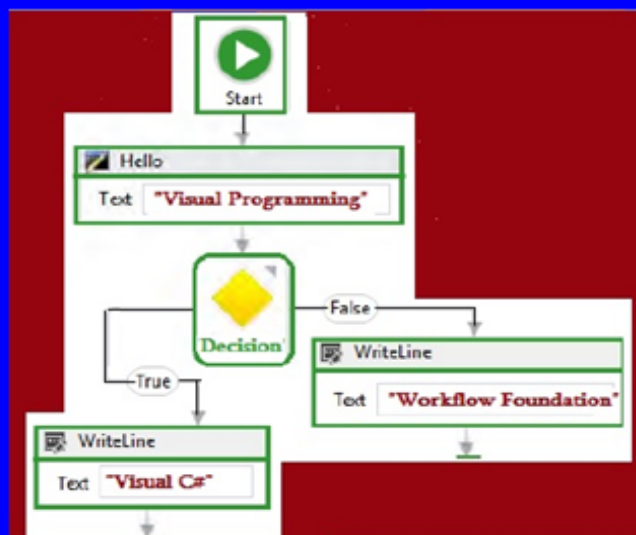


ბიზ სურბულაკი, ლილი კებრიაშვილი

ვიზუალური დაპროგრამება (C#.NET, Workflow Foundation.NET)



„ტექნიკური უნივერსიტეტი“

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე, ლილი პეტრიაშვილი

ვიზუალური დაპროგრამება (C#.NET, Workflow Foundation.NET)



რეკომენდებულია საქართველოს
ტექნიკური უნივერსიტეტის
სარედაქციო -საგამომცემლო
საბჭოს მიერ 29.03.2017, ოქმი N1

თბილისი
2017

უაკ 004.5

განხილულია მართვის საინფორმაციო სისტემების პროგრამული უზრუნველყოფის აგების ვიზუალური, ობიექტორიენტირებული ენა C#.NET და ახალი ტექნოლოგია Workflow Foundation - MsVisual Studio.NET 2015 ვერსიის ბაზაზე. შემოთავაზებულია პრაქტიკული ღირებულების 15 ლაბორატორიული ამოცანა ვიზუალური დაპროგრამების საწყისებისა და ძირითადი პრინციპების შესასწავლად. ექსპერიმენტებისათვის გამოყენებულია MsVisual Studio.NET Framework 4.5 ინტეგრირებული პლატფორმა.

დამხმარე სახელმძღვანელო განკუთვნილია მართვის საინფორმაციო სისტემების (Management Information Systems) და პროგრამული ინჟინერიის (Software Engineering) სპეციალობის ბაკალავრიატის სტუდენტებისა და მაგისტრანტებისათვის. აგრეთვე სხვა სფეროს სპეციალისტებისათვის, რომელთაც სურვილი აქვთ შეისწავლონ დაპროგრამების ვიზუალური მეთოდები და ინსტრუმენტული საშუალებანი Ms VisualStudio.NET Framework 4.5 ვერსიის C# ენის ბაზაზე.

რეცენზენტები:

საქართველოს ტექნიკური უნივერსიტეტის
კომპიუტერული ინჟინერიის დეპარტამენტის
პროფესორი *რომან სამხარაძე*,

საქართველოს ტექნიკური უნივერსიტეტის
მართვის ავტომატიზებული სისტემების
დეპარტამენტის პროფესორი *ეკატერინე თურქია*

© საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, 2017

ISBN 978-9941-20-880-5

<http://www.gtu.ge>

ყველა უფლება დაცულია. ამ წიგნის არც ერთი ნაწილის (იქნება ეს ტექსტი, ფოტო, ილუსტრაცია თუ სხვა) გამოყენება არანაირი ფორმით და საშუალებით (იქნება ეს ელექტრონული თუ მექანიკური) არ შეიძლება გამომცემლის წერილობითი ნებართვის გარეშე.

საავტორო უფლებების დარღვევა ისჯება კანონით.



ავტორთა შესახებ:

გია სურგულაძე - სტუ-ს „მართვის ავტომატიზებული სისტემების (პროგრამული ინჟინერიის)“ დეპარტამენტის უფროსი, პროფესორი, ტექნიკის მეცნიერებათა დოქტორი, გაეროსთან არსებული „ინფორმატიზაციის საერთაშორისო აკადემიის (IIA)“ ნამდვილი წევრი, სტუ-ს „IT-კონსალტინგის სამეცნიერო ცენტრის“ ხელმძღვანელი, გერმანიის DAAD-ის გრანტის მრავალგზის მფლობელი, ბერლინის ჰუმბოლდტის, ნიურნბერგ-ერლანგენის, მაგდებურგის, პასაუს და სხვა უნივერსიტეტების მიწვეული პროფესორი 1991-2016 წწ. 350-ზე მეტი სამეცნიერო ნაშრომის ავტორი, მათ შორის 71 წიგნის და 55 ელ-სახელმძღვანელოსი, მართვის საინფორმაციო სისტემების ინჟინერინგის სფეროში.

ლილი პეტრიაშვილი - სტუ-ს „მართვის ავტომატიზებული სისტემების (პროგრამული ინჟინერიის)“ დეპარტამენტის პროფესორი, ტექნიკის მეცნიერებათა კანდიდატი, გერმანიის DAAD-ის გრანტის მრავალგზის მფლობელი, ბერლინის ტექნიკური უნივერსიტეტის, ნიურნბერგ-ერლანგენის, კაიზერსლაუტერნის და სხვა უნივერსიტეტების მიწვეული პროფესორი 1998-2016 წწ. 100-ზე მეტი სამეცნიერო ნაშრომის ავტორი, მათ შორის 14 სახელმძღვანელოსი. წლების განმავლობაში წაკითხული აქვს ლექციები საქართველოს წამყვან უნივერსიტეტებში, მონაცემთა ბაზებისა და საცავების, პროექტების მენეჯმენტის, ვიზუალური დაპროგრამების ენების, ტელემატიკის, პეტრის ქსელების და სხვა დისციპლინებში ქართულ და გერმანულ ენებზე.

სარჩევი

შესავალი	7
თავი 1. Ms Visual Studio .NET Framework პლატფორმა	8
1.1. სამუშაო გარემო და C# აპლიკაციის პროექტის სტრუქტურა	8
1.2. პროექტის აგება კონსოლის რეჟიმში C#-კოდით და მისი გამართვა	18
1.3. Windows ფორმებთან მუშაობა და ვიზუალური ელემენტები: Button, Label, TextBox	27
1.4. ფორმაზე ვიზუალურ ელემენტებთან მუშაობა (Size, Location),.....	33
1.5. მართვის ელემენტი - ტაიმერი (Timer)	36
1.6. მართვის ელემენტი - რიცხვების არჩევა (NumericUpDown)	38
თავი 2. მართვის კონტეინერული ელემენტები	40
2.1. კონტეინერული ელემენტი - Panel	41
2.2. კონტეინერული ელემენტები: CheckBox, RadioButton, GroupBox	43
2.3. კონტეინერული ელემენტები: GroupBox და TabControl ...	50
თავი 3. სტრიქონული ტიპის მონაცემების დამუშავება	57
3.1. სტრიქონული ტიპის მონაცემები და String კლასი	57
3.2. სტრიქონების დამუშავების მეთოდები: Trim() და Replace()	59
3.3. სტრიქონში ძებნის მეთოდები: IndexOf(), LastIndexOf() და IndexOfAny()	62
3.4. სტრიქონებთან მუშაობა : Insert (), Remove(), Substring()...	67
თავი 4. მართვის ვიზუალური ელემენტები: ListBox და ComboBox	72
4.1. ვიზუალური ელემენტი ListBox, CheckedListBox	72
4.2. ვიზუალური ელემენტი ComboBox და თვისება DropDownStyle	83

თავი 5. მენიუს აგების ვიზუალური საშუალებანი	98
5.1. მთავარი მენიუს აგების ვიზუალური ელემენტები	98
5.2. გრაფიკული მენიუს აგების ვიზუალური ელემენტები...	104
5.3. კონტექსტური მენიუს აგების ვიზუალური ელემენტები.	107
5.4 სტატუსის პანელის შექმნის ვიზუალური ელემენტი	113
5.5. C# ენის ვიზუალური სტანდარტული დიალოგური საშუალებანი	115
თავი 6. რეკურსიული ფუნქციები და შემთხვევით რიცხვთა გენერატორი	122
6.1. ციკლები და რეკურსიული ფუნქციები	122
6.2. ციკლები და შემთხვევით რიცხვთა გენერატორი	134
თავი 7. პროგრამული პროექტის გამართვის ვიზუალური საშუალებები	145
7.1. სინტაქსური შეცდომების აღმოფხვრის საშუალებანი.....	147
7.2. გამონაკლისი შემთხვევები: შეცდომები პროგრამის შესრულებისას და მათი აღმოფხვრა	148
7.3. ლოგიკური შეცდომები და პროგრამის გამართვა ბიჯური შესრულების რეჟიმში	155
7.4. შესატან მონაცემთა კონტროლი	165
თავი 8. მონაცემთა ბაზებთან მუშაობის ვიზუალური საშუალებანი	169
8.1. ცხრილების წარმოდგენის მართვის ელემენტი DataGrid Wiew	169
8.2. ADO.NET: Visual C# კავშირი მონაცემთა ბაზებთან	179
8.3. ADO.NET მონაცემთა არქიტექტურა	180
8.4. მონაცემთა ბაზასთან მიერთება	185
8.5. C#.NET და Ms Access	187
8.6. SQL მოთხოვნების დაპროგრამირების საილიუსტრაციო მაგალითები C#-ში MsAccess ბაზისთვის	216

თავი 9. .NET პლატფორმის კონცეფციის რეალიზაცია	229
9.1. აპლიკაციის დაპროგრამება რამდენიმე ენის საფუძველზე .dll ფაილების შექმნით	229
9.1.1. კლასის შექმნა ახალ პროექტში C++.NET ენაზე	231
9.1.2. კლასის შექმნა ახალ პროექტში Visual Basic .NET ენაზე.	233
9.2. სასტარტო პროექტის დამატება Solution-ში C#.NET ენაზე	238
თავი 10. Workflow Foundation ტექნოლოგია .NET პლატფორმაზე	245
10.1. მარტივი ბიზნესპროცესის (Workflow-ის) აგება	245
10.2. პროცედურული ელემენტები	251
თავი 11. პროგრამული აპლიკაციის დიზაინი XAML ენაზე ..	264
11.1. XAML ენის საფუძვლები	264
11.2. დიზაინერის მართვა და XAML კოდი	273
თავი 12. ბიზნესპროცესების (Workflows) დაპროგრამება	283
12.1. კოდირებული ბიზნესპროცესები	283
12.2. ბიზნესპროცესის დიაგრამა (Flowchart Workflow)	296
12.3. პარალელური ბიზნესპროცესები და ქმედებები	305
ლიტერატურა	319

შესავალი

საინფორმაციო ტექნოლოგიები უახლესი კომპიუტერული ინდუსტრიის ბაზაზე განვითარების მაღალი ტემპებით ხასიათდება. სულ უფრო ფართოვდება კომპიუტერული სისტემების მომხმარებელთა წრე, რაც აქტუალურს ხდის ამ სფეროში მომხმარებელთა მეგობრული ინტერფეისების დამუშავებას, მობილური და სერვისული სისტემების მეთოდური საშუალებების შექმნას და განვითარებას [1,2].

დამხმარე სახელმძღვანელოში გადმოცემულია C#.NET ენის საფუძვლები და ვიზუალური დაპროგრამების მართვის კლასიკური ელემენტები [3]. განიხილება ობიექტორიენტირებული დაპროგრამების მეთოდის ძირითადი კომპონენტების პროგრამული რეალიზაციის ვიზუალური საშუალებანი. ასევე წარმოდგენილია C# ენის საფუძველზე ინტერფეისების დაპროექტება და კავშირები მონაცემთა ბაზების სისტემებთან, განსაკუთრებით MySQL და SQL Server და სხვა პაკეტებთან. განიხილება შესაბამისი საილუსტრაციო ამოცანები. აქ მნიშვნელოვანი ადგილი ეთმობა აგრეთვე C# ენის ვიზუალური ელემენტების გამოყენების საკითხებს გრაფიკული და ანიმაციური დაპროგრამების რეჟიმებისათვის.

წიგნში წარმოდგენილია ვიზუალური დაპროგრამების ახალი ინსტრუმენტი, როგორცაა WF.NET (Workflow Foundation) იგი გამოიყენება მომხმარებელთა ინტერფეისების ასაგებად და ეფუძნება მაღალი დონის ვიზუალური დაპროგრამების პრინციპებს [4,8,14]. WF ტექნოლოგია .NET-ში არის სრულიად ახალი პარადიგმა სამუშაო (ბიზნეს) პროცესებზე (workflow) ბაზირებული აპლიკაციების ასაგებად. იგი ფუნდამენტურად ახლად გააზრებული ტექნოლოგიაა. მასთან ერთად ხშირად გამოიყენება აგრეთვე WPF (Windows Presentation Foundation) და WCF (Windows Communication Foundation) ახალი ტექნოლოგიები [5,6], რომელთა შესწავლა ცალკე თემაა და შემდეგ ეტაპებზე ხორციელდება.

თავი 1

Ms Visual Studio.NET Framework პლატფორმა

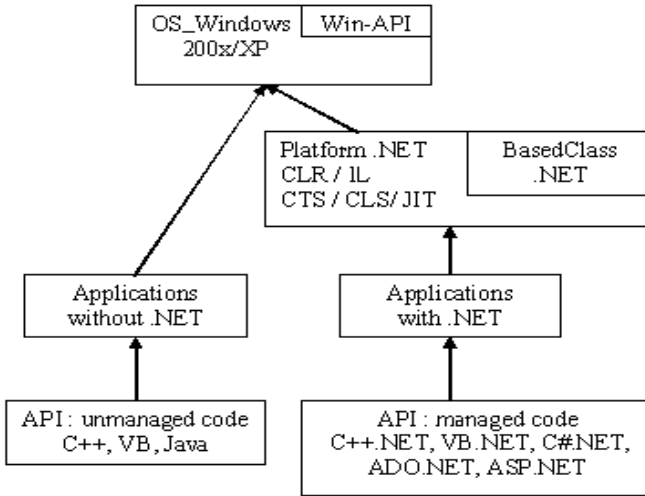
1.1. სამუშაო გარემო და C# აპლიკაციის პროექტის სტრუქტურა

Ms Visual Studio.NET Framework არის ინტეგრირებული პლატფორმა, რომელიც უზრუნველყოფს დესკტოპ- და ვებ-დანართების (Windows და Web-აპლიკაციების) განვითარებასა და გამოყენებას. .NET Framework პროგრამული პლატფორმის უმნიშვნელოვანესი ნაწილია CLR (Common Language Runtime) და CTS (Common Type System), რომელთა ბაზაზეც ხორციელდება პროგრამების ფუნქციონირების პროცესების მართვა და პლატფორმის სხვადასხვა ენებისათვის მონაცემთა საერთო ტიპების სისტემის მხარდაჭერა IL (Intermediate Language) ენის საფუძველზე.

.NET Framework პლატფორმა უზრუნველყოფილია კლასების მდიდარი ბიბლიოთეკით, მომხმარებელთა ინტერფეისულ ფორმებთან, მონაცემთა ბაზებთან და Web-სისტემებთან სამუშაოდ (Windows Forms, ADO.NET, ASP.NET, Windows Presentation Foundation (WPF) და სხვა [1-4,15].

Ms Visual Studio.NET Framework 4.5 მომხმარებელს აძლევს საშუალებას ე. წ. შუამავლის ფუნქცია შეასრულოს Windows ოპერაციულ სისტემასა და გამოყენებით პროგრამულ აპლიკაციებს შორის, რომელთა მიზანია რთული სისტემების მოდელირება, კონსტრუირება და რეალიზაცია უნიფიცირებული პროგრამირების კონცეფციის გამოყენებით (ნახ.1.1) [9,10].

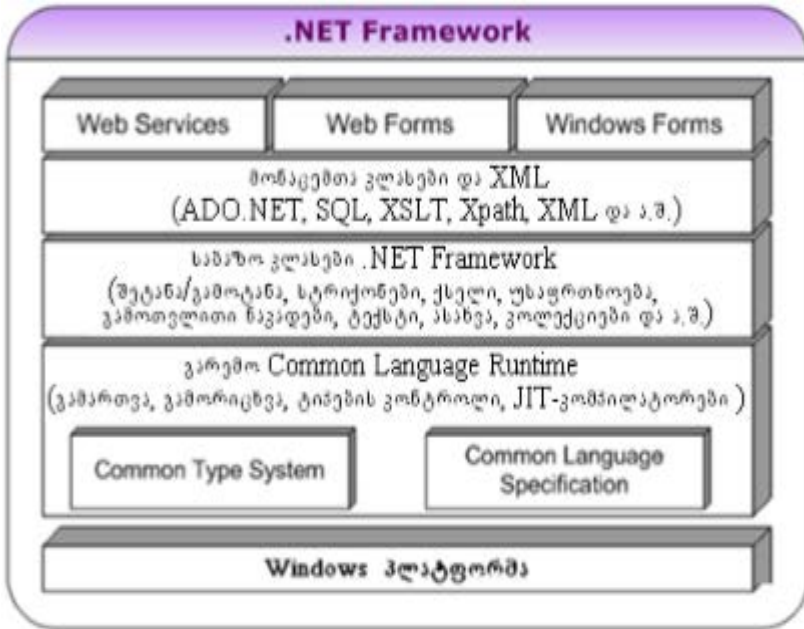
Windows-სისტემა უშუალოდ მუშაობს C++, VB, Java და სხვა ენებზე დაწერილ პროგრამულ API-დანართებთან (Application Programming Interface), რომლებიც რეალიზებულია როგორც უმართავი კოდები (unmanaged code). ამასთანვე იგი მუშაობს C#.NET, C++.NET, VB.NET და ა.შ [12].



ნახ.1.1

ზოგადად .NET-პლატფორმის მიერ მართვად (managed code) პროგრამულ დანართებთან. მართვაში იგულისხმება ის, რომ ეს კოდები ამუშავდება უშუალოდ .NET-ის მიერ, იმართება მათი პროცესები და მონაცემთა ნაკადები, მათ მიეწოდება შესასრულებლად საჭირო დამხმარე რესურსები და ა.შ.

.NET-პლატფორმა ასრულებს „ოპერაციული სისტემის“ გარკვეულ ფუნქციებს და მოქნილად ფუნქციონირებს Windows-თან. იგი სრულად ობიექტორიენტებულია, შედგება ობიექტთა ერთობლიობისგან, რომელთაგანაც თითოეულში რეალიზებულია განსაზღვრულ მეთოდთა ჯგუფები. მაგალითად, ფანჯრებისა და ფორმების ასახვა (Windows GUI), მონაცემთა ფაილებთან ურთიერთობა (ADO.NET), ვებ-გვერდების ორგანიზება და ინტერნეტთან კავშირი (ASP.NET) და სხვ. 1.2 ნახაზზე ნაჩვენებია .NET Framework-ის ზოგადი არქიტექტურა.



ნახ.1.2

მოცემული ნახაზის ერთ-ერთ ბლოკად ნაჩვენებია .NET-runtime - პლატფორმის სამუშაო გარემო (რომელშიც სრულდება პროგრამა), ანუ CLR(Common Language Runtime) და მას შესრულების საერთო გარემოსაც უწოდებენ. ესაა პროგრამული უზრუნველყოფა მომხმარებელთა გამოყენებითი პროგრამების შესასრულებლად. CTS საერთო ტიპების სისტემაა (Common Type System), რომლის საფუძველზეც NET-პლატფორმა უზრუნველყოფს დაპროგრამების სხვადასხვა ენის თავსებადობას. ამასთანავე CTS აღწერს მომხმარებელთა კლასების განსაზღვრის წესებსაც. IL შუალედური გარდაქმნის ენაა (Intermediate Language).

პროგრამები, რომელთა საწყისი კოდები დაწერილია, მაგალითად C#, C++ ან VB ენებზე .NET-ში, კომპილატორი ამ მართვად კოდებს გადაიყვანს შუალედურ IL-ენაზე, რომელთაც შემდეგ CTS სწრაფად აკომპილირებს მანქანურ კოდში [7]. ამგვარად,

ობიექტური კოდები IL-ენის საშუალებით ისე მიიღება, რომ მათში არაა დაფიქსირებული, თუ რომელ ენაზეა დაწერილი საწყისი კოდი. CLS ენის საერთო სპეციფიკაციაა (Common Language Specification), ანუ იმ სტანდარტების მინიმალური ერთობლიობა, რომელიც უზრუნველყოფს კოდებთან მიმართვას .NET-ის ნებისმიერი ენიდან. ამ ენების ყველა კომპილატორს გააჩნია CLS მხარდაჭერა.

JIT (Just-In-Time) ესაა შუალედური კოდის კომპილაციის ფაზა მანქანურ კოდში. სახელწოდება მიუთითებს იმაზე, რომ კოდის მხოლოდ იმ ცალკეული ნაწილების კომპილაცია ხდება, რომლებიც საჭიროა პროგრამის შესასრულებლად დროის მოცემულ მომენტში. .NET Framework -ში გამოიყენება 2-ეტაპიანი კომპილაცია:

1. ეტაპი: კომპილაცია MSIL-ენაში;

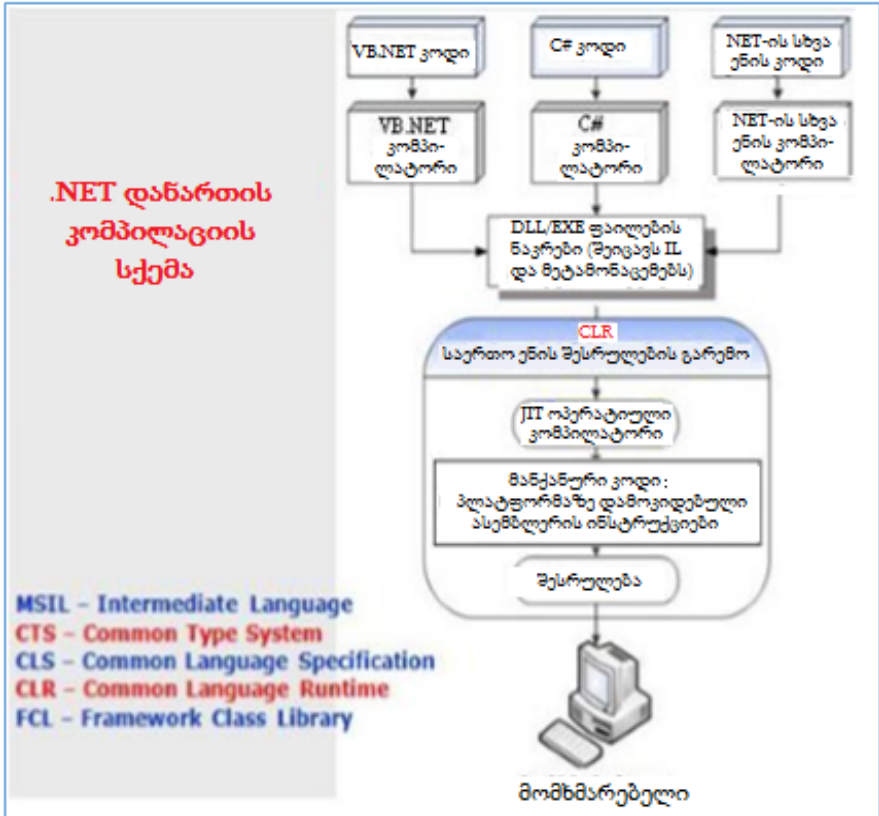
2. ეტაპი: “just-in-time” კომპილაცია უშუალოდ შესრულების პროცესში.

MSIL - ასემბლერული ენაა, რომელიც არაა დამოკიდებული მანქანაზე. ის სრულდება ყველგან, სადაც დაყენებულია CLR.

HTML-ისგან aspx-გვერდი განსხვავდება მასში სერვერული მართვის ელემენტების არსებობით, რომლებიც აღიწერება სპეცტეგებით. 1.3 ნახაზზე ნაჩვენებია სამომხმარებლო აპლიკაციის კომპილაციის სქემა .NET პლატფორმაზე

Visual C# 6.0 (2015 ვერსია) არის ვიზუალური, ობიექტ-ორიენტირებული ენა, რომელიც მაკროსოფტის ფირმამ .NET Framework 4.5/4.6 ინტეგრირებულ გარემოსთან ერთად, ბოლო ვერსიაში წარმოადგინა, Visual Basic.NET, Visual C++.NET, F#.NET და სხვა ენებთან ერთად.

Visual C# 2015 (შემდგომში C#) ენა მთლიანად მოიცავს მის წინამორბედს - Visual C# 2010-13 და ახალ გაფართოებებს.



ნახ.1.3

შესავალში აღწერილია ვიზუალური C# ენის საწყისები და პირველი პროგრამული კოდის აგების ელემენტები ვინდოუს-აპლიკაციისათვის. პრაქტიკული ექსპერიმენტებისათვის შესაძლებელია Visual C# 2015 Express Edition პაკეტის გამოყენება, რომელიც უფასოა და ინტერნეტიდან მისაწვდომი (ნახ.1.4):

<http://www.microsoft.com/visualstudio/en-us/products/2015-editions/visual-csharp-express>



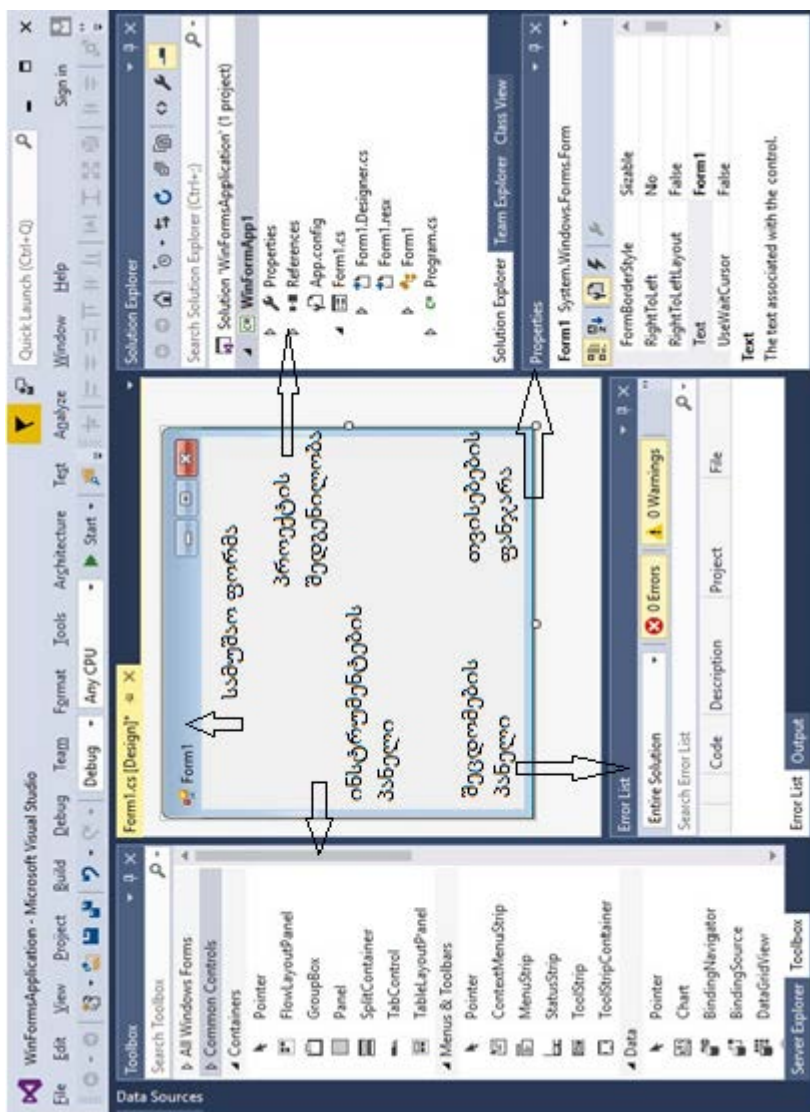
ნახ.1.4. Visual Studio.NET-ის ამუშავების დასაწყისი

იგი მოიცავს ტექსტურ რედაქტორს პროგრამული კოდის ასაგებად, კომპილატორს - მის გასამართად და დებაგერს - შეცდომების აღმოსაჩენად.

1.5 ნახაზზე ნაჩვენებია ახალი პროექტის შექმნისას მისი საწყისი მონაცემების (ენა, აპლიკაციის ტიპი, სახელი, მდებარეობა) განსაზღვრა. ბოლოს Ok ღილაკით გადავდივართ 1.6 ნახაზზე, რომელიც სამუშაო გარემო ანუ პროგრამისტის ინტერფეისია.

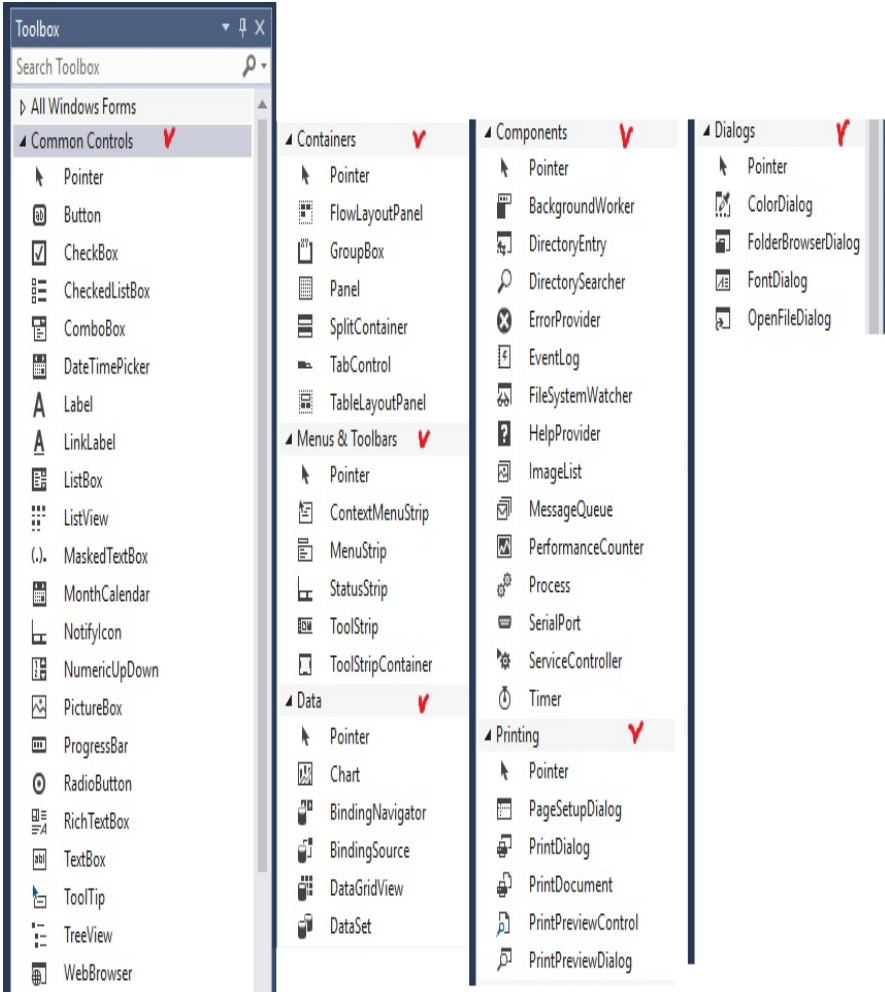


ნახ. 1.5. ახალი Visual C# პროექტის დასაწყისი



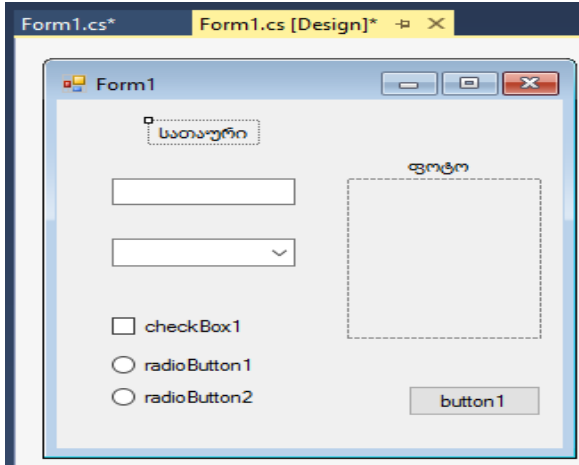
ნახ.1.6. სამუშაო გარემო

ინსტრუმენტების პანელის შედგენილობა მოცემულია 1.7 ნახაზზე. ინტერფეისის დიზაინის აწყობის დროს აქედან მაუსით გადაიტანება შესაბამისი პიქტოგრამა ფორმაზე და შემდეგ მისი თვისებები განისაზღვრება Properties- ფანჯარაში.



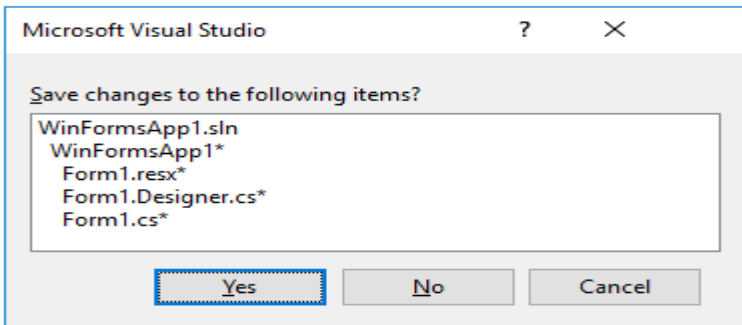
ნახ. 1.7. Toolbox პანელის შედგენილობა

1.8 ნახაზზე ნაჩვენებია საილუსტრაციო მაგალითი, რომლის ფორმაზეც განთავსებულია ინსტრუმენტების პანელის რამდენიმე ელემენტი. მათი და სხვა ელემენტების დანიშნულების შესწავლა ჩვენი დისციპლინის მიზანია.



ნახ.1.8. WindowsFormsApplication1 პროექტი

სამუშაო სეანსის დასრულებისას ხდება შედეგების შენახვის პროცედურის გააქტიურება (ნახ.1.9). მომხმარებლის თანხმობის შემთხვევაში ყველაფერი შეინახება Location-ით (ნახ.1.5) მითითებულ ადგილას.

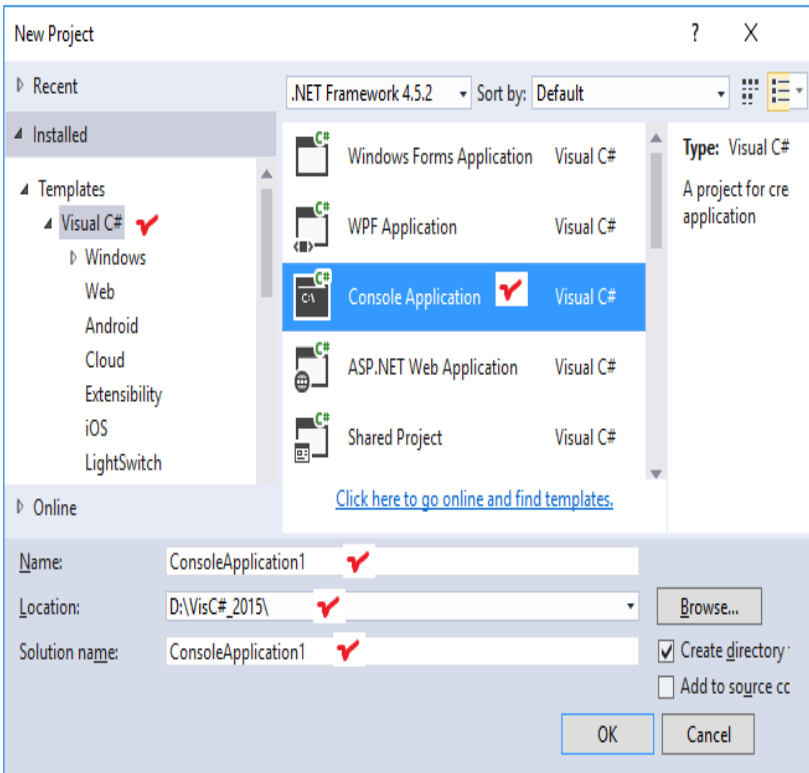


ნახ. 1.9. პროექტის მეტამონაცემების შენახვა

1.2. პროექტის აგება კონსოლის რეჟიმში C#-კოდით და მისი გამართვა

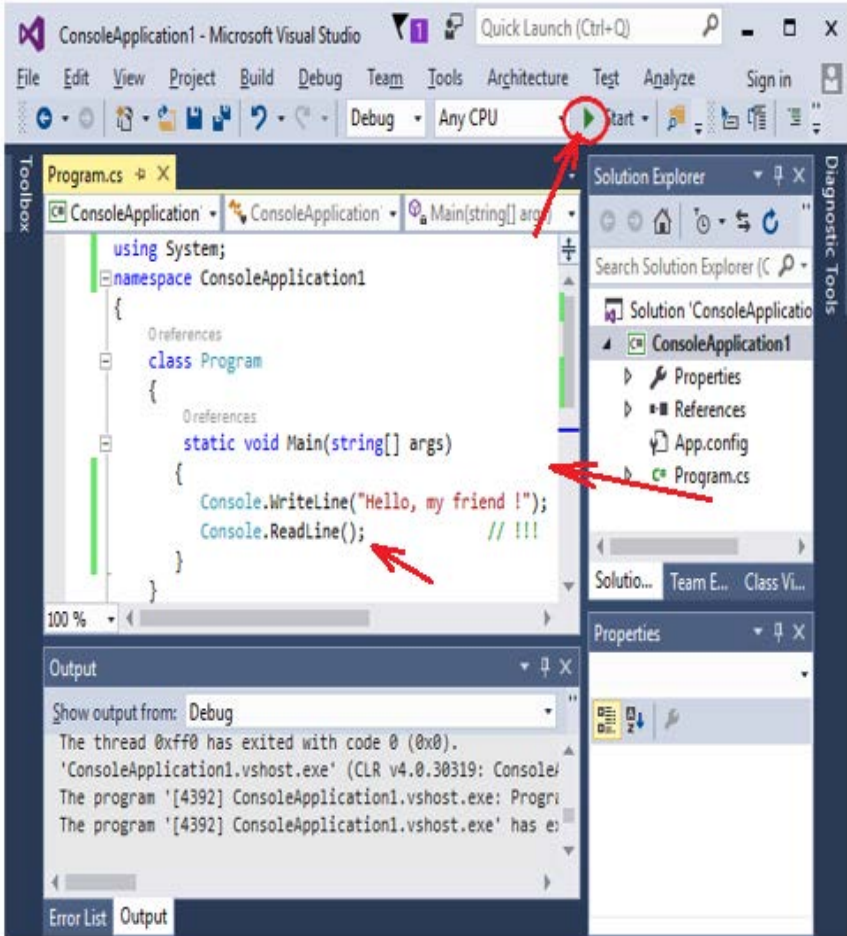
მიზანი: პირველი C# 6.0 კოდის აგება Visual Studio.NET Framework 4.5(4.6) ვერსიის კონსოლის რეჟიმში.

წინასწარ D:\ დისკზე შევქმნათ საკუთარი ფოლდერი მომავალი პროგრამების შესანახად. ავამუშავოთ Visual Studio.Net და კონსოლის რეჟიმში (Console Application) C# -ით შევქმნათ ახალი პროექტი ConsoleApp1 (ნახ.1.10).



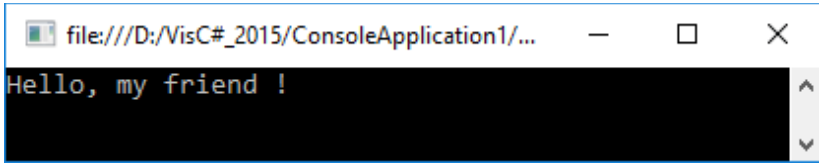
ნახ.1.10

სმოცანა 1.1: პირველი მარტივი პროექტის მაგალითი. შევიტანოთ static void Main(...) – კოდში ორი სტრიქონი (ნახ.1.11).



ნახ.1.11

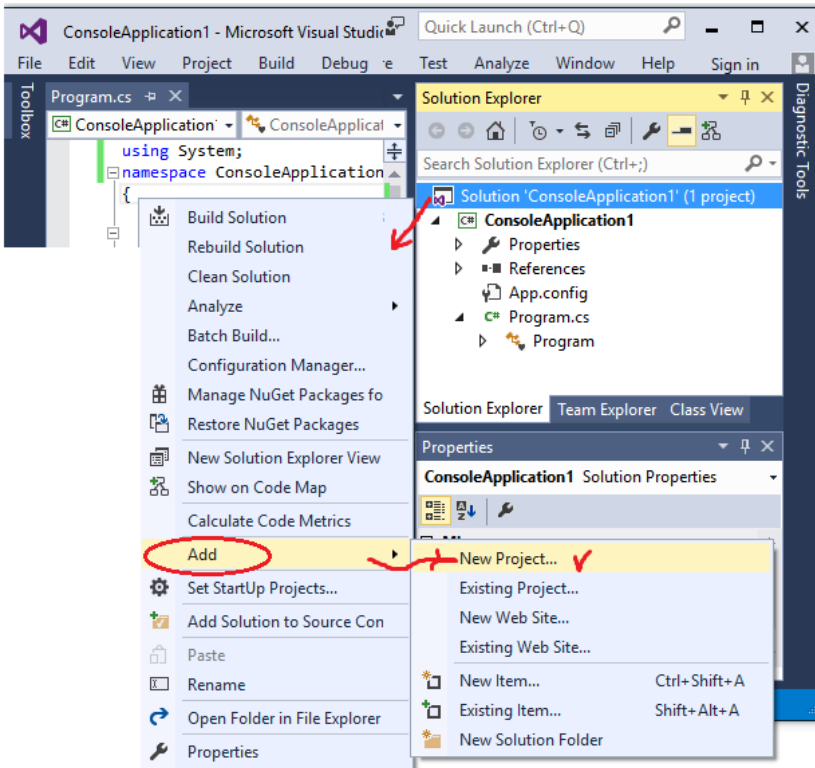
ავამუშავოთ პროგრამა და ვნახოთ შედეგი (ნახ.1.12).



ნახ.1.12

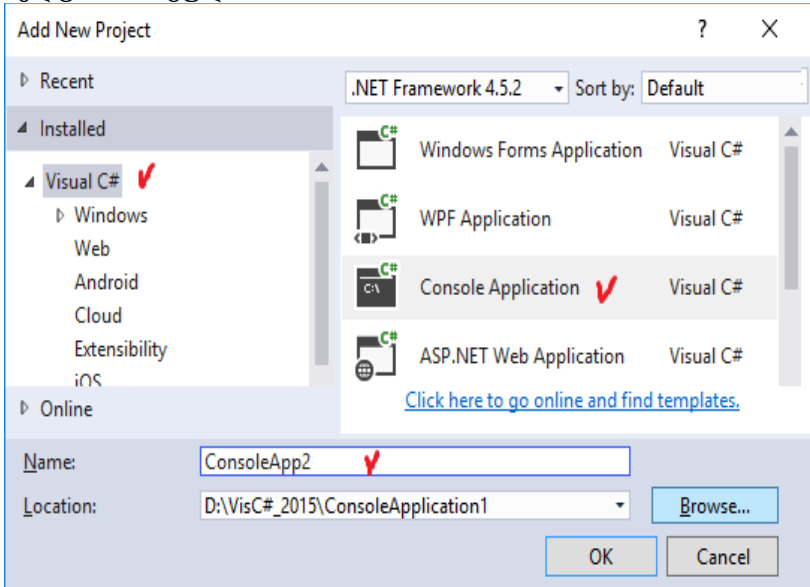
ამოცანა 1.2: ინტერაქტიული კოდის მაგალითი (მონაცემთა ტიპების გარდაქმნით).

Solution „Console Application“- ზე მათსის მარჯვენა ღილაკით დავამატოთ ახალი პროექტი (ნახ.1.13).



ნახ.1.13

სახელი შევირჩიოთ ასე ConsoleApp2 (ნახ.1.14). შევიტანოთ მასში C# კოდი (ლისტინგი_1.1), გვარის (Name), ასაკის (Age) და თვიური_ხელფასის (Money) მონაცემებით. შედეგად პროგრამამ გამოიტანოს კონსოლზე ეს საწყისი მონაცემები და წლიური ხელფასის მოცულობა.



ნახ.1.14

```
//--- ლისტინგი_1,1 ---
```

```
using System;
namespace ConsoleApp2
{
    class Program
    {
        static void Main(string[] args)
        {
            string Name; int Age=0; decimal Money=0.0m;
            // Input data -----
            Console.WriteLine("\aWhat is your name ? : ");
            Name = Console.ReadLine();
        }
    }
}
```

```

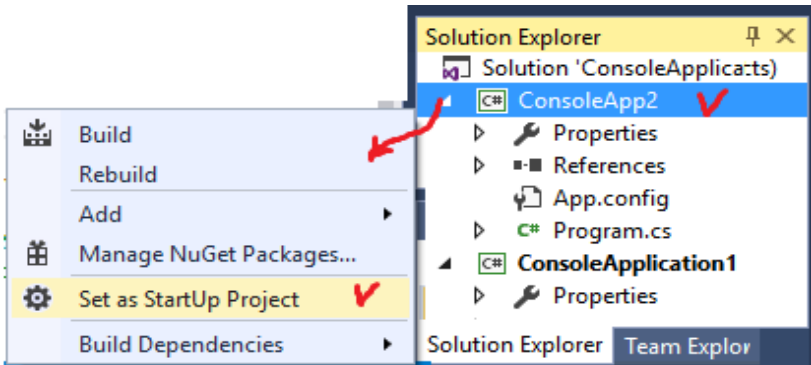
Console.Write("\aHow old are you ? : ");
Age = Convert.ToInt16(Console.ReadLine());
Console.Write("\aYour salary ? : ");
Money = Convert.ToDecimal(Console.ReadLine());
// Output results -----
Console.WriteLine("Hello, {0} !\n", Name);
Console.Write("Your age={0} years \n", Age);
Console.Write("Your money={0} dollars \n", Money);
Console.Write("In Year={0} dollars\n", Money*12);

Console.ReadLine();
} // პროგრამაში გამოყენებულია ტიპების გარდაქმნის
} // მეთოდები:Convert.ToInt16(),Convert.ToDecimal()
}

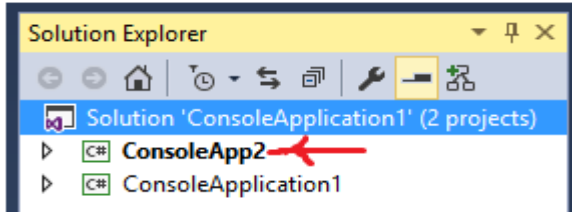
```

ავამუშავოთ პროგრამა, შევიტანოთ მონაცემები ინტერაქტიულ რეჟიმში (დიალოგში) და გავაანალიზოთ შედეგი.

შენიშვნა: შედეგი მივიღეთ ისევ წინა, 1.1 ამოცანისათვის ! Solution Explorer-ში გვაქვს ორი პროექტი. აქ აქტიურია ConsoleApp1, რადგან იგი მუქი ფერითაა აღნიშნული. საჭიროა გავააქტიუროთ ConsoleApp2 მაუსის მარჯვენა ღილაკით და Set as StartUp Project -ის არჩევით (ნახ.1.15). მიიღება შედეგი (ნახ.1.16).

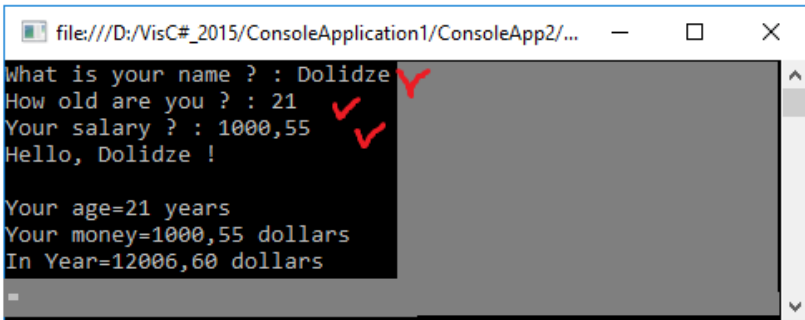


ნახ.1.15



ნახ.1.16.

კვლავ ავამუშავოთ პროგრამა და კონსოლზე უკვე მივიღებთ 1.2 ამოცანის დიალოგურ პროცედურას (ნახ.1.17).



ნახ.1.17

სავარჯიშო დავალება: ააგეთ პროექტი კონსოლის რეჟიმში C# კოდით, რომელიც ინტერაქტიულად შეგვატანიწებს ორ რიცხვს (a,b) და გაიანგარიშებს მათ ჯამს (S), სხვაობას (Dif), ნამრავლს (M), განაყოფს (D) და მოდულს (Mod), შედეგებს გამოიტანს კონსოლზე.

ამოცანა 1.3: დაწერეთ C# ინტერაქტიული კოდი მარტივი კალკულატორის მაგალითისათვის. დიალოგში შეიტანება ორი მთელი რიცხვი და ერთი არითმეტიკული ოპერაცია (+, -, * ან /). პროგრამას გამოაქვს გამოთვლის შედეგი. პროცესის გაგრძელება (ციკლური გამეორება) „Yes” ან დასასრული „No” (პროგრამიდან გამოსვლა).

Solution Explorer-დან ვამატებთ ახალ პროექტს ConsoleApp3 სახელით და მასში ვათავსებთ ამოცანის გადაწყვეტის შესაბამის კოდს. აქ გამოყენებულია ციკლის (while (...)) და გადამრთველის [switch(op), სადაც op - არითმეტიკული ოპერაციის კოდია] ოპერატორები. პროგრამის ტექსტი მოცემულია ლისტინგში_1.2.

///-- ლისტინგი_1,2 ---

```
using System;
namespace ConsoleApp3
{
    class Program
    {
        static void Main(string[] args)
        {
            int x, y, s;
            char op, yn;
            Console.Write("Calculation - y, End - n: ");
            yn = Convert.ToChar(Console.ReadLine());
            while (yn == 'y' || yn == 'Y')
            {
                Console.Write("Input \n");
                Console.Write("First number: ");
                x = Convert.ToInt32(Console.ReadLine());
                Console.Write("Second number: ");
                y = Convert.ToInt32(Console.ReadLine());
                Console.Write("Operacia: ");
                op = Convert.ToChar(Console.ReadLine());
                switch (op)
                {
                    case '+':
                        s = x + y;
                        Console.Write("Shedegi: ");
```

```

        Console.WriteLine("Sum = " + s);
        break;
    case '-':
        s = x - y;
        Console.Write("Shedegi: ");
        Console.WriteLine("Dif = " + s);
        break;
    case '*':
        s = x * y;
        Console.Write("Shedegi: ");
        Console.WriteLine("Prod = " + s);
        break;
    case '/':
        s = x / y;
        Console.Write("Shedegi: ");
        Console.WriteLine("Div = " + s);
        break;
    case '%':
        s = x % y;
        Console.Write("Shedegi: ");
        Console.WriteLine("Rest = " + s);
        break;
    default:
        Console.Write("operation not Correct !");
        break;
    }
    Console.Write("\nCalculation - y, End - n: ");
    yn = Convert.ToChar(Console.ReadLine());
}
Console.Write("End the process !");
}
}
}

```

შედეგების ფრაგმენტი მოცემულია 1.18 ნახაზზე.

```
file:///D:/VisC#_2015/ConsoleApplication1/ConsoleApp3/...
Calculation - y, End - n: y
Input
First number: 1000
Second number: 5
Operacia: *
Shedegi: Prod = 5000
-----
Calculation - y, End - n: y
Input
First number: 555
Second number: 222
Operacia: -
Shedegi: Dif = 333
-----
Calculation - y, End - n: y
Input
First number: 77
Second number: 6
Operacia: %
Shedegi: Rest = 5
-----
Calculation - y, End - n: y
Input
First number: 77
Second number: 6
Operacia: /
Shedegi: Div = 12
-----
Calculation - y, End - n: n
End the process !
```

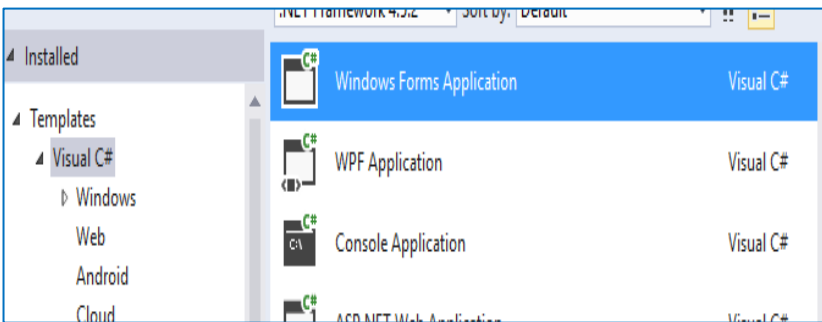
ნახ.1.18.

1.3. Windows ფორმებთან მუშაობა და ვიზუალური ელემენტები: Button, Label, TextBox

ამოცანა 1.4: ავაგოთ ორი ფორმა (Form1, Form2), შემდეგი ელემენტებით: Button, Label, TextBox. პირველი ფორმის ტექსტოქსში შევიტანოთ „სახელი და გვარი“. ამავე ფორმაზე ღილაკის - „მეორე ფორმა“ ამოქმედებით უნდა გაიხსნას მეორე ფორმა და მის ტექსტოქსში გამოჩნდეს პირველ ფორმაში შეტანილი სიტყვა.

მეორე ფორმის ტექსტოქსში შევცვალოთ სახელი. აქვე „დახურვა“ ღილაკის ამოქმედებით უნდა დაიხუროს მეორე ფორმა და პირველ ფორმას გადაეცეს კორექტირებული სტრიქონი (ახალი სახელით).

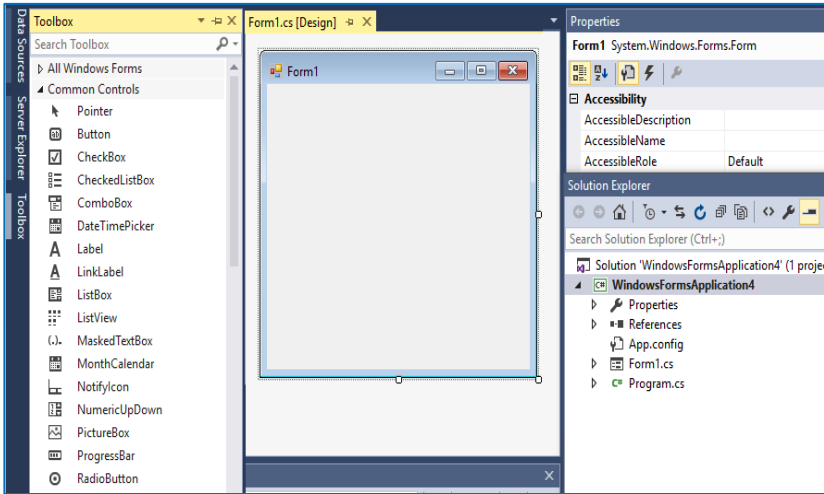
Visual Studio -ს სამუშაო გარემოდან New Project ის, პროექტის დასახელების და შენახვის კატალოგი განსაზღვრის შემდეგ, აპლიკაციის Windows Form Application გააქტიურებით (ნახ.1.19)



ნახ.1.19

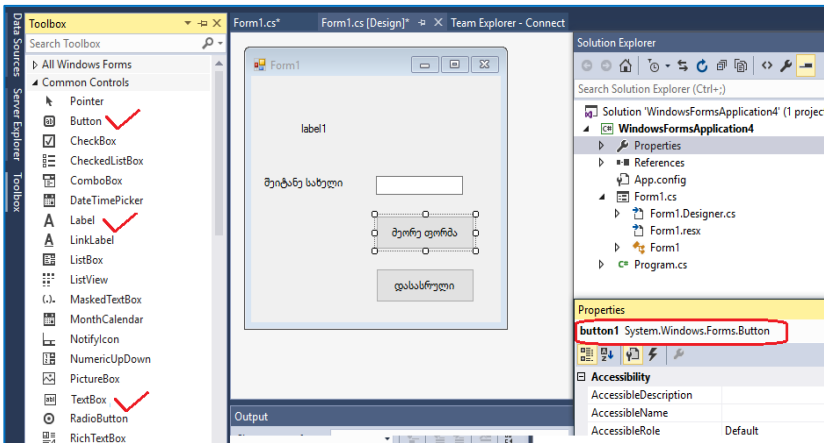
შესაძლებელია მომხმარებლის სამუშაო ფორმა Form1-ის და ToolBox-ინსტრუმენტების პანელის მიღება (ნახ.1.20):

ვიზუალური დაპროგრამება (C#.NET & Workflow Foundation NET)



ნახ.1.20

ინსტრუმენტების პანელიდან ხდება პროგრამისთვის საჭირო მართვის ელემენტის გადატანა ფორმაზე. 1.21 ნახაზის ქვედა მარჯვენა ნაწილში მოთავსებულია ფორმის ელემენტების თვისებათა (Properties) ფანჯარა. იგი ცალსახად აღწერს ფორმის ან მისი მონიშნული ელემენტ(ებ)ის თვისებებს



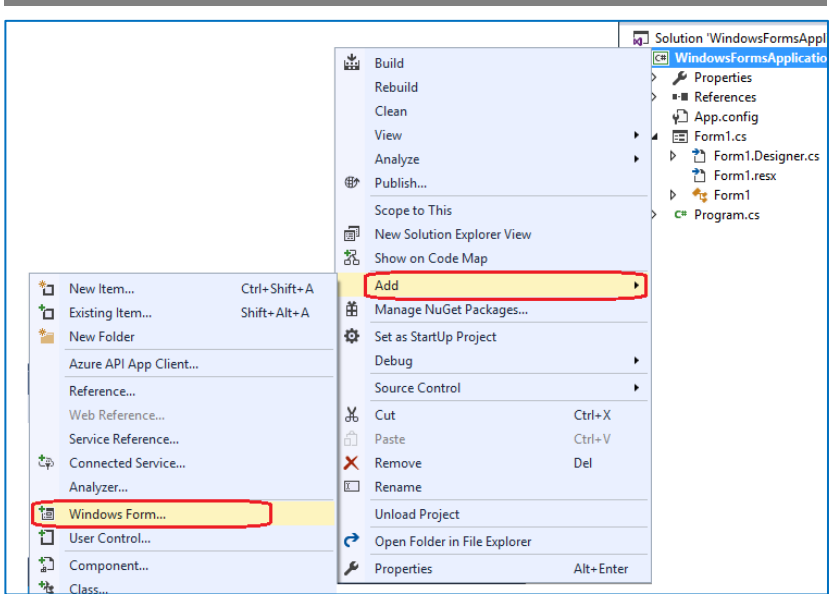
ნახ.1.21

```
// --- ლისტინგი_1.3 -----  
// WindowsFormsApplication4.cs---: 1-ელი ფორმისთვის -----  
using System;  
using System.Windows.Forms;  
  
namespace WindowsFormsApplication4  
{  
    public partial class Form1 : Form  
    {  
        public Form1()  
        {  
            InitializeComponent();  
        }  
  
        private void button1_Click(object sender, EventArgs e)  
        {  
            // შეტანილი სტრიქონის გამოტანა label1-ში -----  
            label1.Text = textBox1.Text;  
        }  
    }  
}
```

პროექტის შედგენილობა ასახულია Solution Explorer - ფანჯარაში, სადაც იერარქიული ხის სტრუქტურით, განთავსებულია მისი შემადგენელი კომპონენტები: ფორმები, რესურსები, პროგრამული კოდები (Program.cs) და ა.შ. აქ შესაძლებელია ახალი ელემენტების დამატება. მეორე ფორმის დამატებისთვის უნდა შესრულდეს შემდეგი ბრძანებები თანამიმდევრულად:

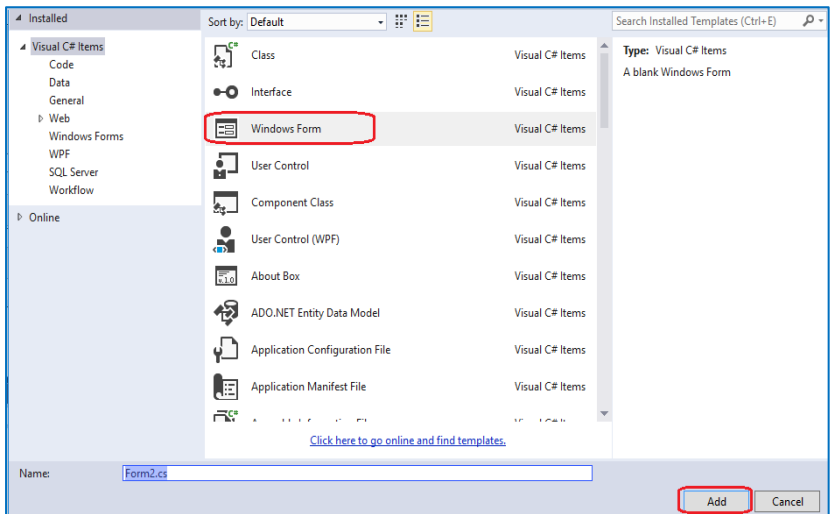
WindowsFormsApplication4 →Add→WindowsForm (ნახ.1.22)

ვიზუალური დაპროგრამება (C#.NET & Workflow Foundation NET)



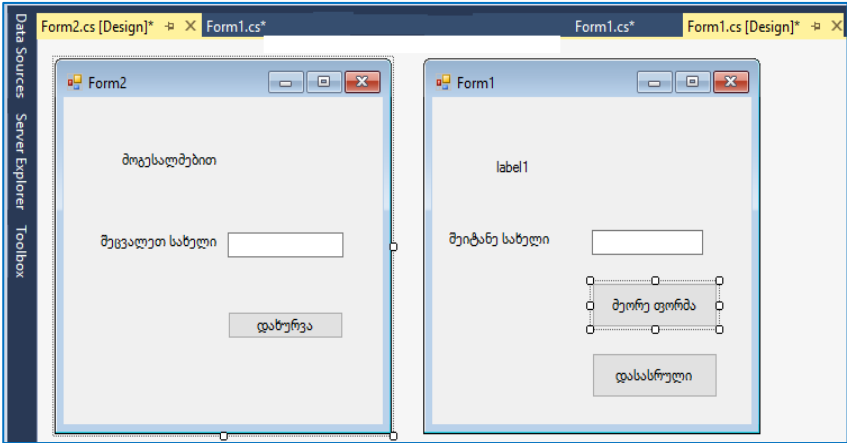
ნახ.1.22

შედეგად მიიღება ნახ.1.23 ფანჯარა:



ნახ. 1.23

მომხმარებელს ეძლევა საშუალება მეორე ფორმაზეც განათავსოს ელემენტები და ორივე ფორმასთან (ნახ.1.24) იმუშაოს ერთდროულად:



ნახ.1.24

```
private void button1_Click(object sender, EventArgs e)
{
    //--- შეტანილი სტრიქონის გამოტანა label1-ში -----
    label1.Text = textBox1.Text;
    // --- Form2-ის გახსნა Form1-დან -----
    Form2 f2 = new Form2(this); // !!! this
    f2.Show(); // Dialog(); //20
    f2.Controls["textBox1"].Text = textBox1.Text;
    // ----- შეტყობინების გამოტანა -----
    MessageBox.Show(textBox1.Text+" ,\n დახურეთ ეს ფანჯარა !");
}
private void button2_Click(object sender, EventArgs e)
{
    Close();
}
```

```
    }  
  }  
  // ---- WindowsFormsApplication4.cs----: მე-2 ფორმისათვის ---  
  using System;  
  using System.Windows.Forms;  
  
  namespace WindowsFormsApplication4  
  {  
    publicpartialclassForm2 : Form  
    {  
      Form1 f1;  
  
      public Form2(Form1 mainForm)  
      {  
        f1 = mainForm;  
        InitializeComponent();  
      }  
  
      privatevoid Form2_Load(object sender, EventArgs e)  
      {  
        textBox1.Text=f1.Controls["textBox1"].Text;  
      }  
      privatevoid button1_Click(object sender, EventArgs e)  
      {  
        f1.Controls["textBox1"].Text = textBox1.Text;  
        Close();  
      }  
    }  
  }  
}
```

ავამუშავოთ პროგრამული აპლიკაცია და შევასრულოთ მონაცემების შეტანის, კორექტირების და გამოტანის ფუნქციები.

1.4. ფორმაზე ვიზუალურ ელემენტებთან მუშაობა (Size, Location)

ვინდოუს-ფორმის დიზაინის სრუყოფისა და მმართველი ელემენტების ეფექტურად ასაგებად შესაძლებელია სისტემის სხვადასხვა საშუალებების გამოყენება. მაგალითად:

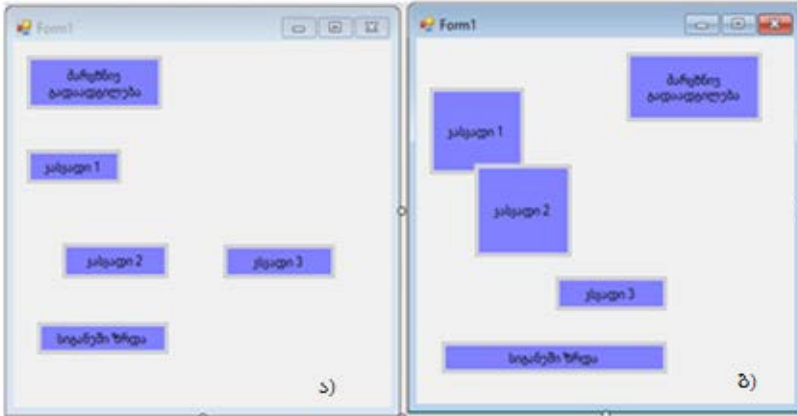
- *ელემენტების ფორმატირება:* ვერტიკალურად ან ჰორიზონტალურად თანაბარი (ან ჩვენთვის საჭირო) მანძილებით. ყველა ელემენტი შეიძლება ერთბაშად მოინიშნოს (Shift/Ctrl) და შემდეგ ჩატარდეს მათი თვისებების დაყენება. მაგალითად, ყველა ტექსტბოქსში ერთი ზომისა და ფერის ქართული ფონტის დაყენება;

- *ელემენტების კოპირება:* პროექტის სწრაფად ასაგებად ხელსაყრელია ერთხელ მომზადებული მმართველი ელემენტის მრავალჯერადი გამოყენება კოპირების საშუალებით (Ctrl+C და Ctrl+V). ამ დროს ელემენტის ყველა თვისება გადაიტანება ახალ, კოპირებულ ელემენტში და აღარაა საჭირო მათი ხელმეორედ დაყენება Properties-ში;

- *ელემენტთა თვისებების შეცვლა შესრულების პროცესში:* ფორმაზე ელემენტებს აქვს თვისებები Size: Width/Height (ზომა: სიგანე/სიმაღლე) და Location: X/Y (მდებარეობა: კოორდინატები ფორმის ზედა-მარცხენა კუთხიდან). სიდიდეები პიქსელებში მოიცემა. 1.25-ა,ბ ნახაზებზე მოცემულია ფორმა ხუთი ღილაკით (ა-რედაქტირების რეჟიმი, ბ-მუშაობის რეჟიმი) და კოდის ლისტინგი_1,4 შესაბამისი ღილაკებისთვის, რომელთა საშუალებითაც შესაძლებელია ბუტონების ზომის და მდებარეობის ცვლილება მუშაობის რეჟიმში;

- *ელემენტთა სახელები:* ყველა ელემენტს თავისი საკუთარი სახელი უნდა ჰქონდეს. რეკომენდებულია სახელის წინ სამი ასოს გამოყენება, რომელიც ელემენტის ტიპს და ფუნქციონალობას

მიუთითებს. მაგალითად, Label-სთვის lbl..., TextBox-სთვის txt..., Button-სთვის btn და ა.შ.;



ნახ.1.25

ამოცანა_1.5: ავარგოთ ფორმა (Form1) ხუთი ღილაკით (button) და ქართული წარწერებით (ნახ.1.15-ა). თითოეული ღილაკის სამუშაოდ დაწერეთ კოდი (მოვლენა - Event), რომელიც გადაადგილებს ან ზომებს შეუცვლის ამ ღილაკებს (ლისტინგი_1.4).

//--- ლისტინგი_1.4 ---

//--- ფორმის ელემენტების ზომის და მდებარეობის ცვლილება---

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsMultiButtons
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e) {}
        private void button1_Click(object sender, EventArgs e)
```

```

{ //კასკადი-1
    button1.Size = new Size(100, 100);
}
private void button2_Click(object sender, EventArgs e)
{ // გადაადგილება მარჯვნივ 20 პიქსელით
    button2.Location = new Point(button2.Location.X +20,
        button2.Location.Y);
}
private void button3_Click(object sender, EventArgs e)
{ // კასკადი-3
    button3.Location = new Point(120, 220);
}
private void button4_Click(object sender, EventArgs e)
{ // სივანეში გაფართოება 20-პიქსელით მაუსის ერთ დაკლიკვაზე
    button4.Size = new Size(button4.Size.Width + 20,
        button4.Size.Height);
}
private void button5_Click(object sender, EventArgs e)
{ // კასკადი-2
    button5.Size = new Size(100,100);
}
}
}

```

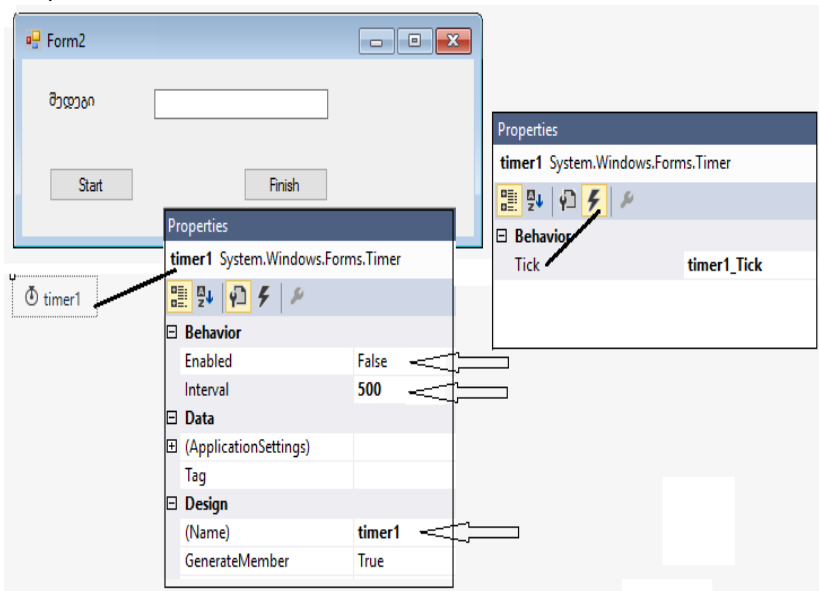
სავარჯიშო დავალება: ფორმაზე ტექსტურ ელემენტში მრავალსტრიქონიანი ტექსტის გადაბმა და ეკრანზე გამოტანა. „+“ სიმბოლო გამოიყენება სტრიქონების გადასაბმელად (concatenation). მაგალითად, label1-ელემენტში, რომლის სახელია lblText (Properties: Name), უნდა გამოიტანოთ „კასკადი-3“ ბუტონის ფორმაზე **მდებარეობის** და **ზომის** ამსახველი სტრიქონები. 1.26-ა ნახაზზე ნაჩვენებია ღილაკი, რომლის ამოქმედებით მიიღება ტექსტური შედეგი label1-ელემენტში (ნახ.1.26-ბ).



ნახ.1.26

1.5. მართვის ელემენტი Timer

კლასი Timer უზრუნველყოფს მოვლენის ამოქმედებას მომხმარებლის მიერ განსაზღვრულ ინტერვალში. ის გამოიყენება ვინდოუსის ფორმების აპლიკაციებში. მოვლენის სახელია Tick და ის ხდება მაშინ, როდესაც მოცემული დროის პერიოდი გავიდა და ტაიმერი ჩართულია. 1.27 ნახაზზე წარმოდგენილ პანელზე ჩანს ღილაკი Timer, რომლის ამუშავებითაც გამოიძახება Form2 და მომზადდება მასზე ორი ღილაკი: Timer-ის მოვლენის ამოქმედების (Start) და მოვლენის შეჩერების (Stop). ამ ღილაკების მანიპულირებით „შედეგში“ გამოიტანება „G“ სიმბოლოს კონკაქტუნაციით მიღებული სტრიქონი - ინტერვალით 500, რაც შეესაბამება 0.5 წამს.



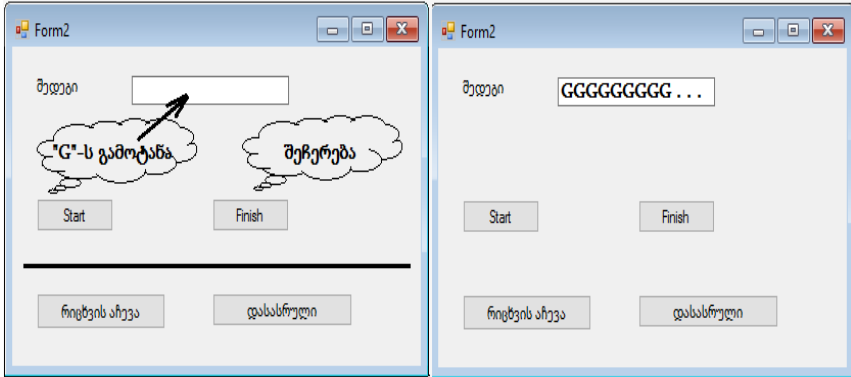
ნახ.1.27

ნახაზზე ნაჩვენებია Form2 თავისი ელემენტებით და თვისებებით. Timer გადმოიტანება კომპონენტების პანელიდან, იგი თავსდება ავტომატურად ფორმის ქვემოთ. მისი მონიშვნის შემდეგ

Properties-ში დაყენდება საჭირო მნიშვნელობები. პროგრამის ტექსტი მოცემულია 1.5 ლისტინგში, სადაც Form2-ზე განლაგებული Start, Stop ღილაკებიცაა აღწერილი. 1.28 ნახაზზე ნაჩვენებია საბოლოო შედეგის ამსახველი ფორმა.

// ლისტინგი_1.5 ---- Timer - ელემენტი -----

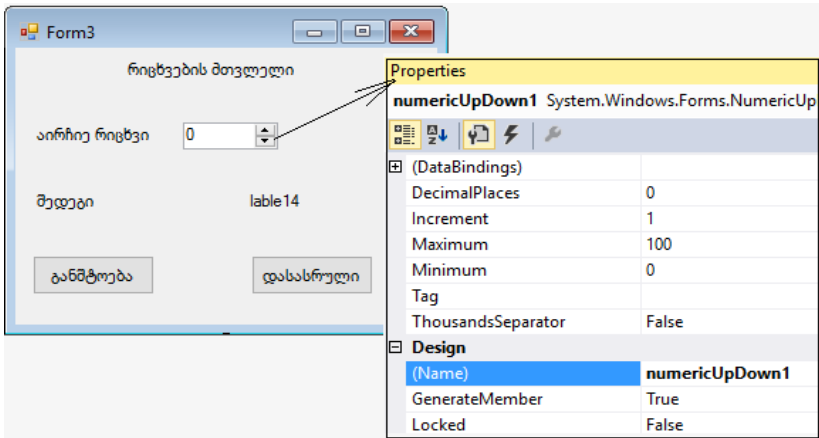
```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormPanel
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e) // Start
        {
            timer1.Enabled = true;
        }
        private void button2_Click(object sender, EventArgs e) // Stop
        {
            timer1.Enabled = false;
        }
        private void timer1_Tick(object sender, EventArgs e) // მოვლენის
            // ამუშავება და შედეგის გამოტანა
        {
            label1.Text += "G";
        }
        private void button7_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```



ნახ.1.28. საწყისი და საბოლოო შედეგის ფანჯრები

1.6. მართვის ელემენტი - რიცხვების არჩევა: **NumericUpDown**

ელემენტი NumericUpDown საშუალებას იძლევა შევარჩიოთ საჭირო რიცხვი მთვლელის რეჟიმში (პატარა ისრები მარჯვენა მხარეს, რომლებითაც ხდება საწყისი რიცხვის (Value) მატება ან კლება Increment-თვისებაში მითითებული ბიჯით) ან ავკრიფოთ იგი ხელით კლავიატურიდან (ნახ.1.29).



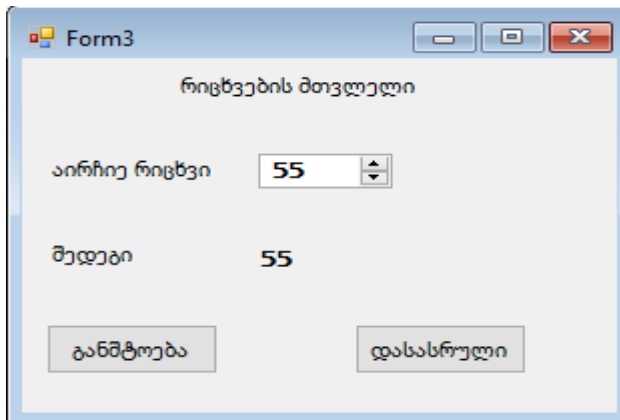
ნახ.1.29

Properties-ში მიეთითება აგრეთვე რიცხვის სავარაუდო Maximum და Minimum მნიშვნელობები. შედეგი აისახება label4-ში. კოდის ფრაგმენტი მოცემულია 1.6 ლისტინგში.

// ლისტინგი_1.6 – ValueChanged მოვლენა ----

```
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
    label4.Text = numericUpDown1.Value.ToString();
}
```

საბოლოო შედეგი მოცემულია 1.30 ნახაზზე.



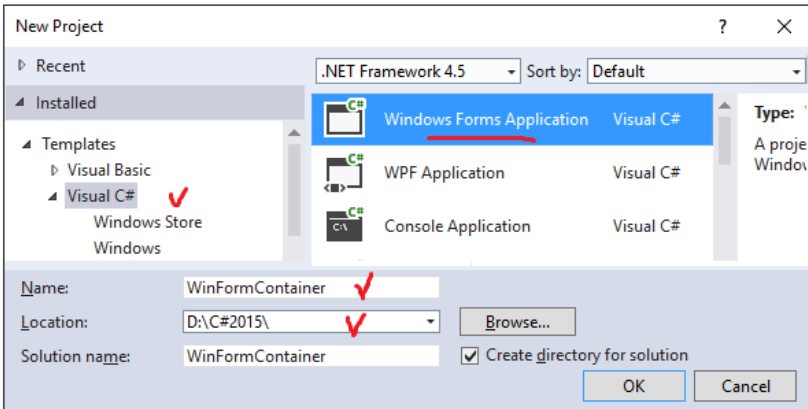
ნახ.1.30

სავარჯიშო დავალება: ააგეთ კალკულატორის შედარებით რეალური მოდელი. ელემენტებად გამოყენებულია დისპლეის ერთი textBox ელემენტი (Name=txtDisplay), რიცხვებისთვის 0,1,..9 button, +, -, *, /, % - ოპერაციებისთვის ოთხი button, „მცოცავი“ წერტილის „.“ – button, შედეგის ფიქსირების „=“ – button და დისპლეის გასუფთავების “C” (Clear) button.

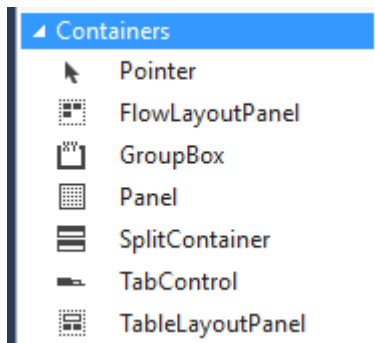
თავი 2

მართვის კონტეინერული ელემენტები

კონტეინერი ისეთი ელემენტია, რომელიც თვითონ შეიცავს სხვა ელემენტებს. მათ დიდი პრაქტიკული გამოყენება აქვს და ამ თავში ჩვენ სწორედ ამ საკითხებს გავეცნობით. 2.1 ნახაზზე ნაჩვენებია ახალი პროექტის შექმნის დასაწყისი, რომელშიც კონტეინერული ელემენტების (ნახ.2.2) სახესხვაობებს შევხებით.



ნახ.2.1. ახალი პროექტის შექმნა სახელით WinFormContainer

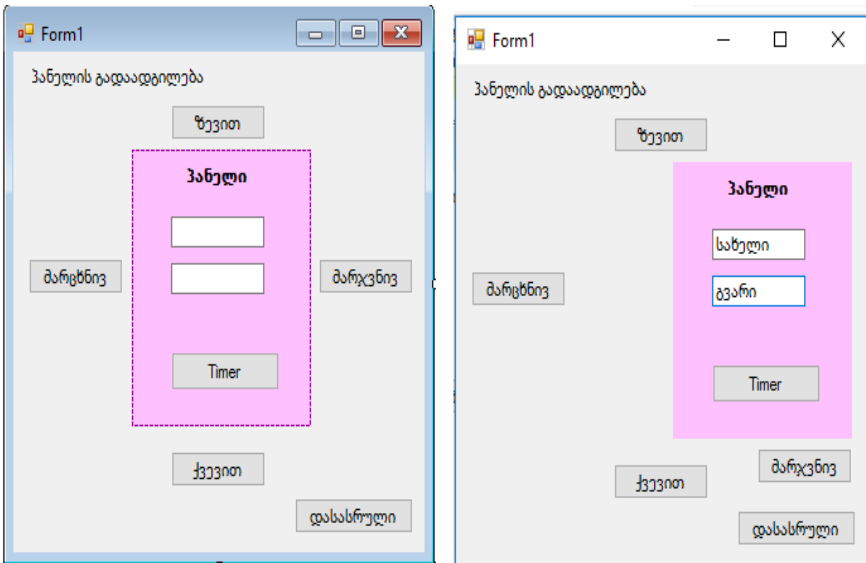


ნახ.2.2. Toolbox პანელის ფრაგმენტი Containers ელემენტებით

2.1. კონტეინერული ელემენტი - Panel

განვიხილოთ Panel ელემენტი და მისი გამოყენების მაგალითები. კონტეინერზე თავსდება მართვის ელემენტები, მაგალითად, label, textBox, comboBox და სხვ. პანელის გადაადგილებით ფორმის შიგნით იგი თან გაიყოლებს მასზე მოთავსებულ კომპონენტებსაც. შესაძლებელია პანელის კოპირება თავის ელემენტებიანა და გადატანა (paste) სხვა ფორმაზე.

ამოცანა_2.1: 2.3 ნახაზზე ილუსტრირებულია პანელის და ოთხი ღილაკის (მარცხნივ, მარჯვნივ, ზევით და ქვევით) მაგალითი. პანელის პარამეტრები (თვისებები) Properties-ში არის: BackColor= ControlDark // პანელის ფონის ფერი; Location=120,80 // პოზიცია ფორმაზე X=120, Y=80; Size=125,125 // პანელის ზომა პიქსელებში: სიგანე, სიმაღლე.



ნახ.2.3

ქვემოთ მოცემულ ლისტინგში მოცემულია პროგრამის კოდი, რომელშიც პანელის გადაადგილება ფორმაზე ხდება ლილაკების გამოყენებით. ნახაზზე ნაჩვენებია პანელისა და მისი „შიგთავსი“ ელემენტების საბოლოო მდებარეობა ფორმის ზედა მარჯვენა კუთხეში.

// ლისტინგი_2.1 – Panel-ის გადაადგილება ფორმაზე

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WinFormPanel
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e) // Top
        {
            panel1.Location = new Point(panel1.Location.X,
                                        panel1.Location.Y - 10);
        }
        private void button2_Click(object sender, EventArgs e) // Bottom
        {
            panel1.Location = new Point(panel1.Location.X,
                                        panel1.Location.Y + 10);
        }
        private void button7_Click(object sender, EventArgs e) // Left
        {
            panel1.Location = new Point(panel1.Location.X-10,
                                        panel1.Location.Y);
        }
        private void button4_Click(object sender, EventArgs e) // Right
```

```
{
    panel1.Location = new Point(panel1.Location.X + 10,
                                panel1.Location.Y);
}
private void button5_Click(object sender, EventArgs e)
{
    Close();
}
}
```

პროგრამის ტექსტში კონსტრუქტორი Point() ასრულებს კლასის ეგზემპლარის ინიციალიზებას ახალი X,Y კოორდინატებით, რომლის შემდეგაც ობიექტი panel1 გადაადგილდება ფორმაზე 10 პიქსელით მითითებული მიმართულებით.

სვარჯიშო დავალება:

ააგეთ პანელი ფორმის ცენტრში, მოათავსეთ მასზედ სხვა ელემენტები, მაგალითად, textBox და label და ამოძრავეთ ოთხივე მიმართულებით. პანელის მისვლისას ფანჯრის ნაპირებთან გაჩერდეს (არ დატოვოს ეკრანი).

2.2. კონტინერული ელემენტები: CheckBox, RadioButton, GroupBox

/ - CheckBox - საკონტროლო უჯრა ან ველია, რომელიც ცარიელი (გამორთულია) ან მონიშნულია (ჩართულია). რამდენიმე CheckBox-ის შემთხვევაში შეიძლება ჩართული იყოს 0, 1 ან ყველა.

/ - RadioButton - გადამრთველი ღილაკი ან ველია, რომლის ჩართვისას მასში თავსდება მრგვალი მარკერი. რამდენიმე RadioButton-ის შემთხვევაში მხოლოდ ერთია აქტიური, რომელიც მონიშნულია (2 ან მეტი არ შეიძლება).

GroupBox - არის აგრეთვე Container კლასის ჯგუფური საკონტროლო უჯრა, შემოსაზღვრული ჩარჩოთი, რომელშიც შეიძლება მოთავსდეს რამდენიმე CheckBox, RadioButton ან სხვა ვიზუალური ელემენტი. ის აერთიანებს ერთგვაროვან მონაცემებს, რომელთა ტექსტური იდენტიფიკაცია ხდება მისი ჩარჩოს ზედა-მარცხენა კუთხეში.

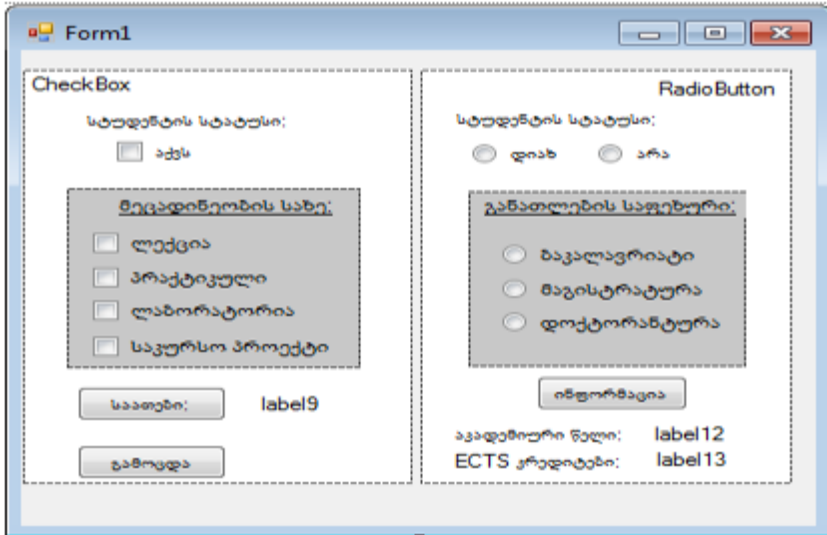
ამოცანა_2.2: საჭიროა ავაგოთ ვინდოუს-ფორმა, რომელსაც აქვს 2.4 ნახაზზე ნაჩვენები სახე. მარცხენაზე მოთავსებულია პანელი 4 checkBox-ით (მეცადინეობის სახე), 2 ღილაკი (საათები - ითვლის მეცადინეობის ჯამურ საათებს; გამოცდა - იძახებს Form2 ფორმას ახალი ქვეამოცანისათვის).

მარცხენა პანელის ზედა ნაწილში ჩანს checkBox – „აქვს“, რომელიც ახორციელებს სტუდენტის სტატუსის განსაზღვრას, ანუ თუ ეს checkBox გამორთულია, მაშინ სუბიექტი არაა სტუდენტი და მისთვის პანელი „მეცადინეობის სახე“ გამორთულია (ნახ.2.5).

თუ checkBox ჩართულია, მაშინ სუბიექტი სტუდენტია და შესაძლებელია პანელზე „მეცადინეობის სახე“ მაუსის დახმარებით ჩაირთოს 1,2 ან ყველა checkBox.

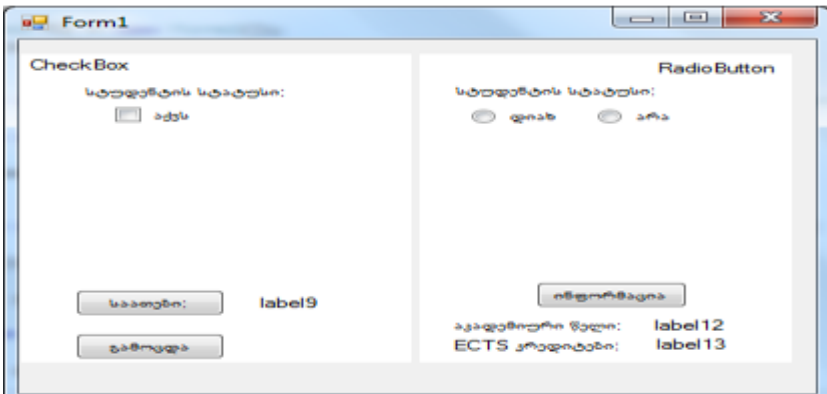
ბოლოს, საათების ღილაკის დახმარებით label9-ში გამოჩნდება სემესტრში ამ კონკრეტული საგნის საათების ჯამური რიცხვი. თუ რომელიმე checkBox-ს გამოვრთავთ, მაშინ ჯამური რიცხვი შემცირდება შესაბამისად.

მარჯვენა პანელზე ნაჩვენებია radioButton გადამრთველის მაგალითი. თუ სტუდენტის სტატუსის „დიახ“ ბუტონში არაა მარკერი, მაშინ პანელი „განათლების საფეხური“ სამი radioButton-ით გამორთულია (ნახ.7.4). თუ არის მარკერი, მაშინ ეს პანელი ხილვადია და შეიძლება ერთ-ერთი ბუტონის არჩევა. შემდეგ „ინფორმაცია“ - ღილაკის ამოქმედებით label12 და label13 უჯრებში გამოჩნდება შესაბამის მონაცემთა მნიშვნელობები: სასწავლო წლების რაოდენობა და ECTS-კრედიტების საერთო რაოდენობა. radioButton-ების გადართით შესაძლებელია სხვა ინფორმაციის მიღებაც.



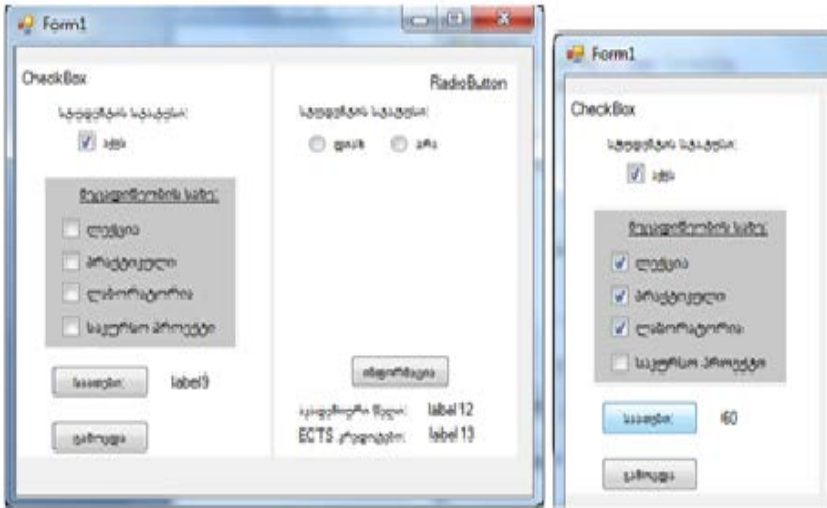
ნახ.2.4

აქ გამოყენებულია ვერტიკალური SplitContainer ორი პანელით.



ნახ.2.5

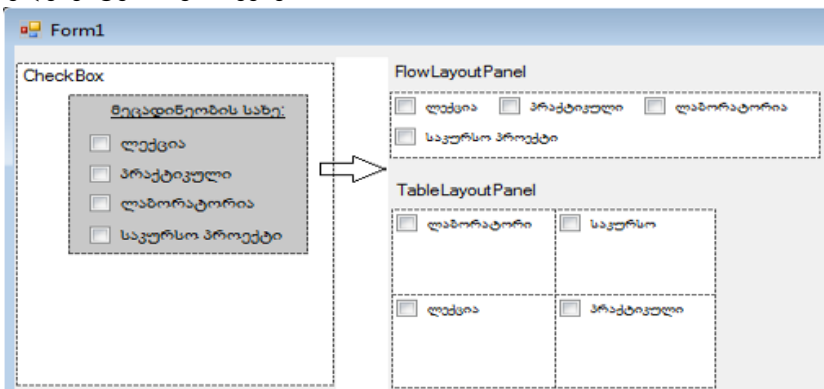
მარცხენა პანელზე ჩავრთეთ სტუდენტის სტატუსის checkBox. 2.6 ნახაზზე გამოჩნდება შედეგი:



ნახ.2.6

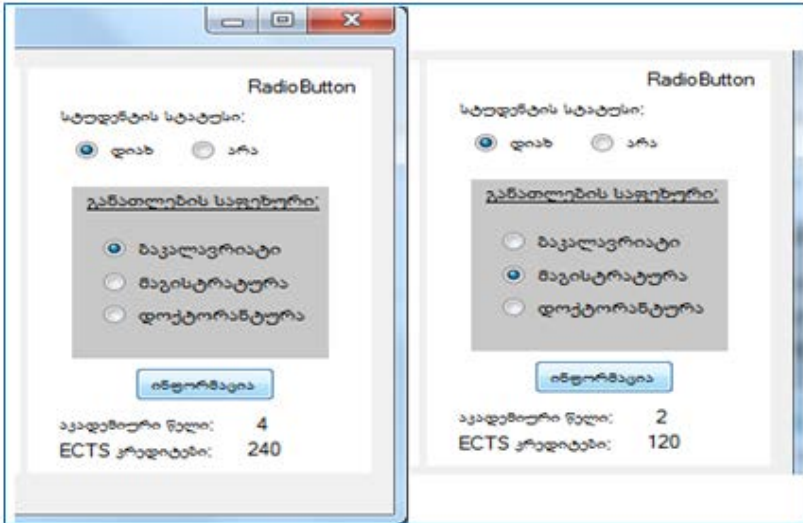
პანელზე „მეცადინეობის სახე“ ჩავრთოთ checkBox-ები „ლეცია“, „პრაქტიკული“ და „ლაბორატორია“. შემდეგ ღილაკით „სათები“ label9-ში გამოჩნდება რიცხვი (ნახ.2.6).

2.7 ნახაზზე ნაჩვენებია checkBox-ების განლაგების ვარიანტები RowLayoutPanel და TableLayoutPanel კონტეინერული ელემენტების გამოყენებით.



ნახ. 2.7

ახლა იგივე პროცედურები ჩავატაროთ radioButton-ის მაგალითზე პანელის მარჯვენა ნაწილში. აქ (ნახ.2.4) ჩავსვათ მარკერი „დიახ“-ში და გამოჩენილ „განათლების საფეხურების“ პანელზე ავირჩიოთ რომელიმე ბუტონი. შემდეგ ღილაკით „ინფორმაცია“ მივიღებთ 2.8 ნახაზზე ნაჩვენებ სურათს.



ნახ.2.8

აღნიშნული ამოცანის რეალიზაციის კოდი მოცემულია 2.2 ლისტინგზე.

// ლისტინგი_2.2 --- CheckBox, RadioButton, Checked,
// CheckedChanged, SplitContainer ----

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WinFormChekRadio
{
    public partial class Form1 : Form
    {
        public Form1()
        {
```

```

        InitializeComponent();
    }

private void button2_Click(object sender, EventArgs e)
{
    Form2 f2 = new Form2();
    f2.Show();
}

private void splitContainer1_Panel1_Paint(object
sender, PaintEventArgs e) { }

private void button1_Click(object sender, EventArgs e)
{
    Close();
}

// მოვლენა CheckChanged - ცვლის ელემენტების მდგომარეობას
private void checkBox1_CheckedChanged(object sender,
EventArgs e)
{
    // checkBox-ის ჩართვით გამოჩნდება პანელი
    if (checkBox1.Checked) // checkBox-ის checked
        // თვისება მდგომარეობის საკონტროლოდ
        {
            panel2.Visible = true;
        }
    else //checkbox-ის გამორთვით დაიმალება პანელი
    {
        panel2.Visible = false;
    }
}

// საათების ანგარიში
private void button4_Click(object sender, EventArgs e)
{
    int s = 0;
    if (checkBox2.Checked)
        s += 15;
    if (checkBox3.Checked)
        s += 15;
    if (checkBox5.Checked)
        s += 15;
}

```

```

        if (checkBox8.Checked)
            s += 30;

        label9.Text = s.ToString();
    }
    // რადიო-ბუტონის საილუსტრაციო კოდი
private void radioButton1_CheckedChanged(object sender,
        EventArgs e)
    {
        if (radioButton1.Checked) // radioButton-ში
            // მარკერის ჩასმით გამოჩნდება პანელი
        {
            panel1.Visible = true;
        }
        else // radioButton-დან მარკერის ამოშლით
            // დაიმალება პანელი
        {
            panel1.Visible = false;
        }
    }
    // ღილაკი „ინფორმაცია“ გამოაქვს შედეგი ----
private void button3_Click(object sender, EventArgs e)
    {
        if (radioButton2.Checked) // ჩადგმული if...else if...else
            // მაგალითი
        {
            label12.Text = " 4";
            label13.Text = "240";
        }
        else if (radioButton3.Checked)
        {
            label12.Text = " 2";
            label13.Text = "120";
        }
        else
        {
            label12.Text = " 3";
            label13.Text = "180";
        }
    }

```

```

    }
}
// ერთი რადიო-ბუტონიდან მეორეზე გადასვლისას
// სუფთავდება ძველი შედეგის მონაცემები
private void radioButton2_CheckedChanged(object sender,
    EventArgs e)
{
    label12.Text = " ";
    label13.Text = " ";
}
private void radioButton3_CheckedChanged(object sender,
    EventArgs e)
{
    label12.Text = " ";
    label13.Text = " ";
}
private void radioButton5_CheckedChanged(object sender,
    EventArgs e)
{
    label12.Text = " ";
    label13.Text = " ";
}
}
}

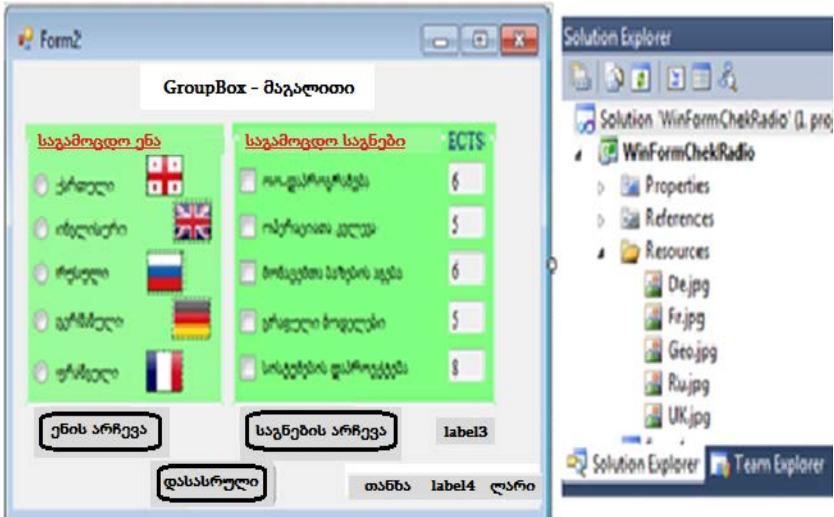
```

2.3. კონტეინერული ელემენტები: GroupBox და TabControl

➤ კონტეინერი GroupBox.

GroupBox - არის Container კლასის ჯგუფური საკონტროლო უჯრა, შემოსაზღვრული ჩარჩოთი, რომელშიც შეიძლება მოთავსდეს რამდენიმე CheckBox, RadioButton ან სხვა ვიზუალური ელემენტი. ის აერთიანებს ერთგვაროვან მონაცემებს, რომელთა ტექსტური იდენტიფიკაცია ხდება მისი ჩარჩოს ზედა-მარცხენა კუთხეში. GroupBox კონტეინერი მსგავსია ჩვენ მიერ ადრე განხილული Panel ელემენტისა, რომელზეც თავსდება სხვა ვიზუალური ელემენტები.

აქვე „გამოცდა“ ღილაკის საშუალებით გავხსნათ Form2 ფანჯარა და GroupBox, ChekBox, RadioButton ელემენტების გამოყენებით გადავწყვიტოთ პროგრამული კოდის (პროექტის) აგების ამოცანა „საგამოცდო საგნების შერჩევის“ შესახებ. ამასთანავე გათვალისწინებულ უნდა იქნას რამდენიმე ენაზე მუშაობის შესაძლებლობა. საწყისი ფორმა მოცემულია 2.9 ნახაზზე.



ნახ.2.9. ინტერფეისის საწყისი ფორმა

ნახაზზე დროშებისათვის (გრაფიკული ელემენტი) გამოიყენება PictureBox მართვის ელემენტი, რომლისთვისაც Resources ფაილებში (იხ. Solution Explorer) ჩასმულია წინასწარ მომზადებული შესაბამისი .jpg-ფაილები.

ელემენტები GroupBox-ის შიგნით განთავსებულია ამ ჯგუფის დასახელების შესაბამისად. ნახაზზე ნაჩვენებია ორი ჯგუფური ფანჯარა: „საგამოცდო ენა“ და „საგამოცდო საგნები“. საჭირო ენის შესაბამისი რადიო-ბუტონის არჩევის შემდეგ ღილაკით „ენის არჩევა“ გადავალთ საგამოცდო საგნების არჩევაზე, გავააქტიურებთ ჩვენთვის საჭირო CheckBox-ებს. თუ ენა შეიცვალა არა-ქართულით, მაშინ საგამოცდო საგნების GroupBox-ის და მისი თანმხლები სხვა

კომპონენტების წარწერებიც შეიცვლება არჩეული ენის შესაბამისად. მაგალითად, 2.10 და 2.11 ნახაზებზე ნაჩვენებია ინგლისური და რუსული ენების ვარიანტები.

საკამოცედო ენა	Courses	ECTS
<input type="radio"/> ქართული	<input checked="" type="checkbox"/> OO-Programming	6
<input checked="" type="radio"/> ინგლისური	<input checked="" type="checkbox"/> Operational research	5
<input type="radio"/> რუსული	<input checked="" type="checkbox"/> DB Systems building	6
<input type="radio"/> გერმანული	<input checked="" type="checkbox"/> Graph-modeling	5
<input type="radio"/> ფრანგული	<input checked="" type="checkbox"/> Projecting of Systems	8

ენის არჩევა Choice 30

დასასრული Money : 750 GLari

ნახ.2.10. ინგლისურენოვანი

საკამოცედო ენა	Предметы	ECTS
<input type="radio"/> ქართული	<input checked="" type="checkbox"/> OO-программирование	6
<input type="radio"/> ინგლისური	<input type="checkbox"/> Исследование операций	5
<input checked="" type="radio"/> რუსული	<input checked="" type="checkbox"/> Построение баз данных	6
<input type="radio"/> გერმანული	<input type="checkbox"/> Графовые модели	5
<input type="radio"/> ფრანგული	<input type="checkbox"/> Проектирование систем	8

ენის არჩევა Выбор 12

დასასრული Стоимость: 300 GLari

ნახ.2.11. რუსულენოვანი

label3-ში გამოიტანება სტუდენტის მიერ არჩეული საგნების ჯამური ECTS-კრედიტების რაოდენობა, label4-ში კი შესაბამისი გადასახდელი თანხის ოდენობა. შესაძლებელია საგნების მოკლება (checkBox-ის გამორთვით) ან ახლის დამატება (checkBox-ის ჩართვით). label3 და label4 უჯრებში მომენტალურად მოხდება გადაანგარიშება.

აღწერილი სისტემის შესაბამისი ელემენტების და ფუნქციონალობის პროგრამული კოდი მოცემულია 2.3 ლისტინგში.

// ლისტინგი 2.3 ---- GroupBox, CheckBox, RadioButton, ImageBox ---

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WinFormChekRadio
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

        private void checkBox1_CheckedChanged(object sender, EventArgs e) { }
        // ქართული ენის არჩევა
        private void button1_Click(object sender, EventArgs e)
        {
            if (radioButton1.Checked)
            {
                groupBox2.Text = "საგამოცდო საგნები";
                button2.Text = "აირჩიეთ";
                checkBox1.Text="ოპორგრამირება";
                checkBox2.Text="ოპერაციათა კვლევა";
                checkBox3.Text="მონაცემთა ბაზების აგება";
                checkBox9.Text="გრაფული მოდელები";
            }
        }
    }
}
```

```
checkbox5.Text="სისტემების დაპროექტება";
label5.Text = "თანხა :";
label6.Text = "ლარი";
}
else if (radioButton2.Checked) // ინგლისური ენის არჩევა
{
    groupBox2.Text = "Curses";
    button2.Text = "Choice";
    checkBox1.Text="OO-Programming";
    checkBox2.Text="Operational research";
    checkBox3.Text="DB Systems building";
    checkBox9.Text="Graph-modeling";
    checkBox5.Text="Projecting of Systems";
    label5.Text = "Money :";
    label6.Text = "GLari";
}
else if(radioButton5.Checked) // რუსული ენის არჩევა
{
    groupBox2.Text = "Предметы";
    button2.Text = "Выбор";
    checkBox1.Text="ОО-программирование";
    checkBox2.Text="Исследование операций";
    checkBox3.Text="Построение баз данных";
    checkBox9.Text="Графовые модели";
    checkBox5.Text="Проектирование систем";
    label5.Text = "Стоимость:";
    label6.Text = "GLari";
}
else // გერმანული ენის არჩევა
{
    groupBox2.Text = "Fachdisziplinen";
    button2.Text = "Wählen Sie bitte";
    checkBox1.Text = "OO-Programmierung";
    checkBox2.Text = "Operationsforschung";
    checkBox3.Text = "Entwicklung von Datenbanken";
```

```
        checkBox9.Text = "Graphische Modelen";
        checkBox5.Text = "Systemsentwurf";
        label5.Text = "Kosten :";
        label6.Text = "GLari";
    }
}

private void button2_Click(object sender, EventArgs e)
{
    // კრედიტების ანგარიში
    int s = 0, Sum=0 ;
    if (checkBox1.Checked)
        s += 6;
    if (checkBox2.Checked)
        s += 5;
    if (checkBox3.Checked)
        s += 6;
    if (checkBox9.Checked)
        s += 5;
    if (checkBox5.Checked)
        s += 8;
    // კრედიტების შესაბამისი თანხის ანგარიში
    Sum = s * 37;
    label3.Text = s.ToString();
    label9.Text = Sum.ToString();
}

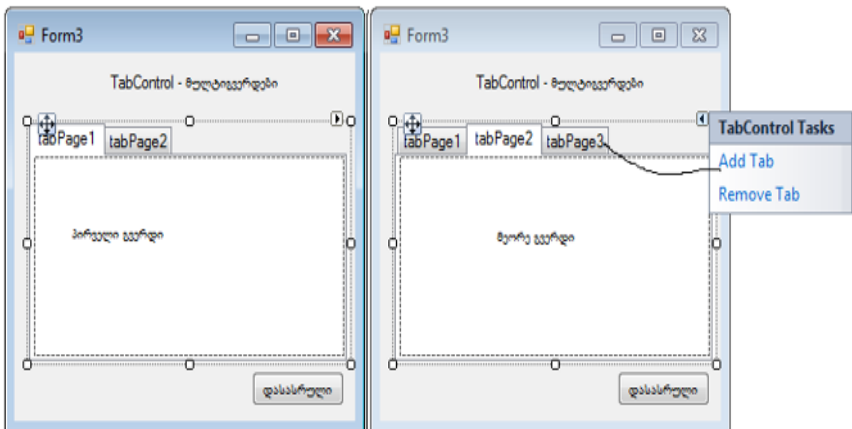
private void button3_Click(object sender, EventArgs e)
{
    Close();
}
}
}
```

➤ კონტეინერი TabControl

კონტეინერული ელემენტი TabControl გამოიყენება ფორმაზე მრავალგვერდიანი დიალოგის ორგანიზებისთვის ურთიერთ-გადაფარვის პრინციპით (ნახ.2.12).

ახალი გვერდის დამატება ხდება კონტექსტურ მენიუდან Add Tab, ხოლო ძველის წაშლა Remove Tab სტრიქონებით, რომელიც გამოიტანება მარჯვენა კუთხეში პატარა ისარზე მაუსის მარჯვენა ღილაკით.

თითოეული გვერდის სახელის ჩაწერა ხდება შესაბამისი გვერდის Properties-დან Text-ში.

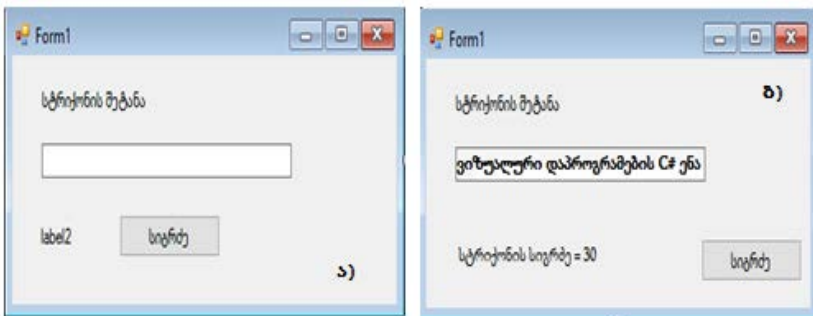


ნახ.2.12. კონტეინერი TabControl

თავი 3
სტრიქონული ტიპის მონაცემების
დამუშავება
3.1. სტრიქონული ტიპის (string) მონაცემები და
String კლასი

სტრიქონების დამახსოვრება ხდება String კლასით, რომლის სინონიმია string მონაცემთა ტიპი. String კლასის ობიექტები, ანუ სტრიქონები ფლობს მრავალ თვისებას და მეთოდს.

თვისება Length გამოიყენება სტრიქონის სიგრძის დასადგენად, ანუ რამდენი სიმბოლოსგან შედგება იგი. დავდოთ ფორმაზე სტრიქონის შესატანი textbox1 ველი და label2 - სტრიქონის სიგრძის მნიშვნელობის გამოსატანი ველი (ნახ.3.1-ა). ღილაკით „სიგრძე“ მოხდეს სტრიქონის სიგრძის დათლა და მიშვნელობის გამობეჭდვა. შედეგი მოცემულია 3.1-ბ ნახაზზე, ხოლო ღილაკის კოდი 3_1 ლისტინგზე.



ნახ.3.1

```
// ლისტინგი_3.1 --- String ----  
private void button1_Click(object sender, EventArgs e)  
{  
    string Striqoni;  
    Striqoni = textBox1.Text;  
    label2.Text = "სტრიქონის სიგრძე = "+Striqoni.Length;  
}
```

ამგვარად, შეტანილი სტრიქონი დამახსოვრებულ იქნა string ტიპის Striqoni ცვლადში. მისი სიგრძის ანგარიში კი მოხდა Striqoni.Length თვისების დახმარებით.

სტრიქონში სიმბოლოები ჩალაგებულია მასივის მსგავსად, ანუ ერთი სიმბოლო ერთი ელემენტია და თავისი ინდექსი აქვს (იხ.ცხრილში [ქეთო და კოტე]).

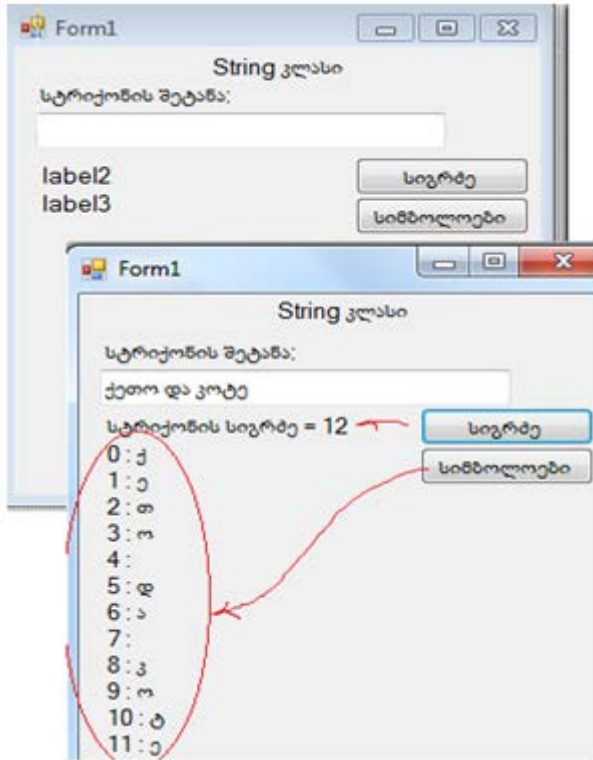
ქ	ე	თ	ო		დ	ა		კ	ო	ტ	ე
0	1	2	3	4	5	6	7	8	9	10	11

დავუმატოთ ფორმას ერთი ღილაკი „სიმბოლოები“ და მივაბათ მას კოდი, მოცემული 3.2 ლისტინგში.

// ლისტინგი_3.2 -- სიმბოლოები -----

```
private void button2_Click(object sender, EventArgs e)
{
    string Striqoni;
    char simbolo; // ერთი სიმბოლო
    int i; // ინდექსი
    Striqoni = textBox1.Text;
    label2.Text="სიმბოლოები";
    label3.Text="";
    for (i = 0; i < Striqoni.Length; i++)
    {
        simbolo = Striqoni[i];
        label3.Text += i.ToString()+" : "+ simbolo +"\n";
    }
}
```

შედეგი მოცემულია 3.2 ნახაზზე.



ნახ.3.2. შედეგები

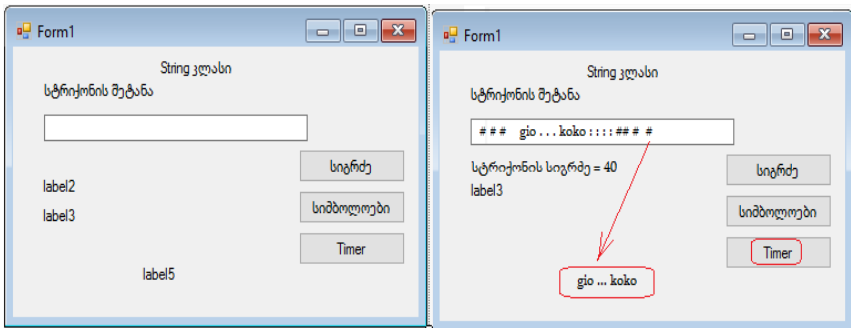
3.2. სტრიქონის დამუშავების მეთოდები: Trim() და Replace()

სტრიქონებთან მუშაობისას ხშირად ვხვდებით ტექსტში ცარიელ სიმბოლოებს (პრობლემებს), რომელთა დამუშავება (ამოყრა, შემცირება და ა.შ.) აუცილებელია. ამისათვის ბიბლიოთეკაში რამდენიმე მეთოდი:

- Trim() – არასასურველი სიმბოლოების ამოყრა სტრიქონის თავიდან ან ბოლოდან, მათ შორის პრობლემებისაც;

- TrimStart() – არასასურველი სიმბოლოების ამოყრა მხოლოდ სტრიქონის თავიდან;
- TrimEnd() – არასასურველი სიმბოლოების ამოყრა მხოლოდ სტრიქონის ბოლოდან.
- Replace() – ტექსტის შუაგულიდან არასასურველი სიმბოლოების (პრობლემებისა) ამოყრა ან შეცვლა.

3.3 ნახაზზე ნაჩვენებია Trim მეთოდის მაგალითი და მისი კოდის ფრაგმენტი 3.3_ლისტინგში.



ნახ.3.3

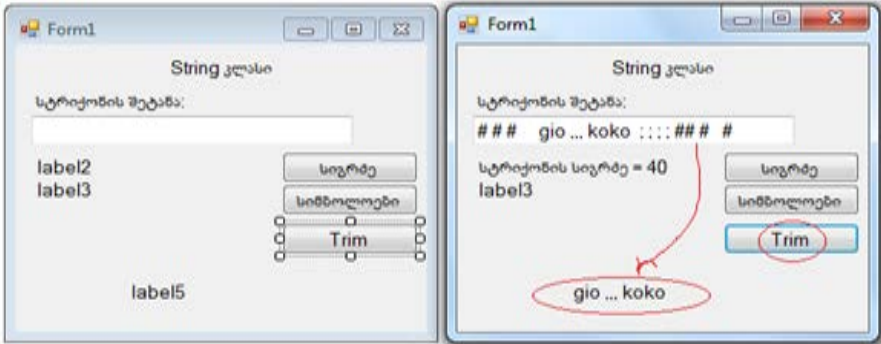
// ლისტინგი_3.3 --- Trim -----

```
private void button3_Click(object sender, EventArgs e)
{
    string Striqoni, SuftaStriqoni;
    Striqoni = textBox1.Text;
    SuftaStriqoni = Striqoni.Trim(' ', ';', '#');
    label5.Text = SuftaStriqoni;
}

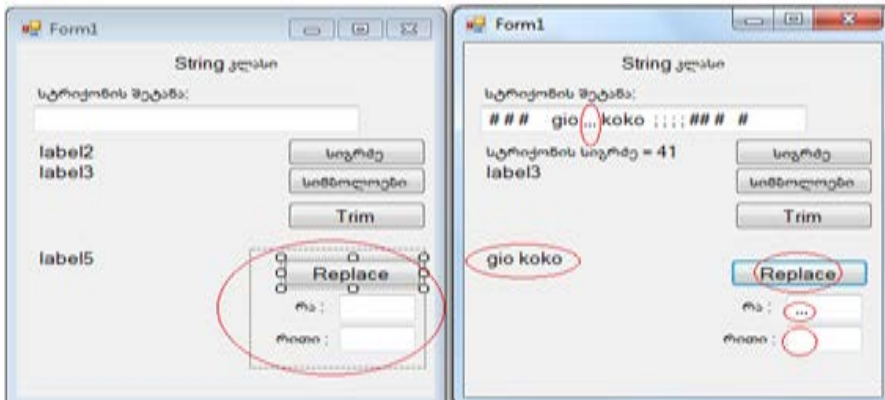
```

შედეგად მიიღება 3.4 ნახაზზე მოცემული ფანჯარა.

როგორც შედეგიდან ჩანს, სტრიქონის თავში და ბოლოში აღარაა ზედმეტი სიმბოლოები. დარჩა მხოლოდ „ „, „ „, სტრიქონის შუაში. ახლა Replace() მეთოდი (ლისტინგი 3.4) გამოვიყენოთ, რისთვისაც ფორმაზე დავამატოთ ეს ღილაკი (ნახ.3.5).



ნახ.3.4



ნახ.3.5

// ლისტინგი_3.4 --- Trim + Replace ----

```
private void button4_Click(object sender, EventArgs e)
{
    string Striqoni, SuftaStriqoni, Ra, Riti;
    Striqoni = textBox1.Text;
    Ra = textBox2.Text;
    Riti = textBox3.Text;
    SuftaStriqoni = Striqoni.Trim(' ', ';', '#');
    SuftaStriqoni = SuftaStriqoni.Replace(Ra, Riti);
    label5.Text = SuftaStriqoni;
}
}
```

Replace() მეთოდი მოითხოვს ორი დამატებითი ტექსტბოქსის გამოყენებას. რომლებშიც მიეთითება სტრიქონის შიგნით „რა“ უნდა შეიცვალოს „რიით“.

სავარჯიშო დავალება: ააგეთ პროექტი C# კოდით, რომელიც ტექსტბოქსში შეტანილ თქვენი „სახელის, ... , ;;; და გვარის“ სტრიქონს დაამუშავებს: დათვლის სიგრძეს, დაშლის სიმბოლოებად, გაასუფთავებს ზედმეტი ნიშნებისგან და კვლავ დათვლის სიგრძეს.

3.3. სტრიქონში ძებნის მეთოდები: IndexOf(), LastIndexOf() და IndexOfAny()

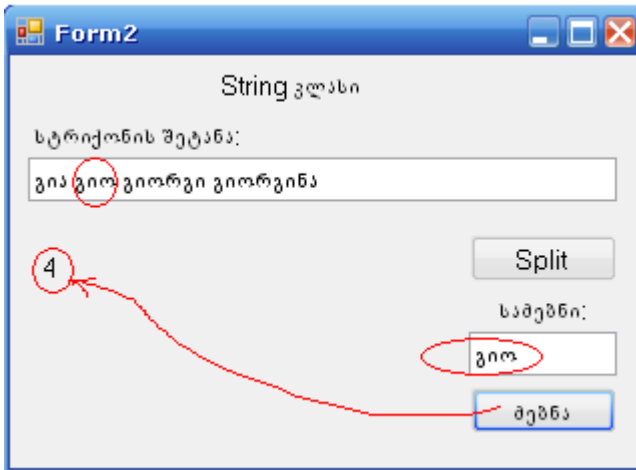
განვიხილოთ სტრიქონებში და ქვესტრიქონებში სიმბოლოთა ძებნის მეთოდები IndexOf(), LastIndexOf() და IndexOfAny(), რომლებიც მეტად მნიშვნელოვანი და პრაქტიკაში ხშირად გამოყენებადი ფუნქციებია.

ფორმაზე ძირითად სტრიქონთან (textBox1) ერთად მოვათავსოთ სამეზბნი ქვესტრიქონი (textBox2) და დილაკზე „ძებნა“ მივაბათ 3,5_ლისტინგის კოდი.

// ლისტინგი_3,5 --- IndexOf () ---

```
private void button2_Click(object sender, EventArgs e)
{
    string Striqoni, Sazebni;
    int pozicia;
    label2.Text = "";
    Striqoni = textBox1.Text;
    Sazebni = textBox2.Text;
    pozicia = Striqoni.IndexOf(Sazebni);
    label2.Text += pozicia;
}
```

3.6 ნახაზზე ნაჩვენებია „გია გიო გიორგი გიორგინა“ სტრიქონში (Striqoni) სამეზნი ქვესტრიქონის „გიო“ (Sazebni) მდებარეობის პოზიცია (=4). ესაა სამეზნი ქვესტრიქონის პირველი ნაპოვნი პოზიცია. თუ პოზიცია 0-ია, მაშინ ის მიუთითებს ძირითადი სტრიქონის დასაწყისზე, ხოლო თუ იგი ”-1”, მაშინ ასეთი ქვესტრიქონი ვერ მოიძებნა.



ნახ.3.6

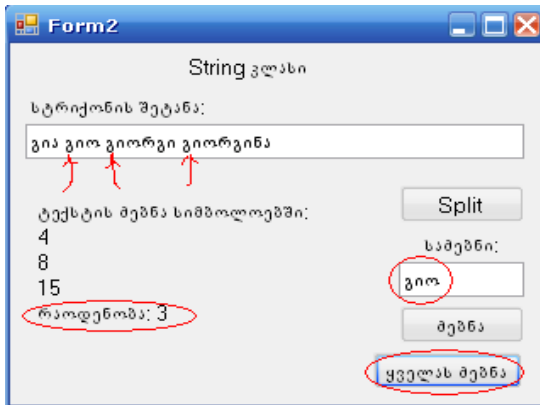
თუ საჭიროა სამეზნი ქვესტრიქონის ძირითად სტრიქონში არსებობის ყველა შემთხვევის დაფიქსირება, მაშინ მეზნის ალგორითმი მიიღებს 3.6_ლისტინგში მოცემული კოდის სახეს, ხოლო შედეგები იქნება 3.7 ნახაზზე მოცემული.

```
// ლისტინგი_3.6 --- All IndexOf ( ) ----
private void button3_Click(object sender, EventArgs e)
{
    string Striqoni, Sazebni;
    int pozicia, zebnisDackeba=0, raod=0;
    label2.Text = "";
    Striqoni = textBox1.Text;
```

```

Sazebni = textBox2.Text;
label2.Text = "ტექსტის მებნა სიმბოლოებში: " + "\n";
do
{
    pozicia = Striqoni.IndexOf(Sazebni, zebnisDackeba);
    zebnisDackeba = pozicia + 1;
    if (pozicia != -1)
    {
        label2.Text += pozicia + "\n";
        raod++;
    }
}
while (pozicia != -1);

label2.Text += "რაოდენობა: " + raod;
}
    
```



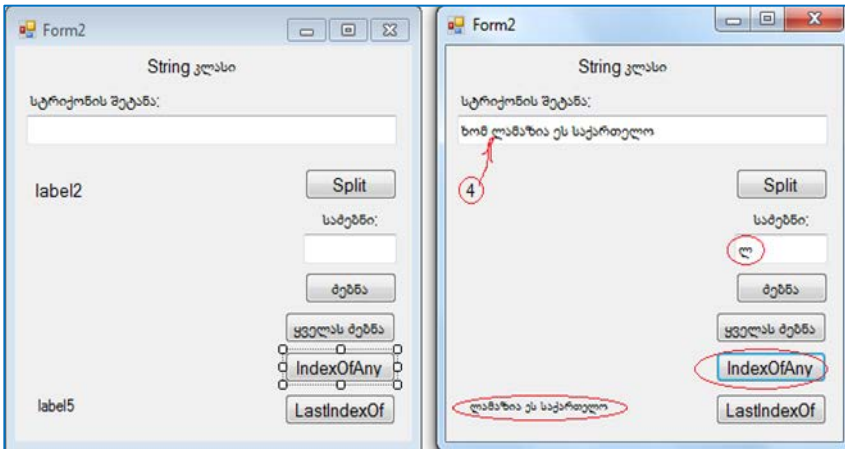
ნახ.3.7

როგორც ვხედავთ, do...while ციკლის შიგნით ხდება სამეზნი ქვესტრიქონის მრავალჯერადი მოძიება ძირითად სტრიქონში. საწყისი სამეზნო პოზიცია 0-ია, შემდეგი კი განისაზღვრება ყოველი ახალი ნაპოვნი პოზიციის შემდეგ. პროგრამა იმახსოვრებს ასევე ნაპოვნი შემთხვევების რაოდენობას.

განვიხილოთ მაგალითი IndexOfAny() მეთოდისთვის, რომლის დანიშნულებაცაა სტრიქონში პირველი ქვესტრიქონის პოვნა მითითებული სიმბოლოთი. 3.8 ნახაზზე ნაჩვენებია ეს შემთხვევა, ხოლო 3.7_ლისტინგში IndexOfAny დილაკის კოდი.

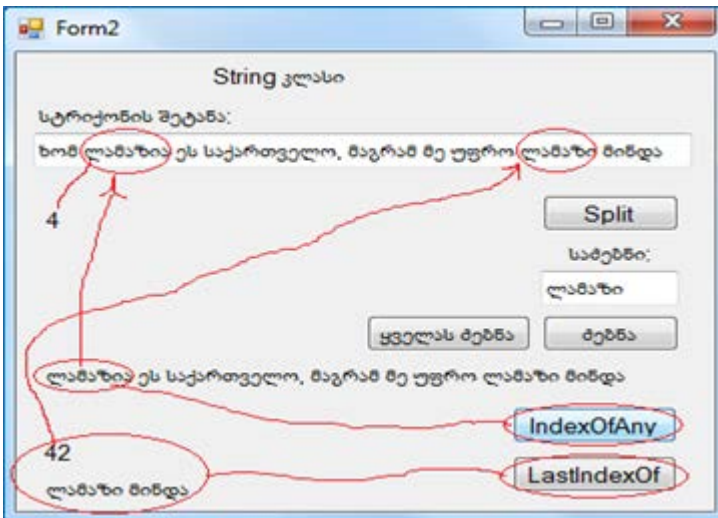
// ლისტინგი_3.7 --- IndexOfAny ----

```
private void button4_Click(object sender, EventArgs e)
{
    string Striqoni;
    int pozicia;
    label2.Text = "";
    Striqoni = textBox1.Text;
    textBox2.Text = "ლ"; // ან "ს"
    pozicia = Striqoni.IndexOfAny(new char[] { 'ლ', 'ს' });
    //pozicia = Striqoni.IndexOfAny(new char[] { 'კ', 'ს' });
    label2.Text += pozicia;
    label5.Text = Striqoni.Substring(pozicia);
}
```



ნახ.3.8

ახლა განვიხილოთ LastIndexOf () მეთოდი. იგი ძირითად სტრიქონში პოულობს მითითებული სიმბოლოს ან სიტყვის მიხედვით ქვესტრიქონს, ოღონდ ბოლოდან (არა პირველს მარცხნიდან). შესაბამისი კოდი მოცემულია 3.8_ლისტინგში, შედეგები კი 3.9 ნახაზზე. აქ კარგად ჩანს განსხვავება IndexOfAny() და LastIndexOf () მეთოდებს შორის.



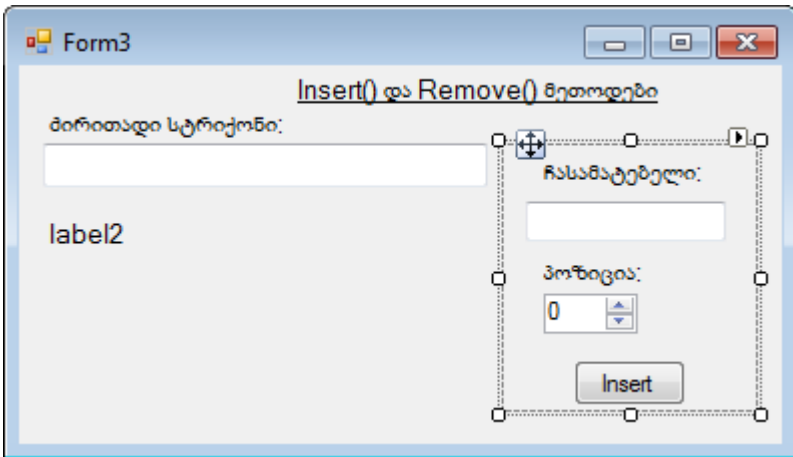
ნახ.3.9

// ლისტინგი_3.8 --- LastIndexOf ----

```
private void button5_Click(object sender, EventArgs e)
{
    string Striqoni;
    int pozicia;
    label2.Text = ""; label15.Text = ""; label17.Text = "";
    Striqoni = textBox1.Text;
    textBox2.Text = "ლამაზი";
    pozicia = Striqoni.LastIndexOf("ლამაზი");
    label15.Text += pozicia;
    label17.Text += Striqoni.Substring(pozicia);
}
```

3.4. სტრიქონებთან მუშაობა: Insert (), Remove() და Substring()

➤ Insert() მეთოდი უზრუნველყოფს ერთი სტრიქონის შეტანას მეორე სტრიქონში. მაგალითად, ძირითად სტრიქონში (textBox1) შეტანილია რაიმე წინადადება. ჩასამატებელ სტრიქონში პანელზე (textBox2) შეიტანება სიმბოლო ან სიმბოლოთა მწკრივი, რომელიც უნდა ჩაჯდეს პოზიციაში მითითებული რიცხვის მიხედვით (numericUpDown1) (ნახ.3.10). თავიდან numericUpDown1 ელემენტის Properties-ში Maximum=0, Minimum=0 და Value=0. ე.ი. ასეთ დროს ქვესტრიქონი მიუერთდება ძირითადს მხოლოდ დასაწყისში. თუ პოზიციას დავაყენებთ „4“-ზე, მაშინ მივიღებთ სწორ შედეგს. პროგრამის კოდი მოცემულია 3.9_ლისტინგში



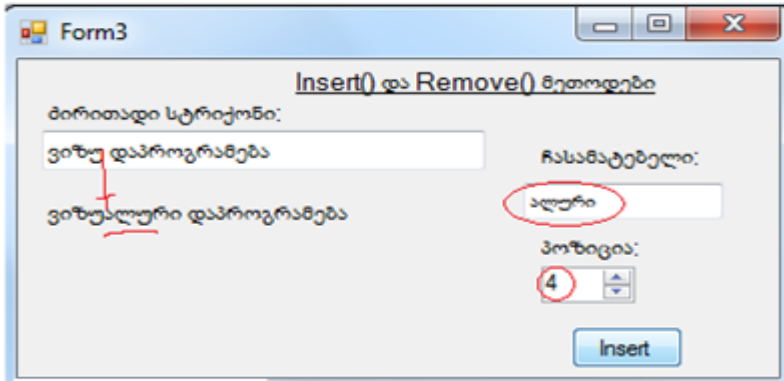
ნახ.3.10

```
// ლისტინგი_3.9--- Insert() -----
private void button1_Click(object sender, EventArgs e)
{
    string Striqoni, qveStriqoni;
    Striqoni = textBox1.Text;
    qveStriqoni = textBox2.Text;
```



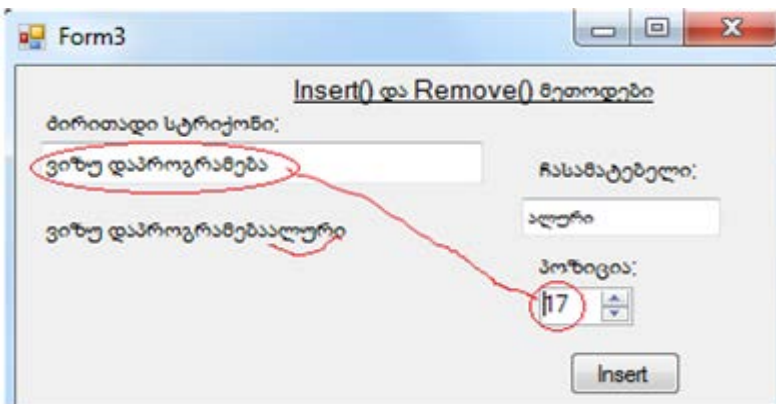
```
label2.Text = Striqoni.Insert((int)  
    numericUpDown1.Value, qveStriqoni);  
}
```

შედეგი 3.11 ნახაზზე ასახული.



ნახ.3.11

აქ ლებელში ჩანს სიტყვა „ვიზუალური“, რომელიც Insert() მეთოდით განხორციელდა. თუ პოზიციას შეეცვლით, მაგალითად „15“-ით, მერე „17“ და მივიღებთ 3.12 ნახაზზე მიღებულ უაზრო შედეგს. ამის მეტი მთვლელი არ გაზრდის მნიშვნელობას, ვინაიდან ის ძირითადი სტრიქონის სიგრძეა.



ნახ.3.12

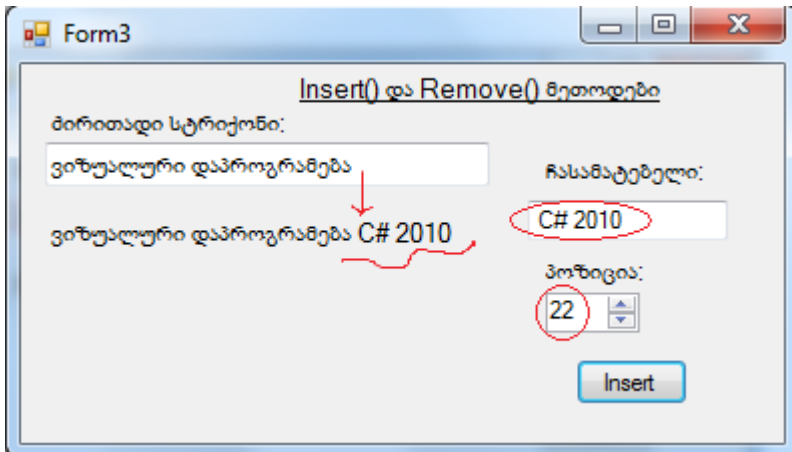
იმისათვის, რომ განხორციელდეს ძირითადი სტრიქონის შეცვლის შესაძლებლობა და მისი სიგრძის პარამეტრის ცვლილება, საჭიროა კოდი, რომელიც 3.10_ლისტინგშია მოცემული.

// ლისტინგი_3.10 -----

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    string Striqoni;
    Striqoni = textBox1.Text;
    numericUpDown1.Maximum = Striqoni.Length;
}
```

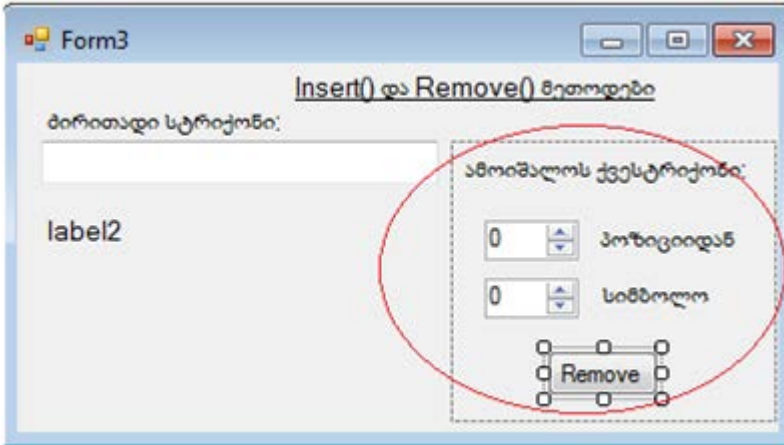
როგორც კი დაიწყება ძირითადი სტრიქონის textBox1-ში ტექსტის ცვლილება, მაშინვე იმუშავებს ეს კოდი და შეიცვლება სტრიქონის სიგრძის შემზღვეველი რიცხვი (ნახ.3.13)

numericUpDown1.Maximum -ში.



ნახ.3.13

➤ Remove() მეთოდი უზრუნველყოფს სტრიქონიდან ქვესტრიქონის ამოშლას, მითითებული პოზიციისა და სიმბოლოების რაოდენობის მიხედვით (ლისტინგი 3.11). დავამატოთ ფორმაზე ეს ელემენტები (ნახ.3.14).

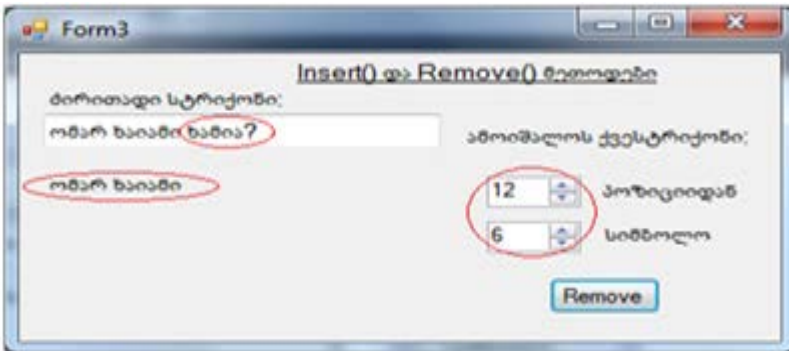


ნახ.3.14

```
// ლისტინგი_3.11 --- Remove() -----
private void button2_Click(object sender, EventArgs e)
{
    string Striqoni;
    Striqoni = textBox1.Text;
    label2.Text = Striqoni.Remove((int)numericUpDown1.Value,
        (int)numericUpDown2.Value);
}
private void textBox1_TextChanged(object sender, EventArgs e)
{
    string Striqoni;
    Striqoni = textBox1.Text;
    numericUpDown1.Maximum = Striqoni.Length-1;
    numericUpDown2.Maximum = Striqoni.Length;
}
```

```
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
    string Striqoni = textBox1.Text;
    numericUpDown2.Maximum = Striqoni.Length -
        numericUpDown1.Value;
}
```

შედეგები ასახულია 3.15 ნახაზზე.



ნახ.3.15

numericUpDown1_ValueChanged() მეთოდის დანიშნულებაა „პოზიციის“ და „სიმბოლოების“ მრიცხველებში დასაშვები რიცხვების კონტროლი. მაგალითად, ჩვენ შემთხვევაშია 12 და 5. ბოლო სიტყვა მთლიანად ამოშლილია. თუ პოზიციაში ჩავწერთ 13-ს, მაშინ სიმბოლოში ავტომატურად გამოჩნდება 5, თუ 14, მაშინ 4 და ა.შ., ბოლოს 17-ზე გვექნება 1. მეტს ვეღარ შევცვლით პოზიციის მეტობისკენ. პირიქით პოზიციის შემცირება დასაშვებია,

მაგალითად, დავაყენოთ 5. მაშინ სიმბოლოების რაოდენობა, რომელიც შეიძლება წავშალოთ მაქსიმალურად იქნება 13 და შედეგად მივიღებთ სიტყვას „ომარ“.

თავი 4

მართვის ვიზუალური ელემენტები ListBox და ComboBox

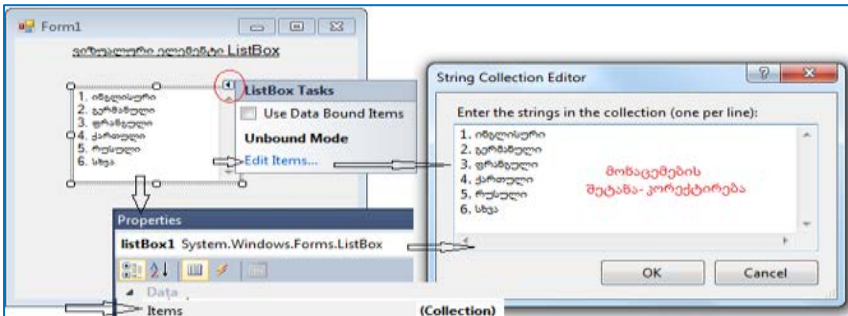
4.1. ვიზუალური ელემენტები: ListBox, CheckedListBox

ListBox ახორციელებს მისი ჩანაწერების სიის ასახვას ფორმაზე. მომხმარებელს შეუძლია ამ ჩანაწერების მრავალჯერადი ამორჩევა და გამოყენება. ჩანაწერების სია შეიძლება იყოს დიდი მოცულობის, რომელიც ფანჯარაში ვერ თავსდება, ამიტომაც ListBox-ს გააჩნია ავტომატურად მომუშავე კომპონენტი - ScrollBar, რომელიც -Properties-იდან ყენდება true-მნიშვნელობით (ნახ.4.1).



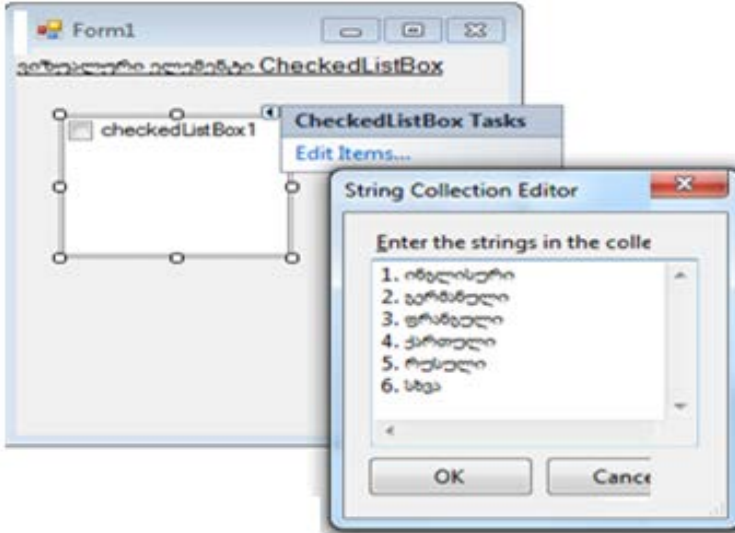
ნახ.4.1

ListBox-ში ტექსტური სტრიქონების შეტანა ხდება Properties->Items თვისებიდან ან ლისტბოქსის ჩარჩოს ზედა-მარჯვენა კუთხეში ისარზე მალსის დაწკაპუნებით გამოტანილ Edit Items არჩევით (ნახ.4.2). ორივე შემთხვევაში გამოიტანება String Collection Editor, რომელშიც ჩაიწერება მონაცემები.



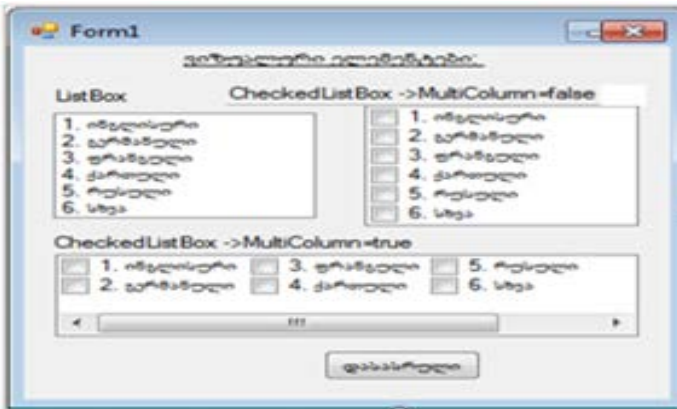
ნახ.4.2

CheckBox ელემენტი მსგავსია ListBox-ისა, ოღონდაც მის სტრიქონს ემატება წინ checkBox-ელემენტი (ნახ.4.3).



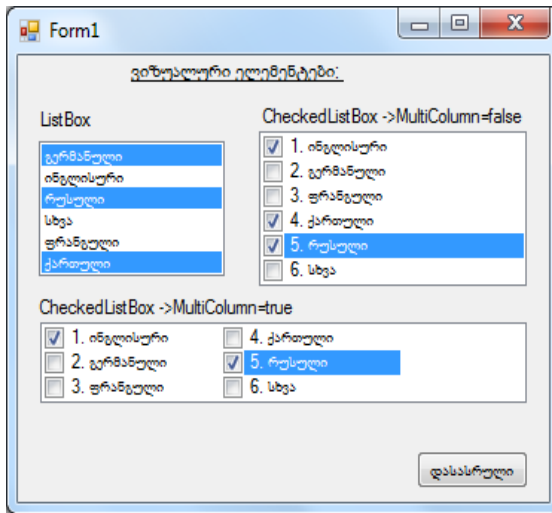
ნახ. 4.3

სტრიქონების შეტანის შემდეგ Edit Items რედაქტორში მიიღება 4.4 ნახაზზე ნაჩვენები სურათი.



ნახ. 4.4

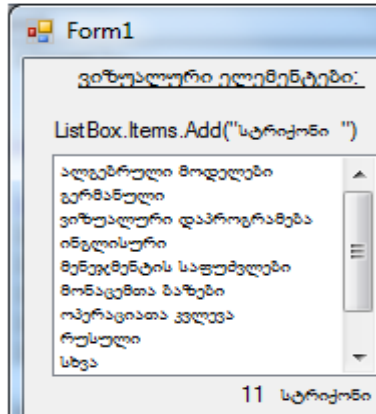
ListBox-ის სტრიქონებს მოვაცილოთ ნომრები და მის Properties-ში თვისება დავაყენოთ: Sorted->>true. სტრიქონები მოწესრიგდება ანბანის შესაბამისად (ნახ.4.5). CheckedListBox-ში შესაძლებელია რამდენიმე სტრიქონის მონიშვნა. ListBox-ში კი საჭიროა ამ მიზნით Properties-ის SelectionMode->MultiExtended არჩევა. ამის შემდეგ შეიძლება Shift და Ctrl კლავიშების დახმარებით რამდენიმე ან ყველა სტრიქონის მონიშვნა.



ნახ. 4.5

ახლა განვიხილოთ ტექსტბოქსებთან პროგრამულად მუშაობის ზოგიერთი მეთოდი.

➤ **Add()** მეთოდი Item თვისებისთვის. პროგრამული კოდის ფრაგმენტი, რომელშიც ხდება სტრიქონების ჩამატება ListBox-ში (ნახაზი 4.6) მოცემულია 4_1 ლისტინგში. როგორც ვხედავთ, სტრიქონები ჩაწერილია Form1_Load (object ...) მეთოდში. ფორმის ცარიელ ადგილას მაუსით დაკლიკვით გადავალთ კოდის ამ ფრაგმენტზე.



ნახ. 4.6

// ლისტინგი_4.1 – ListBox-ის Item-თვისების Add()-მეთოდი----

```
private void Form1_Load(object sender, EventArgs e)
{
    int st;
    listBox1.Items.Add("მონაცემთა ბაზები");
    listBox1.Items.Add("ვიზუალური დაპროგრამება");
    listBox1.Items.Add("ოპერაციათა კვლევა");
    listBox1.Items.Add("ალგებრული მოდელები");
    listBox1.Items.Add("მენეჯმენტის საფუძვლები");
    st = listBox1.Items.Count; //სტრიქონების რაოდენობა
    label5.Text = st.ToString();
}
```

- **Items.Count** - მეთოდით განისაზღვრება LisBox-ში სტრიქონების რაოდენობა. label5-ში გამოიტანება: 11 სტრიქონი (ნახ.4.6).

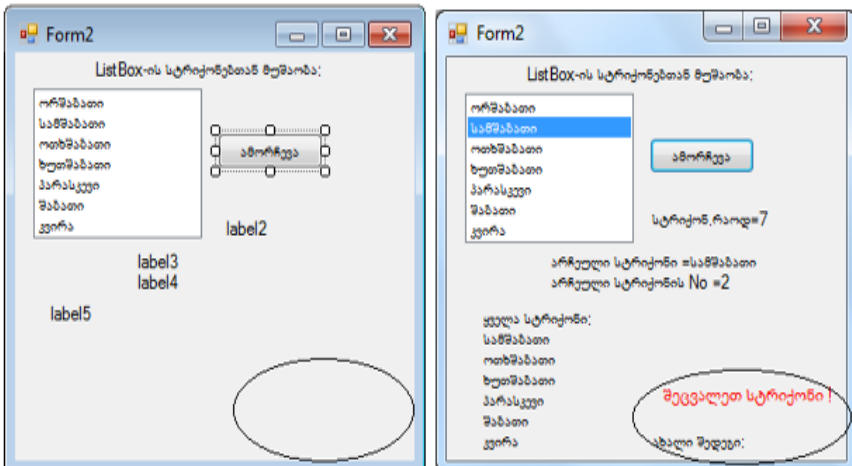
პროგრამულ კოდთან მუშაობისას სტრიქონების იდენტიფიკაციისათვის და ამოსარჩევად გამოიყენება SelectedItem / SelectedItems და SelectedIndex / SelectedIndices თვისებები.

4_2 ლისტინგში მოცემულია ფრაგმენტი ListBox-ის სტრიქონებთან სამუშაოდ. შედეგები „ამორჩევა“ ღილაკის ამოქმედების შემდეგ ასახულია 4.7 ნახაზზე.

//ლისტინგი_4.2 – SelectedItem, SelectedIndex, Items[index] -----

```
private void button1_Click(object sender, EventArgs e)
{
    int st;
    label2.Text = "სტრიქონ.რაოდ=" + listBox1.Items.Count;
    label3.Text = "არჩეული სტრიქონი =" + listBox1.SelectedItem;
    label4.Text = "არჩეული სტრიქონის No =" +
    (listBox1.SelectedIndex+1).ToString();
    label5.Text = "ყველა სტრიქონი:" + "\n";
    for (st = 1; st < listBox1.Items.Count; st++)
        label5.Text += listBox1.Items[st] + "\n";

    label6.Visible = true; // ელემენტი გამოჩნდება ეკრანზე
    label7.Visible = true;
    label8.Visible = false; // ელემენტი არ ჩანს ეკრანზე
}
```

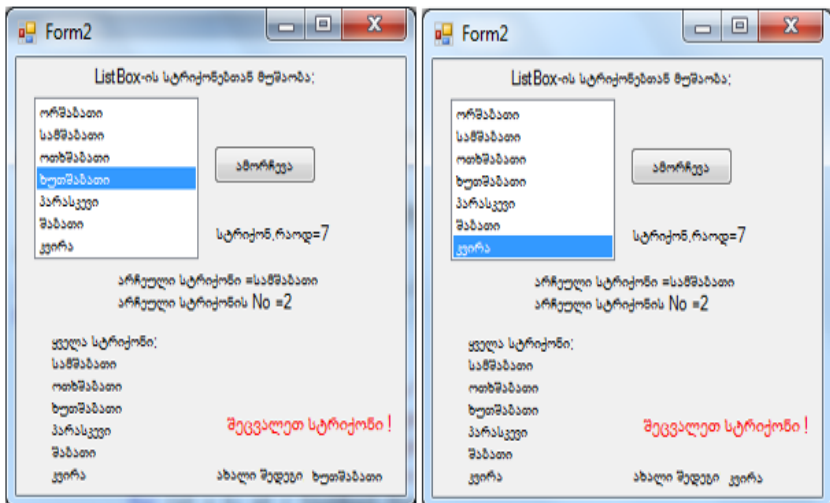


ნახ.4.7

ListBox-ში სტრიქონის შეცვლისას ავტომატურად უნდა შეიცვალოს გამოსატანი ახალი შედეგის მნიშვნელობა (რომელიც ღვალის ადგილას label8-ში უნდა ჩაიწეროს). ის გააქტიურდება და გამოჩნდება ფორმაზე „ამორჩევა“ ღილაკის გააქტიურების შემდეგ. საჭიროა შეიქმნას მოვლენა listBox1_SelectedIndexChanged, რომელიც განახორციელებს ავტომატურ ცვლილებას. მისი ლისტინგია_4.3, ხოლო შედეგები 4.8 ნახაზზეა ასახული (ახალი სტრიქონი: ხუთშაბათი, კვირა).

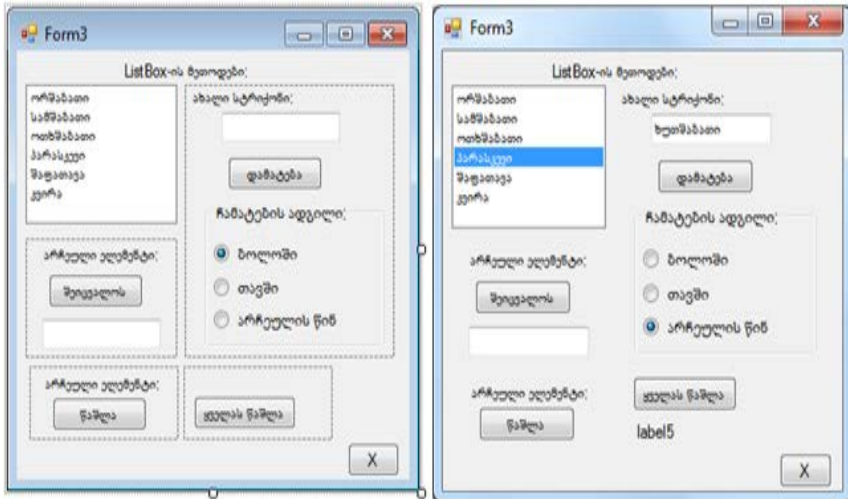
//ლისტინგი_4.3: ListBox-ის სტრიქონის შეცვლის მოვლენის კოდი--
 private void listBox1_SelectedIndexChanged(object sender, EventArgs e)

```
{
    label8.Visible = true;
    label8.Text = " " +listBox1.SelectedItem;
}
```



ნახ.4.8

ამოცანა_4.1: დავაპროგრამოთ ListBox-ისთვის სამი მეთოდი: ახალი_სტრიქონის_შეტანის, არასაჭირო_სტრიქონის_ამოშლის და სტრიქონის_შეცვლის მიზნით. შესაძლებელია Insert() და RemoveAt() მეთოდების გამოყენება. შედეგების ასახვისათვის შევქმნათ Form3 (ნახ. 4.9).



ნახ.4.9

„დამატება“ ლილავისთვის, რომელიც ახალ სტრიქონს ამატებს ListBox-ის თავში, ბოლოში ან მითითებული სტრიქონის წინ, კოდს ექნება 4.4 ლისტინგზე ნაჩვენები სახე.

//--ლისტინგო_4,4 – ListBox-ის Insert() მეთოდი -----

```
private void button1_Click(object sender, EventArgs e) // ჩამატება
{
    if (textBox1.Text == "")
        return;

    if (radioButton2.Checked)
        listBox1.Items.Insert(0, textBox1.Text);
    else if (radioButton3.Checked && listBox1.SelectedIndex != -1)
        listBox1.Items.Insert(listBox1.SelectedIndex, textBox1.Text);
}
```

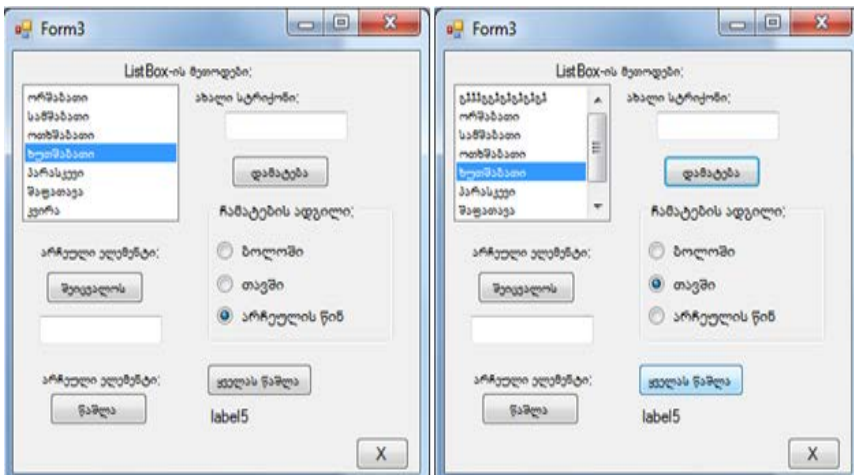
```

else
    listBox1.Items.Add(textBox1.Text);

textBox1.Text = "";
}

```

ახალი მონაცემის (მაგალითად, „ხუთშაბათი“) დამატების შემდეგ ლისტბოქსი მიიღებს 4.10 ნახაზზე მოცემულ სახეს. თავში დავამატეთ აგრეთვე „გჰგჰგჰგჰგჰგჰგჰგჰგ“ სტრიქონი.



ნახ. 4.10

„წაშლა“ ღილაკისთვის, რომელიც არჩეულ სტრიქონს ამოშლის ListBox-იდან, კოდს ექნება 4.5 ლისტინგზე ნაჩენები სახე.

//--ლისტინგი_4,5– ListBox-ის RemoveAt() მეთოდი -----

// არჩეულის წაშლა

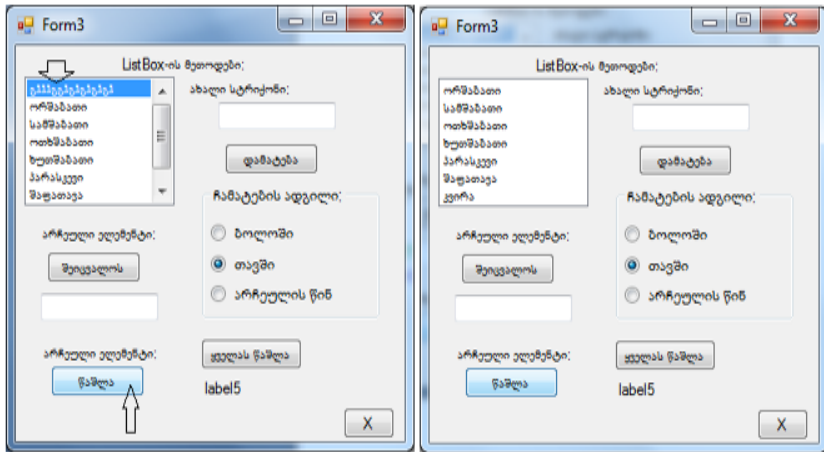
```

private void button2_Click(object sender, EventArgs e)
{
    int st = listBox1.SelectedIndex;
    if (st != -1)
        listBox1.Items.RemoveAt(st);
}

```

ListBox-ზე ავირჩიოთ პირველი სტრიქონი („გჰგჰგჰგჰგჰ“) და „წაშლა“-ლილაკი ავამოქმედოთ. შედეგად ლისტბოქსიდან გაქრება ეს სტრიქონი.

აქ იმუშავა Items.RemoveAt(st) მეთოდმა, რომელსაც SelectedIndex-ით მიეწოდა წასაშლელი სტრიქონის ნომერი. თუ არაა არჩეული, SelectedIndex აბრუნებს „-1“ მნიშვნელობას. შედეგი ასახულია 4.11 ნახაზზე.



ნახ. 4.11

„ყველას_წაშლა“ ლილაკი „სახიფათოა“, რადგან შეიძლება შემთხვევით მონაცემები დავკარგოთ. ეს ლილაკი უნდა შეიცავდეს მომხმარებლის დამატებით გაფრთხილებას. თუ მისგან მიიღებს „დასტურს“ ყველა სტრიქონის წაშლაზე, მხოლოდ მაშინ გაასუფთავებს ლისტბოქსის ჩანაწერებს. 4_5 ლისტინგზე მოცემულია „ყველას_წაშლის“ პროგრამული კოდის ფრაგმენტი.

//---ლისტინგი_4,5 --- ListBox.Items.Clear() მეთოდი -----

// ყველას წაშლა

private void button3_Click(object sender, EventArgs e)

{

 DialogResult all = MessageBox.Show("ნამდვილად წაშალოთ

```

ყველა ?", "გაფრთხილება", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question);
if (all == DialogResult.Yes)
{
    label5.Text = "ყველაფერი იშლება !";
    listBox1.Items.Clear();
}
else
if (all == DialogResult.No)
    label5.Text = "არ იშლება ყველა !";
else
    label5.Text = "Cancel-ია !";
}

```

ჩვენს შემთხვევაში კოდში გამოყენებულია MessageBox კლასის Show(პარამეტრები) მეთოდი, “Yes/No/Cancel” დილაკებით და გამაფრთხილებელი შეტყობინებით (ნახ.4.12). განვიხილოთ კოდის სტრიქონი:

```

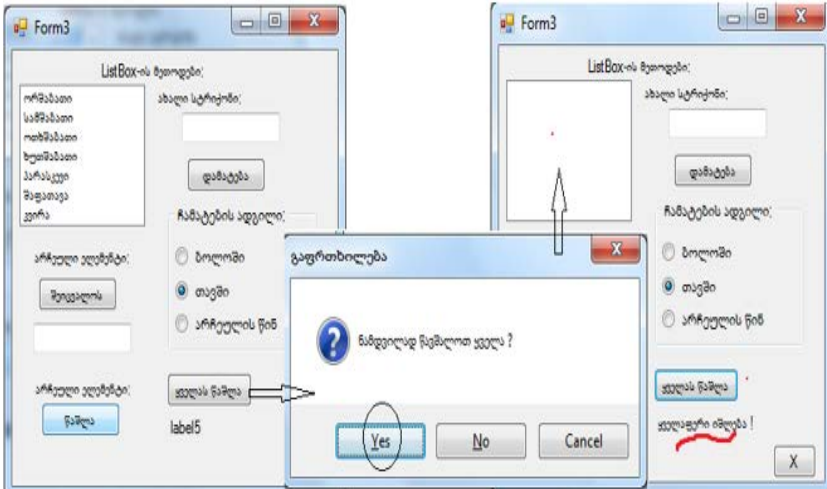
DialogResult all = MessageBox.Show("ნამდვილად წავშალოთ
ყველა ?", "გაფრთხილება",
    MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question);

```

DialogResult არის System.Windows.Forms სახელსივრცის ჩამონათვლის (enum) ტიპი: public enum DialogResult, რომელიც განსაზღვრავს დიალოგური ფანჯრიდან დაბრუნებული იდენტიფიკატორის მნიშვნელობას. all-ს მიენჭება ეს მნიშვნელობა, რომელიც შემდგომ if ბლოკში გამოვიყენეთ.

MessageBoxButtons – იძლევა enum ტიპის კონსტანტებს, რომლებიც განსაზღვრავს MessageBox ფანჯარაში გამოსატან დილაკებს (ჩვენთან: Yes, No, Cancel).

MessageBoxIcon - enum ტიპის კონსტანტაა, რომელიც ასახავს შეტყობინებას. მაგალითად, MessageBoxIcon.Question ესაა ცისფერ წრეში ჩასმული თეთრი ფერის კითხვის ნიშნის სიმბოლო (ნახ.4.12)



ნახ. 4.12

ListBox-ის რამდენიმე სტრიქონის მონიშვნისათვის, როგორც ზემოთ აღვნიშნეთ, მის Properties-ის SelectionMode-თვისებაში ვაყენებთ MultiExtended-მნიშვნელობას (ნახ.4.5). ახალი ლისტ-ბოქსის გახსნისას, მასში დაყენებულია ერთ სტრიქონიანი რეჟიმი: SelectionMode="One".

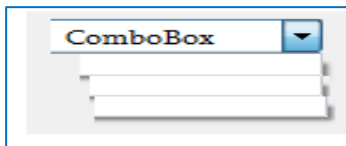
თვისებები SelectedIndices და SelectedItems შეიცავს არჩეული სტრიქონების შესაბამის ნომრებს ან ჩანაწერებს.

სავარჯიშო დავალება:

ამოცანა_4.2: ავავოთ კოდი, რომლითაც შესაძლებელი იქნება ListBox1-ის რამდენიმე სტრიქონის მონიშვნა და მათი ერთდროულად გადატანა (Copy) ListBox2-ში. ამავდროულად, შესაძლებელი უნდა იყოს ListBox2-იდან სტრიქონების უკან დაბრუნება ListBox1-ში. კოდი უნდა შეიცავდეს აგრეთვე ცალკეული სტრიქონების გადატანას (Move) ტექსტბოქსებს შორის და შესაძლებლობას ტექსტბოქსის სტრიქონების მთლიანად წასაშლელად (DeleteAll).

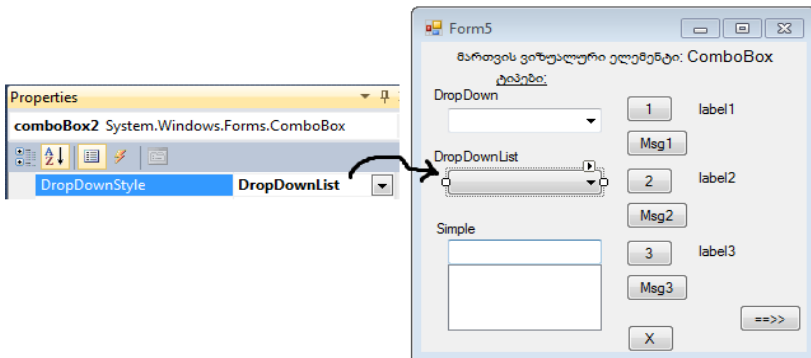
4.2. ვიზუალური ელემენტი ComboBox და თვისება DropDownStyle

ComboBox ბევრად ამარტივებს მომხმარებელთა ინტერფეისებს და მოქნილს ხდის მათ. აქ განვიხილავთ მისი საშუალებით პროგრამული პროექტების აგების საილუსტრაციო მაგალითებს. 4.13 ნახაზზე ნაჩვენებია ComboBox-ს ზოგადი საილუსტრაციო მაგალითი.



ნახ. 4.13

ComboBox ვიზუალური ელემენტი ListBox-ის და TextBox-ის ელემენტთა სიმბიოზია, იგი იყენებს ორივეს მახასიათებლებს, რომლებიც ჩვენ ზემოთ განვიხილეთ. ამავდროულად მისი წარმოდგენის ფორმა განსხვავდება ორივესგან და ეფექტურია (ფორმაზე იკავებს შედარებით მცირე ადგილს და ამორჩევის მექანიზმითაც სარგებლობს). 4.14 ნახაზზე ნაჩვენებია ComboBox-ის სამი ვარიანტი. მისი ტიპი Properties-ში მიეთითება DropDownStyle თვისების ერთ-ერთი მნიშვნელობით.



ნახ. 4.14

- DropDown - სტანდარტულად ეს მნიშვნელობაა დაყენებული. სამკუთხა ისრით ჩამოიშლება სტრიქონების სია ListBox-ის მსგავსად, ან ტექსტურ ველში შეიტანება სტრიქონი TextBox-ით;
- DropDownList - გამორთავს TextBox-ის შესაძლებლობას, ანუ ვეღარ შევიტანთ ტექსტს. კომბობოქსი მუშაობს ლისტბოქსის რეჟიმში და არის ტექსტბოქსით მცირე ზომის;
- Simple - ყოველთვის ღიაა სტრიქონების სია. შეიძლება სტრიქონის არჩევაც და ახლის შეტანაც.

შენიშვნა: ComboBox-ისთვის არ გამოიყენება თვისება SelectionMode.

ამოცანა_4.3: ავავოთ 4.14 ნახაზზე ნაჩვენები ფორმა და ავამოქმედოთ 1,2,3-ლილაკები კომბობოქსის სამი განსხვავებული ტიპის საილუსტრაციოდ. 4_7 ლისტინგზე მოცემულია პროგრამული კოდი. დამატებითი შეტყობინება გამოვიტანოთ MsgBox-ფანჯარაში შესაბამისი კომბობოქსის არჩეული სტრიქონის მნიშვნელობის (comboBox.SelectedItem) და მისი ინდექსის (ნომრის) (comboBox.SelectedIndex) საილუსტრაციოდ.

თითოეული ღილაკისთვის ფორმაზე Properties-ში დავაყენოთ DropDownStyle თვისების ერთ-ერთი მნიშვნელობა: 1- DropDown, 2- DropDownList ან 3- Simple.

Form5_Load(object...)-ში პროგრამულად ჩაიწერება სამივე ComboBox-ის სტრიქონები, მაგალითად, C++, C#, J++, F# და ა.შ. (ან შეიძლება Edit Items რედაქტორის გამოყენება).

// ლისტინგი_4.7: ComboBox-ის ტიპები და თისება: DropDownStyle-

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinListCombo
{
    public partial class Form5 : Form
    {
        public Form5()
        { InitializeComponent(); }
    }
}
```

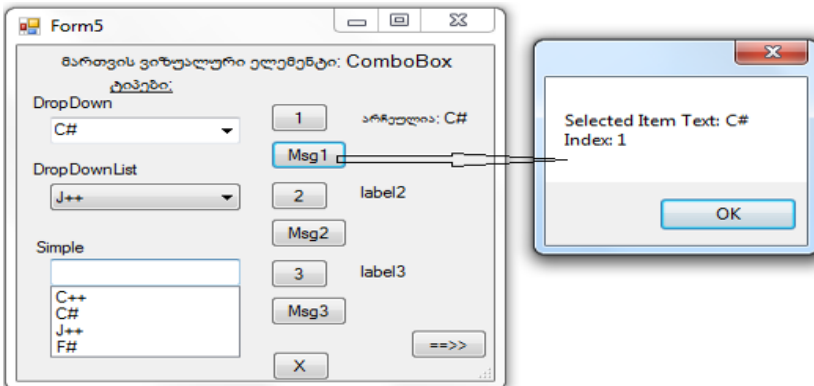
```
private void Form5_Load(object sender, EventArgs e)
{
    comboBox1.Items.Add("C++");
    comboBox1.Items.Add("C#");
    comboBox1.Items.Add("J++");
    comboBox1.Items.Add("F#");
    comboBox2.Items.Add("C++");
    comboBox2.Items.Add("C#");
    comboBox2.Items.Add("J++");
    comboBox2.Items.Add("F#");
    comboBox3.Items.Add("C++");
    comboBox3.Items.Add("C#");
    comboBox3.Items.Add("J++");
    comboBox3.Items.Add("F#");
}
private void button1_Click(object sender, EventArgs e)
{
    label1.Text="არჩეულია: "+ comboBox1.Text;
}
private void button2_Click(object sender, EventArgs e)
{
    label2.Text = "არჩეულია: " + comboBox2.SelectedItem;
}
private void button3_Click(object sender, EventArgs e)
{
    label3.Text = "არჩეულია: " + comboBox3.Text;
}
private void button6_Click(object sender, EventArgs e)
{
    int selectedIndex = comboBox1.SelectedIndex;
    Object selectedItem = comboBox1.SelectedItem;
    MessageBox.Show("Selected Item Text: " + selectedItem + "\n" +
        "Index: " + selectedIndex.ToString());
}
```

```
private void button7_Click(object sender, EventArgs e)
{
    int selectedIndex = comboBox2.SelectedIndex;
    Object selectedItem = comboBox2.SelectedItem;

    MessageBox.Show("Selected Item Text: " + selectedItem + "\n" +
        "Index: " + selectedIndex.ToString());
}
private void button8_Click(object sender, EventArgs e)
{
    int selectedIndex = comboBox3.SelectedIndex;
    Object selectedItem = comboBox3.SelectedItem;

    MessageBox.Show("Selected Item Text: " + selectedItem + "\n" +
        "Index: " + selectedIndex.ToString());
}
private void button5_Click(object sender, EventArgs e)
{
    Close();
}
}
```

პროგრამის მუშაობის შედეგი ნაჩვენებია 4.15 ნახაზზე.

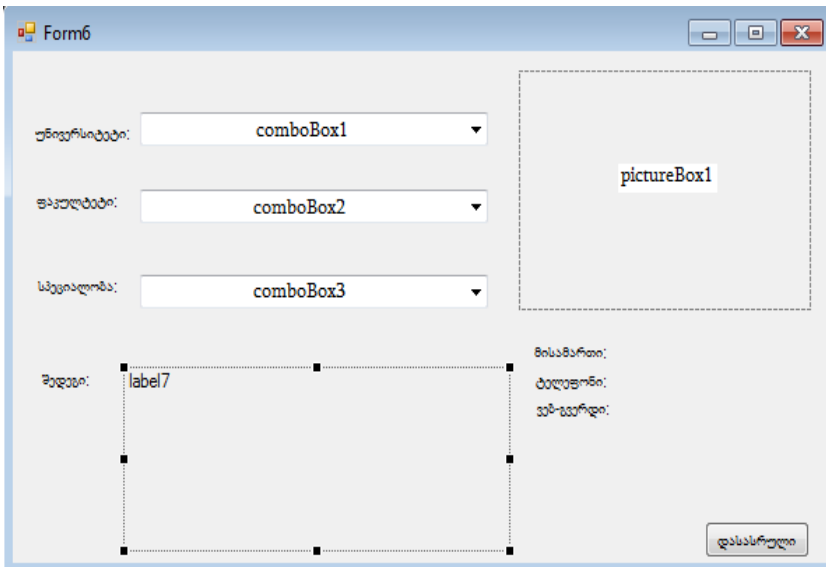


ნახ.4.15

გავლილი მასალის საფუძველზე ავავოთ ახალი პროგრამული პროექტი, რომელშიც გამოყენებულ იქნება ComboBox კლასის თვისებები და მეთოდები.

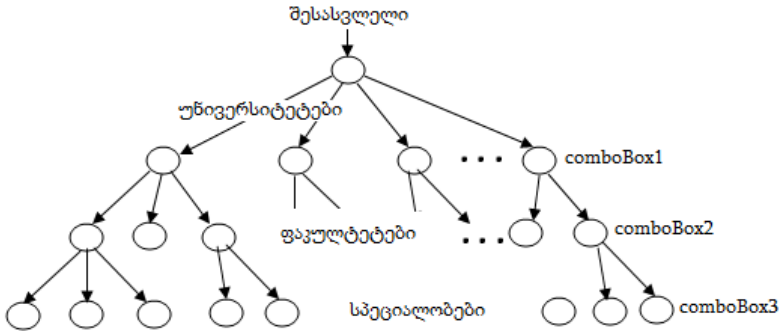
ამოცანა_4.4: შევქმნათ პროგრამული აპლიკაცია „უნივერსიტეტი“, რომელიც იძლევა ინფორმაციას მათი ფაკულტეტებისა და სპეციალობების შესახებ. მომხმარებელი ირჩევს სამდონიანი ComboBox-ების სისტემაში თავის სასურველ მონაცემებს (ნახ.4.16):

უნივერსიტეტი -> ფაკულტეტი -> სპეციალობა



ნახ.4.16

შედეგები აისახება label7 ტექსტურ ველში. ცვლილება ახალი მოთხოვნის შესასრულებლად შესაძლებელია სამივე დონეზე. ზოგადი იერარქიული მოდელი, რომლის პროგრამული რეალიზაცია შესაძლებელია ვიზუალური ელემენტებისა და პროგრამული switch() ... case ჩალაგებული გადამრთველების ერთობლიობით, ნაჩვენებია 4.17 ნახაზზე.



ნახ.4.17

ფორმაზე გამოიტანება აგრეთვე უნივერსიტეტების თანმხლები გრაფიკული და ტექსტური ინფორმაცია, შესაბამისად pictureBox1 და სამი label-ის საშუალებით (მისამართი, ტელეფონი, ვებ-გვერდი). მათი ცვლილება ხდება comboBox1-ით არჩეული მნიშვნელობის შესაბამისად. 4_8 ლისტინგზე ნაჩვენებია ამ პროგრამის რეალიზაციის კოდის ფრაგმენტი.

// ლისტინგი_4.8 -- ComboBox კლასის თვისებები და მეთოდები ----

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WinListCombo
{
    public partial class Form6 : Form
    {
        int fa = 0;
        string itemU, itemF, itemS;
        public Form6()
        {
            InitializeComponent();
        }

        private void Form6_Load(object sender, EventArgs e)
```

```
{
    comboBox1.Items.Add("თბილისის სახელმწიფო
                        უნივერსიტეტი"); // // www.tsu.ge
    comboBox1.Items.Add("საქართველოს ტექნიკური
                        უნივერსიტეტი"); // www.gtu.ge
    comboBox1.Items.Add("თბილისის სახელმწიფო სამედიცინო
                        უნივერსიტეტი"); //www.tsmu.edu.ge
    comboBox1.Items.Add("ილიას სახელმწიფო უნივერსიტეტი");
                        //www.iliauni.edu.ge
}
```

```
private void comboBox1_SelectedIndexChanged(object sender,
                                           EventArgs e)
```

```
{
    int selectedIndexU = comboBox1.SelectedIndex;
    Object selectedItemU = comboBox1.SelectedItem;
    itemU = selectedItemU.ToString();
    // MessageBox.Show("SelectedIndexU: " + selectedIndexU.ToString());
    switch (selectedIndexU)
    {
        case 0:
            comboBox2.Items.Clear();
            comboBox2.Items.Add("ზუსტ და საბუნებისმეტყველო მეცნ.
                                ფაკულტეტი");
            comboBox2.Items.Add("იურიდიული ფაკულტეტი");
            comboBox2.Items.Add("ჰუმანიტარულ მეცნ. ფაკულტეტი");
            comboBox2.Items.Add("სოციალურ-პოლიტიკურ მეცნ.
                                ფაკულტეტი");
            comboBox2.Items.Add("ეკონომიკისა და ბიზნესის
                                ფაკულტეტი");
            comboBox2.Items.Add("სამედიცინო ფაკულტეტი");
            pictureBox1.Image =Image.FromFile
                ("C:\C#2010\4_5\WinListCombo\tsu.jpg");
            label4.Text = "თბილისი, ჭავჭავაძის 1";
            label5.Text="222-22-22";
    }
}
```

```
label10.Text="www.stu.ge";
```

```
fa = 100;
```

```
break;
```

case 1:

```
comboBox2.Items.Clear();
```

```
comboBox2.Items.Add("არქიტექტურის და მშენებლობის  
ფაკულტეტი");
```

```
comboBox2.Items.Add("ენერგეტიკის და კავშირგაბმულობის  
ფაკულტეტი");
```

```
comboBox2.Items.Add("ინფორმატიკის და მართვის  
სისტემების ფაკულტეტი");
```

```
comboBox2.Items.Add("მექანიკური და სატრანსპორტო  
ფაკულტეტი");
```

```
comboBox2.Items.Add("ბიზნესის ინჟინერიის  
ფაკულტეტი");
```

```
comboBox2.Items.Add("სამთო-გეოლოგიური ფაკულტეტი");
```

```
pictureBox1.Image = Image.FromFile
```

```
("C:\\C#2010\\4_5\\WinListCombo\\gtu.jpg");
```

```
label4.Text = "თბილისი, კოსტავას 77";
```

```
label5.Text="237-37-37";
```

```
label10.Text="www.gtu.ge";
```

```
fa = 101;
```

```
break;
```

case 2:

```
comboBox2.Items.Clear();
```

```
comboBox2.Items.Add("მედიცინის ფაკულტეტი");
```

```
comboBox2.Items.Add("სტომატოლოგიის ფაკულტეტი");
```

```
comboBox2.Items.Add("ფარმაციის ფაკულტეტი");
```

```
comboBox2.Items.Add("საზოგადოებრივი ჯანდაცვის  
ფაკულტეტი");
```

```
comboBox2.Items.Add("სპორტული მედიცინის და  
რეაბილიტაციის ფაკულტეტი");
```

```
pictureBox1.Image = Image.FromFile
```

```
("C:\\C#2010\\4_5\\WinListCombo\\meduni.jpg");
```

```
label4.Text = "თბილისი, ვაჟა-ფშაველას 41";
```

```

label5.Text="239-39-39";
label10.Text="www.tsmu.edu.ge";
fa = 102;
break;
case 3:
    comboBox2.Items.Clear();
    comboBox2.Items.Add("ბიზნესის ფაკულტეტი");
    comboBox2.Items.Add("საინჟინრო ფაკულტეტი");
    comboBox2.Items.Add("სამართლის ფაკულტეტი");
    comboBox2.Items.Add("მეცნიერებათა და ხელოვნების
        ფაკულტეტი");
    comboBox2.Items.Add("სპორტის ფაკულტეტი");
    pictureBox1.Image = Image.FromFile
        ("C:\\C#2010\\4_5\\WinListCombo\\iliauni.jpg");
    label4.Text = "თბილისი, ჭავჭავაძის 101";
    label5.Text="239-39-39";
    label10.Text="www.iliauni.edu.ge";
    fa = 103;
    break;
// case 4: და ა.შ. -----
default: break;
}
}

private void comboBox2_SelectedIndexChanged(object sender,
                                                EventArgs e)
{
    int selectedIndexF = comboBox2.SelectedIndex;
    Object selectedItemF = comboBox2.SelectedItem;
    itemF = selectedItemF.ToString();
    // MessageBox.Show("SelectedIndexF: " + selectedIndexF.ToString());
    switch (fa)
    {
        case 100: // თსუ
            switch (selectedIndexF)

```



```
{
  case 0:
    comboBox3.Items.Clear();
    comboBox3.Items.Add("ფიზიკა-მათემატიკის");
    comboBox3.Items.Add("ბიოლოგიის");
    comboBox3.Items.Add("ქიმიის");
    comboBox3.Items.Add("ინფორმატიკის");
    break;
  case 1:
    comboBox3.Items.Clear();
    comboBox3.Items.Add("სისხლის სამართლის");
    comboBox3.Items.Add("სამოქალაქო სამართლის");
    comboBox3.Items.Add("ადმინისტრაციული
      სამართლის");
    break;
  case 2:
    comboBox3.Items.Clear();
    comboBox3.Items.Add("საქრთველოს ისტორიის");
    comboBox3.Items.Add("გეოგრაფიის");
    comboBox3.Items.Add("ვილოლოგიისა და
      ჟურნალისტიკის");
    break;
  case 3:
    // და ა.შ. -----
  default: break;
}
break;
case 101: // სტუ
  switch (selectedIndexF)
  {
    case 0:
      comboBox3.Items.Clear();
      comboBox3.Items.Add("სამოქალაქო მშენებლობის");
      comboBox3.Items.Add("ურბანისტიკის");
      comboBox3.Items.Add("ხუროთმოძღვრების");
```

```
        comboBox3.Items.Add("რკინაბეტონის  
            კონსტრუქციების");  
        break;  
    case 1:  
        comboBox3.Items.Clear();  
        comboBox3.Items.Add("ჰიდროენერგეტიკის");  
        comboBox3.Items.Add("თბოენერგეტიკის");  
        comboBox3.Items.Add("ატომური ენერგეტიკის");  
        break;  
    case 2:  
        comboBox3.Items.Clear();  
        comboBox3.Items.Add("ქსელების და სისტემების");  
        comboBox3.Items.Add("მართვის ავტომატიზებული  
            სისტემების");  
        comboBox3.Items.Add("ეკონომიკური  
            ინფორმატიკის");  
        break;  
    case 3: // და ა.შ. -----  
    default: break;  
}  
break;  
case 102: // სამედიცინო უნივ  
switch (selectedIndexF)  
{  
    case 0:  
        comboBox3.Items.Clear();  
        comboBox3.Items.Add("ფსიქიატრიის");  
        comboBox3.Items.Add("ქირურგიის");  
        comboBox3.Items.Add("კარდიოლოგიის");  
        comboBox3.Items.Add("თერაპიის");  
        break;  
    case 1:  
        comboBox3.Items.Clear();  
        comboBox3.Items.Add("ცხა-სახის ქირურგიის");  
        comboBox3.Items.Add("თერაპიის");
```

```
        comboBox3.Items.Add("პროტეზირების");
        break;
    case 2:
        comboBox3.Items.Clear();
        comboBox3.Items.Add("საპროვიზორო");
        comboBox3.Items.Add("საფარმაცევტო");
        comboBox3.Items.Add("სადიაგნოსტიკო");
        break;
    case 3: // და ა.შ. -----
    default: break;
}
break;
}
}
```

```
private void comboBox3_SelectedIndexChanged(object sender,
                                           EventArgs e)
{
    int selectedIndexS = comboBox3.SelectedIndex;
    Object selectedItemS = comboBox3.SelectedItem;
    itemS = selectedItemS.ToString();
    label7.Text = "თქვენ აირჩიეთ: \n" +
        itemU + "\n" +
        itemF + "\n" +
        itemS + " სპეციალობა";
}
}
}
```

4.18-4.22 ნახაზებზე ილუსტრირებულია ამ აპლიკაციის მუშაობის ფრაგმენტები სხვადასხვა მოთხოვნების შესრულებისას.

ვიზუალური დაპროგრამება (C#.NET & Workflow Foundation NET)

The screenshot shows a web browser window with a form titled "Form6". The form contains the following elements:

- A dropdown menu for "უნივერსიტეტი:" (University) with the selected value "თბილისის სახელმწიფო უნივერსიტეტი".
- A dropdown menu for "ფაკულტეტი:" (Faculty) with the selected value "იურიდიული ფაკულტეტი".
- A dropdown menu for "სპეციალობა:" (Specialty) with the selected value "სისხლის სამართლის".
- A "შედეგი:" (Result) section containing the text: "თქვენ აირჩიეთ: თბილისის სახელმწიფო უნივერსიტეტის იურიდიული ფაკულტეტის სისხლის სამართლის სპეციალობა".
- A photograph of a building on the right side.
- Contact information on the right: "თბილისი, ჭავჭავაძის 1", "222-22-22", and "www.stu.ge".
- A "დასასრული" (End) button at the bottom right.

ნახ.4.18

This screenshot shows the same "Form6" web browser window, but with the "უნივერსიტეტი:" dropdown menu open. The menu displays a list of options:

- თბილისის სახელმწიფო უნივერსიტეტი (selected)
- თბილისის სახელმწიფო უნივერსიტეტი
- საქართველოს ტექნიკური უნივერსიტეტი
- თბილისის სახელმწიფო სამედიცინო უნივერსიტეტი
- ილიას სახელმწიფო უნივერსიტეტი

The rest of the form, including the "ფაკულტეტი:" dropdown, the "შედეგი:" text, the building image, and the contact information, remains the same as in the previous screenshot.

ნახ.4.19

ვიზუალური დაპროგრამება (C#.NET & Workflow Foundation NET)

Form6

უნივერსიტეტი: საქართველოს ტექნიკური უნივერსიტეტი

ფაკულტეტი: ინჟინრების და მართვის სისტემების ფაკულტეტი

სპეციალობა: მართვის ავტომატიზებული სისტემების

მედი: თქვენ აირჩიეთ:
საქართველოს ტექნიკური უნივერსიტეტის
ინჟინრების და მართვის სისტემების ფაკულტეტის
მართვის ავტომატიზებული სისტემების სპეციალობა

თბილისი, ვოსტავას 77
237-37-37
www.gtu.ge

დასასრული

ნახ.4.20

Form6

უნივერსიტეტი: საქართველოს ტექნიკური უნივერსიტეტი

ფაკულტეტი: ენერჯეტიკის და კავშირგაბმულობის ფაკულტეტი

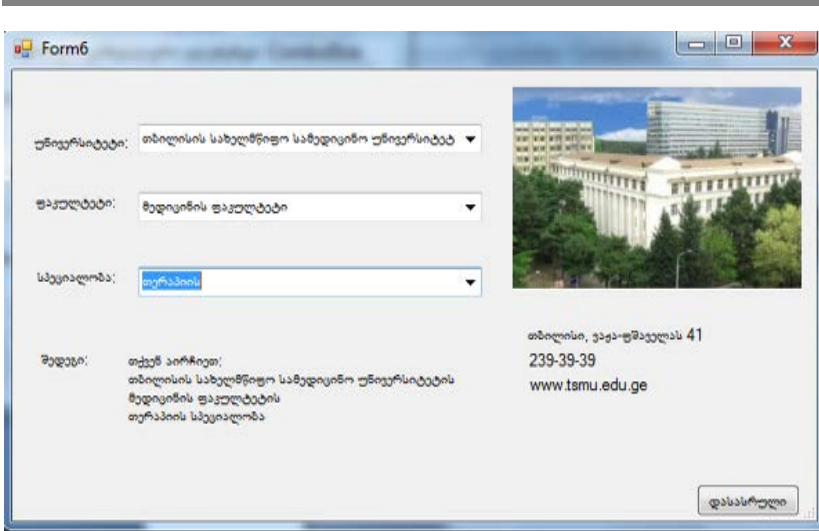
სპეციალობა: ჰიდროენერჯეტიკის

მედი: თქვენ აირჩიეთ:
საქართველოს ტექნიკური უნივერსიტეტის
ენერჯეტიკის და კავშირგაბმულობის ფაკულტეტის
ჰიდროენერჯეტიკის სპეციალობა

თბილისი, ვოსტავას 77
237-37-37
www.gtu.ge

დასასრული

ნახ.4.21



ნახ.4.22

და ა.შ. სისტემის გაფართოვება შესაძლებელია რეალური მონაცემებითაც, თუმცა აპლიკაციის გადასაწყვეტად არსებობს სხვა ხერხები და მეთოდებიც (მაგალითად, მონაცემთა ბაზების გამოყენებით, სადაც მოთავსდება დიდი მოცულობის ინფორმაცია), რომლებიც საგრძნობლად შეამცირებს კოდის მოცულობას. ჩვენ ამ საკითხებს მომავალში განვიხილავთ.

სავარჯიშო დავალება:

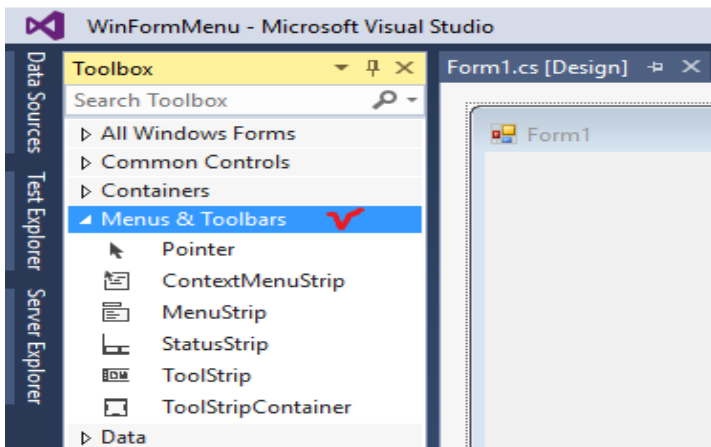
ააგეთ ფორმა „ვალუტის გადაცვლა“: ერთი ComboBox-ით უცხოური ვალუტის ასარჩევად. ორი TextBox-ით: ლარების რაოდენობის და უცხოური ვალუტის დღევანდელი კურსის მნიშვნელობის შესატანად. დაწერეთ ფულის კონვერტაციის C# - კოდი.

თავი 5

მენიუს აგების ვიზუალური საშუალებანი

განსაკუთრებული მნიშვნელობა აქვს კომპიუტერული სისტემების მომხმარებელთა ინტერფეისების დაპროგრამებას. ერთ-ერთი ძირითადი საკითხია მენიუების სისტემის დაპროექტენა და მისი შემდგომი პროგრამული რეალიზაცია. წინამდებარე თავში განხილულია მთავარი მენიუს, კონტექსტური მენიუს და გრაფიკული მენიუს აგების ვიზუალური ელემენტები .

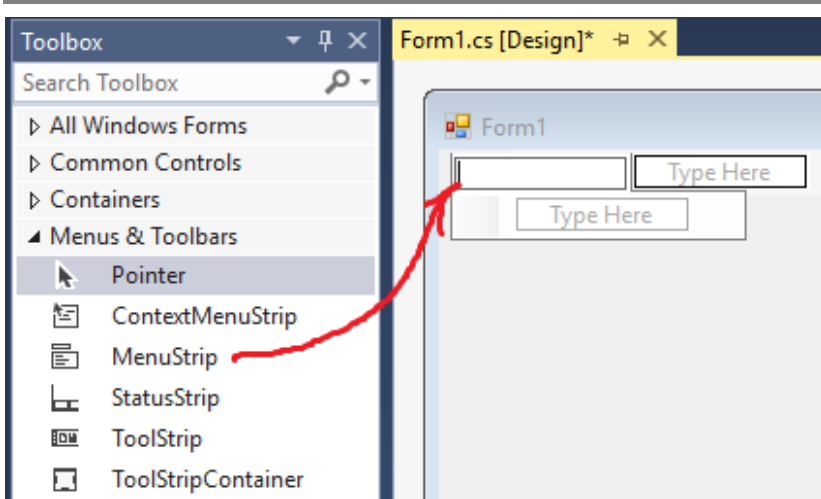
5.1 ნახაზზე მოცემულია Visual Studio.NET პლატფორმის C# ენის ვიზუალური ელემენტები, რომლებიც ინსტრუმენტების პანელზეა განთავსებული.



ნახ. 5.1. Menus & Toolbars ელემენტები

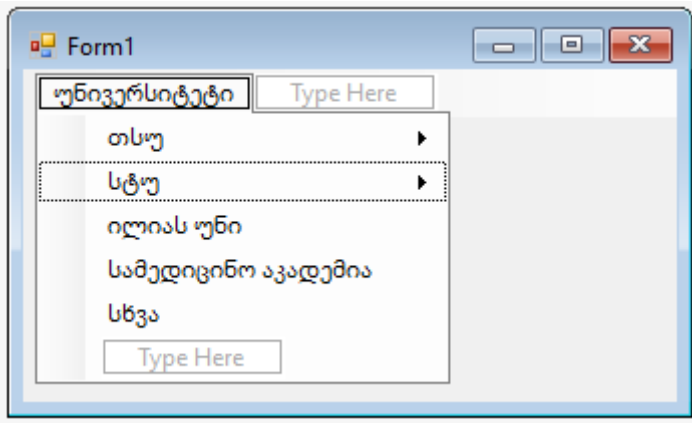
5.1. მთავარი მენიუს აგების ვიზუალური ელემენტები

ამოცანა 5.1: Form1-ზე ავაგოთ მთავარი მენიუ „უნივერსიტეტები“ და ქვემენიუს პუნქტებით „ფაკულტეტები“ და „კათედრები“. ფორმაზე მთავარი მენიუს შესაქმნელად ToolStrip-იდან გადმოვიტანოთ MenuStrip ელემენტი. ფორმას ექნება 5.2-ა ნახაზზე ნაჩვენები სახე.



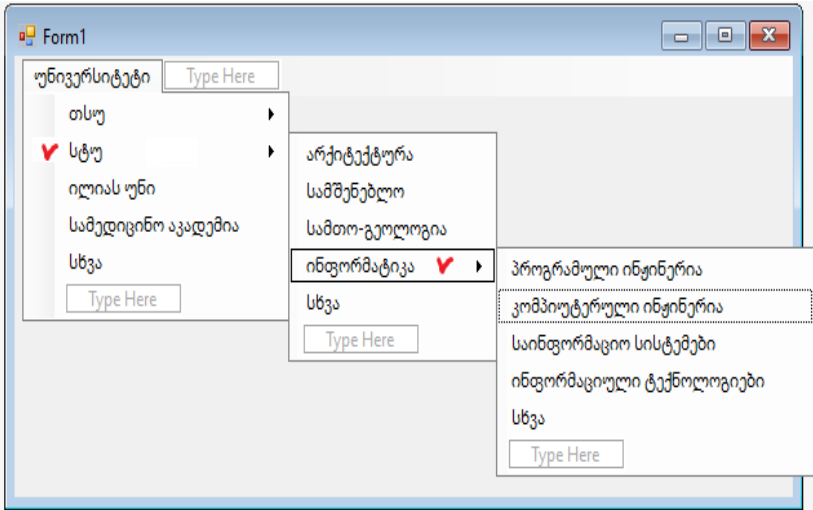
ნახ.5.2. საწყისი მდგომარეობა (MenuStrip)

შევიტანთ მენიუს პუნქტები „უნივერსიტეტი“ (5.2-ბ).



ნახ.5.2-ბ. მთავარი მენიუს შევსება

მენიუს შევსების პროცესი ხორციელდება ვერტიკალურად და/ან ჰორიზონტალურად. თითოეული სტრიქონისთვის შეიძლება ქვემენიუს შექმნა (ნახ.5.3).

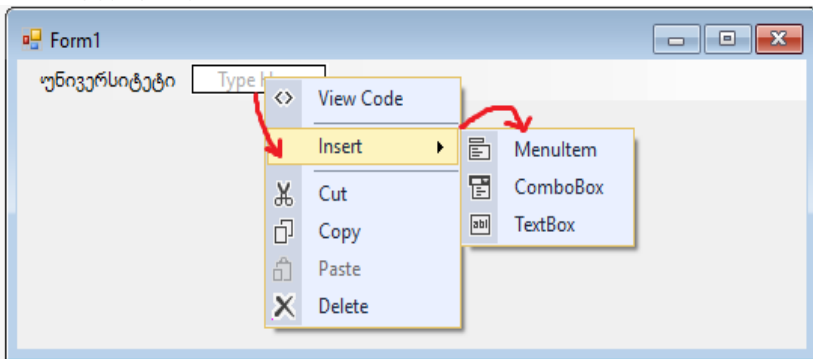


ნახ. 5.3. სამდონიანი მენიუ

მენიუს პუნქტების (სტრიქონების) გადაადგილება შეიძლება Form1[Design] რეჟიმში მაუსის მარცხენა ღილაკით Drag&Drop (გადატანა) საშუალებით.

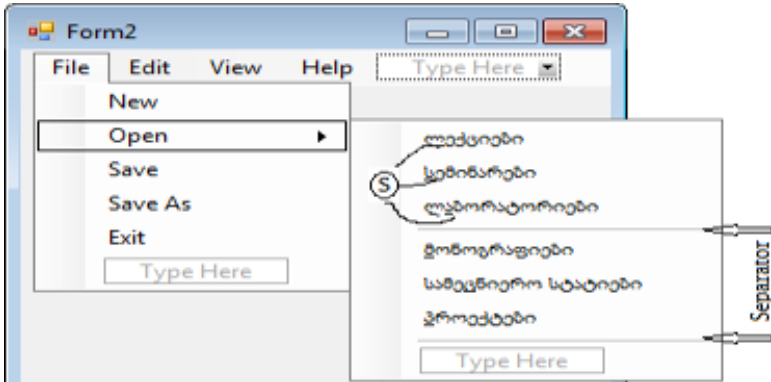
მთავარი მენიუს პუნქტები შიძლება იყოს სამი სახის (ნახ.5.4):

- ჩვეულებრივი შესატანი სტრიქონი;
- კომბობოქსი, რომელშიც მოხდება ამორჩევა ან შეტანა;
- ტექსტბოქსი.



ნახ. 5.4. მენიუს სამი ტიპი

ქვემნიუსთვის დასაშვებია Separator პუნქტიც. რომლის დანიშნულებაც მენიუს პუნქტების ერთმანეთისაგან გამოყოფა ხაზით, რაც ვიზუალურ კომფორტს უქმნის მომხმარებელს (ნახ.5.5).

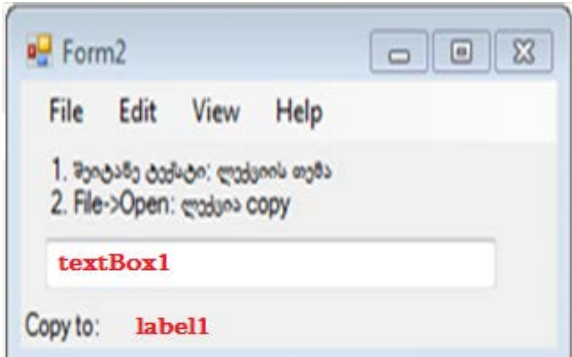


ნახ.5.5. Separator-ის გამოყენება

მენიუში ხშირად იყენებენ სტრიქონის ერთი ასოს გამოყოფას (ქვეშეგახაზვა), რომლითაც ამოქმედდება ეს პუნქტი. ნახაზზე იგი S სიმბოლოთია მითითებული.

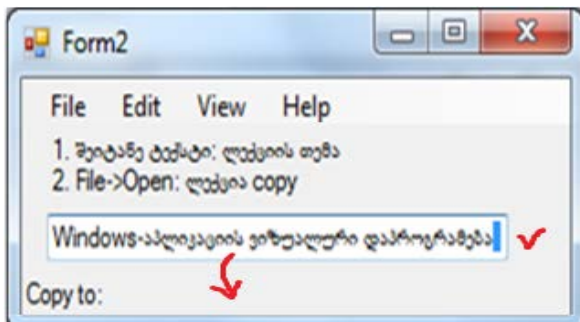
მენიუს პუნქტებით უნდა მოხდეს გარკვეული დავალების შესრულება (საჭირო ინფორმაციისკენ გზის განსაზღვრა და ბოლოს ამორჩევა). ამიტომ ეს პუნქტები დაკავშირებულია მოვლენებთან და მეთოდებთან. მოვლენის მთავარი სახეა Click, რომელზეც მიბმულია შესასრულებელი პროცედურის კოდი. განვიხილოთ ეს საკითხი.

ამოცანა_5.2: ფორმაზე (ნახ.5.6) ავავოთ მთავარი მენიუ, რომლის პუნქტი „ლექციები“ გადააკოპირებს textBox1-ში ჩაწერილ ლექციის თემის დასახელებას label9-ში. მენიუს Exit პუნქტის არჩევისას კი პროგრამა დაასრულებს მუშაობას.



ნახ.5.6. ფორმის მაკეტი

textBox1-ში ჩაწერილი სტრიქონი „Windows-აპლიკაციის ვიზუალური დაპროგრამება“ (ნახ.5.7) მთავარი მენიუს „File->Open->ლექციები“ პუნქტის არჩევით ამ სტრიქონს გადააკოპირებს label9 ველში, რომელიც „Copy to“-ლებელის მარჯვენა მთავსებული (ის არ ჩანს).



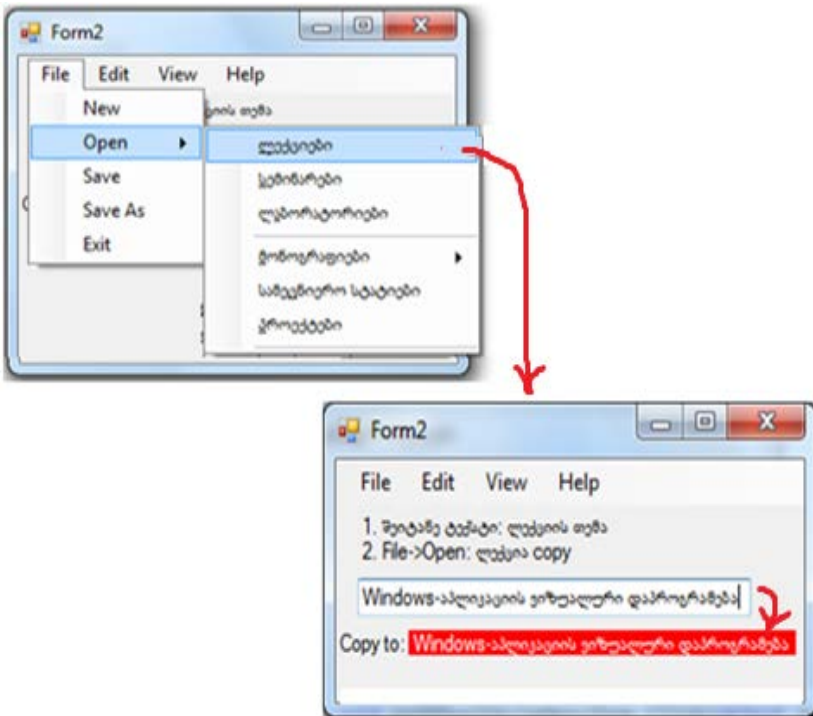
ნახ.5.7. textBox1-ში ჩაწერილი სტრიქონი

ამ მოვლენის დამმუშავებელს აქვს 5.1 ლისტინგში მოცემული კოდის ფორმა.

```
// ლისტინგი_5.1 ---- მთავარი მენიუ: File -> Open -> ლექციები ----  
private void ToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{  
    label9.BackColor = Color.Red;  
    label9.ForeColor = Color.White;  
    label9.Text = textBox1.Text;  
}  
private void exitToolStripMenuItem_Click(object sender, EventArgs e)  
{  
    Close();  
}
```

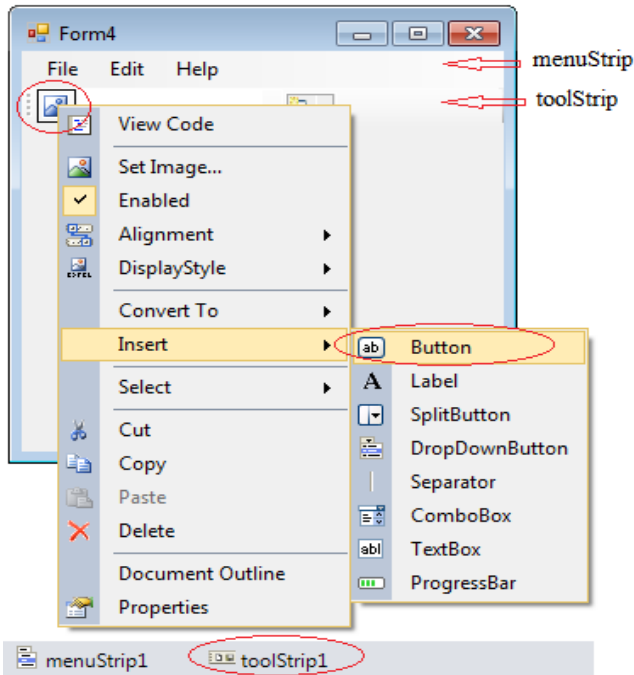
შედეგი მოცემულია 5.8 ნახაზზე, წითელი ფონით და თეთრი ტექსტით.



ნახ.5.8. Copy to -ს label-ში ჩაიწერა სტრიქონი textBox1-დან

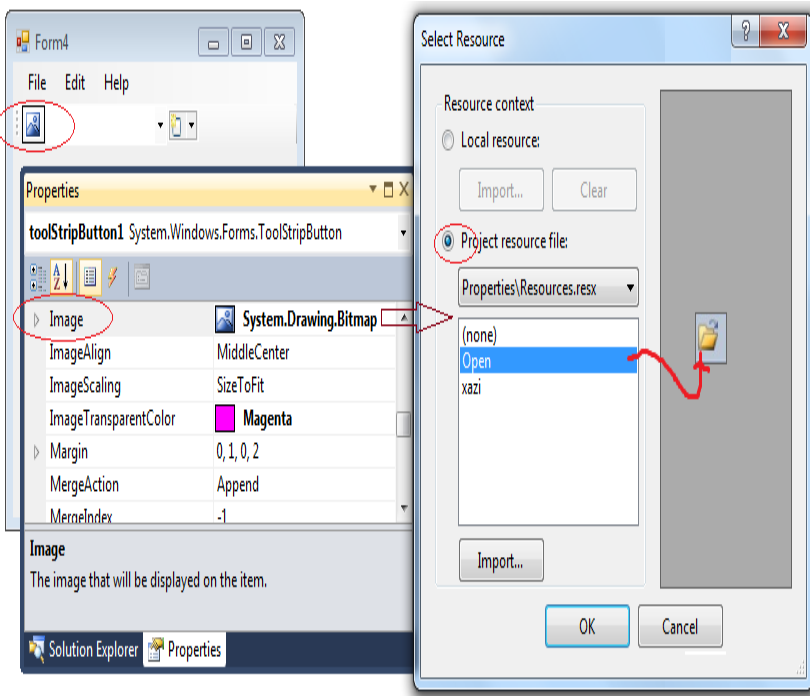
5.2. გრაფიკული მენიუს აგების ვიზუალური ელემენტები

კომპიუტერული სისტემების აგების დროს ინტერფეისში მთავარი მენიუს გარდა ხშირად ხმარობენ გრაფიკულ პიქტოგრამებს (icons), რაც უფრო ეფექტურს და მოქნილს ხდის მის გამოყენებას. C# ენაში ასეთი ვიზუალური ელემენტია Menus&Toolbars პანელის toolStrip ელემენტი (ნახ.5.1). მისი გადატანით ფორმაზე მივიღებთ 5.9 ნახაზზე ნაჩვენებ სურათს. საჭიროა ავირჩიოთ სახე: button, Label, ComboBox და ა.შ.



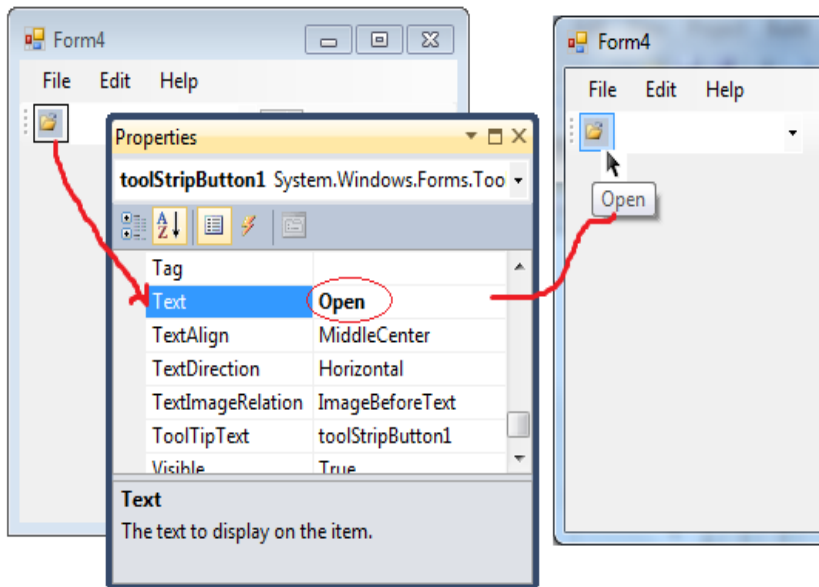
ნახ.5.9

აქ შესაძლებელია მაუსის მარჯვენა ღილაკით სტანდარტულად მიღებული პიქტოგრამის შეცვლა, ჯერ Properties-ში Image თვისების არჩევით და შემდეგ საჭირო პიქტოგრამის Import-ირებით დიალოგური ფანჯრიდან (ნახ.5.10).



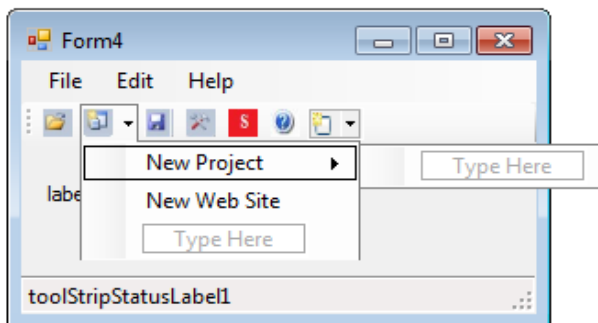
ნახ.5.10

ახალ პიქტოგრამას თვისებაში Text ჩაუწერეთ მისი ფუნქციის შესაბამისი სიტყვა, მაგალითად, Open. 5.11 ნახაზზე ჩანს მიღებული შედეგი.



ნახ.5.11

საბოლოო შედეგები რამდენიმე ახალი პიკტოგრამის ჩასმის შემდეგ, რომელთაგანაც ერთ-ერთი comboBox ტიპისაა, მოცემულია 5.12 ნახაზზე.

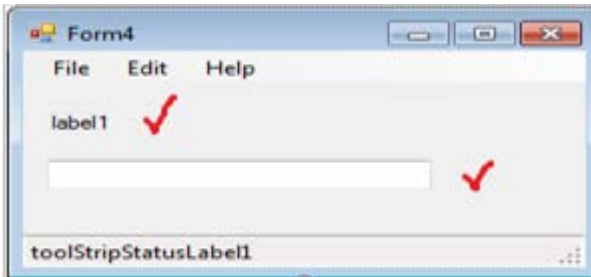


ნახ.5.12

5.3. კონტექსტური მენიუს აგების ვიზუალური ელემენტები

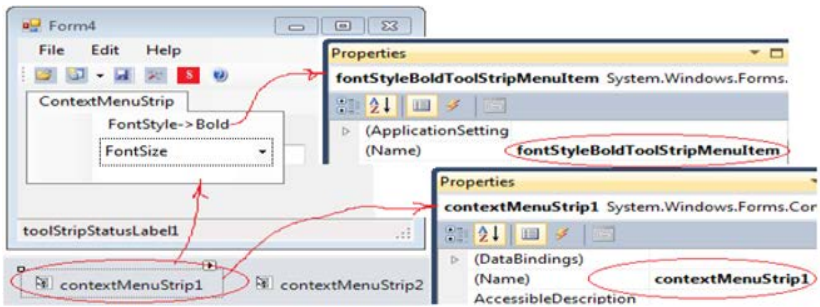
განვიხილოთ კონტექსტური მენიუს (მაუსის მარჯვენა ღილაკის ფუნქცია) აგების საკითხი ToolBox პანელის ContextMenuStrip ელემენტით (ნახ.5.1). კონტექსტური მენიუს სტრიქონთა მნიშვნელობების შესაბამისობაში მოყვანა ფორმის ან ფორმაზე დადებული ელემენტებისთვისაა საჭირო.

ამოცანა_5.3: ავაგოთ ფორმა, მაგალითად, 5.13 ნახაზზე მოცემულია სახით, რომელზეც label1 და textBox1 ელემენტებია დადებული. ამ ელემენტებისთვის უნდა შეიქმნას ორი კონტექსტური მენიუ. textBox1-ში ჩაწერილი სტრიქონი კონტექსტური მენიუთი უნდა გადაკოპირდეს label1-ში. შემდეგ label1-ში გადატანილი სტრიქონის ფორმა (სტილი, ზომა) უნდა შეიცვალოს კონტექსტური მენიუდან.

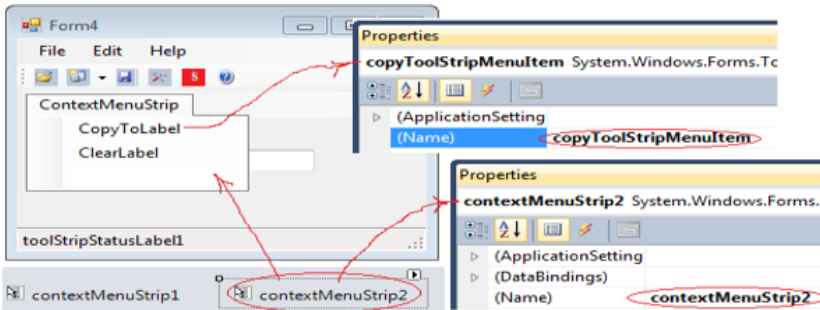


ნახ. 5.13

5.14-ა,ბ ნახაზებზე ნაჩვენებია Form4-ზე მოთავსებული label1 და textBox1 ელემენტებისთვის ჩვენ მიერ შექმნილი კონტექსტური მენიუები, შესაბამისად ContextMenuStrip1 და ContextMenuStrip2. კონტექსტური მენიუები აგებულია. ახლა ისინი უნდა „მივაბათ“ Form4-ის textBox1 და label1 ელემენტებს, რათა მაუსის მარჯვენა ღილაკმა იმუშაოს და ამ ელემენტებზე კურსორის მიტანისას გამოჩნდეს კონკრეტულად მისი შესაბამისი კონტექსტური მენიუ.

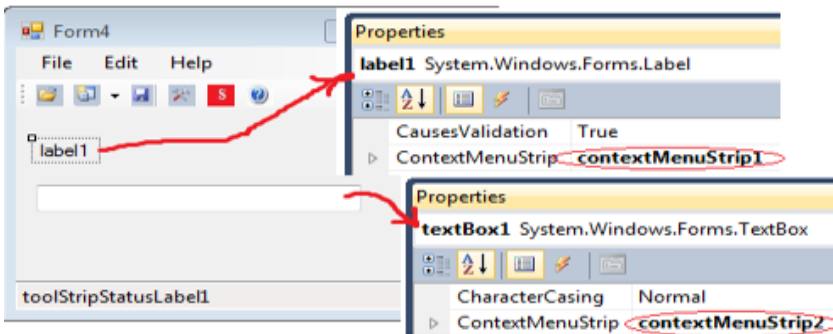


ნახ.5.14-ა



ნახ.5.14-ბ

ამისათვის მოვნიშნოთ label1 და მის Properties-ის ContextMenuStrip თვისებაში ჩავწეროთ contextMenuStrip1 მნიშვნელობა (ნახ.5.15). ასევე textBox1-სთვის ჩავწეროთ contextMenuStrip2.



ნახ.5.15

პროგრამული კოდი მოცემულია 5.2 ლისტინგში, შედეგები კი 5.16 – 5.18 ნახაზებზე.

// ლისტინგი_5.2 -- შრიფტის შეცვლა ---

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;

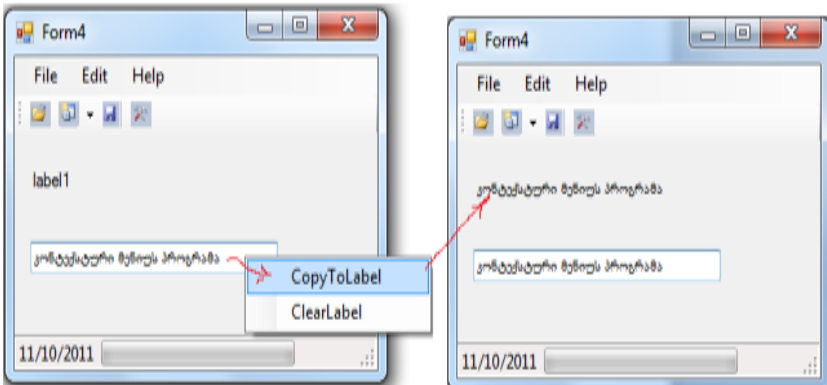
namespace WinMenus
{
    public partial class Form4 : Form
    {
        double DamtavrDro;
        public Form4()
        {
            InitializeComponent();
        }
        private void Form4_Load(object sender, EventArgs e)
        {
            toolStripStatusLabel1.Text = DateTime.Today.ToShortDateString();
        }
        private void Stop_Click(object sender, EventArgs e)
        {
            DamtavrDro = 0;
            timer1.Enabled = true;
        }
        private void timer_Tick(object sender, EventArgs e)
        {
            DamtavrDro += 0.1;
            if (DamtavrDro >= 5)
                Close();
            else
                toolStripProgressBar1.Value = (int)DamtavrDro;
            textBox1.Text = DamtavrDro.ToString();
        }
    }
}
```

```

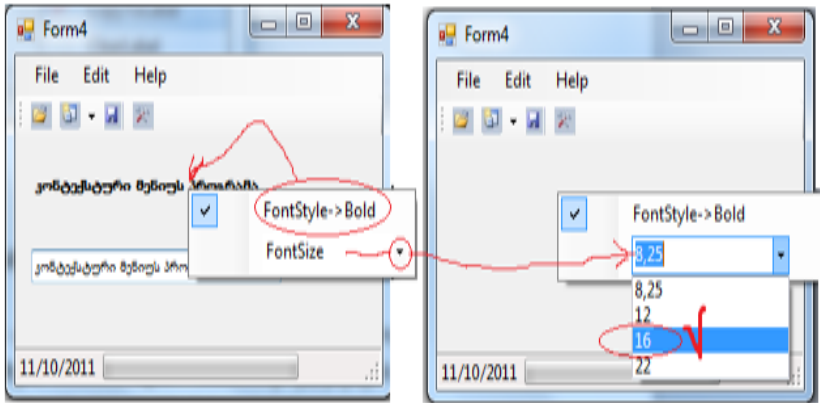
    }
private void copyToolStripMenuItem_Click(object sender, EventArgs e)
{
    label1.Text = textBox1.Text;
    if (label1.Text == "")
        label1.Text = "(leer)";
}
private void clearLabelToolStripMenuItem_Click(object sender,
                                                EventArgs e)
{
    label1.Text = "";
}
private void fontStyleBoldToolStripMenuItem_Click(object sender,
                                                    EventArgs e)
{
    label1.Font = new Font(label1.Font.FontFamily, label1.Font.Size,
                           label1.Font.Style ^ FontStyle.Bold);
    fontStyleBoldToolStripMenuItem.Checked =
        !fontStyleBoldToolStripMenuItem.Checked;
}
private void fontSize16ToolStripMenuItem_Click(object sender,
                                                EventArgs e)
{
    label1.Font = new Font(label1.Font.FontFamily, label1.Font.Size,
                           label1.Font.Style ^ FontStyle.Italic);
    fontStyleBoldToolStripMenuItem.Checked =
        !fontStyleBoldToolStripMenuItem.Checked;
}
private void toolStripComboBox1_TextChanged(object sender,
                                            EventArgs e)
{
    double FontSize;
    try
    {
        FontSize = Convert.ToDouble(toolStripComboBox1.Text);
    }
}

```

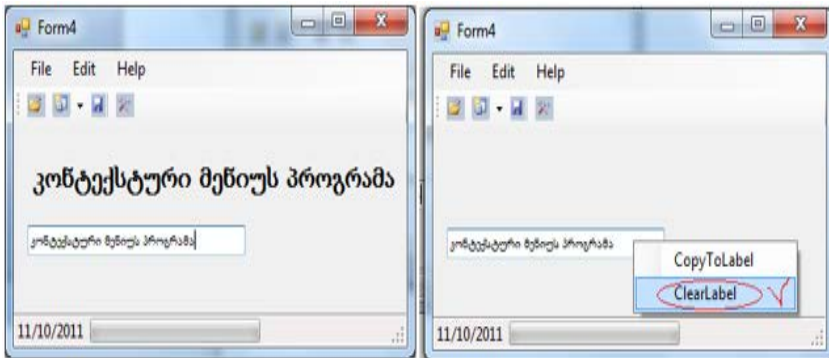
```
    }  
    catch  
    {  
        FontSize = 8.25;  
    }  
  
    label1.Font = new Font(label1.Font.FontFamily, (float)FontSize,  
        label1.Font.Style);  
}  
private void toolStripComboBox1_Click(object sender, EventArgs e)  
{  
    toolStripComboBox1.Items.Clear();  
    toolStripComboBox1.Items.Add("8,25");  
    toolStripComboBox1.Items.Add("12");  
    toolStripComboBox1.Items.Add("16");  
    toolStripComboBox1.Items.Add("22");  
    toolStripComboBox1.SelectedIndex = 0;  
}  
}  
}
```



ნახ.5.16. შედეგები



ნახ.5.17



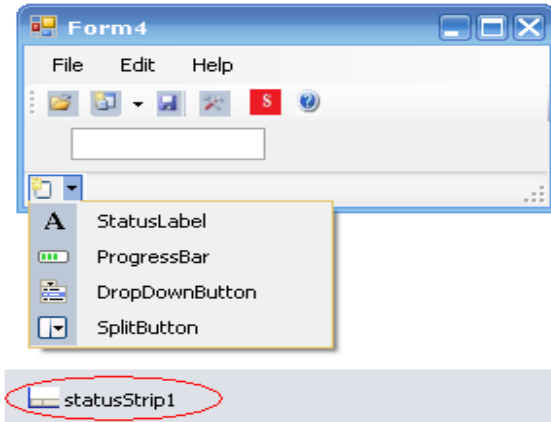
ნახ.5.18

სავარჯიშო დავალება:

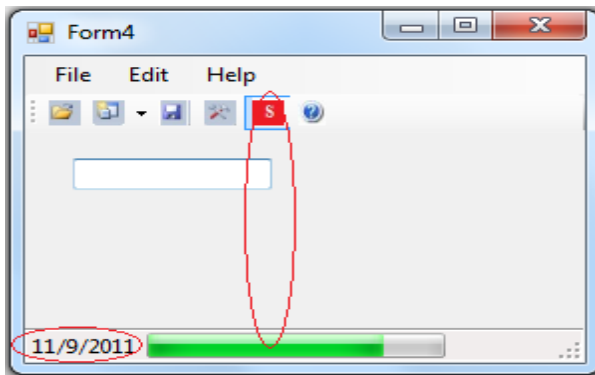
ამოცანა_5.3: Form2-ზე მთავარი მენიუს Edit პუნქტში ჩავდოთ კომბობოქსი შრიფტების არჩევის მიზნით, კერძოდ AcadNusx და AcadMtavr მნიშვნელობებით. ამ ფონტების არჩევით უნდა შეიცვალოს label9-ში ჩაწერილი სტრიქონის შრიფტი.

5.4. სტატუსის პანელის შექმნა

სტატუსის პანელის დანიშნულებაა პროგრამის მუშაობის პროცესში საჭირო ინფორმაციის ასახვა ეკრანზე. იგი იქმნება Toolbox-იდან StatusStrip ელემენტის (ნახ.5.1) გადმოტანით ფორმაზე და თავსდება ფორმის ქვედა ნაწილში (ნახ.5.19). მისთვის ძირითადად გამოიყენება label - ტიპი.



ნახ.5.19-ა



ნახ.5.19-ბ

ავირჩიოთ ჯერ StatusLabel, რომელშიც გამოვიტანთ სისტემურ თარიღს (ანუ დღევანდელს) და მეორე, ProgressBar, რომელიც იმუშავებს გრაფიკულ მენიუდან „S“ (stop) ღილაკის ამოქმედებით. იგი გვაძლევს პროგრამული სისტემის დამთავრების პროცესის ხანგრძლივობას (წამებში). 5.3 ლისტინგში მოცემულია აღწერილი პროცესის პროგრამული კოდი.

// ლისტინგი_5.3 -- შრიფტის შეცვლა ---

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;
namespace WinMenus
{
public partial class Form4 : Form
{
    double DamtavrDro; // პროგრამის დამთავრების დროის ცვლადი
    public Form4() { InitializeComponent(); }
    private void Form4_Load(object sender, EventArgs e)
    { // სტატუს-ველი მიმდინარე თარიღის გამოსატანად ---
        toolStripStatusLabel1.Text =
            DateTime.Today.ToShortDateString();
    }
    private void Stop_Click(object sender, EventArgs e)
    { // მენიუდან “S” ამოქმედება
        DamtavrDro = 0;
        timer1.Enabled = true; // საათის ჩართვა სტატუს-ველის
            // ProgressBar-სთვის
    }
    private void timer_Tick(object sender, EventArgs e)
    { // პროგრამის დამთავრების ხანგრძლივობის დათვლა -----
        DamtavrDro += 0.1;
    }
}
```

```
if (DamtavrDro >= 5)
    Close();
else
    toolStripProgressBar1.Value = (int)DamtavrDro;

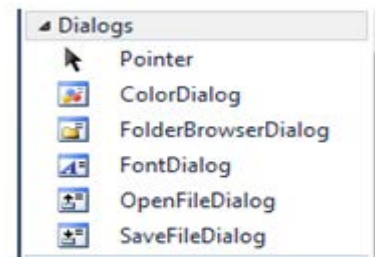
textBox1.Text = DamtavrDro.ToString();
}
}
}
```

5.5. C# ენის ვიზუალური სტანდარტული დიალოგური საშუალებანი

დაპროგრამების ვიზუალურ C# ენაში არსებობს ხუთი სახის დიალოგური კლასი, რომლებიც ხშირად გამოიყენება პროგრამული პროექტების აგების პროცესში:

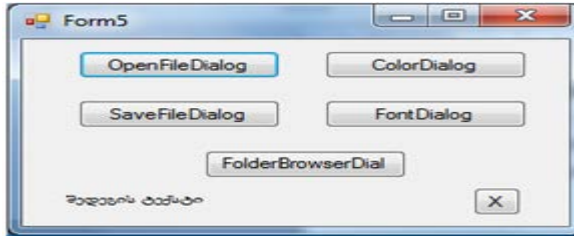
- OpenFileDialog
- SaveFileDialog
- FolderBrowserDialog
- ColorDialog და
- FontDialog.

განვიხილოთ ეს დიალოგები დეტალურად (ნახ.5.20).



ნახ.5.20. Toolbox-ის სტანდარტული დიალოგური
ელემენტები

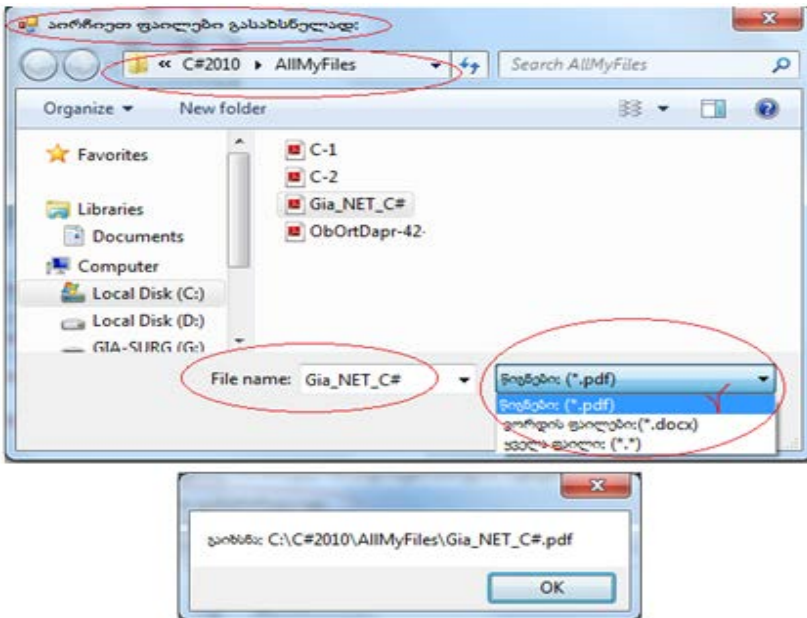
ფორმაზე გადმოვიტანოთ ხუთივე ელემენტი და ჩავატაროთ გაცნობითი ექსპერიმენტი (ნახ.5.21).



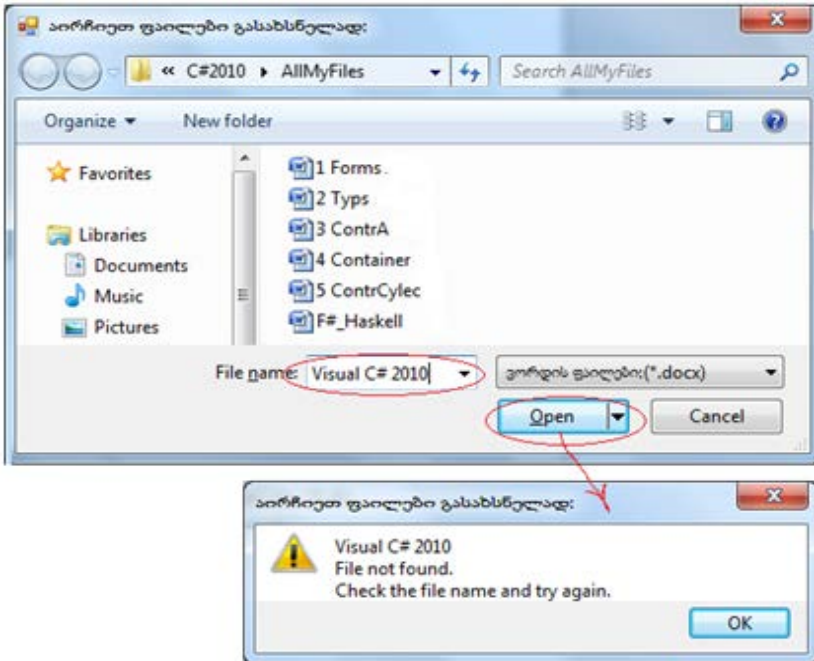
ნახ.5.21

➤ **OpenFileDialog** კლასის ობიექტის დანიშნულებაა გასახსნელი ფაილის ამორჩევა, მითითებული ფოლდერის (InitialDirectory), ფილტრის (Filter - მაგალითად, ფაილის ტიპით და დიალოგური ველის სათაურის (Title) მიხედვით (ნახ.5.22 და 2.23).

დიალოგის შედეგი ფაილის სახელის თვისებისთვის იქნება - FileName.



ნახ.5.22. დადებითი შედეგით



ნახ.5.23. უშედეგოდ დასრულება

// ლისტინგი_5.4 --- OpenFileDialog -----

```
private void button1_Click(object sender, EventArgs e) // openFileDialog
{
    OpenFileDialog f = new OpenFileDialog();
    f.InitialDirectory = "c:\\C#2010\\AllMyFiles";
    f.Filter = "წიგნები (*.pdf)|*.pdf" +
              " ვორდის ფაილები (*.docx)|*.docx|" +
              " ყველა ფაილი (*.*)|*.*";
    f.Title = "აირჩიეთ ფაილები გასახსნელად:";
    if (f.ShowDialog() == DialogResult.OK)
        MessageBox.Show("გაიხსნა: " + f.FileName);
    else
        MessageBox.Show("უშედეგოდ დასასრული!");
}
```

➤ **SaveFileDialog** კლასის ობიექტის დანიშნულებაა იმ ფაილის შეტანა ან ამორჩევა, რომელიც შენახულ უნდა იქნას. შენახვის და დიალოგის განსახორციელებლად მითითებულ უნდა იქნას ფოლდერის (InitialDirectory), ფილტრის (Filter - მაგალითად, ფაილის ტიპით და დიალოგური ველის სათაური (Title)).

// ლისტინგი_5.5 --- SaveFileDialog -----

```
private void button2_Click(object sender, EventArgs e)
{
    SaveFileDialog fs = new SaveFileDialog();
    fs.InitialDirectory = "c:\\C#2010\\AllMyFiles";
    fs.Filter = "წიგნები: (*.pdf)|*.pdf| +
                " ვორდის ფაილები: (*.docx)|*.docx| +
                " ყველა ფაილი: (*.*)|*.*";
    fs.Title = "ფაილების არჩევა შესანახად";
    if (fs.ShowDialog() == DialogResult.OK)
        MessageBox.Show("შენახვა: " + fs.FileName);
    else
        MessageBox.Show("უშედეგო დასასრული!");
}
```

➤ **FolderBrowserDialog** კლასის ობიექტის დანიშნულებაა კატალოგის (ფოლდერის) ამორჩევა, რომელიც იქნება მომდევნო პროგრამული პროცედურების საბაზო წერტილი. შესაძლებელია ასევე ახალი კატალოგის შექმნა. დიალოგური ფორმის გახსნის წინ საჭიროა შემდეგი პარამეტრების მიწოდება: RootFolder: კატალოგი, რომელიც უნდა გამოჩნდეს დიალოგის ველში. ShowNewFolderButton: ახალი კატალოგის შექმნისათვის საჭირო ბუტონის მითითება. Description: დიალოგური ველის სათაური.

// ლისტინგი_5.6 --- FolderBrowserDialog -----

```
private void button3_Click(object sender, EventArgs e)
{
```

```
FolderBrowserDialog fb = new FolderBrowserDialog();  
fb.RootFolder = Environment.SpecialFolder.MyDocuments;  
fb.ShowNewFolderButton = false;  
fb.Description = "კატალოგის არჩევა";
```

```
if (fb.ShowDialog() == DialogResult.OK)  
    MessageBox.Show("წვდომა კლატალოგზე: " + fb.SelectedPath);  
else  
    MessageBox.Show("უშედეგო დასასრული !");  
}
```

სტრიქონით:

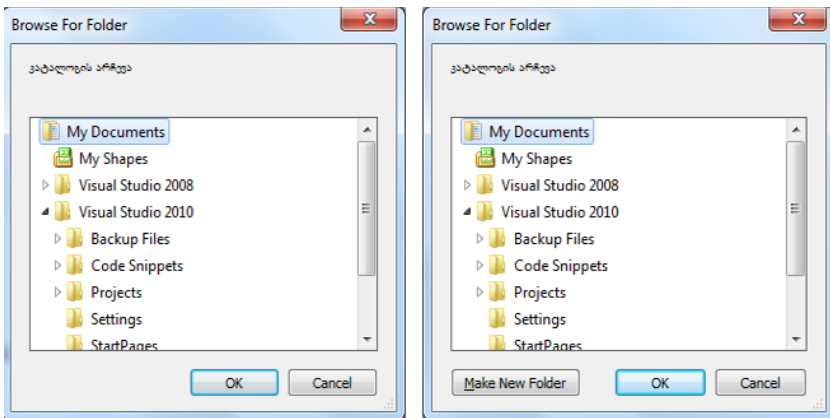
```
fb.RootFolder=Environment.SpecialFolder.MyDocuments;
```

- ფსევურ კატალოგად, ჩვენ შემთხვევაში აიღება „MyDocuments“.

სტრიქონით:

```
fb.ShowNewFolderButton = false;
```

- ახალი კატალოგი არ იქმნება, ხოლო თუ არის **“true”**, მაშინ ფორმაზე ჩნდება ბუტონი “Make New Folder” (ნახ.5.24).

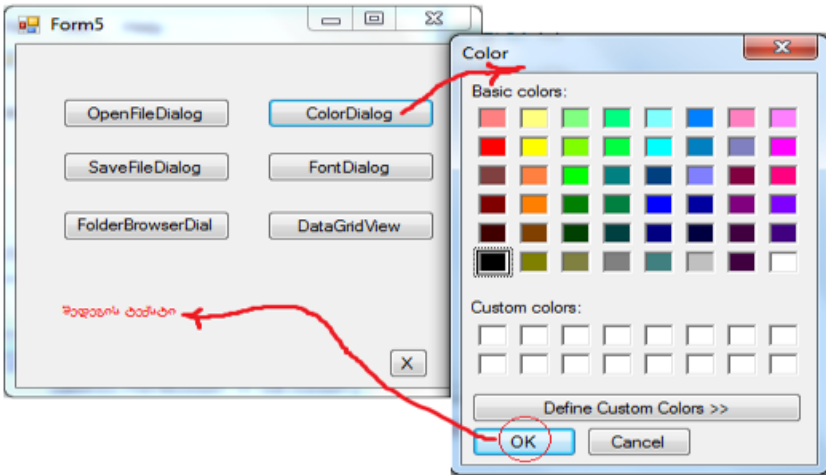


ნახ.5.24

➤ **ColorDialog** კლასის ობიექტის დანიშნულებაა ფერის არჩევა ფორმაზე მოთავსებული მონიშნული ელემენტისათვის (ლისტინგი_5.7 და ნახ.5.25).

// ლისტინგი_5.7 --- ColorDialog -----

```
private void button4_Click(object sender, EventArgs e)
{
    ColorDialog cd = new ColorDialog();
    if (cd.ShowDialog() == DialogResult.OK)
        label1.ForeColor = cd.Color;
    else
        MessageBox.Show("უშედეგო დასასრული");
}
```



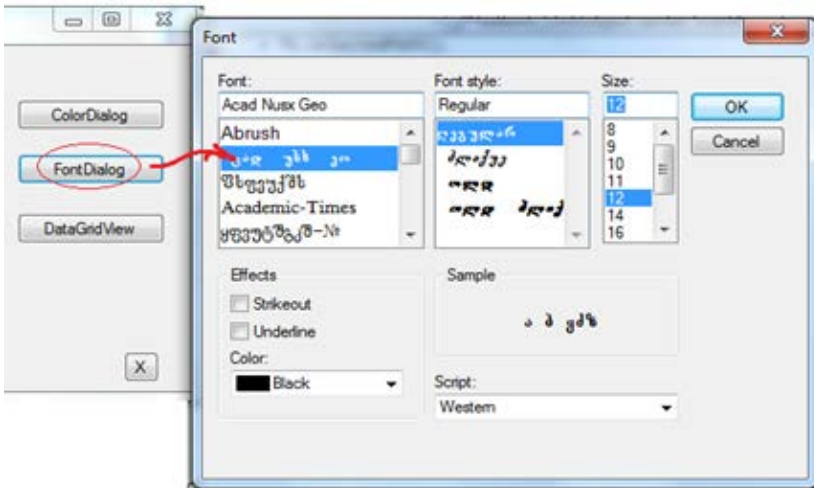
ნახ.5.25

➤ **FontDialog** კლასის ობიექტის დანიშნულებაა შრიფტის (ფონტის) არჩევა ფორმაზე მოთავსებული მონიშნული ელემენტისთვის (ლისტინგი_5.8 და ნახ.5.26).

// ლოსტინგი_5.8 --- FontDialog -----

```
private void button5_Click(object sender, EventArgs e)
{
    FontDialog fd = new FontDialog();
    fd.ShowColor = true;
    fd.MaxSize = 22;
    fd.MinSize = 8;

    if (fd.ShowDialog() == DialogResult.OK)
    {
        label1.Font = fd.Font;
        label1.ForeColor = fd.Color;
    }
    else
    {
        MessageBox.Show("უშედეგო დასასრული");
    }
}
```



ნახ.5.26

თავი 6 რეკურსიული ფუნქციები და შემთხვევით რიცხვთა გენერატორი

პროგრამული აპლიკაციების აგებისას განსაკუთრებული ადგილი უჭირავს სტრუქტურული დაპროგრამების კონცეფციას და პროცედურების მართვის ისეთ საბაზო ელემენტებს, როგორცაა განშტოებები (if...else...), ციკლები (for, while, do...while, foreach...in) და გადამრთველები (switch). მათ საფუძველზე აიგება რეკურსიული ფუნქციები. C# ენა რეკურსიული ენაა (მის ყოველ ფუნქციას (პროგრამას) შეუძლია როგორც სხვა ფუნქციის, ასევე საკუთარი თავის გამოძახება). ასეთი ფუნქციები ხშირად გამოიყენება რიცხვთა მწკრივებთან სამუშაოდ (ჩვენ აქ განვიხილავთ ფიბონაჩის რიცხვთა მწკრივის ამოცანებს). აქვე შემოვიტანთ შემთხვევით რიცხვთა გენერატორის ცნებას და განვიხილავთ პროგრამული პროექტების აგების საილუსტრაციო მაგალითებს.

6.1. ციკლები და რეკურსიული ფუნქციები

ამოცანა_6.1: ავაგოთ პროექტი C# პროგრამული კოდით, რომელიც დაითვლის $n!$ (ფაქტორიალის) მნიშვნელობას წინასწარ განსაზღვრული n -რიცხვისათვის. როგორც ცნობილია,

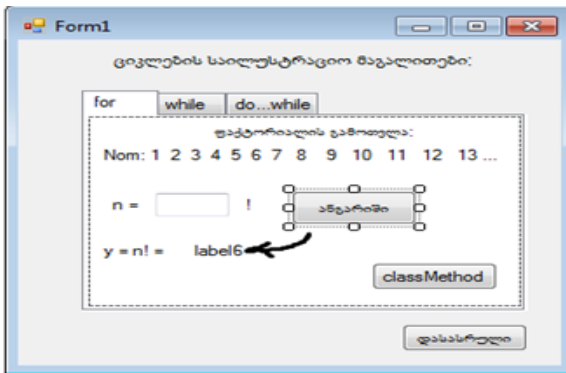
$y = n! = 1 * 2 * 3 * \dots * n$, სადაც n მოცემული მთელი რიცხვია.

ფაქტორიალის გამოთვლის ალგორითმული გადაწყვეტა შესაძლებელია ციკლით და რეკურსიული პროცედურით. ამჯერად for-ციკლი გამოვიყენოთ:

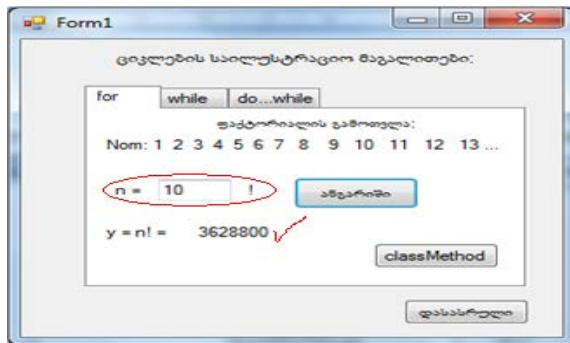
```
// ლისტინგი_6.1 --- ფაქტორიალის ანგარიში for ციკლით ---  
private void button2_Click(object sender, EventArgs e)  
{  
    int i; double Fac;  
    string ns = textBox1.Text;
```

```
int n = Convert.ToInt32(ns);
for (i = n, Fac = 1.0; i > 0; i--) // კლებადი ციკლი
{
    Fac *= i;
}
label6.Text = Fac.ToString();
}
```

6.1-ა ნახაზზე მოცემულია ფორმაზე TabControl კლასით შექმნილი მულტივერდების სურათი, რომლის გადამრთველებზეც მითითებულია ციკლის ტიპის (for, while, do...while) დასახელებები. შედეგი ასახულია 6.1-ბ ნახაზზე.



ნახ.6.1-ა



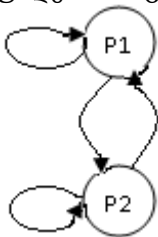
ნახ.6.1-ბ. შედეგი

ახლა განვიხილოთ იგივე ამოცანის გადაწყვეტა რეკურსიის გამოყენებით. რეკურსია ნიშნავს ისეთ პროცედურას, როდესაც ერთ პროგრამას (ან მეთოდს) შუძლია გამოიძახოს მეორე პროგრამა (ან მეთოდი) და პირიქითაც, ან შეუძლია თავის თავის გამოძახება ციკლურად (ნახ.6.2).

ამოცანა_6.2: აიგოს ფაქტორიალის გამოთვლის პროგრამა. ამ შემთხვევაში კლასის შიგნით შეიქმნას სპეციალური მეთოდი, რომელიც გაიანგარიშებს მისთვის გარედან მიწოდებული რიცხვის (n) ფაქტორიალს **რეკურსიის პრინციპის** საფუძველზე და დააბრუნებს შედეგს return-ოპერატორით.

ამოცანის გადაწყვეტის კოდის ფრაგმენტი მოცემულია 6_2 ლისტინგზე, ხოლო შედეგები გამოიტანება classMethod-ლილაკით (ნახ.6.1) Form2-ფანჯარაში (ნახ.6.3-ა). ბ-ნახაზზე ნაჩვენებია მაქსიმალური შესაძლო შედეგი.

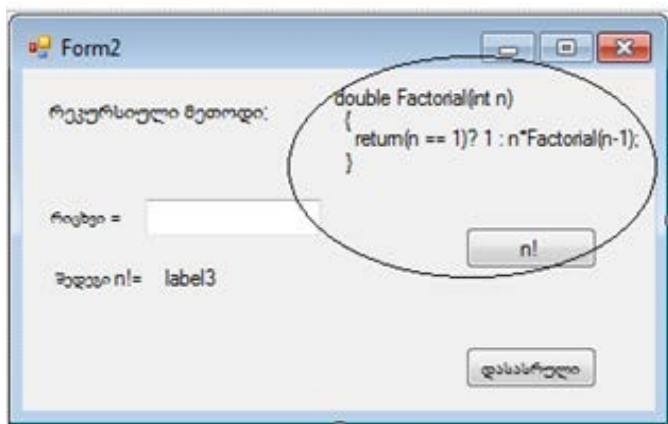
კომენტარი: `button1_Click` მოვლენით (ლილაკის ამოქმედება) გამოძახებულ იქნა მეთოდი (ფუნქცია) `Factorial(int n)` გადასაცემი `n`-არგუმენტით. მართვა გადაეცა `double Factorial()`-მეთოდს, რომლის `return`-ოპერატორში რეალიზებულია რეკურსია, ანუ `Factorial(n-1)` იძახებს „თავის-თავს“, მანამ, სანამ `n` არ შემცირდება 1-მდე. რეალურად, თუ `n=5`, ციკლში ხორციელდება: $5*4*3*2*1$ და ეს ნამრავლი, ანუ ფაქტორიალის მნიშვნელობა უბრუნდება `Fac`-ცვლადს.



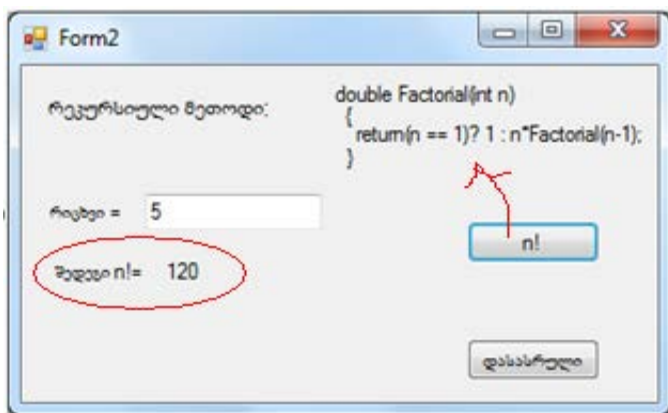
ნახ.6.2

```
// ლისტინგი_6.2 -- რეკურსიული მეთოდი -
using System;
using System.Windows.Forms;
namespace WinFormCycle
{
```

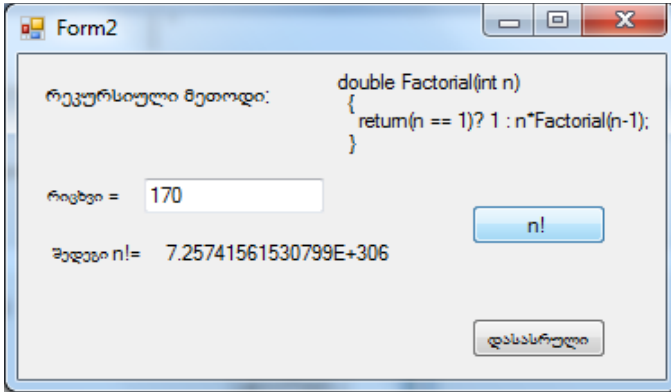
```
public partial class Form2 : Form
{
    public Form2()
        {InitializeComponent(); }
private void button1_Click(object
        sender, EventArgs e)
{
    double Fac;
    string ns = textBox1.Text;
    int n = Convert.ToInt32(ns);
    Fac=Factorial(n); // მეთოდის გამოძახება
    label3.Text = Fac.ToString();
}
// რეკურსიული მეთოდი ფაქტორიალის
// საანგარიშოდ ---
double Factorial(int n)
{ // რეკურსია !
    return(n == 1)? 1:n*Factorial(n-1);
}
}
}
```



ნახ.6.3-ა1



ნახ.6.3-ა2. შედეგი



ნახ.6.3-ბ. n=170 მაქსიმალური შესაძლო რიცხვი

ამოცანა_6.3: ავაგოთ პროექტი C# პროგრამული კოდით, რომელიც:

ა) გაითვლის ნებისმიერი მითითებული მთელი რიცხვისათვის **ფიბონაჩის** მწკრივში მის მომდევნო „ფიბონაჩის რიცხვის“ მნიშვნელობას (მაგალითად, რიცხვი=50, რომელია მის მარჯვნივ მდგარი უახლოესი ფიბონაჩის რიცხვი ?);

ბ) გაითვლის ფიბონაჩის რიცხვის მნიშვნელობას მე-n ელემენტისათვის ფიბონაჩის რიცხვთა მწკრივში. (მაგალითად, რას უდრის მწკრივში რიგით მე-15 ფიბონაჩის რიცხვი ?);

გ) გაითვლის ფიბონაჩის მწკრივის რიცხვთა ჯამს მითითებული მე-n ელემენტის ჩათვლით (მაგ., რას უდრის ფიბონაჩის მწკრივის 1-20 რიცხვების ჯამი ?).

როგორც ცნობილია, ფიბონაჩის რიცხვთა მწკრივი შემდეგი სახისაა:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

ანუ, რიცხვთა მწკრივის აგების წესი ასეთია: მომდევნო რიცხვი არის წინა ორი რიცხვის ჯამი.

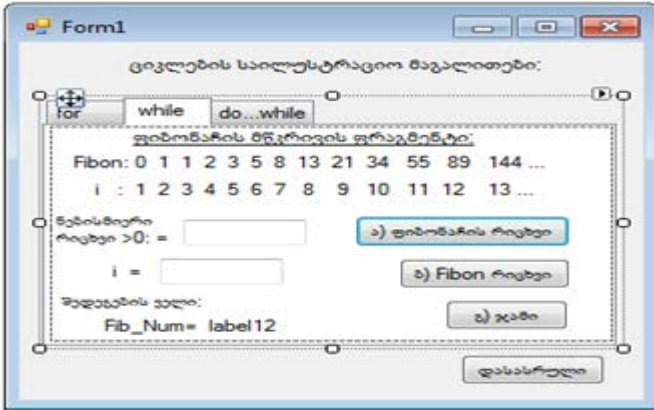
შენიშვნა: „ფიბონაჩის მწკრივი“ მათემატიკურ სამყაროში 1200 წელს შეიქმნა ლეონარდო პიზანსკის (ვსევდონიმი „ფიბონაჩი“) მიერ, თუმცა ასეთი მწკრივი ძველ ინდოეთშიც ყოფილა ცნობილი (იხ. მასალები ვიკიპედიაში). ფიბონაჩის მწკრივს მრავალი პრაქტიკული გამოყენება აქვს დღესაც სხვადასხვა დარგებში (ეს ცალკე თემა). მისი ასეთი აქტუალურობის და მნიშვნელობის გამო ჩვენ განვიხილავთ ციკლური და რეკურსიული ოპერაციების პროგრამული კოდების მაგალითებს ფიბონაჩის რიცხვებისათვის.

6.4-ა ნახაზზე ნაჩვენებია მულტიგვერდი „while“, რომელზეც განლაგებულია ფიბონაჩის მწკრივის ფრაგმენტი (Fibon), ნატურალურ რიცხვთა მწკრივი (i), ჩვენი სამი ამოცანის სამი ბუტონი (ა,ბ,გ), საწყისი მნიშვნელობების შესატანი ორი ტექსტბოქსი („რიცხვი>0: =“ და „i=“) და შედეგის გამოსატანი ველი (Fib_Num=label12).

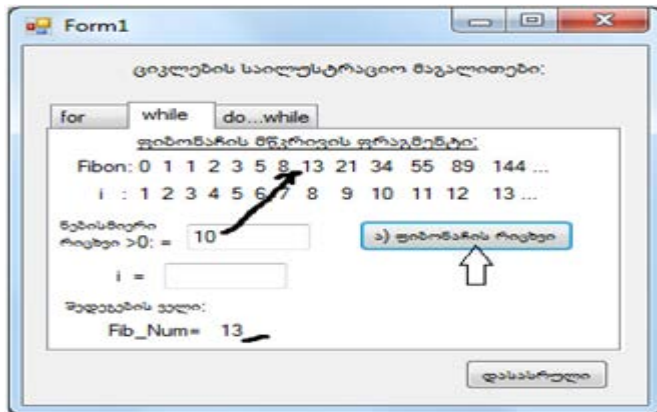
2-ა) ამოცანის გადაწყვეტა:

პროგრამის ამუშავების და „while“ გვერდზე გადართვის შემდეგ საწყისი მონაცემის შეტანა უნდა განხორციელდეს ტექსტბოქსში (ნახ.6.4-ბ). როგორც კი მოხდება ამ ტექსტბოქსის მნიშვნელობის შეცვლა, მაშინვე ფორმაზე დაიმალება ბ და გ ბუტონები. ა-ბუტონი მზადაა დააფიქსიროს შედეგი. მაგალითად, რიცხვისთვის 10, ფიბონაჩის რიცხვის მნიშვნელობა იქნება 13. რიცხვისთვის 100, იქნება 144 და ა.შ.

while() - ციკლის გამოყენებით შექმნილი პროგრამის კოდი, რომელიც ამუშავდება ა-ბუტონის, როგორც მოვლენის ამოქმედებისას, მოცემულია 6_3 ლისტინგში:



ნახ.6.4-ა



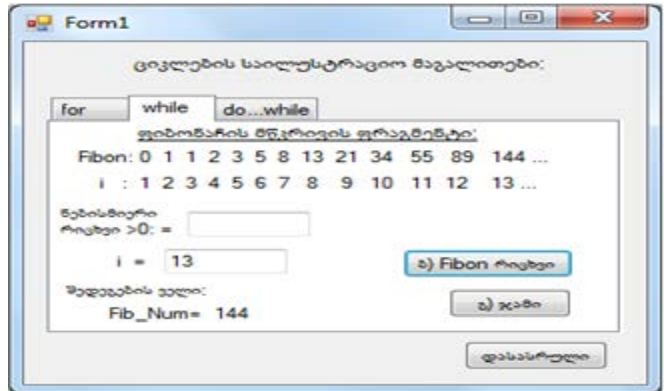
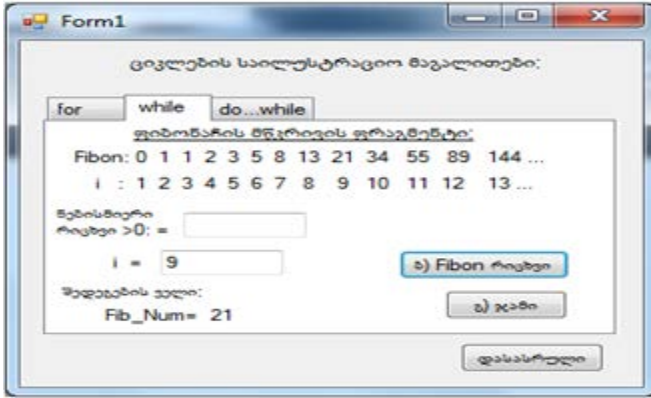
ნახ.6.4-ბ

```
// ლისტინგი_6.3 – while ციკლი + ფიბონაჩი-2-ა ----
private void button3_Click(object sender, EventArgs e)
{ // ა-ლილაკი
  int Previous, Next; // ფიბონაჩის მწკრივის „წინა“ და
  // „მომდევნო“ რიცხვები
```

```
Previous = -1;
Next = 1;
string ns = textBox2.Text; // საწყისი მონაცემის შეტანა
int n = Convert.ToInt32(ns); // ტიპის გარდაქმნა
int fib = 1;
while (fib <= n) // ციკლი !!!
{
    fib = Previous + Next;
    Previous = Next;
    Next = fib;
    label12.Text = " " + fib.ToString(); // შედეგის გამოტანა
}
}
```

2-ბ) ამოცანის გადაწყვეტა:

თუ საწყისი მნიშვნელობა შეიტანება მეორე ტექსტბოქსში (ნახ.6.5), მაშინ იცვლება გვერდის მდგომარეობა, ანუ გააქტიურდება შესაბამისი ამოცანების ბ და გ ლილაკები და დაიმალება არასაჭირო ა-ლილაკი.



ნახ.6.5

6.4_ლისტინგში ნაჩვენებია ბ-ლილაკის კოდი, რომელიც ფიბონაჩის რიცხვის გასათვლელად გამოიყენებს რეკურსიულ მეთოდს (Fibonacci(int n) - ფუნქციას),

// ლისტინგი_6.4-- ფიბონაჩის რიცხვის გამოთვლა რეკურსით --

private void button5_Click(object sender, EventArgs e)

{

int Fibon;

string ns = textBox3.Text;

int n = Convert.ToInt32(ns);


```

Fibon=Fibonacci(n-1);
label12.Text = Fibon.ToString();
}
int Fibonacci(int n) // რეკურსიული მეთოდი
{
    return n > 1 ? Fibonacci(n-1) + Fibonacci(n-2) : n;
}

```

პროგრამაში გათვალისწინებულია სამი ბუტონის გამოჩენა/არგამოჩენა ფორმაზე, რეალიზებულია მათი შესაბამისი ტექსტბოქსის ელემენტების ხილვადობის პარამეტრების მართვით, რაც 6_5 ლისტინგშია აღწერილი.

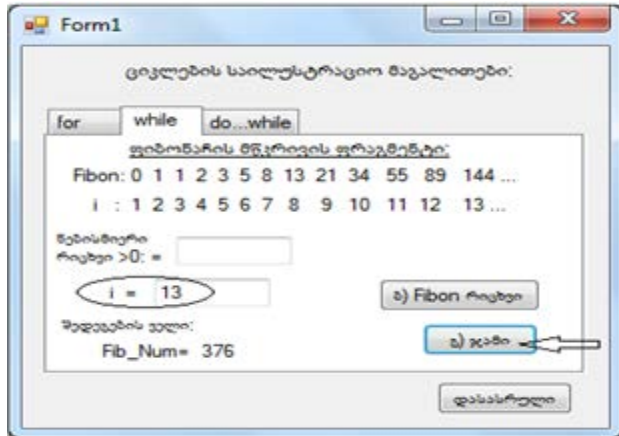
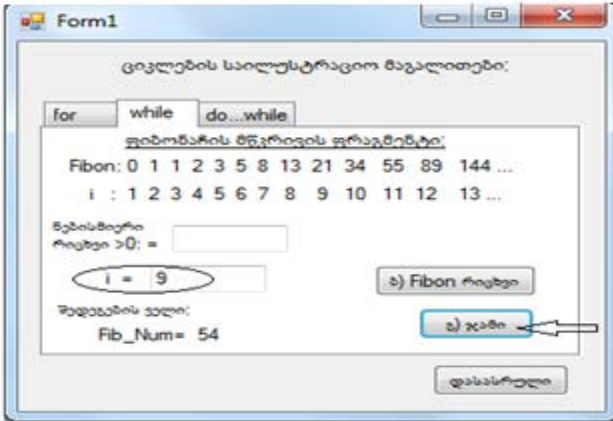
```

// ლისტინგი_6.5 --- Visible თვისების მართვა -----
private void textBox2_TextChanged(object sender,
                                   EventArgs e) // ერთი textBox
{
    if (textBox2.Text != "")
    {
        button3.Visible = true; // ჩაირთო button3
        button5.Visible = false; // გამოირთო button5
        button6.Visible = false; // გამოირთო button6
    }
}
private void textBox3_TextChanged(object sender,
                                   EventArgs e) // მეორე textBox
{ if (textBox3.Text != "")
  {
    button5.Visible = true; // გამოირთო button5
    button6.Visible = true; // გამოირთო button6
    button3.Visible = false; // გამოირთო button3
  }
}
}

```

2-გ.) ამოცანის გადაწყვეტა:

6.6 ნახაზზე ნაჩვენებია გ-ლილაკის ამოქმედებით არჩეული რიცხვებისთვის მწკრივის ჯამის გაანგარიშების შედეგები.



ნახ.5.6

// ლისტინგი_5.6-გ ფიბონაჩის მწკრივის რიცხვთა ჯამის გამოთვლა -

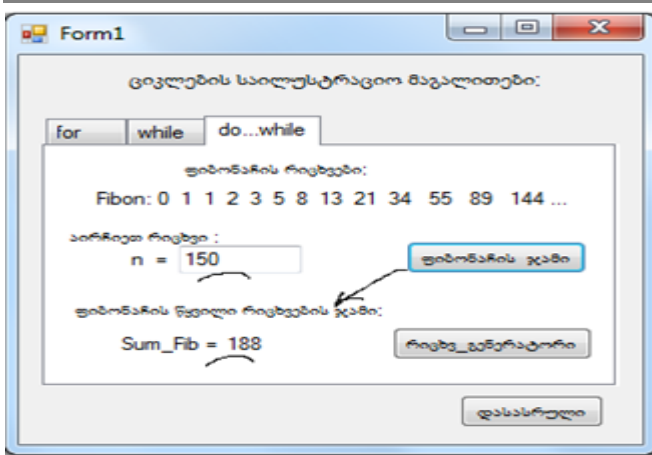
private void button6_Click(object sender, EventArgs e)

```
{
string ns = textBox3.Text;
int n = Convert.ToInt32(ns);
int zinaFib = 0, momdevnoFib = 1;
int SumFib = 0;
int i = 1;
while (i <=n)
{
int temp = momdevnoFib;
momdevnoFib += zinaFib;
SumFib += zinaFib;
zinaFib = temp;
i++;
}
label12.Text = SumFib.ToString();
}
```

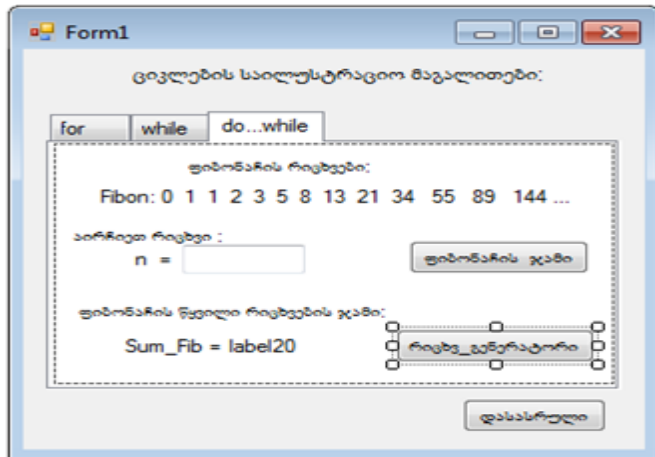
6.2. ციკლები და შემთხვევით რიცხვთა გენერატორი

ახლა გადავიდეთ do...while ციკლის განხილვაზე (ნახ.6.7).

ამოცანა_6.4: საჭიროა პროგრამული კოდის აგება, რომელიც იანგარიშებს do...while ციკლის გამოყენებით ფიბონაჩის რიცხვთა მწკრივის ლუწი რიცხვების ჯამს. მწკრივის სიგრძე, ანუ მისი ელემენტების რაოდენობა უნდა მიეთითოს ტექსტბოქსში შეტანილი n-რიცხვით, მაგალითად, 150. შედეგი გამოტანილ უნდა იქნეს label20 -ში.



ნახ.6.7-ა



ნახ.6.7-ბ: შედეგი

ფიბონაჩის მწკრივის რიცხვთა ჯამის განსაზღვრის სარეალიზაციო კოდი მოცემულია 5.7_ლისტინგში.

// ლისტინგი_6.7 – do...while ციკლი ფიბონაჩისთვის ----

```
private void button7_Click(object sender, EventArgs e)
```

```
{
```

```
    string ns = textBox4.Text;
```

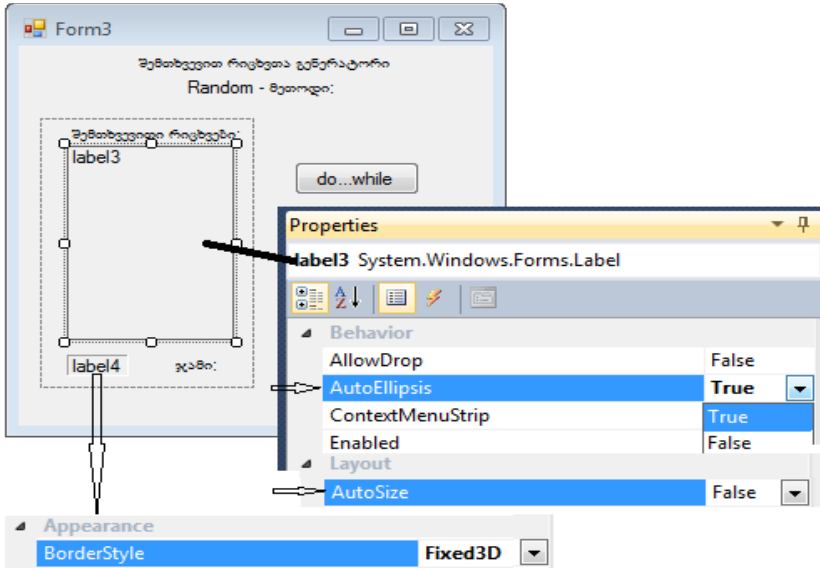
```

uint n = Convert.ToUInt32(ns);
uint SumFib = 0;
uint momdevnoFib = 1;
uint zinaFib = 0;
do
{
    uint tmp = momdevnoFib;
    momdevnoFib = momdevnoFib + zinaFib;
    zinaFib = tmp;
    if (momdevnoFib % 2 == 0 && momdevnoFib < n)
        // წვილობაზე შემოწმება
        SumFib += momdevnoFib;
} while (momdevnoFib < n)
    ; // ცარიელი ოპერატორი
    label20.Text = SumFib.ToString();
}

```

6.7 ნახაზზე ჩანს დილაკი „შემთხვევით_რიცხვთა გენერატორი“, რომელითაც იხსნება Form3 ფანჯარა და შესაბამისი ინტერფეისით.

ამოცანა_6.5: საჭიროა პროგრამის კოდის აგება, რომელიც შემთხვევით რიცხვთა გენერატორის საშუალებით (Random-კლასის ობიექტით) ციკლურად ამოიღებს რიცხვებს მითითებულ დიაპაზონში (Next(min,max-1)) და მოათავსებს label3-ში. უნდა განისაზღვროს ამ რიცხვთა ჯამი label4-ში. 6.8 ნახაზზე ნაჩვენებია ასაგები Form3-ის დიზაინი.



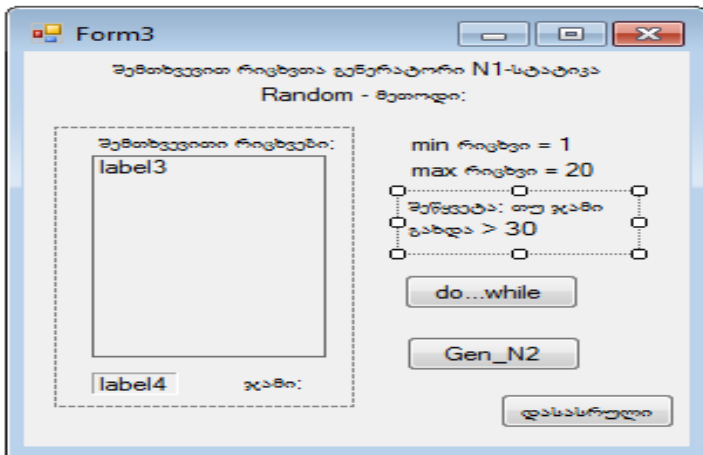
ნახ.6.8

label3-ის თვისებების შეცვლით AutoEllipsis=True და AutoSize=False შესაძლებელია უჯრის გაფართოება და მასზე რამდენიმე სტრიქონის ან ტექსტის გამოტანა. label4 გავხადეთ 3-განზომილებიანი: BorderStyle=Fix3D.

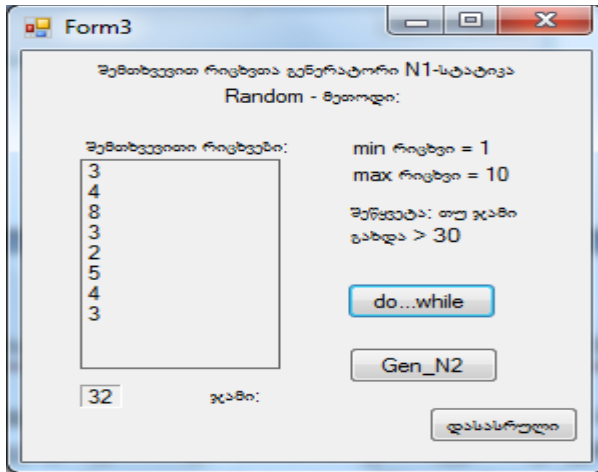
// ლისტინგი_6.8 --do...while ციკლი შემთხვევით რიცხვთა
// გენერატორისთვის (სტატისკური კოდი)--

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormCycle
{
    public partial class Form3 : Form
    {
        Random r = new Random(); //Random-კლასი -შრ-გენერატორი
        public Form3() { InitializeComponent(); }
    }
}
```

```
private void button1_Click(object sender, EventArgs e)
{ // გენერატორის ამოქმედება
  int summe = 0, z;
  label3.Text = "";
  do
  {
    z=r.Next(1, 10); // Next მეთოდი Random-კლასის
    summe += z;
    label3.Text += z.ToString() + "\n";
  }
  while (summe < 30)
  ;
  label4.Text = summe.ToString();
}
private void button2_Click(object sender, EventArgs e)
{ Close(); }
}
}
```



ნახ.6.9-



ნახ.6.9-ბ: შედეგი

6.9 ნახაზზე ნაჩვენებია შედეგები. “do...while” ლილაკის ყოველ ახალ ამოქმედებაზე მუშაობას იწყებს შემთხვევით რიცხვთა გენერატორი და label3-ში გამოაქვს რიცხვები 1-10 დიაპაზონიდან, როგორც ეს Next(1,10) არის მითითებული. while(summe<30) ოპერატორი ამოწმებს ამ რიცხვების ჯამი ხომ არაა მეტი 30-ზე. თუ არაა მეტი, მაშინ სრულდება „ ; “ - ცარიელი ოპერატორი, რომელიც აბრუნებს პროცესს do { } - ციკლში. თუ ჯამი მეტია 30-ზე, მაშინ ციკლი მთავრდება და label4-ში იწერება semme-შედეგი.

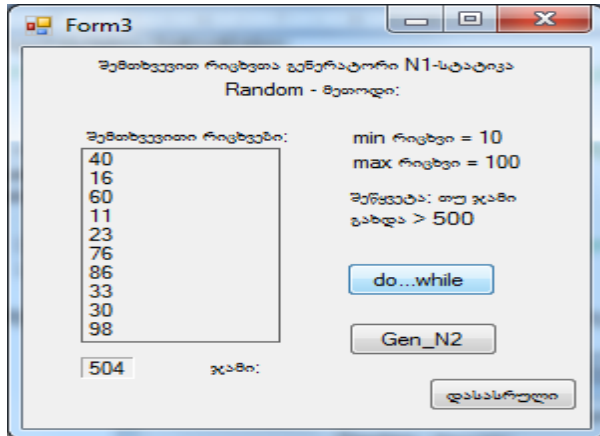
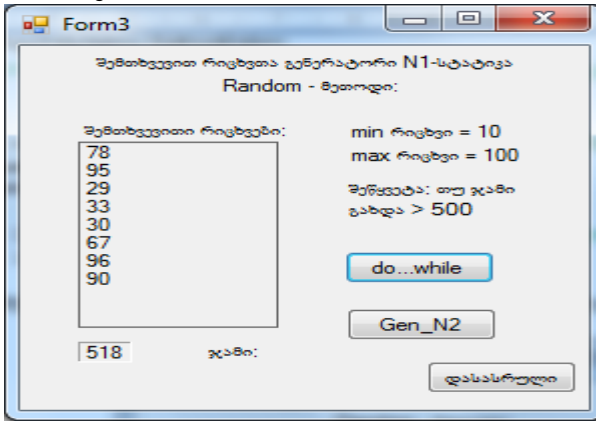
შემთხვევით რიცხვთა დიაპაზონის შესაცვლელად პროგრამაში შეცვალეთ Next, როგორც ეს 6.9_ლისტინგის ფრაგმენტშია ნაჩვენები:

```
// ლისტინგი_6.9---შრ-გენერატორისთვის min და max
// საზღვრების შეცვლა ----
do
{
z = r.Next(10, 100); // მინ-მაქს რიცხვები შეიცვალა
summe += z;
```



```
label3.Text += z.ToString() + "\n";  
}  
while (summe < 500) // საკონტროლო ჯამი შეიცვალა  
    ;
```

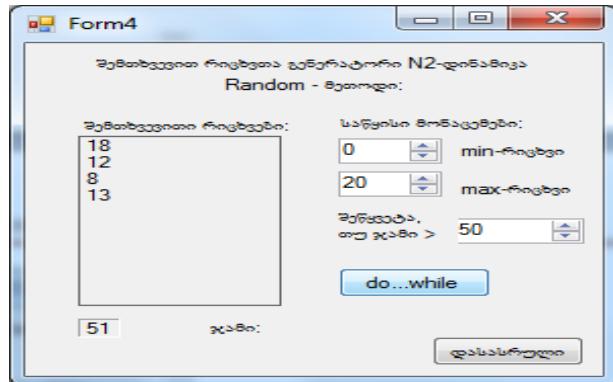
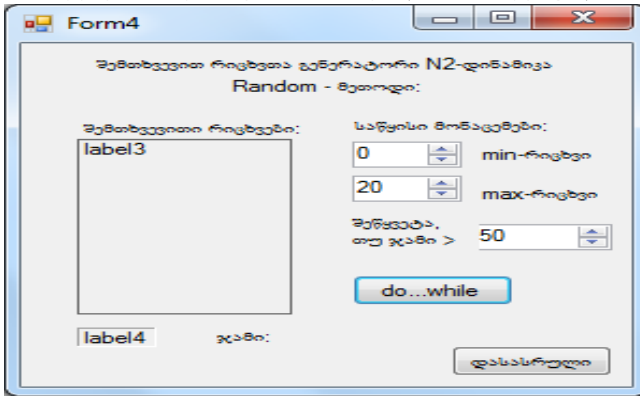
ახალი ექსპერიმენტისთვის შედეგები გამოტანილია 6.10 ნახაზზე.



ნახ.6.10

საწყისი მონაცემების ასეთი ცვლილება, რომელიც ზემოთ ჩავატარეთ, სტატიკურია ანუ ხისტია და მოითხოვს პროგრამის კოდის შესაბამისი ცვლადების, აგრეთვე Properties-ში ფორმის ზოგიერთი მონაცემის ხელით შეცვლას, რაც მთლიანობაში არასასურველია, განსაკუთრებით პროგრამის მომხმარებლისთვის, რომელსაც შეიძლება ჰქონდეს დახურული კოდი.

პროგრამის დინამიური ვარიანტი (ნახ.6.10 ღილაკი Gen_N2) მისცემს მომხმარებელს უფლებას თვითონ განსაზღვროს საწყისი მონაცემები. ასეთი ფორმის მაგალითი მოცემულია 6.11 ნახაზზე.



ნახ.6.11

გამოყენებულია სამი numericUpDown ვიზუალური ელემენტი: 1-მინიმალური რიცხვის, 2-მაქსიმალური რიცხვის და 3-ჯამის ანგარიშის ციკლის შეწყვეტისათვის. პროგრამულ კოდში შეტანილ იქნება ცვლილებები ერთხელ და შემდგომში ის აღარ მოითხოვს ცვლილებებს. პროგრამა იმუშავებს დინამიკურად საწყისი მონაცემებისთვის. კოდის ტექსტი მოცემულია 6.10_ლისტინგში.

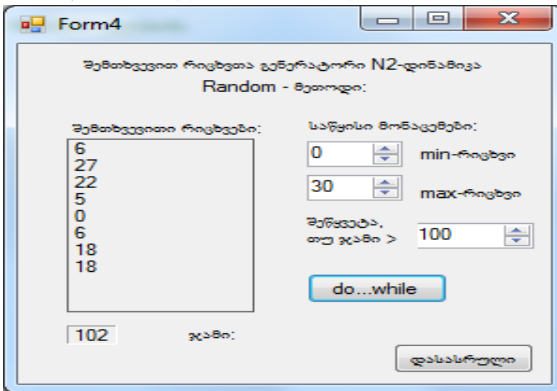
```
// ლისტინგი_6.10 --- do...while ციკლი შემთხ-რიცხვთა
// გენერატორისთვის (დინამიკური კოდი)--
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormCycle
{
    public partial class Form4 : Form
    {
        Random r = new Random(); // Random კლასი - შრგ
        public Form4() { InitializeComponent(); }
        private void button1_Click(object sender, EventArgs e)
        { // ტიპის გარდაქმნა cast(int) -ით
            int minNum = (int)numericUpDown1.Value;
            int maxNum = (int)numericUpDown2.Value;
            int sumControlNum = (int)numericUpDown3.Value;
            int summe = 0, z;
            label3.Text = "";
            do
            { // Next არის მეთოდი Random-კლასის
                z=r.Next(minNum, maxNum);
                summe += z;
                label3.Text += z.ToString() + "\n";
            }
        }
    }
}
```

```

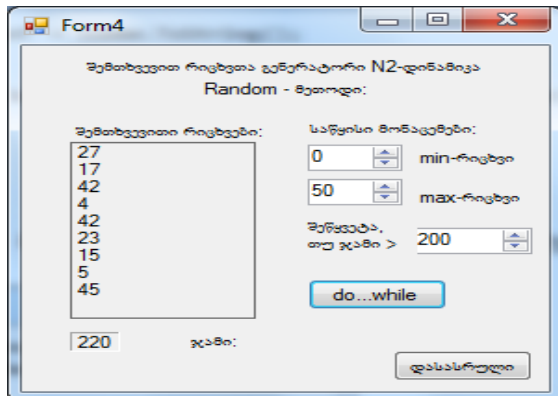
}
    while (summe < sumControlNum)
        ;
    label4.Text = summe.ToString();
}
private void button2_Click(object sender, EventArgs e)
{ Close(); }
}
}

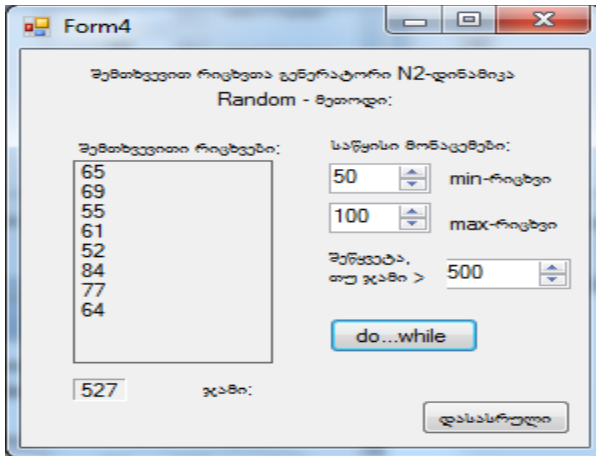
```

6.12 ნახაზზე მოცემულია პროგრამის მუშაობის შედეგები სხვადასხვა საწყისი მონაცემებისთვის:

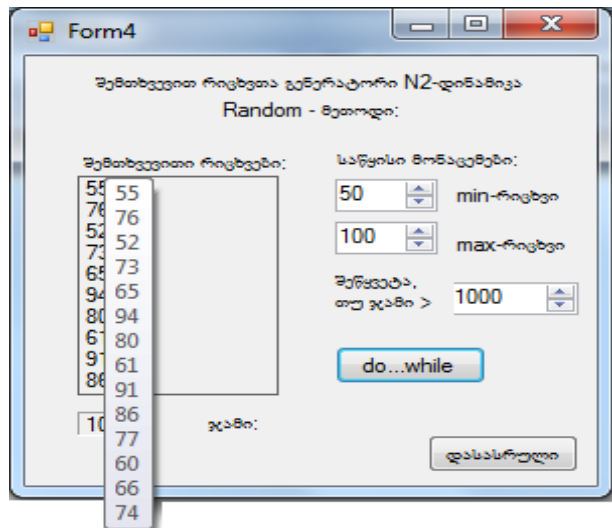


ნახ.6.12-ა





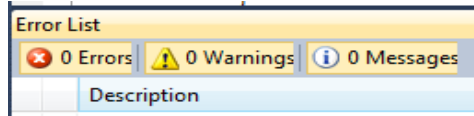
ნახ.6.12-ბ



მეოთხე მაგალითზე შემთხვევით რიცხვთა გენერატორის რიცხვები ვერ ეტევა label-ის ჩარჩოში. ასეთ დროს მაუსის კურსორის მიტანით label-ზე გამოჩნდება ყველა რიცხვის მთლიანი სვეტი.

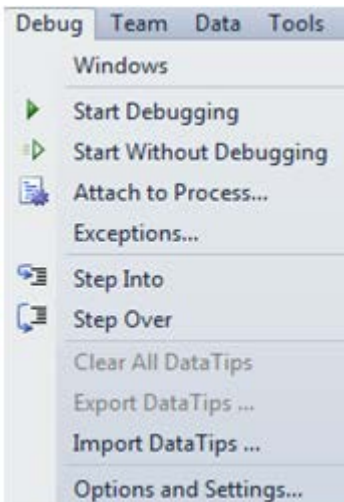
თავი 7 პროგრამული პროექტის გამართვის ვიზუალური საშუალებები

როგორც ცნობილია, პროგრამის გამართვის (debugging) და ტესტირების (შესრულების) პროცესში ადგილი აქვს პროგრამული შეცდომების გამოვლენას. ეს შეცდომები სამი სახისაა: სინტაქსური, პროცედურული და ლოგიკური. სინტაქსური შეცდომების აღმოჩენა ხდება C#-ენის კომპილატორის საშუალებით და გამოიტანება პროგრამული ტექსტების რედაქტორის ქვედა ნაწილში, Error List ფანჯარაში (ნახ.7.1). მათი პოვნა და შესწორება შედარებით ადვილია.



ნახ.7.1

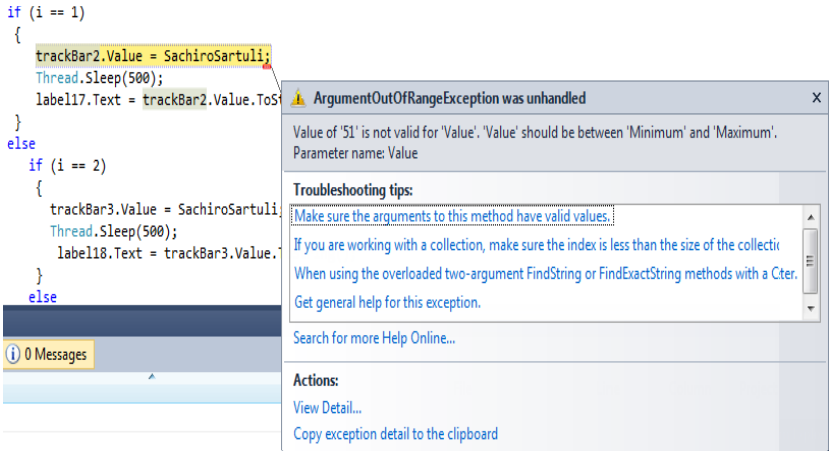
პროცედურული შეცდომები ვლინდება პროგრამის შესრულების პროცესში. როცა პროგრამაში აღარაა სინტაქსური შეცდომები და ხდება მისი ამუშავება: Start Debugging ან F5 (ნახ.7.2).



ნახ.7.2

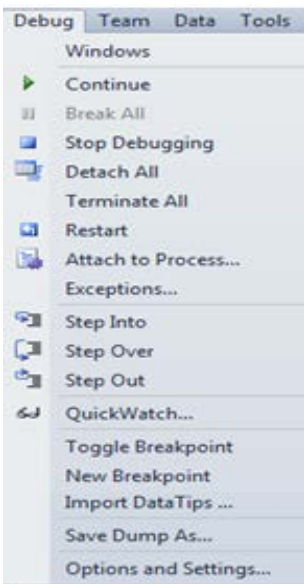
შესაძლებელია ისეთი შეცდომის გამოვლენა, რომელიც წყვეტს პროგრამის შესრულების პროცესს (ანუ სრულდება ავარიულად).

დებაგერს გამოაქვს ამ დროს გარკვეული შეტყობინება (ნახ.7.3), რომელიც მოითხოვს პროგრამის მხრიდან ანალიზს და შეცდომის გამორიცხვას (Exception Handling).



ნახ.7.3

ლოგიკური შეცდომების აღმოჩენა შედარებით რთულია. პროგრამა ამ დროს მუშაობს და სრულდება ნორმალურად (არაავარიულად), მაგრამ შედეგები „საეჭვოა“.



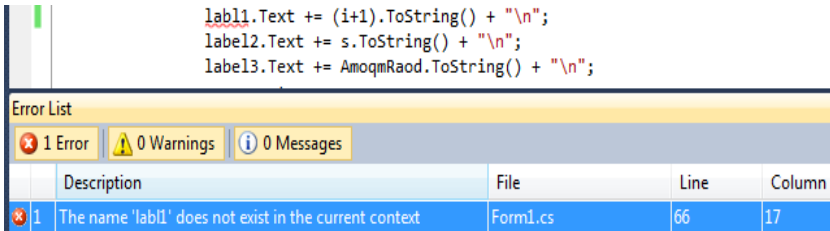
ტესტირების პროცესში, რომელიც აუცილებლად მოსდევს პროგრამის გამართვას, საჭიროა ასეთი „კვლევის“ ჩატარება, რათა გამოვლენილ იქნას მოსალოდნელი, ფარული ლოგიკური შეცდომები.

C# ენის რედაქტორს აქვს კარგი ინსტრუმენტული საშუალებები (Debugger) ამ ტიპის შეცდომების მოსაძებნად და აღმოსაფხვრელად (ნახ.7.4).

ნახ.7.4

7.1. სინტაქსური შეცდომების აღმოფხვრის საშუალებანი

თუ პროგრამის არაკორექტულ კოდში შეცდომითაა ჩაწერილი ენის ოპერატორი (მაგალითად, “Whail” ნაცვლად while - ისა) ან კონსტრუქცია (მაგალითად, “case: “, რომელსაც არ უძღვის წინ switch() {...}) და ა.შ. როგორც ზემოთ აღვნიშნეთ, ამ დროს კომპილატორი მიუთითებს არსებულ შეცდომას და მის ადგილმდებარეობას (ნახ.7.5).



ნახ.7.5

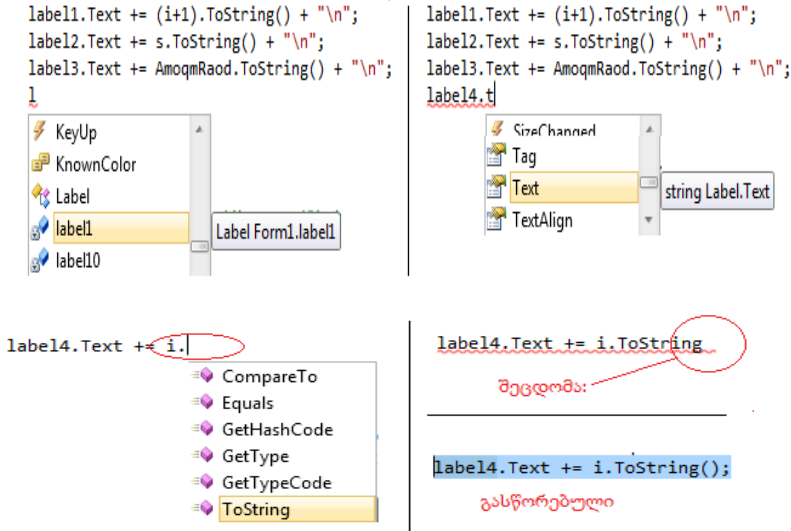
სინტაქსური შეცდომები თუ არ გასწორდა, მაშინ ვერ მოხერხდება პროგრამის ობიექტური (.obj) და შესრულებადი (.exe) კოდების ფორმირება.

სინტაქსური შეცდომების თავიდან ასაცილებლად C# ენის რედაქტორს აქვს სხვადასხვა ვიზუალური დამხმარე საშუალებები. მაგალითად, პროგრამაში ოპერატორების ტექსტის შეტანისას, ან ობიექტის მეთოდის და მოვლენის არჩევისას (წერტილის „.“ დასმისას) ხდება ვიზუალური ბლოკის (Intellisense - ავტოდამატება) შემოთავაზება (ნახ.7.6), საიდანაც ამოირჩევა საჭირო სიტყვა და Enter-კლავიშით სწრაფად ჩაჯდება მითითებულ ადგილას.

ეს გამორიცხავს როგორც ოპერატორის (ობიექტის თვისების, მეთოდის და ა.შ.) არასწორ სინტაქსურ ჩაწერას, ასევე არარელევანტური სიტყვის მითითებას (სიტყვა, რომელიც აქ „უადგილოა“).

შესაძლებლია აგრეთვე კოდში „გახსნილ-დასახურ“ ფრჩხილების რაოდენობის კონტროლი, რაც ძალზე ხშირი

შეცდომების წყაროა. მთლიანად, შეიძლება ითქვას, რომ ენის ასეთი ვიზუალური კონტროლის და დამხმარე საშუალებები ეფექტურს ხდის პროგრამისტის მუშაობას.

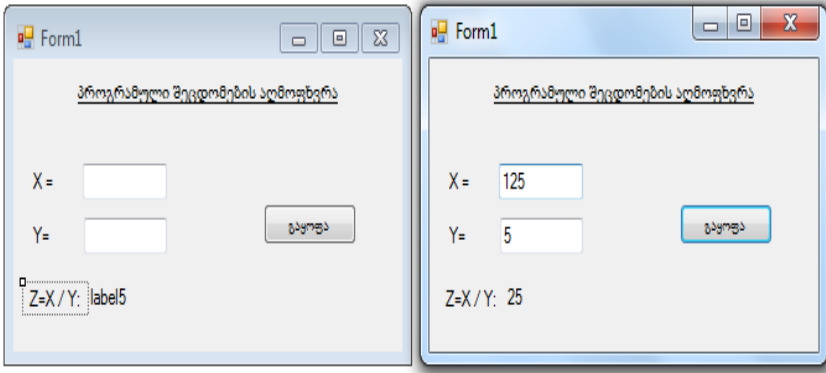


ნახ.7.6

7.2 გამოწვევის შემთხვევები: შეცდომები პროგრამის შესრულებისას და მათი აღმოფხვრა

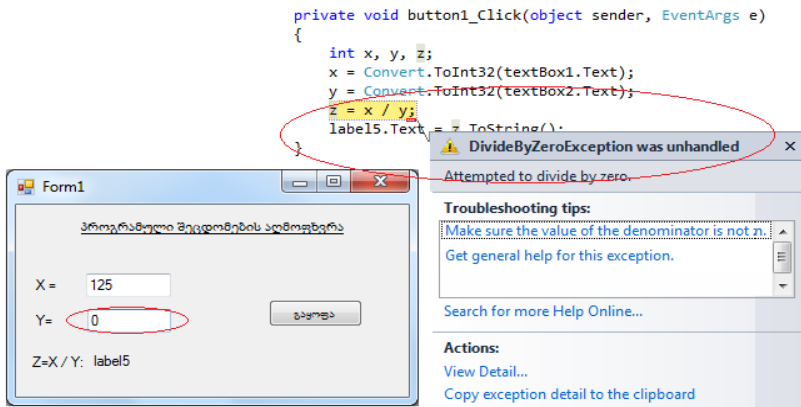
თუ პროგრამის ტექსტი სინტაქსური შეცდომებისგან თავისუფალია, ის შეიძლება ამუშავდეს შესრულებაზე. ამ დროს შესაძლებელია ისეთი შეცდომების გამოვლენა, რომლებიც პროგრამას ავარიულად დაასრულებს, ან საერთოდ არ დაასრულებს („გაჭედავს“). მაგალითად, უსასრულო ციკლი ან სხვ. განსაკუთრებული შემთხვევა. განვიხილოთ ასეთი მაგალითები:

ამოცანა_7.1: ავავთ პროგრამის კოდი, რომელიც შეასრულებს მთელი რიცხვების შეტანას და გაყოფის ოპერაციას. 7.7 ნახაზზე ნაჩვენებია ფორმა ორი ტექსტბოქსით (რიცხვების შესატანად), label5 შედეგის გამოსატანად და ღილაკი „გაყოფა“ პროგრამული კოდით.



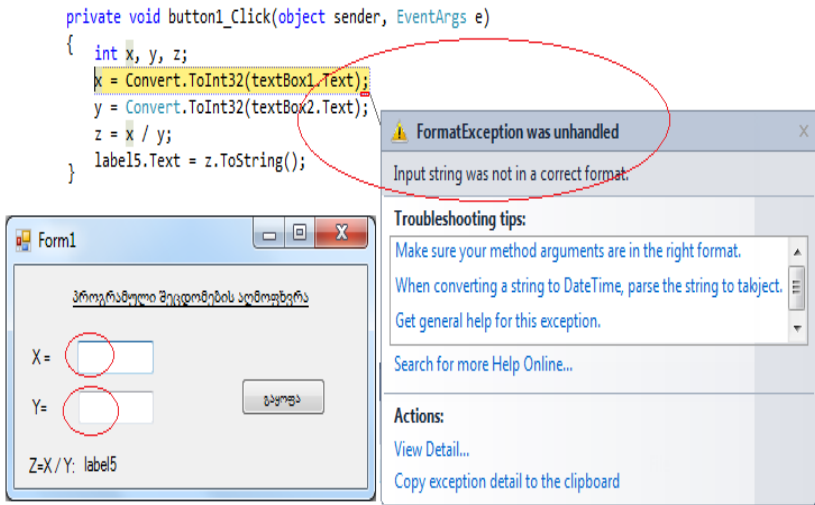
ნახ.7.7

როგორც ვხედავთ, პროგრამა მუშაობს თითქოს ნორმალურად, ასრულებს გაყოფას. ტესტირების პროცესში, რომელიც გულისხმობს კოდის ფუნქციონალობის გამოკვლევას საწყისი მონაცემების სხვადასხვა მნიშვნელობისათვის, ვღებულობთ „ნულზე გაყოფის“ შეცდომას (ნახ.7.8).



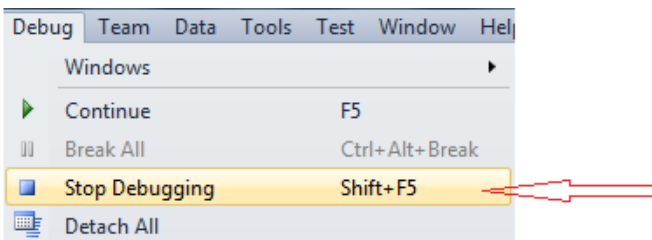
ნახ.7.8

პროგრამაში საჭიროა ამ სიტუაციის გათვალისწინება (მომხმარებელმა ყოველთვის შეიძლება შეიტანოს შემთხვევით ან „არცოდნის“ გამო 0!). ანუ თუ იქნება შეტანილი „0“, მაშინ პროგრამამ „გვერდი აუაროს“ (გამორიცხოს, აღმოფხვრას) ასეთი ტიპის შეცდომა და თან შეატყობინოს მომხმარებელს, რომ შეიტანოს კორექტული რიცხვი (0-სგან განსხვავებული).



ნახ.7.9

რედაქტირების რეჟიმში გადასასვლელად საჭიროა მენიუდან „დებაგერის შეჩერება“ (ნახ.7.10).

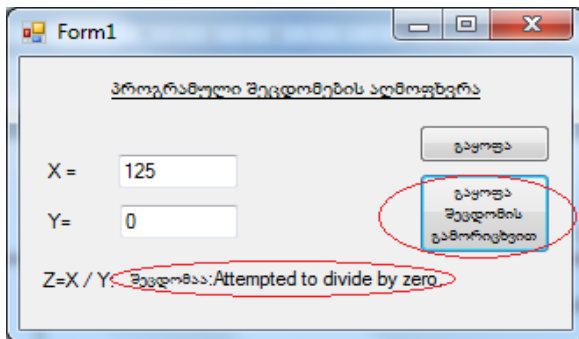


ნახ.7.10

ახლა შესაძლებელია ზემოაღწერილი ტიპის შეცდომებისათვის გამორიცხვის პროცედურის კოდის ფორმირება. მაგალითად, 7_1 ლისტინგზე მოცემულია ასეთი კოდი.

```
//ლისტინგი_7.1 --- Exception -----
private void button2_Click(object sender, EventArgs e)
{
    int x, y, z;
    try
    {
        x = Convert.ToInt32(textBox1.Text);
        y = Convert.ToInt32(textBox2.Text);
        z = x / y;
        label5.Text = z.ToString();
    }
    catch (Exception nul_Div) // ობიექტი nul_Div
    {
        label5.Text = "შეცდომა:" + nul_Div.Message;
    }
}
```

7.11 ნახაზზე ნაჩვენებია ამ კოდის მუშაობის შედეგი, რომელიც მოთავსებულია ღილაკზე „გაყოფა“ შეცდომის გამორიცხვით“.

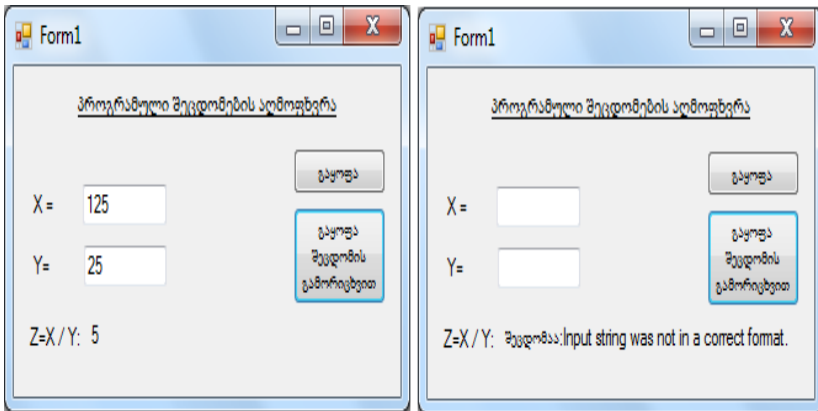


ნახ.7.11

საკურადღებო: გამოყენებულია კონსტრუქცია try { }... catch { }. საკვანძო სიტყვა try ინიცირებას უკეთბს გამორიცხვის მექანიზმს. ამ წერტილიდან პროგრამა იწყებს {..} ბლოკის შესრულებას. ამ ბლოკის ოპერატორების შესრულებისას თუ გაჩნდა გამორიცხვის (Exception) შემთხვევა, მაშინ მას „დაიჭერს“ catch და შეცვლის გამორიცხვის სიტუაციას თავის {...} ბლოკით.

ჩვენ შემთხვევაში catch ბლოკში მოთავსებულია Exception კლასის ობიექტი (nul_Div), რომელიც შეიცავს ინფორმაციას აღმოცენებული შეცდომის შესახებ. Message თვისებით ხდება შეტყობინების გამოტანა ეკრანზე. პროგრამა ბოლომდე სრულდება არა-ავარიულად.

7.12 ნახაზზე მოცემულია try...catch - გამორიცხვის მექანიზმით შესრულებული პროგრამული კოდის შედეგები: როცა რიცხვები შეტანილია ნორმალურად გაყოფის ოპერაციისთვის (ამ დროს არ ხდება „შეცდომის“ დაფიქსირება try-ში), და მეორე, როცა არასწორადაა შეტანილი საწყისისი მონაცემები (აქ იმუშავებს შეცდომების გამორიცხვის ბლოკი).



ნახ.7.12

განსაკუთრებულ შემთხვევათა დამუშავების try...catch მექანიზმი შეიძლება გაფართოვდეს ბიბლიოთეკაში არსებული Exception-კლასის საფუძველზე. აქ იგულისხმება სპეციფიური შეცდომების აღმოჩენის შესაძლებლობა, მაგალითად, ტიპების გარდაქმნისას, ნულზე გაყოფისას და ა.შ. თუ შეცდომის სახე წინასწარ არაა განსაზღვრული, მაშინ გამოიყენება ზოგადი Exception კლასის ობიექტი.

7.2_ლისტინგში მოცემულია ღოლაკის „შეცდომის გამორიცხვა“ შესაბამისი კოდის ფრაგმენტი.

// ლისტინგი_7.2 ---- - სპეციფიური და ზოგადი შეცდომები ---

```
private void button3_Click(object sender, EventArgs e)
{
    int x, y, z;

    try
    {
        x = Convert.ToInt32(textBox1.Text);
        y = Convert.ToInt32(textBox2.Text);
        z = x / y;
        label5.Text = z.ToString();
    }

    catch(FormatException nul_Div) //ტიპის გარდაქმნის შეცდომა
    {
        label5.Text = "შეცდომაა შეტანის ფორმატში\n" +
            nul_Div.Message;
    }

    catch(DivideByZeroException nul_Div) // 0-ზე გაყოფის შეცდ.
    {
        label5.Text = "0-ზე გაყოფის შეცდომაა\n" +
            nul_Div.Message; ;
    }
}
```

```

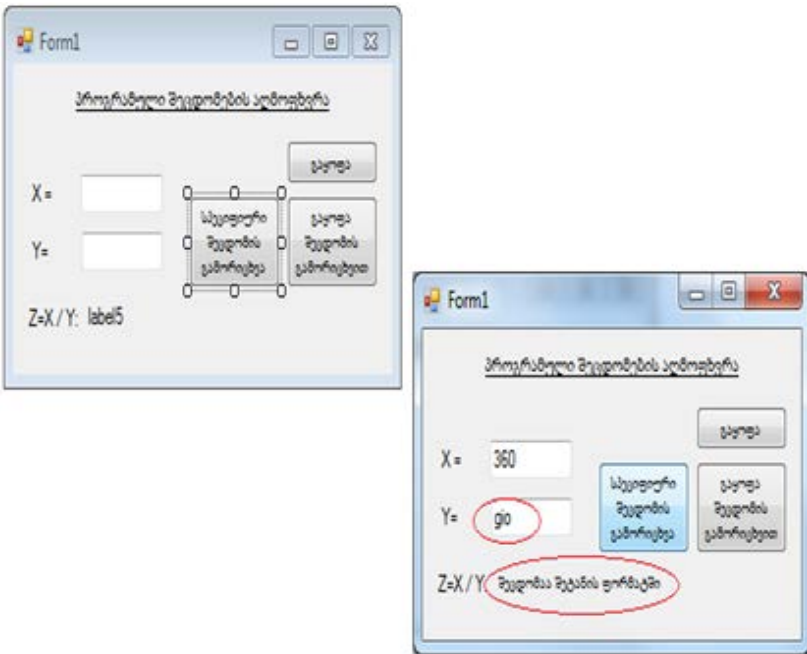
}

catch (Exception nul_Div) // ზოგადი შეცდომა
{
    label5.Text = "ზოგადი, არასპეციფიური შეცდომა\n"+
        nul_Div.Message;
}
}

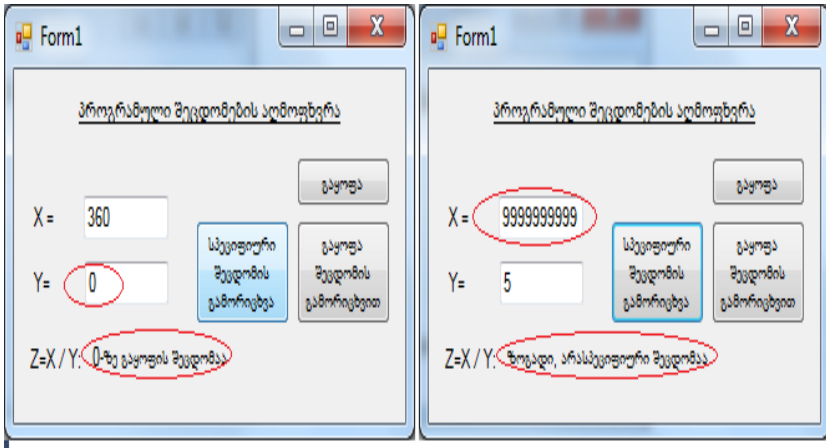
```

ლისტინგში catch{...} ბლოკების მიმდევრობას აქვს მნიშვნელობა (თუ სრულდება პირველი, წყდება პროცესი. თუ არა, გადადის შემდეგზე).

შედეგები ასახულია 7.13-ა,ბ ნახაზებზე.



ნახ.7.13-ა



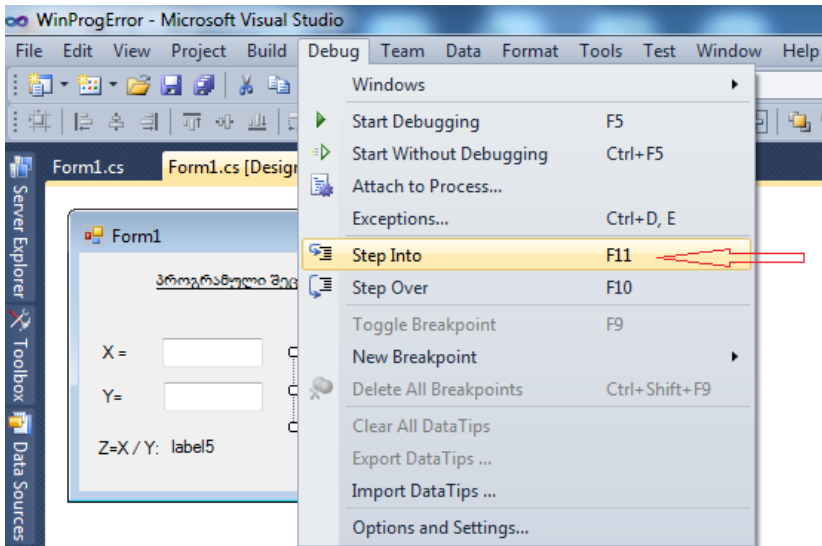
ნახ.7.13-ბ

7.3. ლოგიკური შეცდომები და პროგრამის გამართვა ბიჯური შესრულების რეჟიმში

როგორც აღვნიშნეთ, ლოგიკური შეცდომები მაშინ აღმოჩნდება, როდესაც სინტაქსური და შესრულების პროცესის შეცდომები აღარაა, მაგრამ სასურველ (დაგეგმილ სავარაუდო) შედეგს პროგრამა არ გვაძლევს.

ეს ნიშნავს, რომ პროგრამის აგების ლოგიკა არასწორია !

ასეთი შეცდომების აღმოჩენა საკმაოდ რთულია და მოითხოვს ტესტირების პროცესის და პროგრამის შესრულების მიმდევრობის შედეგების ანალიზს. ვიზუალური C# ენა ფლობს პროგრამის გამართვის (Debugging) კარგ დამხმარე საშუალებებს. განვიხილოთ ისინი ჩვენი WinProgError პროექტის მაგალითზე (ნახ.7.14). გავხსნათ პროექტი, მოვაშადალოთ Form1 ფორმა და მენიუს Debug-ში ავირჩიოთ Step Into (ან F11 ღილაკი კლავიატურის ზედა რიგში).



ნახ.7.14

ჩაირთვება პროგრამის გამართვის ბიჯური (Step Into) რეჟიმი. ეკრანზე დიზაინის ფორმა შეიცვლება (იხ. Solution Explorer) Program.cs დამწყები პროგრამის ტექსტით, რომელშიც მოთავსებულია Main() მთავარი ფუნქცია (ნახ.7.15). მარცხნივ ჩანს ყვითელი ისარი, რომელიც F11-ით ბიჯურად გადაადგილდება იმ სტრიქონზე, რომელიც სრულდება მოცემულ მომენტში.

```

// Program.cs - Main() - დაწყები პროგრამის ტექსტი ----
using System;
using System.Windows.Forms;

namespace WinProgError
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
    
```

ნახ.7.15

Application.Run (new Form1()); სტრიქონის ამუშავებით ისარი გადადის (იხ. Solution Explorer) Form1.Designer.cs პროგრამაში (ნახ.7.16). შემდეგ InitializeComponent()-ში გაივლის ფორმაზე დალაგებულ ყველა ელემენტს.

Form1.Designer.cs პროგრამის ბიჯურად გავლის შემდეგ Main()-იდან ამუშავდება Run და ეკრანზე გამოვა Form1 (ნახ.7.17), სადაც უნდა შევიტანოთ X და Y მნიშვნელობები და ავამოქმედოთ დილაკი „სპეციფიური შეცდომის გამორიცხვა“ (ნახ.7.18).

ბიჯის ისარი გადადის Form1.cs პროგრამის ტექსტზე (ნახ.7.18), სადაც button3_Click მოვლენის შესაბამის try {...} ბლოკში შედის.

```

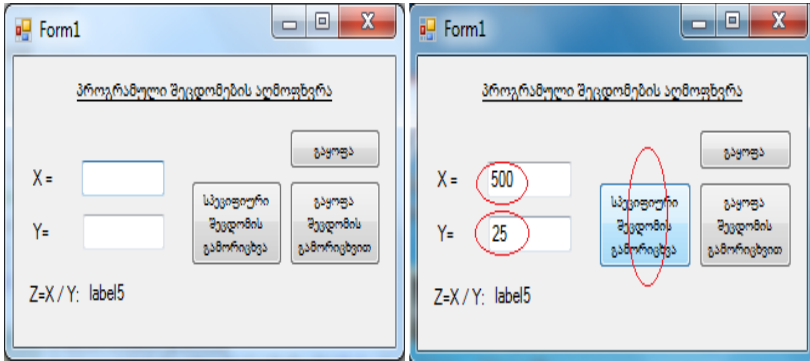
Form1.Designer.cs x Program.cs Form1.cs Form1.cs [Design]
WinProgError.Form1 components
namespace WinProgError
{
    partial class Form1
    {
        /// <summary> ...
        private System.ComponentModel.IContainer components = null;

        /// <summary> ...
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

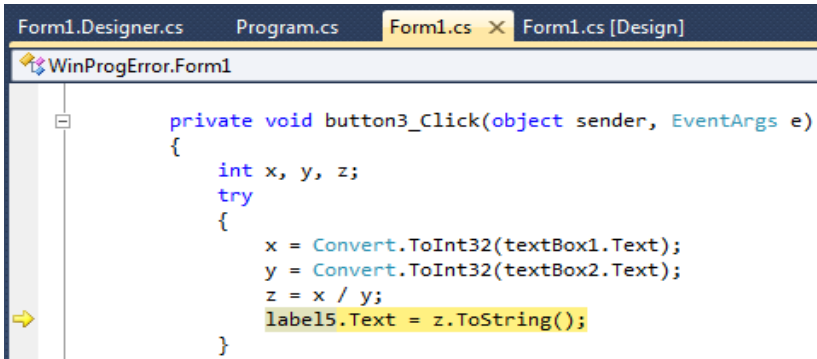
        #region Windows Form Designer generated code

        /// <summary> ...
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.textBox2 = new System.Windows.Forms.TextBox();
            this.label2 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();
            this.label4 = new System.Windows.Forms.Label();
        }
    }
}
    
```

სახ.7.16

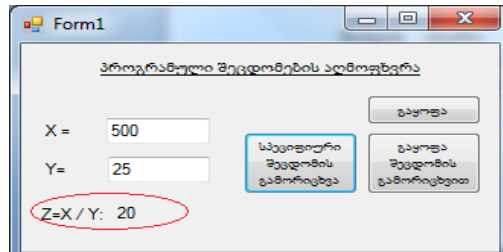


ნახ.7.17



ნახ.7.18

ვინაიდან X და Y-ის შეტანილი მნიშვნელო-ბები დასაშვებია, შედეგში აისახება label5.Text=z.ToString() მნიშვნელობა, ანუ 20 (ნახ.7.19).



ნახ.7.19

თუ ახლა განვიხილავთ $Y=0$ შემთხვევას, და ავამოქმედებთ იგივე ბუტონს, მაშინ try ბლოკში მომზადდება განსაკუთრებული შემთხვევა, როცა გამოვლი 0-ია.

```
private void button3_Click(object sender, EventArgs e)
{
    int x, y, z;
    try
    {
        x = Convert.ToInt32(textBox1.Text);
        y = Convert.ToInt32(textBox2.Text);
        z = x / y;
        label5.Text = z.ToString();
    }
}
```

ნახ.7.20

მოქმედება გადაეცემა catch ბლოკს:

```
catch (DivideByZeroException nul_Div) // 0-ზე გაყოფის შეცდომა
{
    label5.Text = "0-ზე გაყოფის შეცდომაა\n" + nul_Div.Message;
}
```

ნახ.7.21

დავუშვათ, რომ X ან Y არარიცხვითი სიმბოლო ან სტრიქონია, მაგალითად, “G, Gia,...” (ნახ.7.22).

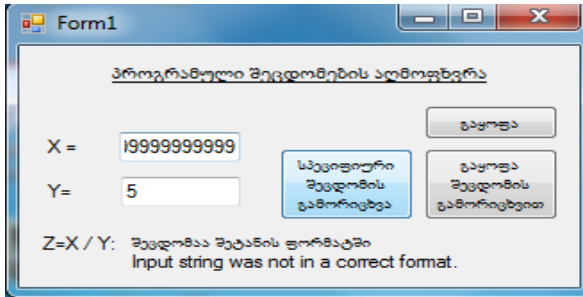
ნახ.7.22

F11-ით გადაადგილების შემდეგ პროგრამის ტექსტის აქტიური ფრაგმენტი იქნება შემდეგი (ნახ.7.23).

```
private void button3_Click(object sender, EventArgs e)
{
    int x, y, z;
    try
    {
        x = Convert.ToInt32(textBox1.Text);
        y = Convert.ToInt32(textBox2.Text);
        z = x / y;
        label5.Text = z.ToString();
    }
    catch (FormatException nul_Div) // ტიპის გარდაქმნის შეცდომა
    {
        label5.Text = "შეცდომა შეტანის ფორმატში" + nul_Div.Message;
    }
}
```

ნახ.7.23

მესამე, ზოგადი ანუ არასპეციფიური შემთხვევაა, ამ დროს სისიტემა თვითონ აღმოაჩენს, თუ რა სახის შეცდომასთან გვაქვს საქმე. მაგალითად, თუ X ან Y - ში შევიტანთ „დიდ რიცხვს“ (ნახ.7.24), მაშინ კოდის ფრაგმენტი ასე გამოიყურება (ნახ.7.25).



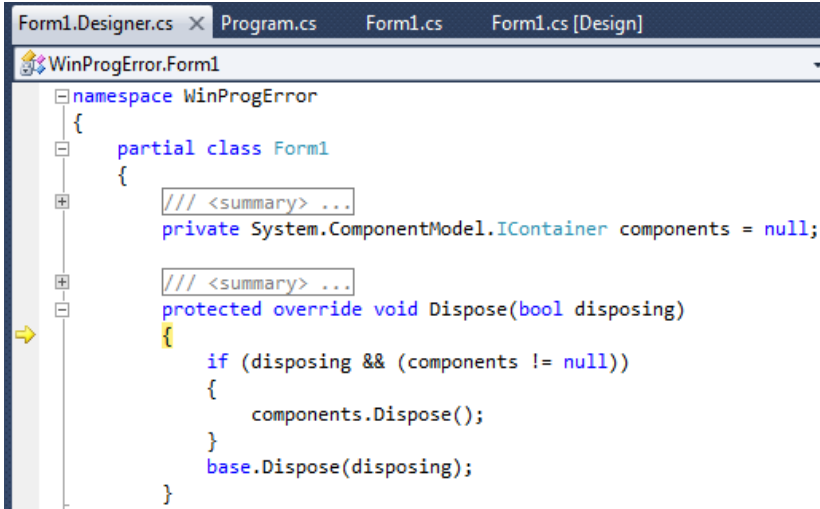
ნახ.7.24

```
catch (Exception nul_Div) // ზოგადი შეცდომა
{
    label5.Text = "ზოგადი, არასპეციფიური შეცდომა" + nul_Div.Message;
}
```

ნახ.7.25

პროგრამასთან მუშაობის დასამთავრებლად დავხუროთ Form1. ამ დროს მართვა (ისარი) გადაეცემა Form1.Designers.cs

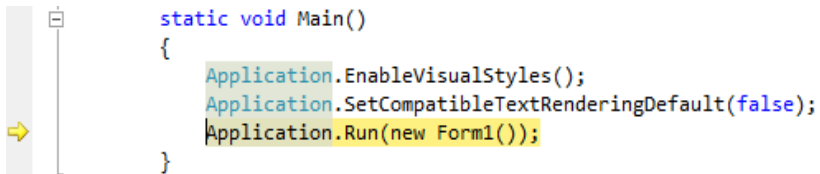
(ნახ.7.26) და ბოლოს პროგრამას Form1.cs, რომელშიც არის Main(), და რომლითაც დაიწყო თავიდან ამ პროგრამის მუშაობა (ნახ.7.27).



```
Form1.Designer.cs x Program.cs Form1.cs Form1.cs [Design]
WinProgError.Form1
namespace WinProgError
{
    partial class Form1
    {
        /// <summary> ...
        private System.ComponentModel.IContainer components = null;

        /// <summary> ...
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

ნახ.7.26



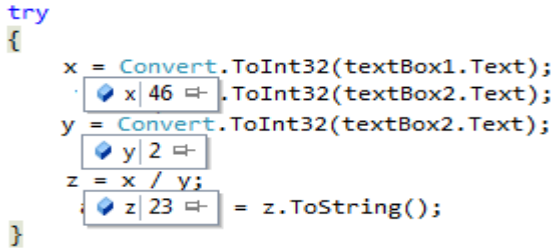
```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());
}
```

ნახ.7.27

ამით დასრულდება დებაგერის მუშაობის პროცესი.

პროგრამის ანალიზის პროცესში შესაძლებელია ცვლადების მნიშვნელობათა ვიზუალური შემოწმება. ამისათვის მაუსის კურსორი უნდა მივიტანოთ ცვლადთან. 7.28 ნახაზზე ნაჩვენებია პროგრამაში X, Y და Z -ის მნიშვნელობები.

```
try
{
    x = Convert.ToInt32(textBox1.Text);
    y = Convert.ToInt32(textBox2.Text);
    z = x / y;
    z = z.ToString();
}
```

A screenshot of a code editor showing a C# code block. The code defines variables x, y, and z. x is assigned the value of textBox1.Text, y is assigned the value of textBox2.Text, and z is assigned the value of x divided by y, then converted to a string. The code is enclosed in a try block. Below the code, there are three small windows showing the values of the variables: x is 46, y is 2, and z is 23.

ნახ.7.28

დიდი პროგრამების ანალიზის დროს ბიჯურ რეჟიმში მუშაობა არაეფექტურია, სჭირდება ხანგრძლივი დრო. უფრო მოსახერხებელია ვიზუალური კონტროლის ორგანიზების მეორე ხერხი, რომელიც წყვეტის წერტილების კონცეფციითაა ცნობილი.

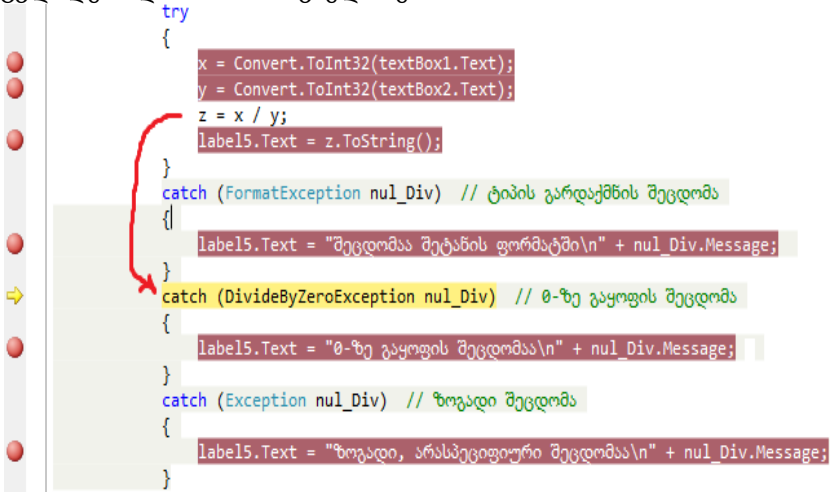
წყვეტის წერტილი არის პროგრამის კოდის ის ადგილი (სტრიქონი), სადაც წყდება პროგრამის შესრულება. ამ სტრიქონში მოთავსებული გამოსახულება (ოპერატორი ან მეთოდი) არ შესრულდება და მართვა მომხმარებელს გადაეცემა.

წყვეტის წერტილის შესაქმნელად კოდის საჭირო სტრიქონის გასწვრივ მარცხენა ველში მოვათავსოთ კურსორი და დავაჭიროთ თავვის მარცხენა კლავიშს (ან F9 კლავიშს). გამოჩნდება შინდისფერი წრე. პროგრამის კოდში შეგვიძლია შევქმნათ წყვეტის რამდენიმე წერტილი (ნახ.7.29).

მენიუდან Debug→Windows→Autos არჩევით რედაქტორის ქვედა ნაწილში გამოიტანება ცვლადების მონიტორინგის ფანჯარა,

რომელშიც ერთდროულად ჩანს რამდენიმე ცვლადი მათი აქტუალური მნიშვნელობებით (ნახ.7.30).

მენიუდან Debug→Windows→Locals პუნქტის არჩევით მონიტორინგის ფანჯარაში გამოიტანება მხოლოდ ლოკალური ცვლადები და მათი მნიშვნელობები.



```
try
{
    x = Convert.ToInt32(textBox1.Text);
    y = Convert.ToInt32(textBox2.Text);
    z = x / y;
    label5.Text = z.ToString();
}
catch (FormatException nul_Div) // ტიპის გარდაქმნის შეცდომა
{
    label5.Text = "შეცდომა შეტანის ფორმატში" + nul_Div.Message;
}
catch (DivideByZeroException nul_Div) // 0-ზე გაყოფის შეცდომა
{
    label5.Text = "0-ზე გაყოფის შეცდომა" + nul_Div.Message;
}
catch (Exception nul_Div) // ზოგადი შეცდომა
{
    label5.Text = "ზოგადი, არასპეციფიური შეცდომა" + nul_Div.Message;
}
```

ნახ.7.29

```

private void button3_Click(object sender, EventArgs e)
{
    int x, y, z;
    try
    {
        x = Convert.ToInt32(textBox1.Text);
        y = Convert.ToInt32(textBox2.Text);
        z = x / y;
        label5.Text = z.ToString();
    }
    catch (FormatException nul_Div) // ტიპის გარდაქმნის შეცდომა
    {
        label5.Text = "შეცდომაა შეტანის ფორმატში\n" + nul_Div.Message;
    }
    catch (DivideByZeroException nul_Div) // 0-ზე გაყოფის შეცდომა
    {
        label5.Text = "0-ზე გაყოფის შეცდომაა\n" + nul_Div.Message;
    }
    catch (Exception nul_Div) // ზოგადი შეცდომა
    {
        label5.Text = "ზოგადი, არასპეციფიკური შეცდომაა\n" + nul_Div.Message;
    }
}

```

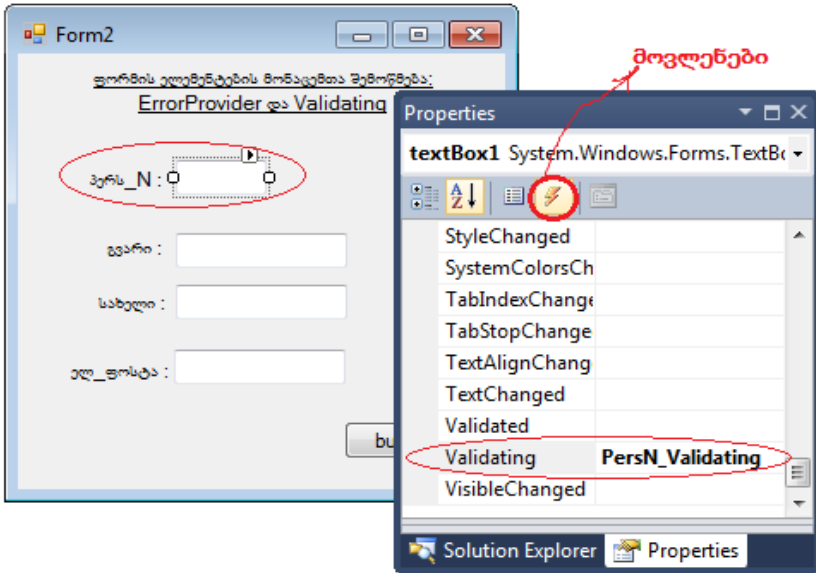
Name	Value	Type
label5	{System.Windows.Forms.Label, Text: 0-ზე გაყოფის შეცდომა}	System.Windows.Forms.Label
label5.Text	"0-ზე გაყოფის შეცდომა\nAttempted to divide by zero"	string
this	{WinProgError.Form1, Text: Form1}	WinProgError.Form1
x	400	int
y	40	int
z	10	int

ნახ.7.30

7.4. შესატან მონაცემთა კონტროლი

ვინდოუს-ფორმის ელემენტების შევსების პროცესში, როცა მომხმარებელს უხდება მათი ხელით შეტანა, შესაძლებელია შეცდომების არსებობა. მაგალითად, ამას ხშირად აქვს ადგილი ტქსტბოქსების შევსების დროს.

იმისათვის, რომ შესაძლებელი იყოს შესატან მონაცემთა კონტროლი, საჭიროა ErrorProvider ვიზუალური კომპონენტის გადმოტანა ინსტრუმენტების პანელიდან და საკონტროლო ელემენტების Properties-ში CausesValidation თვისებაში "True" მნიშვნელობის არსებობა (ნახ.7.31).



ნახ.7.31

ტექსტოქსის შევსების შემდეგ, როცა მომხმარებელი გადადის სხვა ელემენტზე, ხდება ამ ტექსტოქსში შეტანილი მნიშვნელობის შემოწმება. თუ რა ლოგიკით შემოწმდება ტექსტოქსში შეტანილი მონაცემი, დამოკიდებულია ჩვენ მიერ განსაზღვრულ მეთოდზე, რომელიც მიეხმება მოვლენის დამმუშავებელს (Event Handling).

ამგვარად ახლა საჭიროა მოვლენის დამმუშავებელის შექმნა. მაგალითად, ვდებთ „პერს_N“ ტექსტოქსზე და Properties-ში Validating თვისებას ვაძლევთ სახელს: PersN_Validating. დამმუშავებლის მეთოდის კოდი მოცემულია 9_3 ლისტინგში.

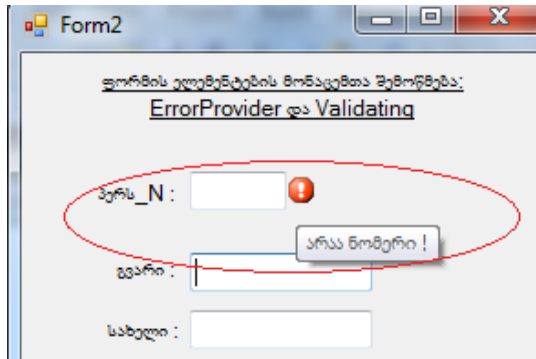
```
// ლისტინგი_7.3 --- მოვლენების დამმუშავებელი -----
private void PersN_Validating(object sender,
                                CancelEventArgs e)
{
    if (textBox1.Text.Length == 0)
```

```

{
    errorProvider1.SetError(textBox1, "არაა ნომერი!");
}
else
    errorProvider1.SetError(textBox1, "");
}

```

შედეგი მოცემულია 7.32 ნახაზზე. ველში „პერს_N“ არ ჩაწერეს მონაცემი, ისე გადავიდნენ სხვა ტექსტბოქსზე. ამ დროს მოხდა მოვლენის შემოწმება და შეცდომის აღმოჩენა, რომ ველში არაა შეტანილი მონაცემი (textBox1.Text.Length არის 0). ჩაირთვება errorProvider1.SetError და „პერს_N“-ის გვერდით გამოიტანს წითელი ფერის ძახილის ნიშანს (გაფრთხილება). მაუსის კურსორის მიტანისას ამ ველზე ჩნდება ქართული წარწერა „არაა ნომერი“ (SetError-ის მეორე პარამეტრი).

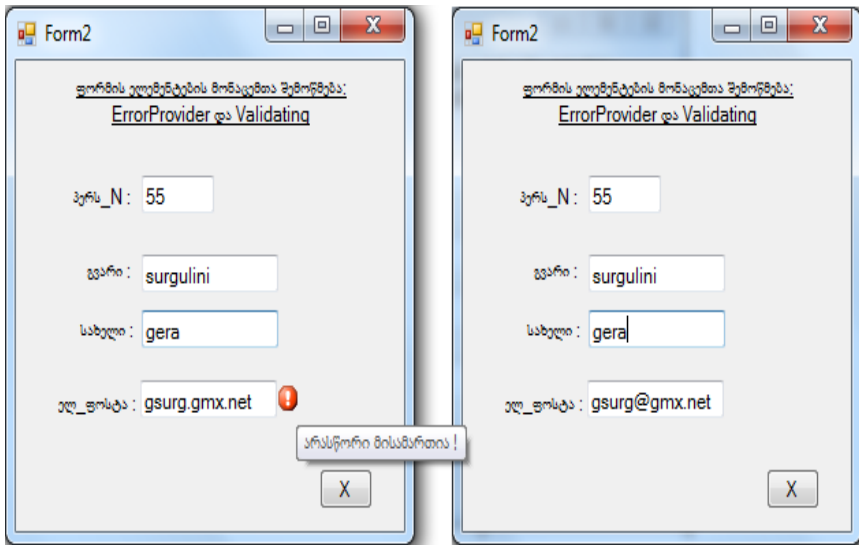


ნახ.7.32

მეოთხე ტექსტბოქსში უნდა ჩაიწეროს ელ_ფოსტის მისამართი. იგი სტრიქონული მონაცემია, რომელიც აუცილებლად უნდა შეიცავდეს '@' და '.' სიმბოლოებს. თუ რომელიმე აკლია, მაშინ შეცდომაა და უნდა ამუშავდეს მოვლენის დამმუშავებელი ამ ველისთვის (7.4_ლისტინგი).

```
// ლისტინგი_7.4 --- eMail_Validating მეთოდი -----
private void eMail_Validating(object sender,
                                CancelEventArgs e)
{
    string email = textBox4.Text;
    // კონტროლის ლოგიკა
    if(email.IndexOf('@')== -1 || email.IndexOf('.')== -1)
    {
        errorProvider1.SetError(textBox4,"არასწორი მისამართია !");
    }
    else
        errorProvider1.SetError(textBox4, "");
}
}
```

შედეგი ნახვენება 7.33 ნახაზზე.



ნახ.7.33

თავი 8 მონაცემთა ბაზებთან მუშაობის ვიზუალური საშუალებანი

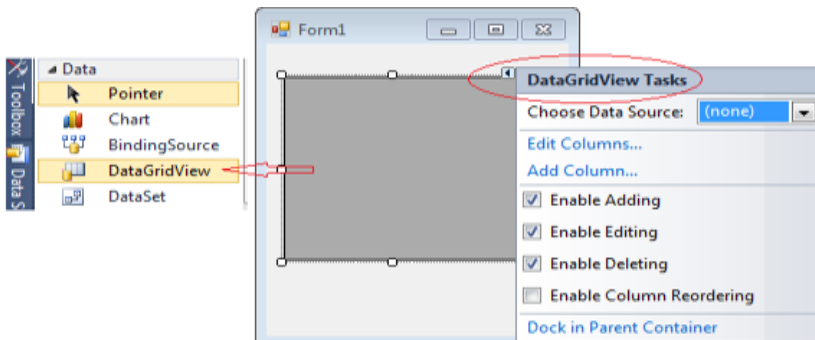
8.1. ცხრილების წარმოდგენის მართვის ელემენტი DataGridView

განიხილეთ C#.NET პროექტში მონაცემთა ბაზის გამოყენების საკითხები. რელაციური მონაცემთა ბაზის ძირითადი კომპონენტები ცხრილებია (Tables). ამიტომაც აქ ჯერ განვიხილავთ მართვის ელემენტებს DataGridView საფუძველზე.

მარტივი სიებისა და მონაცემთა ერთგანზომილებიანი ველების ასახვის მიზნით გამოიყენება ListBox- და ComboBox-ები.

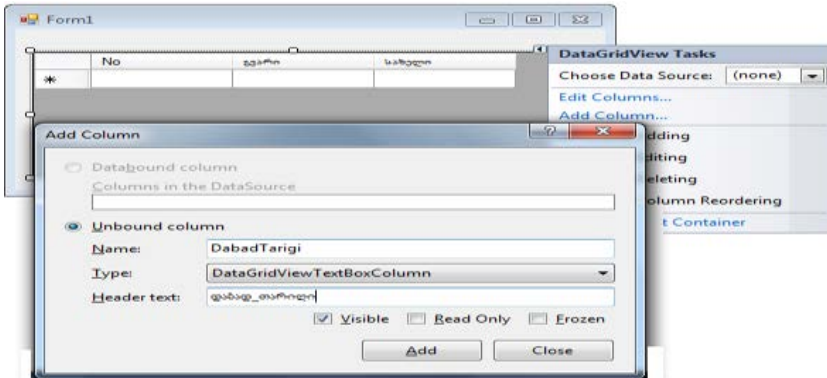
ცხრილები, რომლებიც სტრიქონებისა და სვეტებისგან შედგება, საუკეთესო საშუალებაა მონაცემთა ორგანზომილებიანი ველების, მასივების წარმოსადგენად. C# ენაში ასეთი ობიექტების ასახვის მიზნით გამოიყენება მართვის ელემენტი, ტიპით DataGridView. ამ ელემენტს, პრაგმატული თვალსაზრისით, დიდი გამოყენება აქვს მონაცემთა ბაზების სისტემებში (ADO.NET), რასაც ჩვენ შემდეგში განვიხილავთ.

8.1 ნახაზზე მოცემულია ToolBox-დან Form1-ფორმაზე გადმოტანილი DataGridView ელემენტი და საწყისი სიტუაცია.



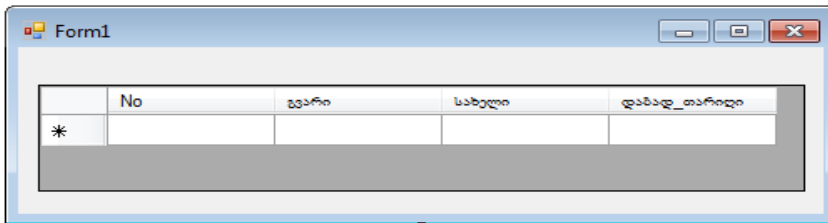
ნახ.8.1

დიალოგურ რეჟიმში ცხრილის ველების (სვეტების) შესატანად ვირჩევთ Add Columns და გადავდივართ 8.2 ნახაზზე მოცემულ ფანჯარაში. შეიტანოთ მიმდევრობით „სტუდენტები“-ს ატრიბუტები, მაგალითად, No, First_Name (გვარი), Last_Name (სახელი), Birth_data (დაბად_თარიღი) და ა.შ.



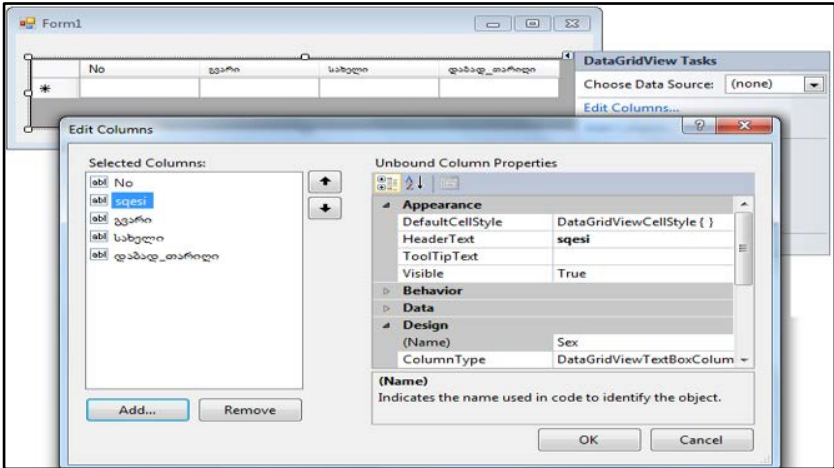
ნახ.8.2

პროგრამის ამუშავების შემდეგ მივიღებთ 8.3 ნახაზზე მოცემულ ცხრილს.

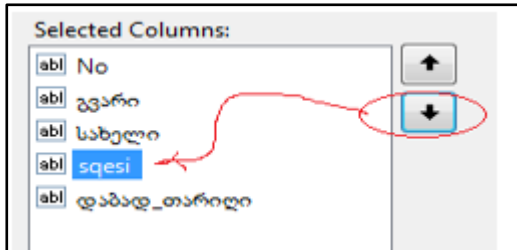


ნახ.8.3

შეტანილი ველების კორექტირებისათვის ვიყენებთ Edit Columns პუნქტს. ჩავამატოთ ველი Sex (სქესი) ან შვასწოროთ უკვე ჩაწერილი დასახელებები. მაგალითად, 8.3 ნახაზზე, ედიტორის ფანჯარაში გვინდა ველი sqesi შევცვალოთ ქართული შრიფტით და ამასთანავე იგი გადავიტანოთ ველი „სახელი“-ს შემდეგ. 8.4 და 5 ნახაზებზე ნაჩვენებია ეს შემთხვევები.

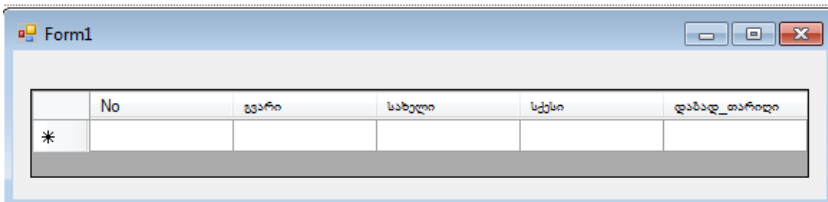


ნახ.8.4



ნახ.8.5

კოდის მუშაობის ახალი შედეგი მოცემულია 8.6 ნახაზზე.



ნახ.8.6

აქვე შეიძლება მონაცემთა სტრიქონების (Rows) შეტანა ველების ქვეშ. მაგალითი ნაჩვენებია 8.7 ნახაზზე.

No	ვარი	სახელი	სქესი	დაბად_თარიღი
93501	არაბული	ავთანდილი	კაცი	1990
93502	ბურდული	ნატალია	ქალი	1991
93515	დოლიძე	ნათია	ქალი	1991
93601	ავალიანი	უშვულა	კაცი	1990
93521	ჯავახიშვილი	ვანო	კაცი	1992

ნახ.8.7

პროგრამის დამთავრებისა და ხელახალი ამუშავების შემდეგ შეტანილი მონაცემები იკარგება, ანუ არაა შენახული მეხსიერებაში. თუ გვინდა, რომ პროექტის გაშვებისას მონაცემები ჩაიტვირთოს პროგრამულად, მაშინ ან უნდა გამოვიყენოთ მონაცემთა ბაზასთან კავშირი (იხ. მომდევნო თავი), ან კოდის Form2_Load მეთოდში უნდა ჩავწეროთ შემდეგი სტრიქონები (ლისტინგი 8_1).

// ლისტინგი_8.1 --- DataGridView -----

```
private void Form2_Load(object sender, EventArgs e)
```

```
{
```

```
    int i;
```

```
    // ველების (სვეტების) შევსება -----
```

```
    dataGridView1.Columns.Add("No", "No");
```

```
    dataGridView1.Columns.Add("FirstName", "გვარი");
```

```
    dataGridView1.Columns.Add("LastName", "სახელი");
```

```
    dataGridView1.Columns.Add("Sex", "სქესი");
```

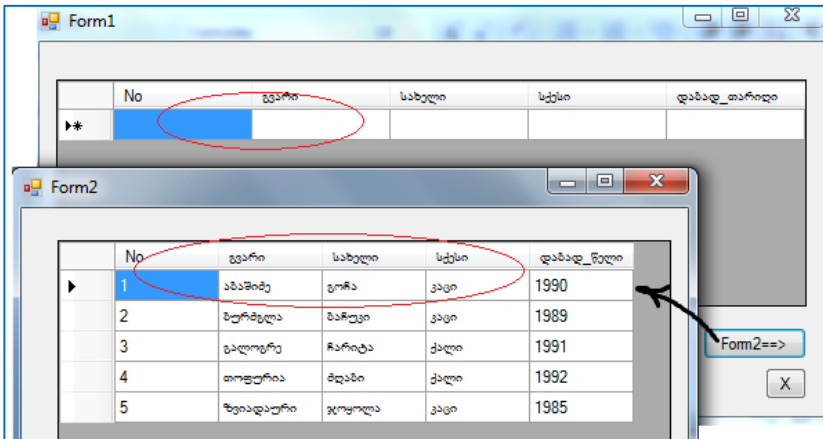
```
    dataGridView1.Columns.Add("Dab_Celi", "დაბად_წელი");
```

```
    // ველების სიგანის დაყენება -----
```

```

for (i = 0; i < dataGridView1.Columns.Count; i++)
    dataGridView1.Columns[i].Width = 75;
// სტრიქონების შევსება -----
dataGridView1.Rows.Add("1", "არაბული", "ავთო", "კ", "1990");
dataGridView1.Rows.Add("2", "ბურდული", "ნატო", "ქ", "1991");
dataGridView1.Rows.Add("3", "დოლიძე", "ნათია", "ქ", "1991");
dataGridView1.Rows.Add("4", "ავალიანი", "უმგულა", "ქ", "1990");
dataGridView1.Rows.Add("5", "ჯავახიშვილი", "ვანო", "კ", "1992");
}
    
```

პროგრამის ამუშავებით მიიღება 8.8 ნახაზზე მოცემული სურათი.



ნახ.8.8

როგორც აღვნიშნეთ, კოდში ხისტადაა შეტანილი კონკრეტული მონაცემები სტუდენტების შესახებ. ნებისმიერი დამატება ან ცვლილება მოითხოვს პროგრამის გადაკეთებას, რაც არაა რეკომენდებული. ამის თავიდან აცილება შესაძლებელია მონაცემთა ბაზის გამოყენებით.

შედგედან ჩანს, რომ Form1 ცარიელია, ხოლო Form2 შევსებულია საწყისი მონაცემებით.

ახლა განვიხილოთ ჩვენი კოდის მაგალითით DataGridView ცხრილში შეტანილი მონაცემების საფუძველზე მომხმარებლის მოთხოვნების პროგრამული დამუშავების შესაძლებლობანი.

ამოცანა_8.1: ვიპოვოთ სახელები და გვარები ყველა 1999 წელს დაბადებული კაცი სტუდენტის.

მოთხოვნის ფორმალური მხარე მდგომარეობს „გვარი“ ველის მნიშვნელობების გამობეჭდვაში, ველი „სქესი“=“კაცი“ მნიშვნელობისთვის.

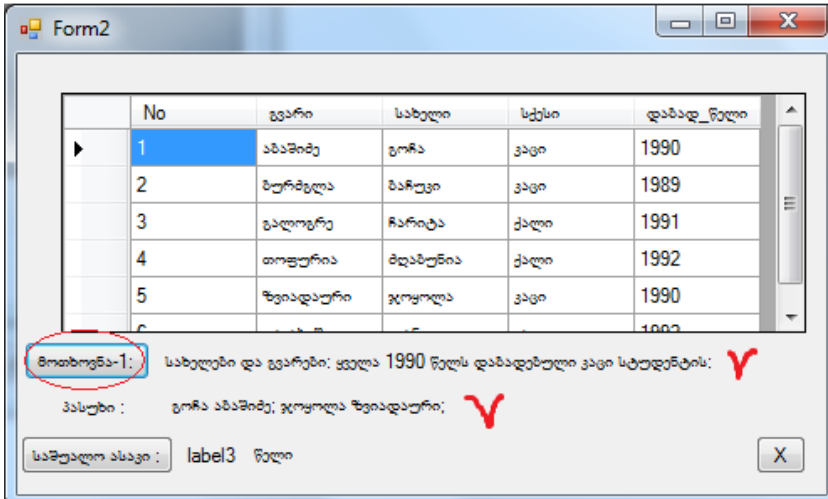
საჭიროა Form2-ზე დავდოთ ბუტონი და მივაბათ მას 8_2 ლისტინგის პროგრამული კოდი.

//-ლისტინგი_8.2---DataGridView-ში სტრიქონების

// ამორჩევა პირობით -----

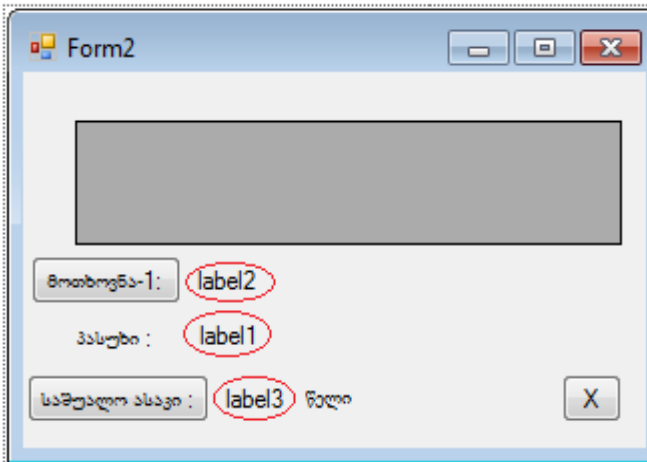
```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = " ";
    label2.Text = "სახელები და გვარები: ყველა 1999 წელს
                    დაბადებული კაცი სტუდენტის:";
    for (int i = 0; i < dataGridView1.Rows.Count; i++)
    {
        if (dataGridView1.Rows[i].Cells[3].Value == "კაცი" &&
            dataGridView1.Rows[i].Cells[4].Value == "1999")
        {
            label1.Text += dataGridView1.Rows[i].Cells[2].Value + " " +
                dataGridView1.Rows[i].Cells[1].Value + ";";
        }
    }
}
```

შედგეები გამოტანილია 8.9 ნახაზზე.



ნახ.8.9

ამოცანა_8.2: შევადგინოტ კოდი ღილაკისთვის „საშუალო ასაკი“, რომელიც გამოიტანს label3-ში ყველა სტუდენტის საშუალო ასაკის მნიშვნელობას (ნახ. 8.10). 8_3 ლისტინგზე მოცემულია ეს კოდი.



ნახ.8.10

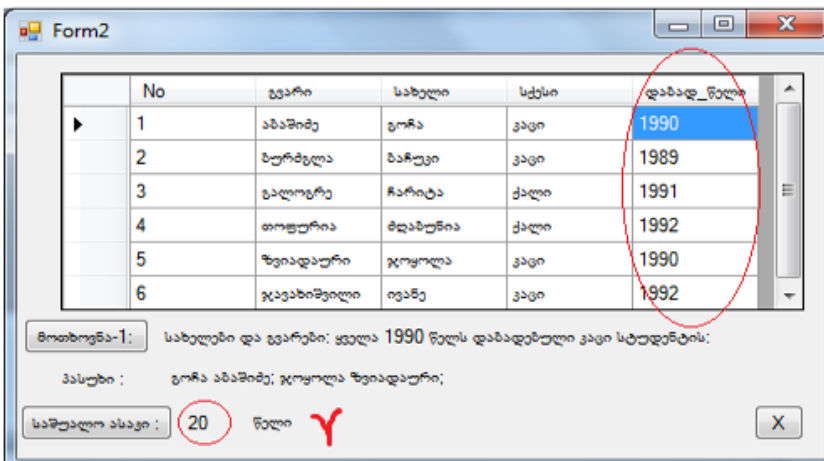
// ლოსტინგი_8.3 --- DataGridView სამუალო ასაკის ანგარიში -----

```
private void button2_Click(object sender, EventArgs e)
{
    int Birth_Year, Averag_Age, Sum=0;
    DateTime now = DateTime.Now; // მიმდინარე თარიღი -
                                // სისტემური

    int age,a,b;
    for (int i = 0; i < dataGridView1.Rows.Count; i++)
    {
```

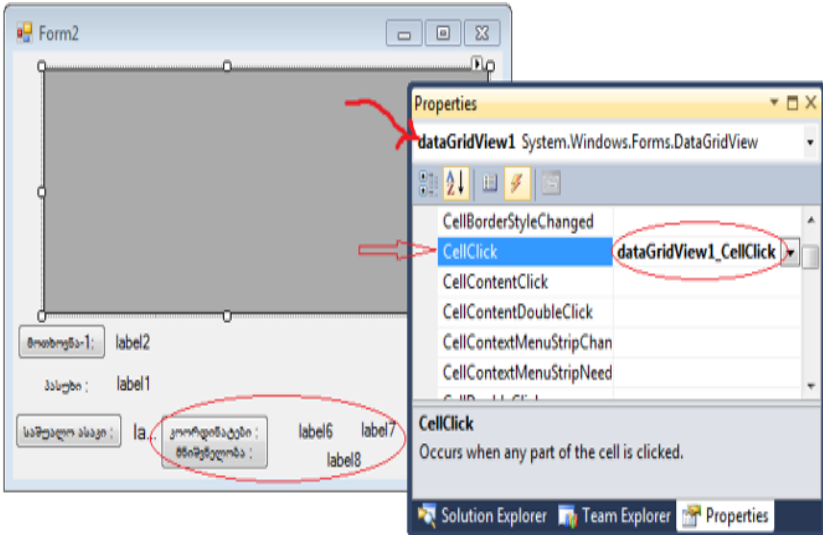
```
        Birth_Year=Convert.ToInt32(dataGridView1.Rows[i].Cells[4].Value);
        a = now.Year;
        b = Birth_Year;
        age = a - b;
        Sum += age;
    }
    Averag_Age = Sum / dataGridView1.Rows.Count;
    label3.Text = Averag_Age.ToString();
}
```

შედეგები ასახულია 8.11 ნახაზზე.



ნახ.8.11

ამოცანა_8.3: ავგოთ კოდი, რომელიც იმუშავებს ცხრილთან. კერძოდ, მაუსის კურსორის ცხრილის რომელიმე უჯრაზე დაწკაპუნებით, ეკრანზე გამოიტანს ამ უჯრის სვეტის და სტრიქონის კოორდინატებს და შიგ მოთავსებულ მნიშვნელობას. მოვლენის შექმნა მოცემულია 8.12 ნახაზზე.

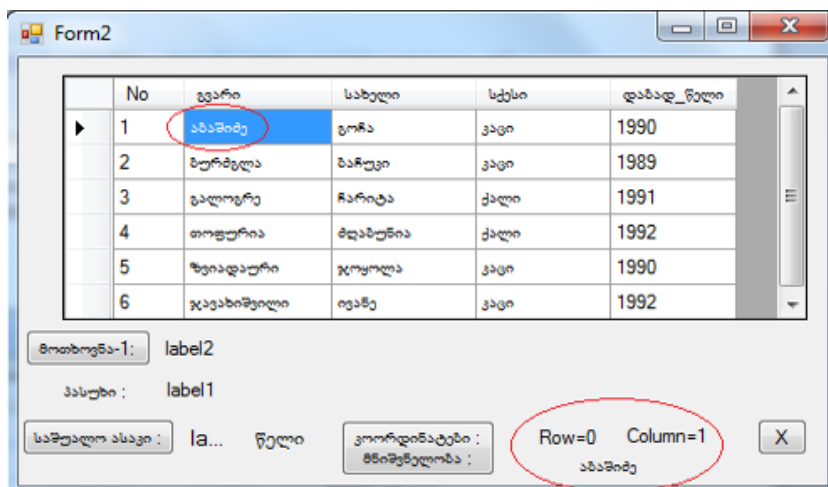


ნახ.8.12

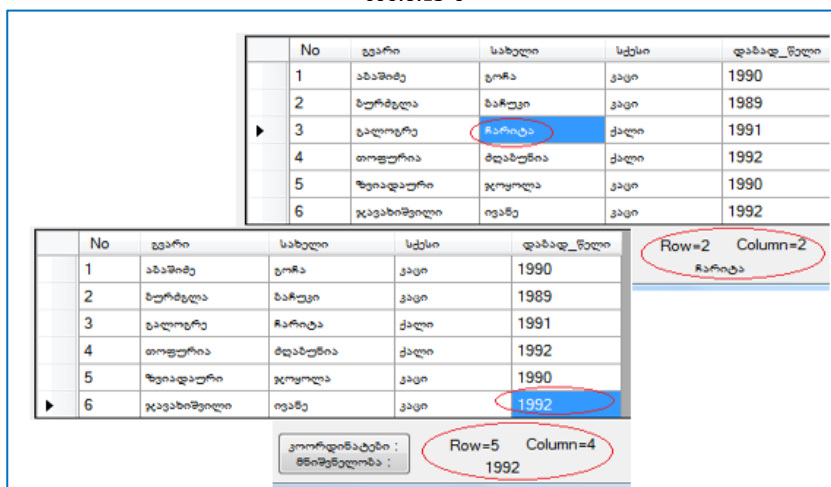
// ლისტინგი_8.4 ---- ცხრილის კოორდინატები ----

```
private void dataGridView1_CellClick(object sender,
DataGridViewCellEventArgs e)
{
    label8.Text = "";
    label6.Text = "Row="+e.RowIndex.ToString();
    label7.Text = "Column="+e.ColumnIndex.ToString();
    if(e.RowIndex>=0 && e.ColumnIndex >=0)
        label8.Text +=  dataGridView1.Rows[e.RowIndex]
            .Cells[e.ColumnIndex].Value;
}
```

მდეგები ასახული 8.13-ა,ბ ნახაზზე.



ნახ.8.13-ა

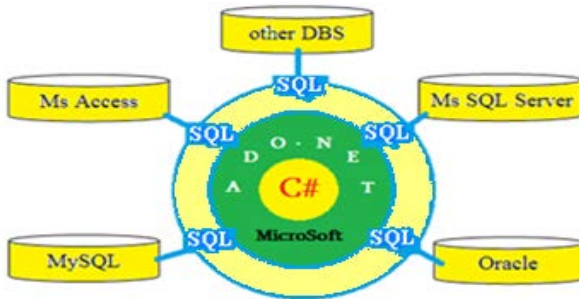


ნახ.8.13-ბ

8.2. ADO.NET: Visual C# კავშირი მონაცემთა ბაზებთან

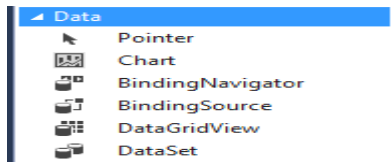
მომხმარებელთა პროგრამულ აპლიკაციებს (დანართებს) აუცილებლად ესაჭიროება ურთიერთქმედება მონაცემთა ცენტრალიზებულ ბაზებთან, მონაცემთა საცავებთან XML-ფორმატით (Extensible Markup Language), ან მონაცემთა ლოკალურ ბაზებთან, როდესაც ისინი მუშაობენ კლიენტის მანქანებზე.

Microsoft Visual Studio .NET Framework 4.5 პლატფორმა, რომელსაც ჩვენ ვიხილავთ C#-ენის საფუძველზე, მონაცემთა ბაზებთან სამუშაოდ იყენებს ADO.NET ტექნოლოგიას და SQL ენას (ნახ.8.14). პირველი დამაკავშირებელი დრაივერია C#-ენასა და ბაზებს შორის, მეორე კი - მომხმარებლის საკონტაქტო ენაა ბაზებთან, ე.წ. სტრუქტურირებულ მოთხოვნათა ენა [1,13].



ნახ.8.14. C# <-> ADO.NET <-> DBS

C# ენა .NET გარემოში მონაცემთა ბაზებთან სამუშაოდ გვთავაზობს შემდეგ კომპონენტებს (ნახ.8.15), რომელთა საფუძველია ADO.NET.



ნახ.8.15

ADO.NET - მაიკროსოფტის ტექნოლოგიაა (ActiveX Data Object), რომელიც გვთავაზობს მონაცემებთან მიმართვისათვის გამოსაყენებლად მარტივ, მაგრამ მეტად მძლავრ საშუალებებს. იგი უზრუნველყოფს სისტემის რესურსების მაქსიმალურად სრულ ურთიერთქმედებას [23-25]. წინამდებარე პარაგრაფში გავეცნობით:

- ADO.NET დრაივერის მონაცემებთან მიმართვის ძირითად კომპონენტებს;

- თითოეული კომპონენტის როლს ფუნქციონალობას;

- ADO.NET-ის მონაცემებთან მიმართვის ორგანიზაციის სცენარის აღწერას.

- C# პროგრამული აპლიკაციის კავშირს მონაცემთა რელაციური ბაზების მართვის სისტემებთან: Ms Access, MySQL, Ms SQL Server.

სხვადასხვა დანართები მონაცემებთან მიმართვის ორგანიზაციისათვის აყენებს სხვადასხვა მოთხოვნებს. მნიშვნელობა არა აქვს იმას, თუ რას აკეთებს დანართი: ასახავს ცხრილების შინაარსს, თუ გადაამუშავებს და განახლებს მონაცემებს ცენტრალურ SQL-სერვერზე. ADO.NET აძლევს მომხმარებელს მონაცემებთან მიმართვის მარტივ და ეფექტურ საშუალებებს რეალიზაციის სხვადასხვა სცენარით.

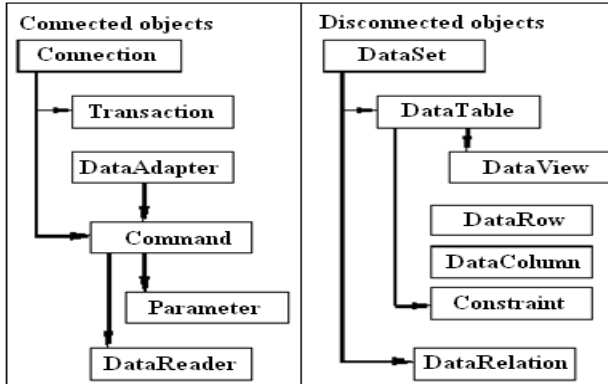
8.3. ADO.NET მონაცემთა არქიტექტურა

სისტემის ობიექტური მოდელი ორი ნაწილისგან შედგება: მარცხენა - მიერთებადი ობიექტები (Connected Objects) და მარჯვენა - განცალკევებადი ობიექტები (Disconnected Objects). 8.16 ნახაზზე ნაჩვენებია ADO.NET ობიექტური მოდელის შემადგენელი კლასები, რომელთა დანიშნულებასაც მოკლედ შევხებით ამ პარაგრაფში.

მონაცემებთან მიმართვა ADO.NET-ში ხორციელდება ორი კომპონენტით:

- მონაცემთა ერთობლიობით (DataSet ობიექტით), რომელშიც მონაცემები ინახება ლოკალურ კომპიუტერში;

- მონაცემთა მიმწოდებლით (DataProvider პროვაიდერით), რომელიც ასრულებს შუამავლის ფუნქციას პროგრამასა და მონაცემთა ბაზას შორის.



ნახ.8.16

ობიექტი DataSet: ესაა მონაცემთა წარმოდგენა კომპიუტერის მეხსიერებაში მონაცემთა წყაროსგან იზოლირებულად. ეს ობიექტი შეიძლება განვიხილოთ, როგორც მონაცემთა ბაზის ფრაგმენტის ლოკალური ასლი (კოპიო).

DataSet-ში მონაცემთა ჩატვირთვა შესაძლებელია ნებისმიერი დასაშვები წყაროდან, მაგალითად, SQL Server, Ms Access ბაზებიდან ან XML-ფაილიდან. დასაშვებია მეხსიერებაში ამ მონაცემებით მანიპულირება, აგრეთვე მათი განახლება მთავარი წყაროსაგან დამოუკიდებლად.

ობიექტი DataSet შედგება DataTable ობიექტთა ერთობლიობისგან (ის შეიძლება ცარიელიც იყოს, ანუ არ შეიცავდეს არც ერთ DataTable-ს).

ყოველი DataTable ობიექტი კომპიუტერის მეხსიერებაში ასახავს ერთ ცხრილს. მისი სტრუქტურა შეიცავს ორ ერთობლიობას: DataColumn, რომელშიც თავსდება ცხრილის სვეტები, და ცხრილის შეზღუდვათა ერთობლიობა. ეს ორი ერთობლიობა ქმნის ცხრილის სქემას.

DataTable ობიექტი შეიცავს აგრეთვე DataRow-ებს, რომლებიც ინახება DataSet-ობიექტის მონაცემები.

გარდა ამისა, DataSet ობიექტი შეიცავს DataRelations ერთობლიობას, რომელიც უზრუნველყოფს კავშირების შექმნას სხვადასხვა ცხრილის სტრუქტურებს შორის. DataRelations შეიცავს DataRelation ობიექტთა ერთობლიობას, რომლებიც განსაზღვრავს ცხრილთაშორის კავშირებს (მაგალითად, 1:M კავშირის სარეალიზაციოდ).

და ბოლოს, DataSet ობიექტი შეიცავს ExtendedProperties ერთობლიობას, რომელიც შეინახება დამატებითი მონაცემები.

მონაცემთა პროვაიდერი: ესაა ურთიერთდაკავშირებულ კომპონენტთა ერთობლიობა, რომელიც უზრუნველყოფს ეფექტურ მაღალმწარმოებლურ კავშირს მონაცემთა ბაზასთან.

.NET Framework-ს აქვს ორი პროვაიდერი: SQL Server .NET Data Provider, რომელიც შექმნილია SQL Server 7.0 ან უფრო მაღალ ვერსიებთან სამუშაოდ, და OleDb .NET Data Provider - სხვა ტიპის მონაცემთა ბაზებთან დასაკავშირებლად.

მონაცემთა ნებისმიერი პროვაიდერი შედგება მსგავსი უნივერსალური კლასების კომპონენტებისგან:

- Connection, რომელიც უზრუნველყოფს მონაცემთა ბაზასთან მიერთებას;

- Command, რომელიც გამოიყენება მონაცემთა წყაროს სამართავად. იგი გამოიყენებს ბრძანებებს, რომლებიც არ აბრუნებს მონაცემებს, მაგალითად, INSERT, UPDATE და DELETE, ან ბრძანებებს, რომლებიც აბრუნებს DataReader ობიექტს (მაგალითად, SELECT);

- DataReader გამოიყენება მხოლოდ ჩანაწერთა ერთობლიობის წასაკითხად მიერთებული მონაცემთა წყაროდან;

- DataAdapter შეავსებს გამოყოფილ DataSet ან DataTable ობიექტს და განაახლებს მათ შედგენილობას.

მონაცემებთან მიმართვა ხორციელდება შემდეგნაირად: ობიექტი Connection აყენებს დანართის (აპლიკაციის) მონაცემთა ბაზასთან მიერთებას, რომელიც პირდაპირ მისაწვდომია Command და DataAdapter ობიექტებისთვის. Command ობიექტი უზრუნველყოფს ბრძანებათა შესრულებას უშუალოდ მონაცემთა ბაზაში. თუ შესასრულებელი ბრძანება აბრუნებს რამდენიმე მნიშვნელობას, მაშინ Command ხსნის მათთან მიმართვას DataReader ობიექტის საშუალებით. მიღებული შედეგები შესაძლებელია დამუშავდეს უშუალოდ დანართის კოდით, ან DataSet ობიექტით, რომელიც შეიცვება DataAdapter ობიექტის დახმარებით. მონაცემთა ბაზის განახლებისთვის ასევე გამოიყენება Command და DataAdapter ობიექტები.

ობიექტი Connection გვთავაზობს მიერთებას მონაცემთა ბაზასთან. Visual Studio .NET –ს აქვს Connection-ის ორი კლასი: SqlConnection (MsSQL_Server-თან შესაერთებლად) და OleDbConnection (სხვა ტიპის მონაცემთა ბაზებთან დასაკავშირებლად). მონაცემთა ბაზასთან კავშირის არხის გასახსნელი აუცილებელი მონაცემები ინახება Connection ობიექტის ConnectionString თვისებაში. ეს ობიექტი ინახავს აგრეთვე რიგ მეთოდებს, რომლებიც საჭიროა მონაცემთა დასამუშავებლად ტრანზაქციების გამოყენებით.

ობიექტს Command აქვს ორი კლასი: SqlCommand და OleDbCommand. იგი უზრუნველყოფს ბრძანებათა გამოყენებას მონაცემთა ბაზაზე, რომელთაგანაც დამყარებულია კავშირი (მიერთება). აქ შეიძლება გამოყენებულ იქნას შენახვადი პროცედურები (Stored Procedures), SQL-ენის ბრძანებები, აგრეთვე ოპერატორები მთლიანი ცხრილების მისაღებად. Command ობიექტს აქვს სამი მეთოდი :

- Execute Non Query: იყენებს ბრძანებებს, რომლებიც არ აბრუნებს მონაცემებს, მაგალითად, INSERT, UPDATE და DELETE;

- Execute Scalar: იყენებს მოთხოვნებს მონაცემთა ბაზისადმი, რომლებიც აბრუნებს მხოლოდ ერთ მნიშვნელობას;

- Execute Reader: აბრუნებს საშუალოდ ერთობლიობას SqlDataReader ობიექტის საშუალებით.

ობიექტი SqlDataReader გვთავაზობს ნაკადს მონაცემთა ბაზის ჩანაწერების ერთობლიობით, ოღონდ მხოლოდ ერთი მიმართულებით წასაკითხად. მონაცემთა პროვაიდერის სხვა კომპონენტებისგან განსხვავებით SqlDataReader-ის ეგზეკუტარების შექმნა პირდაპირ არაა დასაშვები. მისი მიღება შეიძლება Command ობიექტის ExecuteReader მეთოდებით:

- SqlCommand.ExecuteReader მეთოდი აბრუნებს SqlDataReader ობიექტს;

- OleDbCommand.ExecuteReader მეთოდი კი - OleDbDataReader ობიექტს.

თუ SqlDataReader ობიექტის შემცველი მონაცემების ჩაწერა დისკზე არაა საჭირო, მაშინ ეს სტრიქონები შეიძლება პირდაპირ გადაეგზავნოს დანართს. ვინაიდან დროის ნებისმიერ მომენტში მებსიერებაში იმყოფება მხოლოდ ერთი სტრიქონი, SqlDataReader ობიექტის გამოყენება თითქმის არ ამცირებს სისტემის მწარმოებლურობას, ოღონდ მოითხოვს მონოპოლურ მიმართვას გახსნილ Connection-ობიექტზე SqlDataReader ობიექტის სასიცოცხლო დროის განმავლობაში.

ობიექტი DataAdapter არის ADO.NET-ის ძირითადი კლასი, რომელიც უზრუნველყოფს გამოყოფილ მონაცემებთან მიმართვას. არსებითად, იგი ასრულებს შუამავლის ფუნქციებს მონაცემთა ბაზისა და DataSet-ობიექტის ურთიერთ-ქმედებისთვის.

Fill მეთოდის გამოძახებისას DataAdapter ობიექტი შეავსებს მონაცემებით DataTable-ს ან DataSet-ს მონაცემთა ბაზიდან. მონაცემების დამუშავების შემდეგ, რომლებიც ჩატვირთულია მებსიერებაში, შესაძლებელია მოდიფიცირებული ჩანაწერების მოთავსება მონაცემთა ბაზაში, DataAdapter ობიექტის Update

მეთოდის გამოძახებით. DataAdapter-ს აქვს ოთხი თვისება, რომლებიც წარმოადგენს მონაცემთა ბაზის ბრძანებებს:

- SelectCommand შეიცავს ტექსტს ან ბრძანების ობიექტს, რომელიც ახორციელებს მონაცემთა ბაზიდან ამორჩევას (მაგალითად, მეთოდი Fill);
- InsertCommand შეიცავს ტექსტს ან ბრძანების ობიექტს, რომელიც ახორციელებს სტრიქონის ჩასმას ცხრილში;
- DeleteCommand შეიცავს ტექსტს ან ბრძანების ობიექტს, რომელიც ახორციელებს სტრიქონის წაშლას ცხრილიდან;
- UpdateCommand შეიცავს ტექსტს ან ბრძანების ობიექტს, რომელიც ახორციელებს მნიშვნელობათა განახლებას მონაცემთა ბაზაში;

Update მეთოდის გამოძახებისას ყველა შეცვლილი მონაცემი კოპირდება DataSet ობიექტიდან მონაცემთა ბაზაში, შესაბამისი ბრძანებების InsertCommand, DeleteCommand ან UpdateCommand გამოყენებით.

8.4. მონაცემთა ბაზასთან მიერთება

Visual Studio .NET სისტემას აქვს სტანდარტული ოსტატი პროგრამებისა და დიზაინერების სიმრავლე, რომელთა საშუალებითაც ადვილად და ეფექტურად ხორციელდება მონაცემებთან წვდომის არქიტექტურა დანართების დამუშავების პროცესში. ამასთანავე ADO.NET ობიექტური მოდელის ყველა შესაძლებლობა მისაწვდომია პროგრამულად, რაც უზრუნველყოფს არასტანდარტული ფუნქციების რეალიზაციის ან დანართების აგების შესაძლებლობას, რომლებიც მომხმარებელთა მოთხოვნილებებზეა ორიენტირებული.

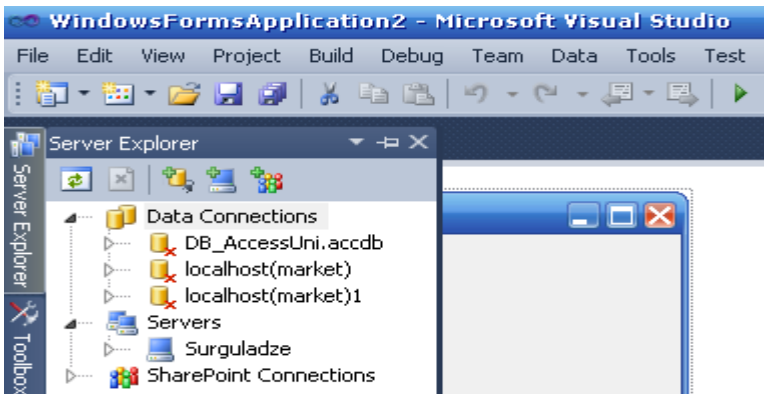
აქ ჩვენ გავეცნობით, თუ როგორ დავუკავშირდეთ მონაცემთა ბაზას ADO.NET-ის გამოყენებით, როგორ ამოვიღოთ საჭირო მონაცემები და გადავცეთ ისინი პროგრამულ აპლიკაციას. ეს

საკითხები შეიძლება შესრულდეს Visual Studio .NET-ის გრაფიკული ინსტრუმენტებითაც და პროგრამულადაც.

C# პროგრამულ აპლიკაციაში არსებობს მონაცემთა ბაზასთან მიერთების რამდენიმე ხერხი. ყველაზე მარტივია ამის განხორციელება Visual Studio .NET-ის გრაფიკული ინსტრუმენტით. მონაცემთა წყაროსთან (DataSource) მიერთების და მისი მართვისათვის გამოიყენება ფანჯარა Server Explorer.

ძირითადი ამოცანა, რომელსაც ჩვენ აქ განვიხილავთ, არის ADO.NET პროგრამული პაკეტის გამოყენებით მომხმარებელთა სამუშაო ინტერფეისის დამუშავების სადემონსტრაციო მაგალითის აგება. ამასთანავე, მონაცემთა ბაზების სახით უნდა გამოვიყენოთ Ms_Access, MySQL და Ms_SQL_Serever პაკეტებით აგებული ცხრილები.

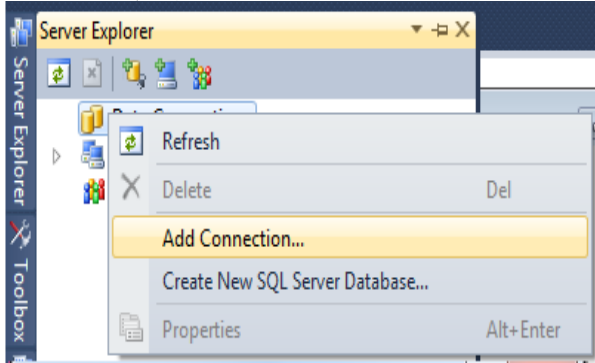
დავუშვათ, რომ ზემოაღნიშნული მონაცემთა ბაზების მართვის სისტემები დაინსტალირებულია ჩვენს კომპიუტერზე. თვით Visual Studio .NET -ის დაინსტალირებისას ავტომატურად ყენდება Ms SQL Server Expression, რომელზეც ასევე შესაძლებელია ექსპერიმენტის ჩატარება. .NET სამუშაო გარემოს ჩატვირთვის შემდეგ საჭიროა Server Explorer-ის გახსნა და ბაზებთან კავშირის შემოწმება (მაგალითად, ნახ.8.17).



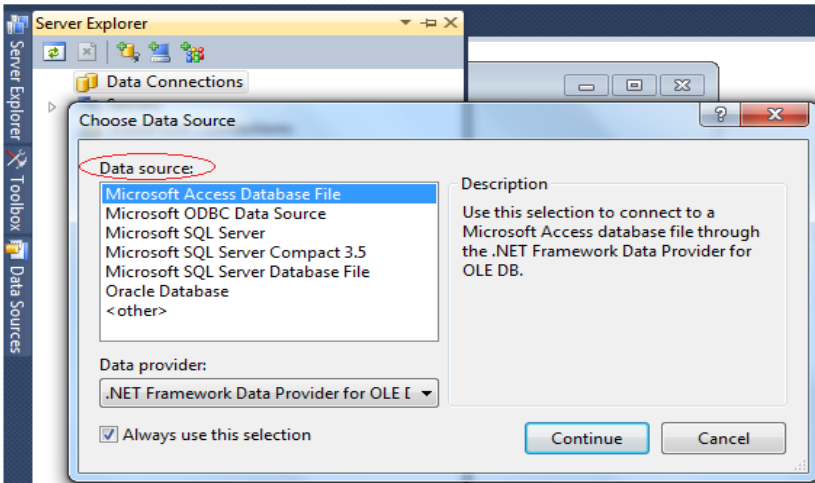
ნახ.8.17

8.5. C#.NET და Ms Access

სისტემის მენიუდან View | Server Explorer-ით გამოვიტანოთ ფანჯარა (ნახ.8.18) და ავირჩიოთ Add Connection.



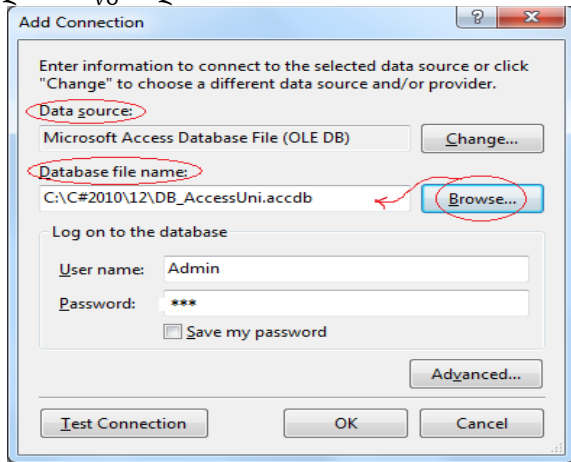
ნახ.8.18



ნახ.8.19

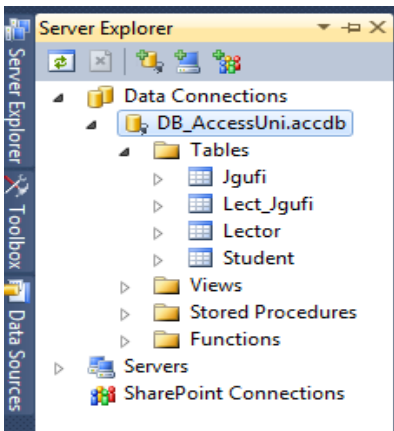
8.19 ნახაზზე Data Source ველში ავირჩიოთ სტრიქონი Microsoft Access Database File და Continue. მივიღებთ 8.20 ფანჯარას, რომელშიც Browse ღილაკით გამოვიძახებთ კატალოგის მართვის დიალოგის ფანჯარას და მივუთითებთ ჩვენს მიერ წინასწარ

მომზადებულ Ms Access-ის ბაზის ფაილს. მაგალითად, როგორც ეს Database file name ველშია ჩაწერილი.



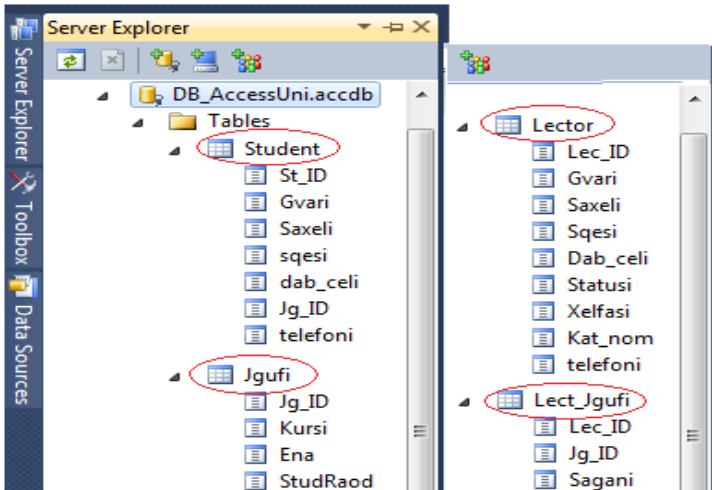
ნახ.8.20

აქვე, საჭიროების შემთხვევაში, მიეთითება User name და Password. ამის შემდეგ Server Explorer-ში გამოჩნდება 8.21 ნახაზზე მოცემული სურათი.



ნახ.8.21

როგორც ვხედავთ, Data Connection-ში უნივერსიტეტის მონაცემთა ბაზის ფაილი - **DB_AccessUni.accdb** გამოჩნდა, რომელიც შედგება ოთხი ცხრილისგან: Jgufi, Lect_Jgufi, Lector და Student. ცხრილები შედგება ველებისგან, რომელთაგან ერთ-ერთი გასაღებურია (ინდექსი): Lec_ID, St_ID, Jg_ID და ერთივე შედგენილი გასაღებია ორი ატრიბუტით: Lec_ID+Jg_ID (ნახ.8.22).

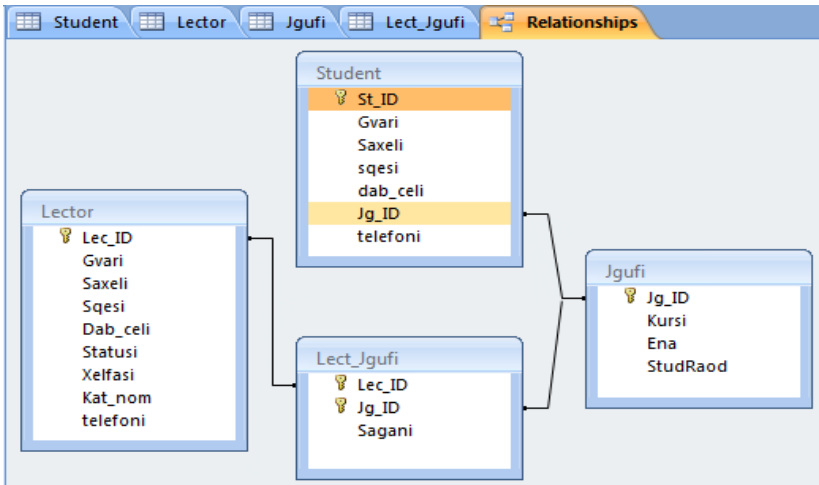


ნახ.8.22

რელაციური კავშირები მათ საფუძველზეა აგებული (ნახ.8.23).

ცხრილში **სტუდენტი** პირველადი გასაღებური ატრიბუტია St_ID ინდექსი, ხოლო მისი მეორადი გასაღებია Jg_ID, რომლითაც იგი უკავშირდება ცხრილს **ჯგუფი**, პირველადი ინდექსით Jg_ID. ესაა კავშირი 1:N, რომელიც ასახავს ბიზნეს-წესს (არსებულ კანონზომიერებას), რომ ერთი სტუდენტი შეიძლება იყოს მხოლოდ ერთ ჯგუფში და ერთ ჯგუფში შეიძლება იყოს რამდენიმე (N) სტუდენტი. ასევე, **ლექტორი** ასწავლის რამდენიმე (N) ჯგუფს, მაგრამ ჯგუფსაც ჰყავს რამდენიმე (M) ლექტორი. ესაა M:N კავშირი.

მისი რეალიზაცია არაა შესაძლებელი **ლექტორი**-ს და **ჯგუფი**-ს პირდაპირი კავშირით (განმეორებადი ველების პრობლემა !). ამისათვის შემოტანილია დამატებითი ცხრილი (რელაცია) **ლექტორი_ჯგუფი**. მასში შედგენილი ინდექსი იქმნება Lec_ID+Jg_ID, რომლებიც უკავშირდება ცალკ-ცალკე **ლექტორს** და **ჯგუფს** (ნახ.8.23).



ნახ.8.23

Student	Lector	Jgufi	Lect_Jgufi	Relationships		
St_ID	Gvari	Saxeli	sqesi	dab_cel	Jg_ID	telefoni
8	აკოფოვი	რობერტიზო	კაცი	1989	108936	297-11-11-11
1	ალავიძე	ალეკო	კაცი	1990	108935	222-22-20
2	ბურდული	ნინო	ქალი	1991	108935	137-33-33
3	ბურბულა	დიტო	კაცი	1989	108935	599-10-20-20
4	გაბედავა	ვახტანგ	კაცი	1992	108935	333-67-89
5	გაბელია	ცუცა	ქალი	1992	108935	222-45-67
13	გალოგრე	ხვიჩა	კაცი	1989	108937	270-44-44
6	დანელია	მიმოზა	ქალი	1990	108935	577-44-44-45
9	დოლიძე	რიჩარდი	კაცი	1990	108936	597-34-56-78
14	დუნდუა	გოჩა	კაცი	1991	108937	599-22-33-44
15	ვასაძე	სოკრატე	კაცი	1985	108937	577-33-67-55
10	ზარანდია	მუშნი	კაცი	1980	108936	597-12-23-34
11	თოფურია	მღაბი	ქალი	1991	108936	577-10-10-10
12	კეკელია	კეკელა	ქალი	1992	108936	579-30-30-30
7	ხვიტია	მუკუ	კაცი	1992	108935	593-45-67-89
16	ჯალალონია	მაცი	კაცი	1992	108937	577-99-00-00

ნახ.8.24-ა

ვიზუალური დაპროგრამება (C#.NET & Workflow Foundation NET)

Student	Lector	Jgufi	Lect_Jgufi	Relationships				
Lec_ID	Gvari	Saxeli	Sqesi	Dab_cel	Statusi	Xelfasi	Kat_nomr	telefoni
6	ბარათელი	ანი	ქალი	1970	ასოც.პროფესორი	600	94 599-23-23-23	
7	კუცია	თეა	ქალი	1987	ლაბორანტი	280	94 577-12-13-14	
8	გაბედავა	ომიკო	კაცი	1950	სრ.პროფესორი	900	94 577-33-55-22	
9	დეალი	დავითი	კაცი	1950	სრ.პროფესორი	900	86 599-99-90-90	
10	ფიფია	კოჩი	კაცი	1960	ასოც.პროფესორი	650	86 233-33-46	
11	მეფარია	თვარისა	ქალი	1980	ას.პროფესორი	450	94 593-55-55-55	
12	წინიძე	ლია	ქალი	1980	ასოც.პროფესორი	600	94 577-78-89-90	
13	ოდინარია	ოდინა	კაცი	1956	ას.პროფესორი	450	86 236-37-38	
14	სამხარაძე	ვახუშა	კაცი	1970	ასოც.პროფესორი	690	51 577-88-99-00	

ნახ.8. 24-ბ

Student	Lector	Jgufi	Lect_Jgufi	Relation
Jg_ID	Kursi	Ena	StudRaod	
108050	2	ქართული	29	
108051	2	ქართული	30	
108059	2	რუსული	12	
108835	4	ქართული	29	
108836	4	ქართული	25	
108935	3	ქართული	28	
108936	3	ქართული	30	
108937	3	ქართული	17	
108940	3	ინგლისური	20	

ნახ.8. 24-გ

Student	Lector	Jgufi	Lect_Jgufi	Relation
Lec_ID	Jg_ID	Sagani		
	5	კომპიუტერის არქიტექტურა		
	8	6 კომპიუტერის არქიტექტურა		
	8	7 სერვერული ტექნოლოგიები		
	8	8 სერვერული ტექნოლოგიები		
	8	9 კომპიუტერის არქიტექტურა		
	9	7 მათემატიკა		
	9	8 მათემატიკა		
	10	12 მონაცემთა ბაზები		
	10	13 მონაცემთა ბაზები		

ნახ.8. 24-დ

ჩანაწერის წინ „+“ სიმბოლო ხსნის კავშირს მეორე, იერარქიულად დაქვემდებარებულ ცხრილთან (ნახ.8.25).

Student	Lector	Jgufi	Lect_Jgufi	Katedr
Lec_ID	Gvari	Saxeli	Sqesi	
6	ბარათელი	ანი	ქალი	
7	კუცია	თეა	ქალი	
8	გაბედავა	ომიკო	კაცი	
Jg_ID	Sagani		Add	
5	კომპიუტერის არქიტექტურა			
6	კომპიუტერის არქიტექტურა			
7	სერვერული ტექნოლოგიები			
8	სერვერული ტექნოლოგიები			
9	კომპიუტერის არქიტექტურა			
*				
9	დევალი	დავითი	კაცი	
10	ფიფია	კოჩი	კაცი	

ნახ.8.25

ამგვარად, მონაცემთა ბაზა DB_AccessUni.acce მზადაა. ახლა განვიხილოთ C# პროგრამიდან მონაცემთა ბაზასთან წვდომის საკითხი. ეს პროცესი შედგება ოთხი ბიჯისგან:

- მონაცემთა ბაზასთან მიერთება (რაც ზემოთ განვიხილეთ Server Explorer->Data Connection-ში);
- SQL-ბრძანების გადაცემა მონაცემთა ბაზაზე;
- SQL-ბრძანების შეფასება (და შესრულება);
- მონაცემთა ბაზასთან კავშირის დახურვა.

SQL-ბრძანება წარმოადგენს სტრუქტურირებული მოთხოვნების ენაზე დაწერილ სკრიპტს, რომელიც გასაგებია მონაცემთა ბაზების მართვის სისტემისთვის და ასრულებს მას. ძირითადად, არსებობს ორი ტიპის მოთხოვნა:

- select : მონაცემთა ამორჩევის SQL-ბრძანება;
- insert, delete, update : მონაცემთა ბაზაში ცვლილებების განსახორციელებელი SQL-ბრძანებები.

C# პროგრამულ პროექტს მონაცემთა ბაზასთან სამუშაოდ სჭირდება სახელსივრცე OleDb, რომელიც using.System.Data.OleDb ბრძანებითაა რეალიზებული.

SQL-ბრძანებები Ms_Access ბაზაში გადაიგზავნება OleDb სახელსივრცის OleDbCommand კლასის ობიექტით. ამ კლასის ორი მნიშვნელოვანი თვისებაა: Connection (დავალება ბაზასთან

დასაკავშირებლად, საითაც გაიგზავნება SQL-მოთხოვნა) და CommandText (თვით SQL ბრძანების ტექსტი).

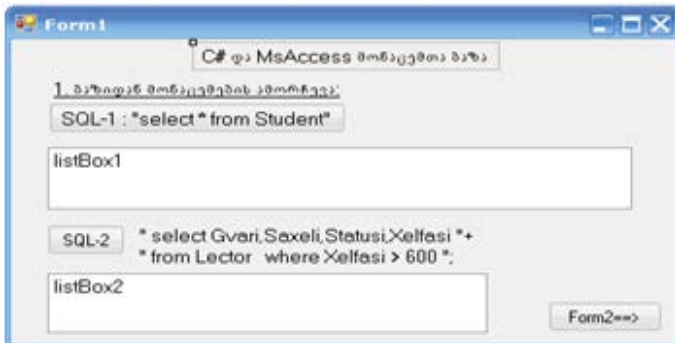
მოთხოვნის ტიპებისგან დამოკიდებულებით (არჩევითი, ცვლილებების), OleDbCommand კლასს გააჩნია შემდეგი მეთოდები:

ExecuteReader() - აქვს select მოთხოვნის გაგზავნის და შედეგების მიღების ფუნქცია;

ExecuteNonQuery() - ემსახურება ცვლილების (insert, delete, update) აქციის ჩატარებას და რიცხვის მიღებას, რომელიც გვიჩვენებს, თუ მონაცემთა ბაზის რამდენ ჩანაწერს შეეხო ეს პროცედურა. მონაცემთა ბაზიდან select-მოთხოვნით ამორჩეული ჩანაწერები (ველებით და მნიშვნელობებით) ინახება OleDb სახელსივრცის OleDbReader კლასის ობიექტში. განვიხილოთ კონკრეტული მაგალითი.

ამოცანა_8.1: DB_AccessUni.acce უნივერსიტეტის მონაცემთა ბაზაში: 1 - ვნახოთ ყველა სტუდენტის ყველა ატრიბუტის მნიშვნელობა (ანუ მთლიანი ინფორმაცია); 2 - ვიპოვოთ იმ ლექტორთა გვარი, სახელი, სტატუსი და ხელფასი, რომელთა ხელფასი მეტია 600 ლარზე.

8.26-ა ნახაზზე 1-ელ მოთხოვნას შესაბამება SQL-1, ხოლო მეორეს კი - SQL-2 “select” კონსტრუქცია. ისინი ორ დილაკზეა მიმაგრებული. საილუსტრაციო მაგალითში შედეგები გამოიტანება listBox1 და listBox2 ველებში.



ნახ.8.26-ა

C#-პროგრამის კოდი ამ ორი ღილაკისთვის მოცემულია 8.5_ლისტინგში.

// ლისტინგი_8.5 – C# + Ms Access -----

```
private void button1_Click(object sender, EventArgs e)
{
    OleDbConnection con = new OleDbConnection();
    OleDbCommand cmd = new OleDbCommand();
    OleDbDataReader reader;
    con.ConnectionString =
        "Provider=Microsoft.ACE.OLEDB.8.0;" +
        "Data Source=C:\\C#2010\\WinADO\\DB_AccessUni.accdb";
    cmd.Connection = con;
    cmd.CommandText = "select * from Student";
    try
    {
        con.Open();
        reader = cmd.ExecuteReader();
        listBox1.Items.Clear();
        while (reader.Read())
        {
            listBox1.Items.Add(reader["St_ID"] + " : " +
                reader["Gvari"] + " : " +
                reader["Saxeli"] + " : " +
                reader["sqesi"] + " : " +
                reader["dab_celi"] + " : " +
                reader["Jg_ID"] + " : " +
                reader["telefoni"]);
        }
        reader.Close();
        con.Close();
    }
}
```

```

catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}
// end button1_click -----
private void button2_Click(object sender, EventArgs e)
{
    OleDbConnection con = new OleDbConnection();
    OleDbCommand cmd = new OleDbCommand();
    OleDbDataReader reader;
con.ConnectionString =
    "Provider=Microsoft.ACE.OLEDB.8.0;" +
    "Data Source=C:\\C#2010\\WinADO\\DB_AccessUni.accdb";
cmd.Connection = con;
cmd.CommandText =
    "select Gvari,Saxeli,Statusi,Xelfasi " +
    "from Lector " +
    "where Xelfasi > 600 ";
try
{
    con.Open();
    reader = cmd.ExecuteReader();
    listBox2.Items.Clear();
    while (reader.Read())
    {
        listBox2.Items.Add( reader["Gvari"] + " : " +
            reader["Saxeli"] + " : " +
            reader["Statusi"] + " : " +
            reader["Xelfasi"]);
    }
}

```



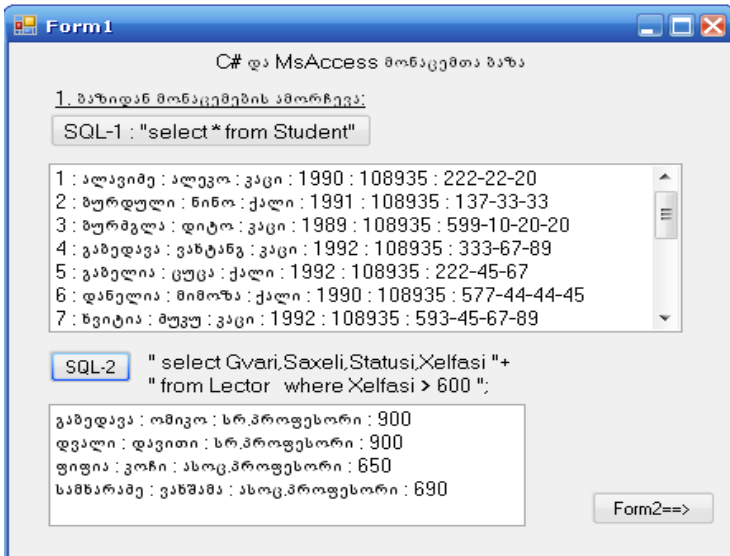
```

reader.Close();
con.Close();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}
// end button2_click -----

```

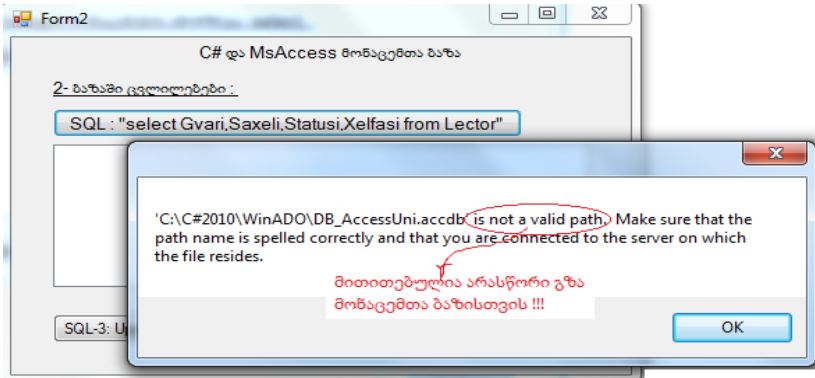
შენიშვნა: პროგრამის თავში using-სტრიქონებში უნდა ჩაემატოს“
using System.Data.OleDb;

8.27-ბ ნახაზზე ნაჩვენებია აღნიშნული მოთხოვნების შესრულების შედეგები MsAccess ბაზის Student და Lector ცხრილებიდან. პირველში „ * ”- ნიშნავს „ყველა“ - ველს, ხოლო მეორეში აიღება მხოლოდ „გვარი,სახელი, სტატუსი და ხელფასი“.



ნახ.8.27-ბ

პროგრამაში **try...catch** ბლოკით ხდება მონაცემთა ბაზასთან წვდომის პროცესში შესაძლო შეცდომების აღმოჩენა, რაც აადვილებს პროგრამისტის მუშაობას. მაგალითად, ინფორმაციის მიღება, რომ მონაცემთა ბაზის ფაილი არაა მითითებულ patch-კატალოგში (ნახ.8.28), ან რომ SQL-მოთხოვნის სინტაქსში შეცდომაა და ა.შ.



ნახ.8.28

Open() მეთოდით ხდება პროგრამის კავშირის გახსნა ბაზასთან. შემდეგ, ExecuteReader() მეთოდით მოთხოვნა გადაეგზავნება ბაზას. შედეგები ბრუნდება OleDbReader კლასით, რაც ხორციელდება reader მიმთითებლით (მაჩვენებლით).

ვინაიდან წინასწარ არაა ცნობილი თუ რამდენი სტრიქონი იქნება შედეგში, გამოიყენება ListBox, რომელიც წინასწარ სუფთავდება.

Read() მეთოდი გვაწვდის ბაზიდან ერთ ჩანაწერს (სტრიქონს) და ამავდროულად სპეც-მაჩვენებლით მიუთითებს მომდევნო ჩანაწერზე. თუ ჩანაწერი ბოლოა, მაშინ მაჩვენებლის მნიშვნელობა ხდება false. ეს მართვა ხორციელდება while ციკლით try-ბლოკში.

ჩანაწერის შიგნით ველების მნიშვნელობები შეესაბამება მათ ნომრებს ან დასახელებებს. შესაძლებელია ასევე ყველა ველის გამოტანა (*-ით), რომლებიც სპეც-გამყოფითაა (მაგ., „ : “) დაცილებული.

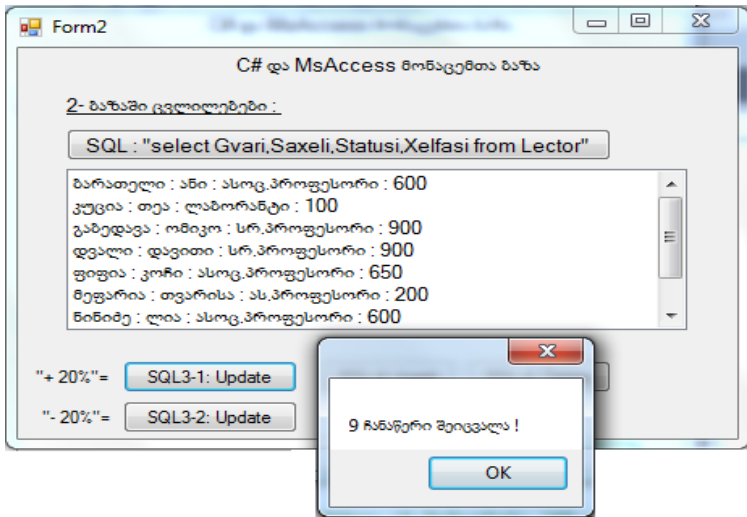
ბოლოს, Reader ობიექტი და კავშირი უნდა დაიხუროს Close() მეთოდით.

ამოცანა_8.2: მონაცემთა ბაზაში „უნივერსიტეტი“ DB_AccessUni.acce საჭიროა ცვლილებების განხორციელება insert(), delete() და update() მეთოდების გამოყენებით. Form2-ზე მოვათავსოთ შესაბამისი ელემენტები და განვახორციელოთ ტრანზაქციები (max.8.29):

ა) ყველა ლექტორის ხელფასი გაიზარდოს 20%-ით. (ამასთანავე შესაძლებელი უნდა იყოს საწყისი მონაცემების აღდგენა, ანუ შემცირდეს ხელფასები 20 %-ით);

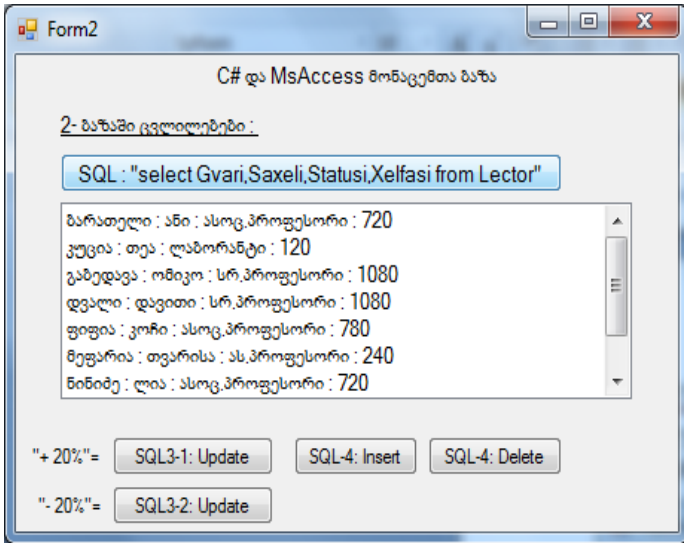
ბ) 108935 ჯგუფში დაემატოს ახალი სტუდენტი გვარით „ახალაძე“, სახელით „ნოუთბუკა“ და ა.შ.;

გ) ამოიშალოს ბაზიდან 108836 ჯგუფი, რომელმაც დაასრულა 4-წლიანი სწავლების კურსი.



ნახ.8.29

ლექტორთა ხელფასების შეცვლილი შედეგები მოცემულია 8.30 ნახაზზე.



ნახ.8.30

SQL3-2 ლილავით შემცირდება ხელფასის რაოდენობა 20%-ით, ანუ აღდგება პირველადი მონაცემები ბაზაში.

შესაბამისი update - კოდი მოცემულია 8.2_ლისტინგში.

// ლისტინგი_8.2 --- Ms Accesses "update" -----

```
using System;
using System.Data.OleDb;
using System.Windows.Forms;
namespace WinADO
{
public partial class Form2 : Form
{
public Form2() { InitializeComponent(); }
}
```

```
private void button1_Click(object sender, EventArgs e)
{
    OleDbConnection con = new OleDbConnection();
    OleDbCommand cmd = new OleDbCommand();
    int Raod;
    // ხელვასის ზრდა ან შემცირება 20%-ით ----
    con.ConnectionString = "Provider=Microsoft.ACE.OLEDB.8.0;" +
        "Data Source=C:\\C#2010\\12\\DB_AccessUni.accdb";
    cmd.Connection = con;
    if(ReferenceEquals(sender,button1))
        // op = "*" or "/" -----
        cmd.CommandText = "update Lector set Xelfasi=Xelfasi * 1.2";
    else
        cmd.CommandText = "update Lector set Xelfasi=Xelfasi / 1.2";
    try
    {
        con.Open();
        Raod = cmd.ExecuteNonQuery();
        MessageBox.Show(Raod + " ჩანაწერი შეიცვალა !");
        con.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void button4_Click(object sender, EventArgs e)
{
    OleDbConnection con = new OleDbConnection();
    OleDbCommand cmd = new OleDbCommand();
    OleDbDataReader reader;
```

```
con.ConnectionString =
    "Provider=Microsoft.ACE.OLEDB.8.0;" +
    "Data Source=C:\\C#2010\\12\\DB_AccessUni.accdb";
cmd.Connection = con;
cmd.CommandText = "select Gvari,Saxeli,Statusi,
                    Xelfasi from Lector";

try
{
    con.Open();
    reader = cmd.ExecuteReader();
    listBox1.Items.Clear();
    while (reader.Read())
    {
        listBox1.Items.Add(reader["Gvari"] + " : " +
            reader["Saxeli"] + " : " +
            reader["Statusi"] + " : " +
            reader["Xelfasi"]);
    }
    reader.Close();
    con.Close();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

კოდში update ტიპის ცვლილებისათვის DB_MsAccessUni.accdb ფაილში გამოყენებული კონსტრუქცია:

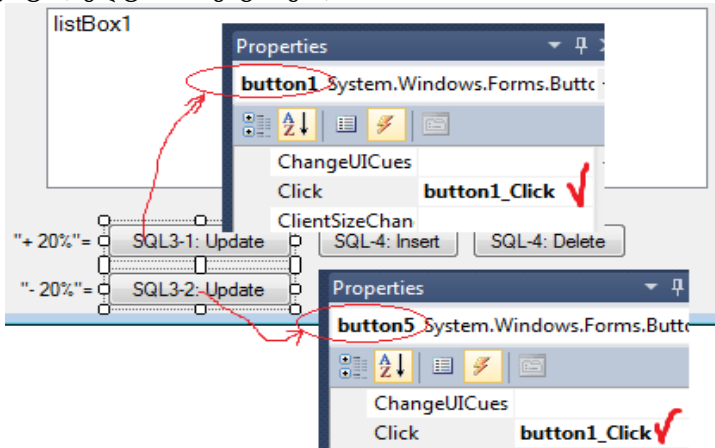
"update Lector set Xelfasi=Xelfasi * 1.2"

რაც ნიშნავს ცხრილის აქტუალიზაციას (update... set), სახელით Lector და ცხრილის ველის (ატრიბუტის) ცვლილების ალგორითმს (Xelfasi=Xelfasi * 1.2), რომელიც უნდა შესრულდეს ჩანაწერებზე;

try...catch ბლოკში იხსნება ბაზა (Open-ით) და გამოიძახება მეთოდი ExecuteNonQuery(), რომელიც დააბრუნებს ჩანაწერების როდენობას (Raod), რომელთაც შეეხო ცვლილება. ეს ინფორმაცია გამოიტანება MessageBox-ით.

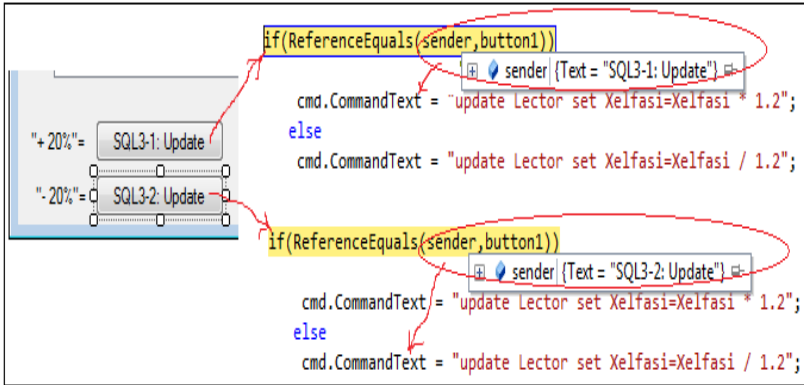
მონაცემთა ბაზის ცხრილში Lector ველისათვის Xelfasi უნდა მოხდეს მნიშვნელობათა გაზრდა 20%-ით (button1: SQL3-1) ან შემცირება (button5: SQL3-2), ნახ.8.31.

ამის განსახორციელებლად კოდში გამოყენებულია მეთოდი ReferenceEquals(sender object). ამ მეთოდით განისაზღვრება - აქვს თუ არა ორ ობიექტს ერთი მისამართი, ანუ ინახება თუ არა ისინი მესსიერების ერთიდაიმავე უჯრედებში. თუ „კი“, მაშინ „true“ და სრულდება გამრავლება (ხელფასის მომატება), ხოლო თუ სხვადასხვა ობიექტია, მაშინ გვექნება “false” და შესრულდება გაყოფა (ხელფასის შემცირება).



ნახ.8.31

კოდის ReferencialEquals(sender, button1) მეთოდში, იმისდა მიხედვით, თუ რომელი ბუტონი (SQL3-1 თუ SQL3-2) იქნება არჩეული, sender-ს მიენიჭება button1 ან button5 მიმთითებლის მნიშვნელობა (ნახ.8.32).



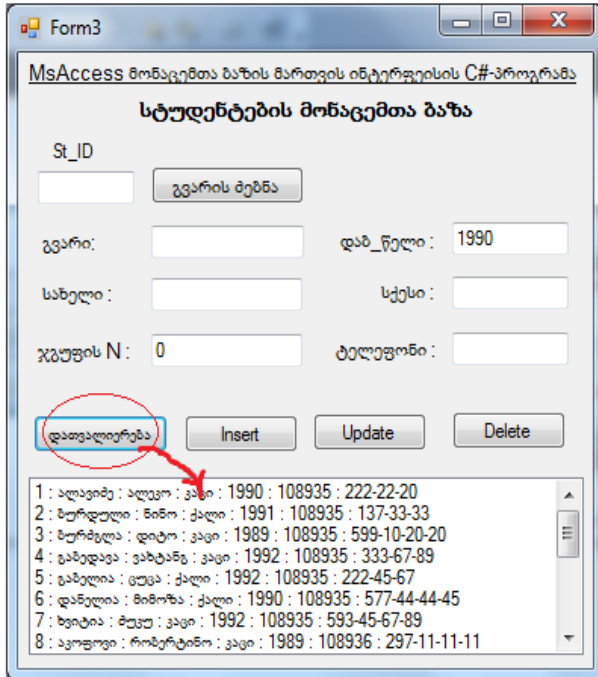
ნახ.8.32

ამგვარად, როდესაც sender და button1 ერთიდაიმავე მისამართზეა, მაშინ ხდება ხელფასის მომატება. დასასრულ, პროგრამის button4_Click ღილაკით listBox1-ში გამოიტანება MsAccess ბაზის Lector ცხრილის განახლებული მონაცემები (მომატებული ან დაკლებული ხელფასებით).

ამოცანა_8.3: საჭიროა აიგოს კოდი მომხმარებლის ინტერფეისი სტუდენტთა რეგისტრაციისათვის, რომელიც Ms Access მონაცემთა ბაზასთან იმუშავებს, განახორციელებს ჩანაწერების ძებნის, ჩამატების, წაშლის და კორექტირების სერვისულ ფუნქციებს.

საწყისი ფორმა 8.33 ნახაზზეა ნაჩვენები. "დათვალიერება" ღილაკი კავშირშია ცხრილ სტუდენტთან და გამოაქვს ჩანაწერები textBox1-ში.

პროგრამის საწყისი ფრაგმენტი (ბაზასთან მიერთება, მონაცემთა ინიციალიზაცია) და ღილაკი დათვალიერება მოცემულია 8.3_ლისტინგში.



ნახ.8.33

// ლისტინგი_8.3 --- MsAccess ბაზის მონაცემების მართვა ----

```
using System;
using System.Collections;
using System.Data.OleDb;
using System.Drawing;
using System.Windows.Forms;
namespace WinADO
{
    public partial class Form3 : Form
    {
        public Form3() {InitializeComponent(); }
        OleDbConnection con = new OleDbConnection();
```

```
OleDbCommand cmd = new OleDbCommand();
OleDbDataReader reader;
ArrayList stNummer = new ArrayList();

private void Form3_Load(object sender, EventArgs e)
{
    con.ConnectionString =
        "Provider=Microsoft.ACE.OLEDB.8.0;" +
        "Data Source=C:\\C#2010\\12\\DB_AccessUni.accdb";
    cmd.Connection = con;
}

private void button1_Click(object sender, EventArgs e)
{
    Datvaliereba();
}

private void Datvaliereba()
{
    try
    {
        con.Open();
        cmd.CommandText = "select * from Student";
        Ekranze_gamotana();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    con.Close();
    textBox1.Text=""; // St-ID
    textBox2.Text=""; // Gvari
}
```

```

textBox3.Text=""; // Saxeli
textBox4.Text=""; // sqesi
textBox5.Text="1990"; // dab_celi
textBox6.Text="0"; // Jg_ID
textBox7.Text=" "; // telefoni
}
private void Ekranze_gamotana()
{
    DateTime DabTarigi;
    reader = cmd.ExecuteReader();
    listBox1.Items.Clear();
    stNummer.Clear();
    while (reader.Read())
    {
        listBox1.Items.Add(
            reader["St_ID"] + " : " +
            reader["Gvari"] + " : " +
            reader["Saxeli"] + " : " +
            reader["sqesi"] + " : " +
            reader["dab_celi"] + " : " +
            reader["Jg_ID"] + " : " +
            reader["telefoni"]);
    }
    reader.Close();
}

private void button2_Click(object sender, EventArgs e)
{ // Insert დილაგის მეთოდი -----
    int Raod;
    try

```

```

{con.Open();
cmd.CommandText="insert into Student "+
    "(St_ID,Gvari,Saxeli,sqesi,dab_celi,Jg_ID,telefoni) "+
" values (" + textBox1.Text + "," +
    textBox2.Text + "," + textBox3.Text + "," +
    textBox4.Text + "," + textBox5.Text + "," +
    textBox6.Text + "," + textBox7.Text + ")";
MessageBox.Show(cmd.CommandText);
Raod=cmd.ExecuteNonQuery();
if(Raod >0)
    MessageBox.Show("ერთი სტრიქონი ჩაიწერა !");
}
catch(Exception ex)
{
    MessageBox.Show(ex.Message);
    MessageBox.Show("შეიტანეთ გვარი და სხვა მონაცემები...");
}
con.Close();
Datvaliereba();
}
}
}

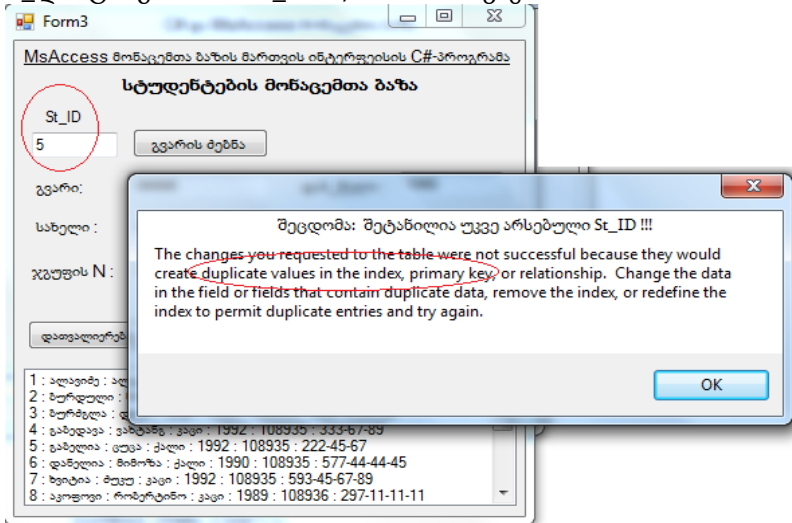
```

კოდის დასაწყისში Form3() კლასსა და Form3_Load მეთოდში გადმოტანილია ის ძირითადი კონსტრუქციები, რომლებიც საერთოა ფორმაზე მოთავსებული ელემენტებისათვის. ესაა con, cmd, stNummer ობიექტების შექმნა OleDbConnection(), OleDbCommand() და ArrayList() კლასების საფუძველზე. აგრეთვე ბაზასთან მიერთება მისი პროვაიდერისა და კატალოგის გზის მითითებით. SQL მოთხოვნებისა და კავშირის მაჩვენებლებისთვის განისაზღვრა OleDbDataReader-ის reader.

პირველი რეალიზებული მეთოდი, რომლის ლისტინგიც ზემოთ განვიხილეთ, არის აგებული Form3-ის ღილაკისთვის „დათვალიერება“ სტუდენტთა ცხრილისათვის. მისი გამოყენება შესაძლებელია მრავალჯერადად, ბაზის მონაცემთა ცვლილებების მონიტორინგის მიზნით.

აქ try...catch ბლოკში მოთავსებულია ბაზის გახსნა-დახურვის, select-მოთხოვნის სტრიქონი, შეცდომების „დაჭერის“ და შეტყობინების გამოცემის მექანიზმი და ბოლოს, ამორჩეული სტრიქონების გამოტანის მეთოდი Ekranze_gamotana(). შედეგები აისახება listBox1-ში.

მეორე რეალიზებული მეთოდი ეკუთვნის Insert -ღილაკს, ანუ შესაძლებელია Form3-ზე textBox-ებში ახალი სტუდენტის მონაცემების შეტანა და შემდეგ აღნიშნული ღილაკით (იხ. 8.3_ლისტინგში button2_Click) მისი მოთავსება Access ბაზაში.



ნახ.8.34

თუ მომხმარებელმა შეცდომით შეიტანა რომელიმე მონაცემი, სისტემა პოულობს მას და გამოაქვს შესაბამისი შეტყობინება.

მაგალითად, თუ შეტანილია სტუდენტის ინდექსი St_ID, რომელიც უკვე არსებობს, მაშინ მივიღებთ შეტყობინებას, რომელიც 8.34 ნახაზზეა ნაჩვენები. ბაზაში შეცდომიანი სტრიქონი არ ჩაიწერება.

ახლა განვიხილოთ ბაზაში ცვლილებების შეტანის (Update) და ჩანაწერების წაშლის (Delete) მეთოდების დაპროგრამების საკითხები.

ამოცანა_8.4: Ms Access ბაზაში არსებული “სტუდენტები”-ს ცხრილიდან Windows-ფორმაზე ListBox-ში გამოვიტანოთ ჩანაწერები. ავირჩიოთ შესაცვლელი სტრიქონი, რომლის სვეტის მნიშვნელობები აისახება ფორმის შესაბამის textBox-ველებში. აქ შევცვალოთ ამ ველების მნიშვნელობები. Update დილაკით ცვლილებები უნდა მოთავსდეს ბაზაში. შედეგების შემოწმების სისწორე განხორციელდება სტრიქონების ხელახალი ასახვით ListBox-ში.

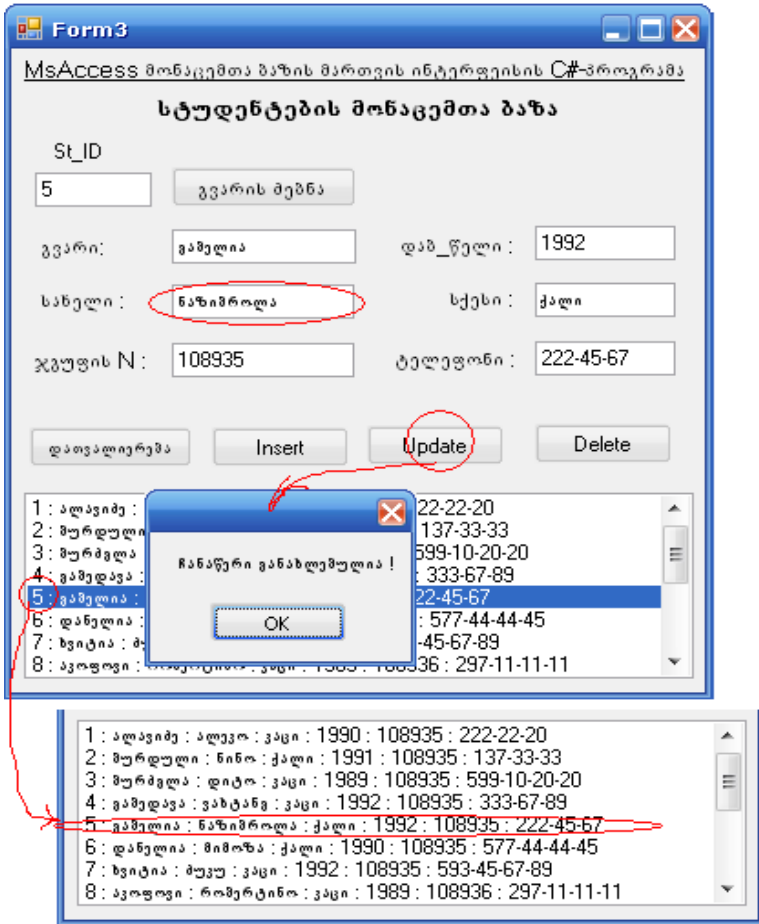
8.35 ნახაზზე მოცემულია ფორმის საწყისი მდგომარეობა. არჩეულია მე-5 სტრიქონი. textBox-ებში ჩაიწერა ამ სტრიქონის მონაცემები. საჭიროა შეიცვალოს სახელი=“ცუცა” ახლით.

The screenshot shows a Windows application window titled "Form3" with a blue title bar. The main area has a white background and contains the following elements:

- Title Bar:** "Form3" with standard Windows window controls (minimize, maximize, close).
- Form Header:** "MsAccess მონაცემთა ბაზის მართვის ინტერფეისის C#-პროგრამა" and "სტუდენტების მონაცემთა ბაზა".
- Fields:**
 - St_ID:** Text box containing "5".
 - გვარის მკვნა:** Button.
 - გვარი:** Text box containing "ვაშლია".
 - დაბ_წელი:** Text box containing "1992".
 - სახელი:** Text box containing "ცუცა".
 - სქესი:** Text box containing "ქალი".
 - ჯგუფის N:** Text box containing "108935".
 - ტელეფონი:** Text box containing "222-45-67".
- Buttons:** "დააქტიურება" (circled in red), "Insert", "Update", and "Delete".
- List Box:** A list of 8 student records. The 5th record is selected and highlighted in blue: "5: ვაშლია : ცუცა : ქალი : 1992 : 108935 : 222-45-67".

ნახ.8.35

8.36 ნახაზზე ნაჩვენებია განახლების პროცესის ფრაგმენტები. ველი **სახელი**=**ნაზიმროლა** (შესაძლებელია სხვა ველების ცვლილებაც), შემდეგ ღილაკით Update გამოიწვევა შეტყობინება "ჩანაწერი განახლებულია" (თუ ტრანზაქცია შესრულდა სწორად). თუ რამე შეცდომაა, გამოვა შესაბამისი შეტყობინება.



ნახ.8.36

8.4_ლისტინგის კოდი არის 8.3_ლისტინგის გაფართოება. აქ ორი მოვლენა პროგრამირდება. ერთი, listBox1-ში სტრიქონის არჩევა ცვლილების მიზნით, რომელსაც textBox-ებში გამოაქვს შესაბამისი მონაცემები. და მეორე, Update ღილაკის დაჭერით შესრულებული პროცედურა (მონაცემთა ბაზაში შენახვა).

//ლისტინგი_8.4 --- Update() -----

```
private void listBox1_SelectedIndexChanged(object sender,
                                         EventArgs e)
{ // სტრიქონის არჩევა ----
  try
  {
    con.Open();
    cmd.CommandText = "select * from Student " +
      " where St_ID = " + stNumber[listBox1.SelectedIndex];

    reader = cmd.ExecuteReader();
    reader.Read();
    textBox1.Text="" + reader["St_ID"];
    textBox2.Text="" + reader["Gvari"];
    textBox3.Text="" + reader["Saxeli"];
    textBox4.Text="" + reader["sqesi"];
    textBox5.Text="" + reader["dab_celi"];
    textBox6.Text="" + reader["Jg_ID"];
    textBox7.Text="" + reader["telefoni"];
    reader.Close();
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
  }
  con.Close(); }
```



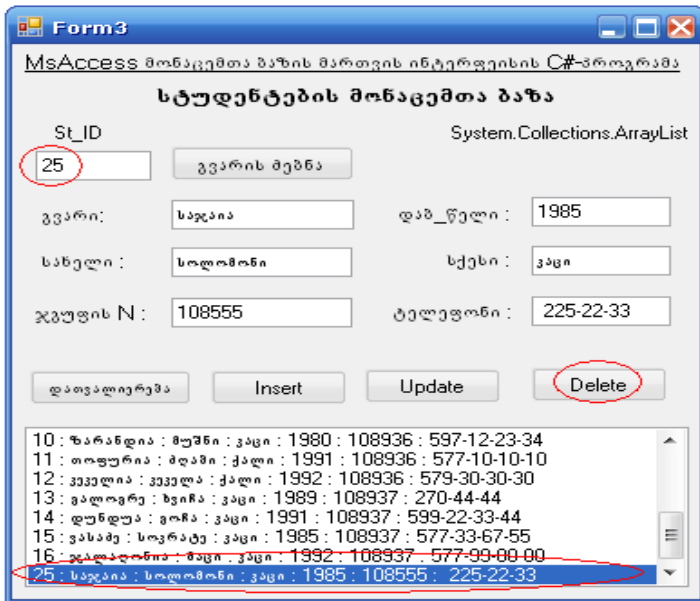
```

private void button3_Click(object sender, EventArgs e)
{ // update -----
    int Raod;
    try
    {
        con.Open();
        cmd.CommandText = "update Student set " +
            "St_ID = " + textBox1.Text + ", " +
            "Gvari = " + textBox2.Text + ", " +
            "Saxeli = " + textBox3.Text + ", " +
            "sqesi = " + textBox4.Text + ", " +
            "dab_celi = " + textBox5.Text + ", " +
            "Jg_ID = " + textBox6.Text + ", " +
            "telefoni = " + textBox7.Text + "" +
            " where St_ID = " + stNummer[listBox1.SelectedIndex];
        MessageBox.Show(cmd.CommandText);
        Raod = cmd.ExecuteNonQuery();
        if (Raod > 0)
            MessageBox.Show("ჩანაწერი განახლებულია !");
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message);
        MessageBox.Show("შეასწორეთ ერთი ჩანაწერი " +
            "აირჩიეთ რომელიმე გვარი," +
            " ცალსახა სტუდ_ნომერი და " +
            " სხვა მონაცემები !!!");
    }
    con.Close();
    Datvaliereba();
} // end Update

```

ამოცანა_8.5: განვიხილოთ Ms Access ბაზის ცხრილში “სტუდენტები” ჩანაწერის წაშლის პროცედურა, რომელიც Delete დილაკზე იქნება მიბმული.

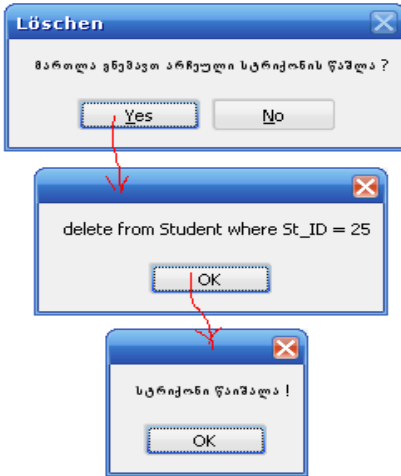
8.37-ა ნახაზზე ნაჩვენებია ფორმაზე მოთავსებული ცხრილის ”სტუდენტები” ჩანაწერები, რომელთაგან, ერთ-ერთი, კერძოდ 25-ე ჩანაწერი არჩეულია წასაშლელად.



ნახ.8.37-ა

Delete-დილაკის ამოქმედებით მიმდევრობით გამოჩნდება 8.37-ბ ნახაზზე ნაჩვენები შეტყობინებები. თუ ყველაფერი ნორმალურადაა, ბაზიდან მოხდება 25-ე ჩანაწერის ამოშლა.

წაშლის პროგრამის ტექსტი მოცემულია 8.5_ლისტინგში. აქაც try...catch ბლოკში cmd.CommandText სტრიქონში ჩაწერილია წაშლის SQL-ბრძანება, რომელიც შემდეგ ExecuteNonQuery() მეთოდით გადაეცემა შესასრულებლად. შედეგის ნახვა ხორციელდება პროგრამის ბოლოს Datvaliereba() მეთოდით.



ნახ.8.37-ბ

// ლისტინგი_8.5 --- Delete -----

```
private void button4_Click(object sender, EventArgs e)
{
    int Raod;
    if (textBox1.Text == "")
    {
        MessageBox.Show("აირჩიეთ ერთი სტრიქონი !");
        return;
    }
    if (MessageBox.Show("შართლა გნებავთ არჩეული" +
        " სტრიქონის წაშლა ?", "Löschen",
        MessageBoxButtons.YesNo) == DialogResult.No)
        return;
    try
    {
        con.Open();
        cmd.CommandText = "delete from Student " +
            "where St_ID = " + stNummer[listBox1.SelectedIndex];
        MessageBox.Show(cmd.CommandText);
    }
}
```

```

Raod = cmd.ExecuteNonQuery();
if (Raod > 0)
    MessageBox.Show("სტრიქონი წაიშალა!");
}
catch (Exception ex)
{ MessageBox.Show(ex.Message); }
con.Close();
Datvaliereba();
} //end Delete -----

```

ამოცანა_8.6: ავავოთ პროგრამა, რომელიც გვარის ველში შეტანილი ერთი, ორი ან მეტი სიმბოლოთი იპოვის მრავალჩანაწერიან ბაზაში შესაბამის სტრიქონებს და გამოიტანს მათ listBox1-ში (ნახ.8.38).

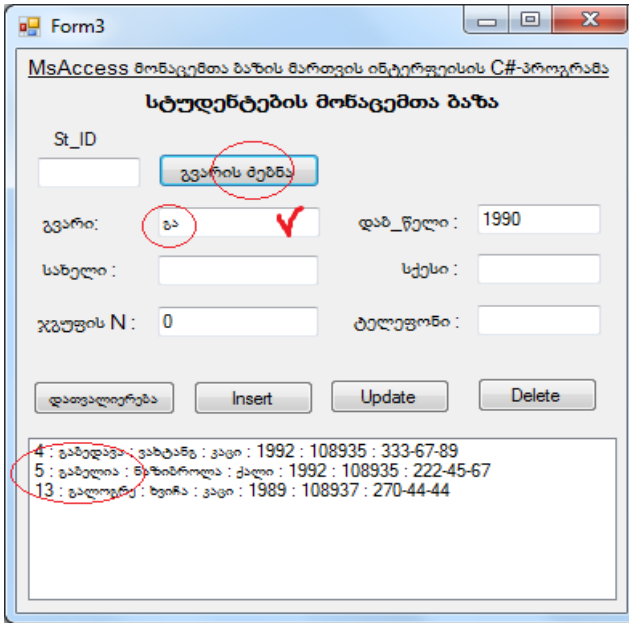
ეს კოდი მოცემულია 8.6_ლისტინგში.

// ლისტინგი_8.6 ---- გვართი ძებნა ----

```

private void button5_Click(object sender, EventArgs e)
{ try
    {
        con.Open();
        cmd.CommandText = "select * from Student where" +
            " Gvari like '%" + textBox2.Text + "%'";
        MessageBox.Show(cmd.CommandText);
        Ekranze_gamotana();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    con.Close();
}

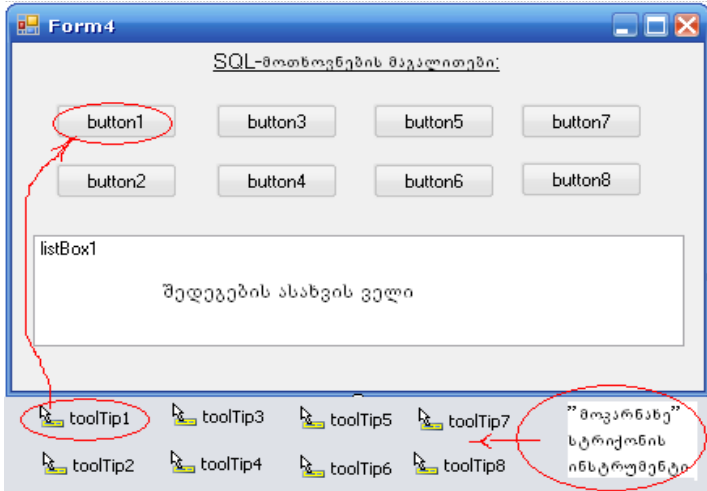
```



ნახ.8.38

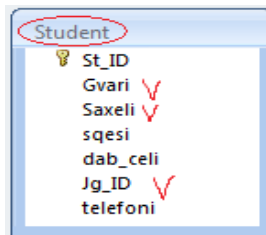
8.6. SQL მოთხოვნების დაპროგრამირების საილიუსტრაციო მაგალითები C#-ში MsAccess ბაზისთვის

ამოცანა 8.7: ავგაოთ C# კოდი - მომხმარებლის ინტერფეისი MsAccess მონაცემთა ბაზის რამდენიმე ცხრილთან ერთდროულად სამუშაოდ. გამოვიყენოთ DB_AccessUni ბაზის Table-ები: Student, Group, Lector და Lect_Jgufi (ნახ.8.21-8.23). საჭიროა მოთხოვნების წინასწარ დაპროექტება და მისი SQL-ფორმატით წარმოდგენა. შემდეგ ამ სტრუქტურირებული მოთხოვნის ჩასმა C# კოდში (მიმაგრება button_Click... ღილაკებზე ნახ.8.39)



ნახ.8.39

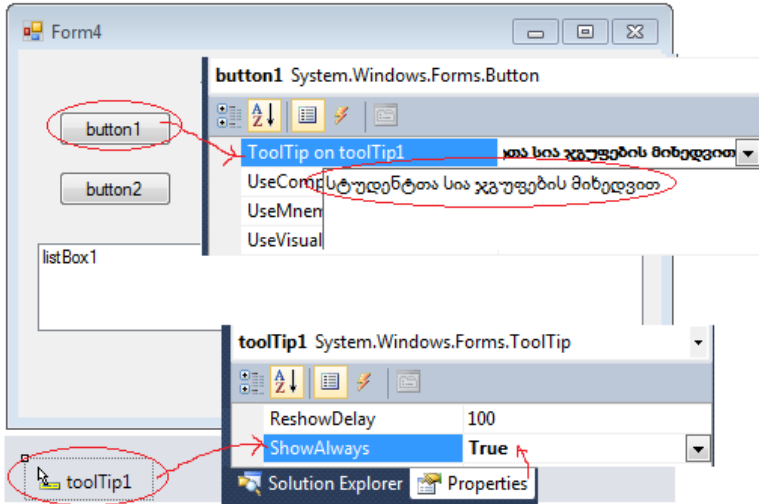
მოთხოვნა_1: "ეკრანზე გამოვიტანოთ სტუდენტთა სია: გვარი, სახელი, ჯგუფის_ნომერი, მოწესრიგებული ჯგუფის ნომრით და გვარით" (ნახ.8.40) .



ნახ.8.40

სასურველია მოთხოვნის დილაკების „გასემანტიკურება“ (მაუსის კურსორის მიტანისას დილაკზე გამოჩნდეს „მოკარნახე ტექსტი (hint-მინიშნება), თუ რას აკეთებს ეს დილაკი“).

ამისათვის ToolBox-პანელიდან ფორმაზე გადმოვიტანოთ ToolTip1-არავიზუალური ელემენტი, რომელიც მოთავსდება ფორმის ქვემოთ (ნახ.8.41). მოვნიშნოთ იგი და Properties-ში დავაყენოთ თვისება ShowAlways = true.



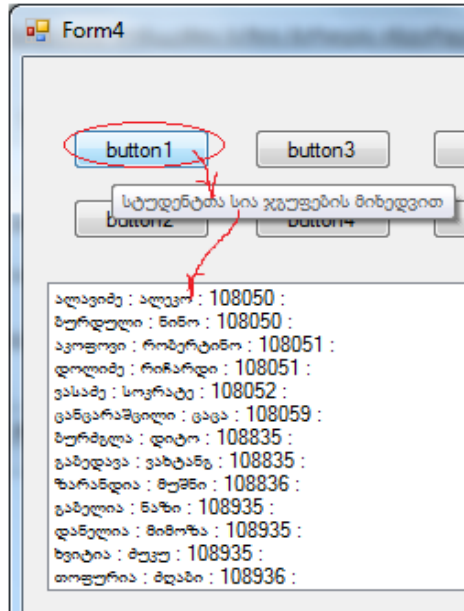
ნახ.8.41

შემდეგ მოენიშნოთ ღილაკი button1 და Properties-ში თვისებაში ToolTip on toolTip1 ჩაწეროთ „მოკარნახე“ ტექსტი. ავამუშავოთ პროგრამა, მივიტანოთ მაუსის კურსორი button1 ღილაკთან. გამოჩნდება ტექსტი „სტუდენტთა სია ჯგუფების მიხედვით“. თუ არ გვინდა ეს ინფორმაცია, გადავალთ სხვაზე. თუ ავამოქმედებთ button1-ს, მივიღებთ შედეგს (ნახ.8.42).

button1 ღილაკის პროგრამის კოდის ფრაგმენტი, რომელშიც ასახულია SQL-ტიპის მოთხოვნა, მოცემულია 8.7_ლისტინგში.

// ლისტინგი_8.7 --- button1 ----

```
private void button1_Click(object sender, EventArgs e)
{
    Amorcheva("select * from Student order by Jg_ID, Gvari",
              "Gvari", "Saxeli", "Jg_ID");
    MessageBox.Show("select * from Student order by Jg_ID,
                    'Gvari', 'Saxeli', 'Jg_ID' ");
}
```



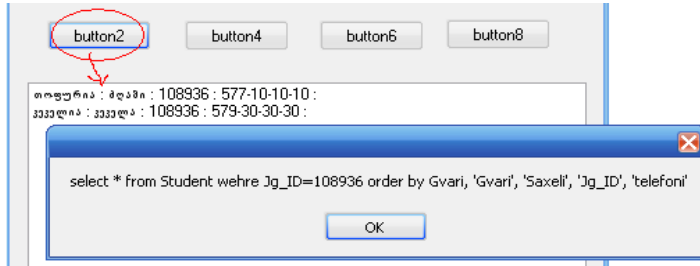
ნახ.8.42

მოთხოვნა_2: ვიპოვოთ ის სტუდენტები, რომლებიც სწავლობენ 108936 ჯგუფში. გამოვიტანოთ მათი გვარები, სახელები, ჯგუფის_ნომრები.

// ლისტინგი_8.8 --- button2 ----

```
private void button2_Click(object sender, EventArgs e)
{
    Amorceva("select * from Student where Student.Jg_ID=108936 "+
        "order by Gvari", "Gvari", "Saxeli", "Jg_ID", "telefoni");
    MessageBox.Show("select * from Student wehre Jg_ID=108936 "+
        "order by Gvari, 'Gvari', 'Saxeli', 'Jg_ID', 'telefoni' ");
}
```

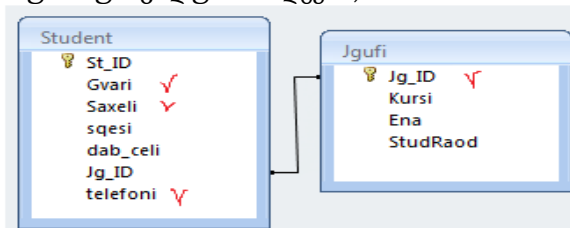
შედეგი მოცემულია 8.43 ნახაზზე.



ნახ.8.43

მოთხოვნა_3: "ვიპოვოთ იმ სტუდენტთა გვარები, სახელები, ჯგუფის_ნომრები და ტელეფონები, რომლებიც სწავლობენ ინგლისურენოვან ჯგუფში".

ამ შემთხვევაში საჭიროა ბაზიდან ორი ცხრილის გამოყენება: Student და Jgufi. 8.44 ნახაზიდან ჩანს, რომ ორი ეს ცხრილები კავშირშია ერთმანეთთან ატრიბუტით Jgufi.Jg_ID (პირველადი გასაღები, ანუ უნიკალური ინდექსი) და Student.Jg_ID (მეორადი გასაღები, ანუ არაუნიკალური ინდექსი).



ნახ. 8.44

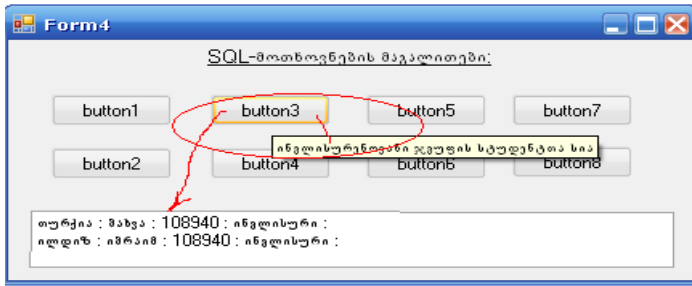
მათი გათვალისწინებით SQL-მოთხოვნას ექნება 8.9-ლისტინგზე მოცემული სახე.

// ლისტინგი_8.9 --- button3 ----

```
private void button3_Click(object sender, EventArgs e)
{
    Amorcheva("select * from Student,Jgufi " +
        "where Student.Jg_ID=Jgufi.Jg_ID and Ena='ინგლისური' " +
```

```

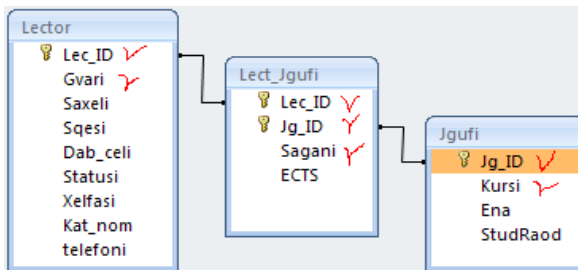
"order by Jgufi.Jg_ID, Gvari", "Gvari",
                "Saxeli", "Student.Jg_ID", "Ena");
MessageBox.Show("select * from Student,Jgufi "+
"where Student.Jg_ID=Jgufi.Jg_ID and Ena='ინგლისური' "+
"order by Gvari, 'Gvari', 'Saxeli',
                'Student.Jg_ID', 'Ena' ");
}
    
```



8.45

მოთხოვნა_4: „რომელი კურსის რომელ ჯგუფებს და რა საგნებს ასწავლის ლექტორი გაზედავა?“

ბაზიდან უნდა გამოვიყენოთ სამი ცხრილი: Lector, Jgufi და Lect_Jgufi, რათა კონკრეტული ლექტორისთვის ვიპოვოთ მისი საგნები, ჯგუფები და კურსები (ნახ.8.32).

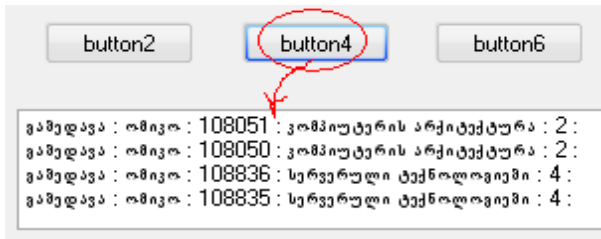


8.46

კოდის ტექსტი მოცემულია 8.10_ლისტინგში, ხოლო შედეგები 8.46 ნახაზზე.

// ლისტინგი_8.10 --- button4 ----

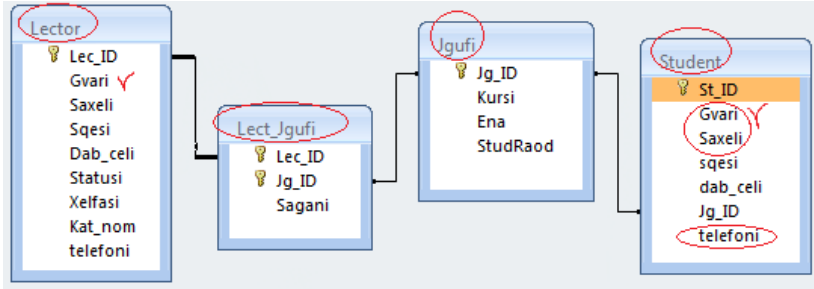
```
private void button4_Click(object sender, EventArgs e)
{
    Amorcheva("SELECT * FROM Lector,Jgufi,Lect_Jgufi " +
        "where Jgufi.Jg_ID = Lect_Jgufi.Jg_ID and "+
        "Lector.Lec_ID = Lect_Jgufi.Lec_ID and "+
        "Lector.Gvari='გაბედავა' " +
        "order by Lect_Jgufi.Sagani ", "Gvari", "Saxeli",
        "Jgufi.Jg_ID", "Sagani", "kursi");
    MessageBox.Show("SELECT * FROM Lector,Jgufi,Lect_Jgufi " +
        "where Jgufi.Jg_ID = Lect_Jgufi.Jg_ID and "+
        "Lector.Lec_ID = Lect_Jgufi.Lec_ID and
        Lector.Gvari='გაბედავა' " +
        "order by Lect_Jgufi.Sagani , 'Gvari',
        'Saxeli', 'Jgufi.Jg_ID', 'Sagani', 'kursi'");
}
```



ნახ. 8.46

მოთხოვნა_5: რომელ სტუდენტებს ასწავლის ლექტორი გაბედავა ? საჭიროა გვარი, სახელი და ჯგუფის ნომერი.

მონაცემთა ბაზიდან ამჯერად დაგვჭირდება ოთხივე ცხრილი (ნახ.8.47). კონკრეტული ლექტორიდან (მარცხენა ნაპირა ცხრილი) უნდა მივიღეთ კონკრეტულ სტუდენტამდე (მარჯვენა ნაპირა ცხრილი). შესაბამისი SQL-მოთხოვნის ტექსტი მოცემულია 8.11_ლისტინგში.

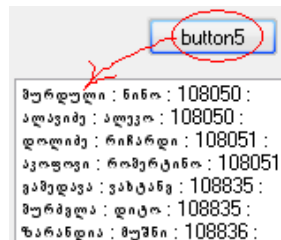


ნახ. 8.47

// ლოსტინგი_8.11 --- button5 ----

```
private void button5_Click(object sender, EventArgs e)
{
    Amorcheva("SELECT * FROM Lector,Jgufi,Lect_Jgufi,Student " +
        "where Jgufi.Jg_ID = Lect_Jgufi.Jg_ID and " +
        "Lector.Lec_ID = Lect_Jgufi.Lec_ID and " +
        "Student.Jg_ID=Jgufi.Jg_ID and Lector.Gvari='გაბედავა' " +
        "order by Jgufi.Jg_ID ", "Student.Gvari",
        "Student.Saxeli", "Student.Jg_ID");
    MessageBox.Show("SELECT * FROM
        Lector,Jgufi,Lect_Jgufi,Student " +
        "where Jgufi.Jg_ID = Lect_Jgufi.Jg_ID and
        Lector.Lec_ID = Lect_Jgufi.Lec_ID and " +
        "Student.Jg_ID=Jgufi.Jg_ID and Lector.Gvari='გაბედავა' " +
        "order by Jgufi.Jg_ID, 'Student.Gvari',
        'Student.Saxeli', 'Student.Jg_ID' ");
}
```

შედეგი მოცემულია 8.48 ნახაზზე.



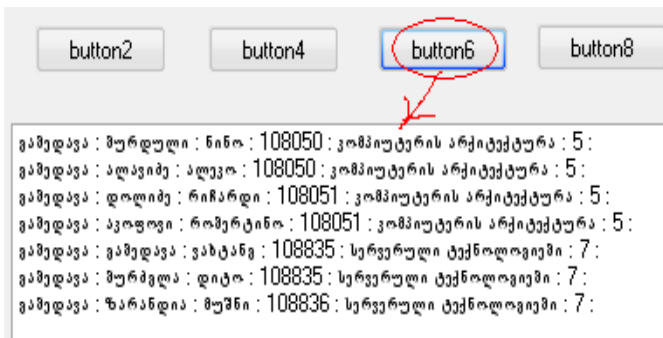
ნახ.8.48

მოთხოვნა_6: ლექტორ გაბედავასთან რომელი ჯგუფის რომელ სტუდენტებთან აქვს ლექციები რომელ საგნებში და რამდენ კრედიტიანებია ეს საგნები ?

// ლისტიנגი_8.12 --- button6 ----

```
private void button6_Click(object sender, EventArgs e)
{
    Amorceva("SELECT * FROM Lector,Lect_Jgufi, Jgufi,Student " +
        "where Lector.Lec_ID=Lect_Jgufi.Lec_ID and "+
            " Jgufi.Jg_ID = Lect_Jgufi.Jg_ID and "+
        "Student.Jg_ID=Jgufi.Jg_ID and Lector.Gvari='გაბედავა' " +
        "order by Jgufi.Jg_ID ", "Lector.Gvari",
        "Student.Gvari", "Student.Saxeli", + "Jgufi.Jg_ID", "Sagani", "ECTS");
    MessageBox.Show("SELECT * FROM Lector,Lect_Jgufi, Jgufi, "+
        "Student " + "where Lector.Lec_ID=Lect_Jgufi.Lec_ID and "+
            " Jgufi.Jg_ID = Lect_Jgufi.Jg_ID and "+
        "Student.Jg_ID=Jgufi.Jg_ID and "+ "Lector.Gvari='გაბედავა' " +
            "order by Jgufi.Jg_ID, 'Lector.Gvari', "+
        'Student.Gvari', 'Student.Saxeli', 'Jgufi.Jg_ID', 'Sagani', 'ECTS' ");
}
```

შდეგები გამოტანილია 8.49 ნახაზზე.



ნახ.8.49

მოთხოვნა_7: „რა საგნები ისწავლება და რამდენ კრედიტიანებია ?“

ეს ინფორმაცია მისაწვდომია ერთი ცხრილიდან Lect_Jgufi. თუ მოთხოვნა ასე დაიწერება:

```
Amorcheva("select Sagani, ECTS from Lect_Jgufi order by  
Sagani", "Sagani", "ECTS");
```

მაშინ მივიღებთ ასეთ შედეგს (ნახ.8.50), რაც არაკორექტულია განმეორებადი სტრიქონების გამო:

```
ვიზუალური დაპროგრამება : 5 :  
ვიზუალური დაპროგრამება : 5 :  
კომპიუტერის არქიტექტურა : 5 :  
კომპიუტერის არქიტექტურა : 5 :  
კომპიუტერის არქიტექტურა : 5 :  
მათემატიკა : 4 :  
მათემატიკა : 4 :  
მენეჯმენტის საფუძვლემი : 4 :  
მენეჯმენტის საფუძვლემი : 4 :  
მონაცემთა შაზემი : 6 :  
მონაცემთა შაზემი : 6 :  
სერვერული ტექნოლოგიემი : 7 :  
სერვერული ტექნოლოგიემი : 7 :
```

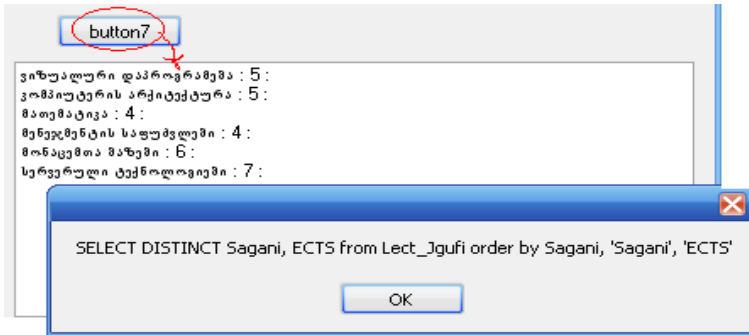
ნახ.8.50

საჭიროა SELECT DISTINCT... კონსტრუქციის გამოყენება, რაც გამორიცხავს განმეორებად სტრიქონებს რელაციურ ცხრილებში. ამგვარად შესაძლებელია სტრიქონთა "სიმრავლის" მიღება. სწორი ტექსტი მოცემულია 8.13_ლისტინგში.

// ლისტინგი_8.13 --- button7 ----

```
private void button7_Click(object sender, EventArgs e)  
{  
    Amorcheva("select distinct Sagani, ECTS from Lect_Jgufi "+  
        " order by Sagani", "Sagani", "ECTS");  
    MessageBox.Show("SELECT DISTINCT Sagani, ECTS "+  
        "from Lect_Jgufi order by Sagani, 'Sagani', 'ECTS' ");  
}
```

8.51 ნახაზზე ასახულია სწორი შედეგები.



ნახ.8.51

მოთხოვნა_8: „რამდენი კრედიტი შეიძინეს სტუდენტებმა დამატებით სემესტრში და რას უდრის ჯამური შემოსავალი თანხაში?“

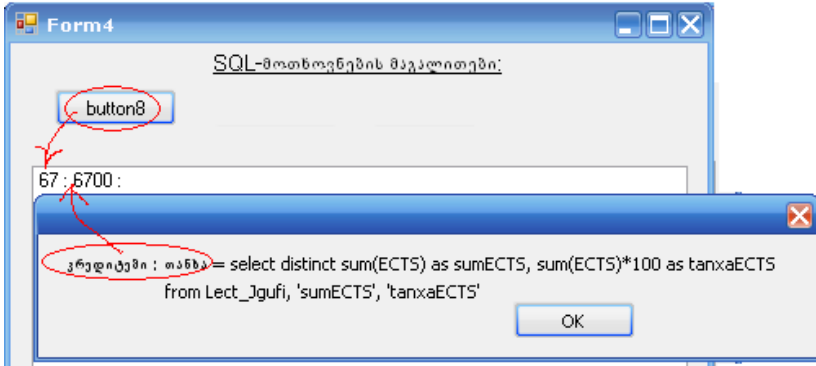
ცხრილში Lect_Jgufi მოთავსებულია ოპერატიული ინფორმაცია მიმდინარე სემესტრის საგნების შესახებ, ანუ, აქედან უნდა მოხდეს ECTS-ატრიბუტის მნიშვნელობათა შეჯამება და მისი გამრალება 1 კრედიტის ღირებულებაზე. სიმარტივისთვის დავეუშვათ, რომ ის 100 ლარია.

8.14 ლისტინგში მოცემულია კოდის ტექსტი შესაბამისი SQL მოთხოვნით.

// ლისტინგი_8.14 --- button7 ----

```
private void button8_Click(object sender, EventArgs e)
{
    Amorcheva("select sum(ECTS) as sumECTS, "+
              "sum(ECTS)*100 as tanxaECTS "+
              "from Lect_Jgufi ", "sumECTS", "tanxaECTS");
    MessageBox.Show(" კრედიტები : თანხა = select "+
                    "sum(ECTS) as sumECTS, sum(ECTS)*100 as tanxaECTS "+
                    "from Lect_Jgufi, 'sumECTS', 'tanxaECTS' ");
}
```

შედეგი მოცემულია 8.52 ნახაზზე.



ნახ.8.52

განხილული პროგრამის დასაწყისი და საერთო ნაწილი ყველა მოთხოვნისა მოცემულია 8.14_ლისტინგში.

// ლისტინგი_8.14 --- Syetem+Contact with DBS+ try...catch ----

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WinADO
{
    public partial class Form4 : Form
    {
        public Form4() { InitializeComponent(); }
        private void Amorcheva(string sqlbefehl,
                                params string[] velebi)
        {
            OleDbConnection con = new OleDbConnection();
            OleDbCommand cmd = new OleDbCommand();
            OleDbDataReader reader;
            int i;
            string striqoni;
```



```

con.ConnectionString = "Provider=Microsoft.ACE.OLEDB.8.0;" +
    "Data Source=C:\\C#2010\\WinADO\\DB_AccessUni.accdb";
cmd.Connection = con;
cmd.CommandText = sqlbefehl;
try
{ // აქ ხდება SQLმოთხოვნის წაკითხვა, ანალიზი ატრიბუტებით
con.Open();
reader = cmd.ExecuteReader();
listBox1.Items.Clear();
while (reader.Read())
{
    striqoni = "";
    for (i = 0; i < velebi.Length; i++)
        striqoni += reader[velebi[i]] + " : ";

    listBox1.Items.Add(striqoni); // ეკრანზე გამოტანა ---
}
reader.Close();
con.Close();
}
catch (Exception ex)
{ // შეცდომის არსებობისას გამოსცემს შეტყობინებას -----
    MessageBox.Show(ex.Message);
}
}
// აქ უერთდება button_Click - მეთოდები ----- //...

```

თავი 9

.NET პლატფორმის კონცეფციის რეალიზაცია

9.1. აპლიკაციის დაპროგრამება რამდენიმე ენის საფუძველზე .dll ფაილების შექმნით

განხილულია ობიექტ-ორიენტირებული დაპროგრამების ენების: C++, Visual Basic და C# გამოყენებით ერთი პროგრამული პროექტის აგების ტექნოლოგიის შესწავლა .NET პლატფორმაზე. ამ ენებზე იქმნება .dll ფაილები და ხორციელდება მათი ერთად მუშაობა.

ილუსტრირებულია .NET ტექნოლოგიის ერთ-ერთი ძირითადი პრინციპი, რომ პროგრამული აპლიკაციის დამუშავება შესაძლებელია დეველოპერების გუნდში სხვადასხვა პროგრამული ენის მცოდნე სპეციალისტების მიერ, კერძოდ C++, Visual Basic და C# ენების საფუძველზე. პროექტის აგებისას გათვალისწინებულ უნდა იქნას საერთო მოთხოვნები ღია ცვლადებზე, მეთოდებსა და თვისებებზე.

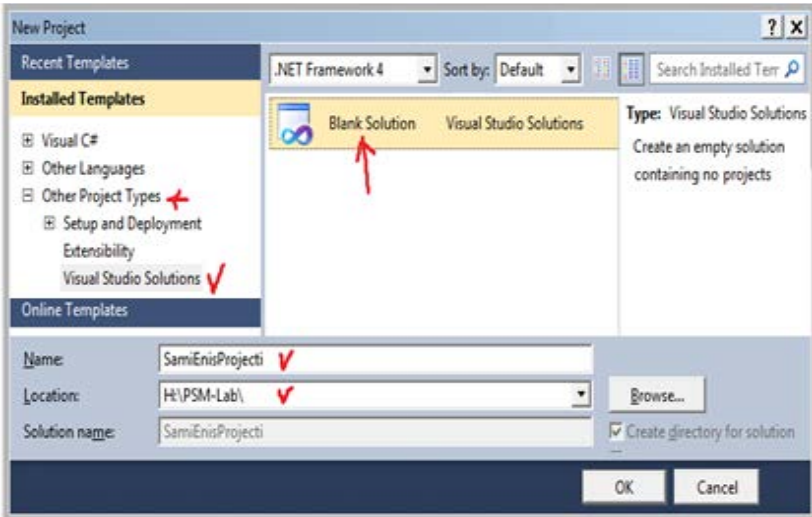
სამუშაო გეგმა:

- შეიქმნას C++ -ის საბაზო კლასი;
- შეიქმნას Visual Basic.NET კლასი, რომელიც იქნება C++ - კლასის მემკვიდრე;
- შეიქმნას C# კლასი, რომელიც კონსოლის აპლიკაციაში შექმნის Visual Basic.NET კლასის ეგზემპლარს და გამოიძახებს მის მეთოდს.

განვახორციელოთ პროექტის პროგრამული რეალიზაცია Visual Studio .NET Framework 4.0/4.5 სამუშაო გარემოში.

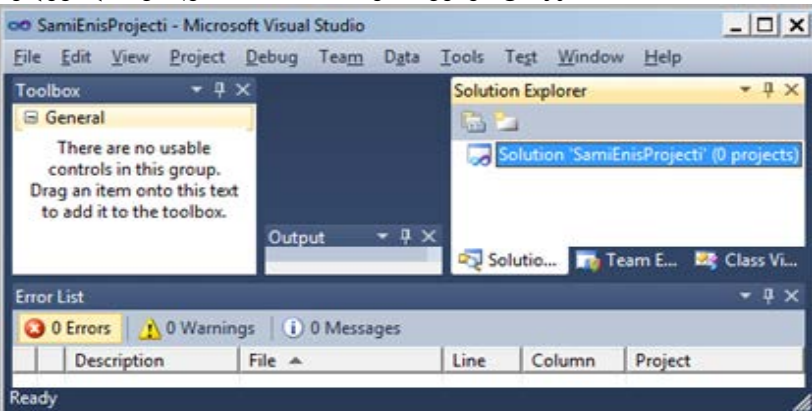
შევარჩიოთ ახალი პროექტის განთავსების ადგილი და მენიუდან გამოვიძახოთ ახალი ცარიელი პროექტის შექმნის პროგრამა (ნახ.9.1):

(New->Project და Visual Studio Solution-> Blank Solution)



ნახ.9.1

პროექტის სახელი (Name: SamiEnisProjecti) და მისი შენახვის ადგილი (Location) მივუთითოთ ჩვენი სურვილით. OK-ის შემდეგ Blank Solution შაბლონის დახმარებით შეიქმნება აპლიკაცია, რომელიც *ნეიტრალური* იქნება დაპროგრამების ენებისადმი. შედეგად მივიღებთ 9.2 ნახაზზე ნაჩვენებ ფანჯარას.

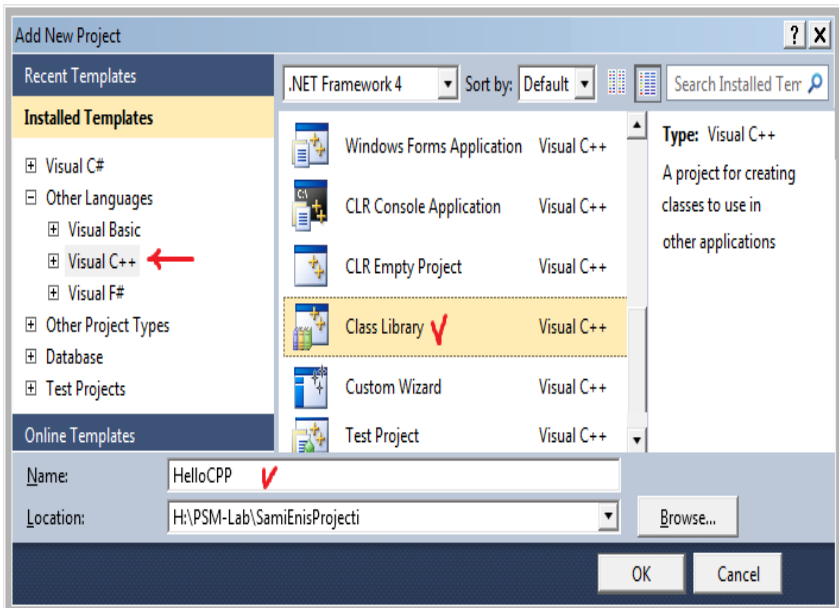


ნახ.9.2

9.1.1. კლასის შექმნა ახალ პროექტში C++.NET ენაზე

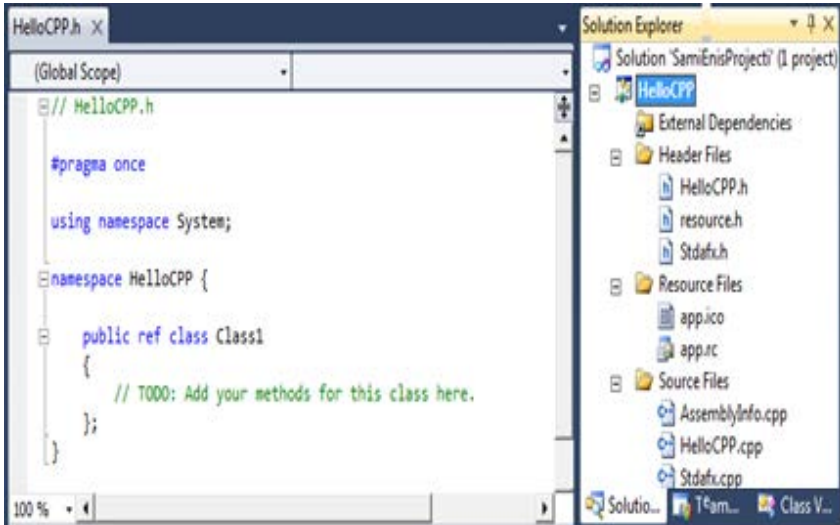
ცარიელ პროექტს (გადაწყვეტა - Solution 'SamiEnisProject') დავამატოთ ახალი (ქვე)პროექტი მასზე მარჯვენა ღილაკის დაწკაპუნებით და შემდეგ Add -> New Project არჩევით. 9.3 ნახაზზე ნაჩვენებია თუ როგორ მიეთითება VisualC++ ენა, Class Library და ქვეპროექტის სახელი Name: HelloCPP, შემდეგ OK.

Solution Explorer -ში გამოჩნდება შედეგი (ნახ.9.4). გავხსნათ Hello.CPP.h ფაილი რედაქტირებისთვის. ახლადშექმნილი კლასის Class1 მაგივრად ჩავწეროთ სახელი HelloCPP.



ნახ.9.3

HelloCPP.h კოდი მოცემულია 9.1 ლისტინგში.



ნახ.9.4

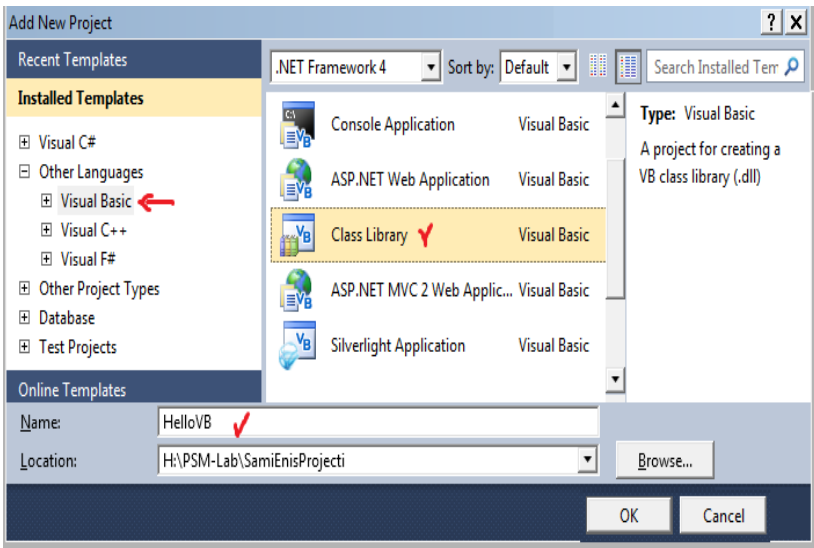
```
// ---- ლისტინგი_9.1 -- შეცვლილი HelloCPP.h ----
#pragma once
using namespace System;
namespace HelloCPP
{
    public ref class HelloCPP
    {
        // --- აქ ჩაემატება კლასის მეთოდი ----
    public:
        virtual void Hello()
        {
            Console::WriteLine("Mogesalmebit C++ enis
                                garemodan da !\n
                                viZaxeb Visual Basics !\n\n");
        }
    };
}
```

HelloCPP::Hello() მეთოდი გამოიყენებს .NET Framework კლასების ბიბლიოთეკიდან System::Console::WriteLine() ფუნქციას, რათა კონსოლზე გამოიტანოს მისალმება C++ კოდიდან.

პროექტი გავუშვით სინტაქსური შეცდომების გასასწორებლად, თუ ასეთი იქნება. იგი ჯერ არ უნდა ავამუშავოთ შესრულებაზე.

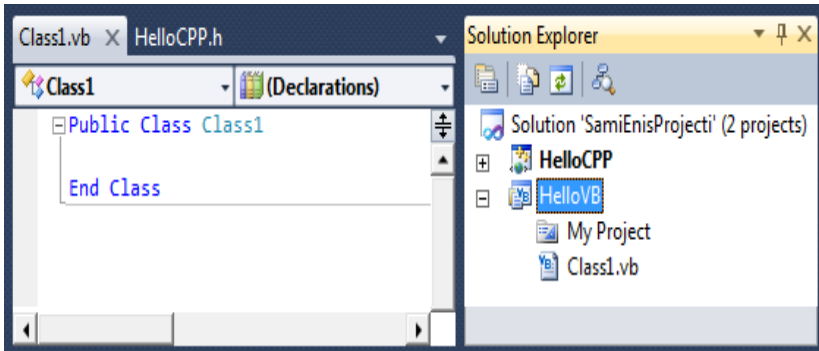
9.1.2. კლასის შექმნა ახალ პროექტში Visual Basic .NET ენაზე

დავამატოთ Solution-ში ახალი პროექტი და მივუთითოთ, რომ იგი შეიქმნას Visual Basic .NET -ში სახელით Name: HelloVB (ნახ.9.5).



ნახ.9.5

მივიღებთ ასეთ შედეგს (ნახ.9.6). ახალი პროექტი HelloVB დაემატება Solution Explorer პანელზე თავისი შემადგენელი ფაილებით.



ნახ.9.6

ახლა უკვე გვაქვს HelloCPP და HelloVB ორი პროექტი.

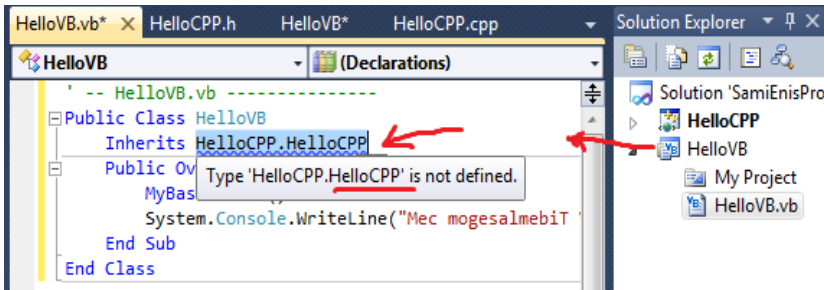
მეორეში ავირჩიოთ Class1.vb ფაილი და შევუცვალოთ სახელი შინაარსის გათვალისწინებით, მაგალითად, HelloVB.vb, შემდეგ გავსნათ იგი და ჩავწეროთ ჩვენთვის საჭირო ტექსტი (ლისტინგი_9.2).

'--ლისტინგი_9.2 --- HelloVB.vb -----'

```
Public Class HelloVB
    Inherits HelloCPP.HelloCPP
    Public Overrides Sub Hello()
        MyBase.Hello()
        System.Console.WriteLine("Mec mogesalmebiT
                                Visual Basic.NET-idan !!")
    End Sub
End Class
```

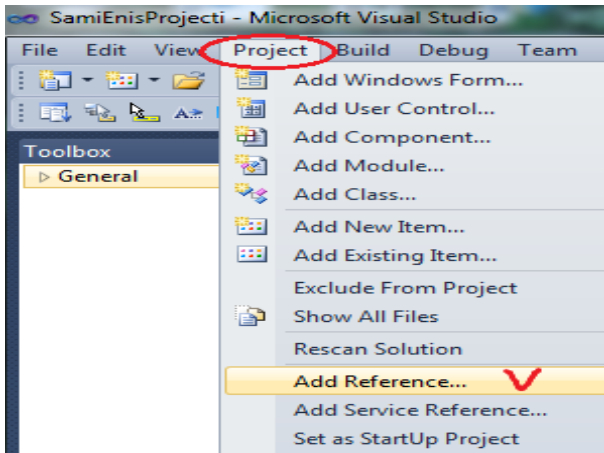
ეს კოდი განსაზღვრავს HelloVB კლასს, რომელიც მემკვიდრეობით იქნა მიღებული C++ ის მმართველი კლასიდან HelloCPP. ამგვარად, HelloVB კლასი ჩაანაცვლებს (Overrides) მშობელი HelloCPP კლასის ვირტუალურ Hello() მეთოდს, ოღონდ ჯერ გამოიძახებს Hello() მეთოდის ვერსიას მშობელი კლასიდან HelloCPP, შემდეგ კი გამოყავს ეკრანზე თავისი მისალმება.

საყურადღებოა, რომ კოდის რედაქტორში საბაზო კლასის სახელი (HelloCPP.HelloCPP) და ჩასანაცვლებელი მეთოდის სახელი Hello() ტალღისებური ხაზითაა. ეს ნიშნავს, რომ კოდის რედაქტორი ამ სახელებს ვერ ხედავს და წინასწარ იძლევა სინტაქსურ შეცდომას. თუ კურსორს დავაყენებთ ამ ფრაგმენტზე, იგი მოგვცემს შეცდომის მნიშვნელობას (ნახ.9.7).



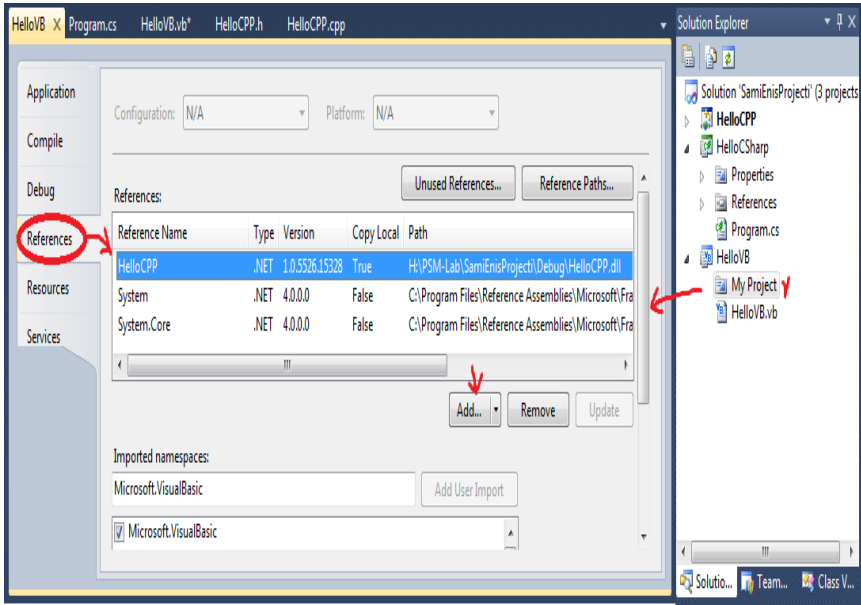
ნახ.9.7

აუცილებელია Visual Basic-ის კომპილატორს მიეთითოს თუ სად იმყოფება ეს ნაკრები (assembly), რომელიც განსაზღვრავს მოცემულ ტიპს. ამისათვის მთავარი მენიუს Project-> Add Reference->Projects-დან (ნახ.9.8) ავირჩევთ HelloCPP-ს და OK.



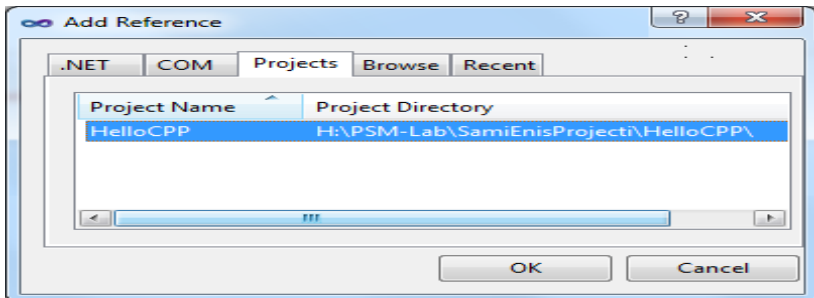
ნახ.9.8

Solution-ში HelloVB-ს MyProject-ის ამოქმედებით გაიხნება 9.9 ნახაზის მსგავსი ფანჯარა, რომელშიც References-ზე გადართვით გაჩნდება შიგთავსი. ვირჩევთ HelloCPP და Add-ით ვადასტურებთ.



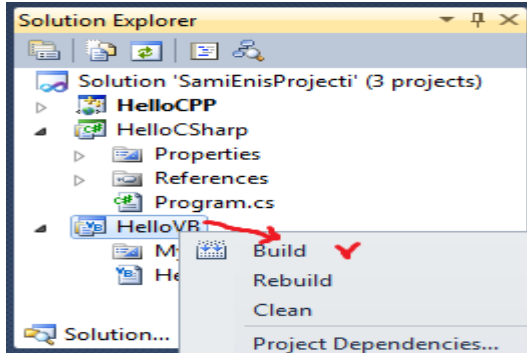
ნახ.9.9

ამის შემდეგ გამოჩნდება Add Reference (ნახ.9.10) და OK ღილაკით ვასრულებთ CPP-კლასთან დაკავშირების პროცესს.



ნახ.9.10

ამის შემდეგ საჭიროა HelloVB პროექტზე მარჯვენა ღილაკით გამოვიტანოთ 9.11 კონტექსტური მენიუ და ავირჩიოთ Build (ეს პროექტი ტრანსლატორით ამუშავდება და სინტაქსური გამართვის შედეგს მოგვცემს).

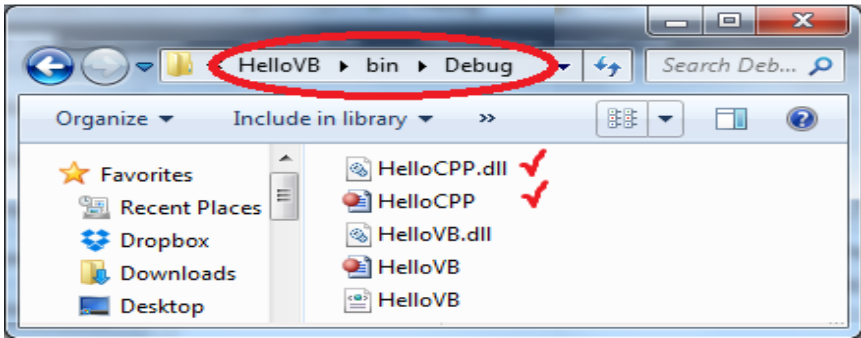


ნახ.9.11

როგორც წესი, თუ შეცდომები არაა HelloVB პროექტში, მაშინ მის შესაბამის ფოლდერის bin->Debug -ში უნდა გამოჩნდეს დაკავშირებული Hello.CPP.dll კლასის ფაილი (ნახ.9.12).

ამგვარად, Hello.CPP პროექტი მიუერთდება გადაწვეტას (Solution-ში) და გამოჩნდება მომავალში HelloVB-ში, როგორც ზემოთ იქნა ნაჩვენები. ამასთანავე, 9.7 ნახაზზე ნაჩვენები „ქვეშეგახაზული“ შეცდომების აღნიშვნები HelloVB.vb პროგრამის ტექსტიდან გაქრა ! პროგრამის საბოლოო ტექსტი მოცემულია ლისტინგში:

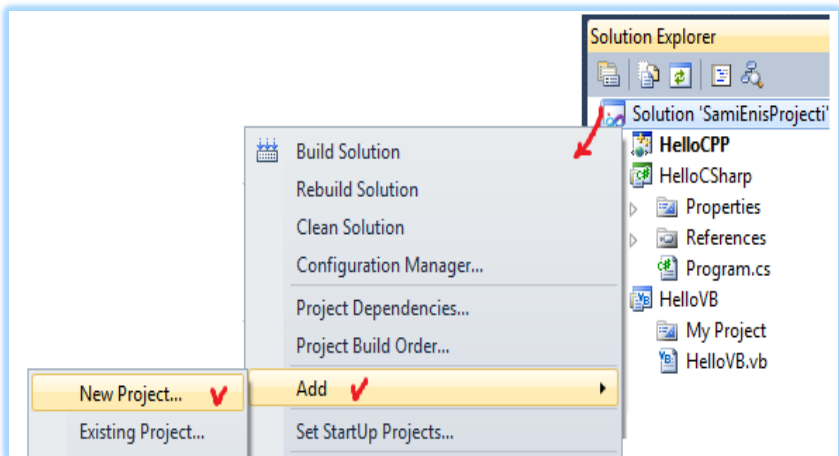
```
'-- HelloVB.vb -----
Public Class HelloVB
    Inherits HelloCPP.HelloCPP
    Public Overrides Sub Hello()
        MyBase.Hello()
        System.Console.WriteLine("Mec mogesalmebiT Visual Basic.NET !!!")
    End Sub
End Class
```



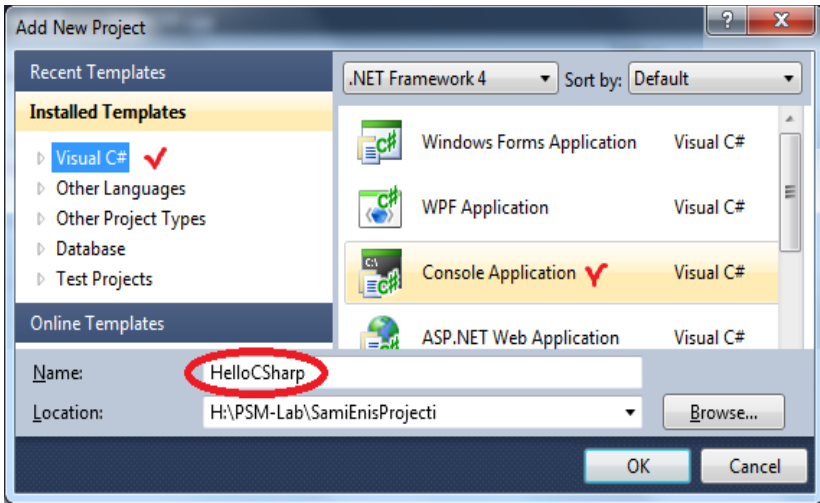
ნახ.9.12

9.2. სასტარტო პროექტის დამატება Solution-ში C#.NET ენაზე

დასკვნით ფაზაში ჩვენი გადაწყვეტისათვის (Solution-ში) SamiEnisProjectი უნდა შეექმნათ მესამე, მთავარი სასტარტო პროექტი, რომელიც აგებული იქნება C#.NET ენაზე Console Application შაბლონის სახით. დავარქვათ მას HelloCSharp. ამგვარად, ვამატებთ პროექტს (ნახ.9.13-ა,ბ) HelloCSharp სახელით.

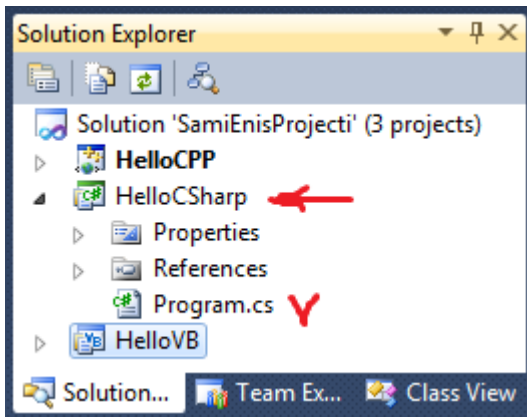


ნახ.9.13-ა



ნახ.9.13-ბ

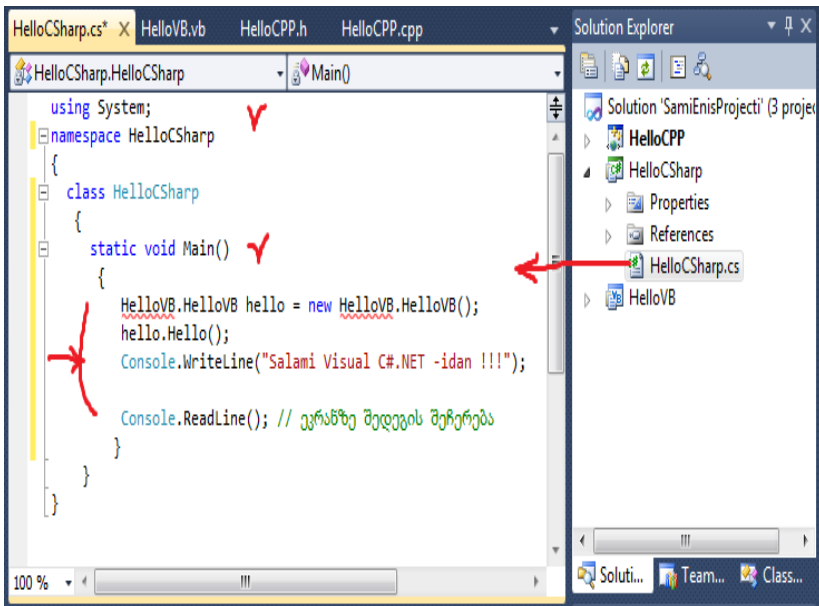
OK ღილაკის ამოქმედების შემდეგ მივიღებთ Solution-ში ახალ, HelloCSharp პროექტს თავისი შემადგენელი კომპონენტებით (ნახ.9.14).



ნახ.9.14

გადავარქვით სახელი Program.cs ფაილს ჩვენი პროექტის შინაარსის შესაბამისად: HelloCSharp.cs.

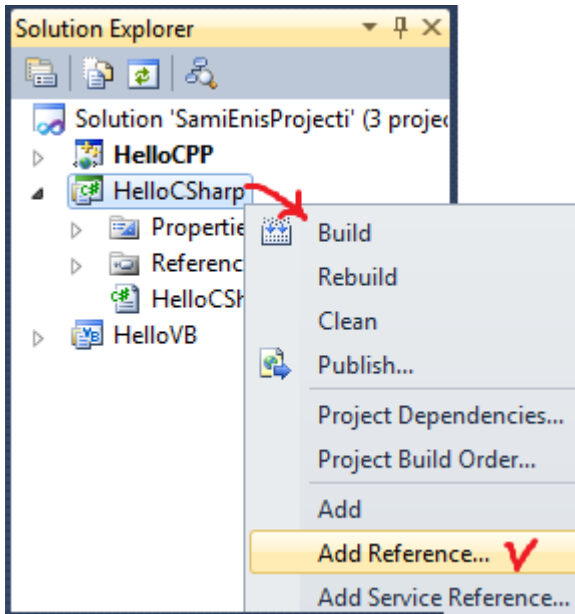
გამოვიტანოთ ეს ფაილი რედაქტირების ფანჯარაში და ჩავამატოთ Main() -ში შესაბამისი სტრიქონები. ასევე შეიძლება Main() -ის არგუმენტების წაშლა, აქ არ გვჭირდება. რედაქტირების შემდეგ ტექსტი მოცემულია 9.15. ნახაზზე.



ნახ.9.15

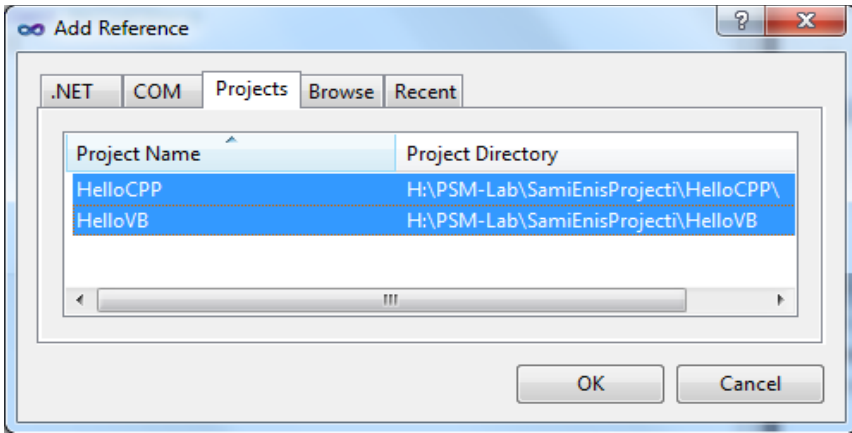
პროგრამაში Main() სტატიკური მეთოდი არის აპლიკაციაში შესვლის წერტილი. ეს მეთოდი ქმნის HelloVB კლასის ახალ ობიექტს hello და მისთვის იძახებს Hello() მეთოდს, რომელშიც ინახება ორი შეტყობინება. ერთი CPP.NET-იდან, მეორე VB.BET-დან, რომლებიც ადრე მოვამზადეთ. მესამე შეტყობინებას თვით C#.NET გამოიტანს, დამშვიდობებასთან ერთად.

ახლა საჭიროა HelloCSharp პროექტში ჩავამატოთ მიმთითებლები (კავშირები) HelloCPP და HelloVB პროექტებზე. მაშინ 9.15 ნახაზზე ნაჩვენები შეცდომები (ქვემხაზგასმული HelloVB) გასწორდება. ამისათვის ვირჩევთ Add Reference...-ს (ნახ.9.16).

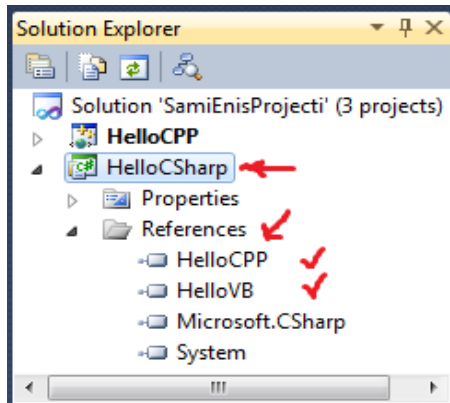


ნახ.9.16

Add Reference ფანჯარაში გადავრთოთ Project-ზე, მოვნიშნოთ ორივე, HelloCPP და HelloVB პროექტები და დავაწკიწოთ OK (ნახ.9.17). შედეგად გაქრება შეცდომები, ანუ 9.15 ნახაზზე „ხაზები“, რაც მიუთითებს იმაზე, რომ კავშირი პროექტებს შორის კარგად შესრულდა. ეს შედეგი აისახება Solution-ფანჯარაში HelloCSharp პროექტის References –ში (ნახ.9.18).

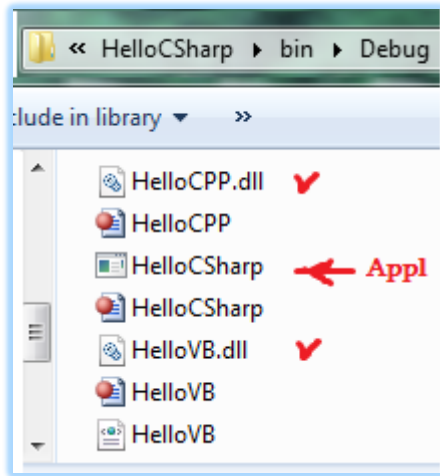


ნახ.9.17



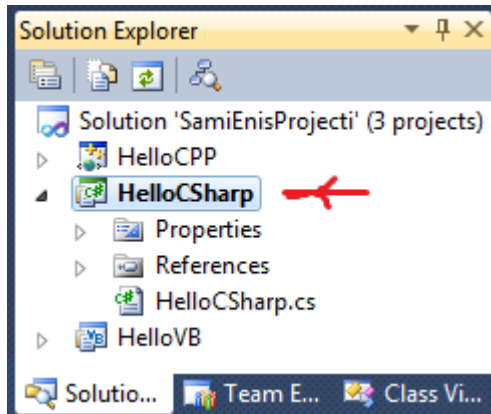
ნახ.9.18

ავამუშავოთ HelloCSharp პროექტი მარჯვენა ღილაკით და build-ით, რათა სინტაქსურად დამუშავდეს იგი. შეცდომების არარსებობის შემთხვევაში მოხდება HelloCPP.dll და HelloVB.dll ფაილების განთავსება HelloCSharp პროექტის bin->Debug ფოლდერში (ნახ.9.19). აქვეა HelloCSharp.exe აპლიკაციის ფაილიც.

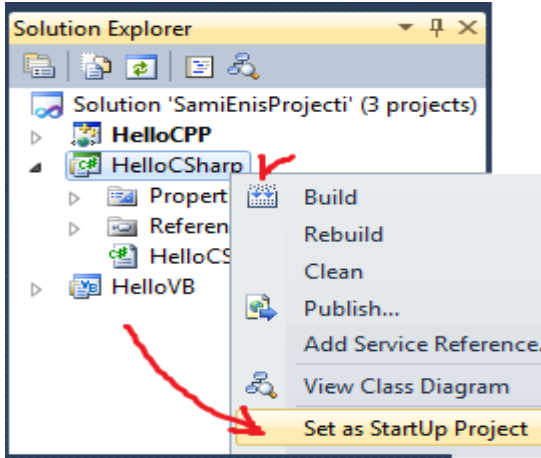


ნახ.9.19

Solution-ში HelloCSharp პროექტი გადავაკეთოთ სასტარტო ფაილად (ნახ.9.20-ა,ბ).

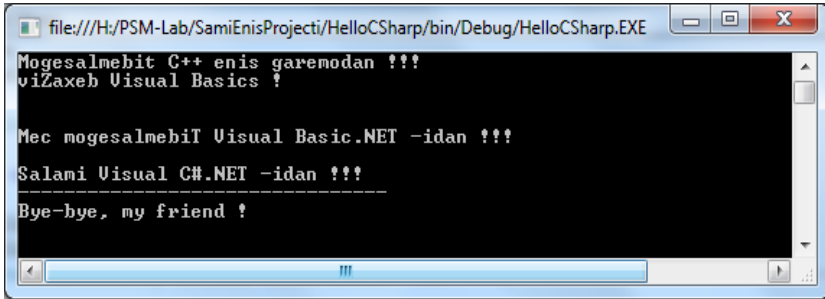


ნახ.9.20-ა



ნახ.9.20-ბ

ბოლოს, ავამუშავოთ აპლიკაცია და მივიღებთ შედეგს (ნახ.9.21).



ნახ.9.21

რეზიუმე: აპლიკაციის სხვადასხვა პროექტები დამუშავდა დაპროგრამების სხვადასხვა ენებზე, რომლებიც გამოიყენება .NET გარემოში. მრავალჯერადი გამოყენების ფაილები შემუშავებულ იქნა კლასების დახმარებით და რეალიზებულია დინამიკური .dll - ფაილების სახით.

თავი 10

Workflow Foundation ტექნოლოგია .NET პლატფორმაზე

ვიზუალური დაპროგრამების სფეროში კორპორაცია „მაიკროსოფტის“ პროდუქტი WF (Workflow Foundation) „ახალი ტექნოლოგიაა“, რომელიც .NET პლატფორმაზე, WPF და WCF ტექნოლოგიებთან ერთად, სრულიად ახალი პარადიგმის მატარებელი ტექნოლოგიების სახელით დამკვიდრდა ბიზნეს-პროცესებზე (Workflows) ბაზირებული აპლიკაციების ასაგებად [1].

Workflow ტექნოლოგიის საშუალებით შესაძლებელია სამი ტიპის პროცესის აღწერა:

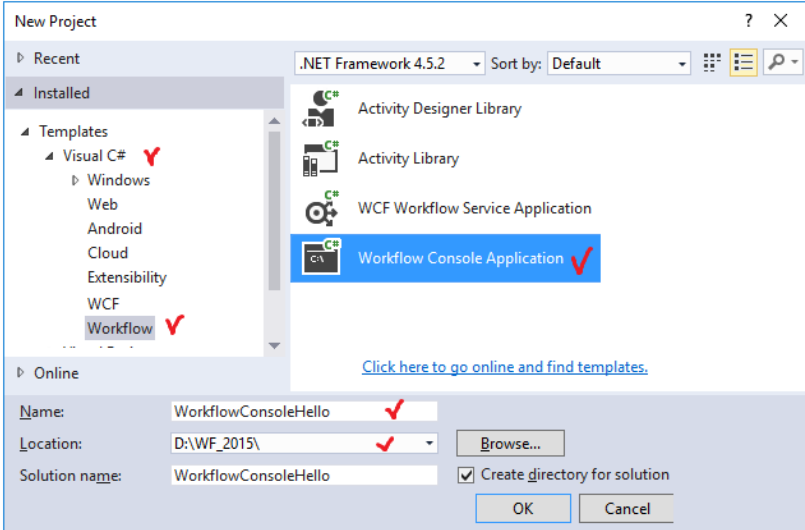
- მიმდევრობითი პროცესი (Sequential Workflow) — ერთი ბიჯიდან მეორეზე გადასვლა უკან დაბრუნების გარეშე;
- წესებით მართვადი პროცესი (Rules-driven Workflow) — ესაა მიმდევრობითი პროცესის კერძო შემთხვევა, რომელშიც გადასვლა შემდგომ ბიჯზე განისაზღვრება წესების ერთობლიობით;
- სასრული ავტომატი (State-Machine Workflow) — გადასვლა ერთი მდგომარეობიდან სხვა მდგომარეობაზე, ასევე შესაძლებელია ნებისმიერი უკან დაბრუნება წინა მდგომარეობებში.

ჩვენი მიზანია გავაცნოთ მკითხველს ბიზნეს-პროცესების (Workflow) ვიზუალური დაპროგრამების შესაძლებლობები და შესაბამისი აპლიკაციების პროექტების აგება Visual Studio.NET პლატფორმაზე.

10.1. მარტივი ბიზნესპროცესის (Workflow-ის) აგება

განვიხილოთ მარტივი სამუშაო პროცესის (workflow-ის) შექმნა Visual Studio.NET გარემოში. შევარჩიოთ ახალი პროექტის სახელი (მაგალითად, WorkflowConsoleHello), მისი შენახვის ადგილი (C:\WF\). აგრეთვე ავირჩიოთ Template-ში Visual C# და

Workflow, ხოლო შუა ფანჯრიდან Workflow Console Application (ნახ.10.1).

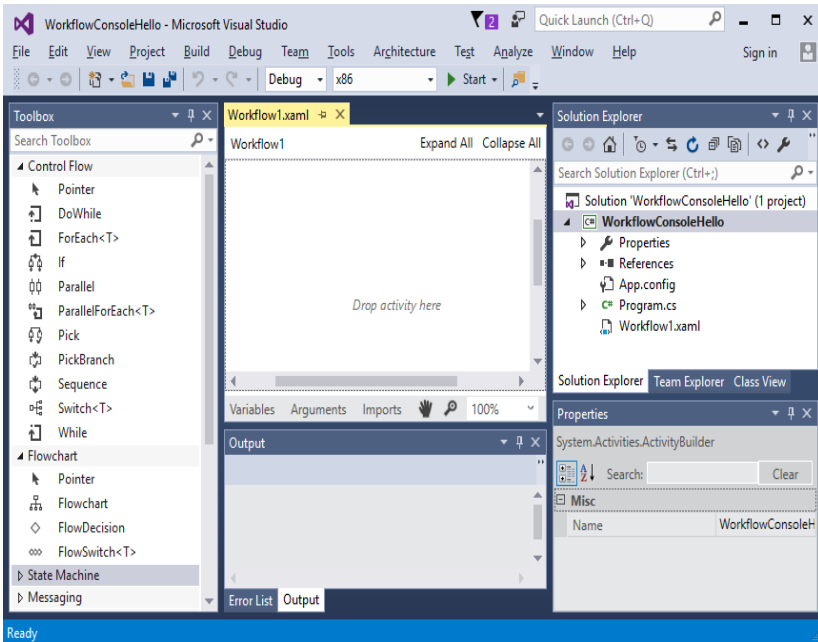


ნახ.10.1. ახალი workflow პროექტის შექმნა

შაბლონი (Template) აგენერირებს Program.cs ფაილს, რომელიც რეალიზებას უკეთებს კონსოლურ აპლიკაციას (დანართს). იგი ასევე აგენერირებს Workflow1.xaml ფაილს, რომელიც განსაზღვრავს ქმედებას (აქტიურობას) სამუშაო პროცესში (workflow-ში). XAML ენა გამოიყენება პროგრამული ელემენტების გამოსაცხადებლად (როგორც WPF აპლიკაციაში). ოღონდაც ლებელის, ტექსტოქსის და ბადის ნაცვლად ეს ფაილი შეიცავს წარმოებული ელემენტების აქტიურობებს ჩვენს მიერ განსაზღვრულ სამუშაო პროცესში. Visual Studio იძლევა დიზაინერისთვის ქმედებების (აქტიურობების) გრაფიკულად ნახვის და რედაქტირების საშუალებას.

10.2 ნახაზზე ნაჩვენებია Visual Studio-ის ინტეგრირებული დამუშავების გარემო (IDE).

ვიზუალური დაპროგრამება (C#.NET & Workflow Foundation NET)



ნახ.10.2. Visual Studio-ს სტანდარტული IDE გარემო

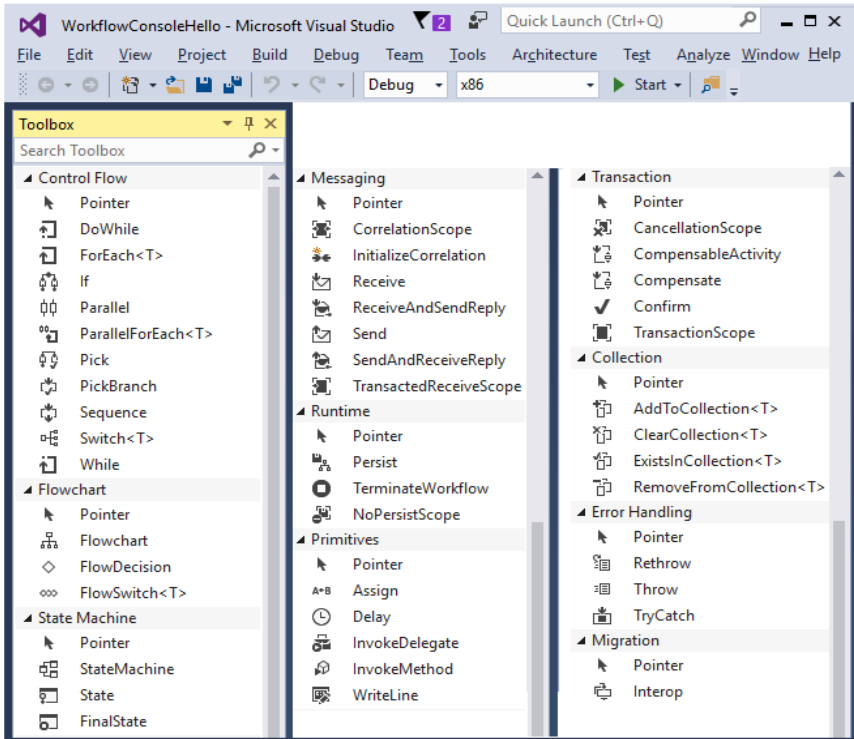
მარცხნივ მოთავსებულია ინსტრუმენტების პანელი, რომელიც მოიცავს ჩადგმულ და მომხმარებლის აქტიურობებს, რომელთა გაფართოება შესაძლებელია სხვა ქმედებებითაც (ნახ.10.3).

Solution Explorer და თვისებათა ფანჯარა მარჯვნივაა მოთავსებული (ნახ.10.4).

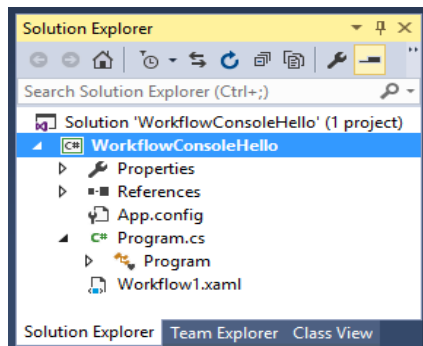
ქვემოთ ფანჯარაში გამოიტანება შეცდომების შეტყობინებები, შუალედური შედეგები და სხვა ინფორმაცია.

WF 4.5 დიზაინერი მოთავსებულია შუაში. ქვედა მარჯვენა კუთხეში არის მასშტაბირების კომპოზოქსი, სურათის გამადიდებელი და აქტიურობის (ქმედების) მოსაძებნი ღილაკები.

ვიზუალური დაპროგრამება (C#.NET & Workflow Foundation NET)



ნახ.10.3. Workflow-ის ინსტრუმენტების პანელი



ნახ.10.4. Workflow-ის Solution Explorer ფანჯარა

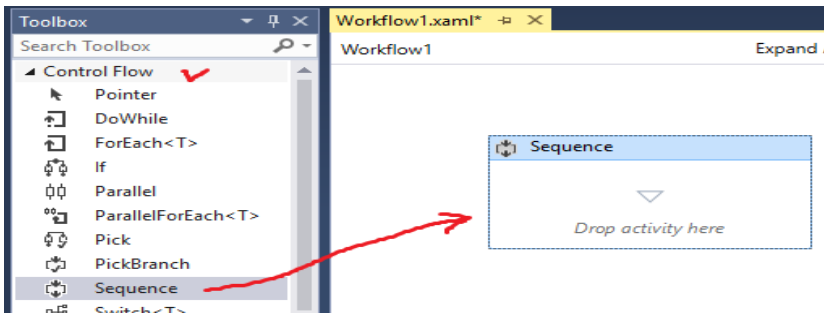
ქვედა მარცხენა კუთხეში სამი ელემენტი: ცვლადების, არგუმენტების და იმპორტული ანაწყოების. თუ მუშა პროცესი კლასია, მაშინ ცვლადები იქნება კლასის წევრები. მათი ნახვა შეიძლება გახსნით (ნახ.10.5).



ნახ.10.5. სამუშაო პროცესის ცვლადების ნახვა

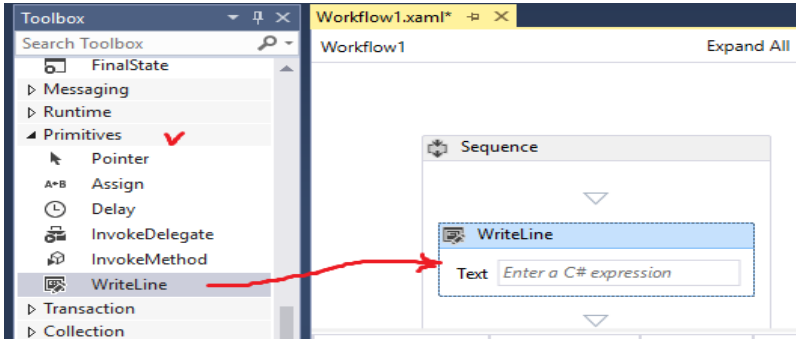
➤ workflow-ის (სამუშაო პროცესის) დაპროექტება

თავიდან სამუშაო ნაკადის კონსტრუქტორი ცარიელია. ინსტრუმენტების პანელიდან საჭირო აქტიურობა გადაიტანება დიზაინერის გარემოში, რითაც განისაზღვრება სამუშაო პროცესის ყოფაქცევა. ჩვენი პროექტი თავიდან უნდა ასახავდეს მისალმებას „Hello, World !“, შემდეგ კი დაემატება სხვა პროცედურები. გადმოვიტანოთ Sequence ქმედება დიზაინერზე (ნახ.10.6).



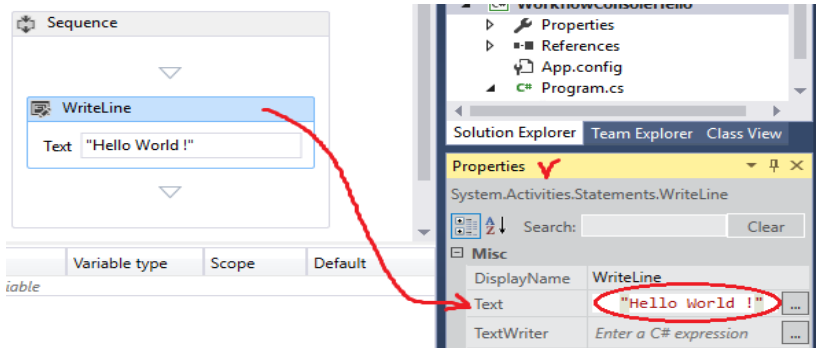
ნახ.10.6. Sequence ქმედების გადმოტანა

შემდეგ კი – WriteLine ქმედება ამ Sequence-ზე. სქემას ექნება 10.7 ნახაზზე ნაჩვენები სახე.



ნახ.10.7. WriteLine ქმედების დამატება Sequence-ში

WriteLine-ის Properties-ში ტექსტის შეტანა ნაჩვენებია 10.8 ნახაზზე.



ნახ.10.8. Text –ის მნიშვნელობის შეტანა Properties-ში

სისტემის ამუშავების შემდეგ საბოლოო შედეგი მოცემულია 10.9 ნახაზზე.



ნახ.10.9. კონსოლზე გამოსული შედეგი

გავხსნათ პროგრამა Program.cs, რომელიც ამუშავებს კონსოლის აპლიკაციას (ლისტინგი_10.1):

```
// -- ლისტინგი_10.1 ----
```

```
using System;
using System.Linq;
using System.Activities;
using System.Activities.Statements;

namespace WorkflowConsoleHello
{
    class Program
    {
        static void Main(string[] args)
        {
            WorkflowInvoker.Invoke(new Workflow1());
            // Console.WriteLine("Press ENTER to exit");
            // Console.ReadLine();
        }
    }
}
```

პროგრამაში ხელითაა ჩამატებული ბოლო ორი სტრიქონი (კომენტარი მოხსენით), რათა შესაძლებელი იყოს შედეგის ნახვა კონსოლზე.

10.2. პროცედურული ელემენტები

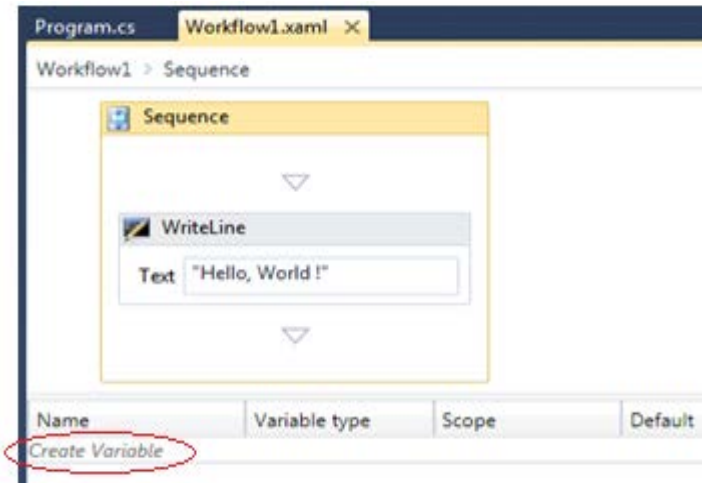
განვიხილოთ Workflow-ის ინსტრუმენტების პანელის ძირითადი პროცედურული ელემენტების გამოყენების საკითხი.

WF-ს აქვს პროცედურული ელემენტები, როგორცაა, მაგალითად, If, While, Assign, Sequence და სხვა. მათი

ფუნქციონირების სადემონსტრაციოდ გამოვიყენოთ ზემოთ აგებული მისალმების კოდი. წარმოვადგინოთ ძველებური საათის მექანიზმი, რომელიც ყოველ საათზე გამოსცემს ზარს [4,13]. გავხსნათ Workflow1.xaml ფაილი.

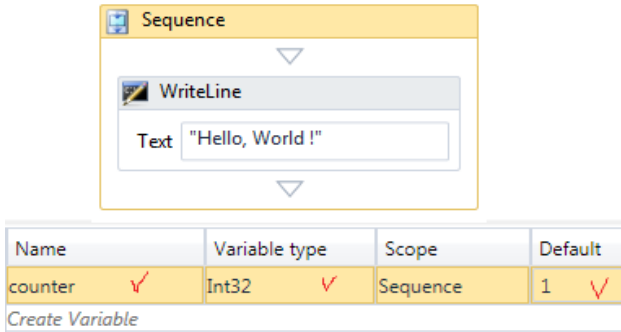
➤ **ცვლადების გამოყენება**

WF-ში უნდა გამოვაცხადოთ ყველა ცვლადი, რომელიც გამოიყენება მუშა ელემენტებში. ჩვენ შემთხვევაში საჭირო იქნება ორი ცვლადი: ერთი – ზარების რაოდენობისათვის, მეორე – მთვლელისათვის, რომელიც დაითვლის თუ რამდენი ზარი იქნა ამოქმედებული აქამდე. დავაჭიროთ ღილაკს Variables. თუ მასში არაა ელემენტები, ე.ი. არ მომხდარა მათი გამოცხადება. ახლა დავდგეთ მთავარ ქმედებაზე – Sequence, ცვლადების ფანჯარას ექნება 10.8 ნახაზზე ნაჩვენები სახე.



ნახ.10.8

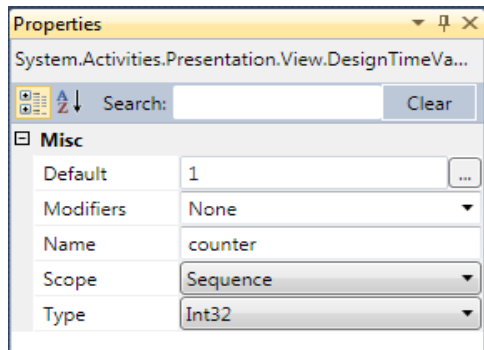
დავკლიკოთ *Create Variable* და შევიტანოთ ცვლადის სახელი, ტიპი და მნიშვნელობა (ნახ.10.9).



ნახ.10.9

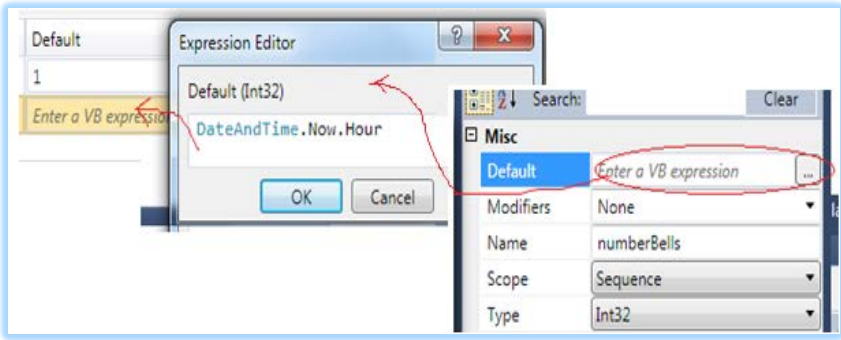
ცვლადი counter ხილვადია როგორც Sequence ქმედების, ისე მისი შვილობილი ელემენტებისთვის (მაგალითად, WriteLine). ცვლადის მნიშვნელობა შევითანეთ 1.

10.10 ნახაზზე ნაჩვენებია ცვლადების სტრიქონის Properties-ის ფანჯარა. აქაც შესაძლებელია ცვლადების მნიშვნელობათა შეტანა, მაგალითად, Default-ში „1“.



ნახ.10.10

მეორე სტრიქონში, შეიტანება ზარების რაოდენობის ცვლადის მნიშვნელობა, რომელიც გამოიყენებს „Enter a VB expression“-ს (ნახ.10.11).



ნახ.10.11

საბოლოო შედეგს ცვლადებისათვის ექნება 10.12 ნახაზზე ნაჩვენები სახე.

Name	Variable type	Scope	Default
counter ✓	Int32	Sequence	1 ✓
numberBells ✓	Int32	Sequence	DateTime.Now.Hour ✓
<i>Create Variable</i>			

Variables Arguments Imports

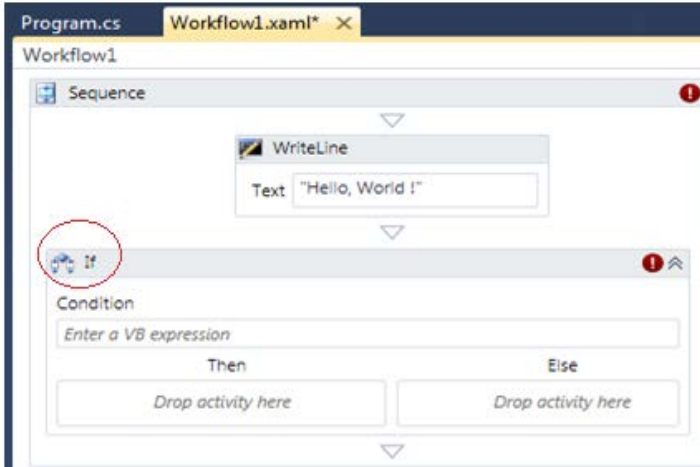
ნახ.10.12

➤ If - განშტოების პროცედურა

DateTime კლასის Hour-წევრი აბრუნებს დროის მნიშვნელობას 24-საათიანი ფორმატით. მაგალითად, 2 PM-თვის ის დააბრუნებს 14-ს. ამიტომაც ჩვენ უნდა ავაწყოთ სისტემა ისე, რომ ზარი იყოს 2-ჯერ და არა 14-ჯერ. ამისთვის კოდში უნდა ჩაიწეროს შემდეგი:

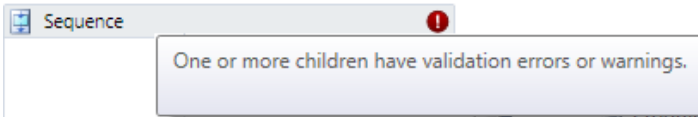
```
if (numberBells > 12)
    numberBells -= 12;
```

ამ მიზნით გამოიყენება if და Assign ქმედებები. გადმოვიტანოთ ინსტრუმენტების პანელიდან if აქტიურობა Hello აქტიურობის ოდნავ ქვემოთ. დიაგრამა მოცემულია 10.13 ნახაზზე.



ნახ.10.13. if ქმედების დამატება

შენიშვნა: ნახაზზე წითელი ძახილის ნიშნით მიეთითება, რომ არის შეცდომა/გაფრთხილება. მაუსის კურსორის მიტანით ჩნდება ტექსტი (ნახ.10.14, 10.15).



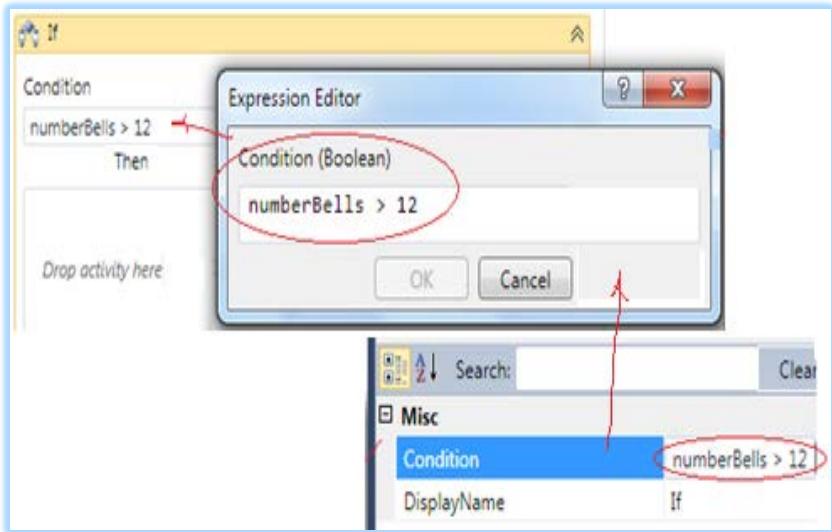
ნახ.10.14. გაფრთხილება Sequence-ში, რომ შვილობილს აქვს შეცდომა



ნახ.10.15. გაფრთხილება if-ში, რომ ქმედებას არ აქვს მითითებული პირობა

თვისებების ფანჯარაში შევცვალეთ DisplayName-მნიშვნელობა Adjust for PM-ით. if ქმედება შედგება სამი ელემენტისგან.

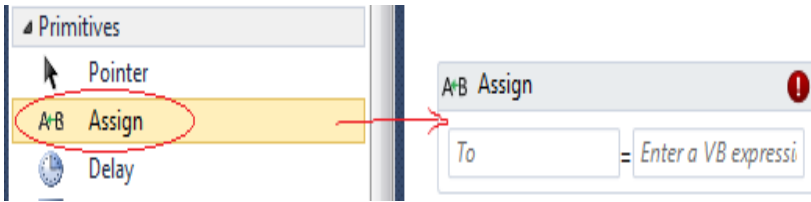
პირობა Condition განსაზღვრავს ლოგიკას, რომელიც შეფასდება. ეს იქნება ლოგიკური მნიშვნელობა („ჭეშმარიტი“ ან „მცდარი“). იგი შეიცავს ქმედებებს, რომლებიც სრულდება, როცა პირობა „ჭეშმარიტია“, ან სხვა ქმედებებს, როცა პირობა „მცდარია“. მხოლოდ ერთია აუცილებელი. შევიტანოთ პირობა `numberBells > 12` (ნახ.10.16).



ნახ.10.16. Condition პირობის შეტანა

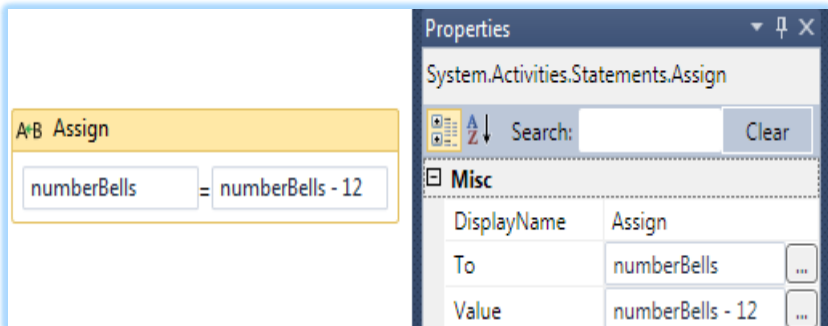
➤ Assign - მინიჭების პროცედურა

გადავიტანოთ ინსტრუმენტების პანელიდან Assign ქმედება. იგი საშუალებას იძლევა ცვლადს ან არგუმენტს მივანიჭოთ მნიშვნელობა. გრაფიკულად მიიღება ასეთი სურათი (ნახ.10.17).



ნახ.10.17. Assign ქმედების დამატება

Assign-ში To და Value, ორივე ლებულობს მნიშვნელობას. შეტანა შეიძლება უშუალოდ ველში, ან Properties-ის ფანჯარაში (“...“-ით გამოიძახება გამოსახულების შეტანის რედაქტორი). თვისებისთვის (To) შევიტანოთ numberBells. თვისების მნიშვნელობისთვის (Value) კი: numberBells-12. თვისებათა ფანჯარას ექნება ასეთი სახე (ნახ.10.18).

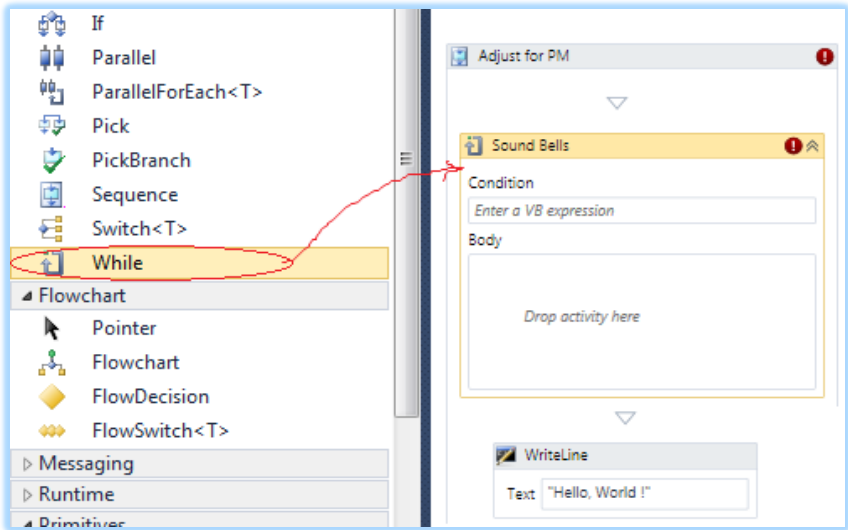


ნახ.10.18. Assign ქმედების თვისებათა ფანჯარა

მრავალი ქმედება არის შედგენილი აქტიურობა (ქმედება), რაც ნიშნავს, რომ იგი შეიძლება შეიცავდეს სხვა სახის ქმედებებს.

➤ While - ციკლის პროცედურა

დავამატოთ While ქმედება ზარის დასარეკად. გადმოვიტანოთ ინსტრუმენტების პანელიდან While აქტიურობა „Adjust for PM“-ის ქვემოთ. შევცვალოთ თვისებებში DisplayName სახელით Sound Bells (ნახ.10.19).



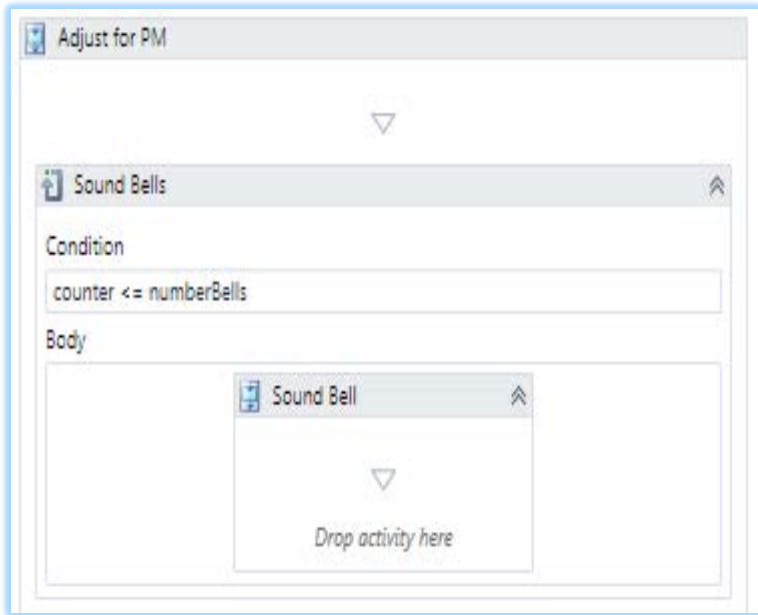
ნახ.10.19. While ქმედების დამატება Sound Bell დასახლებით

While ქმედებაში Body სექციის მოქმედება სრულდება მანამ, სანამ პირობა (Condition) ჭეშმარიტია. თავიდან პირობა მოწმდება და თუ ის არის „true“, მაშინ ქმედებები სრულდება. ეს მეორდება მანამ, სანამ პირობა გახდება "false".

შენიშვნა: DoWhile ქმედება იდენტურია While ქმედების, ოღონდაც ჯერ აქტიურობა სრულდება ერთხელ და შემდეგ მოწმდება პირობა.

შევიტანოთ Condition (პირობა) **counter <= numberBells.**

გადმოვიტანოთ ინსტრუმენტების პანელიდან Sequence ქმედება Body-სექციაში. მისი DisplayName შევცვალოთ Sound Bell-ით. მივიღებთ 10.20 ნახაზზე ნაჩვენებ სურათს.



ნახ.10.20. While ქმედება შეიცავს Sequence-ს

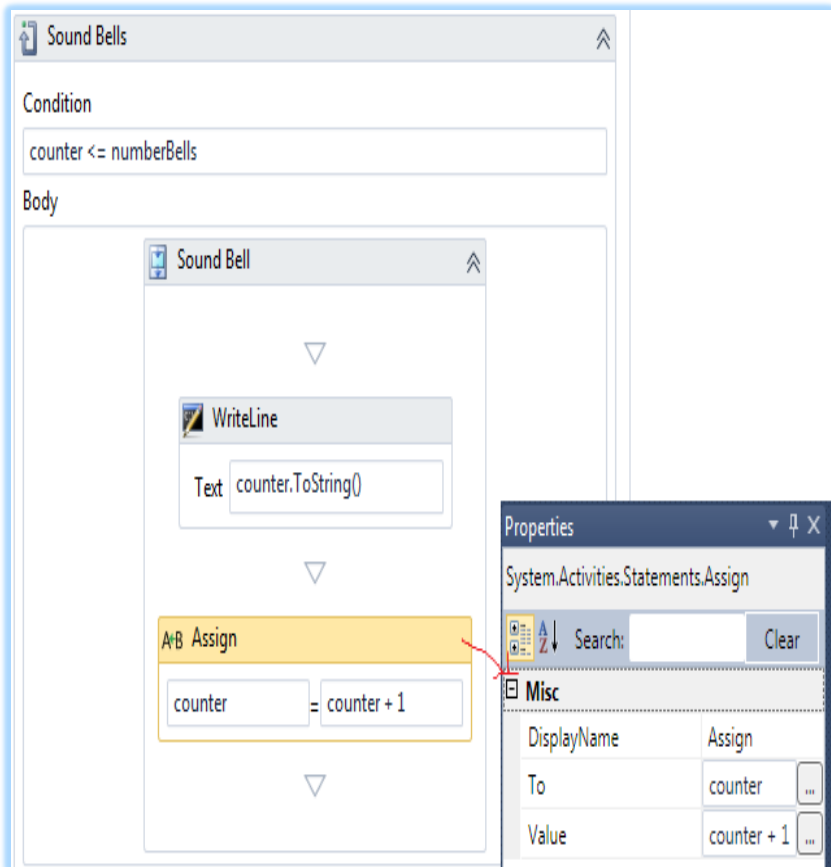
➤ **Sequence - პროცედურა**

გადმოვიტანოთ სამი სახის ქმედება Sequence-ზე „Sound Bell“. ამ სავარჯიშოში ზარი არ დარეკავს, მაგრამ კონსოლის სტრიქონი გამოიტანს ზარის რეკვის რაოდენობას. გადმოვიტანოთ „Sound Bell“-ში WriteLine ქმედება. თვისებების Text-ში შევიტანოთ ბრძანება:

```
counter.ToString()
```

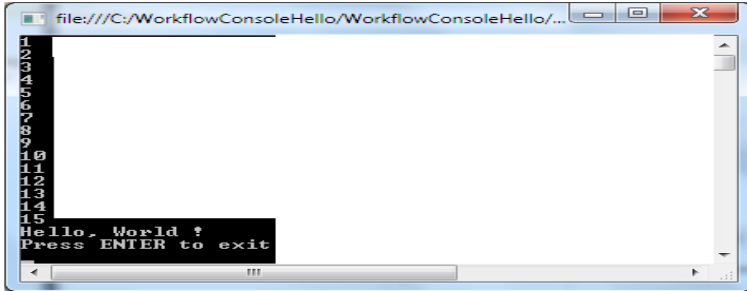
იგი გამოიტანს კონსოლზე მთვლელის მიმდინარე მნიშვნელობას.

გადმოვიტანოთ Assign ქმედება WriteLine ქმედების ქვემოთ. To თვისებისთვის შევიტანოთ counter; Value თვისებისთვის კი counter+1. ესაა მარტივი ინკრემენტის მაგალითი (ნახ.10.21).



ნახ.10.21. საბოლოო Sequence დიაგრამის კომპლექტი

სისტემის ამუშავების შემდეგ კონსოლზე გამოჩნდება ასეთი შედეგი (ნახ.10.22).



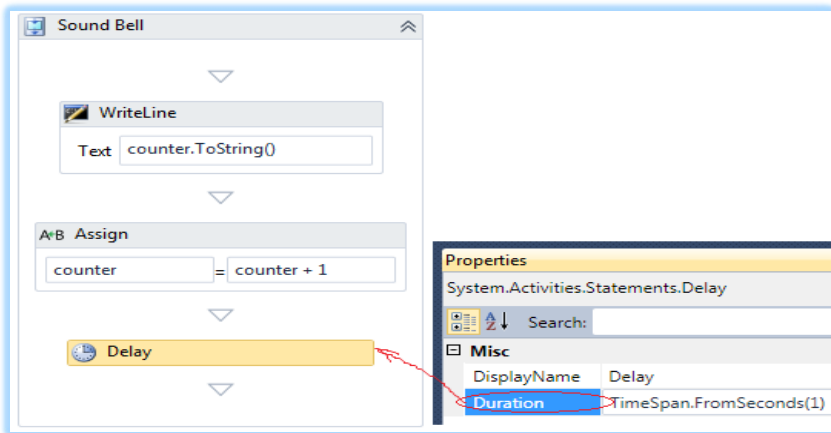
ნახ.10.22. შედეგი

➤ Delay ქმედება

გადმოვიტანოთ ინსტრუმენტების პანელიდან დაყოვნების Delay ქმედება Assign ქმედების ქვემოთ. დაყოვნების აქტიურობა განსაზღვრავს პაუზის ხანგრძლივობას. იგი მიეთითება როგორც TimeSpan კლასი. შევიტანოთ შემდეგი გამოსახულება: (ნახ.10.23).

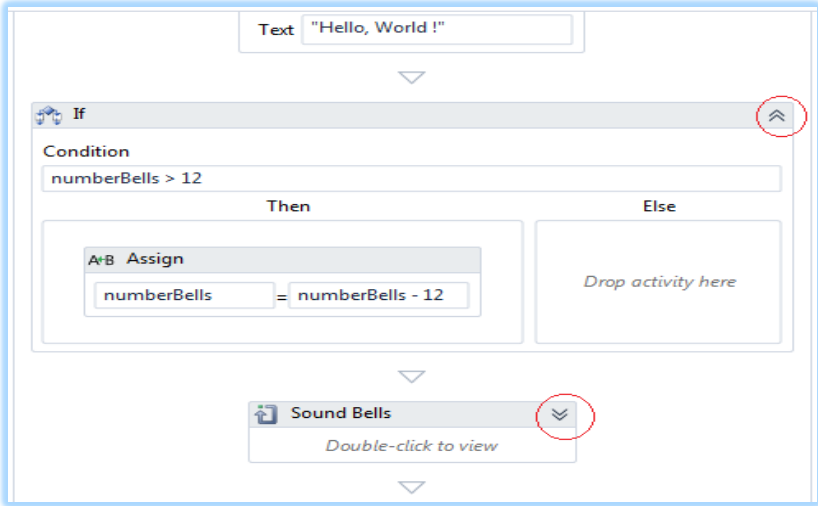
TimeSpan.FromSeconds(1)

მივიღებთ 10.23 ნახაზზე მოცემულ სურათს.



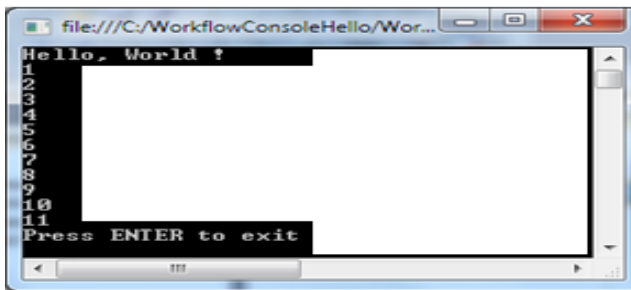
ნახ.10.23. დაყოვნების ქმედების დამატება Sequence-ში

10.24 ნახაზზე რგოლებით შემოხაზულია ჩაკეცვა-გამლის ისრები. Sound Bell-სთვის (ნახ.10.23) ის ჩაკეცილია და ჩანს მხოლოდ სათაური.



ნახ.10.24

თუ ავამუშავებთ ამ სიტუაციას, მივიღებთ 1.25 სურათს.



ნახ.10.25

რიცხვები 1–11 გამოიტანება მიმდევრობით, ოდნავ დაყოვნებით.

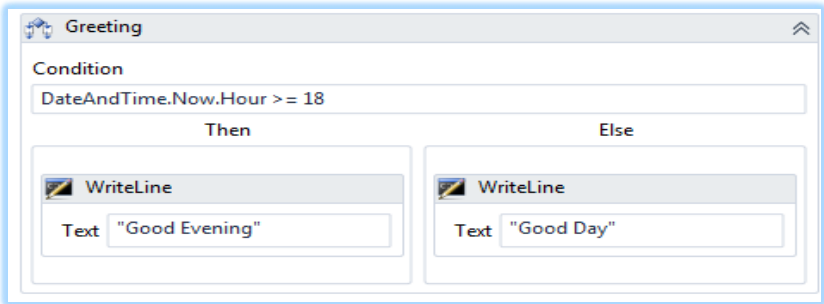
ახლა გადმოვიტანოთ WriteLine ქმედება Sound Bell-ის ქვემოთ. შევცვალოთ DisplayName სახელით Display Time. თვისებისთვის Text შევიტანოთ გამოსახულება:

„The time is: " + DateAndTime.Now.ToString()

გადმოვიტანოთ if ქმედება „Display Time“-ს ქვემოთ და DisplayName შევცვალოთ Greeting-ით. Condition პირობისთვის შევიტანოთ გამოსახულება:

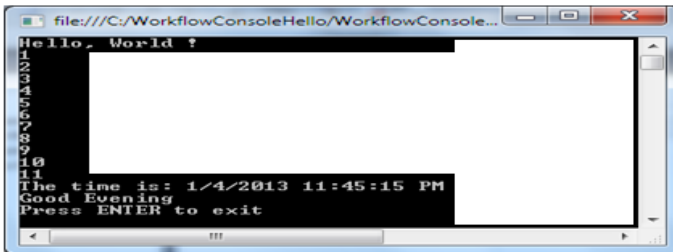
DateAndTime.Now.Hour >= 18

გადმოვიტანოთ WriteLine ქმედება ორივე Then და Else სექციებში. Then სექციისთვის Text-ში შევიტანოთ „Good Evening“; Else სექციისთვის Text-ში შევიტანოთ „Good Day“. ამგვარად, „Greeting“ აქტიურობა მიიღებს ასეთ სახეს (ნახ.10.26).



ნახ.10.26. Greeting ქმედება

ავამუშავოთ აპლიკაცია F5-ით.



ნახ.10. 27. შედეგი

თავი 11

პროგრამული აპლიკაციის დიზაინი XAML ენაზე

11.1. XAML ენის საფუძვლები

მომხმარებელთა ინტერფეისების აგება WPF- და Silverlight-დანართებისთვის (აპლიკაციებისთვის) ხორციელდება XAML (Extensible Application Markup Language - აპლიკაციების გაფართოებადი ფორმატირების ენა) ენის გამოყენებით. XAML-დოკუმენტი შეიცავს ფორმატს, რომელიც აღწერს დანართის ფანჯრის (ან გვერდის) გარეგან სახეს და ქცევას, ხოლო მასთან კავშირში მყოფი C# კოდის ფაილები კი - დანართის ლოგიკას.

XAML ენა უზრუნველყოფს დანართის დიზაინის პროცესის (გრაფიკული ნაწილი) გამოყოფას ბიზნეს-ლოგიკის (პროგრამული კოდი) დამუშავების პროცესისგან, დიზაინერებსა და დეველოპერებს შორის [4,5].

WPF-ის XAML არის XML-ენის ქვესიმრავლე, გაფართოებული დამატებითი ფუნქციებით. იგი უზრუნველყოფს WPF-შიგთავსის აღწერას ისეთი ელემენტებით, როგორცაა ვექტორული გრაფიკა, მართვის ელემენტები და დოკუმენტები.

XAML-ის საფუძველია XML და მისი სინტაქსი განისაზღვრება შემდეგი წესებით:

- XAML-დოკუმენტის ყოველი ელემენტი აისახება .NET კლასის რომელიღაც ეგზემპლარში. ასეთი ელემენტის სახელი ზუსტად შეესაბამება კლასის სახელს. მაგალითად, <Button> ელემენტი ემსახურება WPF-ინსტრუქციას Button-კლასის ობიექტის აგების მიზნით;

- XAML-ის ელემენტები შეიძლება ერთმანეთში ჩალაგდეს. ელემენტების ჩალაგების ფორმატი ასახავს ინტერფეისის ელემენტების ჩალაგებას;

- კლასის თვისებები განისაზღვრება ატრიბუტებით ან ჩალაგებული დესკრიპტორების დახმარებით, სპეცსინტაქსით.

XAML-ენა ხასიათდება თვითაღწერადობით. XAML-დოკუმენტში ყოველი ელემენტი არის ტიპის სახელი (მაგალითად, Button, Window ან Page) მოცემული სახელსივრცის ჩარჩოებში.

ელემენტთა ატრიბუტები გამოიყენება შესაბამისი ობიექტების თვისებების (მაგალითად, Name, Height, Width და ა.შ.) და მოვლენების (Click, Load და ა.შ.) მოსაცემად.

WPF-დანართის MyFirstWpfProject შექმნის დროს VisualStudio აგენერირებს შემდეგ XAML-დოკუმენტს:

```
<Window x:Class="MyFirstWpfProject.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/
                2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
  <Grid>
    . . .
  </Grid>
</Window>
```

WPF-დანართის XAML-დოკუმენტი MyFirstWpfProject იწყება დესკრიპტორით < Window...>.

XAML-დოკუმენტის ყველა დესკრიპტორი იწყება „<“ - სიმბოლოთი და მთავრდება „>“ - სიმბოლოთი. ნებისმიერი XAML-დოკუმენტი შედგება XAML-ელემენტებისგან. ყოველი XAML-დოკუმენტი (XAML-ელემენტი) იწყება გახსნის დესკრიპტორით (მაგალითად, < Window >), რომელსაც მოჰყვება დოკუმენტის შიგთავსი (მაგალითად, ტექსტური სტრიქონი ან სხვა XAML-ელემენტები). გახსნის დესკრიპტორში შეიძლება იყოს მოთავსებული ატრიბუტების აღწერა (მაგალითად, Class, xmlns, Title, Height, Width და სხვ.). XAML-დოკუმენტი (XAML-ელემენტი)

უნდა დასრულდეს დახურვის დესკრიპტორით (მაგალითად, „/>“ ან „</Window>“).

XAML დოკუმენტის ტექსტი უნდა შეიცავდეს ერთ ფესვურ ელემენტს - ჩალაგების უმაღლესი დონის ელემენტი. WPF-დანართის MyFirstWpfProject XAML-დოკუმენტში ასეთი ელემენტია <Window>. ფესვურ ელემენტში შეიძლება დაემატოს XAML-ის სხვა ელემენტებიც. მაგალითში ასეთი ელემენტია <Grid>.

WPF-დანართის XAML-დოკუმენტის კომპილაციის პროცესში სინტაქსურ ანალიზატორს გადაჰყავს XAML ფაილები აპლიკაციის ორობითი ფორმატირების ფაილებში BAML (Binary Application Markup Language), რომლებიც შემდეგ ჩაშენდება პროექტის ნაკრებში რესურსების სახით. WPF-დანართის კლასების ასაგებად სინტაქსური ანალიზატორი გამოიყენებს სახელსივრცეს, რომელიც განსაზღვრულია XAML-დოკუმენტის ფესვურ დესკრიპტორში.

XAML-დოკუმენტში სახელსივრცე მოიცემა xmlns ატრიბუტის საშუალებით. ზემოაღწერილ დოკუმენტში გამოცხადებულია ორი საბაზო სახელსივრცე:

- xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation - ესაა WPF-ის საბაზო სახელსივრცე, რომელიც მოიცავს WPF-ის ყველა კლასს, მართვის ელემენტების ჩათვლით. ისინი გამოიყენება მომხმარებლის ინტერფეისის ასაგებად. ვინაიდან სახელსივრცე ცხადდება პრეფიქსის გარეშე, იგი ვრცელდება მთელი XAML-დოკუმენტისთვის;

- xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" - ესაა XAML-ის სახელსივრცე. იგი შეიცავს XAML უტილიტების სხვადასხვა თვისებებს, რომლებიც გავლენას ახდენს იმაზე, თუ XAML-დოკუმენტი როგორ ინტერპრეტირდება. მოცემული სახელსივრცე აისახება x პრეფიქსზე. ეს პრეფიქსი შეიძლება მოთავსდეს ელემენტის სახელის წინ (მაგალითად, x:ელემენტის_სახელი).

მეორე სახელსივრცე გამოიყენება XAML-ის სპეციფიური ლექსემების („საკვანძო სიტყვები“) ჩასასმელად. 11.1 ცხრილში მოცემულია შედარებით ხშირად გამოყენებადი ასეთი სიტყვები.

XAML-ის საკვანძო სიტყვები		ცხრ.11.1
N	საკვანძო სიტყვა	დანიშნულება
1.	x:Array	წარმოადგენს .NET-ის მასივის ტიპს XAML-ზე
2.	x:Class	XAML-ფაილის კლასის სახელი
3.	x:ClassModifier	უზრუნველყოფს კლასის ტიპის ხილვადობის (internal ან public) განსაზღვრას, რომელიც Class საკვანძო სიტყვითაა აღნიშნული
4.	x:Code	პროგრამული კოდი შეიძლება უშუალოდ ჩაიდოს XAML-კოდში
5.	x:FieldModifier	უზრუნველყოფს ტიპის წევრის ხილვადობის (internal, public, private ან protected) განსაზღვრას ფესვის ნებისმიერი სახელმინიჭებული ელემენტისთვის (საკვანძო სიტყვით Name)
6.	x:Key	უზრუნველყოფს გასაღების მნიშვნელობის დაყენებას XAML ელემენტისთვის, რომელიც უნდა მოთავსდეს ლექსიკონის ელემენტში
7.	x:Name	უზრუნველყოფს C#-ით გენერირებული სახელის მითითებას მოცემული XAML ელემენტისთვის
8.	x:Null	წარმოადგენს null-მითითებელს

ვიზუალური დაპროგრამება (C#.NET & Workflow Foundation NET)

9.	x:Shared	შესაძლებელია მხოლოდ იმ რესურსებისთვის, რომლებიც ერთხელ იძლევა ეგზემპლარს. ჩვეულებრივად, ყოველი წვდომისას იწარმოება რესურსის ახალი ეგზემპლარი
10.	x:Static	უზრუნველყოფს ტიპის სტატიკურ წევრზე მიმართვას
11.	x:Subclass	ეს კონსტრუქცია ქმნის წარმოებულ კლასს და გამოდგება დაპროგრამების მხოლოდ იმ ენებისთვის, რომელთაც არ აქვს დანაწევრებული (partielle) კლასების მხარდაჭერა
12.	x:Type	XAML-ეკვივალენტია C#-ია typeof ოპერაციის (იმახებს System.Type მითითებული სახელის საფუძველზე)
13.	x:TypeArgument	უზრუნველყოფს ელემენტის დაყენებას, როგორც განზოგადებული ტიპისას განსაზღვრული პარამეტრებით
14.	x:Uid	x:Name –ს პარალელურად შეუძლია მართვის ელემენტს ამ ატრიბუტით ცალსახა სახელი მიიღოს, რომელიც შემდგომი მთარგმანისთვის იქნება გამოყენებული
15	x:XData	ქმნის „მონაცემთა კუნძულს“ XAML-ის შიგნით, შეუძლია მარტივი XML-მონაცემთა კონსტრუქციით წარმოება

WPF-დანართებში საბაზო სახელსივრცეთა გარდა იყენებენ ასევე სპეციალურს, რომლებიც არააუცილებელია:

- <http://schemas.openxmlformats.org/markup-compatibility/2006> - XAML-ის სახელსივრცეა, დაკავშირებული ფორმატირების თავსებადობის პრობლემასთან სამუშაო გარემოსთან. ეს

სახელსივრცე გამოიყენება XAML-ის სინტაქსური ანალიზატორის ინფორმირებისათვის იმის შესახებ, თუ რომელი ინფორმაცია დასამუშავებელი და რომელი საიგნორირო;

- <http://schemas.microsoft.com/expression/blend/2008> - XAML-ის სახელსივრცეა, რომელსაც აქვს მხარდაჭერა Expression Blend და Visual Studio პროგრამებიდან. გამოიყენება გვერდის გრაფიკული პანელის ზომების დასაყენებლად.

Window ობიექტის ატრიბუტებში შეიძლება დაემატოს შემდეგი XAML-აღწერები:

```
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="600"
```

მოცემული XAML-აღწერა აცხადებს არასავალდებულო სახელსივრცეებს პრეფიქსებით mc და d. თვისებები DesignHeight და DesignWidth იმყოფება სახელსივრცეში, რომელსაც აქვს პრეფიქსი d. ეს თვისებები განსაზღვრავს, რომ დანართის პროექტის დამუშავებისას Visual Studio დიზაინერში ფანჯარას უნდა ჰქონდეს ზომები 300x600. თვისება Ignorable მდებარეობს სახელსივრცეში, რომელიც აღნიშნულია პრეფიქსით mc და ის სინტაქსურ ანალიზატორს აინფორმირებს, რომ მან იგნორირება გაუკეთოს XAML-დოკუმენტის ნაწილს, რომელიც აღნიშნულია d პრეფიქსით.

WPF-დანართის XAML-დოკუმენტში ხშირად საჭიროა განხორციელდეს წვდომა პროექტის სხვა რომელიმე სახელსივრცესთან. ამ დროს აუცილებელია ახალი პრეფიქსის განსაზღვრა და მიეცეს სახელსივრცე. თუ პროექტში არის სახელსივრცე MyFirstWpfProject.Commands, მაშინ მის მიერთებას WPF -დანართის XAML-დოკუმენტთან ექნება შემდეგი სახე (command - გამოიყენება პრეფიქსის სახით).

```
xmlns:command="clr-namespace: MyFirstWpfProject.Commands"
```

პრეფიქსი (command) გამოიყენება მიმართვისთვის სახელსივრცეზე XAML-დოკუმენტში. clr-namespace ლექსემს ენიჭება სახელსივრცის დასახელება .NET ნაკრებში.

XAML-დოკუმენტში კლასის აღსაწერად გამოიყენება ატრიბუტი Class. XAML-დოკუმენტის სტრიქონი

```
<Window x:Class="MyFirstWpfProject.MainWindow" ...>
```

ითვალისწინებს MyFirstWpfProject.MainWindow კლასის შექმნას Window კლასის ბაზაზე. Class ატრიბუტის x პრეფიქსი განსაზღვრავს იმას, რომ ეს ატრიბუტი თავსდება XAML-ის სახელსივრცეში.

MainWindow კლასი გენერირდება ავტომატურად კომპილაციის დროს. კლასის ნაწილისთვის ავტომატურად გენერირდება კოდი (ნაწილობრივი (partial) კლასი):

```
namespace MyFirstWpfProject
{
    // <summary>
    // ურთიერთქმედების ლოგიკა MainWindow.xaml - ისთვის
    // </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

როდესაც სრულდება დანართის კომპილაცია, XAML-ფაილი, რომელიც განსაზღვრავს მომხმარებლის ინტერფეისს (MainWindow.xaml), ტრანსლირდება CLR ტიპის გამოცხადებაში, რომელიც ერთიანდება დანართის ლოგიკასთან გამოყოფილი კოდის კლასის ფაილიდან (MainWindow.xaml.cs).

InitializeComponent() მეთოდი გენერირდება დანართის კომპილაციის დროს და საწყის კოდში არ თავსდება.

XAML-დოკუმენტში აღწერილი მართვის ელემენტების პროგრამულად სამართავად, აუცილებელია მართვის ელემენტს მიეცეს XAML ატრიბუტი Name. მაგალითად, Grid ელემენტისთვის ჩაიწერება ასე:

```
<Grid Name="grid">
```

```
</Grid>
```

მარტივი თვისებები XAML-დოკუმენტში მოიცემა შემდეგი სინტაქსის შესაბამისად:

თვისების_სახელი = "მნიშვნელობა"

მაგალითად, **Name = "grid1"**

თვისების მისაცემად, რომელიც არის სრულფასოვანი ობიექტი, გამოიყენება *რთული თვისებები* „თვისება-ელემენტი“ სინტაქსის შესაბამისად:

მშობელი.თვისების_სახელი

მაგალითად, StackPanel კონტეინერისთვის აუცილებელია მიეცეს გრადიენტული ფუნჯი პანელის შესავსებად, რაც განისაზღვრება Background ატრიბუტით. იგი რეალიზდება დესკრიპტორებით:

```
<StackPanel.Background> . . . </StackPanel.Background>.
```

თვისების მნიშვნელობის მისაცემად გამოყოფილი კლასიდან გამოიყენება ფორმატირების გაფართოება, რომელიც უზრუნველყოფს XAML გრამატიკის გაფართოებას ახალი ფუნქციონალობით. ფორმატირების გაფართოება შეიძლება გამოყენებულ იქნას ჩალაგებულ დესკრიპტორებში ან XAML-ატრიბუტებში. როცა იყენებენ ატრიბუტებს, მაშინ აუცილებელია ფიგურული ფრჩხილების {...} გამოყენება.

ფორმატირების გაფართოებები იყენებს შემდეგ სინტაქსს:

{ფორმატირების_გაფართოების_კლასი არგუმენტი}

ფორმატირების გაფართოებები რეალიზდება კლასებით, რომლებიც შვილობილია System.Windows.Markup.MarkupExtention კლასის. MarkupExtention საბაზო კლასს აქვს ProvideValue() მეთოდი, რომელიც იძლევა ატრიბუტისთვის საჭირო მნიშვნელობას. მაგალითად, იმისათვის, რომ Button-ობიექტის Foreground ატრიბუტს მიეცეს სტატიკური თვისება, რომელიც სხვა კლასშია განსაზღვრული, აუცილებელია შემდეგი XAML-აღწერის შექმნა:

```
<Button Foreground="{x:Static SystemColors.ActiveCaptionBrush}" />
```

კომპილაციის დროს სინტაქსური ანალიზატორი შექმნის Static Extention კლასის ეგზემპლარს, შემდეგ გამოიძახებს ProvideValue() მეთოდს, რომელიც ამოიღებს საჭირო მნიშვნელობას და დააყენებს მას Foreground თვისებისთვის.

ფორმატირების გაფართოებები შეიძლება გამოყენებულ იქნას როგორც ჩალაგებული თვისებები.

მიერთებული თვისებები აღწერს თვისებებს, რომელთა გამოყენება შეიძლება რამდენიმე მართვის ელემენტთან, ოღონდ რომლებიც განსაზღვრულია სხვა კლასში. WPF-დანართებში მიერთებული თვისებები ხშირად გამოიყენება ინტერფეისის ელემენტების დაკომპლექტების სამართავად. მიერთებული თვისებების სინტაქსი შემდეგია:

განსაზღვრელი_ტიპი.თვისების_სახელი

მაგალითად, თუ საჭიროა ღილაკის მოთავსება ბადის 0-ოვან სტრიქონში, მაშინ აუცილებელია შემდეგი XAML აღწერის შექმნა:

```
<Button ... Grid.Row="0" >
```

```
....
```

```
</Button>
```

აქ მიერთებული თვისებაა Grid.Row, ანუ Grid-ელემენტის Row-თვისება, რომელიც არაა Button ობიექტის თვისება. თვისება Row მიუერთდება Button ობიექტის თვისებებს, ვინაიდან ეს ობიექტი განთავსებულია Grid კონტეინერში.

ობიექტის ატრიბუტები შეიძლება გამოყენებულ იქნას მოვლენათა დამმუშავებლების მისაერთებლად, შემდეგი სინტაქსის გამოყენებით:

მოვლენის სახელი = "მოვლენის_დამმუშავებლის_მეთოდის_სახელი"
მაგ., ღილაკის Click მოვლენისთვის (მისი დაჭერისას), შეიძლება დაყენდეს მოვლენის დამმუშავებელი Exit_Click.

```
<Button Name="Exit" Content="გამოსვლა" Click="Exit_Click" />
```

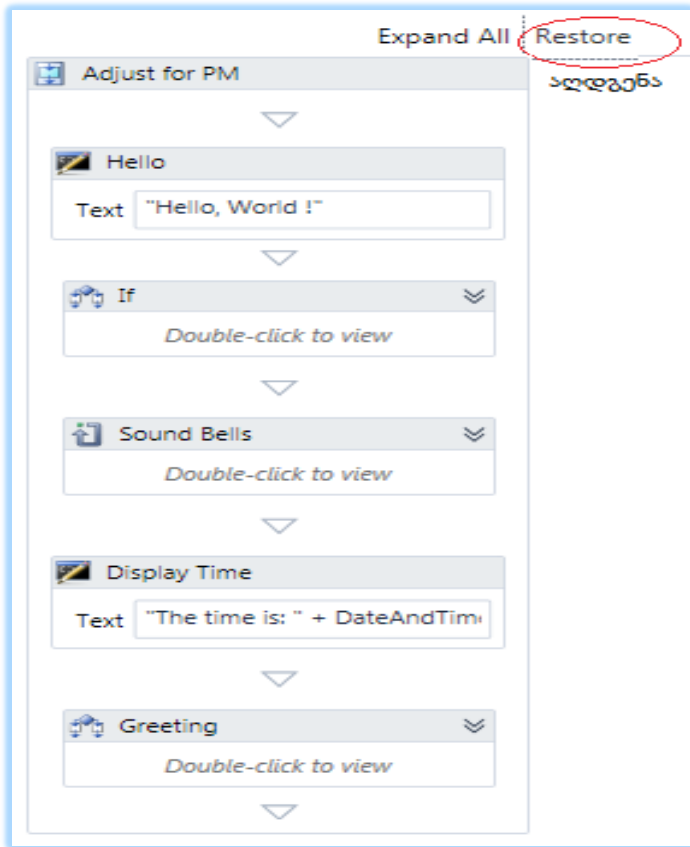
XAML-აღწერაში Exit_Click დამმუშავებლის განსაზღვრისას, აუცილებელია კლასის კოდში გვექნოდეს მეთოდი კორექტული სიგნატურით. ქვემოთ მოყვანილია კოდი, რომელიც გენერირდება ავტომატურად მოვლენის დამმუშავებლის აღწერის შექმნისას XAML-დოკუმენტში.

```
private void Exit_Click(object sender,RoutedEventArgs e)
{
    ...
}
```

11.2. დიზაინერის მართვა და XAML კოდი

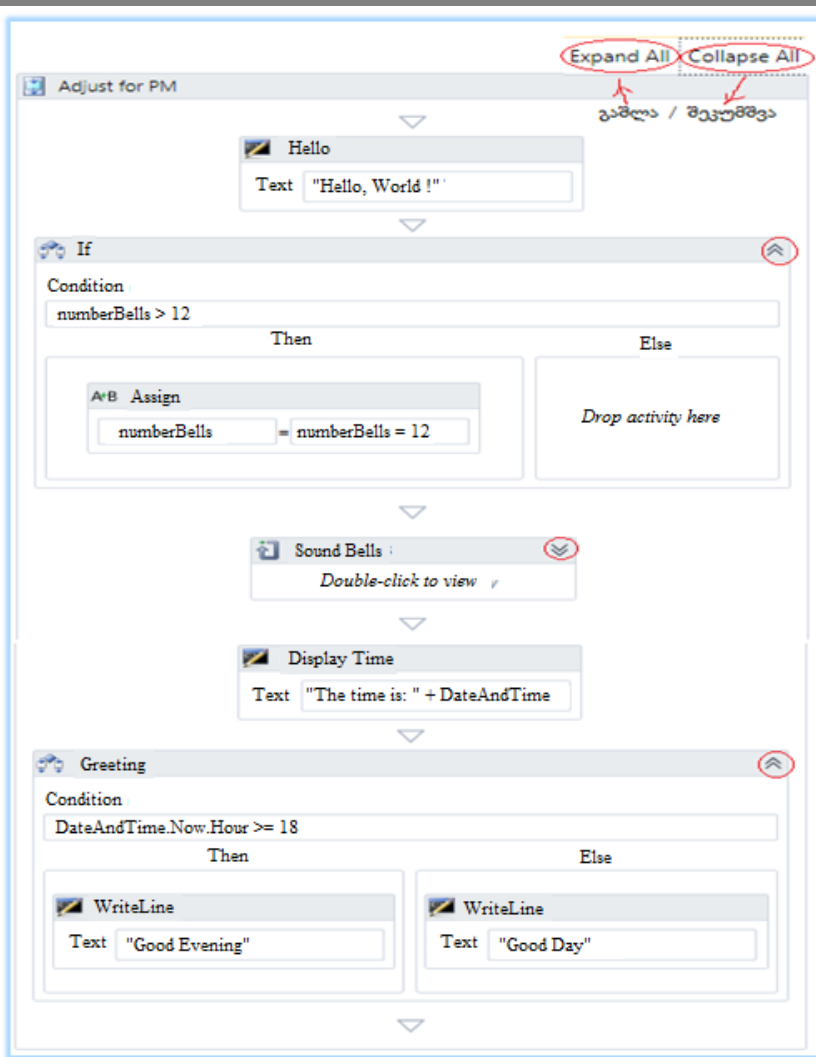
Visual Studio.NET პაკეტის WF-ის დიზაინერის სამუშაო გარემოში მნიშვნელოვანია დიდი პროექტების და სექციების მართვის საშუალებების გაცნობა (Navigating the Designer).

მარტივი მაგალითიდან დავინახეთ, რომ საკმაოდ რთულია სამუშაო პროცესების დიდი სექციების მართვა და მათი მთლიანობაში წარმოდგენა. დიდი პროექტების დროს ის კიდევ უფრო რთულია. ამისათვის დიზაინერებს ეძლევათ საშუალება ჩაკეცონ სექციის სექციები, მათი მთლიანი სტრუქტურის დათვალიერების და მართვის მიზნით. მაგალითად, ჩვენი პროექტი ჩაკეცვის (Collapse All) შემდეგ ასე გამოიყურება (ნახ.11.1).



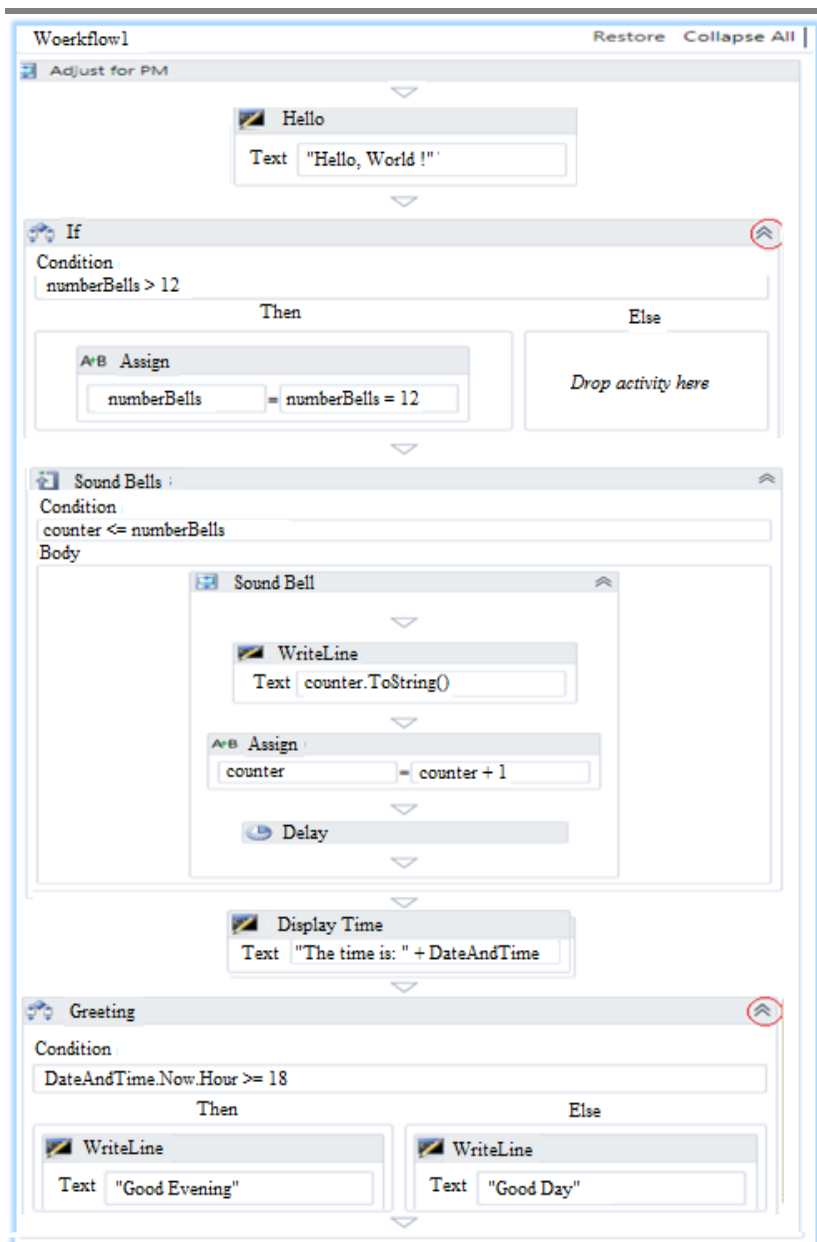
ნახ.11.1. შეკუმშული workflow დიაგრამა

თუ აღდგენის (Restore) ღილაკს გამოვიყენებთ, მაშინ მივიღებთ სურათს შეკუმშვამდე (ნახ.11.2).



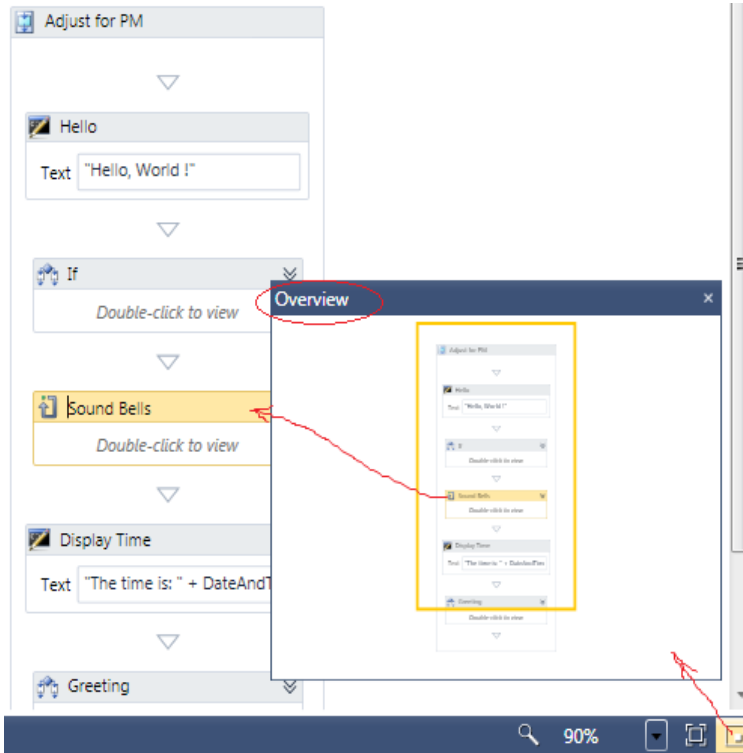
ნახ.11.2. ნაწილობრივ შეკუმშული workflow დიაგრამა

აქ რამდენიმე ბლოკი გაშლილია მთლიანად, რამდენიმე კი ჩაკეცილია (მაგალითად, Bound Bells). Expand All ღილაკით კი გაიშლება ყველა ჩაკეცილი ბლოკი (ნახ.11.3).



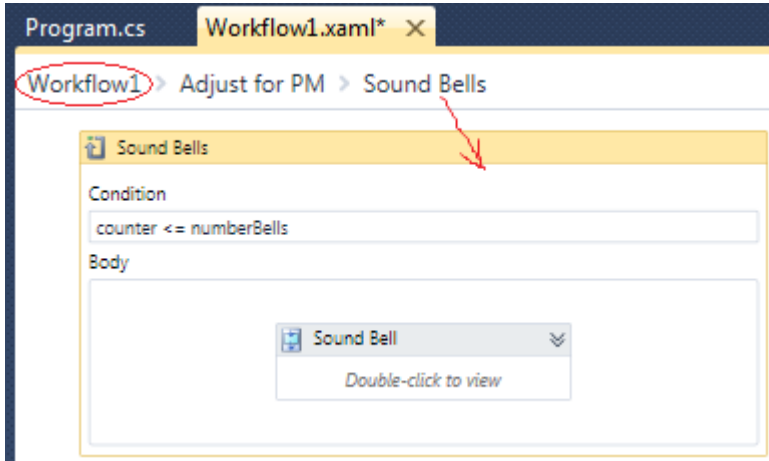
ნახ.11.3. მთლიანად გაშლილი workflow დიაგრამა

დიზაინერის ქვედა მარჯვენა კუთხეში არის Overview-ლილაკი, რომელიც ხსნის პროექტის მონიტორინგის ფანჯარას (ნახ.11.4). მოყვითალო ფერის ზოლი მიუთითებს აქტიურ ხილვად ბლოკზე. აქვეა ეკრანის ზომების ცვლილების (მასშტაბირების) ღილაკებიც.



ნახ.11.4. Overview - ღილაკი

„Sound Bell“ ქმედების დაკლიკვით გამოჩნდება მხოლოდ ეს ბლოკი თავისი შვილობილი (ჩადგმული) კომპონენტებით (ნახ.11.5). აქ Window1 გადამრთველის არჩევით დიზაინერში გამოჩნდება ისევ 11.1 ნახაზზე ნაჩვენები სქემა.



ნახ.11.5. გამოყოფილი Sound Bell ბლოკი

➤ დიზაინერის XAML-კოდი

Solution Explorer-ში დავკლიკოთ Workflow1.xaml და დავათვალიეროთ შესაბამისი კოდი (ლისტინგი_11.1):

<! ---- ლისტინგი_11.1 ---- >

```
<Activity mc:Ignorable="" x:Class="WorkflowConsoleHello.Workflow1"
  xmlns="http://schemas.microsoft.com/netfx/2009/xaml/activities"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:mv="clr-namespace:Microsoft.VisualBasic;assembly=System"
  xmlns:mva="clr-namespace:Microsoft.VisualBasic.Activities;assembly=
  System.Activities"
  xmlns:s="clr-namespace:System;assembly=microsoftcorlib"
  xmlns:s1="clr-namespace:System;assembly=System"
  xmlns:s2="clr-namespace:System;assembly=System.Xml"
  xmlns:s3="clr-namespace:System;assembly=System.Core"
  xmlns:s4="clr-namespace:System;assembly=System.ServiceModel"
  xmlns:sa="clr-namespace:System.Activities;assembly=System.Activities"
```

```

xmlns:sad="clr-namespace:System.Activities.Debugger;assembly=
                System.Activities"
xmlns:sap="http://schemas.microsoft.com/netfx/2009/xaml/activities/
                presentation"
xmlns:scg="clr-namespace:System.Collections.Generic;assembly=System"
xmlns:scg1="clr-namespace:System.Collections.Generic;assembly=
                System.ServiceModel"
xmlns:scg2="clr-namespace:System.Collections.Generic;assembly=
                System.Core"
xmlns:scg3="clr-namespace:System.Collections.Generic;assembly=microsoftlib"
xmlns:sd="clr-namespace:System.Data;assembly=System.Data"
xmlns:sl="clr-namespace:System.Linq;assembly=System.Core"
xmlns:st="clr-namespace:System.Text;assembly=microsoftlib"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
<sap:WorkflowViewStateService.ViewState>
<scg3:Dictionary x:TypeArguments="x:String, x:Object">
<x:Boolean x:Key="ShouldCollapseAll">True</x:Boolean>
<x:Boolean x:Key="ShouldExpandAll">False</x:Boolean>
</scg3:Dictionary>
</sap:WorkflowViewStateService.ViewState>

<Sequence DisplayName="Sequence"

sad:XamlDebuggerXmlReader.FileName="C:\WorkflowConsoleHello\Workflow
                ConsoleHello\Workflow1.xaml"
sap:VirtualizedContainerService.HintSize="233,564">
<Sequence.Variables>
<Variable x:TypeArguments="x:Int32" Default="1" Name="counter" />
<Variable x:TypeArguments="x:Int32" Default="[DateTime.Now.Hour]"
Name="numberBells" />
</Sequence.Variables>

```

```
<sap:WorkflowViewStateService.ViewState>
  <scg3:Dictionary x:TypeArguments="x:String, x:Object">
    <x:Boolean x:Key="IsExpanded">True</x:Boolean>
  </scg3:Dictionary>
</sap:WorkflowViewStateService.ViewState>
```

```
<WriteLine DisplayName="Hello"
sap:VirtualizedContainerService.HintSize="211,62"
```

```
Text="Hello, World !" />
```

```
<If Condition="[numberBells > 12]"
sap:VirtualizedContainerService.HintSize="211,52">
  <sap:WorkflowViewStateService.ViewState>
    <scg3:Dictionary x:TypeArguments="x:String, x:Object">
      <x:Boolean x:Key="IsExpanded">True</x:Boolean>
      <x:Boolean x:Key="IsPinned">False</x:Boolean>
    </scg3:Dictionary>
  </sap:WorkflowViewStateService.ViewState>
  <If.Then>
    <Assign sap:VirtualizedContainerService.HintSize="291,100">
      <Assign.To>
        <OutArgument x:TypeArguments="x:Int32">[numberBells]
          </OutArgument>
        </Assign.To>
      <Assign.Value>
        <InArgument x:TypeArguments="x:Int32">[numberBells - 12]
          </InArgument>
        </Assign.Value>
      </Assign>
    </If.Then>
  </If>
```

```

<While DisplayName="Sound Bells"
sap:VirtualizedContainerService.HintSize="211,52">
  <sap:WorkflowViewStateService.ViewState>
    <scg3:Dictionary x:TypeArguments="x:String, x:Object">
      <x:Boolean x:Key="IsExpanded">False</x:Boolean>
      <x:Boolean x:Key="IsPinned">False</x:Boolean>
    </scg3:Dictionary>
  </sap:WorkflowViewStateService.ViewState>
  <While.Condition>[counter &lt;= numberBells]</While.Condition>
  <Sequence DisplayName="Sound Bell"
sap:VirtualizedContainerService.HintSize="438,100">
    <sap:WorkflowViewStateService.ViewState>
      <scg3:Dictionary x:TypeArguments="x:String, x:Object">
        <x:Boolean x:Key="IsExpanded">False</x:Boolean>
        <x:Boolean x:Key="IsPinned">False</x:Boolean>
      </scg3:Dictionary>
    </sap:WorkflowViewStateService.ViewState>

    <WriteLine sap:VirtualizedContainerService.HintSize="242,62"
Text="[counter.ToString()]" />
    <Assign sap:VirtualizedContainerService.HintSize="242,58">
      <Assign.To>
        <OutArgument x:TypeArguments="x:Int32">[counter]</OutArgument>
      </Assign.To>
      <Assign.Value>
        <InArgument x:TypeArguments="x:Int32">[counter + 1]</InArgument>
      </Assign.Value>
    </Assign>

    <Delay Duration="[TimeSpan.FromSeconds(1)]"
sap:VirtualizedContainerService.HintSize="242,22" />

```

```
</Sequence>
</While>

<WriteLine DisplayName="Display Time"
sap:VirtualizedContainerService.HintSize="211,62"
    Text="[&quot;The time is: &quot; + DateAndTime.Now.ToString()]" />

<If Condition="[DateAndTime.Now.Hour &gt;= 18]" DisplayName="Greeting"
    sap:VirtualizedContainerService.HintSize="211,52">
    <If.Then>
        <WriteLine sap:VirtualizedContainerService.HintSize="219,100" Text="Good
            Evening" />
    </If.Then>
    <If.Else>
        <WriteLine sap:VirtualizedContainerService.HintSize="220,100" Text="Good
            Day" />
    </If.Else>
</If>
</Sequence>
</Activity>
```

თავი 12

ბიზნესპროცესების (Workflows) დაპროგრამება

12.1. კოდირებული ბიზნესპროცესები

განვიხილოთ ბიზნესპროცესების (Workflows) დაპროგრამებისა და კოდის შერულების საშუალებები Visual Studio.NET პაკეტის WF-ის სამუშაო გარემოში.

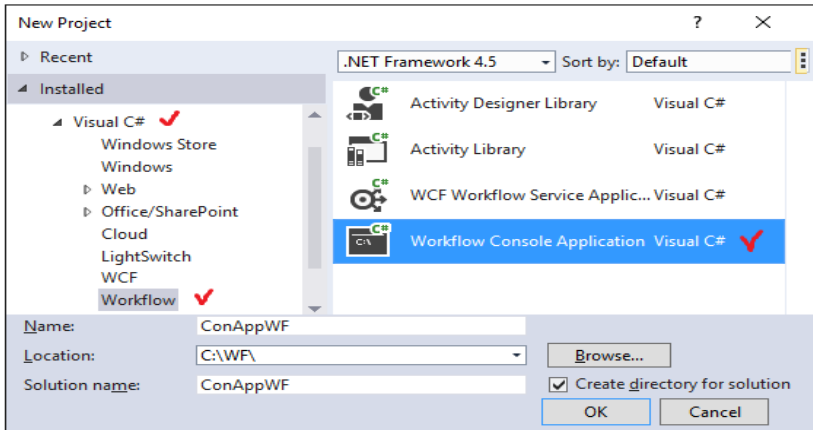
წინა თავში ჩვენ ავაგეთ მარტივი სამუშაო პროცესი (workflow) დიზაინერის გამოყენებით. ახლა განვიხილოთ იგივე სამუშაო პროცესის შესრულება კოდის გამოყენებით.

ნებისმიერი ბიზნესპროცესი შეიძლება განხორციელდეს **კოდის ან დიზაინერის** გამოყენებით (შემსრულებლის გემოვნების საკითხია).

კოდის გამოყენება საშუალებას იძლევა უფრო კარგად გავიგოთ თუ როგორ ხდება სამუშაო პროცესის (workflow) რეალიზაცია და ფუნქციონირება.

➤ კონსოლური აპლიკაცია აგება

Visual Studio.NET გარემოში შევქმნათ ახალი კონსოლური აპლიკაცია სახელით ConAppWF (ნახ.12.1).



ნახ.12.1. კონსოლის აპლიკაციის შექმნა

დავამატოთ მიმართვა (reference) **System.Activities**. იგი საშუალებას იძლევა აპლიკაციაში გამოყენებულ იქნას სამუშაო პროცესის (workflow) ქმედებები (აქტიურობები).

შევცვალოთ Program.cs ფაილში სახელსივრცეები შემდეგი სახით (ლისტინგი_1.3).

// ----- ლისტინგი_12.1 -----

```
using System;
using System.Activities;
using System.Activities.Statements;
using System.Activities.Expressions;
namespace ConAppWF
{
    class Program
    {
        static void Main(string[] args)
        {
            WorkflowInvoker.Invoke(CreateWorkflow());
            Console.WriteLine("Press ENTER to exit");
            Console.ReadLine();
        }
    }
}
```

Main()-ის კოდი მსგავსია ჩვენ მიერ წინა მასალაში (თ.10) განხილული იგივე კოდისა. არის ერთი განსხვავება, რომ

```
//აქ CreateWorkflow()-ია
WorkflowInvoker.Invoke(CreateWorkflow());
```

```
// იქ new Workflow1() -ია
WorkflowInvoker.Invoke(new Workflow1());
```

Workflow1 იყო განსაზღვრული Workflow1.xaml ფაილში, რომელიც შეიქმნა Workflow Designer-ით.

CreateWorkflow () კი არის მეთოდი, რომლის რეალიზაცია ახლა უნდა მოვახდინოთ.

➤ **სამუშაო პროცესის (workflow) განსაზღვრა**

როგორც ზემოთ აღვნიშნეთ, სამუშაო პროცესი არის ჩადგმული კლასების და მათი თვისებების ერთობლიობა. განვიხილოთ ეს საკითხი კოდის გამოყენებით. დავამატოთ Program.cs ფაილში შემდეგი მეთოდი (ლისტინგი_1.4).

```
// --- ლისტინგი_12.2 -----  
static Activity CreateWorkflow()  
{  
    Variable<int> numberBells = new Variable<int>()  
    {  
        Name = "numberBells",  
        Default = DateTime.Now.Hour  
    };  
    Variable<int> counter = new Variable<int>()  
    {  
        Name = "counter",  
        Default = 1  
    };  
    return new Sequence()  
    {  
    };  
}
```

CreateWorkflow() მეთოდი პირველად ქმნის ორ Variable<T> კლასის შაბლონს ტიპით int, სახელით numberBells და counter. ეს ცვლადები გამოიყენება სხვადასხვა ქმედებებში.

CreateWorkflow() მეთოდი გამოიყენება იმ აქტიურობაზე დასაბრუნებლად, რომელსაც ელოდება WorkflowInvoker კლასი. იგი ფაქტობრივად აბრუნებს Sequence კლასის ანონიმურ ეგზემპლარზე.

Activity კლასი არის საბაზო, რომლისგანაც იწარმოება სამუშაო პროცესის ყველა ქმედება, ასევე Sequence-ც.

ასე რომ კომპილატორი აბრუნებს Sequence ეგზემპლარს და Activity-ის, როგორც მის საბაზო კლასს.

➤ **რეალიზაცია 1-ელ დონეზე**

ამგვარად, ჩვენ განვსაზღვრეთ ცარიელი Sequence ქმედება. ეს ეკვივალენტურია ახალი სამუშაო პროცესის შექმნის, რომელსაც აქვს Sequence, ქმედებების გარეშე. ახლა განვსაზღვროთ ქმედებები ამ Sequence-ზე return new Sequence() გამოძახებით და შეცვლით 12.3 ლისტინგზე მოცემული კოდით.

```
// --- ლისტინგი_12.3 -----
return new Sequence()
{
    DisplayName = "Main Sequence",
    Variables = { numberBells, counter },
    Activities =
    {
        new WriteLine()
        {
            DisplayName = "Hello",
            Text = "Hello, World!"
        },
        new If()
        {
            DisplayName = "Adjust for PM"
            // Code to be added here in Level 2
        },
        new While()
        {
            DisplayName = "Sound Bells"
            // Code to be added here in Level 2
        }
    }
}
```

```

    },
    new WriteLine()
    {
        DisplayName = "Display Time",
        Text = "The time is: " + DateTime.Now.ToString()
    },
    new If()
    {
        DisplayName = "Greeting"
        // Code to be added here in Level 2
    }
}
};

```

ეს კოდი თავიდან განსაზღვრავს DisplayName და ობიექტის დამოკიდებულ ცვლადებს ამ ქმედებასთან. შემდეგ იგი აინიციალიზებს წევრ-ქმედებებს, როგორც ქმედებათა კოლექციას. კერძოდ, იგი ქმნის 12.1 ცხრილში მოცემულ ქმედებებს.

ცხრ.12.1

Type	DisplayName
WriteLine	“Hello”
If	“Adjust for PM”
While	“Sound Bells”
WriteLine	“Display Time”
If	“Greeting”

WriteLine ქმედებებისთვის განსაზღვრულია Text თვისებები. სხვა ქმედებებისთვის რეალიზაცია განსაზღვრულ იქნება შემდეგ დონეზე.

➤ რეალიზაცია მე-2 დონეზე

პირველი If ქმედებისთვის შევიტანოთ შემდეგი კოდი:

```

DisplayName = "Adjust for PM",

// მე-2 დონეზე დასამატებელი კოდი
Condition = ExpressionServices.Convert<bool>
    (env => numberBells.Get(env) > 12),
Then = new Assign<int>()
{
    DisplayName = "Adjust Bells"
}
// მე-3 დონეზე დასამატებელი კოდი
    
```

ეს კოდი განსაზღვრავს მდგომარეობას (Condition) და Then-ის თვისებებს (აქ არაა Else ნაწილი). Assign ქმედება იქნება რეალიზებული მომდევნო დონეზე. Condition თვისების განსაზღვრა საჭიროებს მცირე კომენტარს.

➤ გამოსახულებების შეტანა

ExpressionServices კლასის სტატიკური Convert<T>() მეთოდი გამოიყენება InArgument <T> კლასის შესაქმნელად, რასაც ელოდება მდგომარეობის (Condition) თვისება. ეს კლასები და მეთოდები იყენებს საერთო (T) ტიპს, ამიტომაც ისინი შეიძლება ნებისმიერი ტიპისთვის იქნას გამოყენებული. ამ შემთხვევაში ჩვენ უნდა გამოვიყენოთ BOOL ტიპი, ვინაიდან If ქმედების პირობის თვისება ელოდება true-ს ან false-ს.

გამოსახულების შეტანა რეალიზდება ლამბდა-გამოსახულებით (LINQ სინტაქსის ანალოგიურად), მონაცემების ამოსაღებად მუშა პროცესის გარემოდან (workflow environment). ლამბდა გამოსახულებაში „ => “ –ს უწოდებენ ლამბდა ოპერატორს. მის მარცხნივ თავსდება შემავალი პარამეტრები, ხოლო მარჯვენა მხარეს

განისაზღვრება ფაქტობრივი გამოსახულება. ENV-ის მნიშვნელობა მიიღება შესრულების დროს, როცა ის ცდილობს მდგომარეობის (Condition) შეფასებას.

ფაქტობრივად, სამუშაო პროცესი მდგომარეობის გარეშეა, იგი არ ინახავს ელემენტთა მონაცემებს. კლასის ცვლადები განსაზღვრულია მარტივი მონაცემებით. იმისათვის, რომ მივიღოთ ფაქტობრივი მონაცემები კლასის ცვლადებიდან, უნდა გამოვიყენოთ საკუთარი Get () მეთოდი. ის მოითხოვს სორტის მარკერს, რომელიც არის ActivityContext კლასი. ეს საჭიროა, რათა განვასხვავოთ მუშა პროცესის კონკრეტული ეგზემპლარი სხვებისგან, რომლებიც შეიძლება ერთდროულად იქნას გაშვებული. Get (env)-ით დაბრუნებული მნიშვნელობა შეუდარდება, არის თუა არა ის მეტი 12-ზე.

შევიტანოთ შემდეგი კოდი While ქმედებისთვის:

```
DisplayName = "Sound Bells",  
// მე-2 დონეზე დასამატებელი კოდი  
Condition = ExpressionServices.Convert<bool>  
    (env => counter.Get(env) <= numberBells.Get(env)),  
Body = new Sequence()  
{  
    DisplayName = "Sound Bell"  
    // მე-3 დონეზე დასამატებელი კოდი  
}
```

While ქმედების Condition თვისება იდენტურია If ქმედების. ის იყენებს აგრეთვე ExpressionServices კლასს InArgument<T> კლასის შესაქმნელად. აგრეთვე bool ტიპს. შემთხვევაში ის აფასებს, არის თუ არა count <= numberBells. ამ ორივე ცვლადისთვის იგი იყენებს Get(env) მეთოდს ფაქტობრივი მნიშვნელობის მისაღებად.

მეორე If ქმედებისთვის (სახელით "Greeting") შევიტანოთ შემდეგი კოდი:

```
DisplayName = "Greeting",
// მე-2 დონეზე დასამატებელი კოდი
Condition = ExpressionServices.Convert<bool>
    (env => DateTime.Now.Hour >= 18),
Then = new WriteLine() { Text = "Good Evening" },
Else = new WriteLine() { Text = "Good Day" }
```

ამ პირობისთვის (Condition) შემავალი env-პარამეტრი არ გამოიყენება, მაგრამ იგი მაინც უნდა გამოცხადდეს გამოსახულებაში. ლოგიკა იყენებს მიმდინარე დროს, რათა დავინახოთ არის თუ არა ის 6:00 PM. ორივე Then და Else თვისებისთვის იქმნება WriteLine ქმედება. ერთი ამბობს „სალამო მშვიდობისა“ (Good Evening), მეორე კი – „გამარჯობათ“ (Good Day).

➤ რეალიზაცია მე-3 დონეზე

პირველი If ქმედებისთვის (სახელით "Adjust for PM"), შექმნილია ცარიელი Assign ქმედება Then თვისებაში. შევიტანოთ შემდეგი ტექსტი ამ რეალიზაციაში:

```
DisplayName = "Adjust Bells",
// მე-3 დონეზე დასამატებელი კოდი
To = new OutArgument<int>(numberBells),
Value = new InArgument<int>(env => numberBells.Get(env) - 12)
```

➤ Assign ქმედება (დანიშვნა, მინიჭება)

Assign კლასი არის უნივერსალური (ზოგადი), ამიტომაც მას შეუძლია მონაცემთა ნებისმიერი ტიპის მხარდაჭერა. ამ შემთხვევაში ის ანიჭებს მთელ მნიშვნელობას, ამიტომ შექმნილია როგორც Assign<int>. თვისებები To და Value აგრეთვე იყენებს შაბლონურ კლასებს და უნდა შეიქმნას იგივე ტიპით (<int>).

To თვისება არის OutArgument კლასის, რომელიც იღებს კლასის ცვლადს კონსტრუქტორიდან. Value თვისება იყენებს InArgument კლასს. ის გამოყენებულ იქნა აქამდე Condition თვისების If და While -ში. მისი კონსტრუქტორისთვის გამოიყენება ლამბდა გამოსახულება, როგორც ეს იყო Condition თვისებისთვის.

➤ Sequence ქმედება

While ქმედებაში Execute თვისებისთვის შევქმენით ცარიელი Sequence. ის განსაზღვრავს შესასრულებელ ქმედებათა მიმდევრობას ციკლის შესრულების ყოველი მომენტისთვის. შევიტანოთ შემდეგი ტექსტი Activities თვისების შესავსებად:

```
DisplayName = "Sound Bell",
```

```
// მე-3 დონეზე დასამატებელი კოდი
```

```
Activities =
```

```
{
```

```
  new WriteLine()
```

```
  {
```

```
    Text = new InArgument<string>(env => counter.Get(env).ToString())
```

```
  },
```

```
  new Assign<int>()
```

```
  {
```

```
    DisplayName = "Increment Counter",
```

```
    To = new OutArgument<int>(counter),
```

```
    Value = new InArgument<int>(env => counter.Get(env) + 1)
```

```
  },
```

```
  new Delay()
```

```
  {
```

```
    Duration = TimeSpan.FromSeconds(1)
```

```
  }
```

```
}
```

ეს კოდი ამატებს სამ ქმედებას Sequence-ში:

- WriteLine აქტიურობას counter-ის გამოსატანად ეკრანზე;
- Assign აქტიურობას counter-ის ინკრემენტისთვის;
- Delay აქტიურობას მცირე პაუზისთვის იტერაციებს შორის.

ამ WriteLine ქმედებისთვის Text-ის თვისება არაა სიმბოლური string, როგორც სხვა პირობა იყო. ამ შემთხვევაში ეკრანზე გამოსატანი მნიშვნელობა განსაზღვრულია გამოსახულებაში. Text-ის თვისება ელოდება string-ს, ამიტომაც იგი ქმნის InArgument<string> კლასს. ჩვენ ამ დროს ვიყენებთ ლამბდა გამოსახულებას. Get(env) მეთოდი Variable კლასისთვის უზრუნველყოფს მიმდინარე ცვლადისთვის მთელ ტიპს (integer). ToString() მეთოდი გარდაქმნის მას სტრიქონად.

Delay ქმედებისთვის ხანგრძლივობის (Duration) თვისება გადაეცემა როგორც TimeSpan კლასი, რომელიც შექმნილია FromSeconds() სტატიკური მეთოდით.

- **Running the Application (ამუშავება)**

F5 ღილაკით ავამუშავოთ აპლიკაცია. დღე-ღამის დროისგან დამოკიდებულებით შედეგები იქნება შემდეგი:

```
Hello, World!
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
The time is: 10/5/2009 7:02:41 PM
```

```
Good Evening
```

```
Press ENTER to exit
```

რეალიზაციის პროგრამის სრული ტექსტი მოცემულია 12.4 ლისტინგში.

```
// ---- ლისტინგი_12.4 -----
using System;
using System.Activities;
using System.Activities.Statements;
using System.Activities.Expressions;
namespace Chapter02
{
    class Program
    {
        static void Main(string[] args)
        {
            WorkflowInvoker.Invoke(CreateWorkflow());
            Console.WriteLine("Press ENTER to exit");
            Console.ReadLine();
        }
        static Activity CreateWorkflow()
        {
            Variable<int> numberBells = new Variable<int>()
            {
                Name = "numberBells",
                Default = DateTime.Now.Hour
            };
            Variable<int> counter = new Variable<int>()
            {
                Name = "counter",
                Default = 1
            };
            return new Sequence()

```

```

{
  DisplayName = "Main Sequence",
  Variables = { numberBells, counter },
  Activities =
    { new WriteLine()
      { DisplayName = "Hello",
        Text = "Hello, World!"
      },
    new If()
      { DisplayName = "Adjust for PM",
        // Code to be added here in Level 2
        Condition = ExpressionServices.Convert<bool>
          (env => numberBells.Get(env) > 12),
        Then = new Assign<int>()
          { DisplayName = "Adjust Bells",
            // Code to be added here in Level 3
            To = new OutArgument<int>(numberBells),
            Value = new InArgument<int>
              (env => numberBells.Get(env) - 12)
          }
      },
    new While()
      { DisplayName = "Sound Bells",
        // Code to be added here in Level 2
        Condition = ExpressionServices.Convert<bool>
          (env => counter.Get(env) <= numberBells.Get(env)),
        Body = new Sequence()
          { DisplayName = "Sound Bell",
            // Code to be added here in Level 3
            Activities =
              { new WriteLine()
                { Text = new InArgument<string>

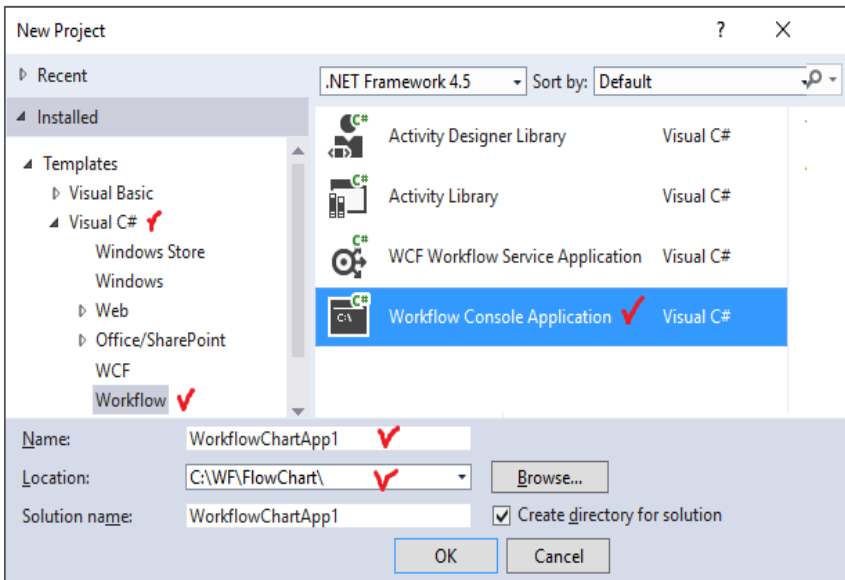
```


12.2. ბიზნესპროცესის დიაგრამა (Flowchart Workflow)

ამჯერად უნდა ავაგოთ ბიზნესპროცესი, რომელიც გამოიყენებს აქტიურობათა დიაგრამას. როგორც სათაურიდან ჩანს, ქმედებათა დიაგრამა მუშაობს როგორც ბლოკ-სქემა, ქმედებათა სახეები დაკავშირებულია ერთმანეთთან გადაწყვეტილების ხეებით (decision trees). ქმედებათა მიმდევრობითობის გამოყენებით, შვილი-პროცესები სრულდება მიმდევრობით ზემოდან-ქვევით (top-down). ამისდა მიუხედავად, შვილი-ქმედებები შეიძლება შესრულდეს ნებისმიერი მიმდევრობით, გადაწყვეტილების მიმღები პირის მიერ.

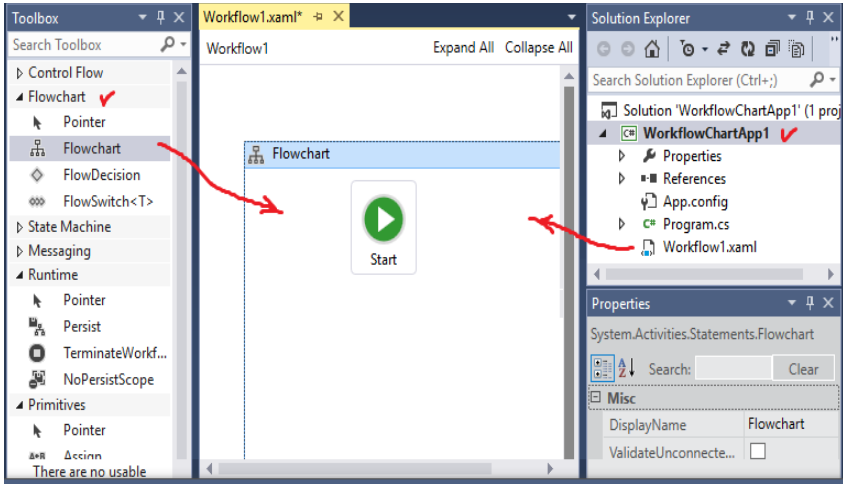
- ბიზნესპროცესის დიაგრამის შექმნა

შევქმნათ ახალი პროექტი (solution), ავირჩიოთ workflow და Console Application (ნახ.12.2).



ნახ.12.2

ინსტრუმენტების პანელიდან Flowchart გადმოვიტანოთ ფორმაზე Flowchart-ქმედება. საწყისი მუშა პროცესის დიაგრამა სასტარტო მწვანე წრით მოცემულია 12.3 ნახაზზე. ცარიელი სივრცე მის ქვეშ გამოიყენება ასაგები ბიზნესპროცესის ქმედებების დასამატებლად.

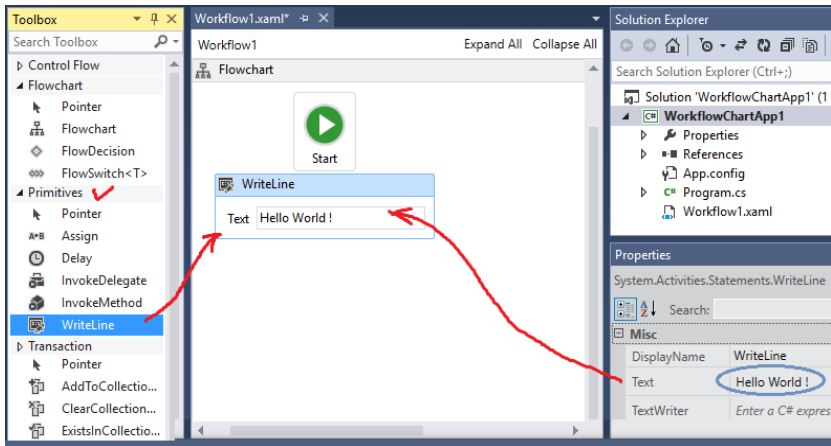


ნახ.12.3

ძირითადი განსხვავება Flowchart ქმედებასა და Sequence ქმედებას შორის ისაა, თუ როგორაა დაკავშირებული შვილი-აქტიურობები. როგორც ვიცით, ქმედებების დამატებისას მიმდევრობითობაში ისინი სრულდება დადმავალი (top-down) რიგითობით. შესაძლებელია მიმდევრობის კონტროლი (მართვა), ქმედებათა გადაადგილებით, ოღონდაც ისინი ყოველთვის გასწორებულია ვერტიკალში და განაწილებულია თანაბრად. ისრები ქმედებებს შორის კი აგებულია ავტომატურად.

Flowchart აქტიურობით შესაძლებელია ქმედებათა განთავსება პალიტრის ნებისმიერ ადგილას. მნიშვნელოვანია ისიც, რომ აქ ისრები ხელით უნდა გაკეთდეს და შეერთება დასაშვებია უკან, წინა ქმედებასთანაც !

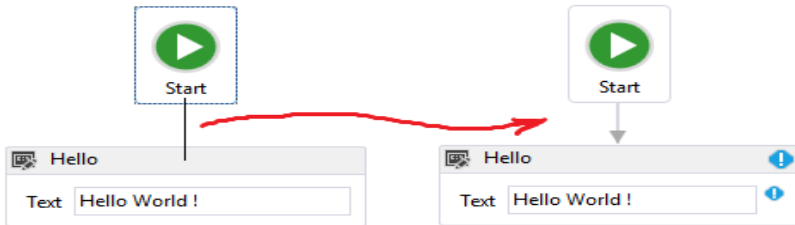
ჩვენ აპლიკაციაში დისპლეიზე უნდა გამოვიტანოთ მისაღმებები, შესაბამისად დღე-ღმის დროისა. დავიწყით სტანდარტული მისაღმების ასახვით „Hello, World“. ამისათვის გადმოვიტანოთ WriteLine ქმედება ინსტრუმენტების პანელიდან მწვანე წრის ქვეშ. დავაყენოთ DisplayName-ში “Hello” და Text-ში “Hello, World !” (ნახ.12.4).



ნახ.12.4

- მიერთების განსაზღვრა

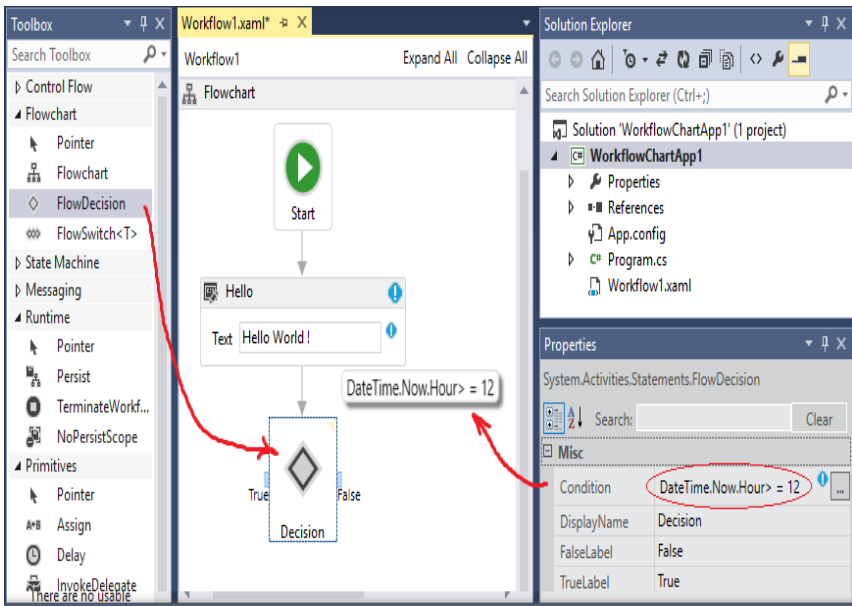
მაუსით მოვნიშნოთ სასტარტო მწვანე წრე, გავატაროთ კავშირი და ავტომატურად შეიქმნება ისარი ორ ქმედებას შორის (ნახ.12.5).



ნახ.12.5. მიერთების საწყისი და ბოლო წერტილები

- გადაწყვეტილების ნაკადი (FlowDecision)

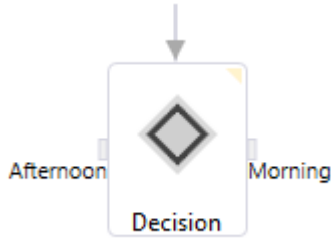
გადმოვიტანოთ სქემაზე FlowDecision ქმედება “Hello” ქმედების შემდეგ. FlowDecision ქმედება გამოიყურება ნაცრისფერი რომბის სახით, როგორც განშტოების სიმბოლო ნორმალურ ბლოკ-სქემაში, თვისებათა ფანჯარაში შევიტანოთ მდგომარეობა (Condition) `DateTime.Now.Hour >= 12`. მაუსის კურსორის მიტანით FlowDecision ქმედებაზე გამოჩნდება ასეთი სურათი (ნახ.12.6).



ნახ.12.6

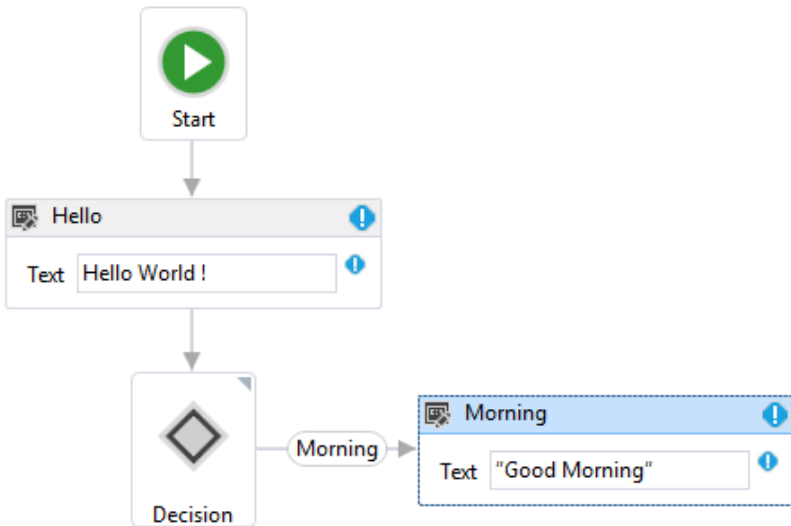
ქმედებას აქვს მარცხნიდან მიერთების წერტილი true შტოსთვის, ხოლო მარჯვნიდან – false შტოსთვის. ზედა მარჯვენა კუთხეში პატარა სამკუთხედით ჩაირთვება პირობის გამოსახულების მუდმივად გამოსაჩენი თვისება (უმაჟსოდ).

შეიძლება ტექსტის შეცვლა true/false შტოებისთვის. მაგალითად, FalseLabel-თვის შევიტანოთ Morning, და TrueLabel-თვის Afternoon. მიიღება სურათი 12.7.



ნახ.12.7

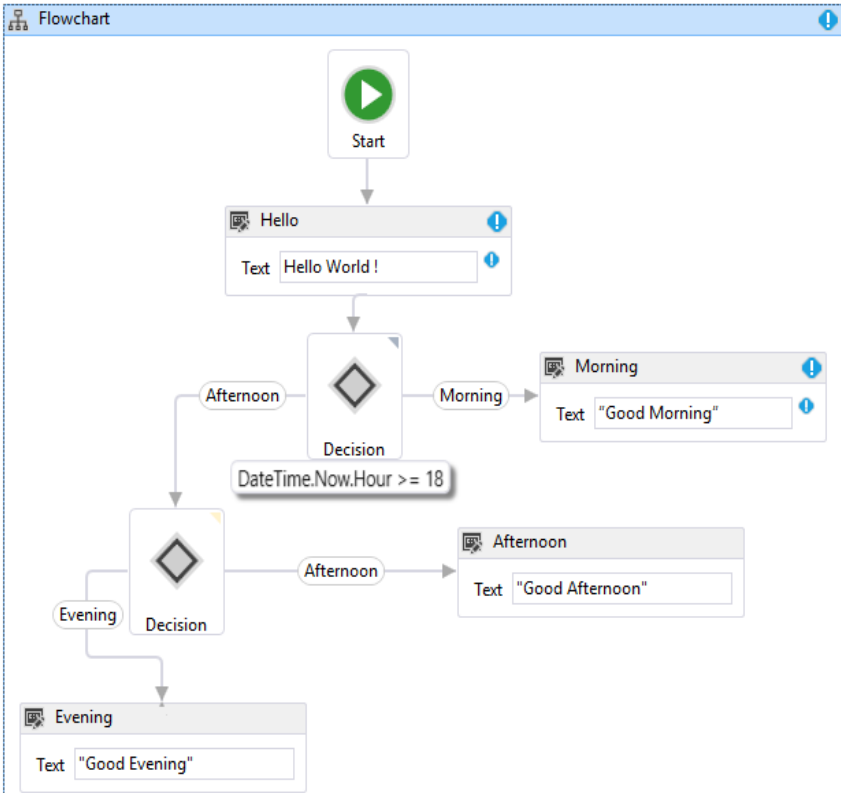
შევაერთოთ FlowDecision ქმედება მარჯვნიდან WriteLine ქმედებას, რომელშიც ჩაწერილი გვაქვს Morning/"Good Morning" (ნახ.12.8).



ნახ.12.8

FlowDecision ქმედებას არ აქვს DisplayName თვისება, მაგრამ შეუძლია პირობის ჩვენება და true/false შტოების შემოწმება და კორექტირება. ამგვარად ამ აქტიურობის მიზანი და ფუნქციონალობა ცალსახად ცხადია.

დავამატოთ FlowDecision ქმედების მარცხენა true შტოს ახალი FlowDecision ქმედება, პირობის გამოსახულებით DateTime.Now.Hour >= 18. მისი მარცხენა TrueLabel შტო იყოს Evening, ხოლო მარჯვენა – ისევ Afternoon. ორივეს მივუერთოთ ახალი WriteLine ქმედებები: Evening და Afternoon. 12.9 ნახაზზე მოცემულია ეს სურათი.

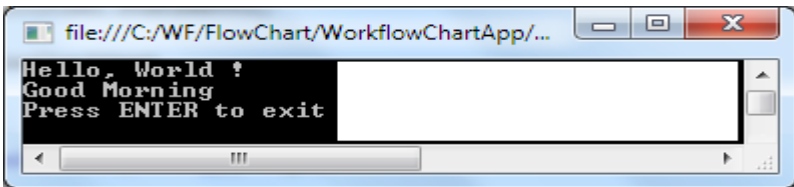


ნახ.12.9

სანამ ავამუშავებთ მიღებულ აპლიკაციას, გავხსნათ Program.cs ფაილი. ჩავამატოთ აქ შემდეგი კოდი WorkflowInvoker კლასის გამოძახების შემდეგ. ის დაგვანახებს შედეგებს პროგრამიდან გამოსვლამდე.

```
Console.WriteLine("Press ENTER to exit");  
Console.ReadLine();
```

აპლიკაციის ამუშავება: F5 (ნახ.12.10)

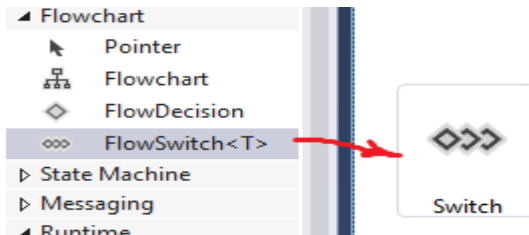


ნახ.12.10. შედეგები

შედეგი დამოკიდებულია იმაზე თუ დღის რომელ მონაკვეთში ავამუშავებთ აპლიკაციას.

- **პროცესის გადამრთვლი (Flow Switch)**

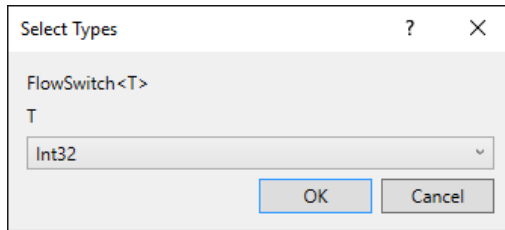
FlowSwitch ქმედება ფუნქციონირებს როგორც FlowDecision იმ გამონაკლისით, რომ არაა შეზღუდვა მხოლდ true/false ორი განშტოებით. შესაძლებელია შტოების ნებისმიერი რაოდენობის განსაზღვრა ისე, როგორც ეს იყო C# ენაში. 12.11 ნახაზზე ნაჩვენებია FlowSwitch აქტიურობის პიქტოგრამა ინსტრუმენტების პანელზე.



ნახ.12.11. პიქტოგრამა FlowSwitch<T>

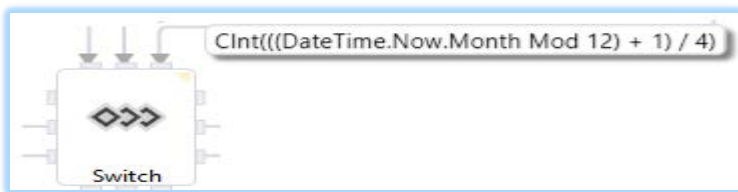
- **FlowSwitch** ქმედების დამატება

გადმოვიტანოთ ინსტრუმენტების პანელიდან FlowSwitch აქტიურობა ჩვენი ბოლო პროექტის სქემის ქვეშ. იგი არის <T> კლასის შაბლონი და უნდა მიეთითოს ტიპი. ჩვენ შემთხვევაში ესაა Int32, რომელიც ავტომატურადაა განსაზღვრული. ავირჩევთ OK-ს (ნახ.12.12).



ნახ.12.12. ტიპის განსაზღვრა

შევაერთოთ სქემის Morning, Evening და Afternoon ქმედებები FlowSwitch ქმედებას, რომელსაც აქვს ერთი თვისება გამოსახულებისთვის და იგი განსაზღვრავს გადართვის შტოს ცვლადის მნიშვნელობას (ნახ.12.13). ჩვენ პროექტში განვახორციელებთ გადამრთველის რეალიზებას მისალმების მიზნით წელიწადის დროის მიხედვით.



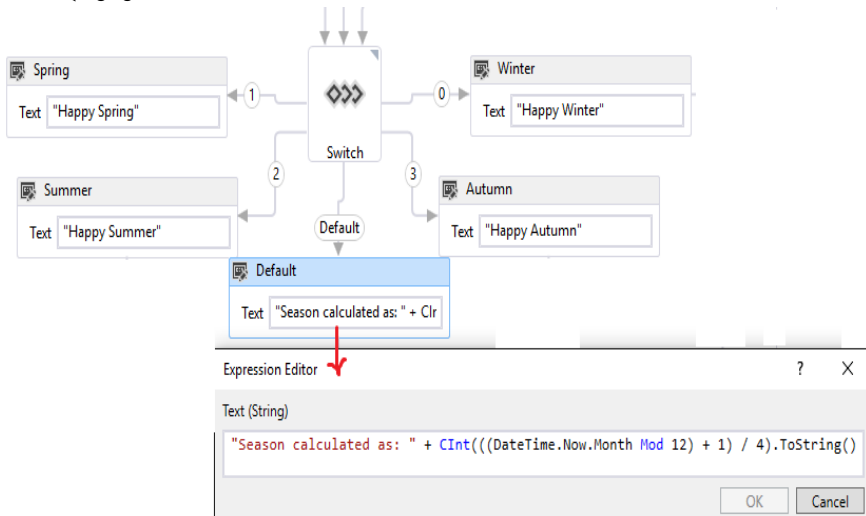
ნახ.12.13

შენიშვნა: გამოსახულების ჩაწერის ფორმა არაა დამოკიდებული გამოყენებული ენის სინტაქსზე. აქ მიღებულია, რომ ეს ფორმა იყოს Visual Basic ენის სინტაქსის მსგავსი.

ჩვენი გამოსახულება განსაზღვრავს წელიწადის დროის (ზამთარი, გაზაფხული, ზაფხული, შემოდგომა) სეზონს. მაგალითად, თუ მიმდინარე (ახლანდელი) თვე არის დეკემბერი, იანვარი ან თებერვალი, მაშინ გამოსახულება გვაძლევს 0–ს. ანალოგიურად მარტი, აპრილი და მაისი მოგვცემს 1–ს და ა.შ. გადამრთველის ოთხ შტოსთვის უნდა განვსაზღვროთ 0,1,2 და 3, ანუ სეზონები.

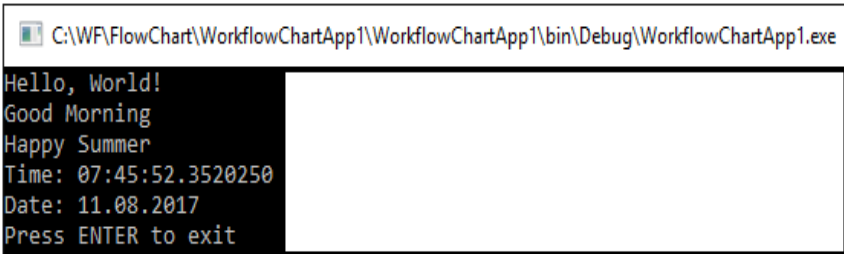
- **FlowStep ქმედების დამატება**

FlowSwitch აქტიურობის ყოველი შტო იძახებს FlowStep ქმედებას. ის Toolbox-ში არაა, ამიტომ მისი ცხადი სახით შექმნა არ შეიძლება. სქემას გადამრთველის გამოსასვლელზე უნდა დაემატოს რამდენიმე ქმედება, მაგალითად, ხუთი WriteLine და მათი შეერთების დროს მოხდება შინაგანად FlowStep–ის მნიშვნელობის განსაზღვრა. WriteLine ქმედებებისთვის ჩავწეროთ DisplayName-ში: Winter, Spring, Summer, Autumn და Default, აგრეთვე შესაბამისი მისალმებები (ნახ.12.14).



ნახ.12.14

პროგრამის ამუშავებით(F5) მივიღებთ შედეგს (ნახ.12.15).



```
C:\WF\FlowChart\WorkflowChartApp1\WorkflowChartApp1\bin\Debug\WorkflowChartApp1.exe
Hello, World!
Good Morning
Happy Summer
Time: 07:45:52.3520250
Date: 11.08.2017
Press ENTER to exit
```

ნახ.12.15. FlowSwitch აპლიკაციის შედეგები

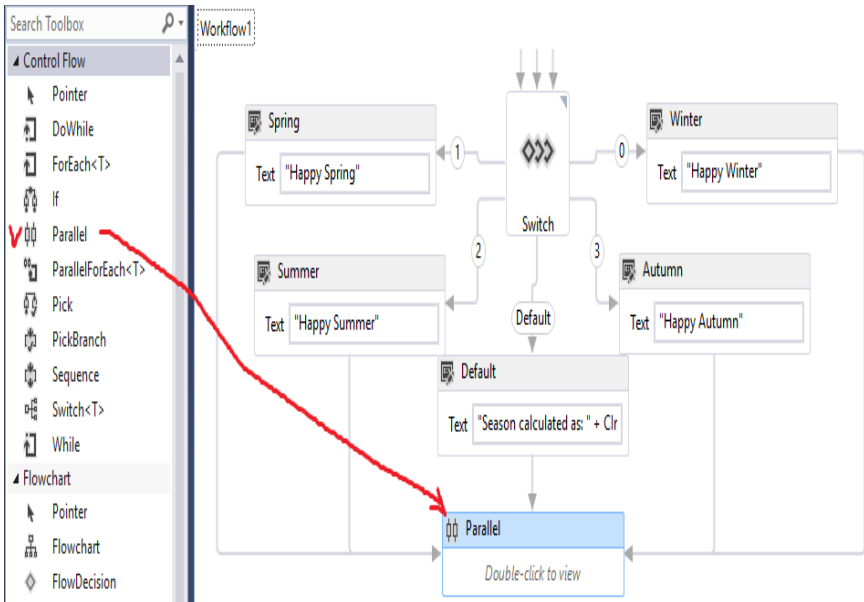
12.3. პარალელური ბიზნესპროცესები და ქმედებები

ჩვენი პროექტის ფარგლებში განვიხილოთ პარალელური პროცესების (Parallel) საკითხი, Parallel ქმედების მაგალითზე, რომელიც საშუალებას იძლევა განვსაზღვროთ Sequence ქმედებათა რაოდენობა, რომლებიც სრულდება პარალელურად (ერთდროულად).

ამ პროექტში ყოველი შტო გამოიტანს ინფორმაციის თავის ნაწილს. გამოტანის რიგითობას არ აქვს არსებითი მნიშვნელობა, მთვარია, რომ ისინი მოთავსებულია ერთ Parallel აქტიურობაში და ყველა სრულდება ერთდროულად.

- **Parallel ქმედების დამატება**

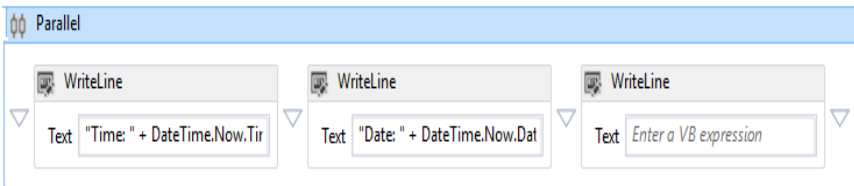
გადმოვიტანოთ Parallel აქტიურობა ჩვენი მუშა პროცესის ბოლოში. ხუთივე WriteLine-დან გავავლოთ კავშირი Parallel ქმედებასთან (ნახ.12.16).



ნახ.12.16

• შტოების დამატება

ორჯერ დავეკლივთ Parallel ქმედება და მის ზედაპირზე გადმოვიტანოთ სამი WriteLine ქმედება (ნახ.12.17).



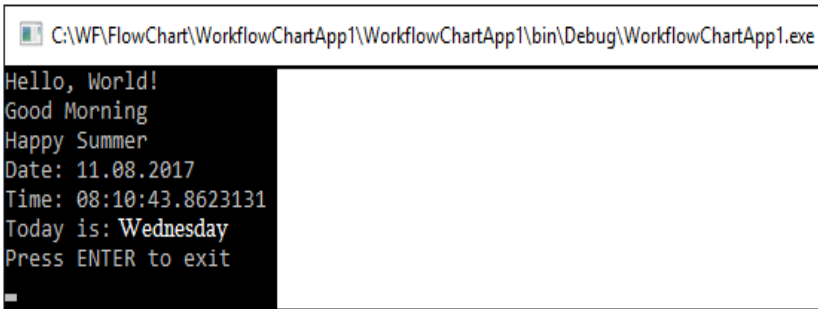
ნახ.12.17

ერთმა უნდა გამოიტანოს მიმდინარე თარიღი, მეორემ – დრო და მესამემ კვირის დღე (ნახ.12.18). მათ Text-ში გამოსახულებებისათვის (expression) შევიტანოთ შემდეგი:

```
"Date: " + DateTime.Now.Date.ToShortDateString()  
"Time: " + DateTime.Now.TimeOfDay.ToString()  
"Today is: " + DateTime.Now.ToString("dddd")
```

შენიშვნა: Parallel ქმედება ნებას იძლევა მხოლოდ ერთი ქმედება განხორციელდეს ერთ შტოში, რაც ჩვენ მაგალითზე კარგად მუშაობს. თუ გვინდა მულტი-ქმედებების გამოყენება თითოეულ შტოში, მაშინ უნდა ვიხმართ Sequence ქმედება. აქ შეიძლება მის შიგნით დავამატოთ ქმედებათა გარკვეული რაოდენობა.

პროგრამის ამუშავების შემდეგ მიიღება ასეთი შედეგები:



```
C:\WF\FlowChart\WorkflowChartApp1\WorkflowChartApp1\bin\Debug\WorkflowChartApp1.exe  
Hello, World!  
Good Morning  
Happy Summer  
Date: 11.08.2017  
Time: 08:10:43.8623131  
Today is: Wednesday  
Press ENTER to exit
```

ნახ.12.18

Workflow1.xaml პროგრამის სრული ტექსტი მოცემულია 12.5 ლისტინგში.

<-- ლისტინგი_12.5 ---- პროექტის xaml-ფაილის ტექსტი ---

```
<Activity mc:Ignorable="sap sads"  
x:Class="WorkflowChartApp1.Workflow1"  
sap:VirtualizedContainerService.HintSize="812,1015"  
mva:VisualBasic.Settings="Assembly references and imported  
namespaces for internal implementation"  
  
xmlns="http://schemas.microsoft.com/netfx/2009/xaml/activities  
"
```



```

xmlns:av="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:mv="clr-namespace:Microsoft.VisualBasic;assembly=System"
  xmlns:mva="clr-namespace:Microsoft.VisualBasic.Activities;assembly=System.Activities"
  xmlns:s="clr-namespace:System;assembly=microsoftcorlib"
  xmlns:s1="clr-namespace:System;assembly=System"
  xmlns:s2="clr-namespace:System;assembly=System.Xml"
  xmlns:s3="clr-namespace:System;assembly=System.Core"
  xmlns:s4="clr-namespace:System;assembly=System.ServiceModel"
  xmlns:sad="clr-namespace:System.Activities.Debugger;assembly=System.Activities"

xmlns:sads="http://schemas.microsoft.com/netfx/2010/xaml/activities/debugger"

xmlns:sap="http://schemas.microsoft.com/netfx/2009/xaml/activities/presentation"
  xmlns:scg="clr-namespace:System.Collections.Generic;assembly=System"
  xmlns:scg1="clr-namespace:System.Collections.Generic;assembly=System.ServiceModel"
  xmlns:scg2="clr-namespace:System.Collections.Generic;assembly=System.Core"
  xmlns:scg3="clr-namespace:System.Collections.Generic;assembly=microsoftcorlib"
  xmlns:sd="clr-namespace:System.Data;assembly=System.Data"
  xmlns:s1="clr-namespace:System.Linq;assembly=System.Core"
  xmlns:st="clr-namespace:System.Text;assembly=microsoftcorlib"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Flowchart
  sad:XamlDebuggerXmlReader.FileName="C:\WF\Chapter03\Chapter03\Workflow1.xaml"

  sap:VirtualizedContainerService.HintSize="772,935">
    <sap:WorkflowViewStateService.ViewState>
      <scg3:Dictionary x:TypeArguments="x:String, x:Object">

```

```

        <x:Boolean x:Key="IsExpanded">False</x:Boolean>
        <av:Point x:Key="ShapeLocation">380,2.5</av:Point>
        <av:Size x:Key="ShapeSize">60,75</av:Size>
        <av:PointCollection
x:Key="ConnectorLocation">410,77.5 410,107.5
410,109.5</av:PointCollection>
        <x:Double
x:Key="Width">757.87026896634893</x:Double>
        <x:Double
x:Key="Height">898.55322639006158</x:Double>
        </scg3:Dictionary>
        </sap:WorkflowViewStateService.ViewState>
        <Flowchart.StartNode>
        <x:Reference>__ReferenceID12</x:Reference>
        </Flowchart.StartNode>
        <FlowStep x:Name="__ReferenceID12">
        <sap:WorkflowViewStateService.ViewState>
        <scg3:Dictionary x:TypeArguments="x:String,
x:Object">
        <av:Point
x:Key="ShapeLocation">304.5,109.5</av:Point>
        <av:Size x:Key="ShapeSize">210,61</av:Size>
        <av:PointCollection
x:Key="ConnectorLocation">409.5,170.5 409.5,200.5
415,200.5 415,202.5</av:PointCollection>
        </scg3:Dictionary>
        </sap:WorkflowViewStateService.ViewState>
        <WriteLine DisplayName="Hello"
sap:VirtualizedContainerService.HintSize="210,61"
Text="Hello, World!" />
        <FlowStep.Next>
        <FlowDecision x:Name="__ReferenceID2"
Condition="[DateTime.Now.Hour &gt;= 12]"
sap:VirtualizedContainerService.HintSize="70,87">
        <sap:WorkflowViewStateService.ViewState>
        <scg3:Dictionary x:TypeArguments="x:String,
x:Object">
        <av:Point
x:Key="ShapeLocation">380,202.5</av:Point>
        <av:Size x:Key="ShapeSize">70,87</av:Size>

```

```

        <x:String
x:Key="FalseLabel">Morning</x:String>
        <x:String
x:Key="TrueLabel">Afternoon</x:String>
        <av:PointCollection
x:Key="FalseConnector">450,246 480,246 480,240
544.5,240</av:PointCollection>
        <av:PointCollection
x:Key="TrueConnector">380,246 235,246
235,312.5</av:PointCollection>
        </scg3:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <FlowDecision.True>
        <FlowDecision x:Name="__ReferenceID4"
Condition="[DateTime.Now.Hour >= 18]"
sap:VirtualizedContainerService.HintSize="70,87">
        <sap:WorkflowViewStateService.ViewState>
            <scg3:Dictionary
x:TypeArguments="x:String, x:Object">
                <av:Point
x:Key="ShapeLocation">200,312.5</av:Point>
                <av:Size
x:Key="ShapeSize">70,87</av:Size>
                <x:String
x:Key="FalseLabel">Afternoon</x:String>
                <x:String
x:Key="TrueLabel">Evening</x:String>
                <av:PointCollection
x:Key="TrueConnector">200,356 109.5,356
109.5,409.5</av:PointCollection>
                <av:PointCollection
x:Key="FalseConnector">270,356 480,356 480,499.8
214.4,499.8 214.4,439.3 244.4,439.3</av:PointCollection>
            </scg3:Dictionary>
        </sap:WorkflowViewStateService.ViewState>
    </FlowDecision.True>
        <FlowStep x:Name="__ReferenceID5">
            <sap:WorkflowViewStateService.ViewState>
                <scg3:Dictionary
x:TypeArguments="x:String, x:Object">

```

```

        <av:Point
x:Key="ShapeLocation">4.5,409.5</av:Point>
        <av:Size
x:Key="ShapeSize">210,61</av:Size>
        <av:PointCollection
x:Key="ConnectorLocation">109.5,470.5 109.5,500.5
337.5,500.5 337.5,542.4</av:PointCollection>
    </scg3:Dictionary>

</sap:WorkflowViewStateService.ViewState>
    <WriteLine DisplayName="Evening"
sap:VirtualizedContainerService.HintSize="210,61"
Text="Good Evening" />
    <FlowStep.Next>
        <FlowSwitch x:TypeArguments="x:Int32"
x:Name="__ReferenceID1"
Expression="[CInt(((DateTime.Now.Month Mod 12) + 1) / 4)]"
sap:VirtualizedContainerService.HintSize="70,87">
            <FlowSwitch.Default>
                <FlowStep x:Name="__ReferenceID9">

<sap:WorkflowViewStateService.ViewState>
                    <scg3:Dictionary
x:TypeArguments="x:String, x:Object">
                        <av:Point
x:Key="ShapeLocation">244.4,708.8</av:Point>
                        <av:Size
x:Key="ShapeSize">210,61</av:Size>
                        <av:PointCollection
x:Key="ConnectorLocation">349.4,769.8 349.4,799.8
350,799.8 350,813.6</av:PointCollection>
                    </scg3:Dictionary>

</sap:WorkflowViewStateService.ViewState>
                    <WriteLine DisplayName="Default"
sap:VirtualizedContainerService.HintSize="210,61"
Text="['&quot;Season calculated as: &quot; +
CInt(((DateTime.Now.Month Mod 12) + 1) / 4).ToString()]"
/>

                <FlowStep.Next>

```

```

<x:Reference>__ReferenceID0</x:Reference>
    </FlowStep.Next>
</FlowStep>
</FlowSwitch.Default>

<sap:WorkflowViewStateService.ViewState>
    <scg3:Dictionary
x:TypeArguments="x:String, x:Object">
    <av:Point
x:Key="ShapeLocation">320,542.4</av:Point>
    <av:Size
x:Key="ShapeSize">70,87</av:Size>
    <av:PointCollection
x:Key="Default">355,629.4 355,656.04 349.9,656.04
349.9,708.8</av:PointCollection>
    <av:PointCollection
x:Key="1">320,598.8 285.7,598.8 285.7,670.7
255.7,670.7</av:PointCollection>
    <av:PointCollection
x:Key="2">380,598.8 410,598.8 410,670
444.4,670</av:PointCollection>
    <av:PointCollection
x:Key="3">380,580 454.4,580</av:PointCollection>
    <av:PointCollection
x:Key="4">380,580 410,580 454.5,580</av:PointCollection>
    <av:PointCollection
x:Key="0Connector">390,585.9 420,585.9 420,579.3
474.4,579.3</av:PointCollection>
    <av:PointCollection
x:Key="2Connector">320,607.65 290,607.65 290,670
254.5,670</av:PointCollection>
    <av:PointCollection
x:Key="3Connector">390,607.65 420,607.65 420,669.8
444.4,669.8</av:PointCollection>
    <av:PointCollection
x:Key="1Connector">320,585.9 290,585.9 290,580
244.5,580</av:PointCollection>
    </scg3:Dictionary>
</sap:WorkflowViewStateService.ViewState>

```

```

        <FlowStep x:Key="0"
x:Name="__ReferenceID11">
    <sap:WorkflowViewStateService.ViewState>
        <scg3:Dictionary
x:TypeArguments="x:String, x:Object">
            <av:Point
x:Key="ShapeLocation">474.4,548.8</av:Point>
            <av:Size
x:Key="ShapeSize">210,61</av:Size>
            <av:PointCollection
x:Key="ConnectorLocation">684.4,579.3 714.4,579.3
714.4,840.1 450,840.1</av:PointCollection>
        </scg3:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <WriteLine DisplayName="Winter"
sap:VirtualizedContainerService.HintSize="210,61"
Text="Happy Winter" />
    <FlowStep.Next>
    <FlowStep
x:Name="__ReferenceID0">
    <sap:WorkflowViewStateService.ViewState>
        <scg3:Dictionary
x:TypeArguments="x:String, x:Object">
            <av:Point
x:Key="ShapeLocation">250,813.6</av:Point>
            <av:Size
x:Key="ShapeSize">200,51</av:Size>
        </scg3:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <Parallel
sap:VirtualizedContainerService.HintSize="200,51">
        <WriteLine
sap:VirtualizedContainerService.HintSize="211.2,62.4"
Text="['&quot;Time: &quot; +
DateTime.Now.TimeOfDay.ToString()]" />
        <WriteLine
sap:VirtualizedContainerService.HintSize="211.2,62.4"

```

```

Text="[&quot;Date: &quot; +
DateTime.Now.Date.ToShortDateString()" />
        </Parallel>
        </FlowStep>
        </FlowStep.Next>
    </FlowStep>
    <FlowStep x:Key="1"
x:Name="__ReferenceID7">

    <sap:WorkflowViewStateService.ViewState>
        <scg3:Dictionary
x:TypeArguments="x:String, x:Object">
            <av:Point
x:Key="ShapeLocation">34.5,549.5</av:Point>
            <av:Size
x:Key="ShapeSize">210,61</av:Size>
            <av:PointCollection
x:Key="ConnectorLocation">34.5,580.7
6.399999999999989,580.7 6.399999999999989,840
250,840</av:PointCollection>
        </scg3:Dictionary>

    </sap:WorkflowViewStateService.ViewState>
        <WriteLine DisplayName="Spring"
sap:VirtualizedContainerService.HintSize="210,61"
Text="Happy Spring" />
        <FlowStep.Next>

    <x:Reference>__ReferenceID0</x:Reference>
        </FlowStep.Next>
        </FlowStep>
        <FlowStep x:Key="2"
x:Name="__ReferenceID8">

    <sap:WorkflowViewStateService.ViewState>
        <scg3:Dictionary
x:TypeArguments="x:String, x:Object">
            <av:Point
x:Key="ShapeLocation">44.5,639.5</av:Point>
            <av:Size
x:Key="ShapeSize">210,61</av:Size>

```

```

                <av:PointCollection
x:Key="ConnectorLocation">149.5,700.5 149.5,840.1
250,840.1</av:PointCollection>
                </scg3:Dictionary>

</sap:WorkflowViewStateService.ViewState>
                <WriteLine DisplayName="Summer"
sap:VirtualizedContainerService.HintSize="210,61"
Text="Happy Summer" />
                <FlowStep.Next>

<x:Reference>__ReferenceID0</x:Reference>
                </FlowStep.Next>
                </FlowStep>
                <FlowStep x:Key="3"
x:Name="__ReferenceID10">

<sap:WorkflowViewStateService.ViewState>
                <scg3:Dictionary
x:TypeArguments="x:String, x:Object">
                <av:Point
x:Key="ShapeLocation">444.4,638.8</av:Point>
                <av:Size
x:Key="ShapeSize">210,61</av:Size>
                <av:PointCollection
x:Key="ConnectorLocation">549.4,699.8 549.4,840.1
450,840.1</av:PointCollection>
                </scg3:Dictionary>

</sap:WorkflowViewStateService.ViewState>
                <WriteLine DisplayName="Autumn"
sap:VirtualizedContainerService.HintSize="210,61"
Text="Happy Autumn" />
                <FlowStep.Next>

<x:Reference>__ReferenceID0</x:Reference>
                </FlowStep.Next>
                </FlowStep>
                </FlowSwitch>
                </FlowStep.Next>
</FlowStep>

```



```

        <WriteLine DisplayName="Morning"
sap:VirtualizedContainerService.HintSize="210,61"
Text="Good Morning" />
        <FlowStep.Next>
            <x:Reference __ReferenceID1</x:Reference>
        </FlowStep.Next>
    </FlowStep>
    </FlowDecision.False>
</FlowDecision>
</FlowStep.Next>
</FlowStep>
<x:Reference __ReferenceID2</x:Reference>
<x:Reference __ReferenceID3</x:Reference>
<x:Reference __ReferenceID4</x:Reference>
<x:Reference __ReferenceID5</x:Reference>
<x:Reference __ReferenceID6</x:Reference>
<x:Reference __ReferenceID1</x:Reference>
<x:Reference __ReferenceID7</x:Reference>
<x:Reference __ReferenceID8</x:Reference>
<x:Reference __ReferenceID9</x:Reference>
<x:Reference __ReferenceID10</x:Reference>
<x:Reference __ReferenceID11</x:Reference>
<x:Reference __ReferenceID0</x:Reference>
</Flowchart>
</Activity>

```

შენიშვნა: პროგრამის კოდი გადმოტანილია მუშა პროექტიდან უცვლელად. დიზაინერი მუშაობს ვიზუალურ გარემოში (Toolbox და Properties), ხოლო თვით WF-სისტემა წერს კოდის ძირითად ნაწილს.

ამ მაგალითში ძირითადი დატვირთვა აქვს xaml-ფაილს, ხოლო C# -ის ნაწილი მცირეა (ლისტინგი 12.6).

//-- ლისტინგი_12.6 -----

```

using System;
using System.Linq;
using System.Activities;

```

```
using System.Activities.Statements;

namespace WorkflowChartApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            WorkflowInvoker.Invoke(new Workflow1());

            Console.WriteLine("Press ENTER to exit");
            Console.ReadLine();
        }
    }
}
```

Workflow Foundation .NET პაკეტში რეალიზებულია რევერსიული დაპროგრამების (დაპროგრამების ავტომატიზაციის) მაღალი დონე, ამიტომაც მომხმარებელი პროგრამისტი სწრაფად აგებს მისთვის საჭირო კოდს.

ლიტერატურა:

1. ჩოგვაძე გ., ფრანგიშვილი ა., სურგულაძე გ. მართვის საინფორმაციო სისტემების დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი. მონოგრ., სტუ, „ტექნიკური უნივერსიტეტი“, თბ., 2017, -1001 გვ.
2. სურგულაძე გ., ურუშაძე ბ. საინფორმაციო სისტემების მენეჯმენტის საერთაშორისო გამოცდილება (BSI, ITIL, COBIT). სახელმძღვ., სტუ. „ტექნიკური უნივერსიტეტი“. თბ., 2014. - 320 გვ.
3. სურგულაძე გ. ვიზუალური დაპროგრამება C#_2010 ენის ბაზაზე. სახელმძღვანელო. სტუ. „ტექნიკური უნივერსიტეტი“, თბ., 2011. -445 გვ.
4. სურგულაძე გ. კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი: Workflow ტექნოლოგია (ნაწ.2). სტუ, თბ. „IT კონსალტინგის ცენტრი“, 2015. -136 გვ.
5. სურგულაძე გ. კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი: WPF ტექნოლოგია (ნაწ.1). სტუ, თბ. „IT კონსალტინგის ცენტრი“ 2014. -202 გვ.
6. სურგულაძე გ. კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი: WCF ტექნოლოგია (ნაწ.3). სტუ, თბ., „IT კონსალტინგის ცენტრი“, 2016. -154 გვ.
7. სურგულაძე გ., თურქია ე. პროგრამული სისტემების მენეჯმენტის საფუძვლები. სტუ, „ტექნიკ. უნივ“. თბ., 2016. -350 გვ.
8. Collins M.J. Beginning WF: Windows Workflow in .NET 4.0. ISBN-13 (pbk): 978-14302-2485-3 Copyright © 2010. USA. <http://www.ebooks-it.net/ebook/beginning-wf>.
9. Booch G., Jacobson I., rambaugh J. (1996). Unified Modeling Language for Object-Oriented Development. Rational Software Corporation, Santa Clara.

10. გოგიჩაიშვილი გ., ბოლხი გ., სურგულაძე გ., პეტრიაშვილი ლ. მართვის ავტომატიზებული სისტემების ობიექტ-ორიენტირებული დაპროექტების და მოდელირების ინსტრუმენტები (MsVisio, WinPepsy, PetNet, CPN). სახელმძღვ., სტუ. თბ., „ტექნიკური უნივერსიტეტი“. 2013, -232 გვ.

11. სურგულაძე გ., გულიტაშვილი მ., კვიციანი ნ. Web-სისტემების ტესტირება, ვალიდაცია და ვერიფიკაცია. მონოგრ. ISBN 9789941-0-7682-4. სტუ. „IT-კონსალტინგის ცენტრი“. თბილისი. 2015. -205 გვ.

12. სურგულაძე გ., თურქია ე. პროგრამული სისტემების მენეჯმენტის საფუძვლები. სახელმძღვ., სტუ, „ტექნიკური უნივერსიტეტი“. თბ., 2016. - 350 გვ.

13. სურგულაძე გ., ბულია ი. კორპორაციულ Web-აპლიკაციათა ინტეგრაცია და დაპროექტება. სტუ, „ტექნიკური უნივერსიტეტი“, თბ., 2012. -324 გვ.

14. Мак-Дональд М. WPF: Windows Presentation Foundation в .NET 3.5 с примерами на С# 2008 для профессионалов. 2-е издание: Пер. с англ. - М. : ООО "И.Д. Вильямс". 2008

15. Petzold Ch. Applications=Code+Markup. A Guide to the MicroSoft Windows Presentation Foundation. St-Petersburg. 2008

იბეჭდება ავტორთა მიერ წარმოდგენილი სახით

გადაეცა წარმოებას 05.06.2017. ხელმოწერილია დასაბეჭდად 22.06.2017. ქალაქის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 14,5.

საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“ ,
თბილისი, კოსტავას 77

