტრი As Integer

    Private Sub გამოვთვალოთ_პერიმეტრი()
        პერიმეტრი = 2 * (სიგანე + სიგრძე)
    End Sub

    Private Function ღობის_ფართობი() As Integer
        გამოვთვალოთ_პერიმეტრი()
        Return პერიმეტრი * ღობის_სიმაღლე
    End Function

    Public Function საღებავის_ხარჯი_ღობეზე() As _
Integer
        Return საღებავის_ხარჯი_კვ_მ_ზე *
ღობის_ფართობი()
    End Function
End Class
```

მიაქციეთ ყურადღება თუ როგორ სევექმენიტ ობიექტი კლასიდან:

ნაკვეთი1 = New ნაკვეთი 'კლასიდან იქმნება ობიექტი

ჩავიხედოთ კლასის კოდში. ჩვენ იქ ვხედავთ 5 ცვლადს, რომლებიც გამოცხადებულია მოდიფიკატორიტ *Public*. ამიტომ ისინი ჩანს ობიექტის გარედან, ისინი ობიექტის ველებს წარმოადგენენ.

ობიექტის შიდა მექანიკა განისაზღვრება მისი პროცედურებით და ფუნქციებით. ობიექტს აქვს ერთადერთი მარტივი ამოცანა, განსაზღვროსრობის შესაღები საღებავის ხარჯი.

ფუნქცია საღებავის ხარჯი ღობეზე აბრუნებს მნისვენელობას, რომელიც საჭიროა გარედან, ამიტომ ის განვსაზღვრეთ როგორც *Public*.

Function ღობის ფორმები აბრუნებს მნიშვნელობას, რომელიც გარედან საჭირო არაა, ამიტომ ის განვსაზღვრეთ როგორც *Private*.

იგივე ეხება პროცედურას გამოვთვალთ პერიმეტრი ().

რათქმაუნდა პერიმეტრიც სეიზღებოდა გამოგვეთვალა ფუნქციის გამოყენებით, მაგრამ მის გამოსათვლელად თვალსაჩინოებისათვის გამოვიყენეთ პროცედურა.

მას ასე ჩვენ შევექმენით კლასი 5 ველით და ერთი მეთოდით, გარდა ამისა მასში არის ცვლადი, პროცედურა და ფუნქცია, რომელიც გარედან არ ჩანს (ინკაფსულაციის პრინციპი).

ნაკვეთები

მფლობელი	გიორგი ლაშვი
სივრძე	1300
სიგანე	830
ღობის სიმაღლე	1
სადგმავის ხარჯი 1კვმ	20

შორე ნაკვეთს სკირდება შეტი სადგმავი ვიდრე პირველს

ნაკვეთი1 ნაკვეთი2 შედარება

კლასის ობიექტების მასივი

თქვენი მეურნეობა იზრდება და უკვე გაქვთ რამოდენიმე ათეული ნაკვეთი. მოგვიწევს პროექტის შეცვლა. ამჯერად ფორმაზე გვექნება მხოლოდ 2 ღილაკი. ყოველი ახალი ობიექტის შექმნა კი შესაძლებელი იქნება ერთიდაიმავე ღილაკზე დაჭერით. როცა ყველა ობიექტი შექმნილია, პროგრამისტს შეუძლია მათი გამოყენება საკუთარი მიზნებისათვის. მაგ: თუ დააჭერს მეორე ღილაკს გამოვა იმ ნაკვეთების მფლობელების სახელები, რომელთა ღობეების შეღებვისას დაიხარჯა 200 კვ-ზე მეტი საღებავი.

```
Public Class Form1

    Dim ნაკვეთი(100) As ნაკვეთები

    Dim k As Integer = 1

    Private Sub Button1_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button1.Click
        ნაკვეთი(k) = New ნაკვეთები           'კლასიდან იქმნება
ობიექტი
        ნაკვეთი(k).ნაკვეთის_ნომერი = k
        ნაკვეთი(k).მფლობელი = TextBox1.Text
        ნაკვეთი(k).სიგრძე = TextBox2.Text
        ნაკვეთი(k).სიგანე = TextBox3.Text
        ნაკვეთი(k).ღობის_სიმაღლე = TextBox4.Text
        ნაკვეთი(k).საღებავის_ხარჯი_კვ_მ_ზე = TextBox5.Text
        k = k + 1
    End Sub
```

```

Private Sub Button2_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Button2.Click
    Dim i As Integer
    For i = 1 To k - 1
        If ნაკვეთი(i).საღებავის_ხარჯი_ღობეზე > 200 _
Then Label6.Text = ნაკვეთი(i).მფლობელი
    Next
End Sub

```

კლასი:

```

Public Class ნაკვეთები
    Public ნაკვეთის_ნომერი As Integer
    Public მფლობელი As String
    Public სიგრძე, სიგანე As Integer
    Public ღობის_სიმაღლე As Integer
    Public საღებავის_ხარჯი_კვ_მ_ზე As Integer
    Private პერიმეტრი As Integer

    Private Sub გამოვთვალოთ_პერიმეტრი()
        პერიმეტრი = 2 * (სიგრძე + სიგანე)
    End Sub

    Private Function ღობის_ფართობი() As Integer
        გამოვთვალოთ_პერიმეტრი()
        Return პერიმეტრი * ღობის_სიმაღლე
    End Function

    Public Function საღებავის_ხარჯი_ღობეზე() As _
Integer
        Return საღებავის_ხარჯი_კვ_მ_ზე *
ღობის_ფართობი()
    End Function
End Class

End Class

```

აქ ჩვენ ორი ნაკვეთის ნაცვლად (ნაკვეთი1 და ნაკვეთი2) გამოვაცხადეთ მასივი შემდგარი 101 ნაკვეთისაგან:

Dim ნაკვეთი(100) As ნაკვეთი

ლილაკის დაჭერისას ცვლადი k -ს მნიშვნელობა ერთით იზრდება.
 $k = k + 1$

ამიტომ ოპერატორით:

ნაკვეთი(k) = New ნაკვეთი 'კლასიდან იქმნება ობიექტი

იქმნება ახალი ობიექტი – რომელიც არის კლასის (ნაკვეთი) ეკზემპლარი.

კლასს ნაკვეთი დაემატა ახალი ველი – ნაკვეთის_ნომერი.

მფლობელი	დავით გერსამია
სივრცე	1200
სივანე	32
ღობის სიმაღლე	1
სადღებავის ხარჯი 1კვმ	2

დავით გერსამია

ნაკვეთი1 სადღებავის ხარჯი >200

ინკაფსულაცია

ინკაფსულაცია წარმოადგენს მექანიზმს, რომელიც აერთიანებს მონაცემებს და მეთოდებს, რომლებიც ახდენენ ამ მონაცემებით მანიპულირებას და იცავს ერთსა და მეორესაც გარეშე ზემოქმედების და არასწორი გამოყენებისაგან. სხვა სიტყვებით რომ ვთქვათ ეს არის თვით კლასის შიგნით არსებული კლასის რეალიზების დაფარული დეტალები. მაგ: თუ ჩვენ გვაქვს კლასი, რომელიც საშუალებას გვაძლევს გადმოვტვირთოთ ფაილი ინტერნეტიდან მთელი ფუნქციონალი რომელიც უზრუნველყოფს ინტერნეტთან დაკავშირებას, მონაცემთა მიმოცვლას, კავსირის დახურვას ყველა გამოყენებული ცვლადები დაფარული უნდა იყოს ამ მოცემულ კლასში. არ არსებობს იმის აუცილებლობა, რომ კლასის მომხმარებელმა დაინახოს კლასის რეალიზაციის ყველა დეტალი.

მოქმედება სრულდება კლასის შიგნით. კლასი წარმოადგენს "შავ ყუთს". მომხმარებელი კი ნახულობს მხოლოდ იმ მინიმუმს (ინტერფეისს) რომელიც ზედაოიროზე გამოდის და რომელიც მას სჭირდება.

წარმოიდგინეთ თვითმფრინავი რომელიც მიფრინავს. ცვენ მას ვხედავთ, მაგრამ არ ვიცით თუ რა ხდება მის სიგნით. როგორ მუშაობს მისი მოტორი და ასშ. ჩვენ ვერანაირად ვერ გავიგებთ ამას და ვერც ვერანაირად მოვახდენთ ზემოქმედებას მასზე. ის ჩვენს წესებს არ ექვემდებარება. მოძრაობს წესებით რომელსაც კარნახობენ აეროპორტიდან.

მემკვიდრეობითობა

კლასები იშვიათად შეიცავენ თავის თავში აბსოლუტურად მთელ ფუნქციონალს. როგორც წესი ფუნქციონალის ნაწილი გადაიტანება სხვა კლასებიდან. ამ პროცესს მემკვიდრეობითობა ეწოდება. Visual Basic 2008-ში ყველა კლასი პირდაპირ ან ირიბად მემკვიდრეა System.Object კლასის.

მემკვიდრეობითობა უზრუნველყოფს შვილეულ კლასებში გამოყენებულ იქნას მშობელი კლასის ფუნქციონალი და საჭიროების შემთხვევაში დაამატოს მას ახალი.

შესაძლებელია არა მარტო ფუნქციონალის დამატება, არამედ არსებულის შეცვლაც. ამისათვის არსებობს პოლიმორფიზმი.

მემკვიდრეობითობა და პოლიმორფიზმი განსაკუთრებით მნიშვნელოვანია იმ პროექტებში სადაც არის არა ერთი და ორი კლასი, არამედ რამოდენიმე.

წარმოვიდგინოთ, რომ ჩვენ შევექმენით კლასი “ავტომობილი” რომელსაც გააჩნია მხოლოდ ბორბლები, საჭე და მოტორი. ჩვენ კიდევ შევექმენით 2 კლასი: ავტობუსი და სატვირთო ავტომობილი. ყველა მატყანს აქვს მოტორი, საჭე და ბორბლები, მაგრამ დაემატა კიდევ სხვა პარამეტრები. მაგ; სატვირთო ავტომობილს აქვს საბარგული, ავტობუსს სალონი მგზავრებისათვის. როგორ შევექმნათ მოცემული სამი კლასის კოდი. ალბათ უმჯობესია მოვახდინოთ კლასის “ავტომობილი” კოპირება და შემდეგ დავამატოთ მას სხვა კლასებისათვის (ამ სამი კლასიდან) დამახასიათებელი პარამეტრები (ცვლადები, პროცედურები და ფუნქციები).

მაგრამ არსებობს უფრო მარტივი და მოსახერხებელი მეთოდი “მემკვიდრეობითობა” ჩვენ უბრალოდ გამოვაცხადებთ, რომ ახალი კლასი “სატვირთო ავტომობილი წარმოადგენს კლასის “ავტომობილი” მემკვიდრეს. ამ შემთხვევაში “სატვირთო ავტომობილი” არაცხადად შეიძენს მისი მშობლის “ავტომობილის” მთელს კოდს. ისე რომ “სატვირთო ავტომობილი”-ის კოდში არ არის ჩანერილი არაფერი, ჩვენ ის უკვე შეგვიძლია გამოვიყენოთ როგორც “ავტომობილი”. იმისათვის კი რომ მივანიჭოთ “სატვირთო ავტომობილი-ს” დამატებითი თვისებები ჩვენ დავწერთ ახალ პროცედურებს და ფუნქციებს. ასევე მოვიქცევით ავტობუსის შემთხვევაშიც.

საინტერესოა ის ფაქტი, რომ თუ ჩვენ შევიტანთ ცვლილებას მშობელ კლასში, ეს ავტომატურად აისახება შვილეულ კლასებზეც. შვილებს შეიძლება ყავდეთ კიდევ შვილები (კლასი “ავტომობილის” შვილიშვილები) და ეს ნესები ვრცელდება მათზეც.

მოცემულია კლასი cls მართკუთხედი. მისი საქმეა – მომხმარებელს მისცეს საშუალება მართკუთხედის სიგრძით და სიგანით გამოთვალოს მისი ფართობი და პერიმეტრი.

```
Public Class cls მართკუთხედი
    Public სიგრძე As Integer
    Public სიგანე As Integer
```

```
    Public Function ფართობი() As Integer
        Return სიგრძე * სიგანე
    End Function
    Public Function პერიმეტრი() As Integer
        Return 2 * სიგრძე + 2 * სიგანე
    End Function
End Class
```

ნარმოვიდგინოთ რომ, შემდეგში გაჩნდა აუცილებლობა, რომ პარალელეპიპედის სიგრძით, სიგანით და სიმაღლით გამოვთვალოთ მისი მოცულობა. ჩვენ შეგვიძლია ამისათვის შევცვალოთ ჩვენი კლასი და დავამატოთ კოდი;

```
Public სიმაღლე As Integer

    Public Function მოცულობა() As Integer
        Return ფართობი() * სიმაღლე
    End Function
```

მაგრამ ჩვენ არ გვსურს კლასის *cls* მართკუთხედი გაფართოება. ასევე ჩვენ მას ვიყენებთ ხშირად და არ გვსურს რომ მან გამოიყენოს კომპიუტერის ზედმეტი რესურსი.

ამიტომ შევქმნით ახალ კლასს – *cls* პარალელეპიპედი:

```
Public Class cls პარალელეპიპედი
    Inherits cls მართკუთხედი
    Public სიმაღლე As Integer

    Public Function მოცულობა () As Integer
        Return ფართობი () * სიმაღლე
```

End Function
End Class

როგორც ვხედავთ კლასი გამოვიდა პატარა, მაგრამ აკეთებს ყველაფერს რაც საჭიროა. ეს მემკვიდრეობითობის დამსახურებაა.

Inherits cls მართკუთხედი საუბრობს იმაზე, რომ მას მემკვიდრეობით გადაეცა კლასი cls მართკუთხედი –ის ყველა კომპონენტი. ამისათვის გამოვიყენეთ ოპერატორი **Inherits**.

პოლიმორფიზმი

პოლიმორფიზმი ეწოდება ფუნქციონალის შეცვლის უნარს, რომელიც მემკვიდრეობით გადაცემულია ბაზისური კლასიდან. პოლიმორფიზმი წარმოადგენს ერთი სახელის მქონე პროცედურების და ფუნქციების მიერ სხვადასხვა მოქმედების შესრულების უნარს.

მაგ; თუ ავტომობილისათვის “გაჩერება” ეს მხოლოდ გაჩერებაა, ავტობუსისათვის ეს გაჩერებაა და თან მის შესახებ მიკროფონით გამოცხადება. მისი განხორციელება ხდება მემკვიდრეობითობით და “ინტერფეისებით”.

მემკვიდრეობითობისას პოლიმორფიზმი ვლინდება მაშინ, როცა მემკვიდრეს ვუცვლით მშობლის პროცედურას. აქ ჩვენ ინტერფეისებს არ განვიხილავთ. შეიძლება ითქვას, რომ ინტერფეისი წარმოადგენს გარკვეულ “ნილაბს”, რომელსაც ობიექტი მოირგებს ჩვენთან ურთიერთობისათვის.

დავუბრუნდეთ ჩვენს მეურნეობას. ვიმუშავეთ ჩვენი პროექტის პირველ ვერსიაზე.

Public Class ნაკვეთი

Public მფლობელი As String

Public სიგრძე ,სიგანე As Integer

Public ღობის სიმაღლე As Integer

Public Shared საღებავის ხარჯი კვ მ-ზე As Integer

Public Function პერიმეტრი() As Integer

Return 2 * (სიგრძე +სიგანე)

End Function

Public Function ღობის ფართობი () As Integer

Return პერიმეტრი() * ღობის სიმაღლე

End Function

Public Function საღებავის ხარჯი ღობეზე () As Integer

Return საღებავის ხარჯი კვ მ-ზე * ღობის ფართობი ()

End Function
End Class

ახლა კი წარმოვიდგინოთ, რომ გაჩნდა ერთი მფლობელი, რომელმაც გადაწყვიტა ქონდეს ღობე არა მთლიანი, არამედ ფიცრებს შორის არეებით. რათქმაუნდა ასეთ ღობეზე დაახლოებით 2-ჯერ ნაკლები საღებავი დაიხარჯება. შევქმნათ ასეთი ნაკვეთებისათვის ახალი კლასი ნაკვეთი არეები.

ახალი კლასი განსხვავებული იქნება კლასისაგან “ნაკვეთი” მხოლოდ ღობის ფართობით. ამიტომ ის შევქმნათ როგორც კლასის “ნაკვეთი” მემკვიდრე.

Public Class ნაკვეთი არეები
Inherits ნაკვეთ

Public **Overrides** Function ღობის ფართობი () As Integer
Return 0.5 * პერიმეტრი() * ღობის სიმაღლე
End Function
End Class

ჩვენ ვხედავთ, რომ მემკვიდრეს განსაზღვრული აქვს ფუნქცია იმავე სახელით როგორც მშობელს.

Overrides, ნიშნავს “გადაძალავს, გარდაქმნის”. იგულისხმება – გადაძალავს მშობლის ფუნქციას. ამავე მიზნით ჩვენ მშობლის ფუნქციის სათაურში დავამატებთ ოპერატორს **Overridable**, რაც საშუალებას გვაძლევს გამოვიყენოთ **Overrides**.

Public **Overridable** Function ღობის ფართობი () As Integer
Return პერიმეტრი() * ღობის სიმაღლე
End Function

შევამონმოთ ჩვენს მიერ შექმნილი კლასი. გამოვიყენოთ ფორმა სამი ღობით. ერთი ღობითი შექმნის ობიექტს ჩვეულებრივი ღობით, მეორე კი ფიცრებს შორის არეებით. მესამე

კი გამოიტანს საღებავის ხარჯს თითოეული ლობისათვის. ჩვენ უკვე გვაქვს ორი ფუნქცია ფართობი. ერთი გააჩნია მშობელს, მეორე კი შთამომავალს.

ახლა როცა ჩვენ შთამომავლის კოდში მივმართავთ ფუნქციას ფართობი, მოხდება მისი ფუნქციის გამოძახება და არა მშობლის. მაგრამ როგორ მოვიქცეთ მაშინ თუ დაგვჭირდა მივმართოთ მშობლის იმავე ფუნქციას? მაგალითად მიგვაჩნია რომ უფრო მარტივია გამოვთვალოთ ჩვეულებრივი ლობის ფართობი და ის ორზე გავყოთ.

შევცვალოთ ფუნქცია კლასში ნაკვეთი არეებ :

Public Overrides Function ლობის ფართობი () As Integer

Return 0.5 * **MyBase.** ლობის ფართობი

End Function

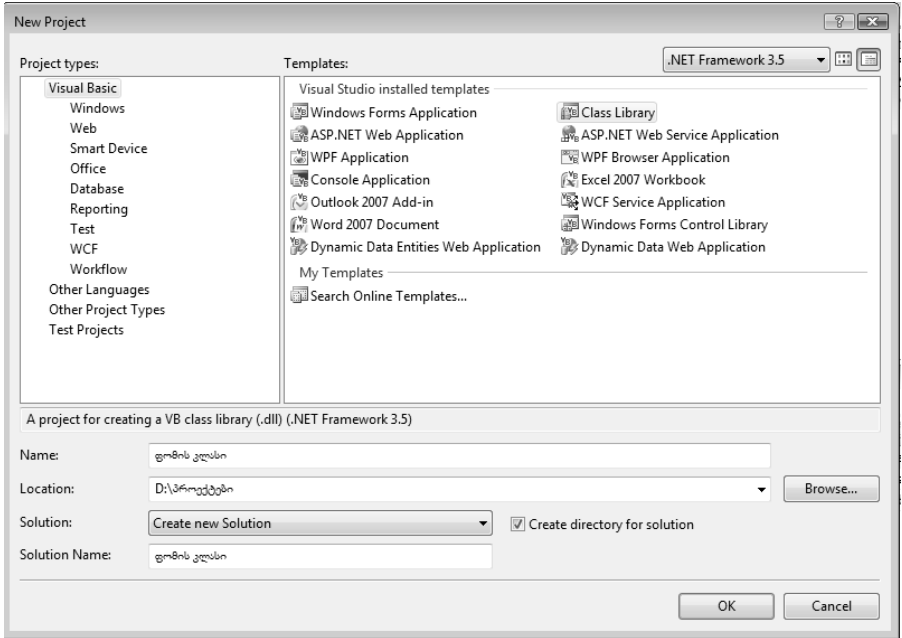
MyBase წერტილით მიანიშნებს, რომ საუბარია მშობლის ფუნქციაზე.

ვიზუალური კლასის შექმნა

Visual Basic 2008-სი ვიზუალური კლასები არაფრით არ განსხვავდება სხვა დანარჩენისაგან. შესაძლებელია მართვის ელემენტების და ფორმის კლასის შექმნა. მართვის ელემენტების სექმნისათვის გამოიყენება პროექტის განსაკუთრებული ტიპი **Windows Forms Control Library**. ის უნდა ავირჩიოთახალი პროექტის შექმნის ფანჯარაში.

ამჯერად ჩვენ შევქმნით ფორმის კლასს. როგორც მოგეხსენებათ ფორმა წარმოადგენს ჩვეულებრივ კლასს. შევქმნათ ფორმის მარტივი კლასი სადაც შეგვიძლია სახელის და გვარის შეყვანა.

- გაუშვით პროგრამა **Visual Basic 2008** (ან 2010).
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project** (პროექტის შექმნა).
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანეჭით ლილაკს **New Project**.
- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Class Library**.



დავამატოთ პროექტში ფორმა. **Solution Explorer**-ში მოვნიშნოთ პროექტის სახელი და დავაჭიროთ მაუსის მარჯვენა ლილაკს, გამოსულ კონტექსტურ მენიუში ავირჩიოთ **Add** შემდეგ **Windows Form**. ფორმას შევუცვალოთ წარწერა (თვისება **Text**) და დავანეროთ “შეიყვანეთ სახელი”. ფორმას ასევე შევუცვალოთ სახელი (თვისება **Name**) და დავრქვათ “**MyForm**”.

ელემენტთა პანელიდან ფორმაზე გადმოვიტანოთ ორი ტექსტური ბლოკი, ორი ელემენტი **Label** და ერთი ლილაკი. ელემენტებს **Label** შევუცვალოთ თვისება **Text** (სახელი, გვარი).

კლასი მზადაა. შევინახოთ ის. აღნიშნული კლასის გამოყენება შესაძლებელია სხვა პროექტებშიც.

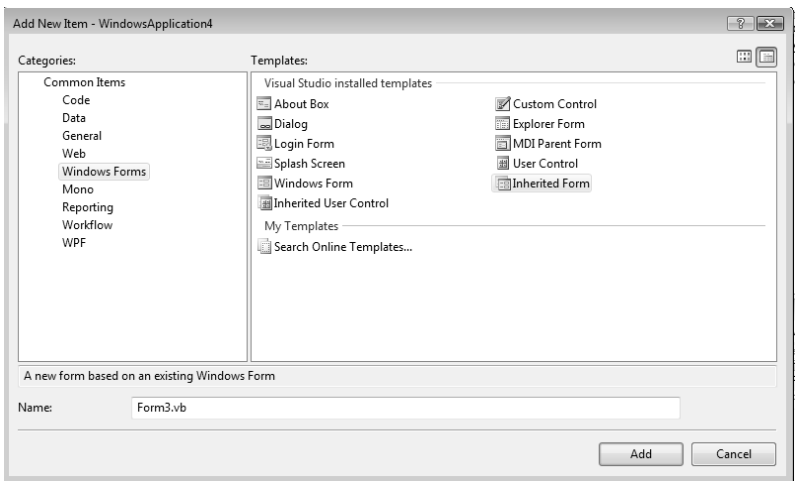
კლასის მუშაობის შესამოწმებლად შევქმნათ ახალი პროექტი;

- გაუშვით პროგრამა **Visual Basic 2008** (ან 2010).

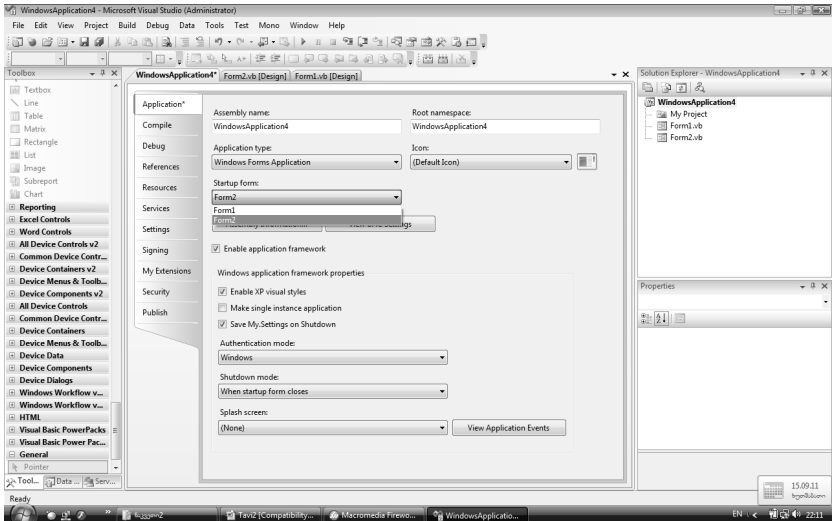
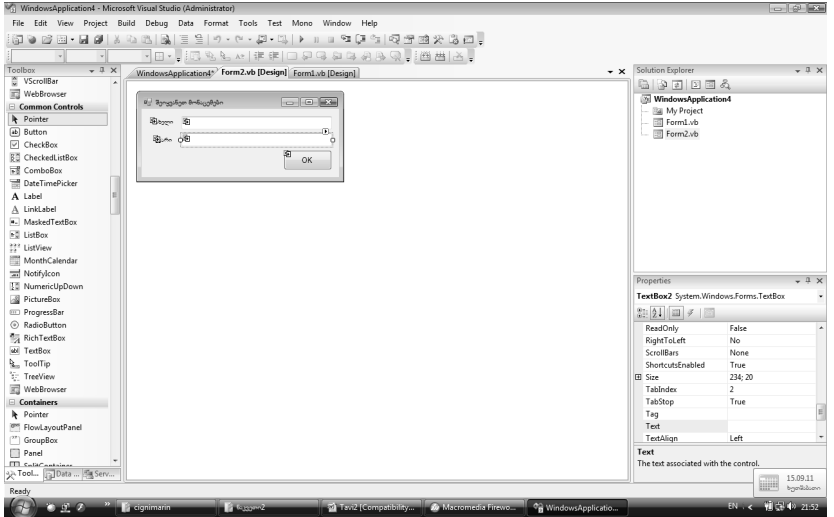
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project** (პროექტის შექმნა).
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანეჭით ლილავს **New Project**.
- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Windows Forms Application (Windows დანართი)**.
- ველში **Name** ჩანერეთ შესაქმნელი პროექტის სახელი.
- შემდეგ დააჭირეთ **OK** ლილავს.

გაიხსნება ცარიელი ფორმა.

დავამატოთ პროექტში ფორმა. **Solution Explorer**-ში მოვნიშნოთ პროექტის სახელი და დავაჭიროთ მაუსის მარჯვენა ლილავს, გამოსულ კონტექსტურ მენიუში ავირჩიოთ **Add** შემდეგ **Windows Form**. შემდეგ კი მოვნიშნოთ პიქტოგრამა **Inherited Form** (მემკვიდრეობითი ფორმა) და დავაჭიროთ ლილავს **Add**. გაიხსნება ფანჯარა **Inheritance Picker**. დააჭირეთ ლილავს **Browse**, მოვნახოთ ჩვენს მიერ შექმნილი კლასი და დავაჭიროთ ლილავს **OK**.

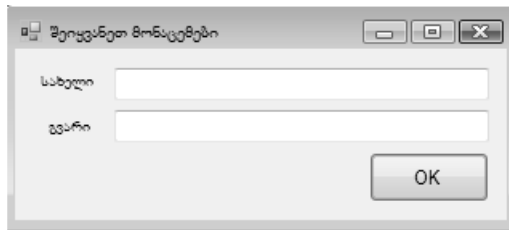


შეიქმნება ახალი კლასი მემკვიდრეობით კლასისაგან MyForm. თუ საჭიროა შეგვიძლია მას დავამატოთ ახალი ელემენტები.



ახლა კი განვსაზღვროთ თუ რომელი იქნება გამშვები ფორმა. ამისათვის **Solution Explorer**-ში მოვნიშნოთ პროექტის სახელი და დავაჭიროთ მაუსის მარჯვენა ღილაკს. კონტექსტურ მენიუში ავირჩიოთ **Properties**. გამოსულ ფანჯარაში ავირჩიოთ გრაფა **Application**, შემდეგ კი ველში **Startup Form** ჩამოვშალოთ სია და ავირჩიოთ **Form2**.

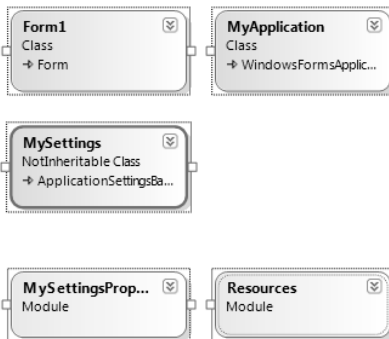
შევვიძლია გავეუშვათ პროგრამა. მას ჯერ არანაირი ფუნქცი არ გააჩნია. საჭიროებისამებრ შევვიძლია დავამატოთ პროცედურები და ფუნქციები.



კლასების დიაგრამა

Visual Basic 2008 საშუალებას გვაძლევს დათვალიერდეს კლასების დიაგრამა. ასევე შევიტანოთ კლასებში ცვლილებები ამ დიაგრამის საშუალებით. ამისათვის შოლუტიონ ხპლორერ-ში ავირჩიოთ პუნქტი **View Class Diagram**. მომხმარებლის მიერ შექმნილი კლასების გარდა დიაგრამაში ვნახავთ ავტომატურად შექმნილ კლასებსა და მოდულებს (მოდული წარმოადგენს სპეციალური ტიპის კლასს). კლასის დიაგრამაზე მაუსის მარჯვენა ღილაკით დაჭერისას გამოჩნდება კონტექსტური მენიუ, საიდანაც შეგვიძლია კლასში ცვლილებების სერტანა.

კლასების დიაგრამები განსაკუთრებით მნიშვნელოვანია რთულ პროექტებში. ის საშუალებას გვაძლევს დავათვალიეროთ კლასები და მათი ურთიერთქმედება.



თამაში “დაჭერობანა”

აქამდე ჩვენ ვქმნიდით ობიექტებს კლასიდან, მაგრამ ისინი რეალური სამყაროს მხიარული ობიექტებისაგან დიდად განსხვავებული იყვნენ. ასევე ალბათ ვერ შევიგრძენით სრულად თუ რა უპირატესობა აქვს ერთი კლასიდან რამოდენიმე ობიექტის შექმნას. ახლა კი შევექმნით პროექტს სადაც ობიექტებს ექნებათ სედარებით რეალური სახე და ვნახავთ რა უპირატესობა აქვს ობიექტებზე ორიენტირებულ დაპროგრამებას.

წარმოვიდგინოთ შემდეგი თამაში: ღილაკზე დაჭერისას 10 წითელი კუბიკი გაიშლება სტარტის ადგილიდან სხვადასხვა მიმართულებით. შემთხვევითი სიჩქარეებით და შემთხვევითი მიმართულებით. ისინი ისე იქცევიან როგორც ბილიარდის ბურთები ბილიარდის მაგიდაზე. როცა ისინი სეეხებიან ვეილს კიდეს უკუიქცევიან რიკოშეტის პრინციპით.

აქვე არის “დამჭერი”, რომელსაც კლავიატურიდან მართავს მოთამაშე. ის მოძრაობს 4 მიმართულებით: მარჯვნივ, მარცხნივ, ზემოთ, ქვემოთ. ასევე შეუძლია გაჩერება. თუ დამჭერის შეეხება კუბიკს ეს უკანასკნელი გაქრება. მოთამაშის ამოცანაა რაც შეიძლება სწრაფად დაიჭიროს 10-ივე კუბიკი. აქვე არის დროის მრიცხველი, რომელიც გაჩერდება როცა ყველა კუბიკი “დაჭერილია”.

რა იქნება ამ პროექტში ობიექტები? ალბათ ხვდებით რომ დამჭერი და კუბიკები.

პროექტში გვექნება ობიექტი “ტაიმერი”. ტაიმერის ყოველი იმპულსე “გააღვიძებს” რიგრიგობით ობიექტებს. ყოველი მათგანი ამის შემდეგ სეასრულებს თავის მოქმედებას. ამიშ შემდეგ გაჩერდება და დაელოდება შემდეგ იმპულსს. იმპულსები კი ძალიან დიდი სიხშირით მოდის და იქმნება შთაბეჭდილება, რომ ობიექტები უწყვეტად მოძრაობენ.

თამაშში 11 ობიექტია “დამჭერი” და 11 კუბიკი. არის 1 ტაიმერი. ის გენერირებს იმპულსებს (100 იმპულსი წამში). როცა დაიწყება

თამაში ტაიმერი გამოუშვებს პირველ იმპულსს, გაცოცხლდება პირველი კუბიკი და ჩართავს თავის თავში ჩადებულ მექანიკას. გადაადგოლდება ერთი ნაბიჯით საწყისი პოზიციიდან. ამის შემდეგ გააქტიურდება მეორე კუბიკი. ისიც როგორც ობიექტი შეასრულებს თავის თავში ჩადებულ მექანიკას და ასე შემდეგ მე-10 კუბიკამდე.

შემდეგ კი გაიღვიძებს “დამჭერი”. ისიც შეასრულებს თავის საქმეს, კერძოდ სეამონმებს დაჭერილი ხო მარ არის შესაბამისი ლილაკები კლავიატურაზე და იმის მიხედვით თუ რომელი ლილაკია დაჭერილი გადაადგილდება განსაზღვრული მიმართულებით.

ანუ მოცემულ მომენტში მოძრაობს მხოლოდ ერთი ობიექტი, მაგრამ რადგანაც იმპულსების სიხშირე საკმაოდ მაღალია იქმნება შთაბეჭდილება რომ ყველაფერი ერთას ხდება. დაფრინავენ კუბიკები, იჭერს “დამჭერი”. ასევე იმას რომ ტაიმერი არაფერს არ ეუბნება ობიექტებს იმის შესახებ თუ რა უნდა გააკეთონ, მათ ეს თვითონ იციან. ტაიმერი უბრალოდ “აღვიძებს” მათ დროის თანაბარ შუალედებში.

ასევე გვექნება მართვის ობიექტი ტექსტური ბლოკი დროის საჩვენებლად და ლილაკი თამაშის თავიდან დასაწყებად.

დავინყოთ პროექტირება. შევქმნათ ახალი პროექტი.

- გაუშვით პროგრამა **Visual Basic 2008** (ან 2010).
- გახსენით დიალოგის ფანჯარა **New Project**.
- ფანჯარაში **Start Page** ამოირჩიეთ ბმული **Create Project**.
- მენიუდან **File** ამოირჩიეთ პუნქტი **New Project**.
- დაანექით ლილაკს **New Project**.
- გახსნილ ფანჯარაში **Templates**, მიუთითეთ შესაქმნელი დანართის (პროგრამის) ტიპი, ჩვენს შემთხვევაში – **Windows Forms Application (Windows დანართი)**.
- ველში **Name** ჩანერეთ შესაქმნელი პროექტის სახელი.
- შემდეგ დააჭირეთ **OK** ლილაკს.

გაიხსნება ცარიელი ფორმა.

მოვათავსოთ ფორმაზე ტექსტური ბლოკი და გავზარდოთ მისი ზომები. ზომის გასაზრდელად დავაჭიროთ პატარა სამკუთხედს ტექსტური ბლოკის ზედა მარჯვენა კიდესთან და მოვნიშნოთ პუნქტი **Multiline**. თვისებათა ფანჯარაში დავარქვათ მას “ველი” (თვისება **Name**).

ასევე მოვათავსოთ ფორმაზე მეორე ტექსტური ბლოკი (დავარქვათ მას “დრო”) და ლილაკი (დავარქვათ მას “თავიდან დაწყება” და დავანეროთ “ახალი თამაში”).

მოვათავსოთ ელემენტი **Label** (დავარქვათ მას დაუჭერელი კუბიკების მთვლელი).

ასევე მოვათავსოთ 2 ელემენტი **Label**. მათ შევუცვალოთ თვისება **Text** “დრო” და “დარჩა” შესაბამისად.

მოვათავსოთ ფორმაზე მართვის ელემენტი **PictureBox** სადაც მოთავსდება დამჭერის გამოსახულება, მიაჩვენებს მას სახელი `pic1` დამჭერი.

ჩასვით მასში სურათი, შეგიძლიათ გამოიყენოთ სურათი, რომელიც არის მოთავსებული CD დისკზე , საქალაქო “დაჭერობანა” ფაილი “დამჭერი”.

გადმოვიტანოთ მართვის ელემენტი **Timer**. თვისებათა ფანჯარაში მის თვისებას **Interval** მივანიჭოთ მნიშვნელობა 10. მის თვისებას **Enabled** კი მნიშვნელობა **True**. მართვის ელემენტი **Timer** ფორმაზე არ გამოჩნდება ის განლაგდება პროგრამის ქვემო პანელიზე.

განვალაგოთ ელემენტები ისე, როგორც ნაჩვენებია სურათზე.

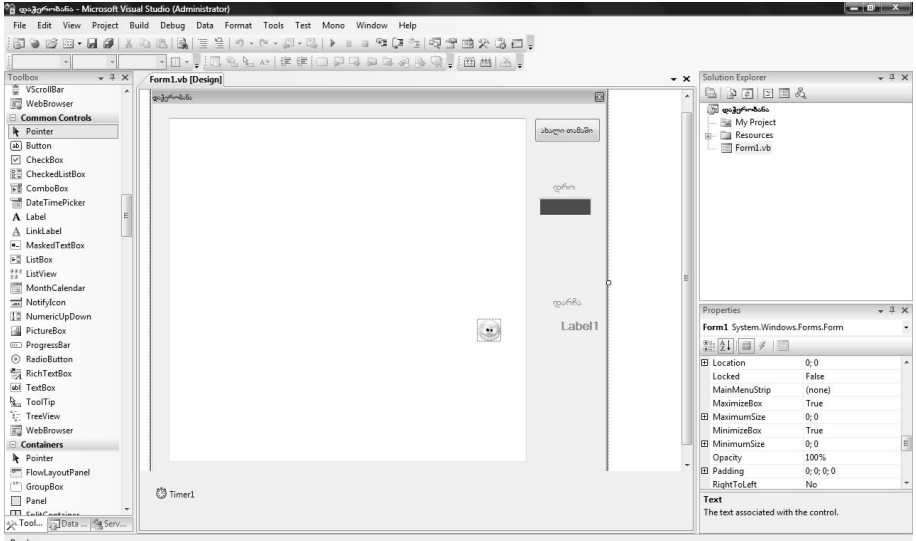
შემდგომში შევექმნით ორ კლასს `cls` დამჭერი და `cls` კუბიკები.

შემდეგ პირველი კლასიდან შევექმნით ობიექტს დამჭერი, მეორესგან კი ობიექტების მასივს კუბიკი.

- პირველ ეტაპზე ჩვენ შევექმნით “დამჭერს”.
- მეორე ეტაპზე ვასწავლით “დამჭერს” მოძრაობას.
- მესამე ეტაპზე კი კუბიკებს.

შევქმნათ კლასი clsდამჭერი.

კლასს არაფერი ვიზუალური არ გააჩნია (თუმცა არსებობს ვიზუალური კლასებიც), ეს არის ცვლადების, კონსტანტების პროცედურების და ფუნქციის ნაკრები. რადგანაც კლასს არ გააჩნია ვიზუალიზაციის საშუალებები მის მოდულს უნევს ფორმის მართვის ელემენტების გამოყენება.



პროგრამის მთლიანი კოდი გამოიყურება შემდეგნაირად:

```
Public Class Form1
```

```
Public დამჭერი As clsდამჭერი
```

```
'გამოვაცხადოთ ობიექტი დამჭერი კლასიდან clsდამჭერი
```

```
Public Const დამჭერის_ზომა As Integer = 30
```

```
Private Const კუბიკების_რაოდენობა As Integer = 10
```

```
Private კუბიკი (კუბიკების_რაოდენობა) As clsკუბიკები
```

```
'გამოვაცხადოთ ობიექტების მასივი კუბიკი კლასიდან clsკუბიკები
```

```

Public pictკუბიკი(კუბიკების_რაოდენობა) As PictureBox
'გამოვაცხადოთ კუბიკების სურათის მასივი
Public დაუჭერელი_კუბიკების_რაოდენობა As Integer

Public Sub Form1_Load(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
MyBase.Load

    KeyPreview = True           'რომ ფორმა რეაგირებდეს
კლავიატურაზე
    დრო.ReadOnly = True
    Randomize()                 'კუბიკები უნდა
გაიფანტოს შემთხვევითი მიმართულებით და შემთხვევითი
სიჩქარეებით

    დამჭერი = New clsდამჭერი           'შევქმნათ ობიექტი
დამჭერი და ობიექტების მასივი კუბიკი:
    Dim i As Integer
    For i = 1 To კუბიკების_რაოდენობა
        კუბიკი(i) = New clsკუბიკები
    Next i
    საწყისი_მდგომარეობა()
    Timer1.Enabled = False

End Sub

Private Sub თავიდან_დაწყება_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) _
Handles თავიდან_დაწყება.Click
    საწყისი_მდგომარეობა()
    Timer1.Enabled = True 'თამაში იწყება

End Sub

Private Sub საწყისი_მდგომარეობა()
    დრო.Text = 0           'დრო გავუტოლოთ 0-ს
    დაუჭერელი_კუბიკების_რაოდენობა = _
კუბიკების_რაოდენობა

```

```

        დაუჭერელი_კუბიკების_მთვლელი.Text = _
კუბიკების_რაოდენობა
        დრო.Focus()
        დამჭერი.საწყისი_მდგომარეობა()      'დამჭერი
უბრუნდება საწყის მდგომარეობას
        Dim i As Integer
        For i = 1 To კუბიკების_რაოდენობა
            კუბიკი(i).საწყისი_მდგომარეობა() 'კუბიკები
უბრუნდება საწყის მდგომარეობას
        Next i

```

```
End Sub
```

```

Public Class clsდამჭერი
    Private x As Double
    Private y As Double
    Private Enum საჭე      'დამჭერი მოძრაობა
        ზემოთ
        მარცხნივ
        ქვემოთ
        მარჯვნივ
        გაჩერება
    End Enum
    Private საჭე1 As საჭე

    Public Sub New()
        Form1.pictდამჭერი.Width = დამჭერის_ზომა
        Form1.pictდამჭერი.Height = დამჭერის_ზომა
    End Sub

    Public Sub საწყისი_მდგომარეობა()
        საჭე1 = საჭე.გაჩერება
        x = Form1.ველი.Left + Form1.ველი.Width * 1 / 4
        y = Form1.ველი.Top + Form1.ველი.Height / 2
        დამჭერი_დავაყენოთ_ადგილზე()
    End Sub

```

```

Private Sub დამჭერი_დავაყენოთ_ადგილზე()
    Form1.pictდამჭერი.Left = x
    Form1.pictდამჭერი.Top = y
End Sub
Public Sub მოქმედება()
    If დამჭერი_კიდესთან() Then
საწყისი_მდგომარეობა()
        ავირჩიოთ_მიმართულება_გადავდგათ_ნაბიჯი()
    End Sub

Private Function დამჭერი_კიდესთან() As Boolean

    If y < Form1.ველი.Top Or y + დამჭერის_ზომა > _
Form1.ველი.Top + Form1.ველი.Height _
    Or x < Form1.ველი.Left Or x + დამჭერის_ზომა > _
Form1.ველი.Left + Form1.ველი.Width Then
        Return True
    Else
        Return False
    End If
End Function

Private Sub
ავირჩიოთ_მიმართულება_გადავდგათ_ნაბიჯი()
    Dim dx As Double = 1 'დამჭერის ნაბიჯი
    ჰორიზონტალურად და ვერტიკალურად ტაიმერის იმპულსებს
    შორის
    Dim dy As Double = 1
    Select Case საჭე1
        Case საჭე.ზემოთ : y = y - dy
        Case საჭე.ქვემოთ : y = y + dy
        Case საჭე.მარცხნივ : x = x - dx
        Case საჭე.მარჯვნივ : x = x + dx
        Case საჭე.გაჩერება
'არაფერი არ უნდა გავაკეთოთ ეი გავჩერდეთ
    End Select
    დამჭერი_დავაყენოთ_ადგილზე()
End Sub

```



```

Public Sub რეაქცია_კლავიატურაზე(ByVal e As _
System.Windows.Forms.KeyEventArgs) _
    Select Case e.KeyCode
        Case Keys.Left : საჭე1 = საჭე.მარცხნივ
        Case Keys.Right : საჭე1 = საჭე.მარჯვნივ
        Case Keys.Up : საჭე1 = საჭე.ზემოთ
        Case Keys.Down : საჭე1 = საჭე.ქვემოთ
        Case Keys.ControlKey : საჭე1 = _
საჭე.გაჩერება
    End Select
End Sub
End Class

```

```

Private Sub Timer1_Tick(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
Timer1.Tick
    დამჭერი.მოქმედება()
'მოქმედებს დამჭერი

    Dim i As Integer
    For i = 1 To კუბიკების_რაოდენობა
        კუბიკი(i).მოქმედება()
'მოქმედებენ კუბიკები
    Next i

    დრო.Text = დრო.Text + 1
    If დაუჭერელი_კუბიკების_რაოდენობა = 0 Then
        Timer1.Enabled = False
    End If

End Sub

```

```

Private Sub Form1_KeyDown(ByVal sender As Object, _
ByVal e As System.Windows.Forms.KeyEventArgs) _
Handles MyBase.KeyDown
    დამჭერი.რეაქცია_კლავიატურაზე(e)

```

```

End Sub
Public Class clsკუბიკები
    Private კუბიკის_ნომერი As Integer
    Private Shared შექმნილი_კუნიკების_რაოდენობა As _
Integer = 0
    Private Const კუბიკის_ზომა As Double = 12
    Private x, y As Double 'კუბიკის
კოორდინატები
    Private dx, dy As Double 'კუბიკის ნაბიჯი
    ჰორიზონტალურად და ვერტიკალურად ტიმერის იმპულსებს
    შორის

    Public Sub New()
        შექმნილი_კუნიკების_რაოდენობა = _
შექმნილი_კუნიკების_რაოდენობა + 1
        კუბიკის_ნომერი = შექმნილი_კუნიკების_რაოდენობა
        'ექმნით მართვის ელემენტს-კუბიკის სურათს:
        Form1.pictკუბიკი(კუბიკის_ნომერი) = New _
PictureBox
        'კუბიკის ზომების დაყენება:
        Form1.pictკუბიკი(კუბიკის_ნომერი).Width = _
კუბიკის_ზომა
        Form1.pictკუბიკი(კუბიკის_ნომერი).Height = _
კუბიკის_ზომა
        Form1.pictკუბიკი(კუბიკის_ნომერი).SizeMode = _
PictureBoxSizeMode.StretchImage

        Form1.pictკუბიკი(კუბიკის_ნომერი).BackColor = _
Color.Red

        Form1.Controls.Add(Form1.pictკუბიკი(კუბიკის_ნომერი))
        'დავამატოთ ახალი კუბიკი

        Form1.pictკუბიკი(კუბიკის_ნომერი).BringToFront()
    End Sub

```

```

Public Sub საწყისი_მდგომარეობა()
    Const მაქს_ნაბიჯი As Double = 1.8
'კუბიკის ნაბიჯის მაქსიმუმი
    'მოვათავსოთ კუბიკი საწყის პოზიციაში:
    x = Form1.ველი.Left + Form1.ველი.Width * 3 / 4
    y = Form1.ველი.Top + Form1.ველი.Height / 2
    მოვათავსოთ_კუბიკის_სურათი_ადგილზე()
    'ნაბიჯის გამოთვლა:
    dx = მაქს_ნაბიჯი * (1 - 2 * Rnd())
'ნაბიჯი ჰორიზონტალურად შემთხვევითია და არ გადააჭარბებს
მაქს_ნაბიჯი-ს
    dy = მაქს_ნაბიჯი * (1 - 2 * Rnd())
'ნაბიჯი ვერტიკალურად შემთხვევითია და არ გადააჭარბებს
მაქს_ნაბიჯი-ს
End Sub

Private Sub მოვათავსოთ_კუბიკის_სურათი_ადგილზე()
    Form1.pictკუბიკი(კუბიკის_ნომერი).Left = x
    Form1.pictკუბიკი(კუბიკის_ნომერი).Top = y
End Sub

Public Sub მოქმედება()
    If დაიჭირეს() Then
კუბიკის_გამოსვლა_თამაშიდან() 'თავიდან კუბიკი განსაზღვრავს,
ხომ არ დაიჭირეს ის,
        რიკოშეტი()
'თუ კიდევ ახლოსაა რიკოშეტი
        ნაბიჯი()
'და, აკეთებს ნაბიჯს
        მოვათავსოთ_კუბიკის_სურათი_ადგილზე()
End Sub

Private Sub ნაბიჯი()
    x = x + dx
    y = y + dy
End Sub

```

```

Private Sub რიკოშეტი()
    If კუბიკი_ჰორიზონტალურ_კიდესთან() Then dy_
= -dy 'რიკოშეტი იატაკიდან ან ჭერიდან
    If კუბიკი_ვერტიკალურ_კიდესთან() Then dx = -
dx 'რიკოშეტი კედლიდან
End Sub

```

```

Private Function
კუბიკი_ჰორიზონტალურ_კიდესთან() As Boolean
    If y < Form1.ველი.Top Or y + კუბიკის_ზომა > _
Form1.ველი.Top + Form1.ველი.Height Then
        Return True
    Else
        Return False
    End If
End Function

```

```

Private Function კუბიკი_ვერტიკალურ_კიდესთან() _
As Boolean
    If x < Form1.ველი.Left Or x + კუბიკის_ზომა > _
Form1.ველი.Left + Form1.ველი.Width Then
        Return True
    Else
        Return False
    End If
End Function

```

```

Private Function დაიჭირეს() As Boolean
    Const სიშორე As Double = 10 'ეს არის
მანძილი რომელზეც დამჭერი მიწვდება კუბიკს
    'თუ ჰორიზონტალური მანძილი დამჭერის და
კუბიკის ცენტრებს შორის ნაკლებია სიშორეზე
    'და თუ ვერტიკალური მანძილი დამჭერის და
კუბიკის ცენტრებს შორის ნაკლებია სიშორეზე, მაშინ
დაჭერილია:

```

```

        If Math.Abs(x -
Form1.pictდამქერი.Location.X - ((დამქერის_ზომა - _
კუბიკის_ზომა) / 2)) < სიშორე _
        And Math.Abs(y - Form1.pictდამქერი.Location.Y - _
((დამქერის_ზომა - კუბიკის_ზომა) / 2)) < სიშორე Then
            Beep()           'ხმოვანი სიგნალი
            Return True
        Else
            Return False
        End If
    End Function

    Private Sub კუბიკის_გამოსვლა_თამაშიდან()
        x = -10000 : y = -10000
        'მოვაშორით კუბიკი თვალთხედვიდან
        dx = 0 : dy = 0
    და რომ არ მოძრაობდეს
        Form1.დაუჭერელი_კუბიკების_რაოდენობა = _
Form1.დაუჭერელი_კუბიკების_რაოდენობა - 1
        Form1.დაუჭერელი_კუბიკების_მთვლელი.Text = _
Form1.დაუჭერელი_კუბიკების_რაოდენობა
    End Sub
End Class
End Class

```

კლასიდან `cls` დამჭერი იქმნება ობიექტი დამჭერი.
ამისათვის გამოიყენება პროცედურა `New`.

შემდეგ ფორმაში გაეშვება `საწყისი_მდგომარეობა`, რომელიც გამოყოფილია ცალკე პროცედურაში, რადგანაც ის გამოიყენება არამართო ფორმის გაშვებისას, არამედ ღილაკზე დაჭერისასაც. მისი საქმეა საწყის მდგომარეობაში მოიყვანოს დროის მთვლელი, კუბიკები და “დამჭერი”.

ინკაფსულაციის პრინციპიდან გამომდინარე დამჭერმა და კუბიკებმა თვითონ უნდა მოათავსონ თავისი თავი საწყის მდგომარეობაში, ამიტომ პროცედურიდან `საწყისი_მდგომარეობა` გაეშვება პროცედურა `დამჭერი.საწყისი_მდგომარეობა ()` და `კუბიკი (i) .საწყისი_მდგომარეობა`.

დამჭერი მოთავსდება ვერტიკალურად ველის შუაში და ჰორიზონტალურად ველის მარცხენა კიდიდან მისი სიგანის მეოთხედზე.

შემდეგ მოხდება `დამჭერი_დავაყენოთ_ადგილზე` პროცედურის გამოძახება.

შეიძლება შეგექმნათ პრობლემები კოდში არსებულ ფორმულებთან მიმართებაში, მაგრამ თუ არ დაიზარებთ და მათ შესწავლაზე მცირედ დროს დახარჯავთ ვფიქრობთ რომ მათში იოლად გაერკვევით.

დამჭერმა უნდა შეასრულოს შემდეგი მოქმედებები:

- შეამოწმოს დაეჯახა თუ არა ველის კიდეს და თუ დაეჯახა დაუბრუნდეს მის საწყის მდებარეობას.
- წინააღმდეგ შემთხვევაში გადაადგილდეს ერთი ნაბიჯით ზემოთ, ვემოთ, მარჯვნივ ან მარცხნივ ან გაჩერდეს როცა დავაჭერთ კლავიშს `Ctrl`.

ეს არის ყველაფერი რაც უნდა გააკეთოს “დამჭერმა”. ის კი რაც უნდა მოხდეს მაშინ, როცა ის დაიჭერს (დაეჯახება) კუბიკს, ჩანერილი იქნება კუბიკის კლასში.

კლასი **clsდამჭერი**-ს პროცედურაში **მოქმედება** დავაპროგრამებთ “დამჭერის” მოქმედებებს. დანარჩენს პროცედურაში **საწყისი_მდგომარეობა**.

პროცედურა **Timer1_Tick** ასრულებს შემდეგ მოქმედებებს:

- გააღვიძებს “დამჭერს” და აიძულებს მას შეასრულოს თავისი პროცედურა **მოქმედება**.
- ერთით გაზარდოს დროის მრიცხველის მნიშვნელობა (მართვის ელემენტი **დრო**).
- ლილაკზე დაჭერისას დროის მრიცხველის მნიშვნელობა გაუტოლოს 0-ს.
- ლილაკზე დაჭერისას აიძულოს “დამჭერი” შეასრულოს პროცედურა **საწყისი_მდგომარეობა**.

რაც შეეხება “დამჭერის” მართვას კლავიატურიდან ჩამონათვალის ტიპის ცვლადი “საჭე” იმახსოვრებს თუ რომელი კლავიშს დააჭირეს.

მოდრაობას კი განაპირობებს პროცედურა **ავირჩიოთ_მიმართულება_გადავღვათ_ნაბიჯი**.

თუ თქვენ ჯერ ვერ გაიგეთ თუ როგორ მუშაობს პროექტი დავყოთ ნაწილებად პროცედურებისა და ფუნქციების გამოძახების თანმიმდევრობა.

პროცედურეს **Form1_Load**, შესრულების შემდეგ სათამაშო “ველზე” არაფერი არ ხდება სანამ არმ ოვა პირველი იმპულსე ტაიმერიდან. როცა მოვა პირველი იმპულსე პირველ რიგში გაეშვება პროცედურა **დამჭერი.მოქმედება**. შემდეგ ერთით გაიზრდება დროის მრიცხველის მნიშვნელობა. ამ ეტაპზე პროცედურა **Timer1_Tick** დაასრულებს მუშაობას. ყველაფერი გაჩერდება მანამ სანამ წამის მეთაფეში არ მოვა შემდეგი იმპულსი ტაიმერიდან.

წარმოვიდგინოთ, რომ ჩვენ ვერ მოვასწარით შეხება კლავიატურასთან. მოვიდა ახალი იმპულსე, ჩართო პროცედურა **დამჭერი.მოქმედება**, ვნახოთ თუ როგორ მუშაობს ის. მისი სხეული სედგება ორი სტრიქონისაგან. პირველი გამოიძახებს

ფუნქციას **დამჭერი_კიდესთან**. რადგან “დამჭერი კიდიდან ჯერ შორსაა ეს ფუნქცია მიიღებს მნიშვნელობას **False**.

შემდეგ ამოქმედდება ფუნქცია **ავირჩიოთ_მიმართულება_გადავდგათ_ნაბიჯი**. რადგან “საჭე” იმყოფება გაჩერების მდგომარეობაში *Select Case არ სეცველის არც x-ს და არც y-ს. ამიტომ პროცედურა დამჭერი_დავაყენოთ_ადგილზე* “დამჭერს” არ გადაადგილებს.

პროცედურა **დამჭერი.მოქმედება** დასრულდება.

შემდეგ კი წამოვა კიდევ მორიგი იმპულსი ტაიმე4რიდან. წარმოვიდგინოთ რომ ჩვენ უკვე დავაჭირეთ კლავიატურაზე ლილაკს, რომელმაც “დამჭერი” მარჯვნივ უნდა გადაადგილოს. მაშინათვე ამუშავდება პროცედურა *Form1_KeyDown*. ის გამოიძახებს პროცედურას **რეაქცია_კლავიატურაზე**. რომელიც მიანიჭებს ცვლადს “საჭე” მნიშვნელობას “მარჯვნივ”.

როცა მოვა შემდეგი იმპულსი, კვლავ გადავალთ პროცედურაზე **დამჭერი.მოქმედება**. კვლავ ამოქმედდება ფუნქცია **ავირჩიოთ_მიმართულება_გადავდგათ_ნაბიჯი**. რახან რულს მინიჭებული აქვს მნიშვნელობა მარჯვნივ, გამოითვლება ხ, რომელიც დხ-ით მეტია წინაზე. x-ის ახალი მნიშვნელობის შესაბამისად *პროცედურა დამჭერი_დავაყენოთ_ადგილზე* გადაადგილებს “დამჭერს” ერთი ნაბიჯით მარჯვნივ.

ეს არის “დამჭერის მთელი “მექანიკა”.

კუბიკები

კუბიკმა უნდა შეასრულოს შემდეგი მოქმედებები:

- პირველ რიგში შეამოწმოს დაიჭირე თუ არა. თუ დაიჭირეს მაშინ გავიდეს თამაშიდან და შეამციროს დაუჭერელი კუბიკების მთვლელი ერთით.
- შეამოწმოს დაეჯახა თუ არა კიდეს და თუ დაეჯახა შეცვალოს მოძრაობის ტრაექტორია “რიკოშეტის” პრინციპით.

- თუ არ მოხდა არცერთი და არც მეორე გადაადგილდეს ერთი ნაბიჯით იმავე მიმართულებით, საითაც მოძრაობდა.

ტაიმერმა კუბიკებთან მიმართებაში უნდა შეასრულოს შემდეგი მოქმედებები:

- “გაადვიდოს” რიგრიგობით ობიექტები კუბიკი(10, კუბიკი(2) კუბიკი(10) და აიძულოს მათ შეასრულოს თავისი სამუშაო.
- თუ ყველა კუბიკი დაჭერილია გააჩეროს თამაში.

ლილაკზე “თავიდან დანყება” დაჭერისას (კუბიკებთან მიმართებაში) კი უნდა შესრულდეს შემდეგი მოქმედებები:

- დაუჭერელი კუბიკების მთვლელის მნიშვნელობა გაუტოლოს 0-ს.
- აიძულოს ყველა კუბიკი დაუბრუნდეს საწყის მნიშვნელობას.
- დაინყოს თამაში.

კუბიკების გამოსახულების მისაღებად შექმნილია მართვის ელემენტის **PictureBox** მასივი. შევქმნით კლასში. გამოცხადებით კი გამოვაცხადებთ ფორმის კოდში.

ახლა კი დანვრილებით ვისაუბროთ კუბიკის კლასზე.

თუ თქვენ კარგად გაერკვიეთ “დამჭერის” კლასში, “კუბიკების” კლასში გარკვევა არ გაგიჭირდებათ.

როგორც პირველ შემთხვევაში აქაც არის ორი მეთოდი **საწყისი_მდგომარეობა და მოქმედება.**

მეთოდი მოქმედება საზღვრავს თუ რა უნდა გააკეთოს კუბიკმა მოძრაობისას.

მან უნდა იცოდეს დაიჭირეს თუ არა და დროა თუ არა უკუიქცეს კიდიდან. ამას ეთმობა პროცედურა მოქმედების ოთხიდან ორი სტრიქონი

მესამე სტრიქონი გამოითვლის x და y კოორდინატებს. მეოთხე სტრიქონი კი გადაადგილებს კუბიკს გამოთვლილ კოორდინატებზე.

ახლა კი ვნახოთ თუ რა ხდება ლილაკზე “თავიდან დაწყება” დაჭერისას.

ამოქმედება ყოველი კუბიკის პროცედურა საწყისი მდგომარეობა.

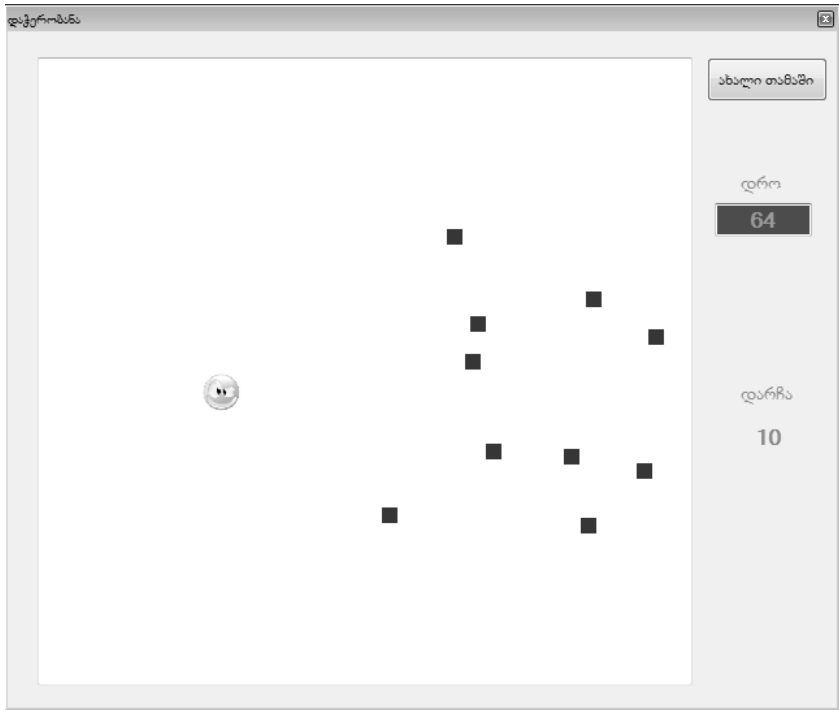
რადგან Rnd წარმოქმნის შემთხვევით რიცხვებს 0 დან 1-მდე, მარტივად შეგვიძლია დავინახოთ, რომ როგორც dx ასევე dy წარმოადგენენ შემთხვევით რიცხვებს $-1,8$ დან $1,8$ დიაპაზონში. ეს საკმარისია იმისათვის რომ კუბიკი გაფრინდეს დიდი შემთხვევითი სიჩქარით შემთხვევითი მიმართულებით.

დანარჩენში ვფიქრობთ იოლად გაერკვევით, მხოლოდ შევჩერდებით პროცედურაზე რიკოშეტი.

აქ თუ კუბიკი ჰორიზონტალურ კიდესთანაა ვიყენებთ ერთადერთ ოპერატორს $dy = -dy$ ანუ ნაბიჯის ვერტიკალურ მდგენელს ვუცვლით ნიშანს. ანალოგიურად $dx = -dx$ თუ კუბიკი ვერტიკალურ კიდესთანაა.

მიაქციეთ ყურადღება რომ კუბიკი გამოგვივიდა “ბრონირებული” ანუ მთელი მექანიკა მასშია. ის არა მარტო მართავს თავის გამოსახულებას ეკრანზე არამედ მას ქმნის კიდეც. სწორედ ეს წარმოადგენს ინკაფსულაციის პრინციპს.

ახლა კი შეგვიძლია გავუშვათ პროექტი და დავინყოთ თამაში.



მონაცემთა ბაზები



უმრავლეს შემთხვევებში მონაცემთა ბაზა ეს სარის ფაილი სადაც შენახულია ერთი ან რამოდენიმე ცხრილი. მა: ბიბლიოთეკის წიგნების მონაცემთა ბაზა, უნივერსიტეტის სტუდენტთა მონაცემთა ბაზა, სიმღერების მონაცემთა ბაზა და სხვ.

ცხრილების სვეტებს ეწოდებათ ველები, სტრიქონებს- ჩანაწერები. სვეტები შეიძლება იყოს ტექსტური, რიცხვითი, თარიღის და დროის ტიპის, შეიცავდნენ ობიექტებს მაგ: სურათებს, ხმას, ვიდეოს). ჩანაწერების რაოდენობა რეალურ მონაცემთა ბაზებში აღწევს რამოდენიმე ათასს.

მონაცემთა ბაზების მართვა რომ მოსახერხებელი იყოს შექმნილია სპეციალური პროგრამები—მონაცემთა ბაზების მართვის სისტემები. ძირითადი რასაც ეს პროგრამები აკეთებს მონაცემთა ბაზებში საჭირო ინფორმაციის მოძიებაა. ასევე აუცილებელია, რომ მოძიებული ინფორმაცია იყოს სორტირებული. ასეთი პროგრამები ასევე საშუალებას გვაძლევს დავამატოთ ჩანაწერი მონაცემთა ბაზაში, ნავშალოთ არასასურველი ინფორმაცია და სხვ.

რატომ არის საჭირო ერთ მონაცემთა ბაზაში რამოდენიმე ცხრილის არსებობა? ავიღოთ მაგალითად მონაცემთა ბაზა “ქართული კალათბურთი”. გარდა კალათბურთელთა ცხრილისა უნდა იყოს მწვრთნელთა ცხრილი, დარბაზების ცხრილი,

თამაშების შედეგების ცხრილი და სხვ. აზრი აქვს ყველა ამ ცხრილის არსებობას ერთ ბაზაში. ყველა ინფორმაციის ერთ ცხრილში არსებობა კი მოუხერხებელია.

იმისათვის რომ ერთდროულად მივმართოთ რამოდენიმე ცხრილს ამისათვის ეს ცხრილები ერთმანეთთან უნდა იყოს დაკავშირებული. ამ წიგნში ჩვენ საუბარი არ გვექნება ურთიერთდაკავშირებულ ცხრილებზე, საქმე გვექნება მხოლოდ ერთი ცხრილისაგან შემდგარ მონაცემთა ბაზასთან.

ერთ-ერთი ყველაზე ცნობილი მონაცემთა ბაზების მართვის სისტემაა Microsoft Access რომელიც შედის Microsoft Office-ის შემადგენლობაში. ჩვენს შემდგომ პროექტში გამოვიყენებთ სწორედ Microsoft Access-ის ბაზას. Visual Basic იყენებს მონაცემთა ბაზებთან მუშაობის მძლავრ საშუალებებს—ADO.NET ტექნოლოგიას.

მონაცემთა ბაზა “სტუდენტები”

თავდაპირველად მონაცემთა ბაზა უნდა შევქმნათ Microsoft Access-ში და შემდეგ დაფუკავშიროთ ის ჩვენს პროექტს.

გავხსნათ Microsoft Access (ჩვენ გამოვიყენეთ Microsoft Access 2007) და ავირჩიოთ Blank database, მარჯვენა მხარეს მონაცემთა ბაზას დავარქვათ სახელი students.mdb და დავაჭიროთ Greate.



გაიხსნება ფანჯარა სადაც ავირჩევთ view->Design View. გახსნილ ფანჯარაში კი ცხრილს მივანიჭებთ სახელს ან დავტოვებთ სახელს Table1.

Table1		Field Name	Data Type
?	ID		AutoNumber
	სახელი		Text
	გვარი		Text
	ფაკულტეტი		Text
	კურსი		Text
	ჯგუფი		Text
	ტელეფონი		Text
	მისამართი		Text

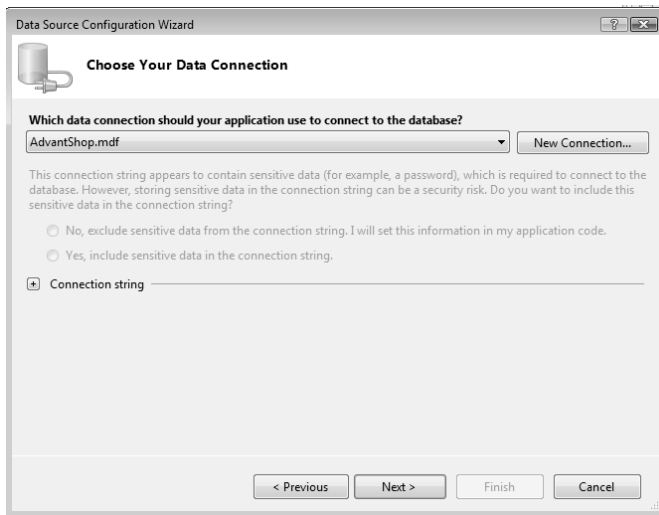
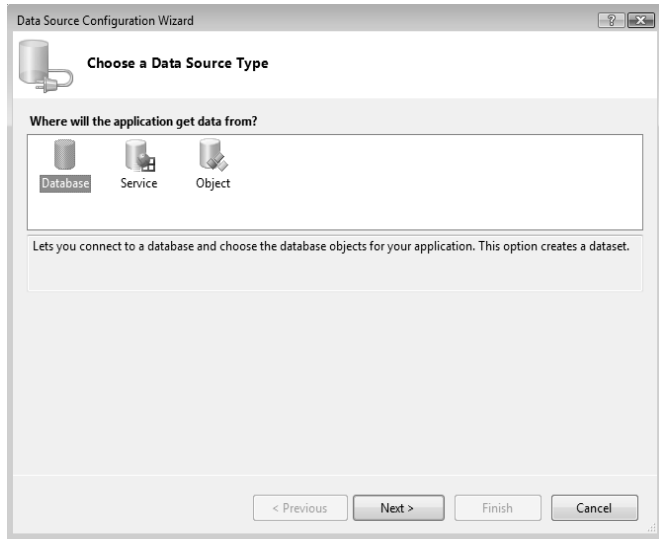
გახსნილი გვერდის **Field name** სვეტში უნდა ჩავწეროთ სტრუქტურების მონაცემთა ბაზის პარამეტრები: სახელი, გვარი, მისამართი, კურსი, ჯგუფი და ტელეფონი.

ამ მომენტისთვის **access**-ში მონაცემთა ბაზა შექმნილია, შევინახოთ და დავხუროთ პროგრამა.

გადავიდეთ **Visual Studio**-ს გარემოში შევიდეთ **File**-ში და ავირჩიოთ **New Project**, გახსნილ ფანჯარაში კი **Windows Forms application**, გაიხსნება ცარიელი ფორმა.

შემდეგ გადავინაცვლოთ მენიუთა სტრიქონში შევიდეთ **Data**-ში და ავირჩიოთ **Show Data Sources**. **Data sources** ფანჯარაში დავაჭიროთ ბრძანებას **Add New Data Sources**.

გაიხსნება ფანჯარა **Data Source Configuration Wizard** სადაც ავირჩევთ **Database**-ს, დავაჭერთ **Next**-ს. სურათი4



შემდეგ დავაჭერთ New connection-ს.

Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
Microsoft SQL Server (SqlClient) Change...

Server name:
[] Refresh

Log on to the server

Use Windows Authentication
 Use SQL Server Authentication

User name: []
Password: []
 Save my password

Connect to a database

Select or enter a database name:
[]

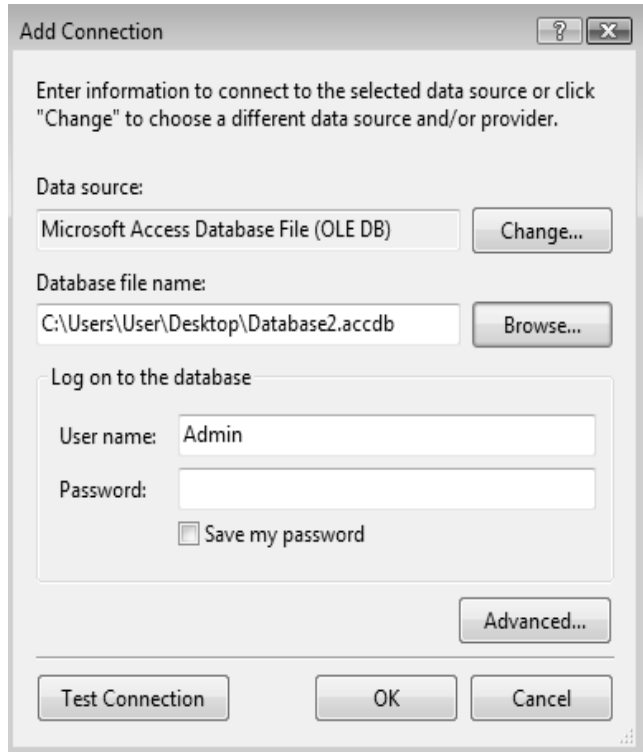
Attach a database file:
[] Browse...

Logical name: []

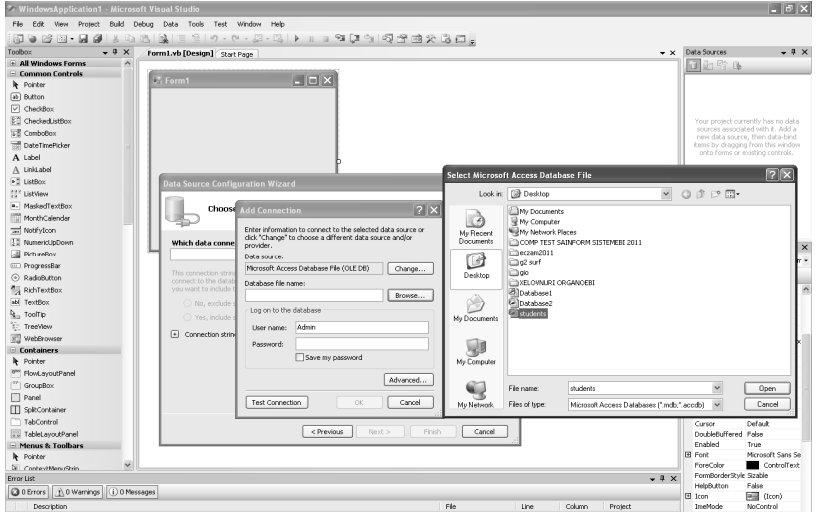
Advanced...

Test Connection OK Cancel

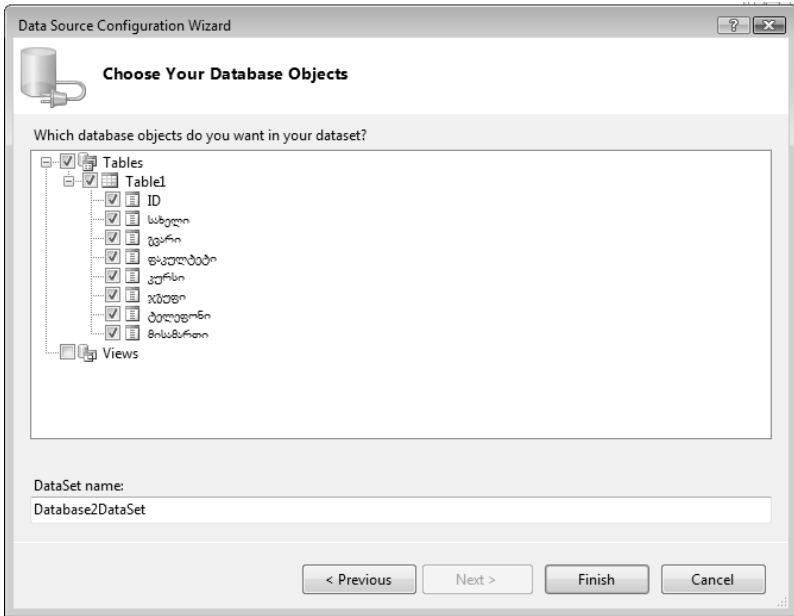
გამოტანილ ფანჯარაში დავაჭერთ **change-ს** და **Change Data source**-ფანჯარაში ავირჩევთ **Microsoft Access Database File-ს**.



გაიხსნება ფანჯარა სადაც დავაჭერთ **Browse** ღილაკს და ავირჩევთ ჩვენს მიერ შექმნილ მონაცემთა ბაზის ფაილს students.mdb.



დავაჭერთ ლილას **Test Connection**, თუ შეერთება მოხერხდა მივიღებთ შესაბამის შეტყობინებას. რომელსაც უნდა დავეთანხმოთ, შემდეგ გამოვა დიალოგური ფანჯარა **Data source Configuration Wizard**. სადაც უნდა მოვნიშნოთ ჩვენი ცხრილი და მისი სვეტები.



შემდეგ დავაჭერთ **Next**-ს და ბოლოს **Finish**-ს.

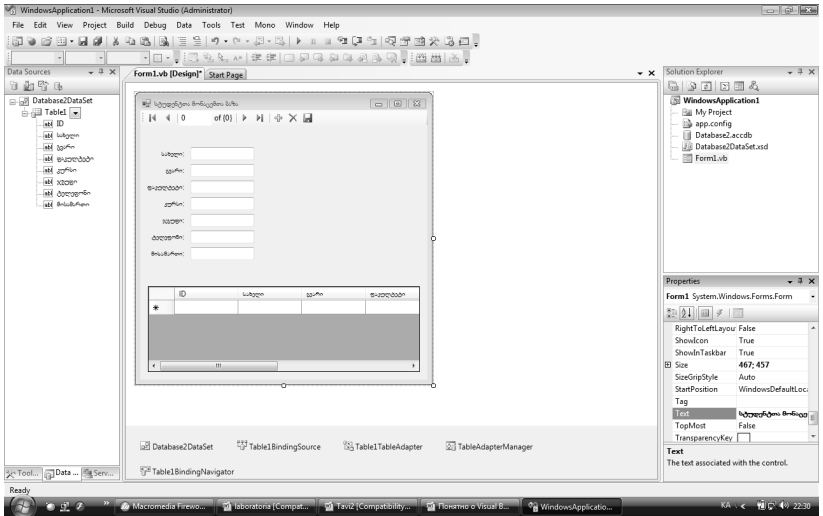
ამ მომენტისთვის ჩვენი დაკავშირება მონაცემთა ბაზასთან დასრულებულია. მარჯნივ **Data source** ფანჯარაში გამოჩნდება ჩვენს მიერ შექმნილი მონაცემთა ბაზის ცხრილი (თუ ვერ ვხედავთ **Data source** ფანჯარას, მენიუთა სტრიქონში შევიდეთ **Data**-ში და მივცეთ ბრძანება **Show Data Source**).

მაუსის ლილაკიდ დაჭერით (აუშვებლად), უნდა გადმოვიტანოთ მისი სტრიქონები ფორმაში. ფორმას ზედა ნაწილში ავტომატურად დაემატება ელემენტი **BindingNavigator**, რომელიც მონაცემთა ბაზაში ნავიგაციის საშუალებას გვაძლევს (გადაადგილება ბაზაში, ახალი ჩანაწერის დამატება, წაშლა, შენახვა).

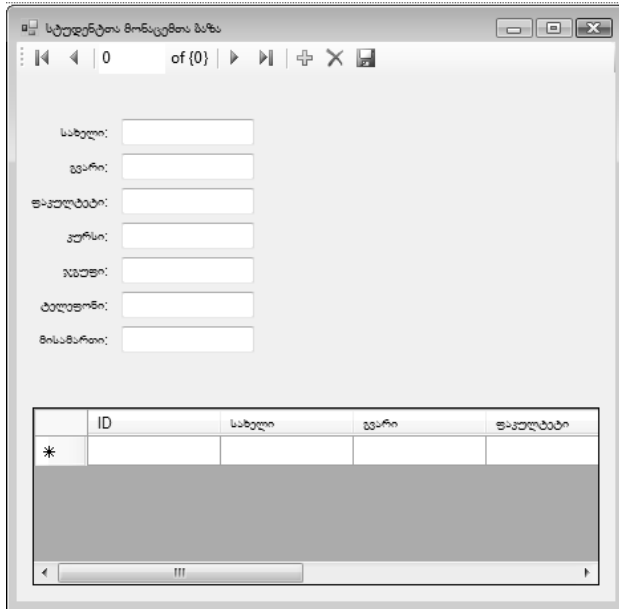
ელემენტთა პანელიდან გადმოვიტანოთ ფორმაზე მართვის ელემენტი **DataGridView**. მოვნიშნოთ ის და დავაჭიროთ

პატარა სამკუთხედს მის მარჯვენა ზედა კიდეზე. გახსნილ ფანჯარაში, გრაფაში **Choose data Source** დაუკავშიროთ მას ჩვენი მონაცემთა ბაზა.

ელემენტი **DataGridView** საშუალებას გვაძლევს ჩვენი მონაცემთა ბაზა წამოდგენილი იყოს ასევე ცხრილის სახით. ასევე შესაძლებელია ჩანანერის დამატება პირდაპირ ამ ელემენტიდან.



გავუშვათ პროგრამა. შეგვიძლია დავინყოთ ჩანანერების დამატება და მონაცემთა ბაზის გამოყენება.



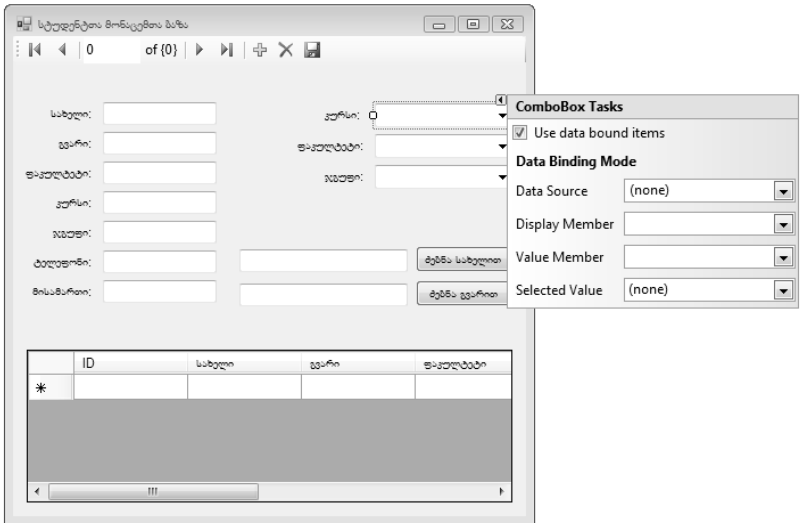
როგორც თქვენ მონაცემთა ბაზის გამოყენების პირველივე ნუთებიდან მიხვდებით, აუცილებელია მას გააჩნდეს მონაცემების სიტყვით ან სხვა პარამეტრით ძებნის ფუნქცია. არსებობს მონაცემთა მოძიების სხვადასხვა მეთოდები, ჩვენ აქ მოვიყვანთ სრული სიტყვით ძებნის მეთოდს.

ასევე მონაცემთა ბაზაში ძებნას **ComboBox** ელემენტების სიით.

წარმოვიდგინოთ, რომ ჩვენს მონაცემთა ბაზაში გვინდა დავათვალიეროთ მხოლოდ ერთი ფაკულტეტის, კურსის ან ჯგუფის სტუდენტთა მონაცემები. ამისათვის უნდა მოვახდინოთ ბაზის შესაბამისი ფილტრაცია.

გადმოვიტანოთ ფორმაზე 3 მართვის ელემენტი **ComboBox**, 2 ელემენტი **TextBox** და 2 ელემენტი **Button**, ასევე 3 ელემენტი **Label**. განალაგეთ ისინი ისე როგორც ნაჩვენებია სურათზე. თვისებათა ფანჯარაში შევცვალოთ **Label** ელემენტების თვისებება **Text** (კურსი, ფაკულტეტი, ჯგუფი). ასევე გავაკეთოთ წარწერა ელემენტებზე **Button** (ძებნა სახელით, ძებნა გვარით).

დავაჭიროთ პატარა სამკუთხედს იმ **ComboBox**-ის მარჯვენა ზედა კიდეზე რომელმაც უნდა გაფილტროს ბაზა “კურსის მიხედვით”. გაიხსნება მენიუ **ComboBox Tasks**. მის გრაფაში **Data Source** ჩამოვშალოთ მენიუ და მოვნიშნოთ ჩვენი ცხრილი (თუ ცხრილის სახელი არ შეგვიცვლია **Table1BindingSource**). შემდეგ გრაფაში **Display Member** მივუთითოთ ვეილს სახელი “კურსი”.



ასევე დავუკავშიროთ მონაცემთა ბაზის შესაბამის ველებს დანარჩენი ორი **ComboBox**.

უკვე შესაძლებელია ბაზის ფილტრაცია კურსით, ფაკულტეტით და ჯგუფით.

ახლა კი გადავიდეთ ძებნაზე სიტყვით. ძებნა მოხდება სტუდენტთა სახელით და გვარით. უნდა მოხდეს სტუდენტის მოძებნა იმ სახელით და გვარით, რომელსაც ჩავწერთ ტექსტურ ბლოკებში.

ორჯერ დავაკლიკოთ ღილაკზე “ძებნა სახელით” და ჩავწეროთ შემდეგი პროგრამული კოდი:

```
Private Sub Button1_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button1.Click
```

```
    Table1BindingSource.Filter = "[სახელი]=''" & _  
    TextBox1.Text & ""
```

```
End Sub
```

თავდაპირველად დაინერება ცხრილის სახელი. ფრჩხილებში ჩაინერება მონაცემთა ბაზის იმ ველის დასახელება, სადაც უნდა მოვახდინოთ ძებნა. ბოლოს კი მითითებულია ის ტექსტური ბლოკი სადაც ჩაინერება საძებნი სიტყვა.

ორჯერ დავაკლიკოთ ლილაკზე “ძებნა გვარი” და ჩავწეროთ შემდეგი პროგრამული კოდი:

```
Private Sub Button2_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button2.Click
```

```
    Table1BindingSource.Filter = "[გვარი]=''" & _  
    Me.TextBox2.Text & ""
```

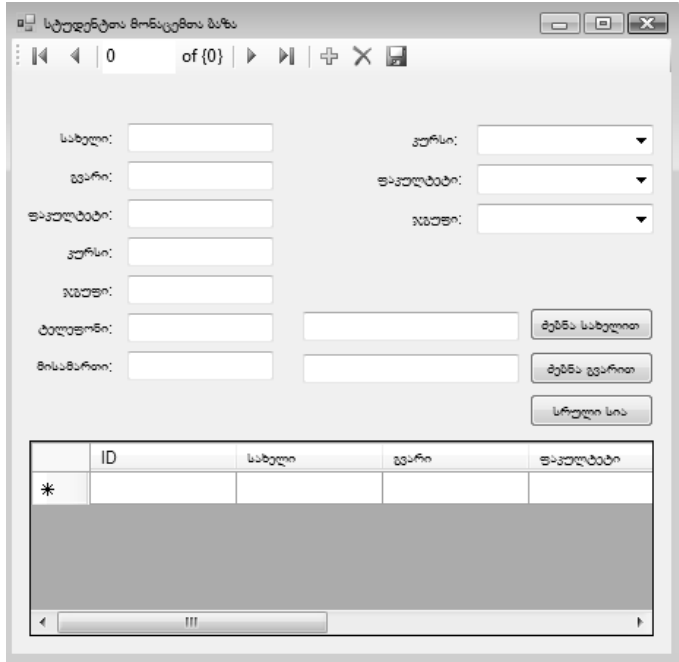
```
End Sub
```

მონაცემთა ბაზის გაფილტვრის შემდეგ საჭიროა რომ კვლავ მივიღოთ სტუდენტთა სრული სია. ამისათვის დავამატოთ კიდევ ერთი ლილაკი წარწერით “სრული სია” და ჩავწეროთ მასში შემდეგი პროგრამული კოდი:

```
Private Sub Button3_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) Handles _  
Button3.Click
```

```
    Table1BindingSource.Filter = ""
```

```
End Sub
```

ამ ეტაპზე ჩვენი მარტივი მონაცემთა ბაზა ფილტრაციის ფუნქციით მიზანშეწონილია. გავუშვათ პროგრამა. შევიყვანოთ სტუდენტთა მონაცემები. მოვახდინოთ ნავიგაცია მონაცემთა ბაზაში. მოვახდინოთ ჩვენთვის სასურველი ფილტრაცია.

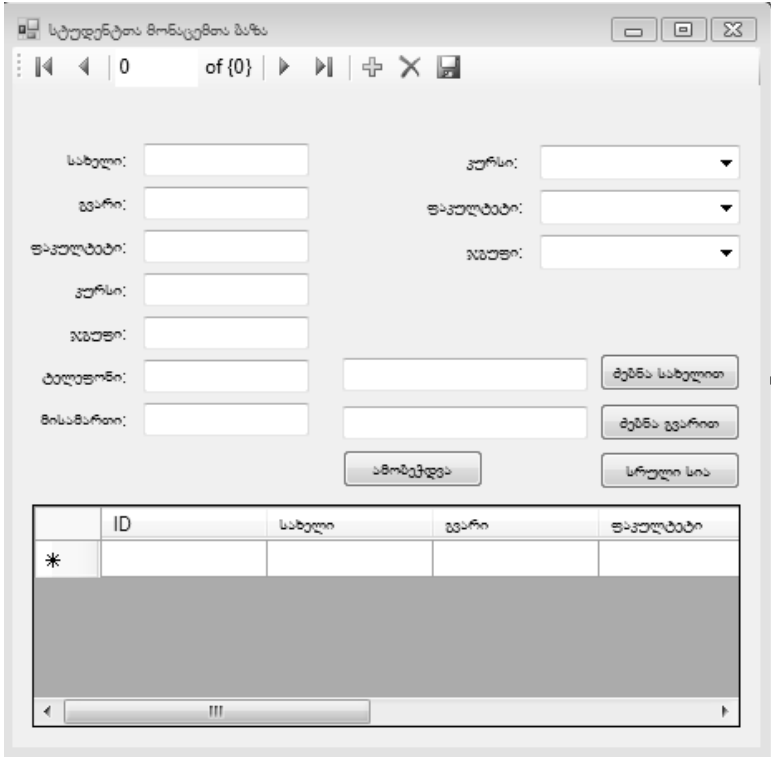


ძალიან ხშირად, განსაკუთრებით ბიზნეს-პროექტებში აუცილებელია ამონაწერების გაკეთება მონაცემთა ბაზებიდან, მისი შენახვა, ექსპორტი და პრინტერზე ამობეჭდვა. **Visual Basic 2008**-ს გააჩნია ამის განსახორციელებლად მძლავრი აპარატი.

შემდეგი პროექტისათვის დაგვჭირდება მონაცემთა ბაზა, რომელიც წინა პროექტში შევქმენით (სტუდენტთა მონაცემთა ბაზა). წარმოვიდგინოთ რომ გვჭირდება სტუდენტთა ბაზა ინახებოდეს არა მხოლოდ ელექტრონული სახით კომპიუტერის მეხსიერებაში, არამედ დოკუმენტის სახით სეიფშიც ან პერიოდულად გვჭირდება ბაზიდან ამონაწერის გაკეთება. ამისათვის საჭიროა რომ პროგრამას გააცნდეს სესაბამისი ფუნქცია.

Visual Basic 2008-ში ამის განსახორციელებლად შესაძლებელია გამოვიყენოთ მართვის ელემენტი **MicrosoftReportViewer** ან **CristalReportViewer**. ჩვენ ჩვენს პროექტში გამოვიყენებთ **MicrosoftReportViewer**-ს.

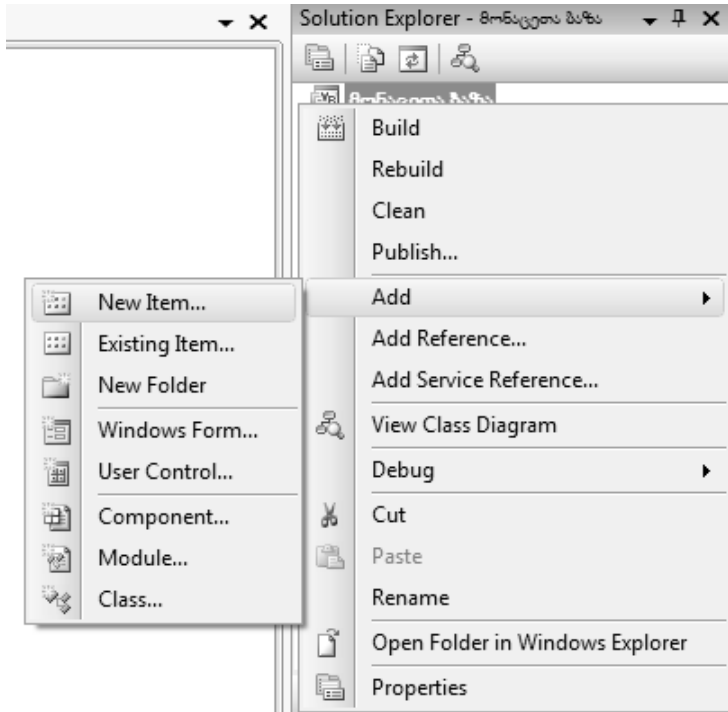
გავხსნათ პროექტი მონაცემთა ბაზა “სტუდენტები”. დავამატოთ ფორმაზე ერთი ლილაკი წარწერით ამობეჭდვა.



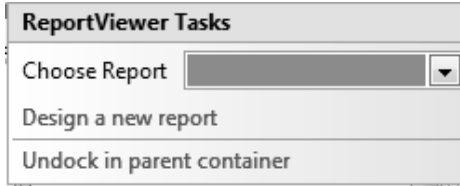
ახლა უნდა დავამატოთ ჩვენს პროექტში ახალი ფორმა, რომელზეც განთავსდება ბაზის ამოსაბეჭდი ვერსია. ამოსაბეჭდ ვერსიას ჩვენ შეგვიძლია მივცეთ ნებისმიერი ჩვენთვის სასურველი სახე.

Solution Explorer-ში მოვნიშნოთ პროექტის სახელი და დავაჭიროთ მაუსის მარჯვენა ლილაკს. გამოჩნდება კონტექსტური მენიუ სადაც ავირჩიოთ **Add** შემდეგ **New Item**.

გახსნილ ფანჯარაში **Add New Item** ავირჩიოთ **Window Form** და დავაჭიროთ ლილაკს **Add**. **Solution Explorer**-ში გამოჩნდება ახალი ფორმა **Form2**. შევცვალოთ მისი თვისება **Text** და დავარქვათ "ამობეჭდვა". წარწერა გამოჩნდება ფორმის თავზე.

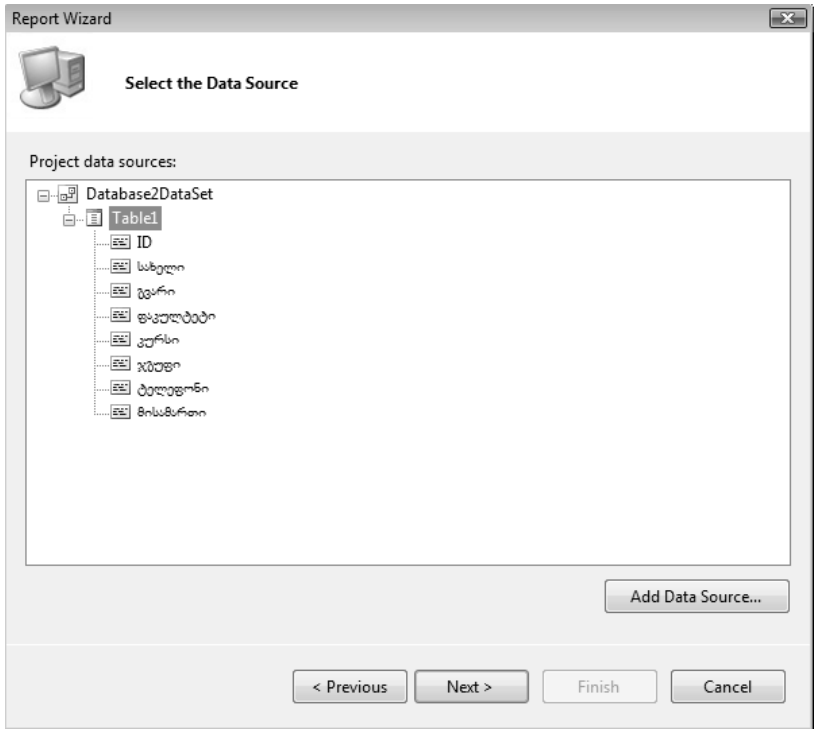


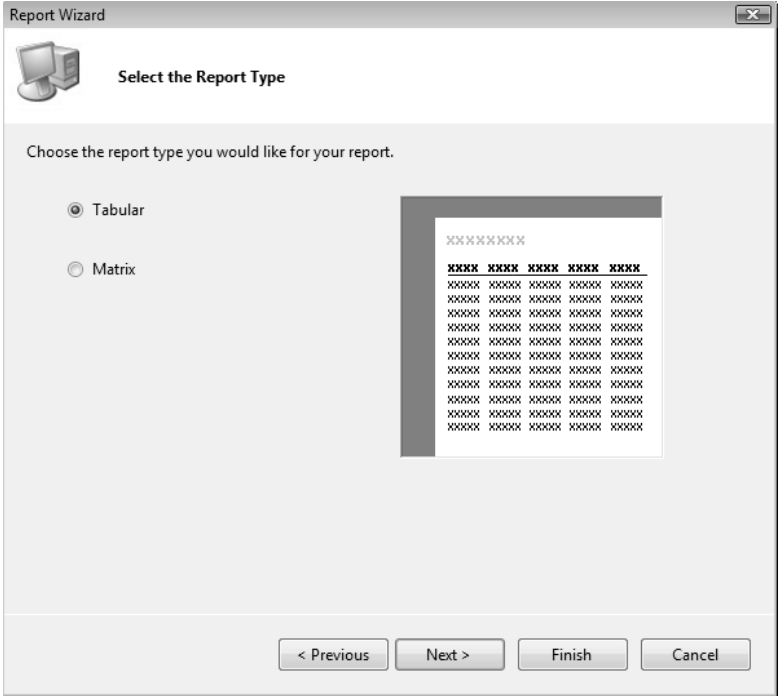
ელემენტთა პანელზე მოვნახოთ ელემენტი **MicrosoftReportViewer** და გადმოვიტანოთ ის ფორმაზე. ფორმის ზომები გავზარდოთ (თვისება **Size 600,700**). დავაჭიროთ პატარა სამკუთხედს **MicrosoftReportViewer**-ის მარჯვენა ზედა კიდეზე და გახსნილ მენიუში **ReportViewerTasks** დავაჭიროთ ბრძანებას **Dock in parent container**. ელემენტის ზომები გაიზრდება და ის მთელ ფორმას მოიცავს.



შემდეგ კვლავ დავაჭიროთ სამკუთხედს და ბრძანებას **Design a new report**. გაიხსნება ფანჯარა **Report Wizard**. დავაჭიროთ ღილაკს **Next**. შემდეგ ფანჯარაში გამოჩნდება ჩვენი მონაცემთა ბაზა. დავაჭიროთ ღილაკს **Next**.

გახსნილ ფანჯარაში უნდა მივუთითოთ რეპორტში მონაცემების განთავსების ტიპი. დავტოვოთ მონიშვნა **Tabular**. დავაჭიროთ ღილაკს **Next**.

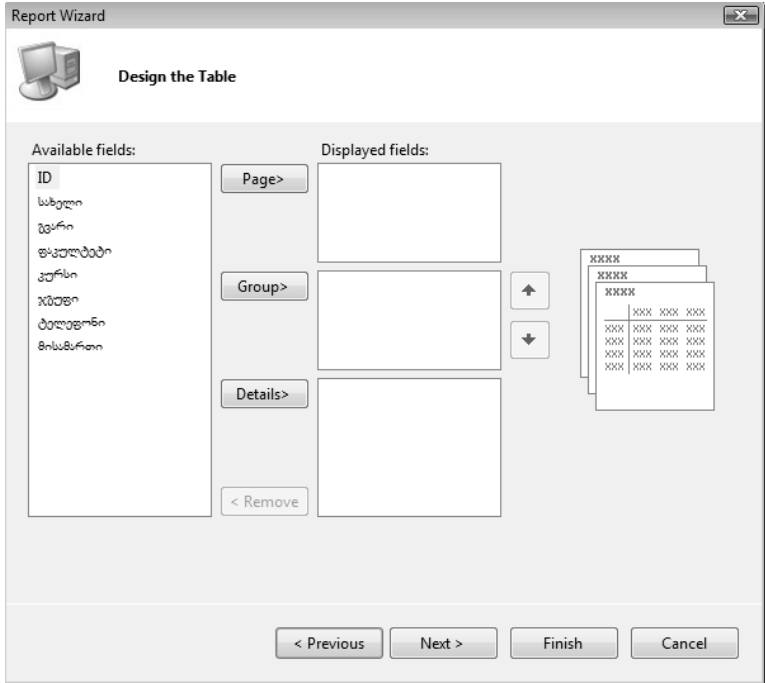




შემდეგ ფანჯარაში ჩვენ შეგვიძლია მივუთითოთ თუ რა მონაცემების ამობეჭდვა გვსურს. თუ გვინდა ამოვბეჭდოთ სტუდენტთა სრული სია თავისი მონაცემებით (ჩვენს შემთხვევაში ასეც მოვიქცეთ) ყველა ველი უნდა გადმოვიტანოთ გრაფაში **Details**, ველის მონიშვნით და ღილაკზე **Details** დაჭერით.

თუ გვსურს სათითაოდ ყოველი სტუდენტის მონაცემების ამობეჭდვა ველი **ID** უნდა გადმოვიტანოთ გრაფაში **Page**.

ჩვენს შემთხვევაში ყველა ველი უნდა გადმოვიტანოთ გრაფაში **Details**.



დავაჭიროთ ლილაკს **Next**. შემდეგ გამოსულ ფანჯარაში კვლავ დავაჭიროთ ლილაკს **Next**.

შემდეგ გამოსულ ფანჯარაში ავირჩიოთ ცხრილის სასურველი დიზაინი და დავაჭიროთ ლილაკს **Finish**. ამ ეტაპზე ჩვენი რეპორტი შექმნილია. ის გამოჩნდება **Solution Explorer**-ში.

ახლა ჩვენი რეპორტი უნდა დავუკავშიროთ მართვის ელემენტს **MicrosoftReportViewer**. ამისათვის კვლავ დავაჭიროთ პატარა სამკუთხედს **MicrosoftReportViewer**-ის მარჯვენა ზედა კიდეზე და გახსნილ მენიუში **ReportViewerTasks**, **Choose report-**ის მარჯვნივ ჩამოვშალოთ სია და მივუთითოთ ჩვენი რეპორტი.

ახლა დავამუშავოთ ჩვენი რეპორტი. ამისათვის **Solution Explorer**-ში ორჯერ დავაკლიკოთ რეპორტის სახელზე.

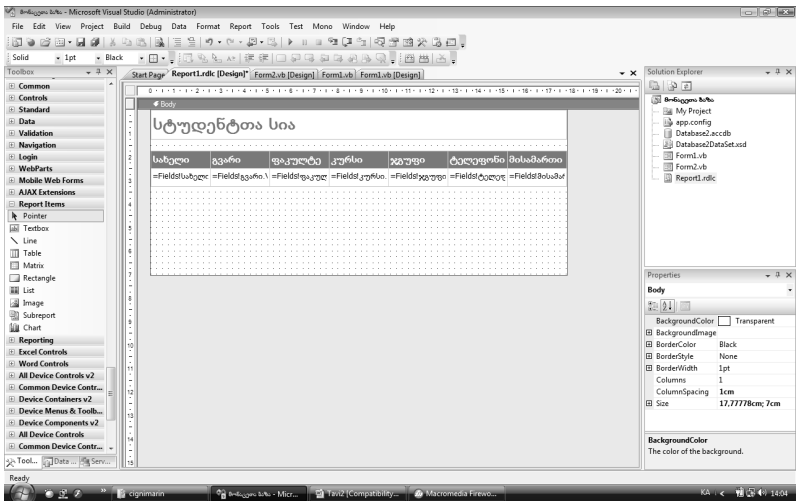
ელემენტთა პანელზე ჩამოვშალოთ **Report Items**. ამ ელემენტებით შევვიძლია შევცვალოთ რეპორტის ვიზუალური

სახე. დავამატოთ გრაფიკა, სურათი, ტექსტი და სხვ. ასევე მრავალი სხვადასხვა ცვლილებების შესაძლებლობა. ჩვენ ყველა მათგანს ამ ნივთში ვერ შევხებით. დამოუკიდებელი ექსპერიმენტებით შესაძლებელია მათში კარგად გარკვევა.

ჩვენ უბრალოდ მოვნიშნოთ წარწერა რეპორტის თავზე **Report1** და შევცვალოთ ის, წარწერით “სტუდენტთა სია”.

ფონტის ზომის შეცვლა შეგვიძლია თვისებათა ფანჯარიდან.

ცხრილის ზომების შეცვლა შეგვიძლია მისი კიდეების მონიშნით და მაუსის მარცხენა ღილაკის აუშვებლად შესაბამისი ზომის მიცემით. ზომების შეცვლა ასევე შესაძლებელია თვისებათა ფანჯარიდან.

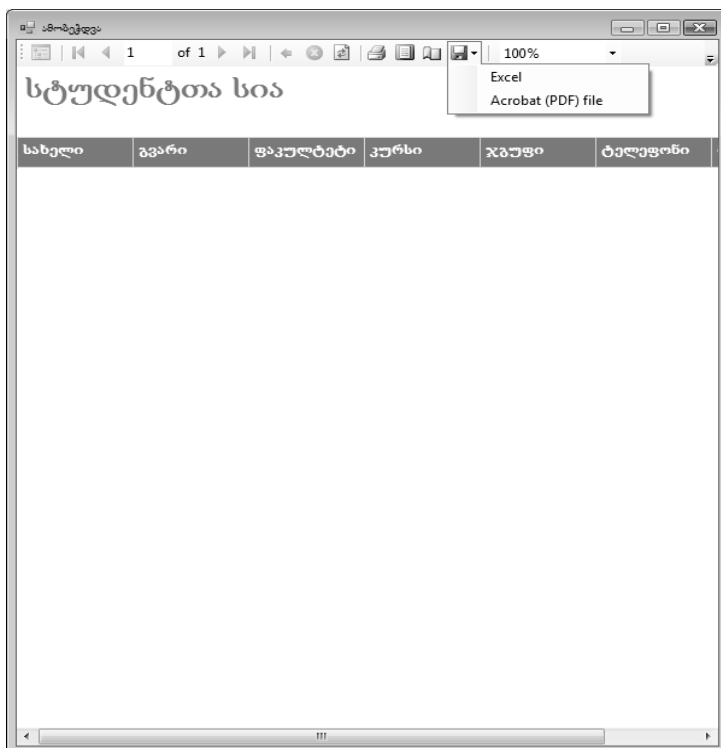


დარჩა ერთი პატარა კოდის ჩანწერა. საჭიროა რომ ჩვენს მიერ შექმნილი რეპორტი გამოჩნდეს ღილაკზე “ამობეჭდვა” დაჭერისას. ამისათვის ორჯერ დავაკლიკოთ პირველ ფორმაზე ღილაკზე “ამობეჭდვა” და ჩავწეროთ შემდეგი კოდი:

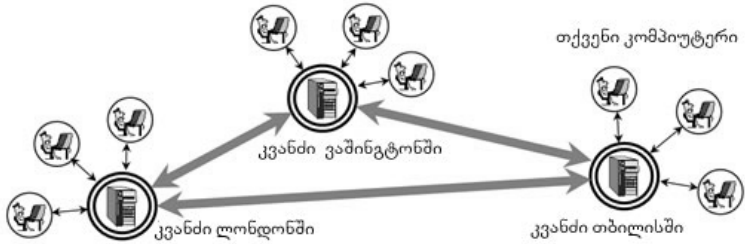
Form2.Show()

პროექტზე მუშაობა ამ ეტაპზე დასრულებულია. გავუშვათ ის და შევიტანოთ მონაცემთა ბაზაში სტუდენტთა მონაცემები. მონაცემების შეტანის შემდეგ აუცილებელია მათი შენახვა ლილაკზე **Save** დაჭერით.

შემდეგ დავაჭიროთ ლილაკზე “ამობეჭდვა”. გაიხსნება მეორე ფორმა რეპორტით. რეპორტის თავზე განთავსებულია ლილაკები, რომლებიც საშუალებას გვაძლევს მოვახდინოთ ნავიგაცია გვერდებს შორის (**Next Page, Last Page**), გვერდების ამობეჭდვა პრინტერზე (**Print**), ამობეჭდვის წინ მათი წინასწარი დათვალიერება (**Print Layout**), გვერდების ექსპორტი **Excel** და **Acrobat**-ში (**Export**).



Visual Basic და ინტერნეტი



ინტერნეტი – ეს არის მილიონობით კომპიუტერი მთელს მსოფლიოში, რომლებიც შეერთებულია სატელეფონო ან სხვა უფრო სწრაფი ინფორმაციის მიმოცვლის საშუალებებით.

თუ თქვენ გინდათ მიუერთოთ თქვენი სახლის კომპიუტერი ინტერნეტს ამას სეძლებთ მხოლოდ რომელიმე კვანძთან (სერვერთან) მიერთებით. ასეთი კვანძის მფლობელია რომელიმე კომპანია, რომელსაც ეწოდება პროვაიდერი, ის უზრუნველყოფს თქვენი კომპიუტერის მიერთებას ინტერნეტის ქსელთან, თქვენ კი ამაში გარკვეულ საფასურს იხდით.

ქვენ იცით რომ ინტერნეტში სეგიძლიათ შეხვიდეთ უამრავ ვებ საიტზე, დაათვალიეროთ უამრავი ვებ-გვერდი, უყუროთ ფილმებს, გადმონეროტ პროგრამები და ასშ. სად ინახება ეს ინფორმაცია და საიდან ხვდება ის თქვენს ბრაუზერში?

ყველა ნებ-გვერდი განთავსებულია ნებ-სერვერებზე, მათ ქმნიას სახლის კომპიუტერებზე და სემდეგ ანთავსებენ სერვერებზე. დღეისათვის თქვენ მარტივად სეგიძლიათ შექმნათ თქვენი ვებ გვერდი, განათავსოთ ის რომელიმე სერვერზე და ანახოთ მთელ მსოფლიოს

წარმოვიდგინოთ თუ რა ხდება მაშინ, როცა ჩვენ კომპიუტერის ეკრანზე ვხედავთ ვებ-გვერდს, რომელიც განთავსებულია სორეულ სერვერზე, მაგალითად ავსტრალიაში. ყოველ გვერდს აქვს თავისი უნიკალური მისამართი, თქვენ

ანვდით თქვენს მიერ სასურველი გვერდის მისამართს თქვენს სერვერს, ის კი უკავშირდება ამ მისამართით ავსტრალიაში არსებულ აღნიშნულ სერვერს იქიდან სასურველი ინფორმაცია ექვზქვენება ჩვენს სერვერს, იქიდან კი ჩვენს კომპიუტერს.

თქვენ იცით, რომ ვებ გვერდები სეიცავენ ფერებს, გრაფიკას, ანიმაციას და ასშ. ეს ინფორმაცია საჭიროებს დიდ მეხსიერებას და მისი გადაგზავნისას ასეც დიდ მანძილებზე იქნებოდა შესაბამისი პრობლემები. როგორ არის ეს პრობლემა გადაწყვეტილი ინტერნეტში?

ნარმოიდგინეთ ასეთი სიტუაცია ერთი მხატვარს უნდა ანახოს მეორეს თავისი ნახატი, მაგრამ ნახატი დიდია დ მისი გაგზავნა ფოსტით შეუძლებელია. ის უგზავნის მას წერილს სადაც არის ინფორმაცია მისი ნახატის შესახებ, ის ეუბნება მას რომ ფურცელის ქვემო ნაწილში დახატოს ბალახი, კონკრეტული ფერის, შემდეგ ცა ღრუბლებით და ასშ. მეორე მხატვარი დახატავს და მიიღებს ნახატის კოპიას, და არა ორიგინალს.

ასეთივე სიტუაცია გვაქვს ინტერნეტშიც. ინტერნეტში ჩვენ გადმოგვეცემა არა ვებ-გვერდი, არამედ მის შესახებ ინფორმაცია: ფერი, ფონტი, კოდირებული სურათები, ვიდეო, ხმა დას ხვ. ჩვენი ბრაუზერი კი აღადგენს სურათს (ცნობილი ბრაუზერებია: **Internet Explorer, Opera, Mozilla** დას სხვ.).

ვებ-გვერდის შესახებ ინფორმაცია კი გადაეცემა სპეციალურ ენაზე რომელსაც ეწოდება **HTML**.

საკუთარი ბრაუზერი

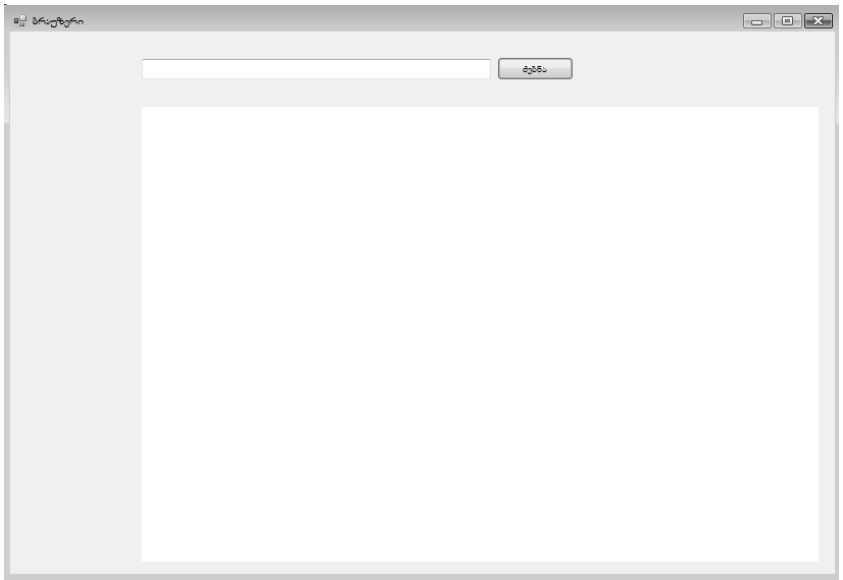
ჩვენ შეგვიძლია საკუთარ პროექტში შევექმნათ საკუთარი ბრაუზერი, რომელიც მოგვცემს საშუალებას დავათვალიეროთ ვებ-გვერდები პირდაპირ ჩვენი პროექტიდან. ფორმაზე გადმოიტანეთ მართვის ელემენტი **WebBrowser**. თუ *Toolbox*-ში მას ვერ ვხედავთ ჯერ ის უნდა მოვახვედროთ მასში. ამისათვის *Toolbox*-ის ნებისმიერ არეში დავაჭიროთ მაუსის მარჯვენა ლილაკს, შემდეგ ამოვირჩიოთ **Choose Items**, გახსნილ ფანჯარაში

მოვნახოთ ჩვენთვის სასურველი ობიექტი, მოვნიშნოთ ის და დავაჭიროთ OK ღილაკს.

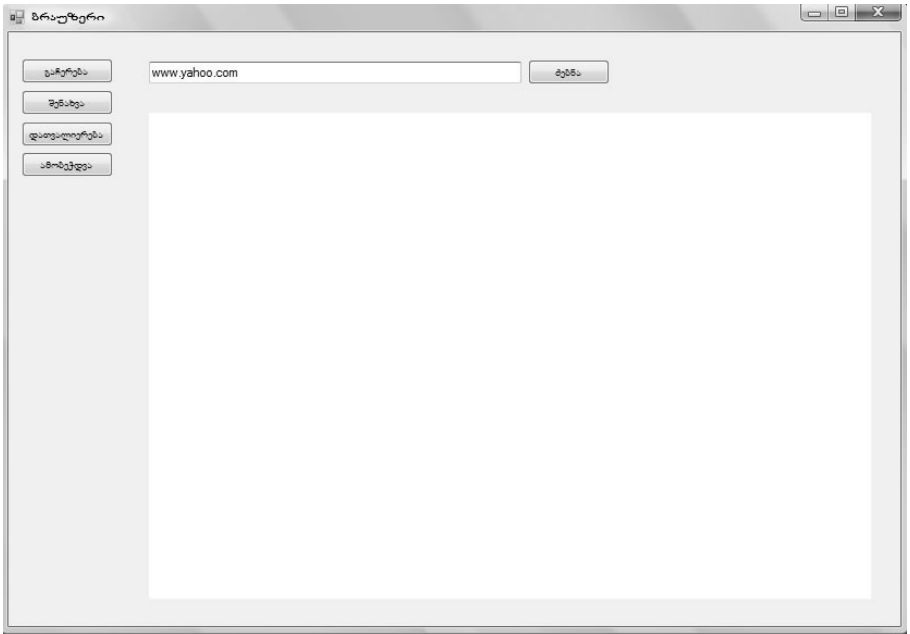
გავადიდოთ **WebBrowser** და მივცეთ მას სასურველი ზომა. შემდეგ ფორმაზე მოვათავსოთ ღილაკი წარწერით “ძებნა” და ტექსტური ბლოკი (**TextBox1**). ღილაკში ჩავწეროთ შემდეგი კოდი:

```
Private Sub Button1_Click(ByVal sender As _  
System.Object, ByVal e As System.EventArgs) _  
Handles Button1.Click  
  
WebBrowser1.Navigate (TextBox1.Text)  
  
End Sub
```

თქვენ შეგიძლიათ შეიყვანოთ სასურველი ვებ-გვერდის მისამართი TextBox1-ში, დაჭიროთ ღილაკს და დაათვალიეროთ ვებ-გვერდები თქვენს პროექტში.



როგორც ხედავთ ჩვენს ბრაუზერს არა აქვს ყველა ღილაკი ინტერნეტში ნავიგაციისათვის. მაგრამ ჩვენ გვაქვს მათი დამატების საშუალება. ამისათვის დავამატოთ ღილაკები ნარწერიით “გაჩერება” “შენახვა”, “დათვალიერება”, “ამოხეჭდვა”.



ღილაკ “გაჩერებაში” ჩავწეროთ შემდეგი კოდი:

```
WebBrowser1.Stop()
```

ღილაკ “შენახვაში” ჩავწეროთ შემდეგი კოდი:

```
WebBrowser1.ShowSaveAsDialog()
```

ლილაკ “დათვალიერებაში” ჩაწეროთ შემდეგი კოდი:

```
WebBrowser1.ShowPrintPreviewDialog
```

ლილაკ “ამობეჭდვაში” ჩაწეროთ შემდეგი კოდი:

```
WebBrowser1.ShowPrintDialog
```

აღნიშნული ლილაკები საშუალებას მოგვცემს შევაჩეროთ ბრაუზერის მიერ ნებ-გვერდის გახსნა, დავათვალიეროთ ვებ-გვერდი პრინტერზე ამობეჭდვამდე. ამოვბეჭდოთ ვებ-გვერდი პრინტერზე და ასევე შევინახოთ ვებ-გვერდი ჩვენს კომპიუტერში.

შესაძლებელია ასევე ბრაუზერებისათვის დამახასიათებელი ნებისმიერი ფუნქციის დამატება, რაზეც ექსპერიმენტები მკითხველისათვის მიგვინდია.

ვებ-გვერდის შექმნა

ვებ-გვერდის შესაქმნელად აუცილებელია HTML ენის ცოდნა, მაგრამ შეიძლება თუ არა აუაროთ მას გვერდი და მაინც შევექმნათ საკმაოდ კარგი ვებ-გვერდები? პასუხი დადებითია, რატემა უნდა HTML-ის ცოდნის გარეშე პროფესიონალი ვებ-დიზაინერი ვერ გახდები, მაგრამ ვებ-გვერდების შექმნა შესაძლებელია. ამისათვის გამოიყენება სპეციალური სამომხმარებლო პროგრამები, რომლებიც HTML კოდს ჩვენს მაგივრად დანერს. ყველაზე ცნობილი პროგრამებია **Adobe Dreamweaver** და **Microsoft Front Page**.

ჩვენს მიერ შექმნილი ვებ-გვერდის დათვალიერება შეგვიძლია ჩვენს კომპიუტერში, როგორ ვაჩვენოთ ის მსოფლიოს? ამისათვის ის უნდა განვათავსოთ რომელიმე სერვერზე. ამ პროცესს ჰოსტინგი ეწოდება. უნდა ვიქირავოთ ვებ-სივრცე რომელიმე სერვერზე (არსებობს ასეთი უფასო სერვერებიც მაგ: www.000webhost.com).

ჰოსტინგის სერვერზე მას მიენიჭება გარკვეული მისამართი, რომელიც არც თუ ისე მარტივი იქნება და მისი დამახსოვრება არა თუ ჩვენი ვებ-გვერდის სტუმრებს, ცვენც კი გაგვიჭირდება. მაშ როგორ მოვიქცეთ?

ჩვენს საიტს უნდა მივანიჭოთ დომენური სახელი (დომენი). საქართველოში **GE** დომენის მიანიჭებას მიმნიჭებელი კომპანია დაუკავშირებს ჩვენს მიერ დარეგისტრირებულ დომენურ სახელს მის მისამართთან სერვერზე და უკვე ამ სახელით შესაძლებელი იქნება ჩვენი საიტის დათვალიერება.

რაც შეეხება ვებ-გვერდების შექმნას, ეს სცილდება ჩვენი ნიგნის საზღვრებს და ამ საკითხებზე საუბარი არ გვექნება. მაშინ რაში გვეხმარება **Visual Basic.Net**? რატომ იყენებენ მას ვებ-გვერდების შესაქმნელად?

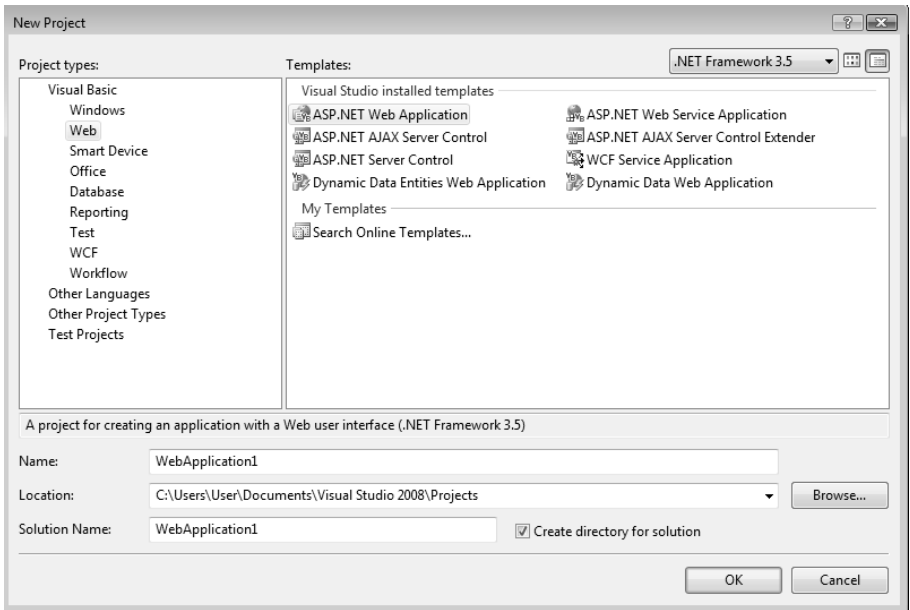
საქმე შემდეგშია: არსებობს დინამიური და სტატიური ვებ-გვერდები. ყველაზე უფრა კარგი საიტები ყოვლთვის

ინტერაქტიულია-ანუ დინამიურია. სტატიური ვებ-გვერდი საშუალებას გვაძლევს დავატვალიეროთ ის, გადავიდეთ გვერდებს შორის, დავათვალიეროთ სურათები, გადმოვწეროთ ფაილები, ვუყუროთ ვიდეოს.

დინამიური ვებ-საიტები კი ურთიერთობს მომხმარებელთან ინტერაქტიულად. ასეთი საიტებია მაგ: სოციალური ქსელები, ფორუმები დას ხვ. სადაც მომხმარებელს შეუძლია შევცვალოს ჩვენი პარამეტრები ბრაუზერიდან, გაგზავნოს და მიიღოს შეტყობინებები სასურველ პიროვნებასთან დინამიურად. მიიღოს ავტომატური შეტყობინება საიტიდან, გაგზავნოს წერილი პირდაპირ ვებ-გვერდიდან დას ხვ. ანუ ყოველივე ამისათვის საჭიროა გარკვეული ალგორითმი და შესაბამისი პროგრამული კოდის დანერა. **Visual Basic.Net** გვაძლევს სწორედ ამის საშუალებას. მისი დახმარებით შეიძლება უკვე არსებული ვებ-გვერდის გახსნა და მისი გადაკეთება დინამიურ ვებ-გვერდად. ან ახალი დინამიური ან სტატიური ვებ-გვერდის შექმნა ნულიდან.

ჩემი პირველი ვებ-გვერდი

გავუშვათ **Visual Basic**, შევიდეთ მენიუში **File** და ავირჩიოთ **New Web Site**. გამოჩენილ დიალოგურ ფანჯარაში ახალი დანართის შესაქმნელად ამოვარჩიოთ **ASP.NET ASP.NET Web Site**. გრაფაში **Name** დავარქვათ საქალაქს სახელი და მივუთითოთ მისი შენახვის მისამართი. დავაჭიროთ ლილაკს **OK**. შეიქმნება ცარიელი ვებ გვერდი. თუ დავაჭერთ პროგრამის გაშვების ლილაკს გამოჩნდება დიალოგის ფანჯარა **Debugging Not Enabled**. მოვნიშნოთ **Run Without Debugging** და დავაჭიროთ ლილაკს **OK**. ჩვენი ვებ-გვერდი გაიხსნება ბრაუზერში. რათქმაუნდა ვებ-გვერდი ცარიელია.

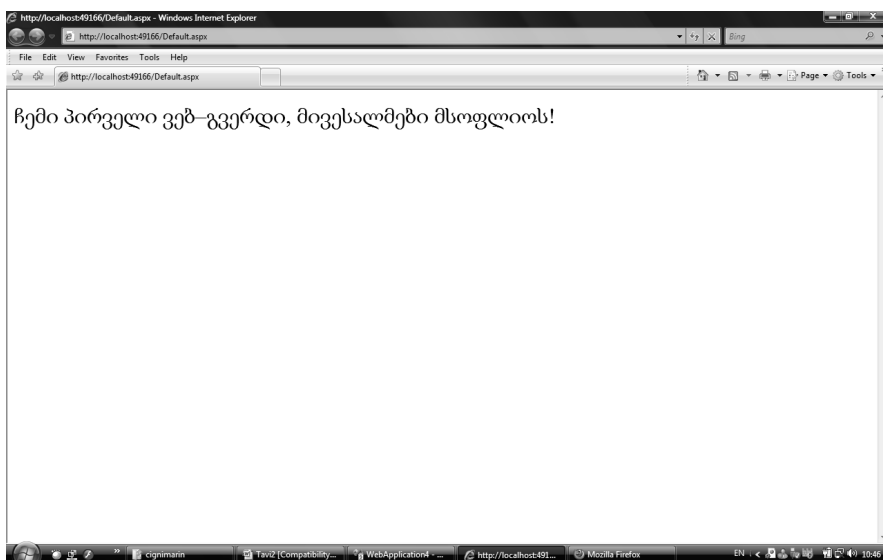


გადავიდეთ ინსტრუმენტების პანელ **Toolbox**-ზე და ჩამოვშალოთ სია **Standard**. გადმოვიტანოთ ჩვენს ცარიელ გვერდზე ელემენტი **Label**. თვისებათა ფანჯარაში შევცვალოთ მისი ტვისება **Text** და ჩავწეროთ შემდეგი წინადადებები “ჩემი პირველი ვებ-გვერდი, მივესალმები მსოფლიოს!”. შევვიძლია ჩავწეროთ **Unicode**-ით ქართულად, ან ჩავწეროთ ლათინური სიმბოლოებით და შემდეგ შევცვალოთ ელემენტის თვისება **Font**, ავირჩიოთ რომელიმე ქართული ფონტი.

ამ უკანასკნელის გაკეთება რეკომენდირებული არ არის. ინტერნეტში სასურველია გამოვიყენოთ ქართული უნიკოდი, რადგან თუ იმ კომპიუტერზე რომელზეც დაათვალიერებენ ჩვენს ვებ-გვერდს არ არის ჩანერილი ის ქართული ფონტი რომელიც

ჩვენ მივუთითეთ, გვერდი არ გაიხსნება ისე როგორც ჩვენ ველოდებით.

გავუშვათ პროგრამა, მოვნიშნოთ **Run Without Debugging** და დავაჭიროთ ლილავს **OK**. ჩვენი პირველი ვებ-გვერდი გაიხსნება ბრაუზერში.



თამაში “ჩაფიქრებული რიცხვის გამოცნობა” ინტერნეტში

ახლა კი შევექმნათ დინამიური ვებ-გვერდი რომელზეც ყველას შეეძლება ითამაშოს თამაში “ჩაფიქრებული რიცხვის გამოცნობა”. ცვენ ეს თამაში უკვე შევექმენით, მოდით ახლა შევექმნათ მისი ინტერნეტ-ვერსია, თამაში პირდაპირ ბრაუზერიდან.

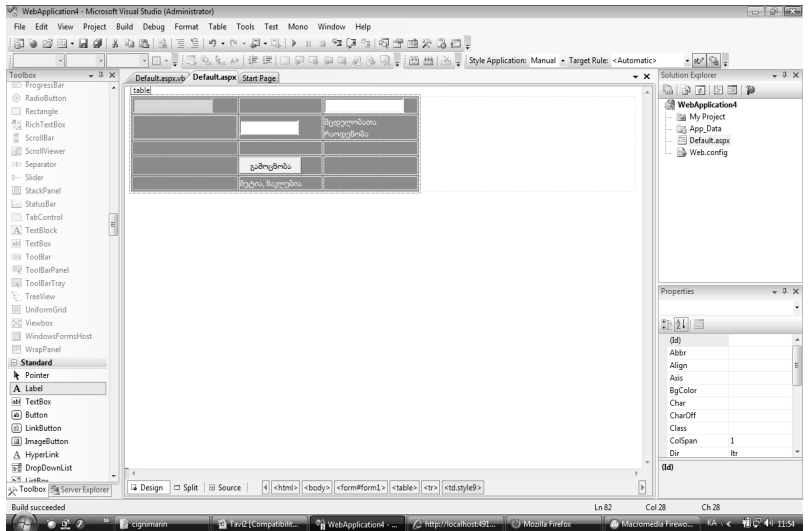
ამისათვის კვლავ სვექმნათ ახალი პროექტი. გავუშვათ **Visual Basic**, შევიდეთ მენიუში **File** და ავირჩიოთ **New Web Site**. გამოჩენილ დიალოგურ ფანჯარაში ახალი დანართის შესაქმნელად ამოვარჩიოთ **ASP.NET Web Site**. გრაფაში **Name** დავარქვათ საქალაქის სახელი და მივუთითოთ მისი შენახვის მისამართი. დავაჭიროთ ღილაკს **OK**. შეიქმნება ცარიელი ვებ გვერდი.

გადავიდეთ ინსტრუმენტების პანელ **Toolbox**-ზე და ჩამოვშალოთ სია **Standard**. გადმოვიტანოთ ჩვენს ცარიელ გვერდზე ელემენტები: სამი ელემენტი **TextBox**, 2 ელემენტი **Label** და 2 ელემენტი **Button**, თვისებათა ფანჯარაში შეცვალებთ მათი თვისებები **Text** (სადაც ეს საჭიროა) და განალაგეთ ისინი ისე როგორც ნაჩვენებია სურათზე.

ვებ-გვერდზე ელემენტების სასურველი პოზიციონირება ისე მარტივი არ არის როგორც ფორმაზე. მათი პოზიციონირებისათვის აუცილებლად დაგვჭირდება ელემენტი **Table**. ვებ გვერდზე ელემენტები გარკვეულ უხილავ ან ხილულ ცხრილებში უნდა მოვათავსოთ. ამისათვის გადავიდეთ ინსტრუმენტების პანელ **Toolbox**-ზე და ჩამოვშალოთ სია **HTML**. ავირჩიოთ ელემენტი **Table** და გადმოვიტანოთ ვებ-გვერდზე. ამ ელემენტზე ექსპერიმენტების შემდეგ შევძლებთ მისთვის სასურველი ფორმის მიცემას. სტრიქონების ან სვეტების დამატება შეგვიძლია მასზე მაუსის მარჯვენა ღილაკის დაჭერით

და კონტექსტურ მენიუში შესაბამისი ბრძანებების არჩევით: **Insert -> Row or Columns.**

ცხრილს შეგვიძლია თვისებათა ფანჯარაში მივცეთ სასურველი ფერი. ჩვენ შემთხვევაში ავირჩიეთ ციფერი ფერი.





შემდეგ Button1-ში და Button2-ში ჩავწეროთ კოდი:

```
Protected Sub Button1_Click(ByVal sender As Object, _  
ByVal e As EventArgs) Handles Button1.Click  
    If Val(TextBox2.Text) > Val(TextBox1.Text) Then  
Label2.Text = "ჩაფიქრებული რიცხვი ნაკლებია"  
    If Val(TextBox2.Text) < Val(TextBox1.Text) Then  
Label2.Text = "ჩაფიქრებული რიცხვი მეტია"  
    If TextBox2.Text = TextBox1.Text Then  
Label2.Text = "თქვენ გამოიგანით ჩაფიქრებული _  
რიცხვი"  
        TextBox1.Visible = True  
    End If    TextBox3.Text = TextBox3.Text + 1  
End Sub
```

```

Protected Sub Button2_Click(ByVal sender As Object, _
ByVal e As EventArgs) Handles Button2.Click
    TextBox1.Visible = False
    Randomize()
    TextBox1.Text = Int(Rnd() * 100)
    TextBox3.Text = 0
    Button1.Enabled = True
End Sub

```

End Class

კოდის დეტალურ განმარტენბას აქ არ მოვიყვანთ, რადგან ამაზე საუბარი უკვე გვქონდა ასეთი თამაშის პროექტის შექმნისას **Windows** ფორმის გამოყენებით.

გავუშვათ პროგრამა, მოვნიშნოთ **Run Without Debugging** და დავაჭიროთ ლილაკს **OK**. ჩვენი ვებ-გვერდი გაიხსნება ბრაუზერში. შეგვიძლია დავინყოთ თამაში!

დინამიური ვებ-გვერდი, მონაცემთა ბაზები

მონაცემთა ბაზები დინამიური ვებ-გვერდების ერთ-ერთი უმნიშვნელოვანესი შემადგენელი ნაწილია. მაგ: სოციალურ ქსელებში, მათში დარეგისტრირებული კლიენტების სესახებ ინფორმაცია ინახება მონაცემთა ბაზაში. სახელის და პაროლის შეყვანის შემდეგ მონაცემთა ბაზიდან მოხდება კლიენტის ინფორმაციის გამოძახება და ის აისახება ბრაუზერის ეკრანზე. მონაცემთა ბაზებზეა აგებული ფორუმების, საინფორმაციო საიტების (არქივის ფუნქციით) მუშაობა და სხვ. დინამიურ ვებ –

გვერდებში საიტის მომხმარებლებს ხსირად ადქვთ საშუალება შეიტანონ თავიანტი ინფორმაცია, დარეგისტრირდნენ დას ხვ.

მოდით შევქმნათ მარტივი მონაცემთა ბაზა ინტერნეტში. სადაც შეგვიძლია ბრაუზერიდან დავამატოთ ინფორმაცია, მაგალითად ინფორმაცია ჩვენს შესახებ. სახელი, გვარი, მისამართი, სამუშაო ადგილი დას ხვ. თავდაპირველად აუცილებელია მკაფიოდ განვსაზღვროთ მონაცემთა ბაზის სტრუქტურა. რადგან ჩვენი ბაზა მარტივია, მისი სტრუქტურის განსაზღვრა დიდ სირთულეს არ წარმოადგენს.

გავუშვათ **Visual Basic**, შევიდეთ მენიუში **File** და ავირჩიოთ **New Web Site**. გამოჩენილ დიალოგურ ფანჯარაში ახალი დანართის შესაქმნელად ამოვარჩიოთ **ASP.NET Web Site**. გრაფაში **Name** დავარქვათ საქალაქს სახელი და მივუთითოთ მისი შენახვის მისამართი. დავაჭიროთ ლილაკს **OK**. შეიქმნება ცარიელი ვებ გვერდი.

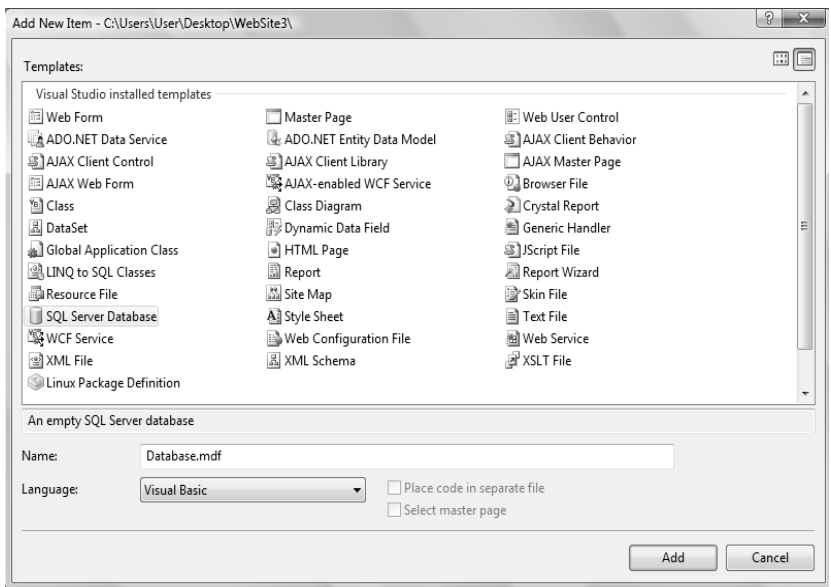
Solution Explorer-ში მოვნიშნოთ პროექტის დასახელება და დავაჭიროთ მაუსის მარჯვენა ლილაკს. კონტექსტური მენიუდან ავირჩიოთ ბრძანება **Add New Item**. გაიხსნება ფანჯარა სადაც დავაჭიროთ ლილაკს **OK**. გაიხსნება ფანჯარა **Add New Item**. მოვნიშნოთ **SQL Server Database**, დავარქვათ მობაცემთა ბაზას სახელი და დავაჭიროთ ლილაკს **Add**.

Solution Explorer-ში გამოჩნდება ჩვენს მიერ შექმნილი მონაცემთა ბაზა. ახლა უნდა შევქმნათ ცხრილი და შევიტანოთ მასში პირველადი მონაცემები. მენიუთა სტრიქონში ავირჩიოთ მენიუ **View->Server Explorer**. გადავიდეთ **Server Explorer**-ზე. შემდეგ მენიუთა სტრიქონში ავირჩიოთ **Data->Add New->Table**. გაიხსნება ცხრილი სადაც ჩვენ შეგვიძლია მივუთითოთ ცხრილის სვეტების სახელები, მათში შესაყვანი ინფორმაციის ფორმატი დას ხვ.

სვეტში **Column Name** შევიყვანოთ შემდეგი დასახელებები: **ID**, სახელი, გვარი, ასაკი, მისამართი, ტელეფონი. შემდეგ მოვნიშნოთ სტრიქონი **ID** და დავაჭიროთ მაუსის მარჯვენა ლილაკს. კონტექსტურ მენიუში ავირჩიოთ ბრძანება—**Set Primary Key**. დავაჭიროთ ლილაკს **Save**. გამოჩნდება ფანჯარა სადაც ჩვენ უნდა დავარქვათ სახელი ჩვენს ცხრილს. დავაჭიროთ ლილაკს **OK**.

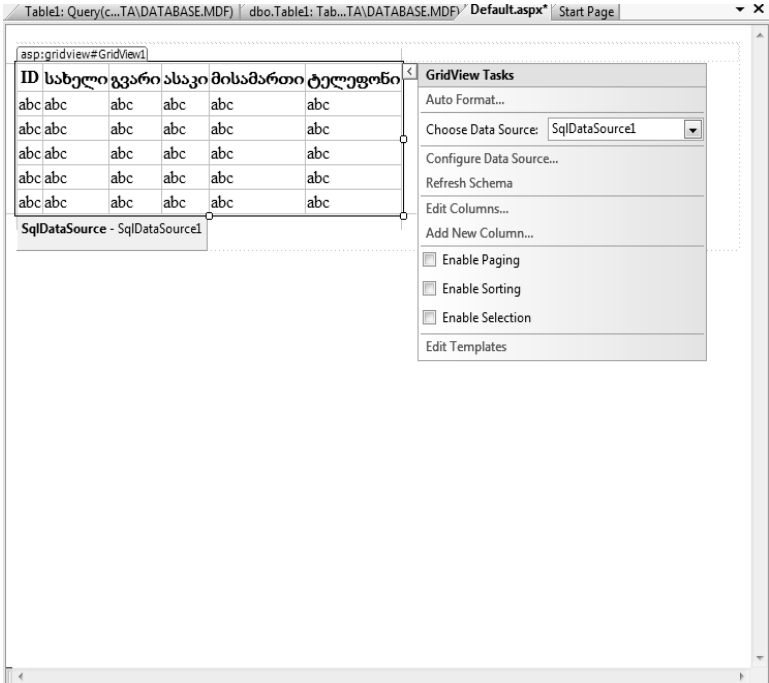
ჩვენი ცხრილი შეიქმნა. ახლა მასში უნდა შევიტანოთ პირველადი მონაცემები. ამისათვის **Server Explorer**-ში მოვნახოთ ჩვენი მონაცემთა ბაზა, ჩამოვშალოთ ის და შემდეგ ჩამოვშალოთ საქალაქდ **Tables**. გამოჩნდება ჩვენს მიერ შექმნილი ცხრილი, მოვნიშნოთ ის. მენიუთა სტრიქონში ავირჩიოთ **Data->Show Table Data**. გაიხსნება ჩვენი ცხრილი სადაც შეგვიძლია შევიყვანოთ პიროვნებების მონაცემები (კონკრეტული სახელი, გვარი...).

დავაჭიროთ ღილაკს **Save**.

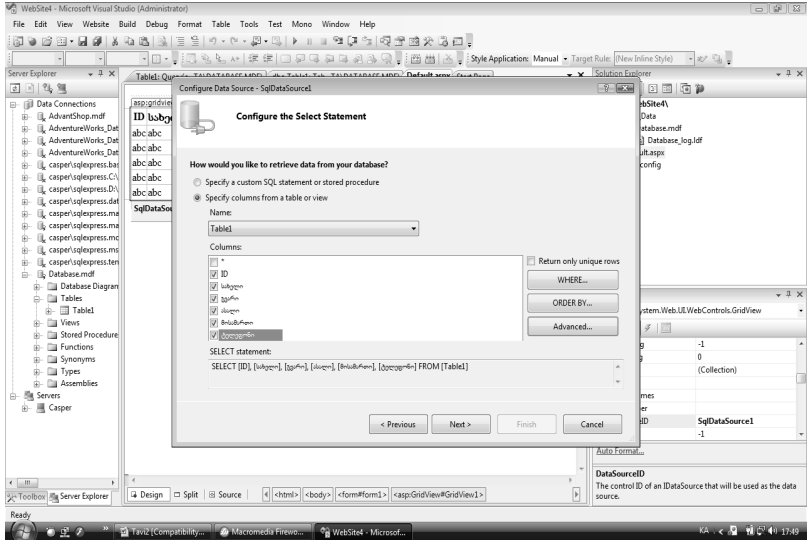


Column Name	Data Type	Allow Nulls
ID	nchar(10)	<input type="checkbox"/>
სახელი	nchar(10)	<input checked="" type="checkbox"/>
გვარი	nchar(10)	<input checked="" type="checkbox"/>
ასაკი	nchar(10)	<input checked="" type="checkbox"/>
მისამართი	nchar(10)	<input checked="" type="checkbox"/>
ტელეფონი	nchar(10)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

შემდეგ გადავიდეთ ჩვენს ცარიელ ვებ-გვერდზე და **Server Explorer**-ში მოვნახოთ ჩვენი მონაცემთა ბაზა, ჩამოვშალოთ ის და შემდეგ ჩამოვშალოთ საქალაქო **Tables**. გამოჩნდება ჩვენს მიერ შექმნილი ცხრილი. გადმოვიტანოთ ის ვებ-გვერდზე (ისევე როგორც მართვის ელემენტები). ვებ-გვერდზე გამოჩნდება ელემენტი **Grid View**.

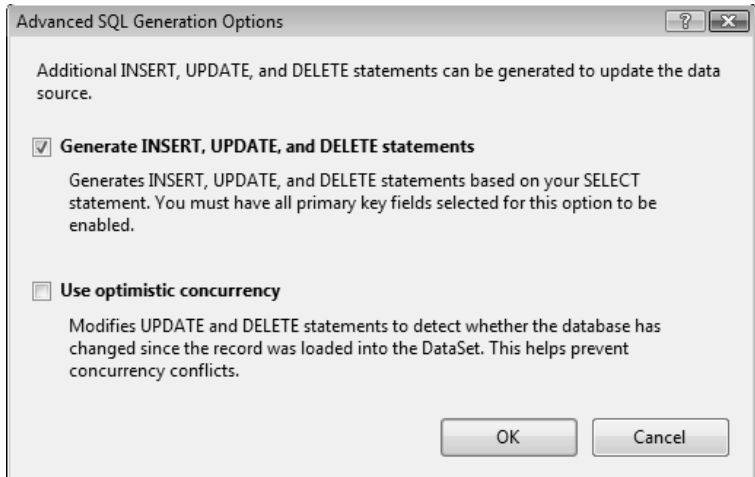


მოვნიშნოთ ელემენტი **Grid View**. დავაჭიროთ პატარა ლილაკს ელემენტის ზედა მარლვენა კიდესთან. გაიხსნება ფანჯარა **GridView Tasks**. დავაჭიროთ ბრძანებას **Configure Data Source**. დავაჭიროთ ლილაკს **Next**. მოვნიშნოთ **Specify columns from table or view**. მოვნიშნოთ ცხრილის სვეტების დასახელებები, და დავაჭიროთ ლილაკს **Advances**.



გახისნება ფანჯარა **Advanced SQL Generation options**. მოვნიშნოთ ოპცია **Generate INSERT, UPDATE, and DELETE statements**. ეს საშუალებას მოგვცემს შესაძლებელი იყოს მონაცემთა ბაზაში ცვლილებების შეტანა (ჩასმა, შეცვლა, ნაშლა). შემდეგ დავაჭიროთ ლილაკს **Next**, შემდეგ გახსნილ ფანჯარაში დავაჭიროთ ლილაკს **Finish**.

ოპცია **Generate INSERT, UPDATE, and DELETE statements** რომ ხელმისაწვდომი იყოს ამისათვის აუცილებელია ჩვენს მონაცემთა ბაზის რომელიმე სვეტს მინიჭებული ქონდეს **“Primary Key”**.



გადავიდეთ ვებ-გვერდზე, მოვნიშნოთ ელემენტი **Grid View**. დავაჭიროთ პატარა ღილაკს ელემენტის ზედა მარჯვენა კიდესთან. გაიხსნება ფანჯარა **GridView Tasks**. **movniSnoT is opciebi**, რომლებიც საშუალებას მოგვცემს განვახორცილოთ ცვლილებები ბაზაში. თუ მონიშნულია ცვლილება, მისი შესაბამისი ბრძანება გამოჩნდება ბრაუზერში პროექტის გაშვებისას (მაგ: **Delete**, **Edit**, **Select**). მათზე ექსპერიმენტებით გაეცნობით მატ ეფექტებს.

ასევე შეგვიძლია გავხსნათ ფანჯარა **Auto format** და შევარჩიოთ ჩვენი ცხრილის სასურველი დიზაინი.

გავუშვათ პროგრამა, მოვნიშნოთ **Run Without Debugging** და დავაჭიროთ ღილაკს **OK**. ჩვენი დინამიური ვებ-გვერდი მონაცემთა ბაზით გაიხსნება ბრაუზერში.

შეგვიძლია პირდაპირ ბრაუზერიდან შევიტანოთ ცვლილებები მონაცემთა ბაზაში. ასეთივე ცვლილებების შეტანა შეეძლება ჩვენი ვებ-გვერდის ყველა სტუმარს.

ID	სახელი	გვარი	ასაკი	მისამართი	ტელეფონი	
Edit Delete Select	1	მაშუა	გიორგაძე	21	თბილისი	234532
Edit Delete Select	2	ირაკლი	ალანია	34	თბილისი	234532
Edit Delete Select	3	გიორგი	კაციტაძე	23	ქუთაისი	579342344
Edit Delete Select	4	ნანა	შხითარიაძე	41	თბილისი	342234
Edit Delete Select	5	ეკა	კუკულაძე	27	თბილისი	554323
Edit Delete Select	6	თუა				

ჩვენს მიერ შექმნილი ვებ-გვერდის დათვალიერება შეგვიძლია ჩვენს კომპიუტერში, რადგანაც Visual Studio-ს მოყვება საკუთარი სერვერი.

იმისათვის ჩვენი საიტი ვაჩვენოთ მსოფლიოს ამისათვის ის უნდა განვათავსოთ რომელიმე სერვერზე. ფაილები რომელსაც შეიცავს ჩვენი პროექტი ატვირთოთ სერვერზე.

Visual Studio-ს გარემოში შექმნილი დინამიური ვებ-გვერდების ფუნქციონერებისათვის აუცილებელია Windows სერვერი, ის არ იმუშავებს Linux სერვერზე.

სერვერზე საიტის ატვირთვა სცილდება ჩვენი წიგნის ფარგლებს და ამაზე საუბარი არ გვექნება, ვფიქრობთ დამოუკიდებლად მისი შესწავლა თქვენთვის პრობლემა არ იქნება.

პროგრამის საინსტალაციო პაკეტის შექმნა



პროექტის შექმნის შემდეგ, რომელიც მუშაობს ჩვენს კომპიუტერზე დგება საკითხი: მუშავენს თუ არა ის ასევე წარმატებით სხვა კომპიუტერებზე. განსაკუთრებით მნიშვნელოვანია ეს საკითხი პროგრამის ტირაჟირებისას.

რომელი ფაილები უნდა გადავიტანოთ სხვა კომპიუტერზე?

აუცილებელია თუ არა მათში ჩანერილი იყოს **Visual Studio**?

სხვა რა პროგრამები უნდა იყოს ჩანერილი კომპიუტერში ჩვენი პროგრამის გამართულად მუშაობისათვის?

კომპიუტერში სადაც უნდა მუშაობდეს ჩვენი პროექტი რათქმაუნდა აუცილებელი არ არის ჩანერილი იყოს **Visual studio**.

მოვნახოთ რომელიმე ჩვენს მიერ შექმნილი პროექტის საქალაღდე. გავხსნათ ის, შემდეგ გადავიდეთ საქალაღდეზე **Bin** და მოვნახოთ საქალაღდე **Debug**. სწორედ აქ არის ჩანერილი ჩვენი პროექტის ვერსია, რომელიც მუშაობს მანქანურ ენაზე ანუ მას არ ესაჭიროება **visual Basic**-ის გარემო.

ნებისმიერ დაპროგრამების ენაზე შექმნილი პროგრამა ბოლოს უნდა გადავიყვანოთ მანქანურ ენაზე (კომპილაცია). ამის შემდეგ მანქანურ ენაზე გადაყვანილი ვერსია დაკარგავს კავშირს იმ დაპროგრამების ენასთან, რომელზეც ის შეიქმნა და იწყებს “დამოუკიდებელ ცხოვრებას” იმუშავენს კომპიუტერებზე სადაც მისი შემქმნელი პროგრამა შეიძლება არ იყოს ჩანერილი. დამოუკიდებელი პროგრამის გამშვები ფაილი **EXE** ფაილია.

Visual studio პროგრამის ყოველი გაშვებისას ახდენს კომპილაციას და ქმნის პროექტის ვერსიას მანქანურ ენაზე საქალაქდებში **Debug**.

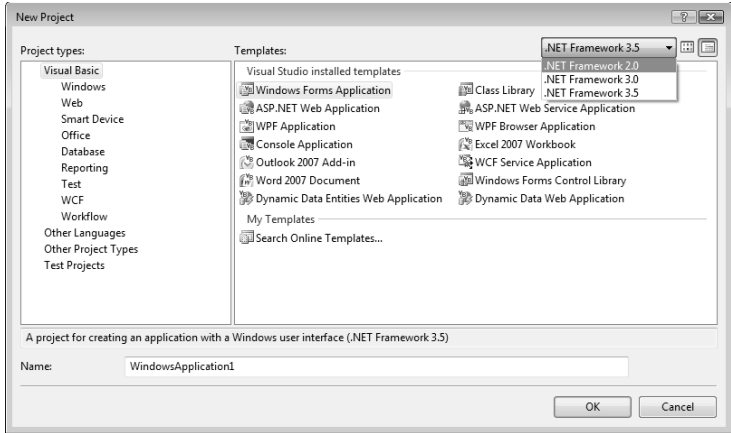
თუ წავიღებთ საქალაქდეს **Debug** და გადავიტანთ სხვა კომპიუტერზე პროგრამა იმუშავებს, მაგრამ ეს ყოვლისაგანაა საკმარისი არ არის. სწორედ ამისათვის არის საჭირო პროექტის საინსტალაციო ვერსიის შექმნა, რომელიც ჩანერს კომპიუტერში ყველა იმ პროგრამას, რომელიც საჭიროა ჩვენი პროექტის გამართული მუშაობისათვის.

პირველ რიგში პროგრამა არ იმუშავებს თუ კომპიუტერში ჩანერილი არ არის **.NET Framework** პლატფორმა. ასევე გასათვალისწინებელია მისი ვერსიები. როდესაც ჩვენ ვქმნით პროექტს საშუალება გვაქვს ავარჩიოთ მისი ვერსიები (**.NET Framework 2.0**, **.NET Framework 3.0**, **.NET Framework 3.5**) ფანჯარაში **New Project**.

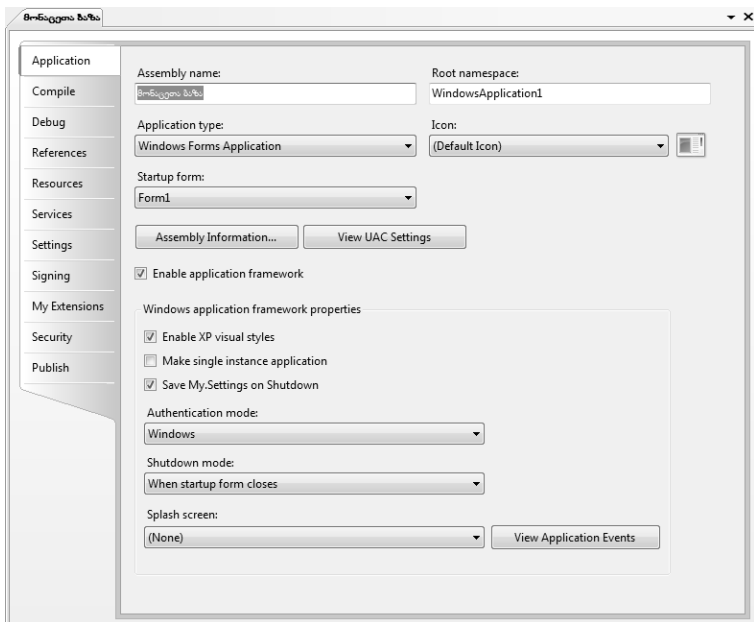
სხვადასხვა პროექტს მისი მუშაობისათვის შეიძლება სხვადასხვა პროგრამა დასჭირდეს (მაგ: თუ პროექტში არის რეპორტი და გამოყენებულია **ReportViewer** საჭიროა იმ კომპიუტერში სადაც ეს პროგრამა იმუშავებს ჩანერილი იყოს **Microsoft Visual Studio ReportViewer**).

საინსტალაციო ვერსია ისე უნდა შეიქმნას, რომ კომპიუტერში ჩანეროს ყველა აუცილებელი კომპონენტი.

მაშ ასე, შევქმნათ რომელიმე ჩვენი პროექტის საინსტალაციო ვერსია. ჩვენ გამოვიყენებთ პროექტი მონაცემთა ბაზა "სტუდენტები".

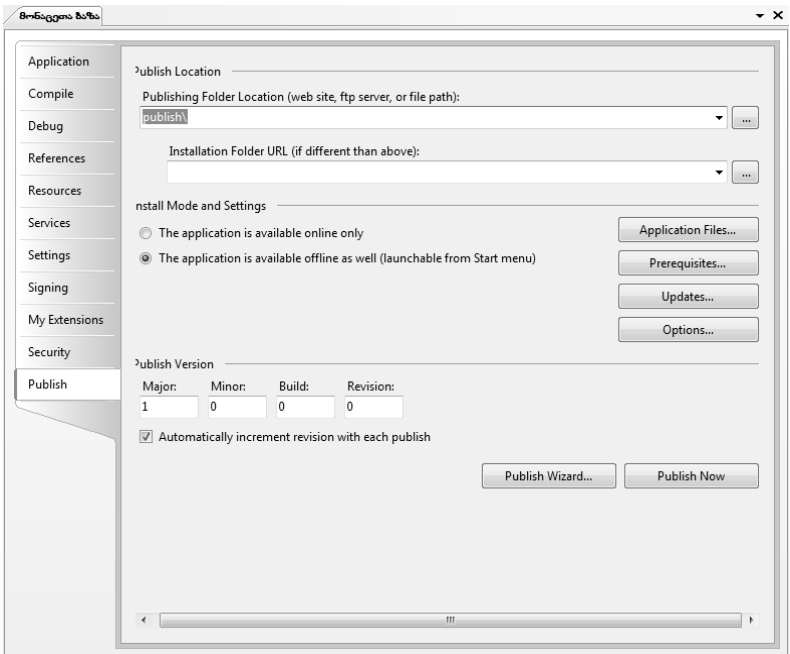


გახსნათ ეს პროექტი Visual Studios გარემოში. Solution Explorer-ში მოვნიშნოთ ჩვენი პროექტის დასახელება და დავაჭიროთ მაუსის მარჯვენა ღილაკს. გახსნილ კონტექსტურ მენიუში ავირჩიოთ მენიუ Properties.



გაიხსნება ფანჯარა **Application**. აქ **Cven SegviZlia SevcrvaloT saxeli Assembly Name**, რომლითაც ჩვენი პროექტი ჩაინსტალირებისას გამოჩნდება კომპიუტერის მენიუში **Start**.

შეგვიძლია ასევე შევარჩიოთ პროექტის იკონა. სხვა პარამეტრებში ვფიქრობთ თქვენით მარტივად გაერკვევით ახლა კი გადავიდეთ ფანჯარაზე **Publish**.

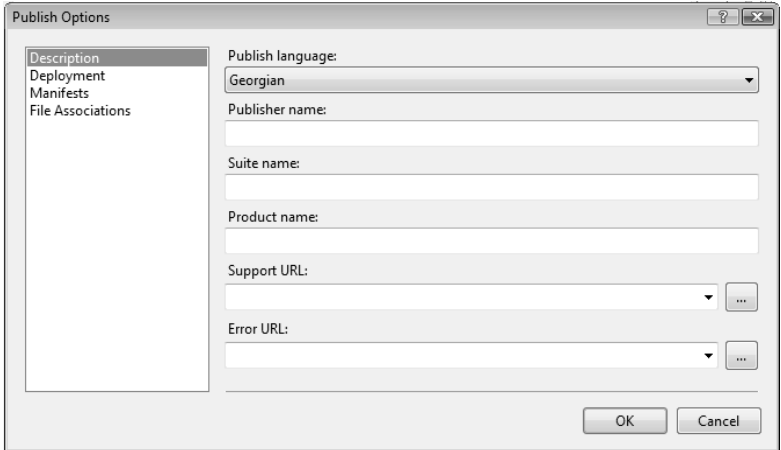


აქ ველში **Publishing Folder Location** უნდა მივუთითოთ საქალაქო სადაც უნდა განთავსდეს პროექტის საინსტალაციო ვერსია. დავაჭიროთ ღილაკს ველის მარჯვენა მხარეს და გახსნილ ფანჯარაში მივუთითოთ სასურველი საქალაქო.

დააკვირდით გრაფას **Publish Version** ფანჯარის ქვედა ნაწილში. აქ საშუალება გვქვია მივუთითოთ პროექტის ვერსია ან საინსტალაციო ვერსიის შექმნისას **Visual Studio** მას ავტომატურად მიანიჭებს (წინა ვერსიას უმცირეს თანრიგში

დაამატებს ერთს თუ მონიშნულია პუნქტი **Automatically increment revision each publish**).

დავაჭიროთ ლილაკს **Options**. ფანჯარაში დესკრიპტიონ შეგვიძლია ავირჩიოთ ენა, ჩავწეროთ პროდუქტის სახელი დას ხვ.



გადავიდეთ ფანჯარაზე **Deployment**. აქ ყველაზე საინტერესოა ის რომ შეგვიძლია მივუთითოთ ინსტალაციის ავტომატური დაწყება როცა დისკი ჩადებულია (**Autorun**), ამისათვის უნდა მოვნიშნოთ პუნქტი **For CD installations, automatically Start Setup when SD is inserted**.

გადავიდეთ ფანჯარაზე **Manifests**. აქ შეგვიძლია მივუთითოთ **Desktop**-ზე პროგრამის **Shortcut**-ის სექმნა. მოვნიშნოთ პუნქტი **Create Desktop Shortcut**. დავაჭიროთ ლილაკს **OK**.

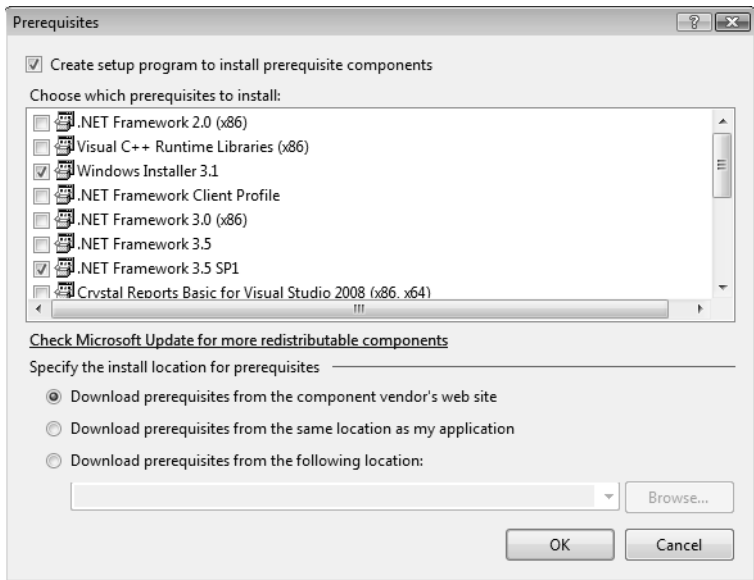
დავაჭიროთ ლილაკს **Prerequisites**. ფანჯარაში **Prerequisites** ჩამოთვლილია ის კომპონენტები რომელიც შესაძლოა ჩაინსტალირდეს კომპიუტერში ჩვენს პროგრამასთან ერთად. აუცილებელი კომპონენტები როგორც ჩვენ ვნახავთ უკვე მონიშნულია. თუ ჩვენ ვთვლით რომ საჭიროა სხვა კომპონენტიც უნდა მოვნიშნოთ შესაბამისი პუნქტიც.

თუ ჩვენ ვთვლით რომ ეს კომპონენტები ჩანერილი უნდა იყოს საინსტალაციო დისკზე ფანჯარის ქვემო ნაწილში უნდა მოვნიშნოთ პუნქტი **Download prerequisites from the same location as my application.**

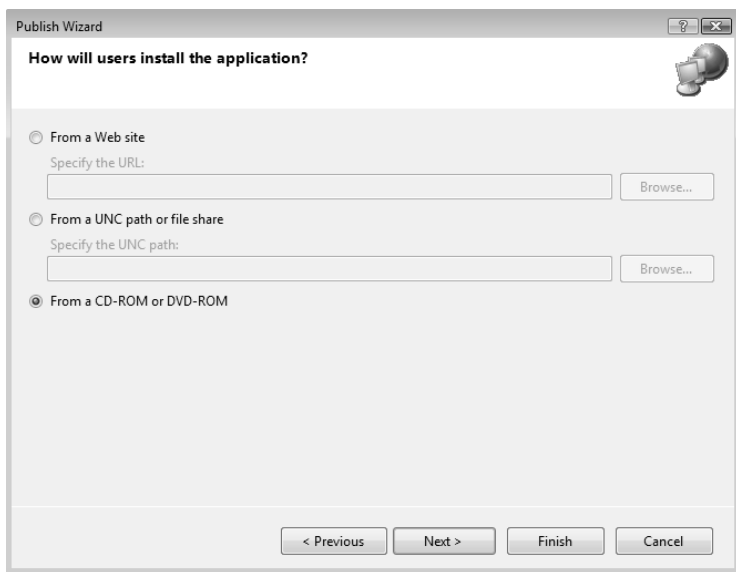
თუ ჩვენ ვთვლით რომ ეს კომპონენტები ჩვენი პროგრამის მომხმარებელმა უნდა გადმოწეროს ინტერნეტიდან უნდა მოვნიშნოთ პუნქტი **Download prerequisites from the component vendor's site.**

მესამე პუნქტში შეგვიძლია მივუთითოთ სხვა მისამართი.

მოვნიშნოთ პუნქტი **Download prerequisites from the same location as my application** დავაჭიროთ ღილაკს **OK.**



დავაჭიროთ ღილაკს **Publish Wizard.** შემდეგ ღილაკს **Next.**



ახალ ფანჯარაში Publish Wizard მოვნიშნოთ პუნქტი **From a CD-ROM or DVD-ROM**. დავაჭიროთ ლილაკს **Finish**. დაიწყება საინსტალაციო ვერსიის შექმნა, პროცესი რამოდენიმე წამს გაგრძელდება. დასასრულს შეგვიძლია გავხსნათ ჩვენი საქალაღდე, რომელიც მივუთითეთ საინსტალაციო ვერსიისათვის. დავაჭიროთ ფაილს **Setup**, დაიწყება პროგრამის ინსტალაცია.

პროგრამის გასაგრცელებლად ან სხვა კომპიუტერზე გადასატანად გადასატან მონყობილობაში უნდა ჩაინეროს საინსტალაციო საქალაღდეში არსებული ფაილები.

Name	Date modified	Type	Size	Tags
Application Files	10.09.11 16:42	File Folder		
setup	10.09.11 16:42	Application	667 KB	
მონაცეთა პაზა	10.09.11 16:42	Application Manif...	6 KB	

რეკომენდირებული ლიტერატურა:

1. თ. მაჭარაძე, ზ. წვერაიძე “ინფორმატიკის საფუძვლები”. თბილისი 2009. საგამომცემლო სახლი “ტექნიკური უნივერსიტეტი” ISBN 978-9941-14-378-6.
2. Microsoft Visual Studio 2008 Programming - By Jamie Plenderleith, Steve Bunn - McGraw-Hill (2009) - Paperback - 412 pages - ISBN 0071604081.
3. Microsoft Visual Studio 2008 unleashed - By Lars Powers, Mike Snell - SAMS (2008) - Paperback - 1219 pages - ISBN 0672329727.
4. WPF in Action with Visual Studio 2008 -By Arlen Feldman, Maxx Daymon - Manning Publications Company (2008) - Paperback - 490 pages - ISBN 1933988223.
5. Professional Visual Studio 2008--Nick Randolph, David Gardner - John Wiley & Sons (2008) - Paperback - 1032 pages - ISBN 0470229888.
6. Professional Visual Studio 2008 [Book] -By Nick Randolph, David Gardner - John Wiley & Sons (2011) - Ebook - 1032 pages - ISBN 1118059522.
7. Microsoft Visual Studio 2008 Автор: Ларс Пауэрс, Майк Снелл Издательство: БХВ-Петербург Год: 2009 ISBN: 978-5-9775-0378-5, 978-0-672-32972-2.
8. Visual Studio .NET: разработка приложений баз данных Автор: Анатолий Постолиит Издательство: БХВ-Петербург Год: 2003 ISBN: 5-94157-309-X.
9. Visual C# 2008: базовый курс. Visual Studio® 200 8 – Карли Уотсон, Кристиан Нейгел, Якоб Хаммер Педерсен, и др. 2009.
10. Книга: C#. Разработка компонентов в MS Visual Studio 2005/2008 Автор: Павел Агуров Страниц: 480 Год издания: 2008.

11. С# 2008 для чайников по Си Шарп. Язык программирования С# 3.0 для .NET 3.5 Стефан Рэнди Дэвис, Чак Сфер 2008
12. Обработка баз данных на Visual Basic .NET Автор: Манус Джеффри П., Голдштейн Джеки, Прайс Кевин Т. Издательство: Вильямс Год: 2003 ISBN: 5-8459-0512-5.