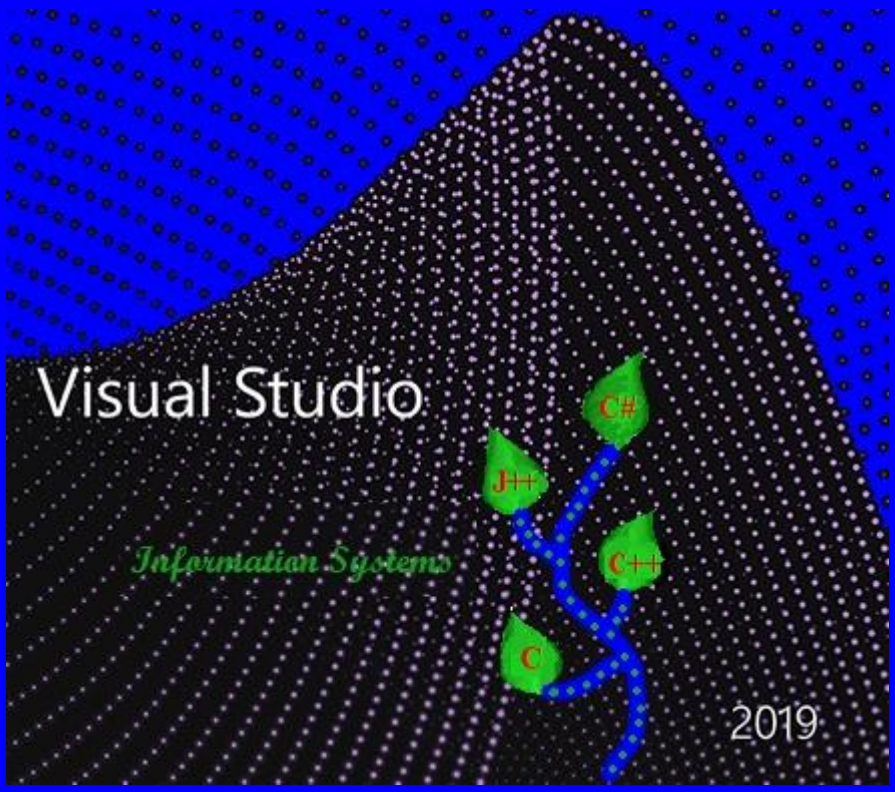
გია სურგულაძე, ლილი პეტრიაშვილი

GTU

ვიზუალური დაპროგრამება C# ენის ბაზაზე ინფორმაციული სისტემებისათვის



სტუ-ს „IT კონსალტინგის ცენტრი“

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე, ლილი პეტრიაშვილი

ვიზუალური დაპროგრამება C# ენის ბაზაზე ინფორმაციული სისტემებისათვის

(Ms Visual Studio.NET 2019 პლატფორმაზე)

დამტკიცებულია:

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრის“ სარედაქციო კოლეგიის მიერ

თბილისი
2019

უაკ 004.5

განხილულია ვიზუალური დაპროგრამების თეორიული და პრაქტიკული ამოცანები და მათი გადაწყვეტის ინსტრუმენტული საშუალებები მაიკროსოფთის ახალი პროგრამული პლატფორმის Visual Studio.NET 2019-ის ბაზაზე. წარმოდგენილია ამ პლატფორმის შექმნისა და განვითარების ეტაპები და ძირითადი ტექნოლოგიური გადაწყვეტები; ინფორმაციული და მართვის სისტემების ამოცანების დეველოპმენტისა და პროგრამული აპლიკაციების აგების კლასიკური ვიზუალური კომპონენტები; მათი ტესტირებისა და გამონაკლისი შემთხვევების ანალიზის პროცესების დამახასიათებელი სირთულეები და გადაწყვეტის რეკომენდაციები.

დამხმარე სახელმძღვანელო გამიზნულია ინფორმატიკის, მართვის საინფორმაციო სისტემების სპეციალობის სტუდენტებისა და ამ საკითხებით დაინტერესებული მკითხველისათვის.

რეცენზენტები:

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

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

ISBN 978-9941-8-1708-3

სარჩევი

შესავალი

21-ე საუკუნე საინფორმაციო სისტემებისა და ტექნოლოგიების სწრაფი განვითარების ტემპით ხასიათდება, რომელსაც ხელს უწყობს ერთი მხრივ, ახალი თაობის ქსელური და კომპიუტერული ინდუსტრიის აღმავლობა და, მეორე მხრივ, ცივილიზებული ინფორმაციული საზოგადოების თანდათანობითი ფორმირება მთელ მსოფლიოში [1]. სულ უფრო ფართოვდება კომპიუტერული სისტემების მომხმარებელთა წრე ქალაქსა და სოფელში, სკოლებსა და უნივერსიტეტებში, სახელმწიფო ორგანიზაციებსა და კერძო სტრუქტურებში, რაც აქტუალურს ხდის ამ სფეროში მომხმარებელთა მეგობრული ინტერფეისების დამუშავებას, მობილური და სერვისული სისტემების მეთოდური რეკომენდაციებისა და ინსტრუმენტული საშუალებების შექმნას და განვითარებას [1,2].

წინამდებარე სახელმძღვანელოში გადმოცემულია ვიზუალური C#.NET ენის საფუძვლები და მისი გამოყენების პრაქტიკული მაგალითები ინფორმაციული სისტემების პროგრამული დეველოპმენტის მიმართულებით ვიზუალური დაპროგრამების კლასიკური მართვის ელემენტებით [3,34].

განიხილება მაიკროსოფთის Visual Studio.NET პლატფორმის შექმნისა და განვითარების ისტორია, ობიექტორიენტირებული დაპროგრამების მეთოდის ძირითადი არსი და მნიშვნელობა, აქტუალური ენების, განსაკუთრებით C#-ის კომპონენტების პროგრამული რეალიზაციის ვიზუალური საშუალებანი. ასევე წარმოდგენილია მომხმარებელთა ინტერფეისების დაპროექტებისა და მათი რეალიზებული ეგზემპლარების კავშირები მონაცემთა ბაზების სისტემებთან, განსაკუთრებით MsSQL Server პაკეტთან. წარმოდგენილია შესაბამისი საილუსტრაციო ამოცანები. აქ მნიშვნელოვანი ადგილი ეთმობა აგრეთვე C# ენის ვიზუალური ელემენტების გამოყენების საკითხებს გრაფიკული და ანიმაციური დაპროგრამების რეჟიმებისათვის.

წიგნში ორიენტაცია მიმართულია სტუდენტების მოსამზადებლად პროგრამირების ახალი ტექნოლოგიებისთვის, როგორებიცაა WPF (Windows Presentation Foundation) და WCF (Windows Communication Foundation) სისტემები. მათ საფუძველზე კი კორპორაციული საინფორმაციო სისტემების აგება Desktop- და Web აპლიკაციებისათვის ASP.NET MVC, .NET.Framework, .NET.Core და .NET Code პლატფორმებზე და ა.შ. [5,6,14,15], რომელთა შესწავლა ცალკე თემაა და შემდეგ ეტაპებზე ხორციელდება.

პირველ თავში გადმოცემულია Ms Visual Studio .NET Framework პლატფორმის ძირითადი კონცეფცია და განვითარების დეტალური ანალიზი. შესავალი C#-ის ვიზუალური დაპროგრამების სტრუქტურაში, მის ელემენტებსა და გამოყენების შესაძლებლობებში საინფორმაციო სისტემების (გამოყენებითი პროგრამული აპლიკაციების) შექმნის, განვითარებისა და ექსპლუატაციის თვალსაზრისით.

მეორე თავის ეხება C# ენის ლაბორატორიული პრაქტიკუმის ამოცანების განხილვას Visual Studio.NET 2019 პლატფორმაზე. წარმოდგენილია პრაქტიკული ამოცანები, როგორც თითოეული ვიზუალური ელემენტის განხილვით, ასევე მათი ინფორმაციულ სისტემაში გამოყენებით, ინჟინრული გადაწყვეტისა და პროგრამული რეალიზაციის საკითხებით [2,7,10].

მესამე თავში მოცემულია C#.NET 8.0-ის სტანდარტული დიალოგების ვიზუალური კომპონენტები და მთავარი, გრაფიკული და კონტექსტური მენიუების აგების ვიზუალური შესაძლებლობები, მომხმარებელთა მეგობრული ინტერფეისების დაპროგრამების მიზნით.

მეოთხე თავში ასახულია C# პროგრამების კავშირი მონაცემთა რელაციური ბაზების SQL Server პაკეტთან. განხილულია ADO.NET დრაივერის დანიშნულება და მისი ძირითადი ობიექტები. პრაქტიკული ამოცანები ეხება ინფორმაციული სისტემების ამოცანების გადაწყვეტას.

მეხუთე თავში წარმოდგენილია .NET პლატფორმაზე ცალკეული საინჟინრო და სამომხმარებლო ამოცანების დასმა და გადაწყვეტა ინფორმაციული სისტემების პროგრამული აპლიკაციების აგების თვალსაზრისით [1, 13]. სტუდენტები ეცნობიან .NET-ის ძირითადი კონცეფციის - პროგრამირებაში საერთო ტიპების სისტემის (CTS) გამოყენების ამოცანას, .dll ფაილების შექმნას საერთო ბიბლიოთეკისათვის. აქვე განხილულია UML უნიფიცირებული მოდელების პროგრამირების საკითხები [9,12,35]. დასასრულ, შემოთავაზებულია პროგრამული პროდუქტების unit-ტესტირების ამოცანა და პროგრამებში გამონაკლისი შემთხვევების ანალიზისა და მისი აღმოფხვრის საკი8თხები, რაც ხელს უწყობს პროგრამული აპლიკაციების ხარისხის გაუმჯობესებას [11].

დასასრულ შეიძლება აღვნიშნოთ, რომ განათლების სფეროს პრიორიტეტულად გამოცხადება ჩვენ ქვეყანაში და მათ შორის საინფორმაციო საზოგადოების ფორმირების უწყვეტი პროცესი განსაკუთრებით შეუწყობს ხელს ჩვენი ქვეყნის ეკონომიკური, ტექნიკური, სოციალური და კულტურული სფეროების მდგრად განვითარებას [37,38].

I თავი

Ms Visual Studio .NET პლატფორმა: კონცეფცია, ისტორიული განვითარების ეტაპები და თანამედროვე მდგომარეობა

1.1. Ms Visual Studio .NET Framework ინტეგრირებული გარემო და ფუნქციონირების კონცეფცია

Ms Visual Studio არის მაიკროსოფთის კორპორაციის IDE (Integrated Development Environment) პროდუქტი, პროგრამული უზრუნველყოფის დამუშავების ინტეგრირებული გარემო (პროგრამულ და სხვა ინსტრუმენტულ საშუალებათა კომპლექსი). იგი გამოიყენება კომპიუტერული პროგრამების, აგრეთვე ვებ-საიტების, ვებ-პროგრამების, ვებ-სერვისებისა და მობილური პროგრამების შესამუშავებლად. სისტემა დაწერილია C++/C# ენების გამოყენებით. პირველი ვერსია გამოვიდა 1997-ში (1.0.0), ბოლო ვერსია - 2019 25 სექტემბერი (16.3.1) [16].

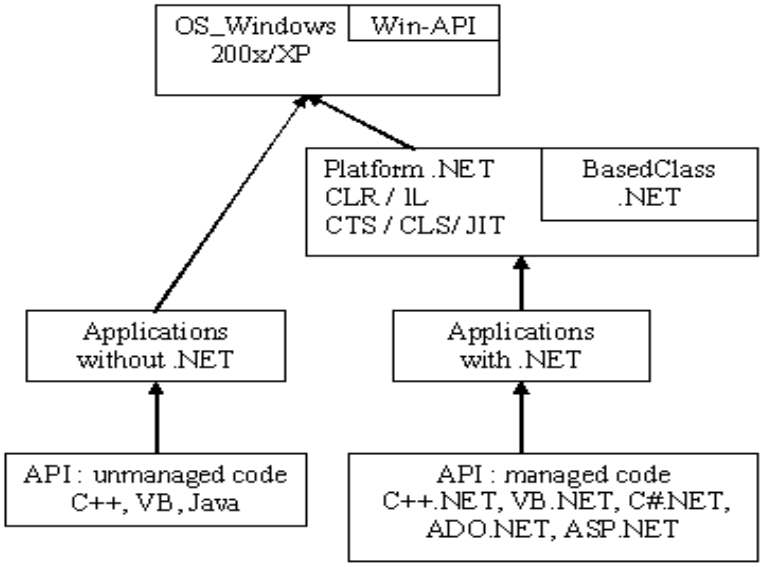
Ms Visual Studio.NET უზრუნველყოფს დესკტოპ- და ვებ-დანართების (Windows და Web-აპლიკაციების) განვითარებას და გამოყენებას.

.NET პროგრამული პლატფორმის უმნიშვნელოვანესი ნაწილია CLR (Common Language Runtime) და CTS (Common Type System), რომელთა ბაზაზეც ხორციელდება პროგრამების ფუნქციონირების პროცესების მართვა და პლატ-ფორმის სხვადასხვა ენებისათვის მონაცემთა საერთო ტიპების სისტემის მხარდაჭერა IL (Intermediate Language) ენის საფუძველზე.

.NET პლატფორმა უზრუნველყოფილია კლასების მდიდარი ბიბლიო-თეკით, 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 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 პროგრამი-რების 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). ამასთანავე იგი მუშაობს C#.NET, C++.NET, VB.NET და ა.შ [12].



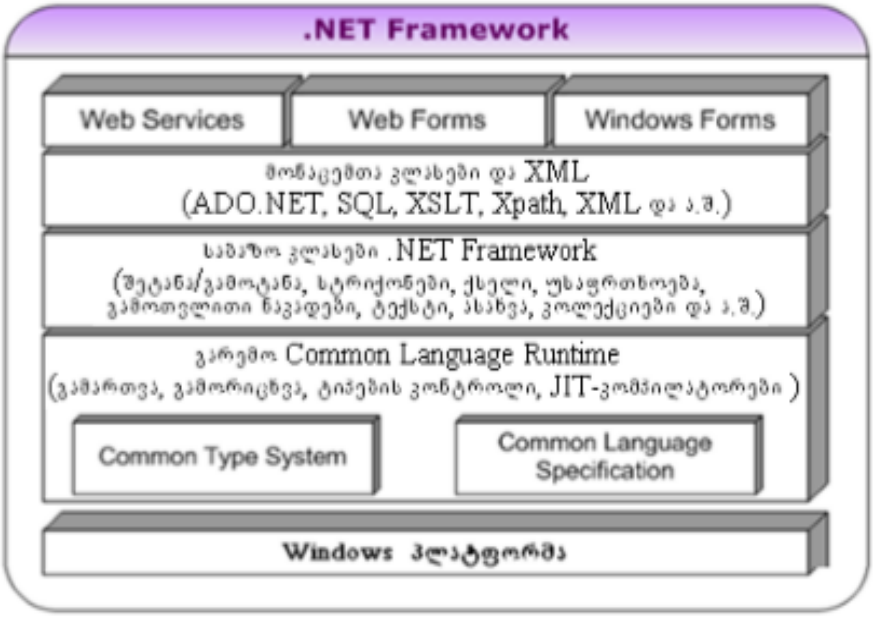
ნახ.1.1. .NET პლატფორმის კონცეფცია

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 მუშაობს Windows-თან.

1.2. .NET პლატფორმის კომპონენტები და მისი აპლიკაციების სტრუქტურა

.NET-პლატფორმა სრულად ობიექტორიენტირებულია, 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-ის ზოგადი არქიტექტურა.



ნახ.1.2. .NET-ის ზოგადი არქიტექტურა

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.

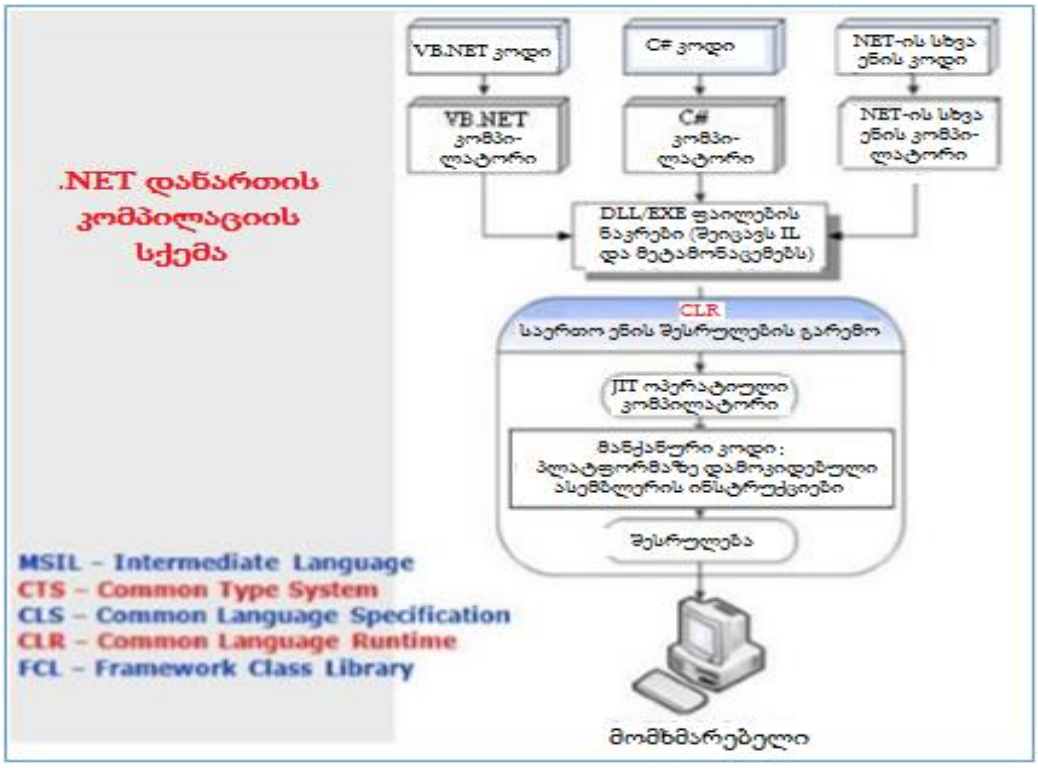
JIT (Just-In-Time) ესაა შუალედური კოდის კომპილაციის ფაზა მანქანურ კოდში. სახელწოდება მიუთითებს იმაზე, რომ კოდის მხოლოდ იმ ცალკეული ნაწილების კომპილაცია ხდება, რომლებიც საჭიროა პროგრამის შესასრულებლად დროის მოცემულ მომენტში. .NET Framework -ში გამოიყენება 2-ეტაპიანი კომპილაცია:

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 სპეციტეგებით. 1.3 ნახაზზე ნაჩვენებია სამომხმარებლო აპლიკაციის კომპილაციის სქემა .NET პლატფორმაზე.



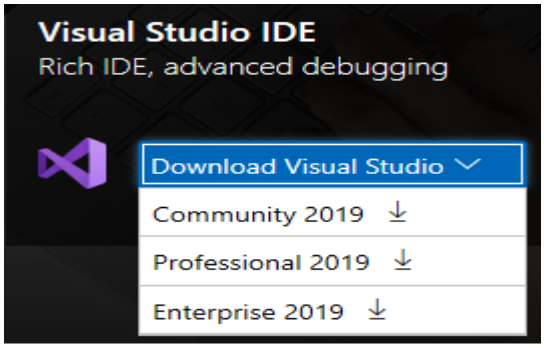
ნახ.1.3. აპლიკაციის კომპილაციის სქემა

1.3. Visual C#.NET 2019

ვიზუალური, ობიექტორიენტირებული ენა Visual C# 8.0 წარმოადგინა მაიკროსოფთმა 2019 წ. სექტემბერში, Visual Studio 16.3.0-ის პაკეტში NET Framework 4.8-თან, Visual Basic.NET, Visual C++.NET, F#.NET და სხვა ენებთან ერთად [16,17].

Visual C# 2019 (შემდგომში C#) ენა მთლიანად მოიცავს მის წინამორბედს - Visual C# 2010-17 და ახალ გაფართოებებს.

შესავალში აღწერილია ვიზუალური C# ენის საწყისები და პირველი პროგრამული კოდის აგების ელემენტები ვინდოუს-აპლიკაციისათვის. პრაქტიკული ექსპერიმენტებისათვის შესაძლებელია Visual Studio .NET-ის უფასო ან ლიცენზირებული პაკეტის ჩამოტვირთვა (ნახ.1.4):



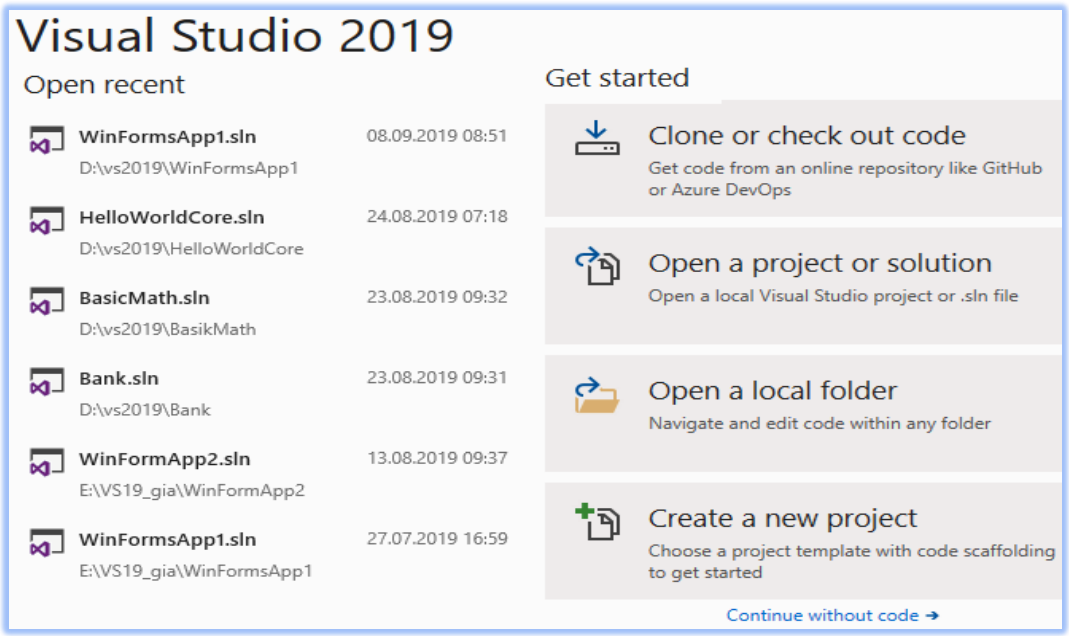
ნახ.1.4. Visual Studio.NET 2019: https://visualstudio.microsoft.com/

სისტემის ინსტალაციის შემდეგ, მისი ამუშავების საწყისი ლოგო ნაჩვენებია 1.5 ნახაზზე.

ნახ.1.5. Visual Studio.NET 2019-ის ამუშავების დასაწყისი

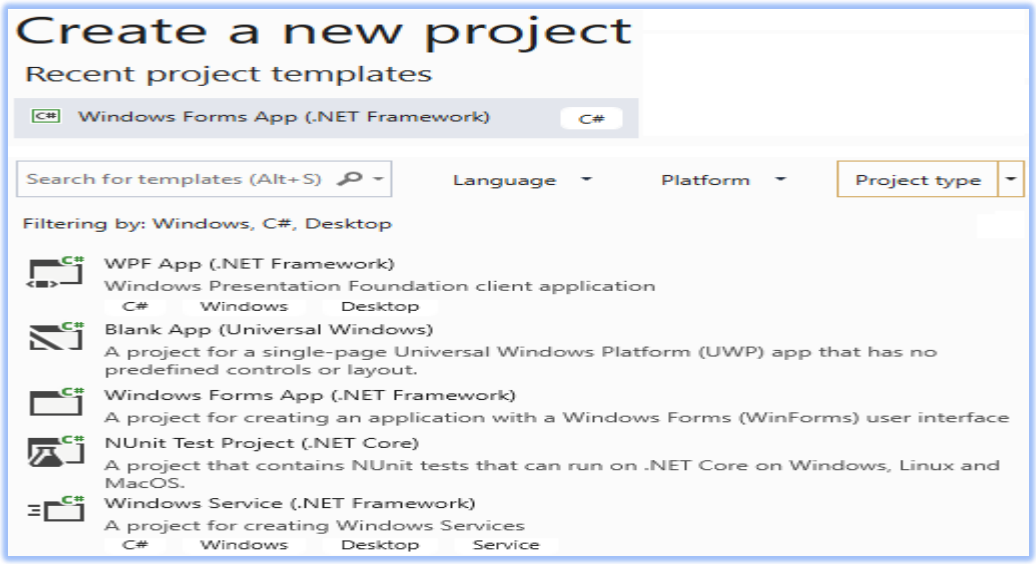
იგი მოიცავს ტექსტურ რედაქტორს პროგრამული კოდის ასაგებად, კომპილატორს - მის გასამართად და დებაგერს - შეცდომების აღმოსაჩენად.

1.6-ა,ბ,გ ნახაზებზე ნაჩვენებია ახალი პროექტის შექმნისას მისი საწყისი მონაცემების (ენა, აპლიკაციის ტიპი, სახელი, მდებარეობა) განსაზღვრა.

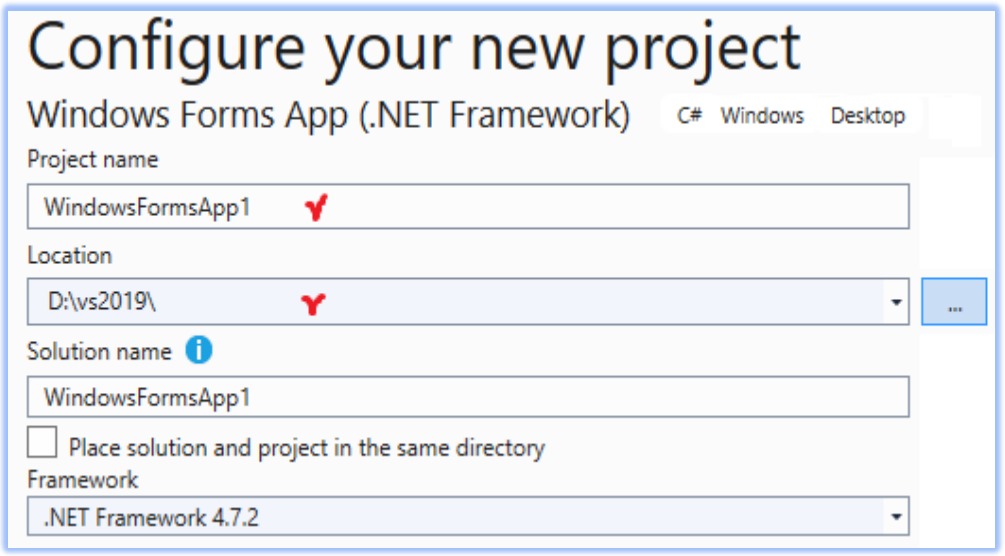


ნახ. 1.6-ა. Visual Studio.NET 2019 საწყისი გვერდი

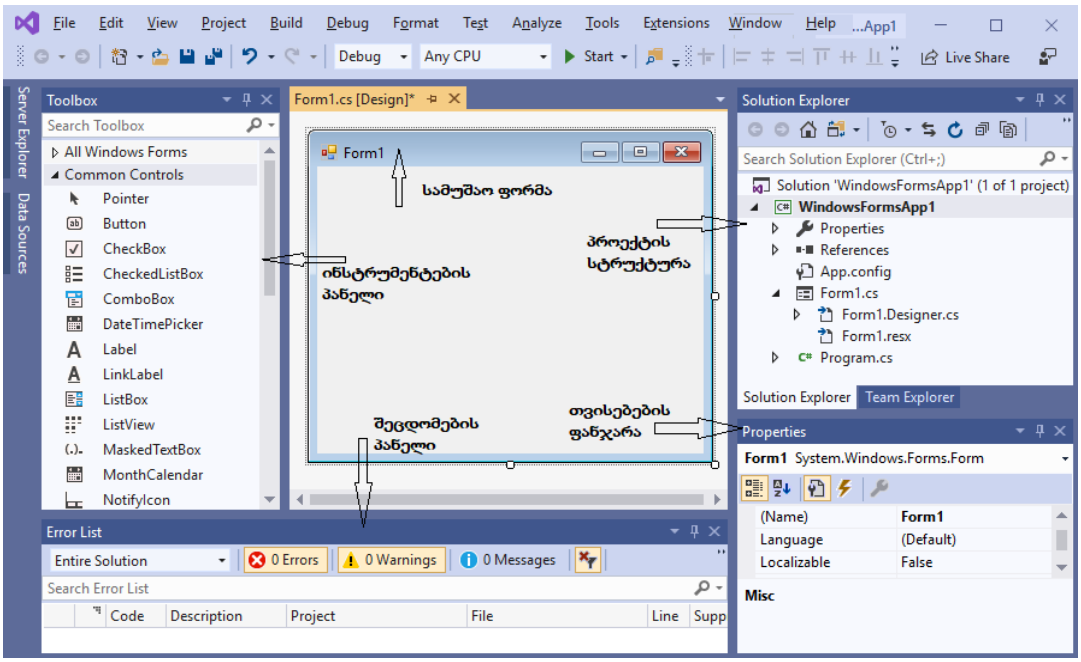
ბოლოს Create ღილაკით გადავდივართ 1.7 ნახაზზე, რომელიც სამუშაო გარემო ანუ პროგრამისტის ინტერფეისია.



ნახ. 1.6-ბ. ახალი პროექტის ტიპის არჩევის ფანჯრის ფრაგმენტი

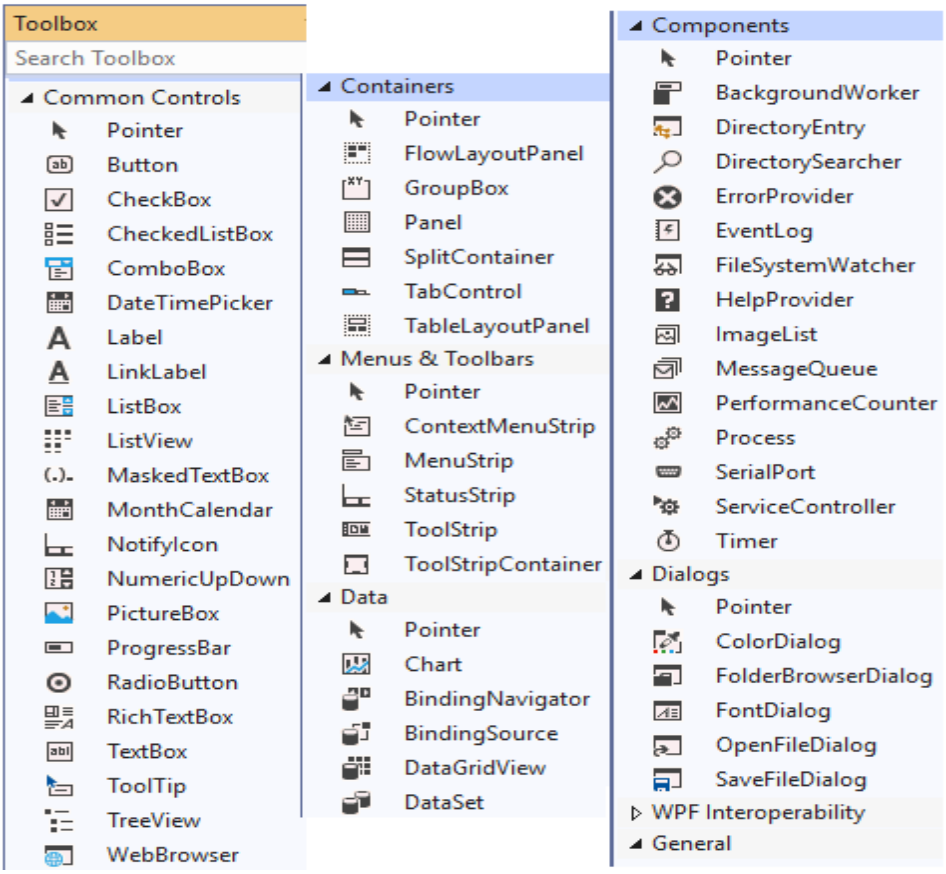


ნახ. 1.6-გ. ახალი პროექტის სახელის და ლოკაციის შერჩევა



ნახ.1.7. პლატფორმის სამუშაო გარემო

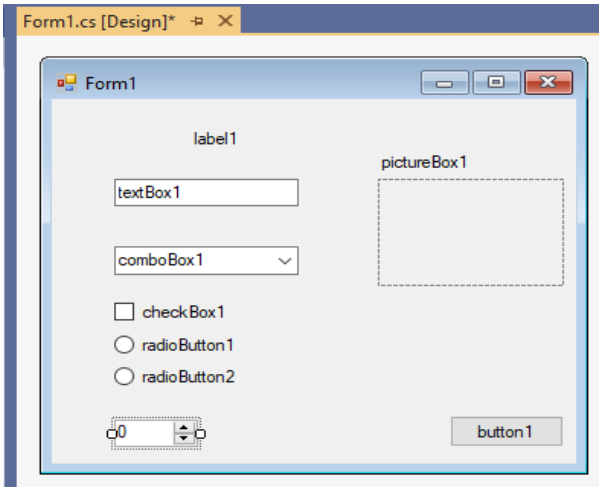
ინტერფეისის დიზაინის აწყობის დროს ინსტრუმენტების პანელიდან (ნახ.1.8) მაუსით გადაიტანება შესაბამისი პიქტოგრამა ფორმაზე და შემდეგ მისი თვისებები განისაზღვრება Properties- ფანჯარაში.



ნახ. 1.8. Toolbox პანელის შედგენილობა

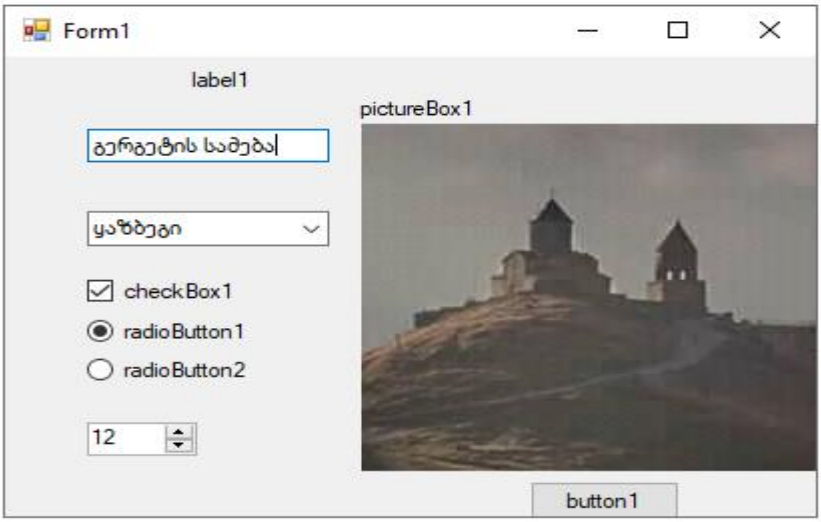
1.9 ნახაზზე ნაჩვენებია საილუსტრაციო მაგალითი, რომლის ფორმაზეც განთავსებულია ინსტრუმენტების პანელის რამდენიმე ელემენტი.

მათი და სხვა ელემენტების დანიშნულების შესწავლა ჩვენი დისციპლინის მიზანია.



ნახ.1.9. WindowsFormsApplication1 პროექტი

სამუშაო სეანსის შედეგის ფრაგმენტი ასახულია 1.10 ნახაზზე. დასრულებისას ხდება შედეგების შენახვა Location-ით (ნახ.1.6-გ) მითითებულ ადგილას.



ნახ.1.10. პროექტის შედეგი

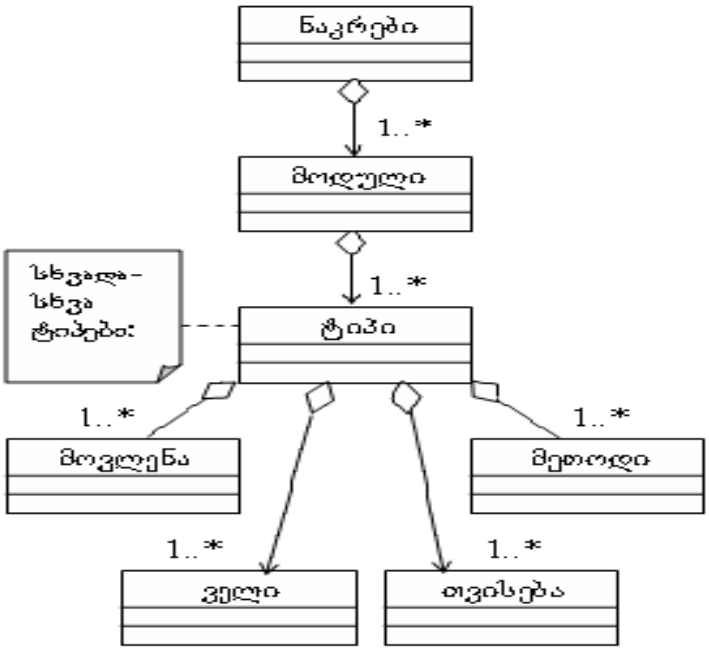
1.4. C#.NET აპლიკაციის კოდის კომპონენტები და სტრუქტურა

როგორც ზემოთ აღვნიშნეთ, .NET პლატფორმის საბაზო კლასების დიდი ნაწილი დაწერილია C# ენის გამოყენებით, ამიტომაც საილუსტრაციო მაგალითებს ამ ენაზე დავურთავთ კომენტარს.

ნაკრები (Assembly) .NET პლატფორმის კომპონენტებიდან ერთ-ერთი მთავარი ბლოკია (ანაწყობია), რომელიც ლოგიკურად აერთიანებს კოდს, რესურსებს და მეტამონაცემებს. იგი ლოგიკური და არა ფიზიკური ერთეულია, რადგან შეუძლია მოთავსდეს რამდენიმე ფაილში. ასეთ შემთხვევაში არსებობს ერთი მთავარი ფაილი, რომელშიც ინახება ინფორმაცია დანარჩენებზე.

1.11 ნახაზზე ნაჩვენებია პროგრამული დანართის (Application) შესაბამისი ნაკრების ზოგადი იერარქიული სტრუქტურა აგრეგაციის კავშირის გამოყენებით.

ნახაზიდან ჩანს, რომ ნაკრები ერთი ან რამდენიმე (1..*) მოდულისაგან (modeule) შედგება. სწორედ მოდულში ინახება დანართის ან ბიბლიოთეკის კოდი, მისი მეტამონაცემებით. მოდულები შეიცავს ტიპებს. ესაა კოდის შაბლონები (კლასები), რომლებშიც ინკაფსულირებულია გარკვეული მონაცემები და მეთოდები.



ნახ.1.11. ნაკრების (Assembly) ზოგადი სტრუქტურა

როგორც წინა პარაგრაფში ვახსენეთ, ტიპები ორი სახისაა: მიმთითებლებით (ანუ კლასები) და მნიშვნელობებით (ანუ სტრუქტურები).

ტიპებს აქვს *ველები, თვისებები* და *მეთოდები.*

ველი გამოყოფს მეხსიერების ადგილს შესაბამის მონაცემთა ტიპისათვის.

თვისებები ველების მსგავსია, ოღონდაც მათი დანიშნულებაა შესაბამის მონაცემთა საწყისი მნიშვნელობების განსაზღვრა და კონტროლი.

მეთოდები განსაზღვრავს მონაცემთა დასამუშავებლად კლასის ქცევას, ანუ რეაქციას გარედან შემოსულ შეტყობინებაზე (მოთხოვნაზე). შეტყობინება ინაფორმაციაა, რომელიც ამა თუ იმ მოვლენის შედეგად ფორმირდება.

პროგრამული ნაკრები შეიძლება იყოს ორი ტიპის: *კერძო* და *საერთო გამოყენების.* პირველ შემთხვევაში ნაკრები ინსტალირდება კერძო მომხმარებლის კატალოგში და მასთან სხვა მიმართვა გამორიცხულია.

საერთო გამოყენების ნაკრები შეიცავს პროგრამულ ბიბლიოთეკებს, რომელთაც იყენებს სხვადასხვა დანართი. აქ საჭიროა სპეციალური დაცვის მექანიზმების გამოყენება (სახელების კოლიზიისა და ნაკრებთა ვერსიების კონტროლის თვალსაზრისით).

კლასებს შორის სახელთა კოლიზიის აღმოფხვრის მიზნით .NET პლატფორმა იყენებს „სახელთა სივრცეს”.

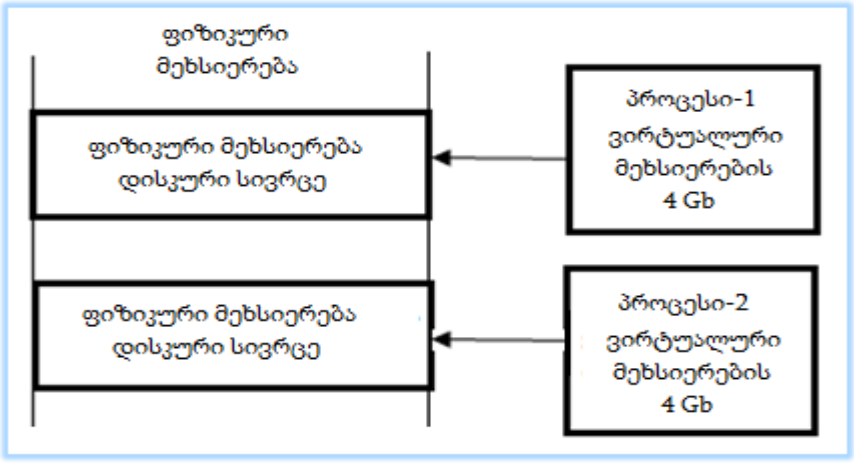
სახელსივრცე (namespace): ესაა მონაცემთა ტიპების უბრალო დაჯგუფება. ყველა მონაცემთა ტიპის სახელს მოცემულ სახელთა სივრცეში ავტომატურად ემატება პრეფიქსი, რომელიც შედგენილია სახელსივრცის დასახელებისაგან. ასევე შესაძლებელია ჩადგმული სახელსივრცეების შექმნა.

მაგალითად, საბაზო კლასების უმრავლესობისათვის, რომლებიც ზოგადი გამოყენებისთვისაა დანიშნული, მოთავსებულია სახელთა სივრცეში System, ვებგვერდებისათვის - System.Web და ა.შ. C#-ის პროგრამის ტექსტის მაგალითზე შეიძლება შემდეგი კომენტარის გაკეთება:

```
namespace Magazia.Web // აქ სახელია Magazia.Web
{
   public class Checkout : PageBase
   {
      // და ა.შ.
```

დანართთა არეები (application area) არის .NET პლატფორმის მნიშვნელოვანი ელემენტი. მათი დანიშულებაა ერთდროულად და ერთმანეთთან მომუშავე დანართების (აპლიკაციების) იზოლაცია, რათა არ მოხდეს მონაცემთა არასასურველი დამუშავება.

პროგრამული დანართების იზოლაციისათვის Windows გამოიყენებს **„პროცესის"** ცნებას, რომელიც მისამართების სივრცეს ეხება. ყოველ პროცესს გამოეყოფა 4 გიგაბაიტი ვირტუალური მეხსიერება. ისინი დისკოზე სხვადასხვა ფიზიკური მისამართებითაა და არ გადაიკვეთება (ნახ.1.12).



ნახ.1.12. აპლიკაციათა არეების იზოლაცია

პროცესებს აქვს მინიჭებული განსაზღვრული პრივილეგიები და ოპერაციული სისტემა აკონტროლებს მათ, თუ რომელ ოპერაციას რომელი პროცესის გამოყენება შეუძლია.

დანართთა არეების გამოყენების იდეა მდგომარეობს იმაში, რომ პროცესებს შორის მოხერხდეს მონაცემთა გაცვლა. ამიტომაც პროცესი იყოფა რამდენიმე დანართის არედ. თითოეულ დანართის არეში თავსდება ერთი დანართის კოდი.

.NET პლატფორმის მნიშვნელოვანი საშუალებაა JIT (Just-In-Time) კომპილატორი. იგი ახორციელებს პროგრამული კოდის ცალკეული ნაწილის დროულად კომპილირებას (საჭიროებისამებრ).

1.5. .NET პლატფორმის აპლიკაციათა ტიპები

მაიკროსოფთის .NET Framework პლატფორმაზე შესაძლებელია სხვადასხვა ტიპის დანართის (აპლიკაციის) დამუშავება. ასეთი პლატფორმის სიმძლავრე და მოქნილობა იმაშიც გამოიხატება, რომ განსხვავებული ტიპის აპლიკაციებისათვის გამოყენებადია სისტემის საბაზო კლასების ერთიანი ბიბლიოთეკა (სტრიქონებთან, გრაფიკასთან და მათემატიკურ ფუნქციებთან სამუშაოდ, მონაცემებთან წვდომისათვის, ფაილებთან სამუშაოდ, კრიპტოგრაფიული ოპერაციების შესასრულებლად, მონაცემთა სინქრონიზაციისათვის და ა.შ.).

განასხვავებენ .NET პლატფორმის შემდეგი ტიპის დანართებს:

- *სამაგიდო აპლიკაციები* (Desktop Applications) – მუშაობს მომხმარებლის ლოკალურ კომპიუტერზე;
- *ვებ-დანართები* (Web Appliocations) – მუშაობს ვებ-სერვერის ფარგლებში და წვდომადია მომხმარებლისთვის ბრაუზერის საშუალებით;
- *ვებ-დანართები მომხმარებლის მდიდარი ინტერფეისით* (Rich Internet Applications, RIA) – მიეწოდება მომხმარებელს HTTP/HTTPS პროტოკოლით ბრაუზერის ფარგლებში და სრულდება კლიენტის მხარეს;
- *ვებ-სერვისები* (Web Services) – პროგრამული კოდები, რომლებიც სრულდება სერვერის მხარეს და შეიძლება გამოძახებულ იქნას კლიენტისაგან რაიმე მონაცემების მისაღებად ან ოპერაციის შესასრულებლად;
- *მობილური დანართები* (Mobile Applications) – სრულდება მობილურ მოწყობილობებზე.

სამაგიდო აპლიკაცია ყველაზე გავრცელებული ტიპის დანართია. მას შეუძლია მიმართვა მომხმარებლის კომპიუტერის რესურსებთან (ვინჩესტერი,

მულტიმედიური მოწყობილობები და სხვ.). ეს აპლიკაციები იყოფა სამი სახის დანართად: ფანჯრული (Windows Forms Application, Windows Presentation Foundation), კონსოლური (Console Application) და სერვისული (Windows Sevice). პირველს აქვს გრაფიკული ინტერფეისი, დანარჩენ ორს – არა.

ვებდანართები განსხვავდება სამაგიდო აპლიკაციებისაგან იმით, რომ ისინი მუშობს ვებსერვერებისგან დაცილებით. მომხმარებელი იყენებს ბრაუზერს და HTTP/HTTPS პროტოკოლს. უპირატესობა არის ის, რომ აპლიკაცია ყენდება სერვერზე, რომლის გამოყენებაც შეუძლიათ ქსელში ჩართულ კლიენტ მომხმარე-ბლებს. ნაკლია ის, რომ მომხმარებლის ინტერფეისი შეზღუდულია (HTML, CSS და JavaScript-ის ფორმატების შეზღუდულობის გამო). მაიკროსოფთის .NET Framework პლატფორმაზე ვებდანართების აგება ხდება ASP.NET-ის (ASP.NET Web Application) და მის საფუძველზე განვითარებული ტექნოლოგიებით (Silverlight, MVC, MVVM და სხვ.).

მომხმარებლის ინტერფეისის სრულყოფის მიზნით შეიქმნა ახალი ტიპის დანართი RIA (Rich Internet Applications). იდეა ისაა, რომ ბრაუზერში ინტეგრირებულია სპეციალური პლაგინი, რომელსაც შეუძლია ასახოს შინაარსის (შიგთავსის) დამატებითი ტიპი. როდესაც მომხმარებელი ხსნის ბრაუზერის გვერდს, კლიენტის მხარეს გადაეცემა პროგრამული კოდი, რომელიც ამ პლაგინით მუშაობს. ამგვარად, კლიენტს მიეწოდება მდიდარი შესაძლებლობები თავისი ინტერფეისის ასაგებად. ასეთი ტექნოლოგებია, მაგალითად, Adobe Flach, Ms Silverlight და ა.შ.

ვებსერვისული აპლიკაცია არის პროგრამული კოდი, განთავსებული სერვერზე და გამოიძახება კლიენტის მოთხოვნის საშუალებით. მაგალითად, ასეთი სერვისების ერთობლიობა შეიძლება ემსახურებოდეს სერვერზე მოთავსებულ მონაცემთა ბაზასთან მუშაობას. სამაგიდო- და ვებ-აპლიკაციები ხშირად მიმართავს სერვისებს რაიმე ოპერაციის შესრულების მიზნით ან სერვერიდან მონაცემების მისაღებად.

სერვისების შესაქმნელად .NET Framework პლატფორმაზე არსებობს რიგი ტექნოლოგიებისა. მათ შორისაა ASP.NET Web Services, რომელიც ქმნის მარტივ ვებ-სერვისებს და მუშაობს HTTP/HTTPS პროტოკოლით. შედარებით ახალი და მოქნილი ტექნოლოგიაა Windows Communication Foundation (WCF), რომელიც იყენებს განსხვავებული ტიპის არხებს (HTTP, TCP, სახელდებული არხები და სხვ.) [6,11,13]. ეს საგრძნობლად აფართოვებს დეველოპერის შესაძლებლობებს სერვისების შესაქმნელად.

მობილური დანართები ფუნქციონირებს მობილურ მოწყობილობებზე Windows Mobile ოპერაციული სისტემის საფუძველზე. .NET Framework პლატფორმას აქვს აგრეთვე .NET Compact Framework შესაძლებლობათა ქვესიმრავლე, ოპერაციული სისტემებით: Windows, Android და iOS.

1.6. ობიექტ-ორიენტირებული დაპროგრამების მეთოდი: კლასები და ობიექტები

.NET გარემოში დაპროგრამება დაფუძნებულია ობიექტებზე. ობიექტი (object) – არის პროგრამული კონსტრუქცია, რომელშიც ინკაფსულირებულია ლოგიკურად დაკავშირებული მონაცემებისა და მეთოდების ერთობლიობა.

ობიექტი (Object) განიხილება, როგორც გარკვეული არსი (Entity), რომელიც ხასიათდება მდგომარეობით (მონაცემთა ერთობლიობა) და ქცევით (ფუნქციური პროგრამები). ობიექტის ქცევა ანუ რეაქცია, რომლის დროსაც მისი ახალი მდგომარეობა განისაზღვრება, დამოკიდებულია გარედან მოსულ ინფორმაციაზე, შეტყობინებებზე. ვინაიდან ობიექტი უმთავრესი ცნებაა, იგი ობიექტ-ორიენტირებული დაპროგრამების საწყისია, ამიტომ დიდი მნიშვნელობა აქვს მის სწორად გაგებას.

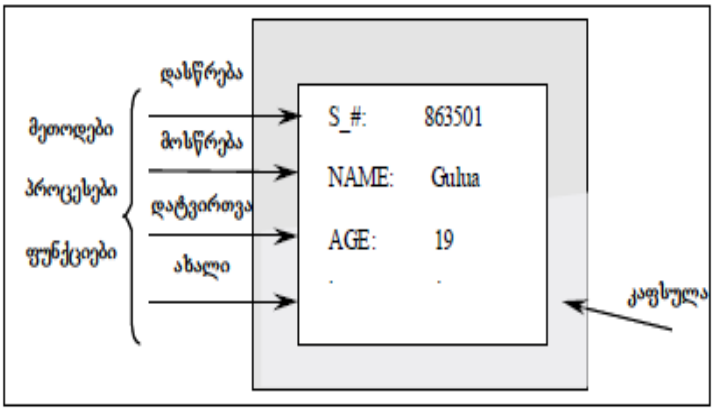
განვიხილოთ კლასისა და ობიექტის არსი ბიოლოგიური და ინფორმაციული უჯრედების მოდელების შედარების საფუძველზე (ნახ.1.13, 1.14) [3,13,18,].

ბიოლოგიური უჯრედი კაფსულირებულია მემბრანის (გარსის) საშუალებით, რომლითაც ის გამოიყოფა სხვა უჯრედებისა და გარემოსაგან. მას უნარი აქვს გარემოსგან მიიღოს ქიმიური სახის ინფორმაცია და უჯრედს შიგნით გადასცეს.

შუაგულში მოთავსებულია ბირთვი, რომელიც უჯრედის ძირითადი ინფორმაციის მატარებელია. იგი შედგება ქრომოსომებისგან, რომლებიც გარკვეული გენეტიკის მქონეა.

უჯრედის დაყოფის (გამრავლების) დროს ხდება მემკვიდრული თვისებების გადაცემა. ბირთვის გარშემო დაჯგუფებულია სხვადასხვა ფუნქციის ელემენტები. მაგალითად, ენდოპლაზმური ბადე – ცილების წარმოქმნის ფუნქცია, მიტოქონდრიები – ენერგიის გარდაქმნის ფუნქცია, ციტოპლაზმა (თხევადი მოძრავი გარემო) – ტრანსპორტირების ფუნქცია და ა.შ.

ნახ.1.13. ბიოლოგიური უჯრედი



ნახ.1.14. ინფორმაციული უჯრედი

როგორია „ინფორმაციული უჯრედის" აგებულება და რა ანალოგია აქვს ბიოლოგიურთან ? 1.14 ნახაზზე განიხილება ობიექტი-სტუდენტი, რომელიც რეალური სამყაროს ნაწილია. იგი კაფსულირებულია, რომლის შიგნითაც მოთავსებულია ბირთვი ობიექტის თვისებების აღმწერ მონაცემთა ელემენტები:

S_# - (studentis nomeri), NAME (gvari, saxeli), AGE (asaki) და ა.შ.

ობიექტები ავტონომიური ელემენტებია, ისინი იქმნება შაბლონით (template), რომელსაც ობიექტორიენტირებულ დაპროგრამების თეორიაში კლასს (class) უწოდებენ.

.NET-ის საბაზო კლასების ბიბლიოთეკა არის კლასთა ერთობლიობა, რომელიც გამოიყენება ობიექტების შესაქმნელად მომხმარებლის კერძო აპლიკაციაში. ამასთანავე პროგრამული პაკეტი Visual Studio საშუალებას იძლევა შეიქმნას საკუთარი კლასები კერძო პროგრამებისთვის.

- **ობიექტები, წევრები და აბსტრაჰირება:**

ობიექტი – პროგრამული კონსტრუქციაა, რომელიც ასახავს განსაზღვრულ არსს (Entity). ესაა რეალურ სამყაროში არსებული ობიექტები, მაგალითად, ადამიანები, მანქანები, ფირმები, ცხოველები, ფრინველები, მცენარეები, ინსტიტუტები, კომპიუტერები და ა.შ. ყოველ ობიექტს აქვს განსაზღვრული, მისთვის დამახასიათებელი ფუნქციუროობა და თვისებები.

აპლიკაციაში ობიექტი შეიძლება იყოს ფორმა, მართვის ელემენტი (ღილაკი, კომბობოქსი, ბაზასთან შეერთება და სხვ.).

ამგვარად, ობიექტი დასრულებული ფუნქციური ერთეულია, რომელიც შეიცავს ყველა მონაცემს და ყველა ფუნქციას, რომლებიც აუცილებელია იმ ამოცანის გადასაწყვეტად, რომლისთვისაც ეს ობიექტი გამოიყენება.

რეალური სამყაროს ობიექტების წარმოდგენას (ასახვას) პროგრამული ობიექტებით უწოდებენ აბსტრაჰირებას (abstraction).

- **კლასები, როგორც ობიექტთა შაბლონები:**

კლასი არის ერთგვაროვან ობიექტთა ერთობლიობა, რომელიც ამ ერთობლიობის სტრუქტურას ასახავს. კლასი განსაზღვრავს ობიექტთა წევრებს და მათ ყოფაქცევას, აგრეთვე საწყის მონაცემთა მნიშვნელობებს, საჭიროებისამებრ.

კლასის ეგზემპლარის შექმნისას კომპიუტერის მეხსიერებაში შეიქმნება ამ კლასის ასლი. ასეთი სახით შექმნილ კლასის ეგზემპლარს უწოდებენ ობიექტს. დაპროგრამების ენაში მისი შექმნა ხდება ოპერატორით new. მაგალითად:

```
   // gamocxaddes cvladi MyDataForm  tipiT DataForm
DataForm  MyDataForm;
   // Seiqmnas obieqtis egzemplari DataForm da
   // Caiweros igi cvladSi MyDataForm
MyDataForm = new DataForm( );
```

- **ობიექტები და წევრები:**

ობიექტები შედგება წევრებისაგან, რომელთაც ეკუთვნის თვისებები, ველები, მეთოდები და მოვლენები. ისინი განსაზღვრავს ობიექტის მონაცემებს და ფუნქციურობას.

ველები და თვისებები შეიცავს ობიექტის მონაცემებს, რომლებიც მის მდგომარეობას ასახავს. მეთოდები განსაზღვრავს მოქმედებებს, რომელთა

შესრულება შეუძლია ობიექტს. მოვლენები კი წარმოადგენს შეტყობინებებს, რომელთაც მიიღებს ობიექტი ან გადასცემს სხვა ობიექტებს. მოვლენის დანიშნულებაა, გააქტიუროს ობიექტის ესა თუ ის მეთოდი, რომელიც გადაამუშავებს მის მონაცემებს.

მაგალითად, ობიექტისთვის „ავტომობილი" თვისებები და ველებია: მოდელი, ფერი, ასაკი, საწვავის-ხარჯი და ა.შ. ეს მონაცემები ასახავს ობიექტის მდგომარეობას. მეთოდის მაგალითებია: სიჩქარის-გადართვა, დამუხრუჭება, საჭის-მობრუნება და სხვ., ისინი ობიექტის ქცევას განსაზღვრავს. ავტომო-ბილისთვის მოვლენები მიიღება შეტყობინების სახით გარედან (მაგალითად, ინფორმაცია გზის მოსახვევის შესახებ, ან სიჩქარის შეზღუდვის შესახებ) ან მანქანის მდგომარეობის ამსახველი ნათურებიდან (წყლის გადახურება, საწვავის დამთავრება და ა.შ.).

- **ობიექტური მოდელები:**

მარტივ ობიექტებს აქვს რამდენიმე თვისება და მეთოდი, ასევე ერთი-ორი მოვლენა. რთულ ობიექტებს გააჩნია მეტი რაოდენობა თვისებების, მეთოდებისა და მოვლენების, აგრეთვე მათ შეიძლება ჰქონდეს "შვილი" ობიექტებიც, რომლებზეც შეუძლია პირდაპირი მიმართვის განხორციელება. მაგალითად, მართვის ელემენტს TextBox, აქვს თვისება Font, რომელიც არის Font-ტიპის ობიექტი. ამ თვალსაზრისით, ნებისმიერი Form კლასის ეგზემპლარი მოიცავს მასზე განთავსებულ მართვის ელემენტებს (Controls ერთობლიობა).

„მშობელი-შვილი" ჩადგმული ობიექტების იერარქია, რომელიც ქმნის ობიექტის სტრუქტურას, არის ობიექტური მოდელი (object model).

მაგალითად, ობიექტი „ავტომობილი" შედგება რამდენიმე „შვილი" ობიექტისგან: ძრავი, ბორბლები, ძარა და ა.შ. „ავტომობილი" ობიექტის ყოფაქცევა, რომელსაც „შვილი" ობიექტისთვის (ძრავი) თვისებაში აქვს ცილინდრების რაოდენობა 4 და 8, იქნება განსხვავებული.

„შვილი" ობიექტი შეიძლება შედგებოდეს თავის საკუთარი „შვილი" ობიექტებისგან და ა.შ. მაგალითად: *ავტომობილი -> ძრავი -> სანთლები.*

- **ინკაფსულაცია:**

ობიექტ-ორიენტირებული დაპროგრამების (ოოდ) ერთ-ერთი საბაზო პრინციპია ინკაფსულაცია, რომლის არსი მდგომარეობს ობიექტის რეალიზაციის გამოყოფაში მისი ინტერფეისისაგან. ანუ პროგრამული დანართი (აპლიკაცია) ურთიერთქმედებს ობიექტთან მისი ინტერფეისის საშუალებით, რომელიც შედგება ღია თვისებებისა და მეთოდებისაგან.

ობიექტები ერთმანეთთან ურთიერთმოქმედებს თავიანთი ღია თვისებებით და მეთოდებით, ამიტომაც ობიექტს უნდა ჰქონდეს ყველა აუცილებელი მონაცემი და მეთოდების ერთობლიობა, ამ მონაცემთა დასამუშავებლად.

ინტერფეისმა არ უნდა მისცეს „სხვას“ ობიექტის შიგა მონაცემებთან წვდომის უფლება, რაც გამორიცხავს ამ შემთხვევაში ობიექტის შიგა მონაცემებისთვის public-მოდიფიკატორის გამოყენებას (გამოიყენება private).

- **პოლიმორფიზმი (polymorphism)** მრავალფორმიანობას ნიშნავს. მას ობიექტ-ორიენტირებულ დაპროგრამებაში განსაკუთრებული როლი აქვს. პოლი-მორფიზმის საშუალებით შესაძლებელია ერთიდაიმავე გახსნილი ინტერფეისის სხვადასხვაგვარი რეალიზაცია სხვადასხვა კლასში, ანუ პოლიმორფიზმის საშუალებით შესაძლებელია ობიექტის მეთოდებისა და თვისებების გამოძახება მათი რეალიზაციის მიუხედავად.

მაგალითად, ობიექტი *მძღოლი* ურთიერთქმედებს ობიექტთან *ავტომობილი* იმავე სახელის მქონე ღია ინტერფეისის საშუალებით. თუ სხვა ობიექტი, მაგალითად, *სატვირთო* ან *სპორტული-მანქანა* ფლობს ამ ღია ინტერფეისის მხარდაჭერას, მაშინ ობიექტს *მძღოლი* შეუძლია მათთანაც ურთიერთქმედება, მიუხედავად ინტერფეისის განსხვავებული რეალიზაციისა.

პოლიმორფიზმის რეალიზაციის ორი ძირითადი მიდგომა არსებობს: ინტერფეისებისა და მემკვიდრეობითობის გამოყენებით. განვიხილოთ თითო-ეული.

- **ინტერფეისი (interface)** – ესაა შეთანხმება, რომელიც ობიექტის ყოფაქცევას განსაზღვრავს. იგი ადგენს კლასის წევრთა სიას, მაგრამ არაფერს ამბობს მათი რეალიზაციის შესახებ. ობიექტში დასაშვებია რამდენიმე ინტერფეისის რეალიზაცია, ხოლო ერთიდაიგივე ინტერფეისი შეიძლება რეალიზებულ იქნას სხვადასხვა კლასში.

ნებისმიერ ობიექტებს, რომლებშიც რეალიზებულია რომელიმე ინტერფეისი, შეუძლია ერთმანეთთან ურთიერთმოქმედება მისი საშუალებით. მაგალითად, ობიექტში *ავტომობილი*, რომელზეც ჩვენ ვსაუბრობთ, შეიძლება *IDrivable*-ინტერფეისის რეალიზაცია (ინტერფეისთა სახელები მიღებულია დაიწყოს “I” ასოთი), მეთოდებით: *მოძრაობა-წინ, მოძრაობა-უკან, გაჩერება*. იგივე ინტერფეისი შეიძლება რეალიზდეს სხვა კლასებშიც, როგორიცაა, მაგალითად, *სატვირთო-მანქანა* ან *კატერი*. შედეგად, ამ ობიექტებს შეუძლია ურთიერთმოქმედება ობიექტთან *მძღოლი*. ეს უკანასკნელი მთლიანად

უხილავია ინტერფეისის რეალიზაციისათვის, რომელთანაც იგი მოქმედებს, მისთვის ცნობილია მხოლოდ ინტერფეისი.

- **მემკვიდრეობითობა (inheritance)** იძლევა ახალი კლასების შექმნის საშუალებას არსებულის ბაზაზე. ამასთანავე ახალ კლასებში ნებადართულია ძველი კლასების სრული ფუნქციონალობის ჩართვა და, საჭიროებისამებრ, შესაძლებელია მათი წევრების მოდიფიცირებაც.

კლასი, რომელიც გამოცხადებულია სხვა კლასის საფუძველზე, უწოდებენ წარმოებულ კლასს (derived class). ყოველ კლასს შეიძლება ჰქონდეს მხოლოდ ერთი პირდაპირი წინაპარი – მისი საბაზო კლასი (base class). წარმოებულ კლასს აქვს საბაზო კლასის წევრების ერთობლიობა. ასევე შესაძლებელია ახალი წევრების დამატება და ძველი წევრების (მემკვიდრეობით მიღებული) რეალიზაციის ცვლილებაც. წარმოებული კლასები ინარჩუნებს თავისი საბაზო კლასის ყველა მახასიათებელს და უნარი აქვს სხვა ობიექტებთან ისეთი ურთიერთმოქმედებისათვის, როგორც საბაზო კლასის ეგზემპლარებს.

მაგალითად, საბაზო კლასიდან *ავტომობილი* შეიძლება წსრმოებული კლასის *სპორტული-ავტომობილი* გამოცხადება, რომელიც თავის მხრივ, როგორც საბაზო კლასი, შეძლებს ახალი წარმოებული კლასის *სპორტული-კაბრიოლეტი* გამოცხადებას. ყოველ წარმოებულ კლასში შესაძლებელია ახალი წევრების (თვისებები, მეთოდები, მოვლენები) შემოტანა, ამსთანავე ისინი ინარჩუნებს საწყისი საბაზო კლასის *ავტომობილი* ფუნქციურობას უცვლელი ფორმით.

1.7. Visual Studio .NET პლატფორმის განვითარების ისტორია

Visual Studio .NET-ში გამოიყენება პროგრამირების მრავალი ენა: C, C ++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML და CSS. სხვა ენებზე, როგორიცაა Python, Ruby, Node.js და M, მხარდაჭერა ხელმისაწვდომია plugin-ების საშუალებით (დამოუკიდებლად კომპილირებადი პროგრამული მოდულები, რომლებიც უერთდება ძირითად პაკეტს და აფართოებს მის შესაძლებლობებს). Java და J# ადრეულ ვერსიებში (2008 წლამდე) იყო. აქვს Ms SQL Server და SQL Server Express მონაცემთა ბაზებთან წვდომა [19].

Visual Studio .NET Framework 1.0, როგორც ცნობილია, 2002 წ. გამოვიდა. ამ პლატფორმის ძირითადი საფუძველი იყო სხვადასხვა პროგრამების შესრულების საერთო გარემო (CLR-Common Language Runtime), როგორც 1.1 პარაგრაფში იყო აღნიშნული, ენების ასეთი თავსებადობა ხორციელდება CTS -

საერთო ტიპების სისტემის საფუძველზე IL შუალედური გარდაქმნის ენით. აქვე ვიხილეთ C# ენის პირველი გამოცემა C# 1.0.

Microsoft Visual Studio .NET 2003 Professional Special Edition ვერსია NET Framework 1.1-ით გამოვიდა დამატებული პროგრამული კომპონენტებით, კერძოდ, Windows Server 2003 Standard Edition და SQL Server 2000 Developer Edition-ით. Visual Studio .NET 2003 წარმოდგენილი იყო რამდენიმე რედაქციით: Academic, Standard, Professional, Enterprise Developer და Enterprise Architect. ეს უკანასკნელი, მაგალითად, უკვე მოიცავდა Ms VISIO 2002-ის მოდელირების ტექნოლოგიის რეალიზაციას, უნიფიცირებული მოდელირების ენის (UML) საფუძველზე აპლიკაციების არქიტექტურის ვიზუალური წარმოდგენების ასაგებად. აქვე გამოჩნდა პირველი ინსტრუმენტული საშუალებები ობიექტ-როლური მოდელირებისათვის (ORM - Object Role Modeling) და მონაცემთა ბაზების მოდელირების ლოგიკური გადაწყვეტა (Logical database-modeling solution) [9,20,21,35,36] .

Visual Studio .NET 2005 .NET Framework 2.0-ისთვის გამოვიდა პირველად Express: ვერსიები: Visual C++ 2005 Express, Visual Basic 2005 Express, Visual C# 2005 Express და სხვ. Visual Studio 2005 განახლდა .NET Framework 2.0-ში დანერგილი სიახლეების მხარდასაჭერად. მათ შორის ASP.NET 2.0-ის და IntelliSense-ს ფუნქციათა განზოგადებისთვის, დაემატა ახალი პროექტის ტიპები ASP.NET-ის ვებ-სერვისებისათვის, აგრეთვე ამოცანების გადაწყვეტის მხარდაჭერა ახალ პლატფორმაზე, ე.წ. Microsoft Build Engine (MSBuild) სახით, რომელიც იყენებს XML-ზე დაფუძნებულ პროექტის ფაილის ფორმატს. Visual Studio 2005-ს აქვს ლიკალური ვებ-სერვერი (გამოყოფილი IIS-დან), რომელსაც შეუძლია ASP.NET აპლიკაციების განთავსება მათი დამუშავებისა და ტესტირების დროს. აქვს აგრეთვე SQL Server 2005 მონაცემთა ბაზის სრული მხარდაჭერა და C# 2.0 ვერსია.

Visual Studio 2008 გამოვიდა .NET Framework 3.5-ით, რომელსაც აქვს WPF (Windows Presentation Foundation), WCF (Windows Communication Foundation), WF (Workflow Foundation) ტექნოლოგიებისა და LINQ (Language Integrated Query) კომპონენტის მხარდაჭერა. Microsoft Expression Web-ის გავლენით სისტემაში ჩაიდო HTML/CSS-ის ახალი რედაქტორი. აგრევე კოდის ანალიზის ახალი ინსტრუმენტი Code Metrics, რომლის დანიშნულება პროგრამული უზრუნველ-ყოფის ზოგიერთი თვისების ან მისი სპეციფიკაციის რაოდენობრივი შეფასების მნიშვნელობის მიღებაა. წარმოდგენილ იქნა C# 3.5 და Visual Basic-ის ენების ახალი ვერსიები.

Visual Studio 2010 (.NET Framework 4.0-ით) გადამუშავებული IDE-ინტეგრირებული გარემოთი ასუფთავებს და ამარტივებს მომხმარებლის ინტერფეისის ორგანიზაციას. გადამუშავება განხორციელდა WPF ტექნოლოგიის გამოყენებით, რის შედეგადაც უკეთესად სრულდება მულტიმონიტორინგი და

მხარდაჭერის სერვისი. ერთ-ერთი ძირითადი სიახლე იყო C# 4.0 და ფუნქციონალური F# ენის შემოტანა, აგრეთვე პარალელური პროგრამირების ინსტრუმენტების გამარტივება და პარალელური აპლიკაციების გამართვის ინსტრუმენტების რეალიზაცია.

Visual Studio 2012 ვრცელდებოდა Visual Studio 2010-ის იმავე რედაქციით, განსხვავება იყო Visual Studio 2012 Express ვერსიაში, სადაც ყველა ენა იყო აქტიური (vs2010-ში იყო მხოლოდ C# Express და VB Express), აგრეთვე არსებობდა Visual Studio Express 2012-ის 5 ვერსია: Web-ის, Windows8-ის, Windows Desktop-ის, Windows Phone-ს და Team Foundation Server-ის. ამასთანავე, Windows8-ზე მუშავდებოდა აპლიკაციები Windows Store-სთვის ახალი ფორმის ინტერფეისით, ხოლო Windows Desktop-ით კი - „კლასიკური“ ფორმატით.

Visual Studio 2013 იყო შემდეგი თაობის პლატფორმა Windows 8.x -სთვის, .NET Framework 4.5.1-ით. იგი უზრუნველყოფდა პროტოტიპის შექმნის, დაპროექტებისა და მოდელირების გაფართოებულ მხარდაჭერას, აგრეთვე გაუმჯობესებული ტესტირების ინსტრუმენტულ საშუალებებს. დეველოპერებს საშუალება ეძლეოდათ შეექმნათ Windows-, ვებ- და ღრუბლოვანი აპლიკაციები.

Visual Studio 2015 მაიკროსოფთის მიერ გამოცხადებული იყო როგორც ამ პროდუქტის ბოლო ვარიანტი. იგი სამი ვერსიის ვრცელდებოდა: უფასო Community Edition, რომელიც მოიცავდა ყველა ექსპრეს ვერსიას, ფასიანი Professional Edition მცირე პროექტებისთვის და Enterprise Edition დიდი პროექტებისათვის. მას გააჩნდა მრავალი ახალი ფუნქცია და ინსტრუმენტი დეველოპერებისთვის, მათ შორის ღია საწყისი კოდით, C# 4.6-ით და .NET 4.6 Preview (საპროექტო ვერსიით).

Visual Studio 2017 გამოვიდა .NET Framework 4.7-ით და C# 4.7-ით. შემოთავაზებულ იქნა ახალი ფუნქციები, NGen (Native Image Generator) უტილიტის მხარდაჭერა, რომელიც იძლევა შესაძლებლობას შეიქმნას CIL (Common Intermediate Language)-კოდები CLR (Common Language Runtime) შესრულებადი გარემოსთვის მანქანურ კოდებში (native image) და მოათავსოს იგი ლოკალური კომპიუტერის ცენტრალური მეხსიერების CLI assembly-ში (Common Language Infrastructure). ესაა კომპილირებული კოდის ბიბლიოთეკა, რომელიც გამოიყენება განთავსების, ვერსიების მართვის და უსაფრთხოების უზრუნველ-საყოფად. აქვე საყურადღებოა .NET Core-ს ინსტრუმენტების ნაკრების არსებობა პროგრამული უზრუნველყოფის დასამუშავებლად, ღია საწყისი კოდით. იგი გამოიყენება Windows, Linux და macOS ოპერაციული სისტემებისთვის. იყენებს ASP.NET MVC და Docker-ის კონტეინერებს (Docker toolset (Preview)). იგი PaaS (platform-as-a-service) პროდუქტების ერთობლიობაა, რომელშიც რეალიზებულია ვირტუალიზაცია ოპერაციული სისტემის დონეზე. გამოჩნდა Xamarin 4.3

(Preview), რომელიც მხარს უჭერს აპლიკაციების აგებას C#-ზე სისტემებისათვის Android, iOS, Windows, Mac [22].

Visual Studio 2019 .NET Framework 4.8. მისი პირველი ვერსია (v.16.0.0) გამოვიდა 2019 აპრილში, ხოლო ბოლო - (v.16.3.4). – 10 ოქტომბერს (მონაცემები დაფიქსირებულია 15.10.2019) [23].

სიახლეები, რომლებიც ამ ვერსიებში იქნა შემოტანილი, განიხილება კოდის დამუშავების (Develop), ერთობლივი მუშაობის (Collaborate) და კოდის გამართვის (Debug) პროცესების ჭრილში [24].

- **Develop:** საჭიროა კონცენტრირება მთავარზე (ყველაზე მნიშვნელოვანზე) და პროდუქტიულობის გაზრდა კოდის ოპტიმიზებული შესრულებით, მისი მყისიერი გაწმენდით და ძიების უფრო ზუსტი შედეგებით;
- **Collaborate:** შესაძლებელია ერთობლივი მუშაობა, კოდის რეალურ დროში რედაქტირებისა და გამართვის ფუნქციების გამოყენება, კოდის მიმოხილვა უშუალოდ Visual Studio-ში;
- **Debug:** შესაძლებელია განსაზღვრული მნიშვნელობების გამოყოფა და მათზე გადასვლა; მეხსიერების გამოყენების ოპტიმიზაცია და აპლიკაციის მუშაობის მომენტალური სურათების ავტომატურად შექმნა.

დასკვნის სახით შეიძლება ითქვას, რომ დღეისათვის არსებობს Visual Studio .NET-ის რამდენიმე პროგრამული პროდუქტი, როგორებიცაა: .NET Standard, .NET Framework, .NET Entity Framework, .NET Core და .NET Code [25]. მოკლედ განვიხილოთ თითოეული.

1.7.1. Visual Studio .NET Standard

.NET პლატფორმების ყოველ რეალიზაციას აქვს საკუთარი საბაზო კლასების ბიბლიოთეკა (BCL - Base Class Library). BCL შეიცავს ისთ კლასებს, როგორიცაა გამონაკლისების დამუშავება (exception handling), სტრიქონები (strings), XML, I/O, ქსელში მუშაობა (networking), კოლექციები (collections) და სხვ.[ლიტ].

.NET სტანდარტი არის BCL-ის განხორციელების სპეციფიკაცია, რომელიც არის .NET-ის ყველა რეალიზაციაში. ამგვარად, ვინაიდან .NET რეალიზაცია ხდება ამ სტანდარტების დაცვით, აპლიკაციების შემქმნელებს არ აქვთ პრობლემები აღნიშნული BCL ვერისების შესახებ.

კლასთა ბიბლიოთეკების ფრეიმვორკი (FCL - Framework Class Libraries), რორროგებიცაა, მაგალითად, WPF, WCF და ASP.NET, არაა BCL-ის ნაწილი და, ამიტომაც ისინი არ შედის .NET Standard-ის შემადგენლობაში. შედარების

თვალსაზრისით შეიძლება ითქვას, რომ .NET Standard-სა და .NET რეალიზაციას შორის ისეთივე დამოკიდებულებაა, როგორიც HTML სპეციფიკაციასა და ბრაუზერს შორის - მეორე არის პირველის განხორციელება (შედეგი),

1.7.2. Visual Studio .NET Framework

.NET Framework – ესაა NET-ის სრული და ტრადიციული ვარიანტი, რომელიც ვრცელდება Windows პლატფორმისთვის (ჩვენ წინა პარაგრაფებში იგი დეტალურად განვიხილეთ). მისი გამოყენება ხდება ვინდოუსის სამაგიდო (Desktop) აპლიკაციების ან ASP.NET-ის (Web) აპლიკაციების შექმნის დროს.

1.7.3. Visual Studio .NET Entity Framework

Entity Framework (EF) – ესაა ობიექტ-რელაციური ასახვის ფრეიმვორკი (ORM - Object Relational Mapping) ღია საწყისი კოდით ADO.NET-ისთვის (Active Data Object). იგი იყო .NET Framework-ის ნაწილი, მაგრამ .NET 4.6 ვერსიიდან გამოეყო მას.

Entity Framework უზრუნველყოფს ობიექტებთან კავშირს როგორც LINQ-ის, ასევე Entity SQL-ის საშუალებით. Web-აპლიკაციების აგების პროცესის გასამარტივებლად იგი იყენებს როგორც ADO.NET Data Services-ს, ასევე კავშირს WCF (Windows Communication Foundation) და WPF (Windows Presentation Foundation)-თან. საშუალებას იძლევა მრავალდონიანი აპლიკაციების ასაგებად MVC (Model-View-Controller), MVP (Model-View-Presenter) ან MVVM (Model–View–VviewModel).

- Model-View-Controller – ესაა აპლიკაციის მონაცემების, მომხმარებლის ინტერფეისისა და მმართველი ლოგიკის დაყოფის სქემა სამ კომპონენტად: მოდელი, წარმოდგენა და კონტროლერი. მოდელი არის მონაცემები და რეაგირებს კონტროლერის ბრძანებებზე, იცვლის თავის მდგომარეობას. წარმოდგენა უზრუნველყოფს მოდელის მონაცემთა ასახვას მომხმარებლისთვის, რომელსაც შეუძლია მოდელის შეცვლა. კონტროლერი ასრულებს (ინტერპრეტაციას უკეთებს) მომხმარებლის ქმედებებს, აცნობებს მოდელს ცვლილებების შესახებ;
- Model-View-Presenter – ესაა MVC-დან წარმოებული არქიტექტურული შაბლონი და ძირითადად გამოიყენება მომხმარებლის ინტერფეისის შესაქმნელად. იგი უზრუნველყოფს ავტომატური მოდულური ტესტირების პროცესის გამარტივებას;

- Model–View–ViewModel – ესაა პროგრამული არქიტექტურული შაბლონი, რომელიც უზრუნველყოფს მომხმარებლის გრაფიკული ინტერფეისის (markup language) დეველოპმენტის გამოყოფას ბიზნეს-ლოგიკის (back-end logic ანუ data model) დეველოპმენტისგან .

1.7.4. Visual Studio .NET Core

.NET Core არის მართვადი ფრეიმვორკის უფასო, კროსპლატფორმული რეალიზაცია ღია საწყისი კოდით. იგი მხარს უჭერს ოთხი ტიპის აპლიკაციას: console, ASP.NET Core, cloud, და Universal Windows Platform (UWP). რაც შეეხება Windows Forms და WPF-ს (Windows Presentation Foundation), ისინი .NET Core 1.0 და 2.0 ვერსიებში არ შედიოდა, ხოლო .NET Core 3.0 ვერსიიდან უკვე შედის [26].

ტექნიკურად, .NET Core მხარს უჭერს მხოლოდ კონსოლის პროგრამებს. ASP.NET Core და UWP არის აპლიკაციების მოდელები, აგებული .NET Core-ს საფუძველზე.

განსხვავებით .NET Framework-იდან, .NET Core არ ითვლება Windows-ის კომპონენტად. ამიტომ, განახლებები ხდება NuGet პაკეტების სახით, არა Windows განახლების საშუალებით.

. NET Core-ს აქვს C# და F# -ის სრული და Visual Basic.NET-ის ნაწილობრივი მხარდაჭერა.

.NET Core ბრძანების სტრიქონი უზრუნველყოფს ოპერაციული სისტემების ინტერფეისს და დეველოპერის მომსახურებას, როგორიცაა კომპილაცია და პაკეტების მართვა.

1.7.5. Visual Studio .NET Code

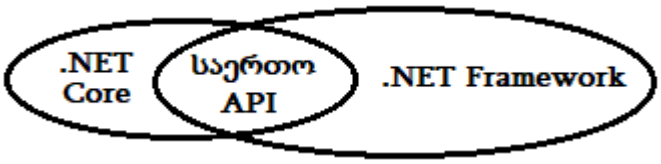
Visual Studio Code – არის საწყისი კოდის რედაქტორი, დამუშავებული მაიკროსოფთის მიერ როგორც კროსპლატფორმული პროდუქტი Windows, Linux და macOS ოპერაციული სისტემებისთვის ვებ- და ღრუბლოვანი აპლიკაციების ასაგებად [25,27]. იგი დაწერილია TypeScript და JavaScript ენებზე. პირველი ვერსია გამოვიდა 2015-ში MIT-ის ლიცენზიით (მასაჩუსეტსის ტექნოლოგიის ინსტიტუტი), ხოლო ბოლო ვერსია - 10.2019 (სისტემა შედარებით ახალი და ჯერ ბოლომდე არასრულყოფილია, მაგრამ მაიკროსოფთი ყოველთვიურად ანახლებს მას). VS_Code-ს აქვს კარგი შესაძლებლობები სხვა სისტემებთან და ენებთან მიმართებასი. მაგალითად, VS Code-ს გამოყენებით შეიძლება ASP.NET Core MVC აპლიკაციის აგება [28].

განსაკუთრებით საინტერესო კონცეფცია და განვითარება მიიღო VS Code და Python ენის ინტეგრირებულმა გამოყენებამ [29].

1.7.6. Visual Studio .NET ვერსიების შედარება

Visual Studio .NET პლატფორმების ვერსიათა მოკლე ანალიზის შემდეგ შეიძლება განვიხლოთ მათი შედარების საკითხი, რათა შესაძლებელი იყოს მომხმარებლისთვის გადაწყვეტილების მიღება - თუ რომელ სიტუაციასი რომელი ვერსია გამოიყენოს. ინტერნეტშიც მოიპოვება საკმაო მასალა ამ თემატიკაზე [30,31].

1.15 ნახაზზე ნაჩვენებია მაგალითად, .NET Framework და .NET Core ვერსიების ურთიერთმიმართება. გადაკვეთაში საერთო API-ია მოთავსებული. API - აპლიკაციების პროგრამირების ინტერფეისი ანუ კლიენტ-სერვერული კომუნიკაციის პროტოკოლი, რომლის დანიშნულებაა კლიენტის მხარეს პროგრამული უზრუნველყოფის შექმნის პროცესის მხარდაჭერა [32].



ნახ.1.15

ის, რაც ორივეს ეკუთვნის, ანუ „საერთო-API“ არის სწორედ .NET Standart. რეკომენდებულია შემდეგი მოსაზრებების გათვალისწინება, როდესაც აღნიშნული პლატფორმების შერჩევას ვაპირებთ [17,32].

- *Microsoft. NET Framework უკეთესი ვარიანტია, თუ:*
  - მომხმარებელს არ სურს ახალი ტექნოლოგიის შესწავლა;
  - მომხმარებელი უკვე არის არსებული API-ს ექსპერტი და სურს იმავე აპლიკაციაზე მეტი ფუნქციების შესწავლა;
  - უკვე არსებობს წარმატებული გუნდი. NET Framework-თან მუშაობის გამოცდილებით;
  - მუშაობისათვის საჭიროა სტაბილური გარემო;
  - მომხმარებელი ფლობს .NET Framework-ის ახალი რელიზების გამოცემების გრაფიკს;
  - მომხმარებელს არ სურს მუდმივად განახლებების და ცვლილებების რეჟიმში მუშაობა;

- მომხმარებელს სურს დესკტოპ-აპლიკაციების შექმნა Windows Forms ან WPF-ის გამოყენებით.

➢ *Microsoft .NET Core უკეთესი ვარიანტია, თუ:*

- მომხმარებელს მოსწონს ახალი საკითხების სწავლა და განახლება.
- საჭიროა აპლიკაციების შემუშავება Windows, Linux და Mac ოპერაციული სისტემებზე .NET Core გამოყენებით.
- მომხმარებელი არ უფრთხის არსებული შედეგების გადაკეთებას, რადგან ASP.NET Core ჯერ კიდევ არ არის ბოლომდე სრულყოფილი;
- მომხმარებელი დეველოპერი მხოლოდ სწავლობს. NET ვერსიებს;
- მომხმარებელს მოწონს open source კოდებთან მუშაობა.

II თავი

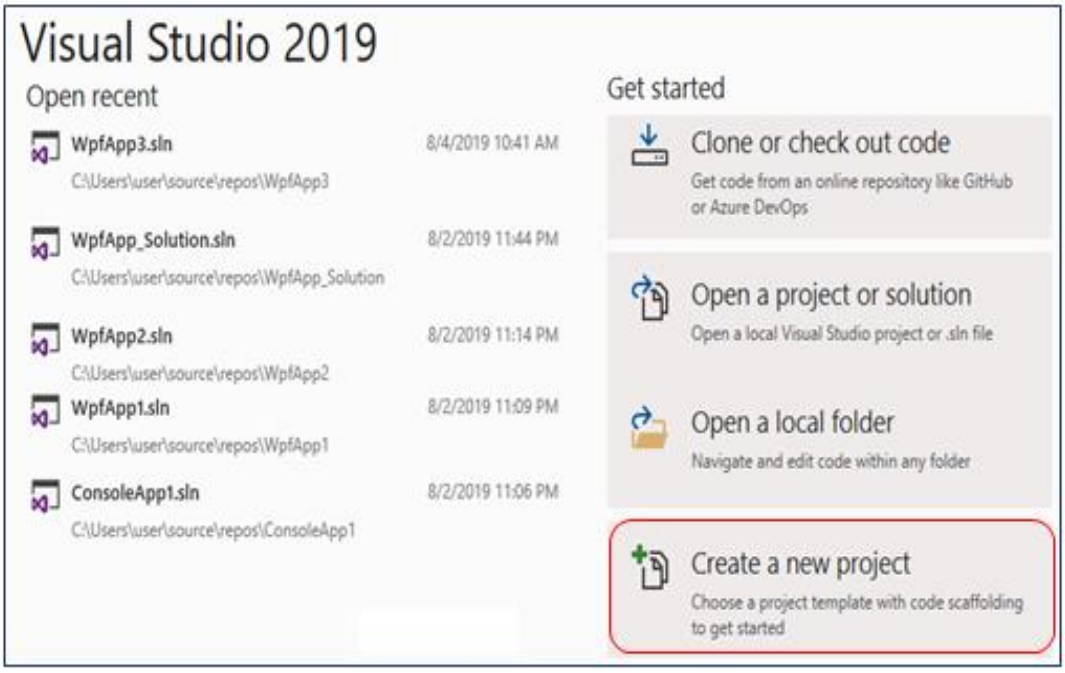
ვიზუალური C# ენის ლაბორატორიული პრაქტიკუმის ამოცანები Visual Studio.NET 2019 პლატფორმაზე

2.1. C#-კოდის აგება და გამართვა კონსოლის რეჟიმში

სამუშაოს მიზანი: პირველი C# კოდის აგება Visual Studio.NET Framework 4.7.2 ვერსიის (2019) კონსოლის რეჟიმში.

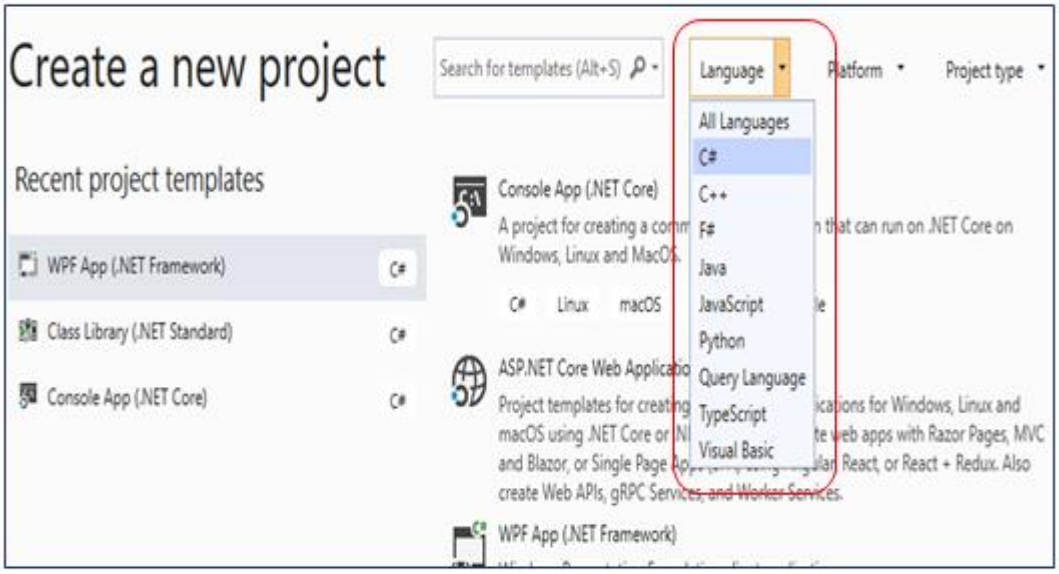
წინასწარ D:\ დისკზე შექმენით თქვენი ფოლდერი მომავალი პროგრამული პროექტების შესანახად. ავამუშავოთ Visual Studio.Net და კონსოლის რეჟიმში (Console Application) C# -ით შექმენით ახალი პროექტი მაგალითად, Lab_N1 (სახელი შეიძლება შეარჩიოთ თვითონ).

1. გააქტიურეთ პროგრამა Visual Studio 2019, მიღებული ფანჯრიდან აირჩიეთ, Create a new project (ნახ.2.1).



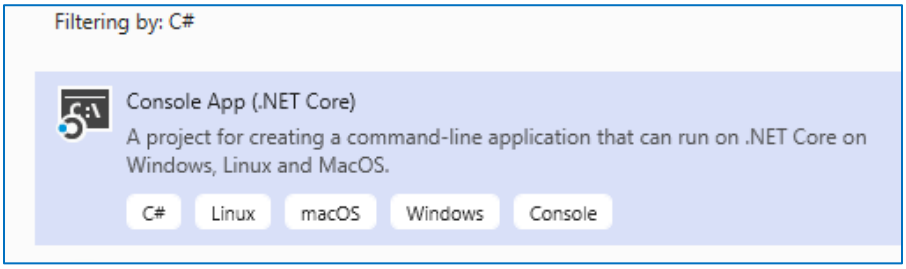
ნახ.2.1. ახალი პროგრამული პროექტის შექმნა

2. ახალი პროექტის შექმნის ფანჯარაში ველიდან Language, ნიმუშის შესაბამისად აირჩიეთ C# ენა (ნახ.2.2).



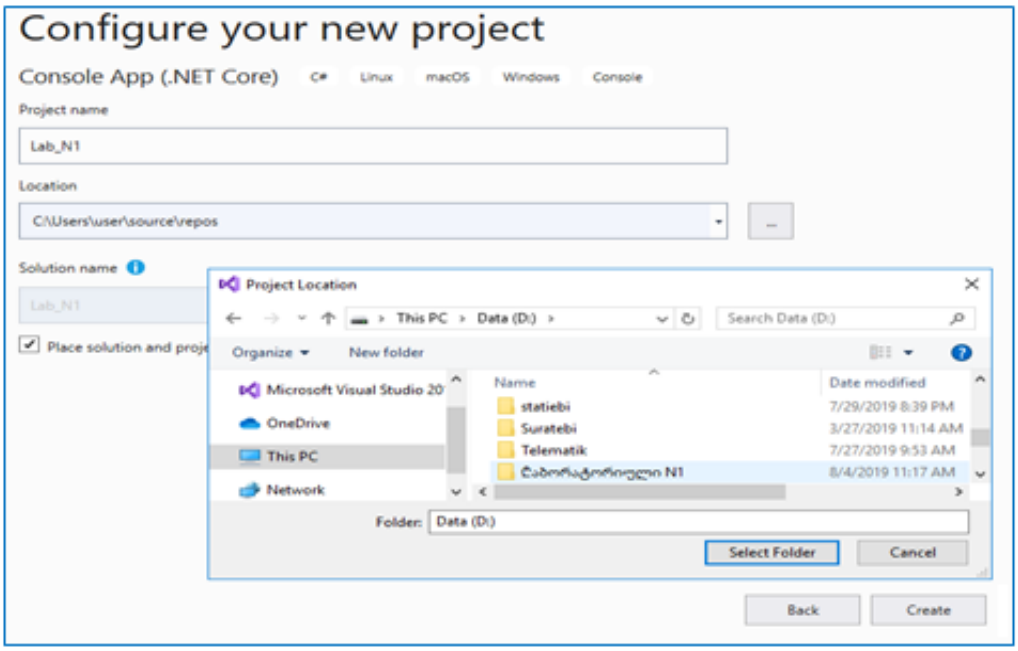
ნახ.2.2. პროგრამული ენის არჩევა პროექტისთვის

3. შედეგად მიიღებთ C# Console Application, ასარჩევ გარემოს (ნახ.2.3).



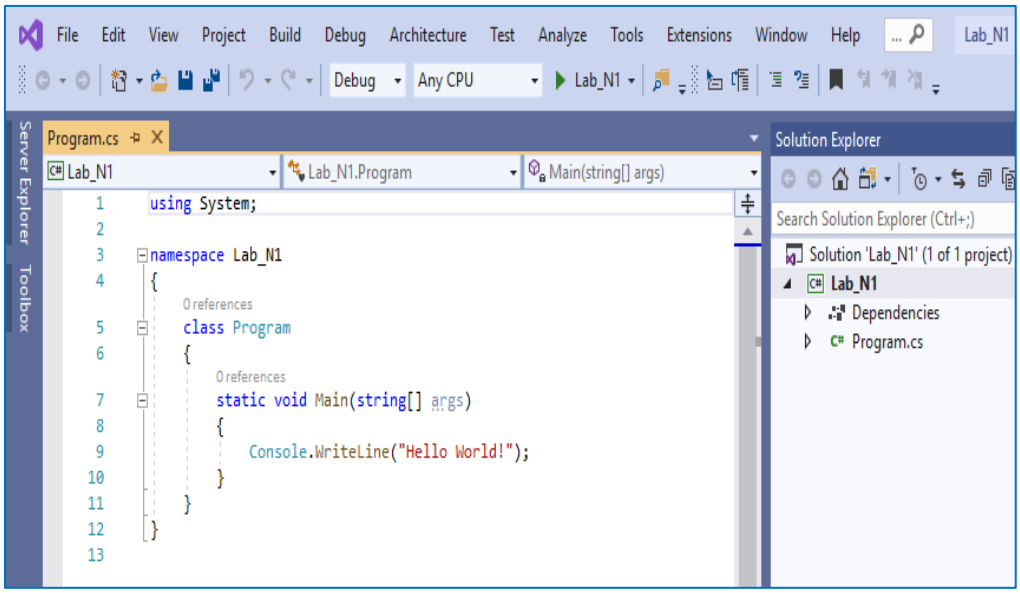
ნახ.2.3. Console App სამუშაო გარემო

4. Console App არის მონიშვნის და ქვედა მარჯვენა მხარეს მოთავსებული Next ღილაკის გააქტიურებით გადახვალთ ფანჯარაში, სადაც ველში Project Name, ჩაწერთ პროექტის სახელს, მაგალითად, Lab_N1, ხოლო Location, აირჩევთ თქვენს მიერ D: დისკზე წინასწარ შექმნილ ფოლდერის სახელს (ნახ.2.4).



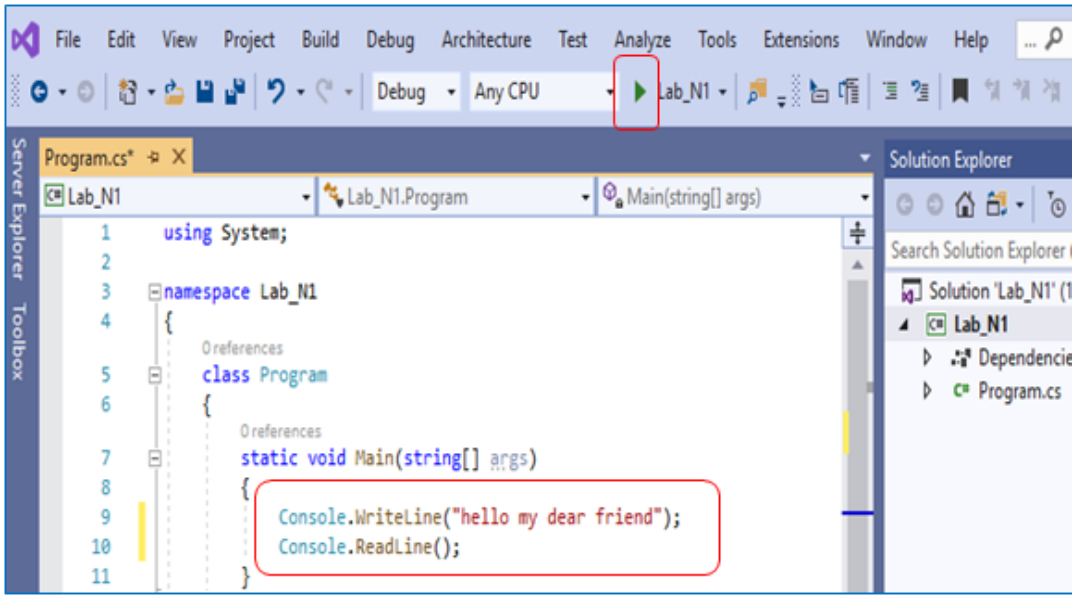
ნახ.2.4. პროექტის სახელის და ლოკაციის ადგილის შერჩევა

5. ქვედა მარჯვენა კუთხეში მოცემული Create ღილაკის დახმარებით, მიიღებთ სამუშაო გარემოს, სადაც ხდება კოდის შეტანა (ნახ.2.5).



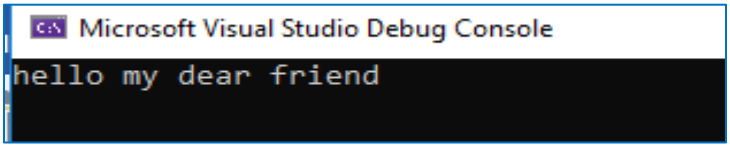
ნახ.2.5. სამუშაო გარემო კოდის შესატანად

ამოცანა 1.1: პირველი პროგრამის მაგალითი. შეტანეთ კოდის static void Main(…) – ში ორი სტრიქონი (ნახ.2.6).



ნახ.2.6. WriteLine() და ReadLine() მეთოდების ჩაწერა

აამუშავეთ პროგრამა ღილაკით  . შედეგად მიიღებთ (ნახ.2.7).

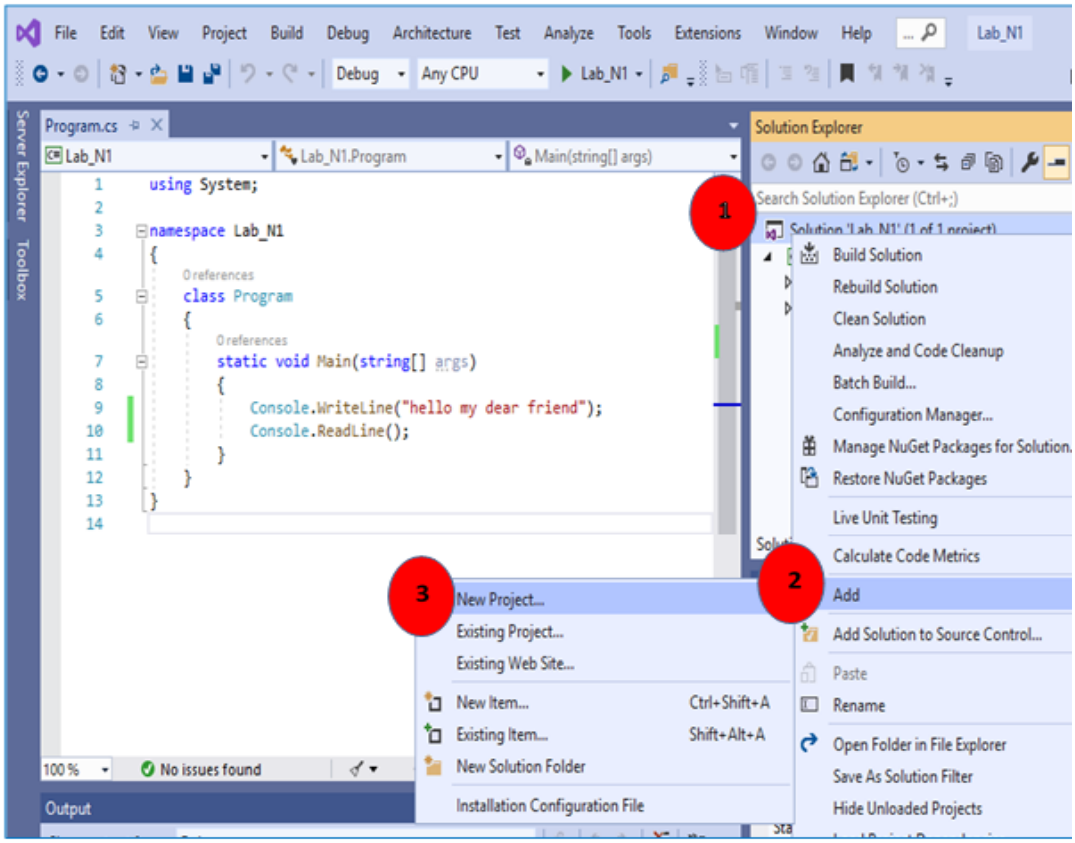


ნახ.2.7. შედეგი

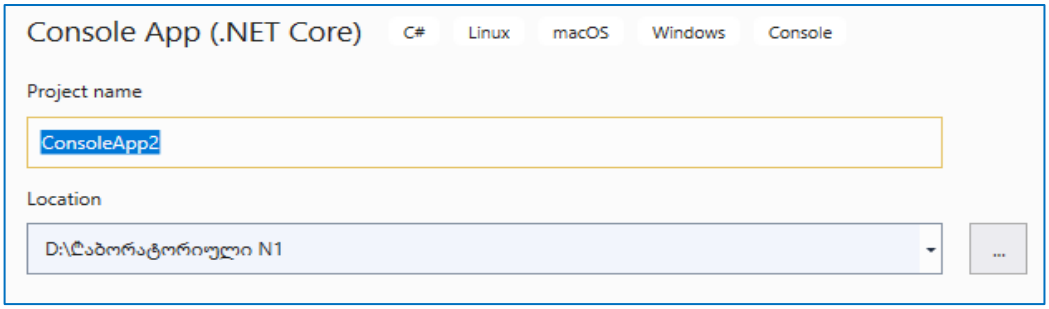
ამოცანა 1.2: ინტერაქტიული კოდის მაგალითი (მონაცემთა ტიპების გარდაქმნით). Solution „Lab_N1“-ზე მაუსის მარჯვენა ღილაკით დავამატოთ ახალი პროექტი (ნახ.2.8).

განსაზღვრეთ ახალი პროექტის სახელი ConsoleApp2 და მისი ლოკაციის ადგილი (ნახ.2.9).

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.



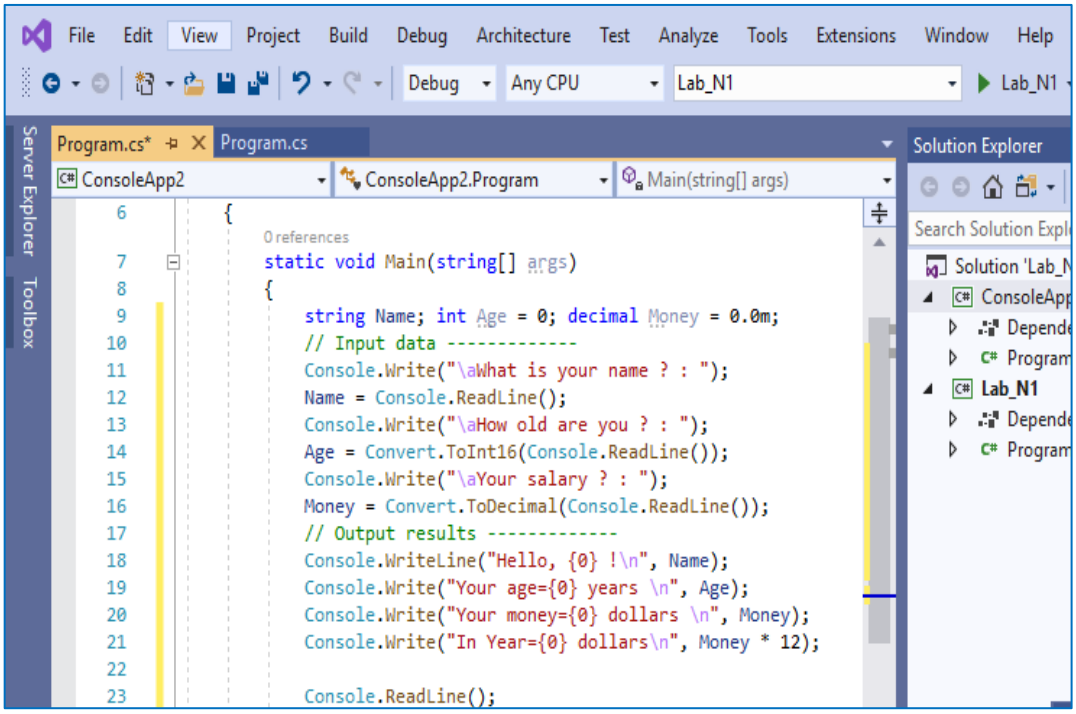
ნახ.2.8. ახალი პროექტის დამატება



ნახ.2.9. პროექტის სახელის და ლოკაციის განსაზღვრა

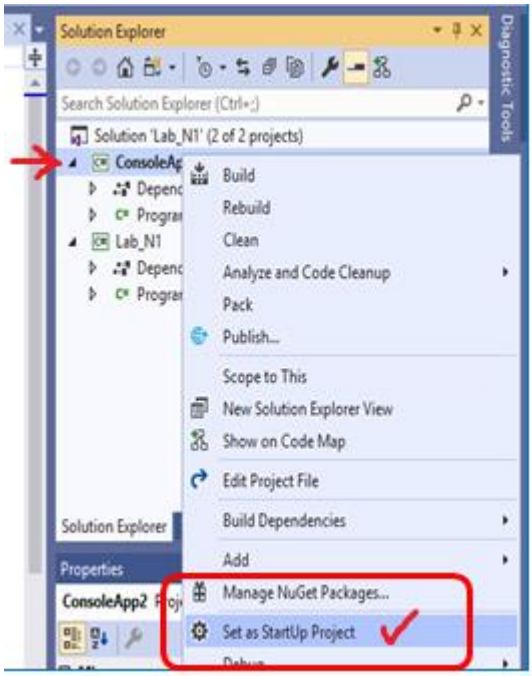
```
//--- ლისტინგი_1,1 ----
using System;
namespace ConsoleApp2
{
  class Program
```

```
{
 static void Main(string[] args)
  {
    string Name; int Age=0; decimal Money=0.0m;
   // Input data -------------
   Console.Write("\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()
}
```

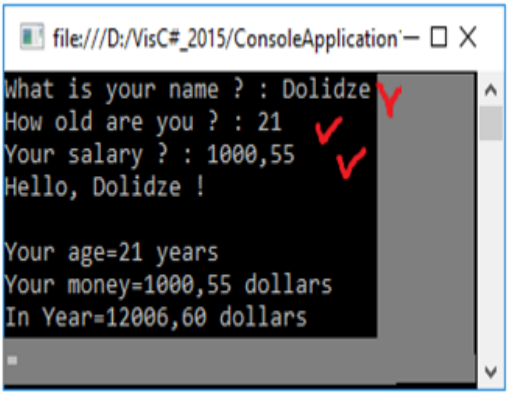


ნახ.2.10. კოდის ნიმუში სამუშაო გარემოში

შენიშვნა: პროგრამის ამუშავების შედეგად მვიღებთ ისევ წინა ამოცანის შედეგს (1.1 ამოცანისათვის!), რადგან გვაქვს ორი პროექტი. აქ აქტიურია Lab_N1, იგი მუქი ფერითაა აღნიშნული. საჭიროა გავააქტიუროთ ConsoleApp2, მაუსის მარჯვენა ღილაკით კონტექსტური მენიუდან ქვემოთ მოცემული ნიმუშის შეაბამისად ავირჩიოთ ბრძანება - Set as StartUp Project (ნახ.2.11).



ნახ.2.11. ახალი პროექტის გააქტიურება (StartUp-ით)



ნახ.2.12. ახალი პროექტის მუშაობის შედეგი

ამის შემდეგ ავამუშავოთ პროგრამა, შევიტანოთ მონაცემები ინტერაქტიულ რეჟიმში (დიალოგში) და გავაანალიზოთ შედეგი (ნახ.2.12).

დამოუკიდებელი სამუშაო: ააგეთ კონსოლის რეჟიმში C# კოდი, რომელიც ინტერაქტიულ რეჟიმში შეგვატანინებს ორ რიცხვს (a,b) და გაიანგარიშებს მათ ჯამს (S), სხვაობას (R), ნამრავლს (M), განაყოფს (D) და მოდულს (Mod). შედეგებს გამოიტანს კონსოლზე.

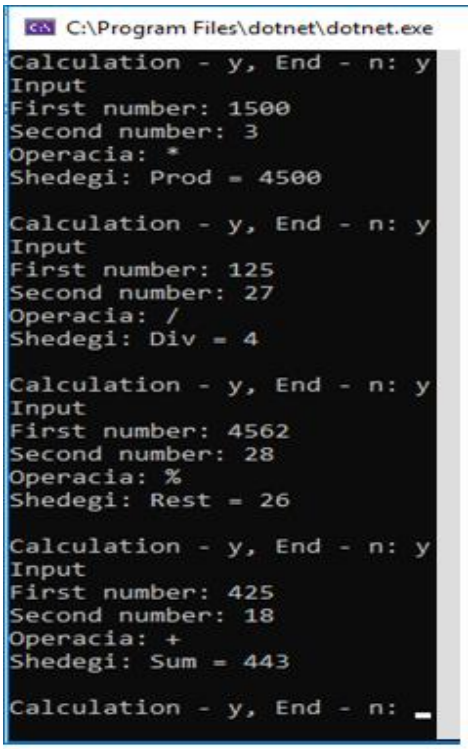
ამოცანა 1.3: დავწეროთ C# ინტერაქტიული კოდი მარტივი კალკულატორის მაგალითისათვის. დიალოგში შეიტანება ორი მთელი რიცხვი და ერთი არითმეტიკული ოპერაცია (+, -, * ან /). პროგრამას გამოაქვს გამოთვლის შედეგი. პროცესის გაგრძელება (ციკლური გამეორება) „Yes” ან დასასრული „No” (პროგრამიდან გამოსვლა).

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 !");
    }
  }
} //
```

არითმეტიკული ოპერატორების გამოყენების შედეგების ფრაგმენტი მოცემულია 2.13 ნახაზზე.



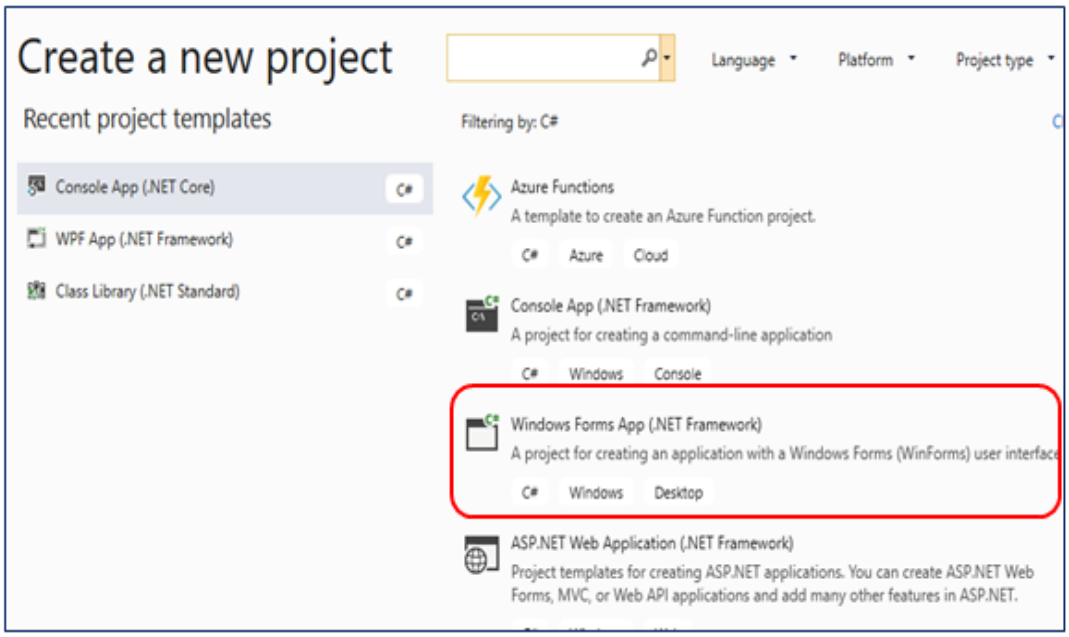
ნახ.2.13. არითმეტიკული ოპერაციების შედეგი

2.2. Windows ფორმებთან მუშაობა და ვიზუალური ელემენტები (Button, Label, TextBox)

სამუშაოს მიზანი: მონაცემების შეტანისა და შედეგების ეკრანზე გამოტანის ვიზუალური ელემენტების გაცნობა Windows Forms რეჟიმში.

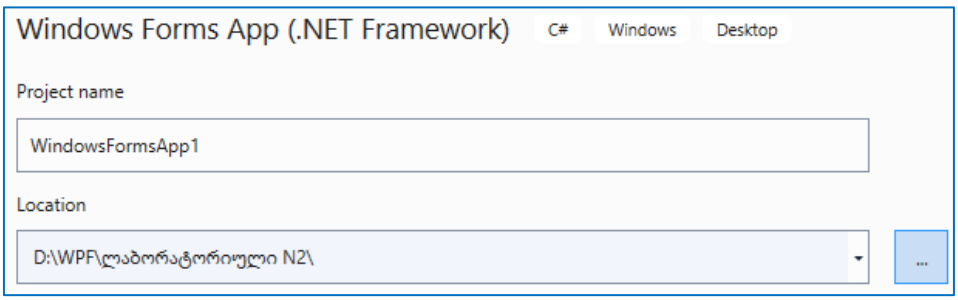
დავალება: ააგეთ ორი ფორმა (Form1, Form2), შემდეგი ელემენტებით: Button, Label, TextBox. პირველ ფორმაზე TextBox-ში შეიტანეთ „სიტყვა“. ღილაკის - „მეორე ფორმა“ ამოქმედებით უნდა გაიხსნას Form2 და მის TextBox-ში გამოჩნდეს პირველ ფორმაში შეტანილი სიტყვა. მეორე ფორმის ტექსტბოქსში შეასწორეთ ან დაამატეთ ახალი სიტყვა. „დახურვა“ ღილაკის ამოქმედებით უნდა დაიხუროს მეორე ფორმა და პირველ ფორმას გადაეცეს კორექტირებული სიტრიქონი.

Visual Studio 2019-ის სამუშაო გარემოდან Create a new project-ის, ინსტრუმენტის გააქტიურების შემდეგ მონიშნეთ Windows Forms App და ღილაკით Next გადადით ფანჯარაში, სადაც თქვენს მიერ D: დისკზე შემნილ ფოლდერში სახელით: ლაბორატორიული N2, შეინახავთ ახალ პროექტს - **WindowsFormsApp1**



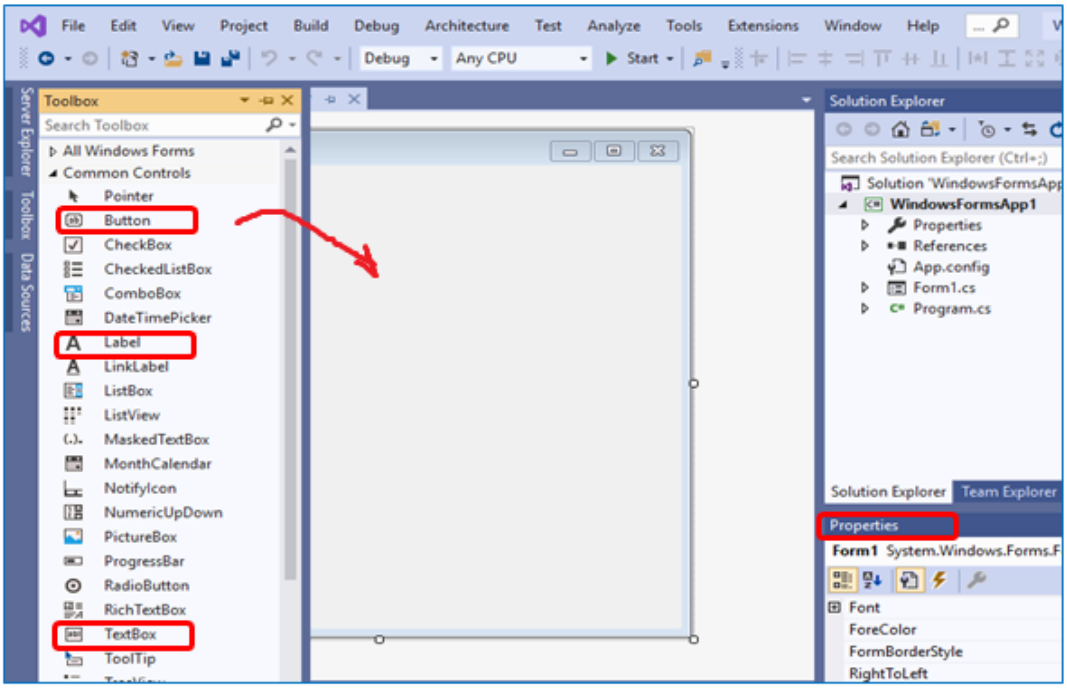
ნახ.2.14. Windows Forms App რეჟიმის არჩევა

შედეგად მიიღებთ 2.15 ნახაზზე ნაჩვენებ ფანჯარას.



ნახ.2.15. პროექტის სახელი და ლოკაციის ადგილი

მიღებული ფანჯრის ქვედა მარჯვენა მხარეს მოთავსებული Create ღილაკით გდახვალთ Visual Studio 2019-ის გარემოში, სადაც შესაძლებელია, 2.16 ნახაზზე მოცემული ნიმუშის შესაბამისი მომხმარებლის სამუშაო ფორმა - Form1-ის და ToolBox - ინსტრუმენტების პანელის მიღება.

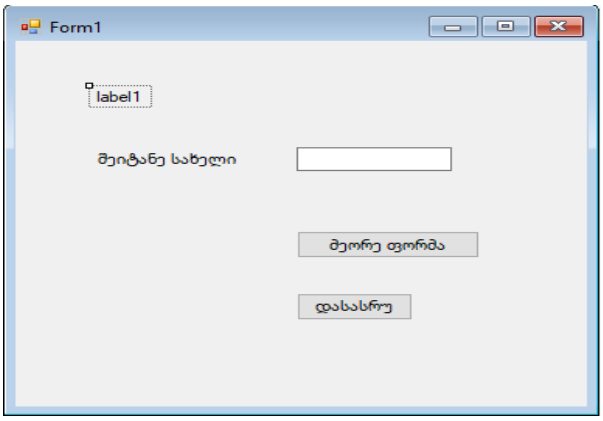


ნახ.2.16. ინსტრუმენტების პანელიდან ვიზუალური ელემენტების გადმოტანა სამუშაო ფორმაზე

ფანჯრის მარცხენა მხარეს მოთავსებული ინსტრუმენტების პანელიდან ხდება პროგრამისთვის საჭირო მართვის ელემენტის გადატანა ფორმაზე. ასევე ქვედა მარჯვენა ნაწილში მოთავსებულია ფორმის ელემენტების თვისებათა

(Properties) ფანჯარა, რომელიც ცალსახად აღწერს ფორმის ან მისი მონიშნული ელემენტ(ებ)ის თვისებებს.

ფორმაზე შესაბამისი ინსტრუმენტების გადატანის შემდეგ მას ექნება შემდეგი სახე (ნახ.2.17).



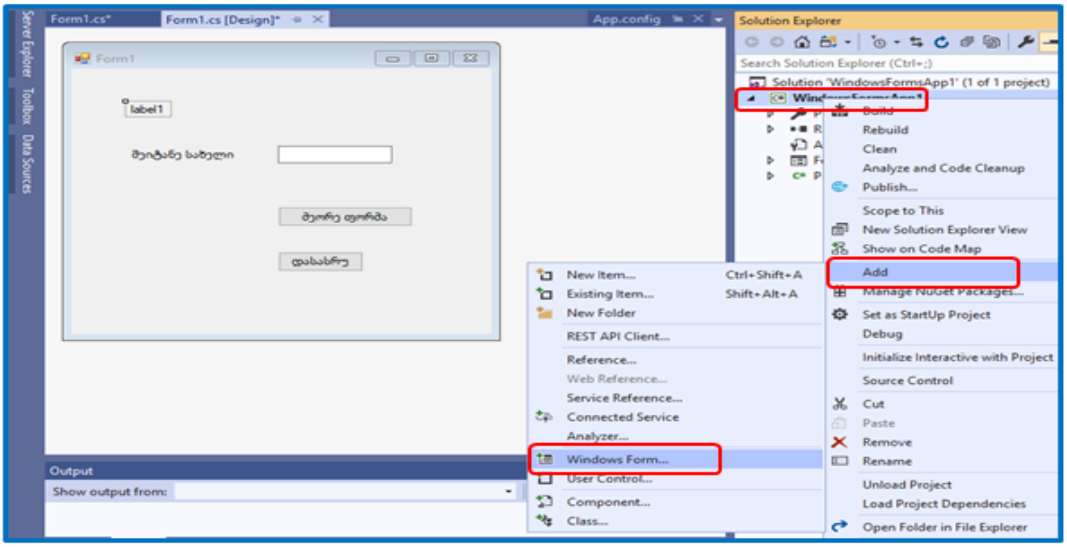
ნახ.2.17. აგებული Form1-ის ნიმუში

პირველი ფორმის პროგრამის ფრაგმენტის ლისტინგი:

```
// WindowsFormsApp1.cs---: 1-ელი ფორმისთვის -------

using System;
using System.Windows.Forms;

namespace WindowsFormsApp1
{
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;

      }

    }
}
```

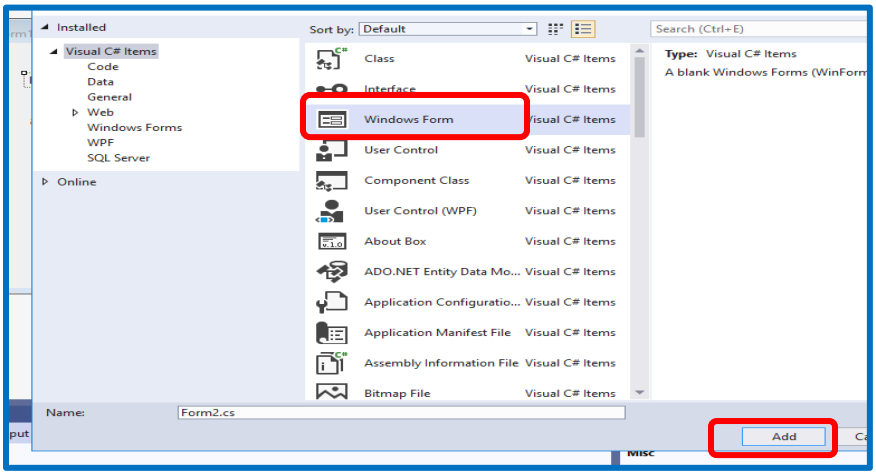
პროექტის შედგენილობა ასახულია Solution Explorer ფანჯარაში, სადაც იერარქიული ხის სტრუქტურით, განთავსებულია მისი შემადგენელი კომპონენტები: ფორმები, რესურსები, პროგრამული კოდები (Program.cs) და ა.შ.

აქ შესაძლებელია ახალი ელემენტების დამატება. მეორე ფორმის დასამატებლად უნდა შესრულდეს შემდეგი ბრძანებები თანამიმდევრულად: WindowsFormsApplication1→Add→WindowsForms (ნახ.2.18).



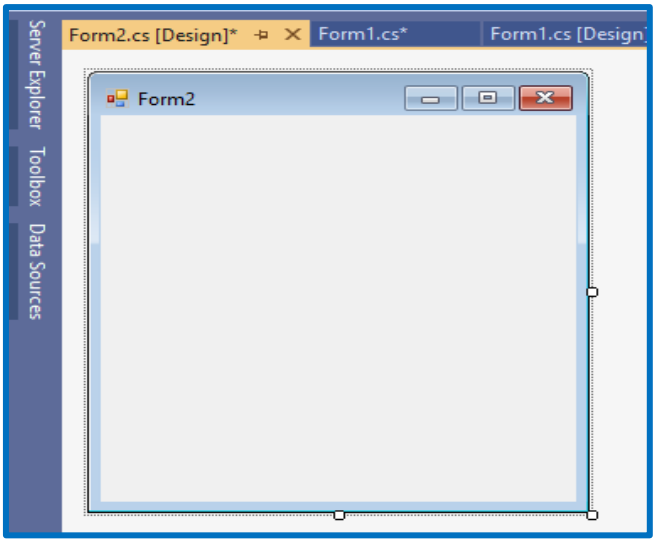
ნახ.2.18. Form2-ის დამატება პროექტში

მიიღებთ ფანჯარას, სადაც მონიშნავთ Windows Forms და აქტიურებთ ღილაკს Add (ნახ.2.19).



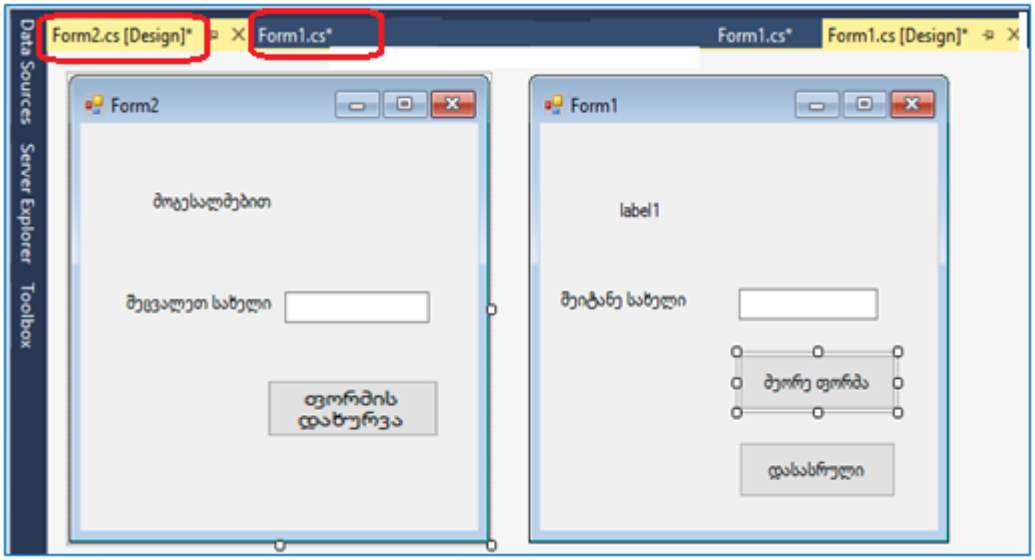
ნახ.2.19. Windows Forms არჩევა მეორე ფორმისთვის

შედეგად მიიღებთ Form2 ფანჯარას (ნახ.2.20).



ნახ.2.20. შედეგი ცარიელი Form2-ით

როგორც ქვემოთ მოცემული ნიმუშიდან ჩანს (ნახ.2.21), მომხმარებელს ეძლევა საშუალება მეორე ფორმაზეც განათავსოს ელემენტები და ორივე ფორმასთან იმუშაოს ერთდროულად.



ნახ.2.21. სამუშაო გარემო ორი ფორმით

Form1-ის კოდში საჭიროა ჩაემატოს ინფორმაცია Form2-ის შესახებ. კერძოდ, მისი როგორც ახალი ობიექტის (f2) შექმნის, მისი გახსნის (Show() მეთოდით) და პირველ ფორმაზე textBox-ში შეტანილი სტრიქონის გადასაცემად:

```
// 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();
    }
  }
}
```

Form2-ის პროგრამულ კოდში საჭიროა ჩაემატოს ინფორმაცია Form1-ის შესახებ. კერძოდ, რომ f1 არის მთავარი ფორმის Form1-ის ობიექტი, ასახოს Form1-დან გადმოცემული ინფორმაცია თავის textBox-ში (Form2_Load).

Form2-ის textBox-ში სტრიქონის ცვლილების შემდეგ „ფორმის დახურვა“ ღილაკის ამოქმედებით, ინფორმაცია დაუბრუნოს Form1-ის textBox-ს.

აღნიშნული კოდი მოცემულია ქვემოთ ლისტინგში.

```
// WindowsFormsApp1.cs---: მე-2 ფორმისთვის ---
using System;
using System.Windows.Forms;
namespace WindowsFormsApp1
{
  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();
    }
  }
}
```

გამართეთ და აამუშავეთ პროგრამული აპლიკაცია.

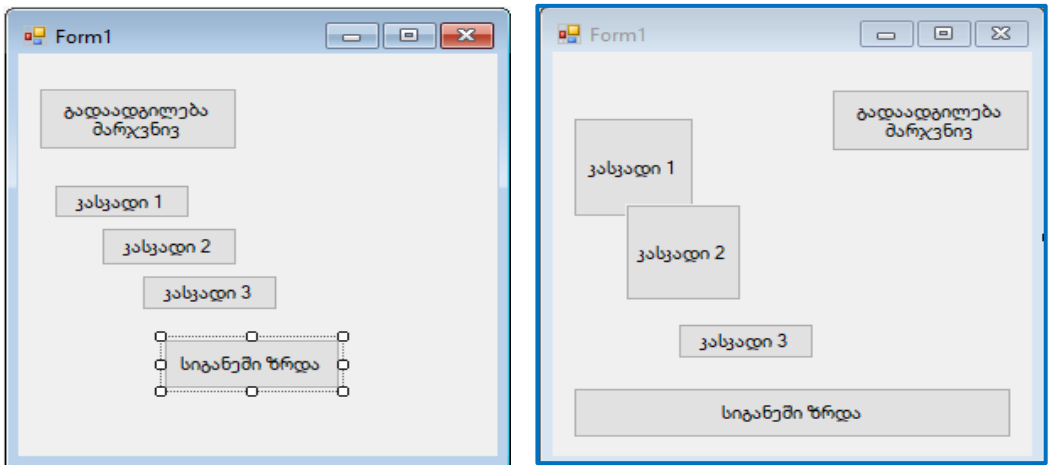
2.3. ვიზუალური ელემენტების ზომებთან და ფორმაზე მათი მდებარეობის კოორდინატებთან მუშაობა (Size, Location)

სამუშაოს მიზანი: ვინდოუს ფორმაზე ელემენტების ასახვის მართვის საშუალებების გაცნობა, შესაბამისი ობიექტების ზომებისა და მათი ფანჯარაში მდებარეობის ან განთავსების გათვალისწინებით.

Windows Forms დიზაინის სრულყოფისა 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 (მაგ., ყველა ტექსტბოქსში ერთი ზომისა და ფერის ქართული ფონტი);
- 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 თვისესბები Size: Width/Height (ზომა: სიგანე/სიმაღლე) და Location: X/Y (მდებარეობა: კოორდინატები ფორმის ზედა-მარცხენა კუთხიდან). სიდიდეები პიქსელებში მოიცემა. 2.22 ნახაზზე წრმოდგენილია ფორმა ხუთი ღილაკით (რედაქტირების და მუშაობის რეჟიმებით), ასევე კოდის ლისტინგი 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.2.22

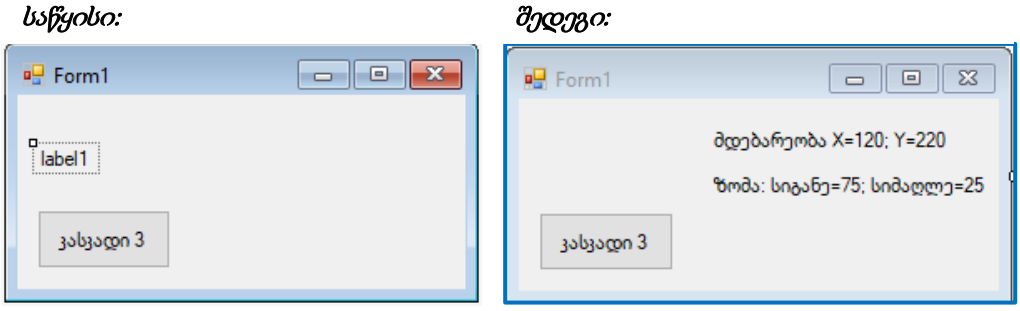
ამოცანა_1: ააგეთ ფორმა (Form1) ხუთი ღილაკით (button) და ქართული წარწერებით. თითოეული ღილაკის სამუშაოდ (მოვლენა - Event) დაწერეთ კოდი, რომელიც გადააადგილებს ან ზომებს შეუცვლის ამ ღილაკებს (იხ. ლისტინგი):

```
// ლისტინგი:  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)
          { // სიგანეში გაფართოება 20-პიქსელით მაუსის ერთ დაწკაპუნებაზე
            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);
          }
    }
}
```

დამოუკიდებელი სამუშაო: ფორმაზე ტექსტურ ელემენტში მრავალ-სტრიქონიანი 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. 2.23 ნახაზზე ნაჩვენებია ღილაკი, რომლის ამოქმედებით მიიღება ტექსტური შედეგი label1-ელემენტში.



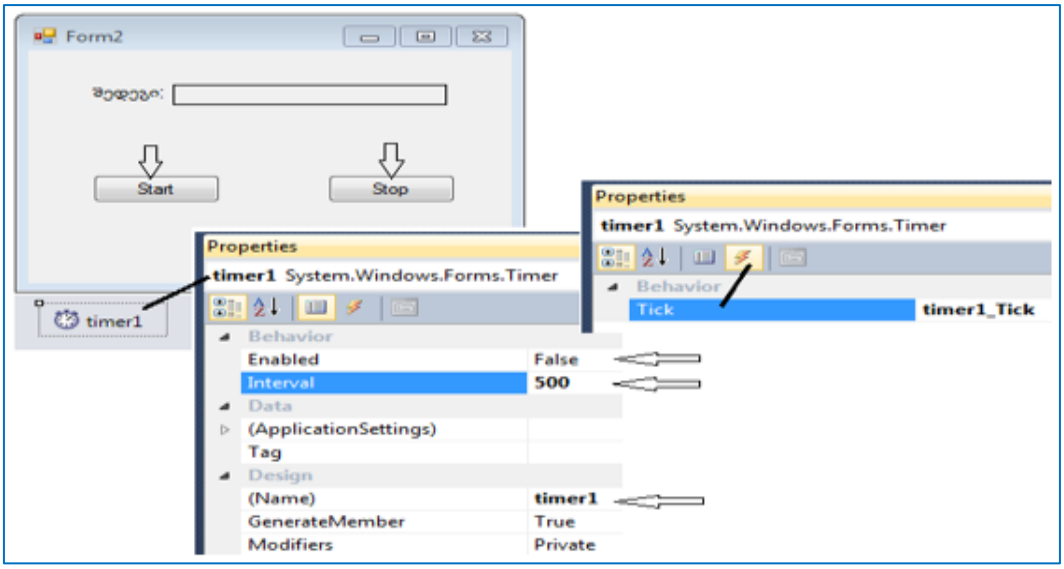
ნახ.2.23. ფორმაზე ელემენტის ზომებისა და მდებარეობის კოორდინატების პროგრამულადგანსაზღვრა

2.4. მართვის ელემენტი Timer

სამუშაოს მიზანი: პროგრამაში მოვლენების ამოქმედების, მათი მუშაობის და დასრულების დროითი მაჩვენებლების ანალიზის შესწავლა, შედეგების ეკრანზე გამოტანით.

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.

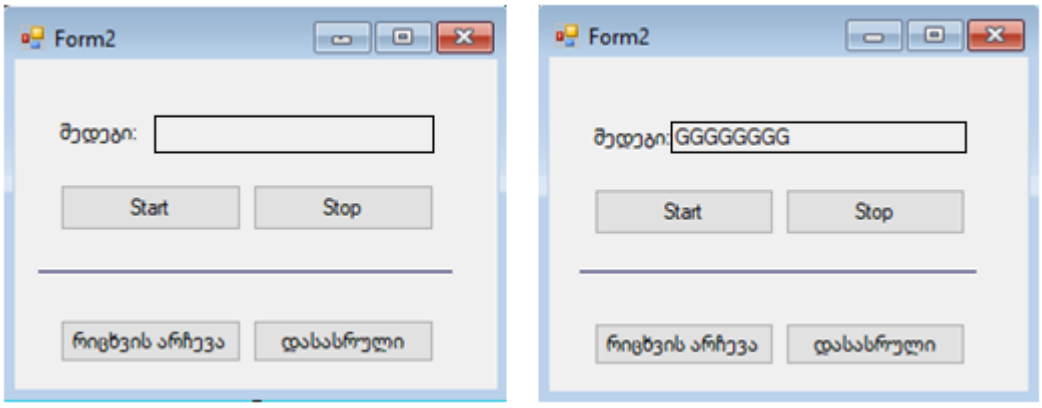
2.24 ნახაზზე წარმოდგენილ პანელზე ჩანს ღილაკი Timer, რომლის ამუშავებითაც გამოიძახება Form2 და მომზადდება მასზე ორი ღილაკი: Timer-ის მოვლენის ამოქმედების (Start) და მოვლენის შეჩერების (Stop). ამ ღილაკების მანიპულირებით „შედეგში“ გამოიტანება „G” სიმბოლოს კონკატენაციით მიღებული სტრიქონი - ინტერვალით 500, რაც შეესაბამება 0.5 წამს.



ნახ.2.24.

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 ლისტინგში, სადაც Form2-ზე განლაგებული Start, Stop ღილაკებიცაა აღწერილი. 2.25 ნახაზზე ნაჩვენებია საბოლოო შედეგის ამსახველი ფორმა.

```
// ლისტინგი---- 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 შედეგით
      {
         label1.Text += "G";
      }
     private void button7_Click(object sender, EventArgs e)
      {
         Close();
      }
  }
}
```

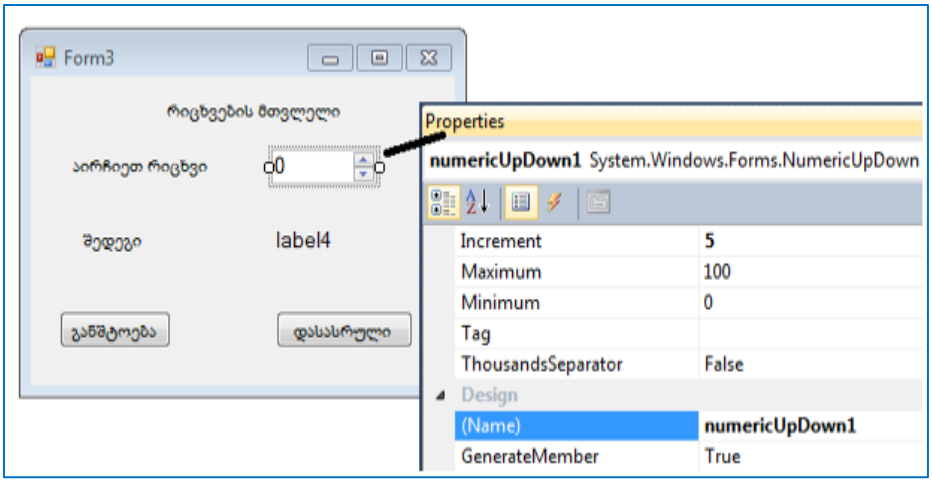


ნახ.2.25. საწყისი და საბოლოო შედეგის ფანჯრები

2.5. მართვის ელემენტი - რიცხვების არჩევა: NumericUpDown

სამუშაოს მიზანი: პროგრამაში NumericUpDown მთვლელის ელემენტის გამოყენების შესწავლა.

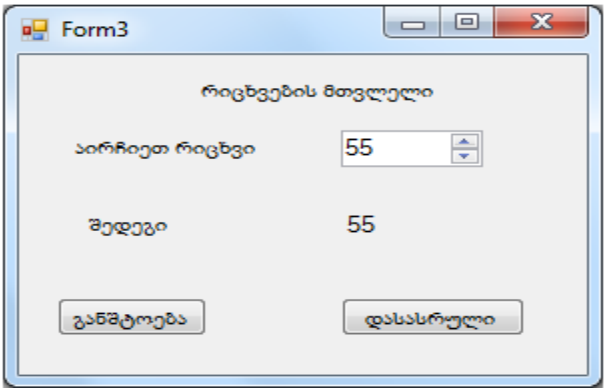
ელემენტი NumericUpDown საშუალებას იძლევა შევარჩიოთ საჭირო რიცხვი მთვლელის რეჟიმში (პატარა ისრები მარჯვენა მხარეს, რომლებითაც ხდება საწყისი რიცხვის (Value) მატება ან კლება Increment-თვისებაში მითითებული ბიჯით) ან ავკრიფოთ იგი ხელით კლავიატურიდან. Properties-ში მიეთითება აგრეთვე რიცხვის სავარაუდო Maximum და Minimum მნიშვნელობები. შედეგი აისახება label4-ში. კოდის ფრაგმენტი მოცემულია ლისტინგში.



ნახ.2.26.

```
// ლისტინგი – ValueChanged მოვლენა ----
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
   {
     label4.Text = numericUpDown1.Value.ToString();
   }
```

საბოლოო შედეგი მოცემულია 2.27 ნახაზზე.

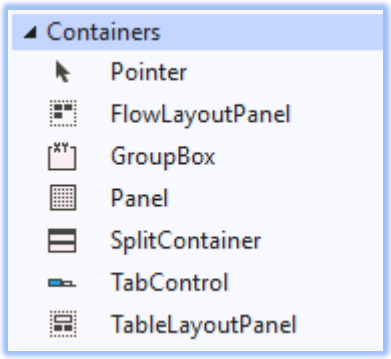


ნახ.2.27

დამოუკიდებელი სამუშაო: 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.

2.6. მართვის კონტეინერული ელემენტები

კონტეინერი ისეთი ელემენტია, რომელიც თვითონ შეიცავს სხვა ელემენტებს. მათ დიდი პრაქტიკული გამოყენება აქვს და წიგნის ამ ნაწილის რამდენიმე პარაგრაფში ჩვენ სწორედ ამ საკითხებს გავეცნობით. 2.28 ნახაზზე ნაჩვენებია Visual Studio.NET-ის ინსტრუმენტების პანელზე კონტეინერული ელემენტების ნაირსახეობა.



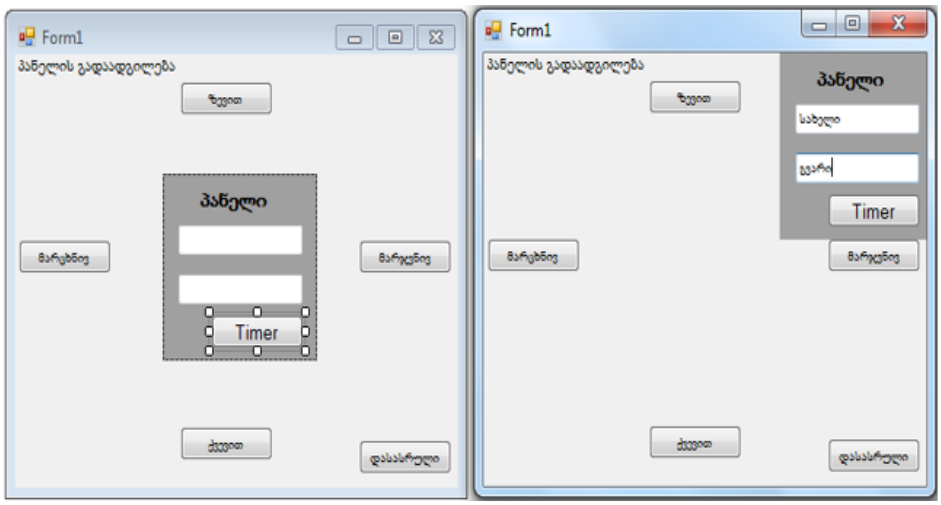
ნახ.2.28. Toolbox Containers - ფრაგმენტი

2.6.1. კონტეინერული ელემენტი - Panel

სამუშაოს მიზანი: Panel კონტეინერის გამოყენების შესწავლა. კონტეინერზე თავსდება მართვის ელემენტები (მაგალითად, label, textBox, comboBox და ა.შ.). პანელის გადაადგილებით ფორმაზე იგი თან გაიყოლებს მასზე მოთავსებულ კომპონენტებსაც. შესაძლებელია პანელის კოპირება თავის ელემენტებიანა სხვა ფორმაზე.

ამოცანა_1: 2.29 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

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



ნახ.2.29. პანელის გადაადგილება

```
// ლისტინგი – 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.

დამოუკიდებელი სამუშაო: ააგეთ პანელი ფორმის ცენტრში, მოათავსეთ მასზედ სხვა ელემენტები, მაგალითად, textBox და label და ამოძრავეთ ოთხივე მიმართულებით. პანელის მისვლისას ფანჯრის ნაპირებთან გაჩერდეს (ისე რომ არ დატოვოს ფორმა).

2.6.2. კონტეინერული და მართვის ვიზუალური ელემენტები: CheckBox, RadioButton, GroupBox

სამუშაოს მიზანი: კონტეინერული ელემენტების CheckBox, RadioButton და GroupBox თვისებების და მათი გამოყენების შესწავლა.

☐ / ☒- 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).

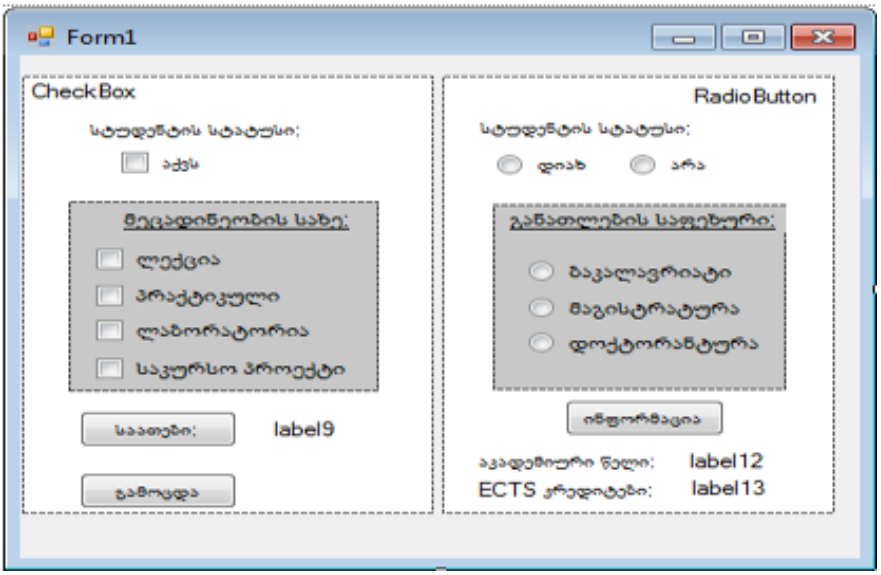
GroupBox - არის აგრეთვე Container კლასის ჯგუფური საკონტროლო უჯრა, შემოსაზღვრული ჩარჩოთი, რომელშიც შეიძლება მოთავსდეს რამდენიმე CheckBox, RadioButton ან სხვა ვიზუალური ელემენტი. ის აერთიანებს ერთგვაროვან მონაცემებს, რომელთა ტექსტური იდენტიფიკაცია ხდება მისი ჩარჩოს ზედა-მარცხენა კუთხეში.

ამოცანა_2: საჭიროა ავაგოთ ვინდოუს-ფორმა, რომელსაც აქვს 2.30 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.2.31). 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.

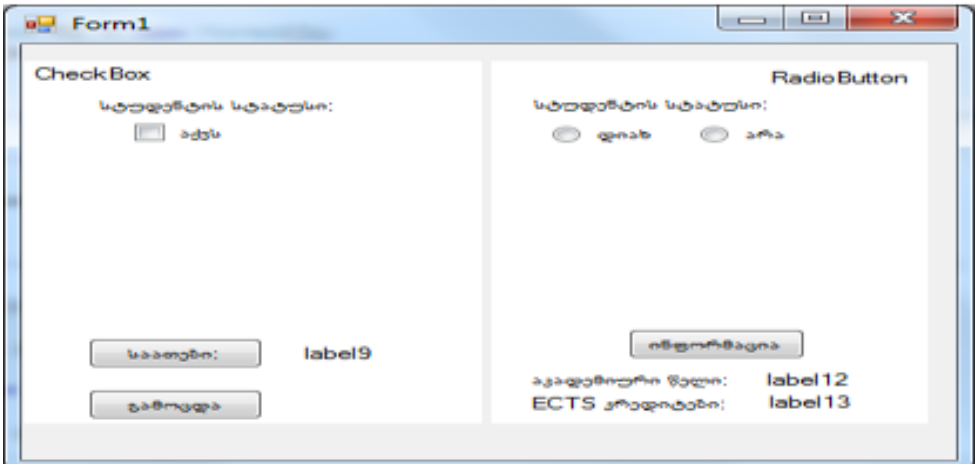
მარჯვენა პანელზე ნაჩვენებია radioButton გადამრთველის მაგალითი. თუ სტუდენტის სტატუსის „დიახ“ ბუტონში არაა მარკერი, მაშინ პანელი „განათლების საფეხური“ სამი radioButton-ით გამორთულია (ნახ.2.31. თუ არის მარკერი, მაშინ ეს პანელი ხილვადია და შეიძლება ერთ-ერთი ბუტონის არჩევა. შემდეგ „ინფორმაცია“ - ღილაკის ამოქმედებით label12 და label13 უჯრებში

გამოჩნდება შესაბამის მონაცემთა მნიშვნელობები: სასწავლო წლების რაოდენობა და ECTS-კრედიტების საერთო რაოდენობა. radioButton-ების გადართით შესაძლებელია სხვა ინფორმაციის მიღებაც.



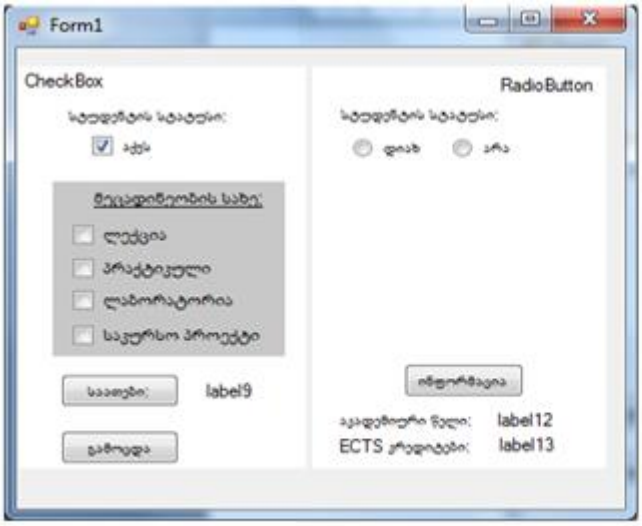
ნახ.2.30. საწყისი ფორმა

აქ გამოყენებულია ვერტიკალური SplitContainr ორი პანელით.



ნახ.2.31. პანელები გამორთულია

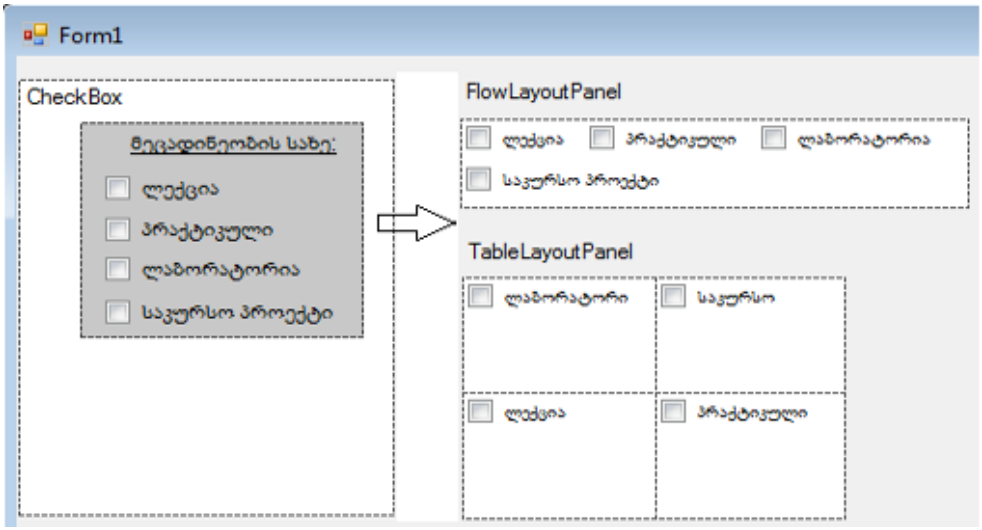
მარცხენა პანელზე ჩავრთოთ სტუდენტის სტატუსის checkBox. 2.32 ნახაზზე გამოჩნდება შედეგი:



ნახ.2.32

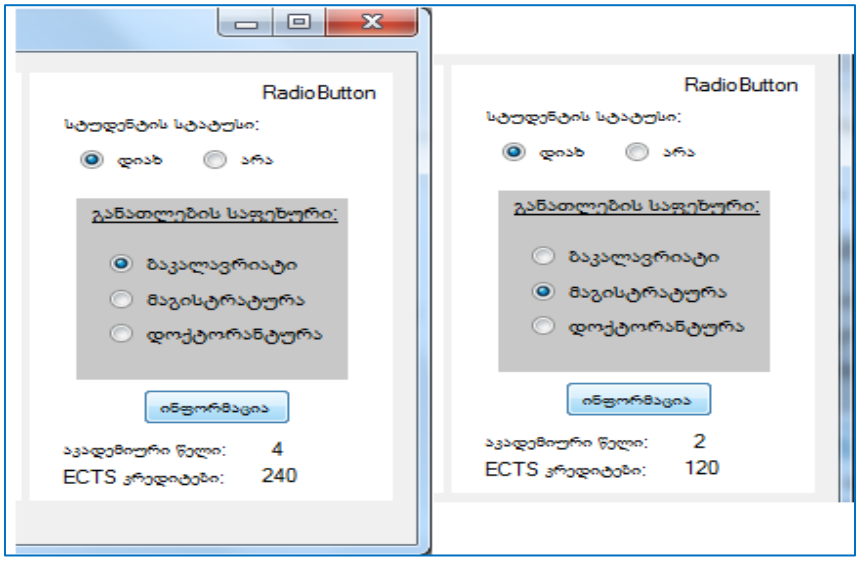
ნახ.2.33

პანელზე „მეცადინეობის სახე“ ჩავრთოთ checkBox-ები „ლექცია“, „პრაქტიკული“ და „ლაბორატორია“. შემდეგ ღილაკით „საათები“ label9-ში გამოჩნდება რიცხვი (ნახ.2.33). 2.34 ნახაზზე ნაჩვენებია checkBox-ების განლაგების ვარიანტები RowLayoutPanel და TableLayoutPanel კონტეინერული ელემენტების გამოყენებით.



ნახ. 2.34

ახლა იგივე პროცედურები ჩავატაროთ radioButton-ის მაგალითზე პანელის მარჯვენა ნაწილში. აქ (ნახ.2.31) ჩავსვათ მარკერი „დიახ“-ში და გამოჩენილ „განათლების საფეხურების“ პანელზე ავირჩიოთ რომელიმე ბუტონი. შემდეგ ღილაკით „ინფორმაცია“ მივიღებთ 2.35 ნახაზზე ნაჩვენებ შედეგს.



ნახ.2.35

აღნიშნული ამოცანის რეალიზაციის კოდი მოცემულია ლისტინგში.

```
// ლისტინგი --- 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 CheckedChanged - 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 = "  ";
    }
  }
}
```

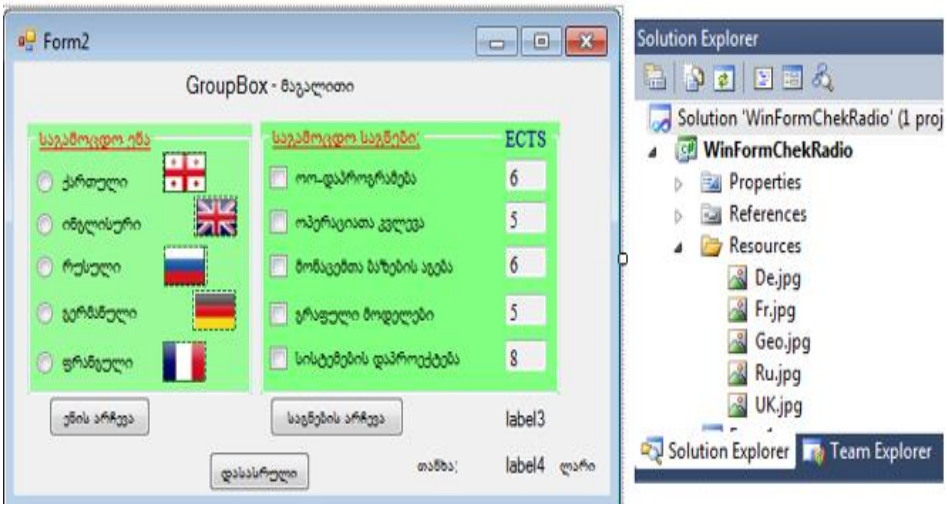
2.6.3. კონტეინერული ელემენტები: GroupBox და TabControl

სამუშაოს მიზანი: კონტეინერული ვიზუალური ელემენტების GroupBox და TabControl შესწავლა.

➢ კონტეინერი GroupBox

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.

GroupBox კონტეინერი მსგავსია ჩვენ მიერ ადრე განხილული Panel ელემენტისა, რომელზეც თავსდებოდა 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 2.36 ნახაზზე.



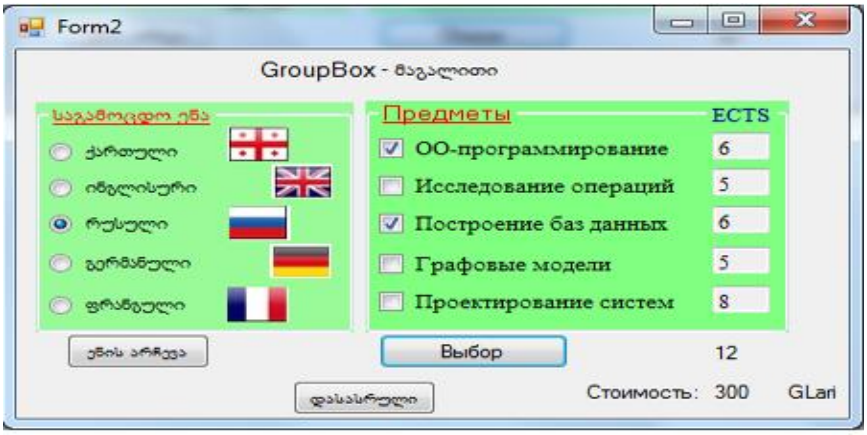
ნახ.2.36. ინტერფეისის საწყისი ფორმა

ნახაზზე დროშებისთვის (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, 2.37 და 2.38 ნახაზებზე ნაჩვენებია ინგლისური და რუსული ენების ვარიანტები.

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 ლისტინგში.

ნახ. 2.37. En



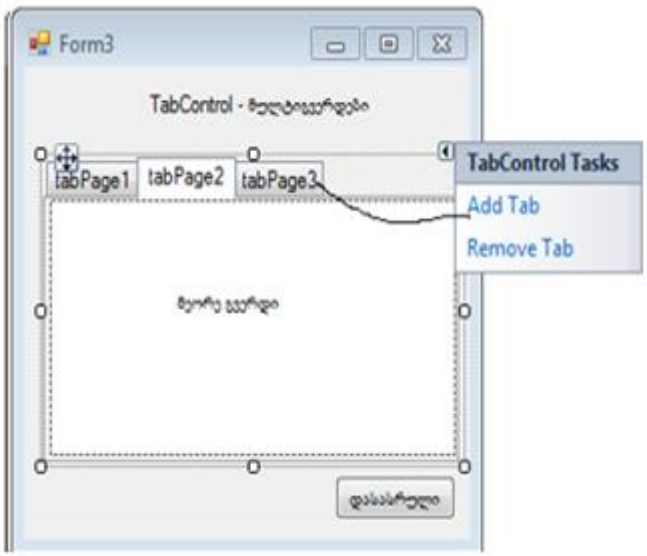
ნახ.2.38. Ru

```
// ლისტინგი ---- 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){ }
 private void button1_Click(object sender, EventArgs e) //ქართ.ენის არჩევა
        {
            if (radioButton1.Checked)
               {
                groupBox2.Text = "საგამოცდო საგნები";
                button2.Text = "აირჩიეთ";
                checkBox1.Text="ოო-პროგრამირება";
                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="Oprational 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 მრავალ-გვერდიანი დიალოგის ორგანიზებისთვის ურთიერთგადაფარვის პრინციპით (ნახ.2.39). ახალი გვერდის დამატება ხდება Add Tab, ხოლო ძველის წაშლა Remove Tab სტრიქონებით კონტექსტურ მენიუდან, რომელიც გამოიტანება მარჯვენა კუთხეში პატარა ისარზე მაუსის მარჯვენა ღილაკით. თითოეული გვერდის სახელის ჩაწერა ხდება შესაბამისი გვერდის Properties-დან Text-ში.



ნახ.2.39. კონტეინერი TbaControl მაგალითი

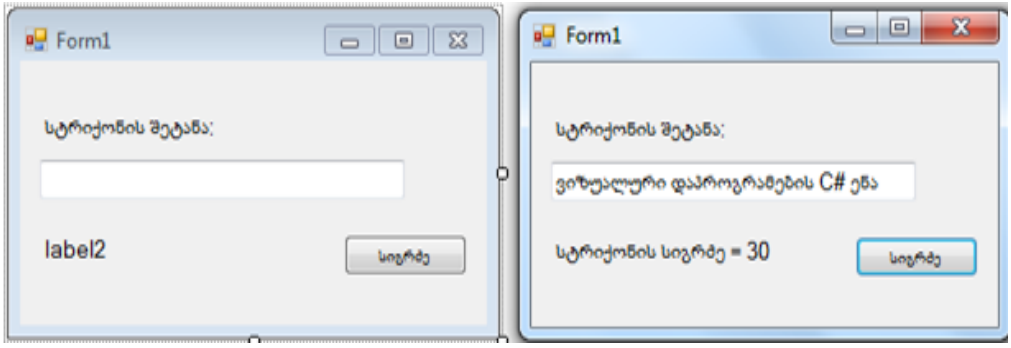
2.7. სტრიქონული ტიპის მონაცემების დამუშავება

2.7.1. string - სტრიქონული ტიპის მონაცემები და String კლასი

სამუშაოს მიზანი: ტექსტებისა და სტრიქონული ტიპის მონაცემების დამუშავების პრინციპების და მეთოდების შესწავლა.

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.

ამოცანა_1: დავდოთ ფორმაზე (ნახ.2.40) სტრიქონის შესატანი textbox1 ველი და label2 - სტრიქონის სიგრძის მნიშვნელობის გამოსატანი ველი. ღილაკით „სიგრძე“ მოხდეს სტრიქონის სიგრძის დათვლა და მიშვნელობის გამობეჭდვა. შედეგი მოცემულია ქვემოთ წარმოდგენილ ნახაზზე, ხოლო ღილაკის შესაბამისი კოდი - ლისტინგში.



ნახ.2.40. Striqoni.Length თვისების გამოყენების მაგალითი

```
// ლისტინგი --- String -- სტრიქონული ტიპი ----
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.

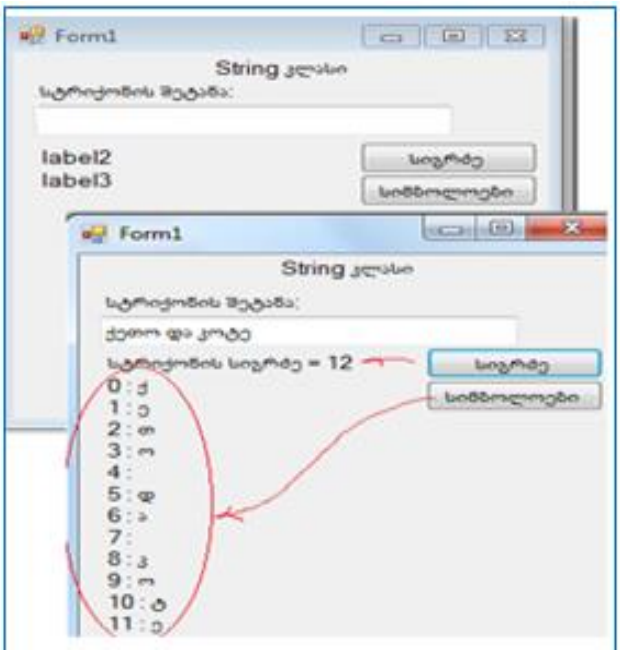
სტრიქონში სიმბოლოები ჩალაგებულია მასივის მსგავსად, ანუ ერთი სიმბოლო ერთი ელემენტია და თავისი ინდექსი აქვს (იხ.ცხრილში [ქეთო და კოტე]).

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

დავუმატოთ Form1 ფორმას ერთი ღილაკი „სიმბოლოები“ და მივაბათ მას კოდი, რომელიც ლისტინგშია მოცემული:

```
// ლისტინგი -- 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";
    }
  }
```

შედეგები მოცემულია 2.41 ნახაზზე.



ნახ.2.41. პროგრამის შედეგი

➢ **სტრიქონის დამუშავების მეთოდები: Trim() და Replace()**

სტრიქონებთან მუშაობისას ხშირად ვხვდებით ტესქტში ცარიელ სიმბოლოებს (პრობელებს), რომელთა დამუშავება (ამოყრა, შემცირება და ა.შ.) აუცილებელია. ამისათვის ბიბლიოთეკაში რამდენიმე მეთოდია:

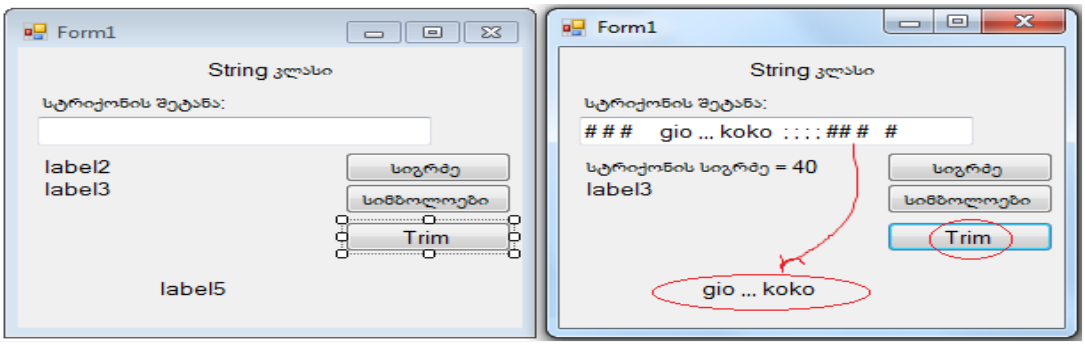
Trim() – არასასურველი სიმბოლოების ამოყრა სტრიქონის თავიდან ან ბოლოდან, მათ შორის პრობელებისაც;

TrimStart() – არასასურველი სიმბოლოების ამოყრა სტრიქონის თავიდან;

TrimEnd() – არასასურველი სიმბოლოების ამოყრა სტრიქონის ბოლოდან;

Replace() – ტექსტის შუაგულიდან არასასურველი სიმბოლოების (პრობელებისაც) ამოგდების ან შეცვლის მეთოდი.

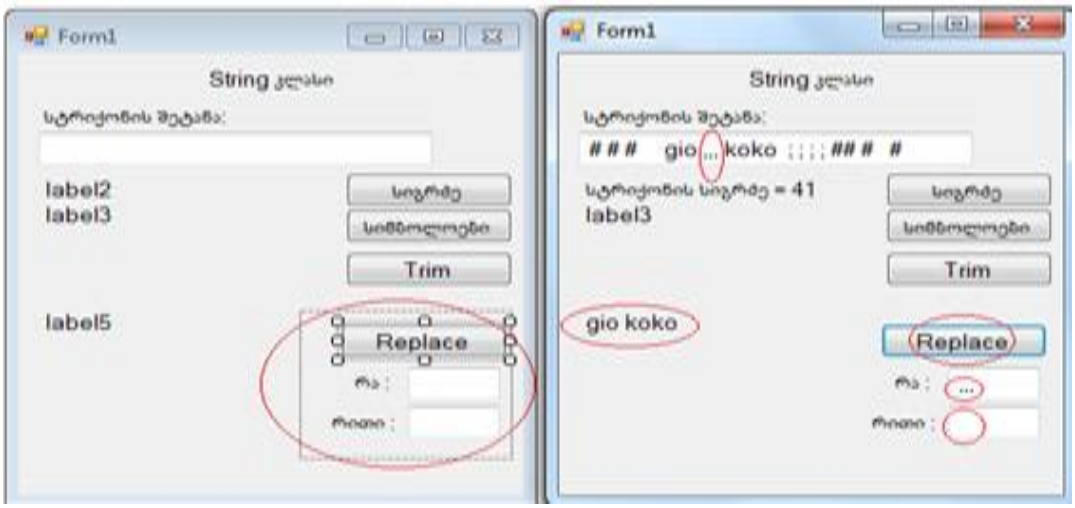
2.42 ნახაზზე ნაჩვენებია Trim მეთოდის მაგალითი, მისი კოდის ფრაგმენტი აღწერილია შესაბამის ლისტინგში.



ნახ.2.42. სტრიქონის გაწმენდა Trim მეთოდით

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

როგორც შედეგიდან ჩანს, სტრიქონის თავში და ბოლოში აღარაა ზედმეტი სიმბოლოები. დარჩა მხოლოდ „ ,,, „ სტრიქონის შუაში. ახლა Replace() მეთოდი (იხ. ლისტინგი) გამოვიყენოთ, რისთვისაც ფორმაზე დავამატოთ ეს ღილაკი (ნახ.2.43).



ნახ.2.43. Replace მეთოდის დამატება

```
    // ლისტინგი ---  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;
  }
```

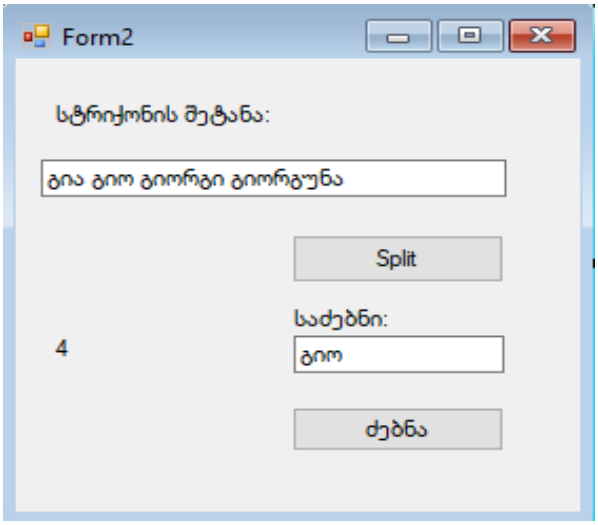
Replace() მეთოდი მოითხოვს ორი დამატებითი ტექსტბოქსის გამოყენებას. რომლებშიც მიეთითება სტრიქონის შიგნით „რა“ უნდა შეიცვალოს „რითი“.

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

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

სამუშაოს მიზანი: სტრიქონებში და ქვესტრიქონებში სიმბოლოთა ძებნის მეთოდების IndexOf(), LastIndexOf() და IndexOfAny() შესწავლა.

ფორმაზე (ნახ.2.44) ძირითად სტრიქონთან (textBox1) ერთად მოვათავსოთ საძებნი ქვესტრიქონი (textBox2) და ღილაკზე „ძებნა" მივაბათ ქვემოთ მოცემული ლისტინგის კოდი.

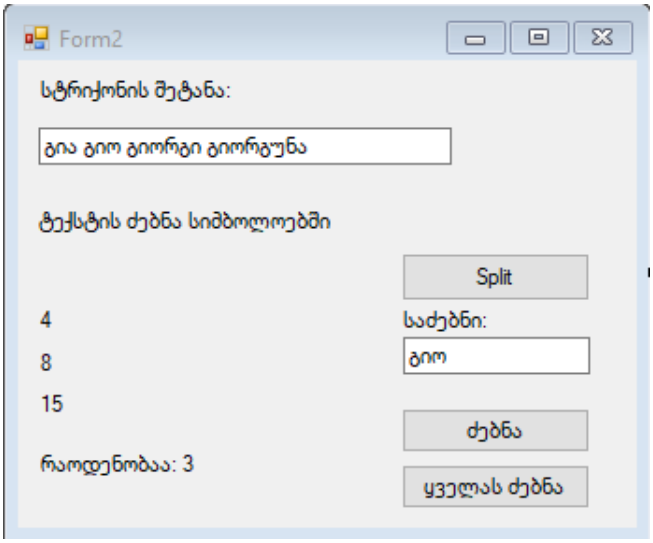


ნახ.2.44.

```
// ლისტინგი--- 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;
       }
```

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

თუ საჭიროა საძებნი ქვესტრიქონის ძირითად სტრიქონში არსებობის ყველა შემთხვევის დაფიქსირება, მაშინ ძებნის ალგორითმი მიიღებს ქვემოთ ლისტინგში მოცემული კოდის სახეს, ხოლო შედეგები იქნება 2.45 ნახაზზე.



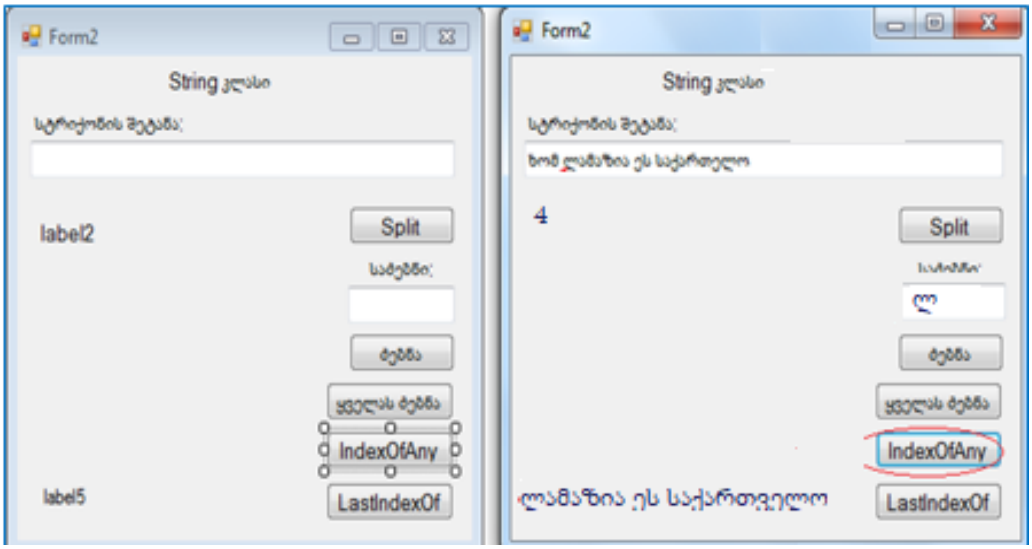
ნახ.2.45. შედეგი: All_IndexOf()

```
// ლისტინგი --- 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 ძებნა 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;
  }
```

როგორც ვხედავთ, do…while ციკლის შიგნით ხდება საძებნი ქვესტრიქონის მრავალჯერადი მოძიება ძირითად სტრიქონში. საწყისი საძიებო 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. 2.46 ნახაზზე ნაჩვენებია ეს შემთხვევა, ხოლო ლისტინგში IndexOfAny ღილაკის კოდი.

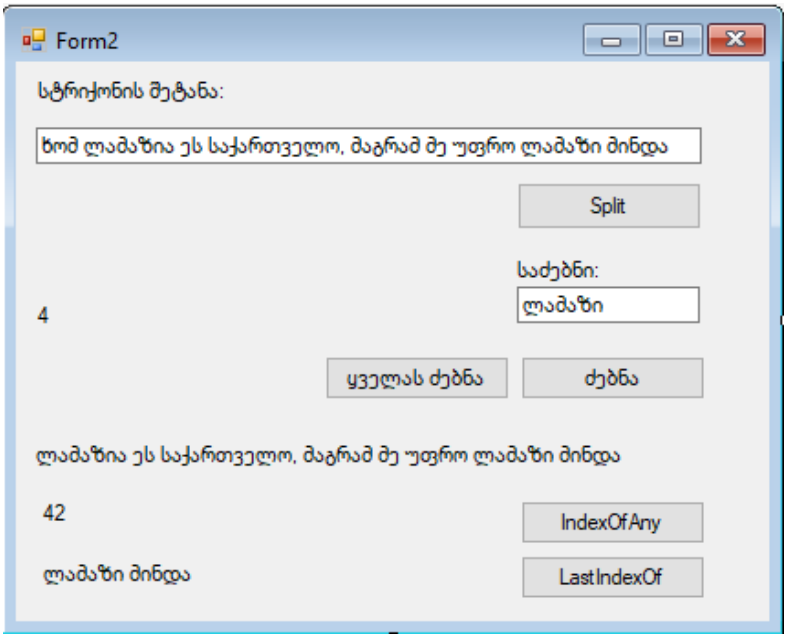
```
// ლისტინგი --- IndexOfAny ----
private void button4_Click(object sender, EventArgs e)
   {
        string Striqoni;
        int pozicia;
        label2.Text = "";
        Striqoni = textBox1.Text;
        textBox2.Text = "ლ";  // ან "ს"
        pozicia = Striqoni.IndexOfAny(new char[] { 'ლ', 'ს'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



ნახ.2.46. შედეგი: IndexOfAny()

ახლა განვიხილოთ LastIndexOf () მეთოდი. იგი ძირითად სტრიქონში პოულობს მითითებული სიმბოლოს ან სიტყვის მიხედვით ქვესტრიქონს, ოღონდ ბოლოდან (არა პირველს მარცხნიდან). შესაბამისი კოდი მოცემულია ქვემოთ ლისტინგში. შედეგებში 2.47 ნახაზზე კარგად ჩანს განსხვავება IndexOfAny() და LastIndexOf () მეთოდებს შორის.

```
// ლისტინგი --- LastIndexOf ----
 private void button5_Click(object sender, EventArgs e)
 {
    string Striqoni;
    int pozicia;
    label2.Text = ""; label5.Text = ""; label7.Text = "";
    Striqoni = textBox1.Text;
    textBox2.Text = "ლამაზი";
    pozicia = Striqoni.LastIndexOf("ლამაზი");
    label5.Text += pozicia;
    label7.Text += Striqoni.Substring(pozicia);
 }
```

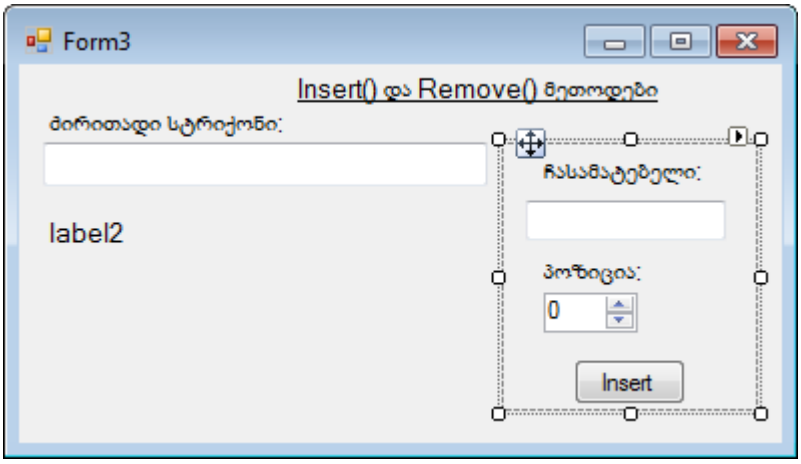


ნახ.2.47. . შედეგი: LastIndexOf ()

2.7.3. სტრიქონებთან მუშაობა : Insert (), Remove(), Substring()

სამუშაოს მიზანი: სტრიქონებში და ქვესტრიქონებში სიმბოლოთა ძებნის მეთოდების Insert (), Remove() და Substring() შესწავლა.

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 მითითებულ რიცხვის მიხედვით (numericUpDown1) (ნახ.2.48). 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.

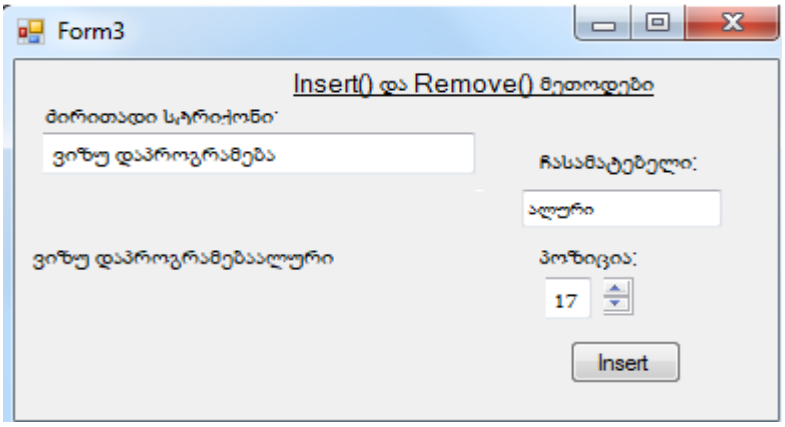


ნახ.2.48. Insert() მეთოდი

პროგრამის კოდი მოცემულია ლისტინგში

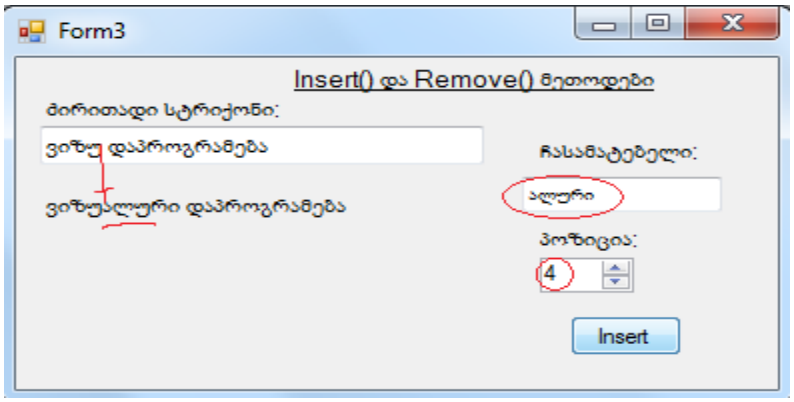
```
// ლისტინგი --- Insert() ----------
private void button1_Click(object sender, EventArgs e)
  {
    string Striqoni, qveStriqoni;
    Striqoni = textBox1.Text;
    qveStriqoni = textBox2.Text;
label2.Text = Striqoni.Insert((int) numericUpDown1.Value, qveStriqoni);
  }
```

შედეგი მოცემულია 2.49 ნახაზზე.



ნახ.2.49. შედეგი: Insert() მეთოდის გამოყენება

აქ ლებელში ჩანს სიტყვა „ვიზუალური“, რომელიც Insert() მეთოდით განხორციელდა. თუ პოზიციას შევცვლით, მაგალითად „15“-ით, მერე „17“ და მივიღებთ 2.50 ნახაზზე მიღებულ უაზრო შედეგს. ამის მეტი მთვლელი არ გაზრდის მნიშვნელობას, ვინაიდან ის ძირითადი სტრიქონის სიგრძეა.

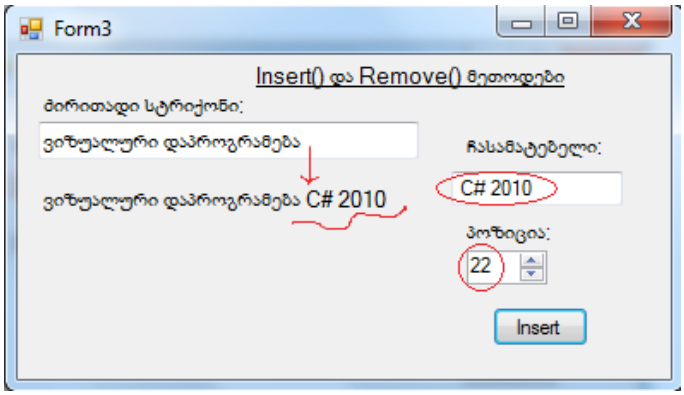


ნახ.2.50. ცვლილების შედეგი

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

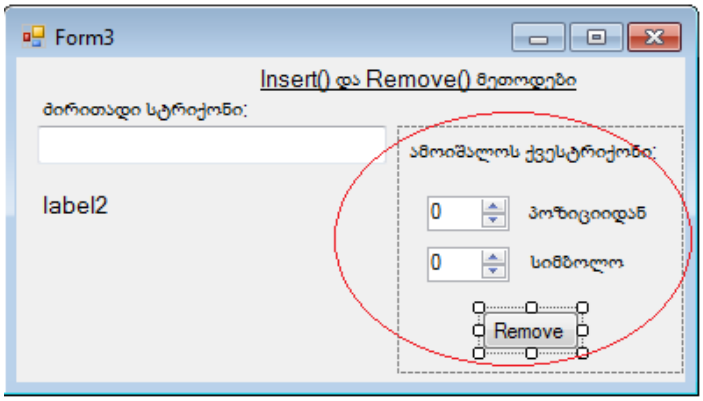
```
// ლისტინგი_6.6 ---
 private void textBox1_TextChanged(object sender, EventArgs e)
    {
        string Striqoni;
        Striqoni = textBox1.Text;
        numericUpDown1.Maximum = Striqoni.Length;
    }
```

როგორც კი დაიწყება ძირითადი სტრიქონის textBox1-ში ტექსტის ცვლილება, მაშინვე იმუშავებს ეს კოდი და შეიცვლება სტრიქონის სიგრძის შემზღუდველი რიცხვი (ნახ.2.51) `numericUpDown1.Maximum` -ში:



ნახ.2.51.

- Remove() მეთოდი უზრუნველყოფს სტრიქონიდან ქვესტრიქონის ამოშლას, მითითებული პოზიციისა და სიმბოლოების რაოდენობის მიხედვით. დავამატოთ ფორმაზე ეს ელემენტები (ნახ.2.52).

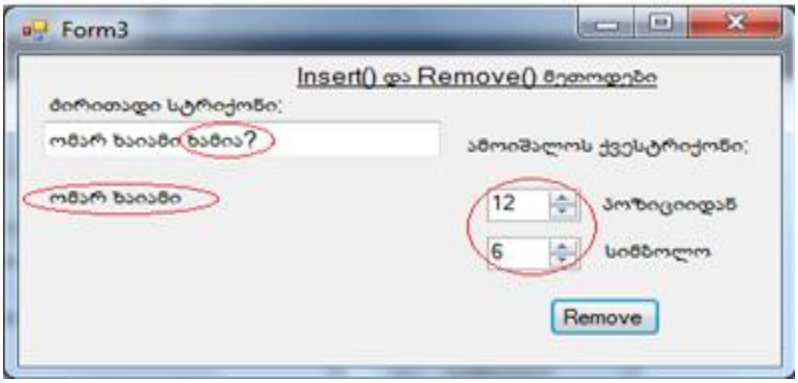


ნახ.2.52

```
// ლისტინგი --- 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 2.53 ნახაზზე.



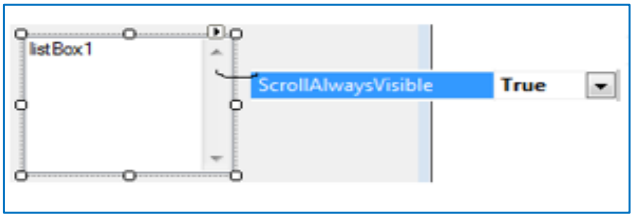
ნახ.2.53

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“.

2.8. მართვის ვიზუალური ელემენტები: ListBox, CheckedListBox

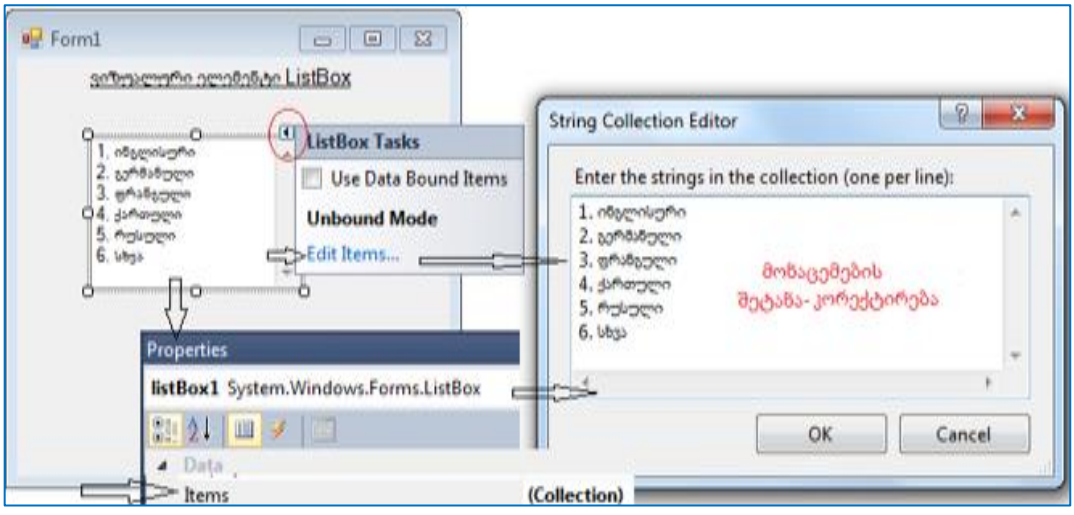
სამუშაოს მიზანი: მართვის ელემენტების ListBox და CheckedListBox გამოყენების სინტაქსის შესწავლა, თვისებები და მეთოდები დაპროგრამების ამოცანებში.

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.2.54).



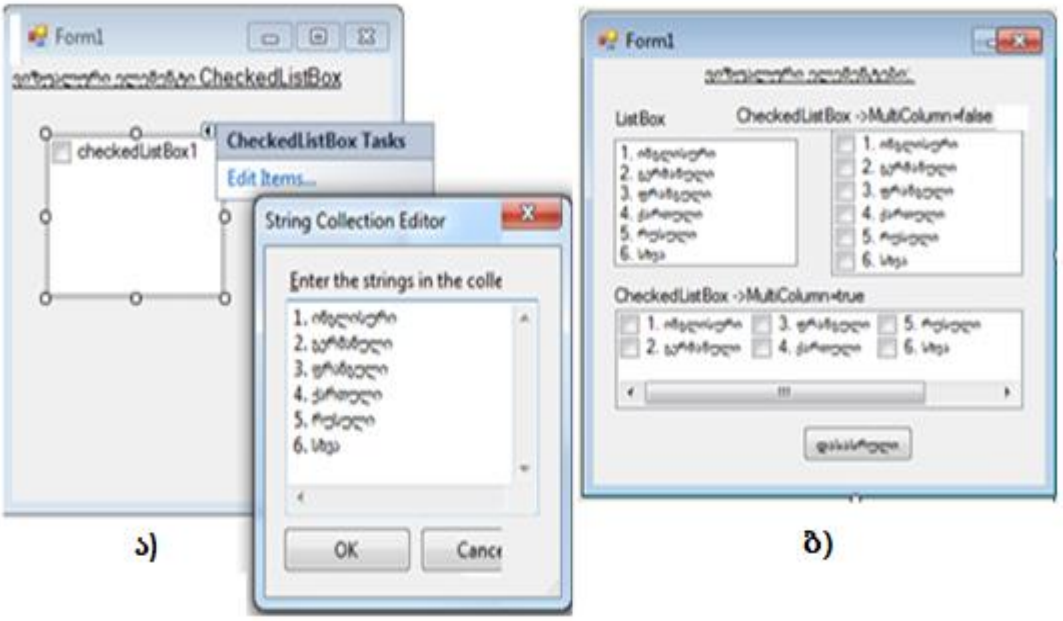
ნახ.2.54. listBox ელემენტი

ListBox-ში ტექსტური სტრიქონების შეტანა ხდება Properties->Items თვისებიდან ან ლისტბოქსის ჩარჩოს ზედა-მარჯვენა კუთხეში ისარზე მაუსის დაწკაპუნებით გამოტანილ Edit Items არჩევით (ნახ.2.55). ორივე შემთხვევაში გამოიტანება String Collection Editor, რომელშიც ჩაიწერება მონაცემები.



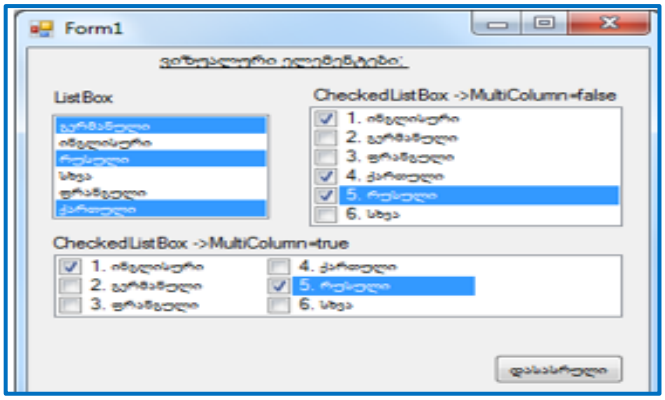
ნახ.2.55. listBox-ის Editor ფანჯარა

CheckListBox ელემენტი მსგავსია ListBox-ისა, ოღონდაც მის სტრიქონს ემატება წინ chekBox-ელემენტი (ნახ.2.56-ა). სტრიქონების შეტანის შემდეგ Edit Items რედაქტორში მიიღება 2.56-ბ ნახაზზე ნაჩვენები სურათი.



ნახ.2.56. CheckListBox-ის ელემენტი Edit Items

ListBox-ის სტრიქონებს მოვაცილოთ ნომრები და მის Properties-ში თვისება დავაყენოთ: Sorted->true. სტრიქონები მოწესრიგდება ანბანის შესაბამისად (ნახ.2.57). 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.

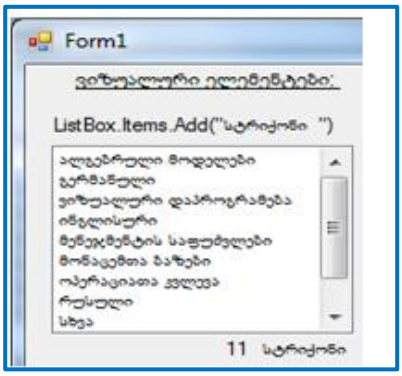


ნახ.2.57.

ახლა განვიხილოთ ტექსტბოქსებთან პროგრამულად მუშაობის ზოგიერთი მეთოდი.

- **Add()** მეთოდი Item თვისებისთვის. პროგრამული კოდის ფრაგმენტი, რომელშიც ხდება სტრიქონების ჩამატება ListBox-ში (ნახაზი 2.58) მოცემულია ქვემოთ ლისტინგში. როგორც ვხედავთ, სტრიქონები ჩაწერილია Form1_Load (object ...) მეთოდში. ფორმის ცარიელ ადგილას მაუსით დაკლიკვით გადავალთ კოდის ამ ფრაგმენტზე.

```
// ლისტინგი – 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();
```



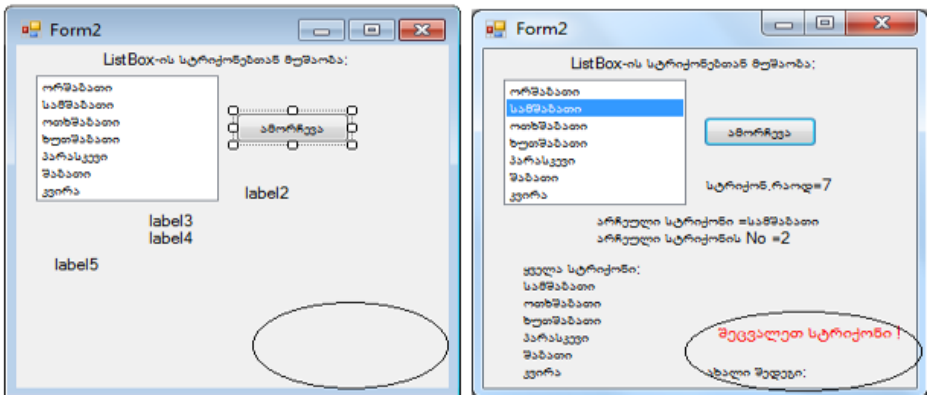
ნახ.2.58. სტრიქონების ჩამატება

- **Items.Count** - მეთოდით განისაზღვრება LisBox-ში სტრიქონების რაოდენობა: `label5`-ში გამოიტანება: 11 სტრიქონი.

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

მომდევნო ლისტინგში მოცემულია ფრაგმენტი ListBox-ის სტრიქონებთან სამუშაოდ. შედეგები „ამორჩევა“ ღილაკის ამოქმედების შემდეგ ასახულია 2.59 ნახაზზე.

```
//ლისტინგი – 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
   }
```

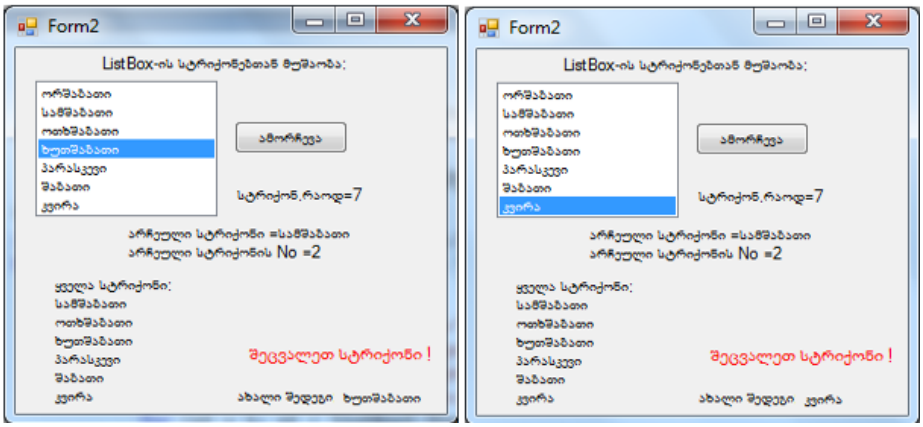


ნახ.2.59. SelectedIndex გამოყენების ფრაგმენტი

ListBox-ში სტრიქონის შეცვლისას ავტომატურად უნდა შეიცვალოს გამოსატანი ახალი შედეგის მნიშვნელობა (რომელიც ოვალის ადგილას label8-ში უნდა ჩაიწეროს). ის გააქტიურდება და გამოჩნდება ფორმაზე „ამორჩევა“ ღილაკის გააქტიურების შემდეგ.

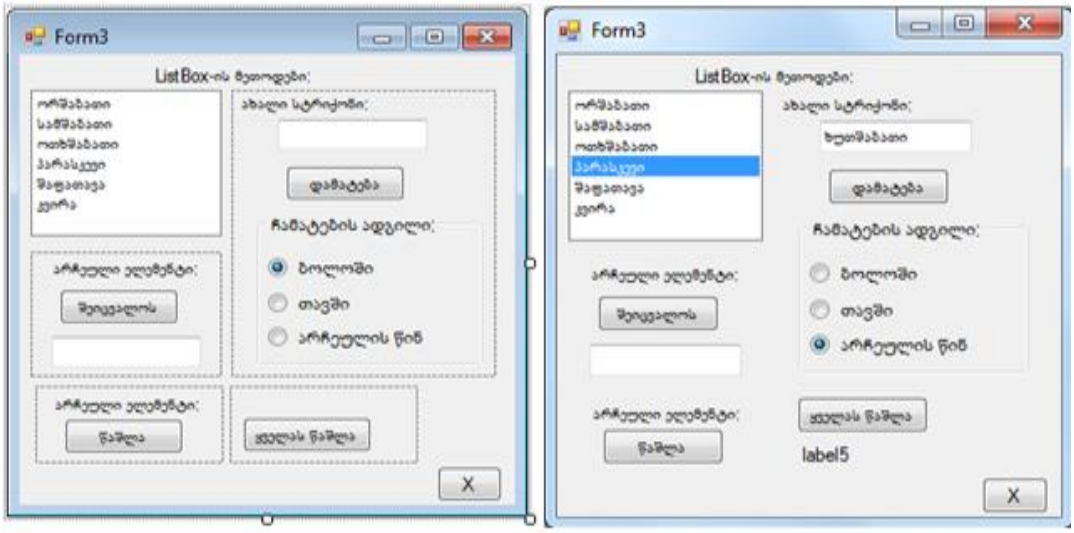
საჭიროა შეიქმნას მოვლენა `listBox1_SelectedIndexChanged`, რომელიც განახორციელებს ავტომატურ ცვლილებას. მისი ლისტინგი მოცემულია ქვემოთ, ხოლო შედეგები 2.60 ნახაზზეა ასახული (ახალი სტრიქონი: ხუთშაბათი, კვირა).

```
//ლისტინგი -- ListBox-ის სტრიქონის შეცვლის მოვლენის კოდი ------
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            label8.Visible = true;
            label8.Text = " "+listBox1.SelectedItem;
        }
```



ნახ.2.60. ავტომატური ცვლილება

ამოცანა_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.2.61).

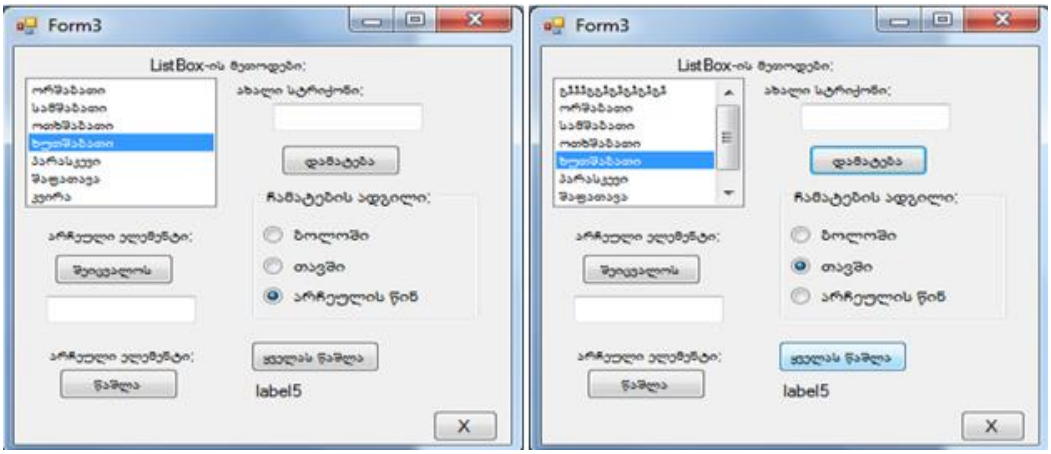


ნახ.2.61

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

```
//ლისტინგი – 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 2.62 ნახაზზე მოცემულ სახეს. თავში დავამატეთ აგრეთვე „გჰგჰგჰგჰგჰგჰგჰ“ სტრიქონი.



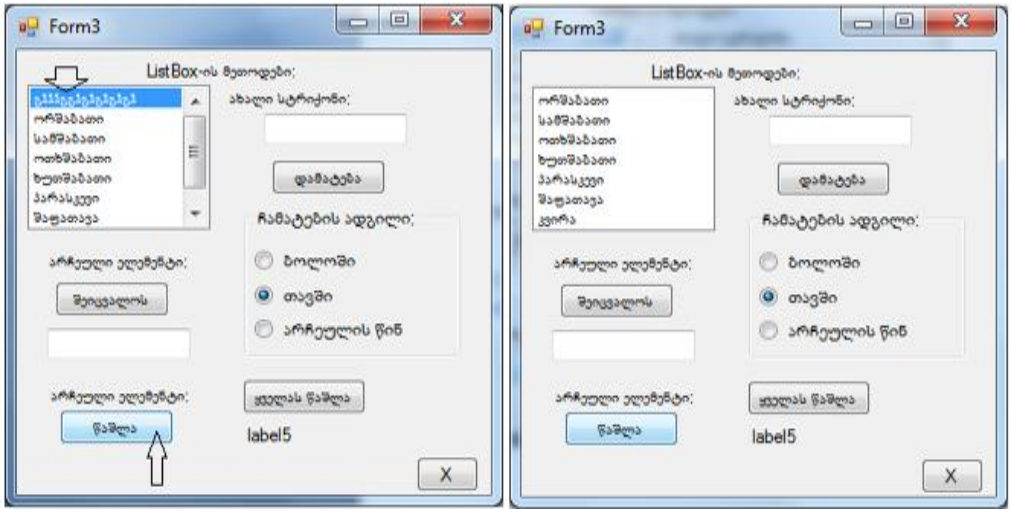
ნახ.2.62

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

```
//ლისტინგი_10.5– ListBox-ის RemoveAt() მეთოდი -------
private void button2_Click(object sender, EventArgs e) // არჩეულის წაშლა
        {
            int st = listBox1.SelectedIndex;
            if (st != -1)
                listBox1.Items.RemoveAt(st);
        }
```

ListBox-ზე ავირჩიოთ პირველი სტრიქონი („გჰგჰგჰგჰგჰ“) და „წაშლა“-ღილაკი 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 2.63 ნახაზზე.



ნახ.2.63

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

```
// ლისტინგი --- ListBox.Items.Clear() მეთოდი ------
private void button3_Click(object sender, EventArgs e)  // ყველას წაშლა
  {
    DialogResult all = MessageBox.Show("ნამდვილად წავშალოთ 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” ღილაკებით და გამაფრთხილებელი შეტყობინებით (ნახ.2.64). განვიხილოთ კოდის სტრიქონი:

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

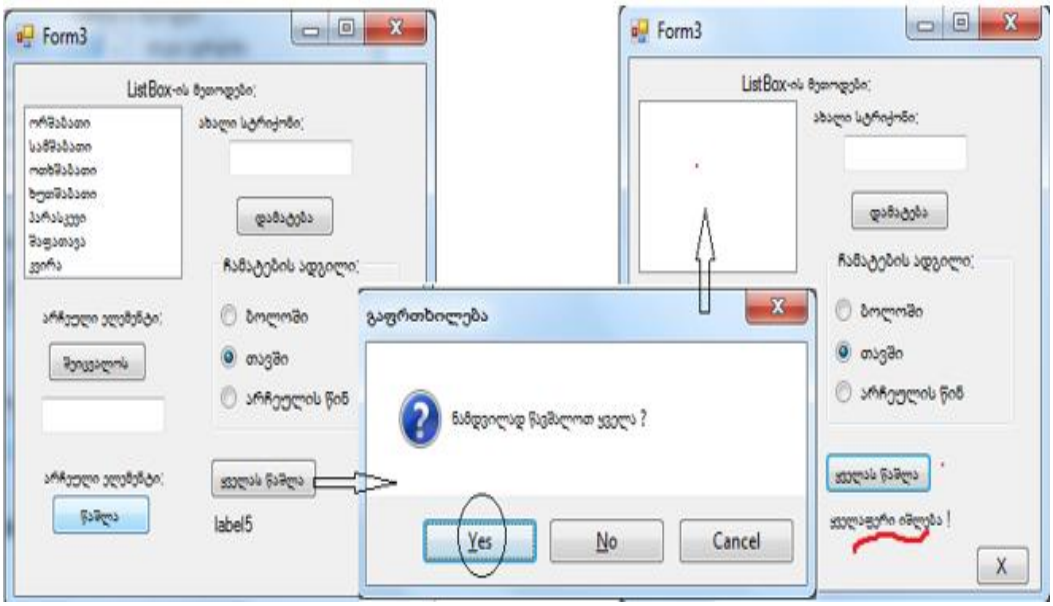
DialogResult არის System.Windows.Forms სახელსივრცის ჩამონათვლის (enum) ტიპი: `public enum DialogResult`, რომელიც განსაზღვრავს დიალოგური ფანჯრიდან დაბრუნებული იდენტიფიკატორის მნიშვნელობას. all-ს მიენჭება ეს მნიშვნელობა, რომელიც შემდგომ if ბლოკში გამოვიყენეთ.

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

`MessageBoxIcon` – enum ტიპის კონსტანტაა, რომელიც ასახავს შეტყობინებას. მაგალითად, `MessageBoxIcon.Question` ესაა ცისფერ წრეში ჩასმული თეთრი ფერის კითხვის ნიშნის სიმბოლო (ნახ.2.64).

ListBox-ის რამდენიმე სტრიქონის მონიშვნისათვის, როგორც ზემოთ აღვნიშნეთ, მის Properties-ის SelectionMode-თვისებაში ვაყენებთ MultiExtended-მნიშვნელობას (ნახ.2.65). ახალი ლისტბოქსის გახსნისას, მასში დაყენებულია ერთ სტრიქონიანი რეჟიმი: SelectionMode=”One” .

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



ნახ. 2.64

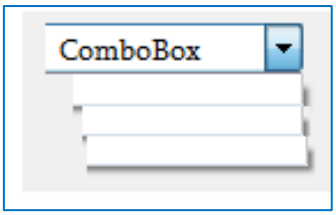
შესასრულებელი დავალება:

ამოცანა_10.2: ავაგოთ 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 ტქსტბოქსის სტრიქონების მთლიანად წასაშლელად (DeleteAll).

2.9. მართვის ვიზუალური ელემენტი ComboBox და თვისება DropDownStyle

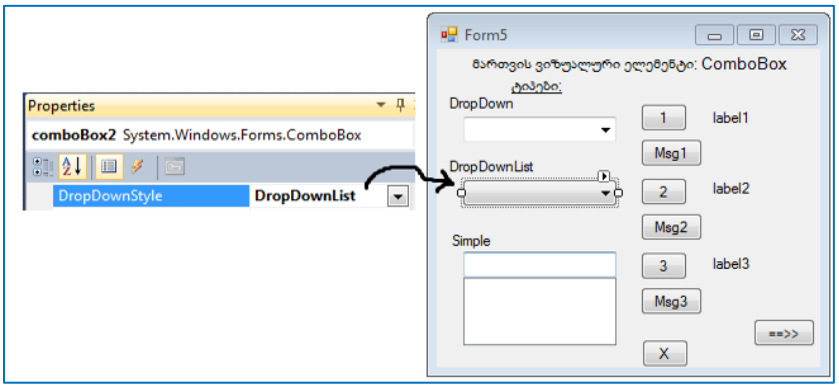
სამუშაოს მიზანი: ComboBox ვიზუალური მართვის ელემენტის ფუნქციონირების შესწავლა.

ComboBox ბევრად ამარტივებს მომხმარებელთა ინტერფეისებს და მოქნილს ხდის მათ. აქ განვიხილავთ მისი საშუალებით პროგრამული პროექტების აგების საილუსტრაციო მაგალითებს. 2.65 ნახაზზე ნაჩვენებია ComboBox-ს ზოგადი საილუსტრაციო მაგალითი.



ნახ.2.65

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). 2.66 ნახაზზე ნაჩვენებია ComboBox-ის სამი ვარიანტი. მისი ტიპი Properties-ში მიეთითება DropDownStyle თვისების ერთ-ერთი მნიშვნელობით.



ნახ.2.66

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

შენიშვნა: ComboBox-ისთვის არ გამოიყენება თვისება SelectionMode.

ამოცანა: ავაგოთ 2.66 ნახაზზე ნაჩვენები ფორმა და ავამოქმედოთ 1,2,3-ღილაკები კომბობოქსის სამი განსხვავებული ტიპის საილუსტრაციოდ. მომდევნო ლისტინგზე მოცემულია პროგრამული კოდი. დამატებითი შეტყობინება გამოვიტანოთ MsgBox-ფანჯარაში შესაბამისი კომბობოქსის არჩეული სტრიქონის მნიშვნელობის (`comboBox.SelectedItem`) და მისი ინდექსის (ნომრის) (`comboBox.SelectedIndex`) საილუსტრაციოდ.

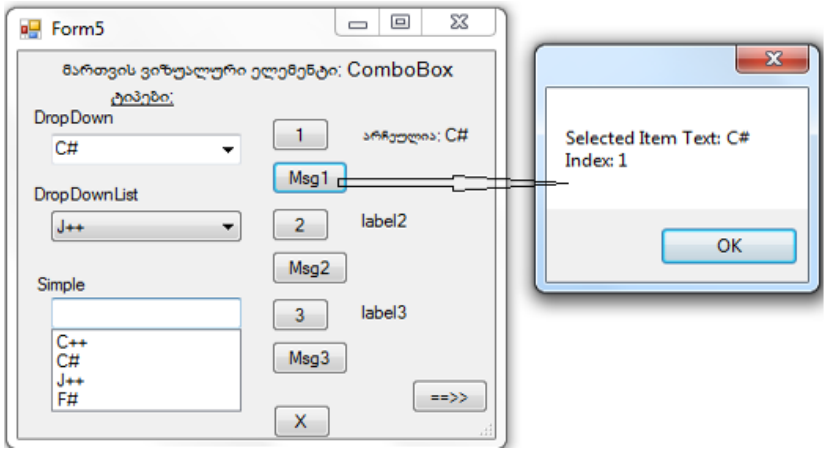
თითოეული ღილაკისთვის ფორმაზე Properties-ში დავაყენოთ DropDownStyle თვისების ერთ-ერთი მნიშვნელობა: 1- DropDown, 2- DropDownStyleList ან 3- Simple.

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 – 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 2.67 ნახაზზე.

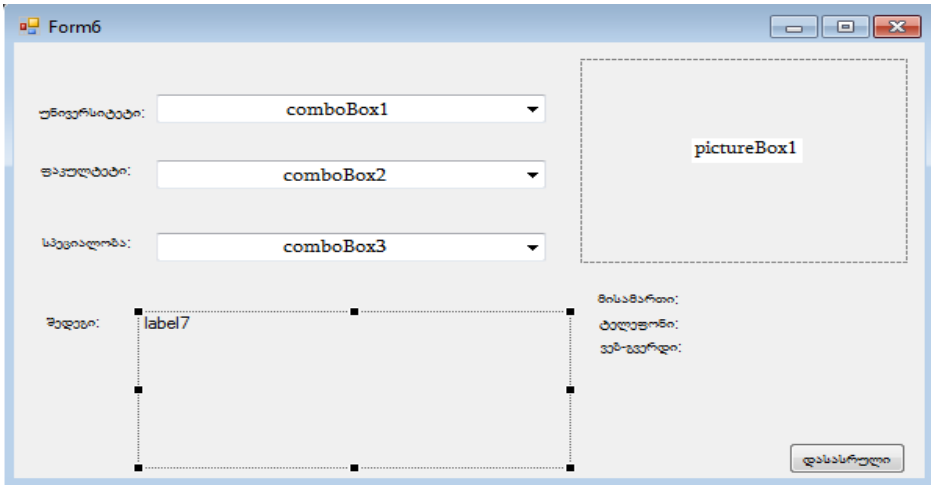


ნახ.2.67

გავლილი მასალის საფუძველზე ავაგოთ ახალი პროგრამული პროექტი, რომელშიც გამოყენებულ იქნება ComboBox კლასის თისებები და მეთოდები.

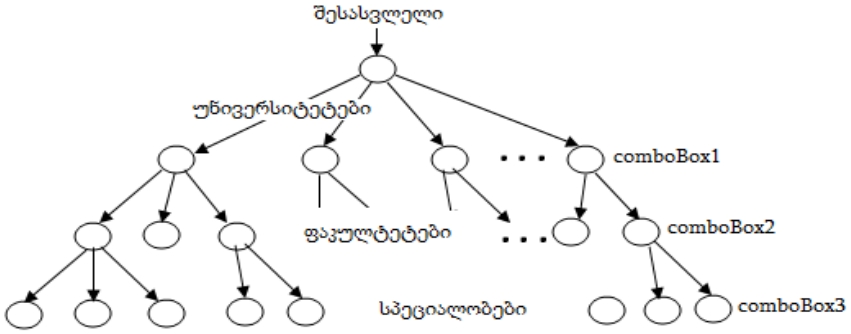
ამოცანა: შევქმნათ პროგრამული აპლიკაცია „უნივერსიტეტები“, რომელიც იძლევა ინფორმაციას მათი ფაკულტეტებისა და სპეციალობების შესახებ. მომხმარებელი ირჩევს სამდონიანი ComboBox-ების სისტემაში თავის სასურველ მონაცემებს (ნახ.2.68):

უნივერსიტეტი -> ფაკულტეტი -> სპეციალობა



ნახ.2.68

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 2.69 ნახაზზე.



ნახ.2.69

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. მომდევნო ლისტინგზე ნაჩვენებია ამ პროგრამის რეალიზაციის კოდის ფრაგმენტი.

```
// ლისტინგი -- 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");
```

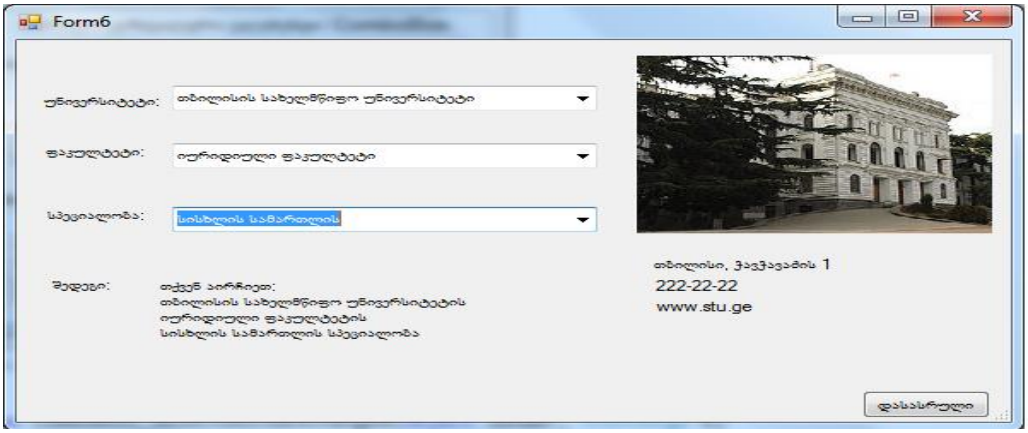
```
            pictureBox1.Image
               =Image.FromFile("C:\\C#2010\\4_5\\WinListCombo\\tsu.jpg");
            label4.Text = "თბილისი, ჭავჭავაძის 1";
            label5.Text="222-22-22";
            label10.Text="www.stu.ge";
            fa = 100;
            break;
        case 1:
            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("საქართველოს ისტორიის");
                      comboBox3.Items.Add("გეოგრაფიის");
                    comboBox3.Items.Add("ფილოლოგიისა 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");
```

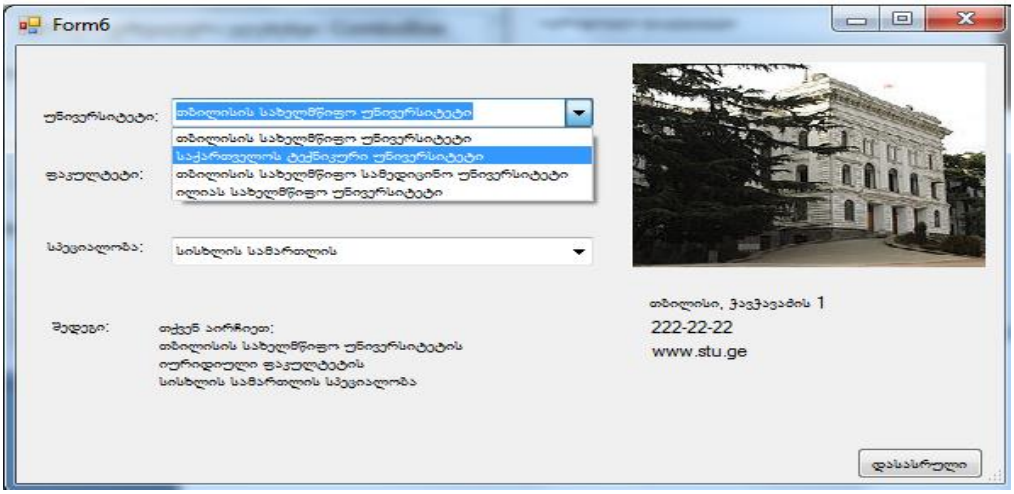
```
                    comboBox3.Items.Add("რკინაბეტონის კონსტრუქციების");
                        break;
                    case 1:
                        comboBox3.Items.Clear();
                        comboBox3.Items.Add("ჰიდროენერგეტიკის");
                        comboBox3.Items.Add("თბოენერგეტიკის");
                        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("ყბა-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 + "ს \n" +
                   itemF + "ს \n" +
                   itemS +" სპეციალობა";

        }
    }
}
```

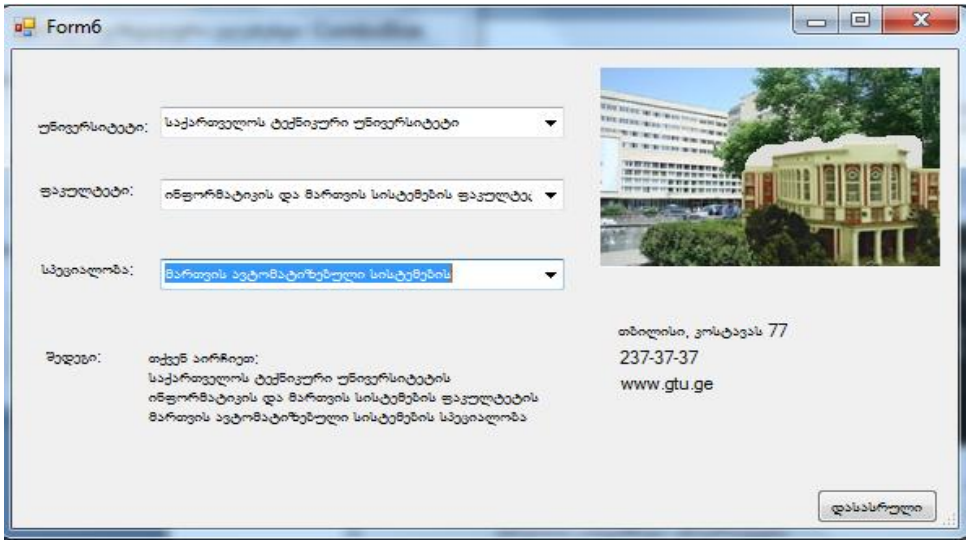
2.70-2.74 ნახაზებზე ილუსტრირებულია ამ აპლიკაციის მუშაობის ფრაგმენტები სხვადასხვა მოთხოვნების შესრულებისას.



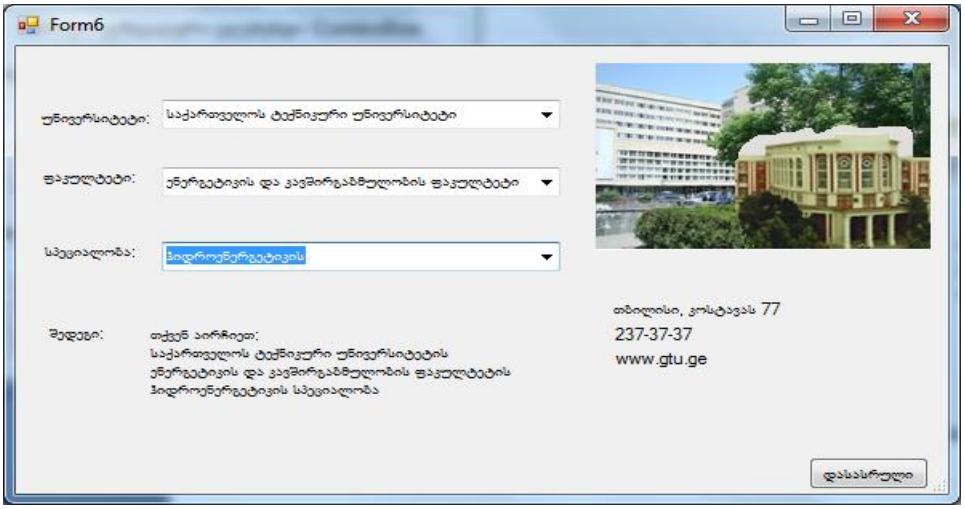
ნახ.2.70



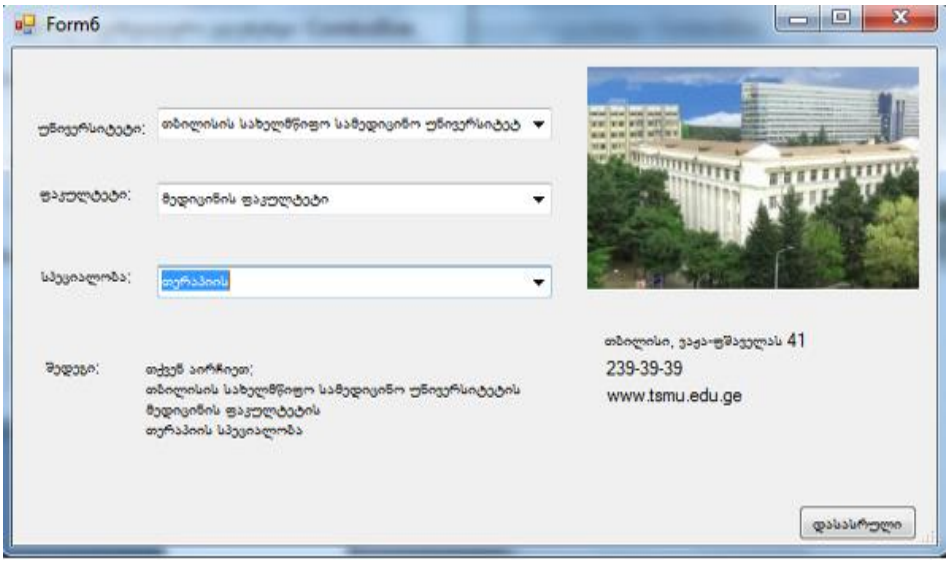
ნახ.2.71



ნახ.2.72



ნახ.2.73



ნახ.2.74

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.

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.

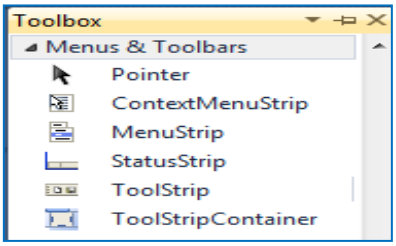
III თავი

მენიუს დაპროგრამების ვიზუალური კომპონენტები

3.1. მთავარი მენიუების აგების ვიზუალური საშუალებები

სამუშაოს მიზანი: კომპიუტერული სისტემის მთავარი მენიუს აგების ვიზუალური მართვის ელემენტების შესწავლა.

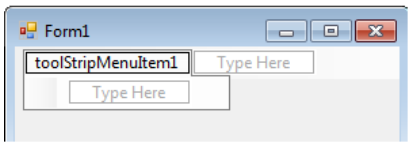
C# ენის ვიზუალური ელემენტები, რომლებიც ინსტრუმენტების პანელზეა განთავსებული, ნაჩვენებია 3.1 ნახაზზე.



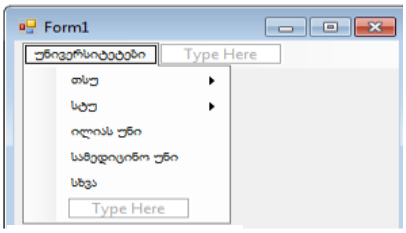
ნახ.3.1. Menus & Toolbars ელემენტები

➢ მთავარი მენიუს ვიზუალური დაპროგრამება

ამოცანა_1: Form1-ზე ავაგოთ მთავარი მენიუ „უნივერსიტეტები“ და ქვემენიუს პუნქტებით „ფაკულტეტები“ და „კათედრები“. ფორმაზე მთავარი მენიუს შესაქმნელად ToolBox-იდან გადმოვიტანოთ MenuStrip ელემენტი. ფორმაზე გაჩნდება 3.2-ა ნახაზზე ნაჩვენები გამოსახულება. შევიტანთ მენიუს პუნქტებს (3.2-ბ).

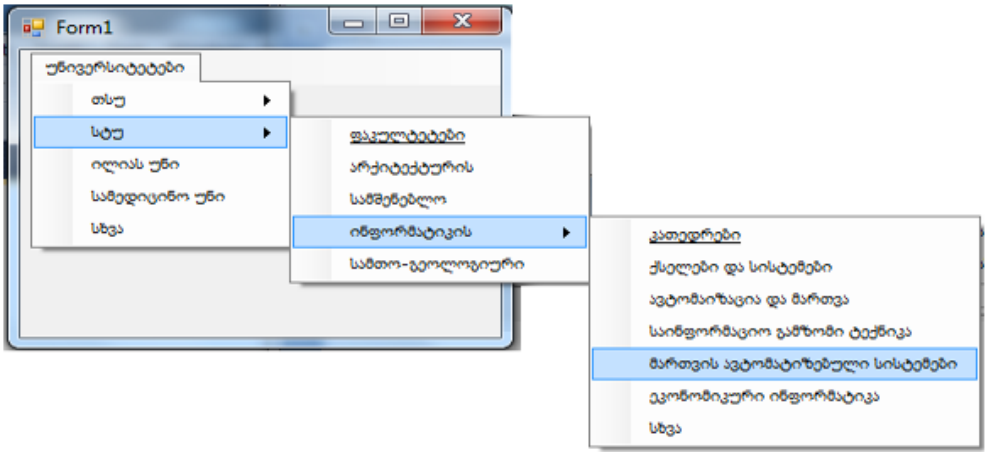


ნახ.3.2-ა



ნახ.3.2-ბ

ეს ხორციელდება ვერტიკალურად და/ან ჰორიზონტალურად. თითოეული სტრიქონისთვის შეიძლება ქვემენიუს შექმნა (ნახ.3.3).

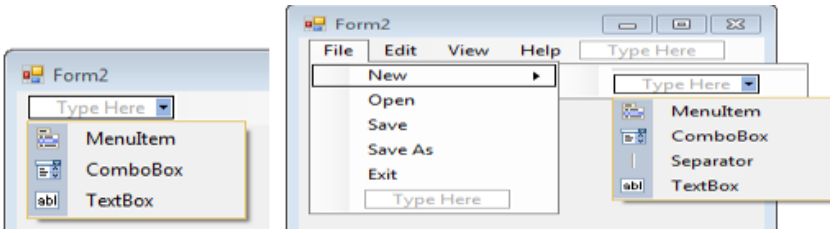


ნახ.3.3. სამდონიანი მენიუ

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

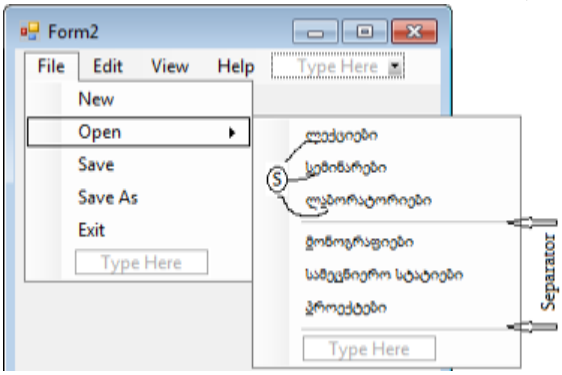
მთავარი მენიუს პუნქტები შიძლება სამი სახისა იყოს (ნახ.3.4):

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



ნახ.3.4

ქვემენიუსთვის დასაშვებია Separator პუნქტიც. რომლის დანიშნულებაა მენიუს პუნქტების ერთმანეთისაგან გამოყოფა ხაზით, რაც ვიზუალურ კომფორტს უქმნის მომხმარებელს (ნახ.3.5).

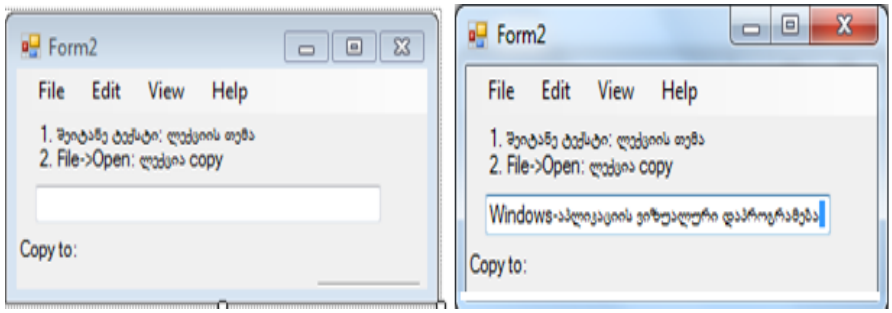


ნახ.3.5. Separator-ის გამოყენება

მენიუში ხშირად იყენებენ სტრიქონის ერთი ასოს გამოყოფას (ქვეშგახაზვა), 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 მოვლენებითან და მეთოდებთან. მოვლენის მთავარი სახეა Click, რომელზეც მიბმულია შესასრულებელი პროცედურის კოდი. განვიხილოთ ეს საკითხი.

ამოცანა_2 : ფორმაზე (ნახ.3.6) ავაგოთ მთავარი მენიუ, რომლის პუნქტი „ლექციები“ გადააკოპირებს textBox1-ში ჩაწერილ ლექციის თემის დასახელებას label1-ში. მენიუს Exit პუნქტის არჩევისას კი პროგრამა დაასრულებს მუშაობას.

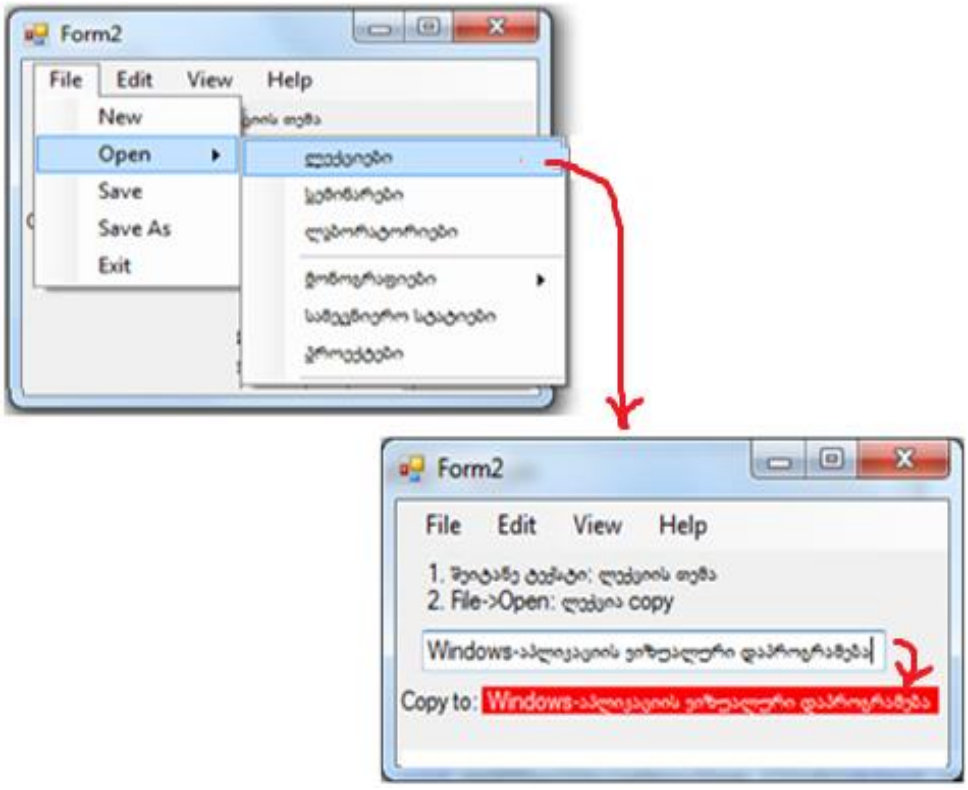


ნახ.3.6. ფორმის მაკეტი

`textBox1`-ში ჩაწერილი სტრიქონი „Windows-აპლიკაციის ვიზუალური დაპროგრამება“ მთავარი მენიუს „File->Open->ლექციები“ პუნქტის არჩევით ამ სტრიქონს გადააკოპირებს `label9` ველში, რომელ;იც „`Copy to`“-ლებელის მარჯვნივაა მოთავსებული (ის არ ჩანს). ამ მოვლენის დამმუშავებელს აქვს ლისტინგში მოცემული კოდის ფორმა:

```
// ლისტინგი ---- 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 3.7 ნახაზზე, წითელი ფონით და თეთრი ტექსტით.

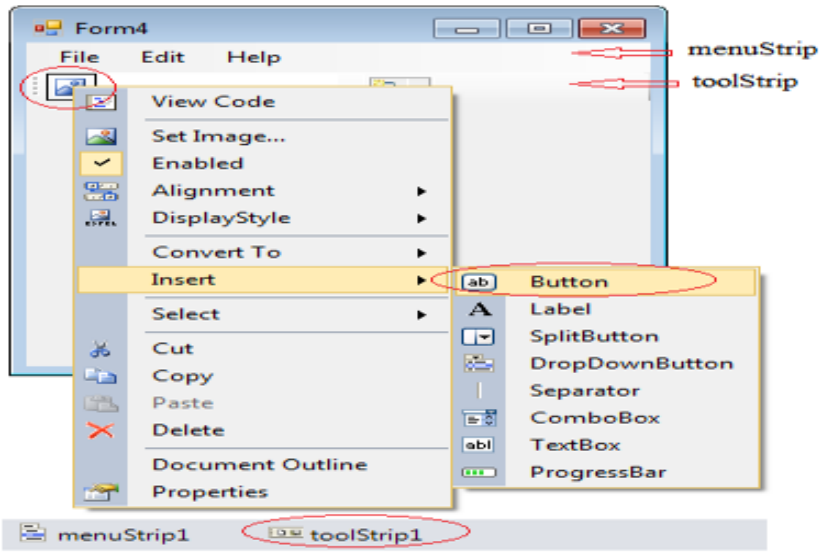


ნახ.3.7. Copy to -ს label-ში ჩაიწერა სტრიქონი textBox1-დან

3.2. გრაფიკული მენიუს აგების ვიზუალური ელემენტი - ToolStrip

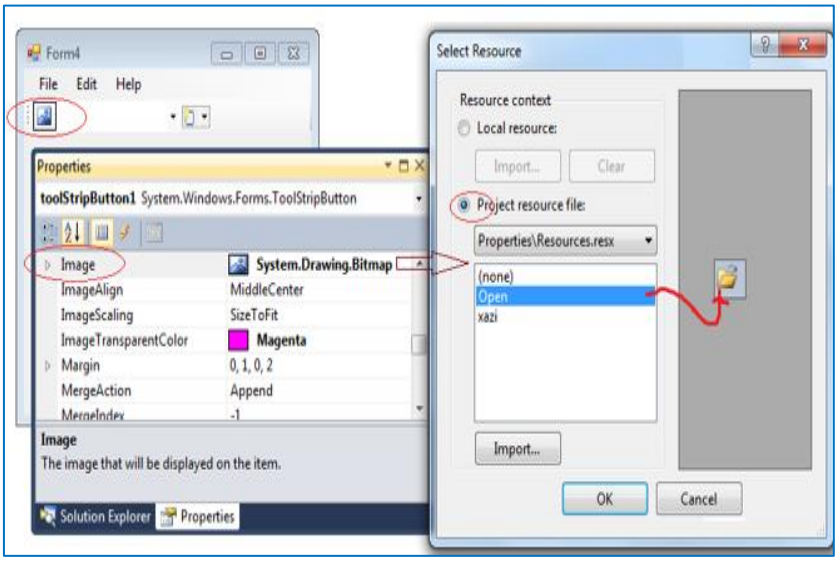
სამუშაოს მიზანი: გრაფიკული მენიუს და დიალოგური პროცესების აგების შესწავლა ვიზუალური მართვის ელემენტების გამოყენებით.

კომპიუტერული სისტემების აგების დროს ინტერფეისში მთავარი მენიუს გარდა ხშირად იყენებენ გრაფიკულ პიქტოგრამების (icons), რაც უფრო ეფექტურს და მოქნილს ხდის მის გამოყენებას. C# ენაში ასეთი ვიზუალური ელემენტია Menus&Toolbars პანელის toolStrip ელემენტი. მისი გადატანით ფორმაზე მივიღებთ 3.8 ნახაზზე ნაჩვენებ სურათს. საჭიროა ავირჩიოთ სახე: button, Label, ComboBox და ა.შ.



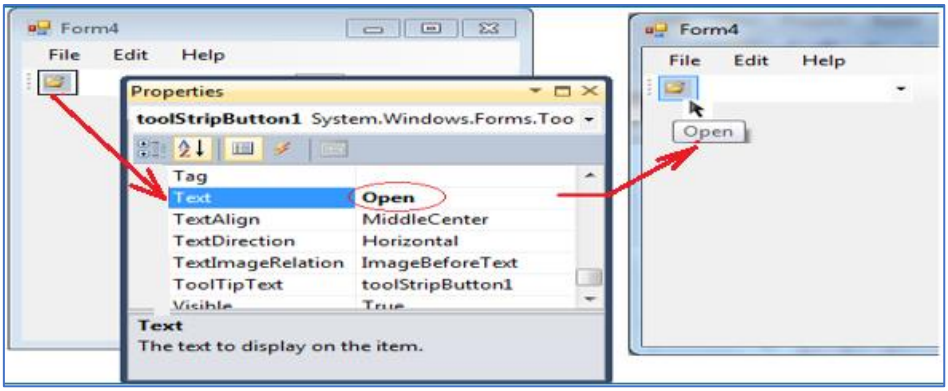
ნახ.3.8

აქ შესაძლებელია მაუსის მარჯვენა ღილაკით სტანდარტულად მიღებული პიქტოგრამის შეცვლა, ჯერ Properties-ში Image თვისების არჩევით და შემდეგ საჭირო პიქტოგრამის Import-ირებით დიალოგური ფანჯრიდან (ნახ.3.9).



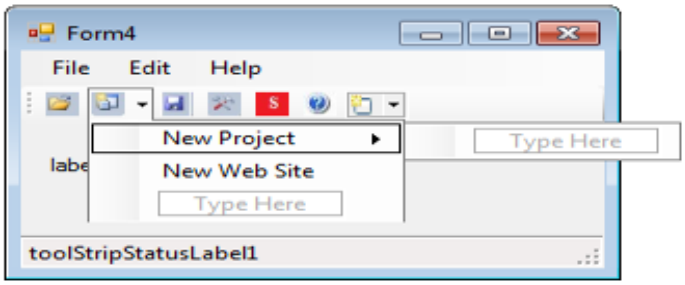
ნახ.3.9

ახალ პიქტოგრამას თვისებაში Text ჩაუწერეთ მისი ფუნქციის შესაბამისი სიტყვა, მაგალითად, Open. 3.10 ნახაზზე ჩანს მიღებული შედეგი.



ნახ.3.10

საბოლოო შედეგები რამდენიმე ახალი პიქტოგრამის ჩასმის შემდეგ, რომელთაგანაც ერთ-ერთი comboBox ტიპისაა, მოცემულია 3.11 ნახაზზე.

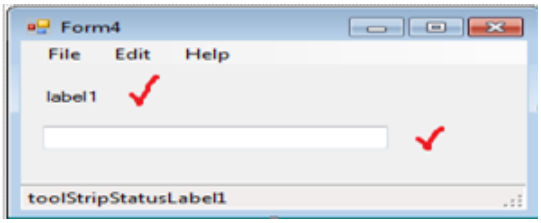


ნახ.3.11

3.3. კონტექსტური მენიუს ვიზუალური აგების ელემენტი ContextMenuStrip

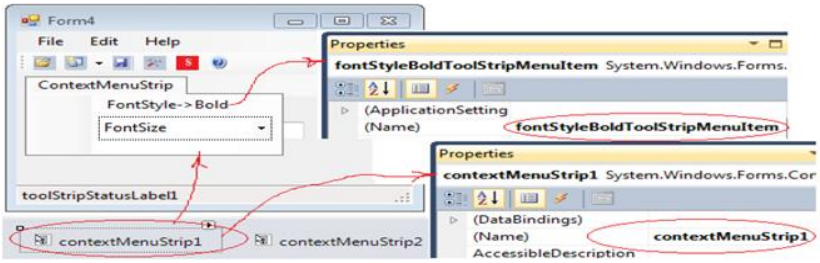
სამუშაოს მიზანი: ToolBox-პანელის ContextMenuStrip ელემენტის გამოყენება. კონტექსტური მენიუს სტრიქონთა მნიშვნელობების შესაბამისობაში მოყვანა ფორმის ან ფორმაზე დადებული ელემენტებისთვის საჭირო ფუნქციებთან.

ამოცანა_3: ავაგოთ ფორმა, მაგალითად, 3.12 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.

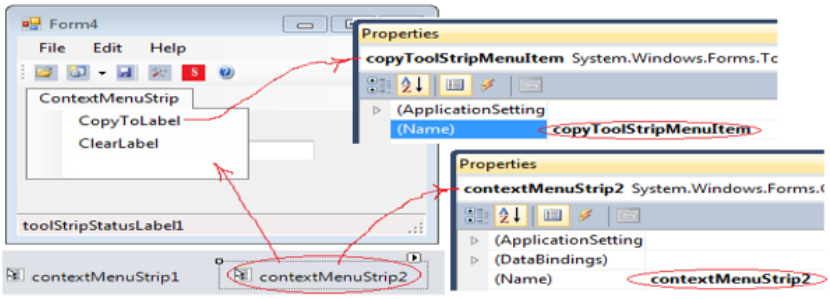


ნახ.3.12

3.13-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.

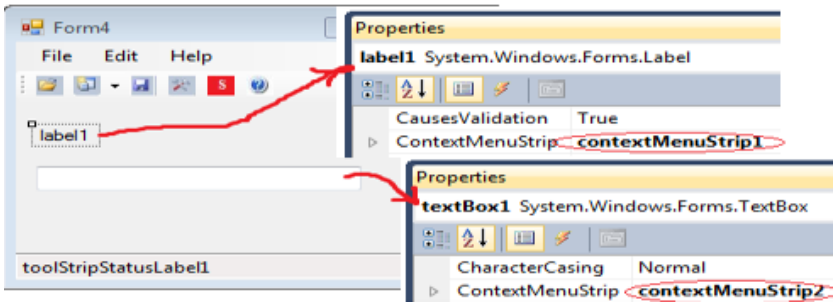


ნახ.3.13-ა



ნახ.3.13-ბ

ამისათვის მოვნიშნოთ label1 და მის Properties-ის ContextMenuStrip-თვისებაში ჩავწეროთ contextMenuStrip1 მნიშვნელობა (ნახ.3.14). ასევე textBox1-სთვის ჩავწეროთ contextMenuStrip2.



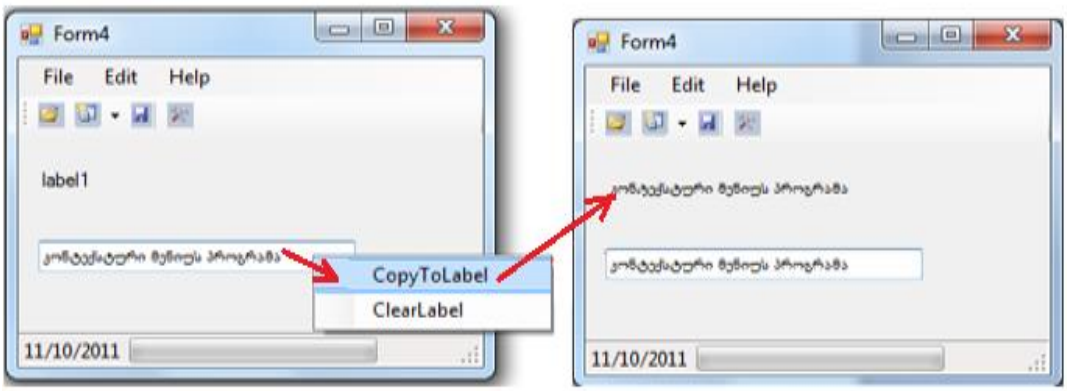
ნახ. 3.14

პროგრამული კოდი მოცემულია მომდევნო ლისტინგში, შედეგები კი 3.15-3.17 ნახაზებზე.

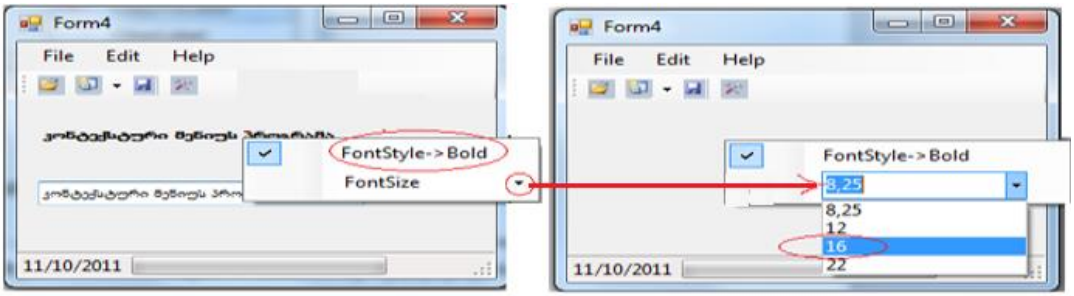
```
// ლისტინგი -- შრიფტის შეცვლა ---
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;

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;
        }
    }
}
```

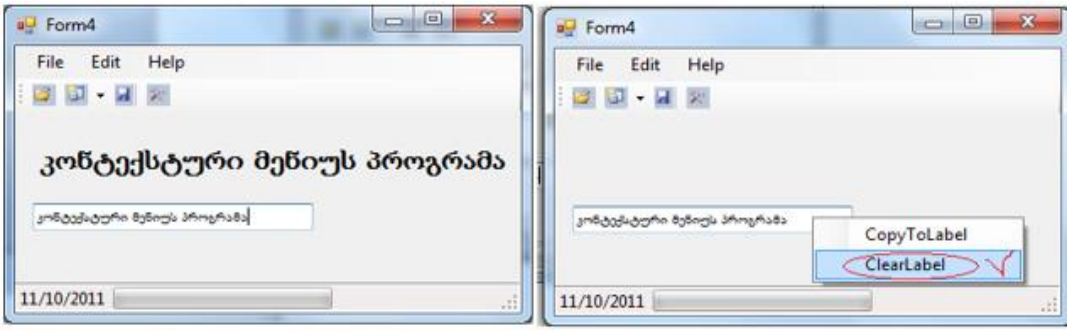
შედეგები:



ნახ.3.15. შედეგები



ნახ.3.16



ნახ.3.17

დავალება:

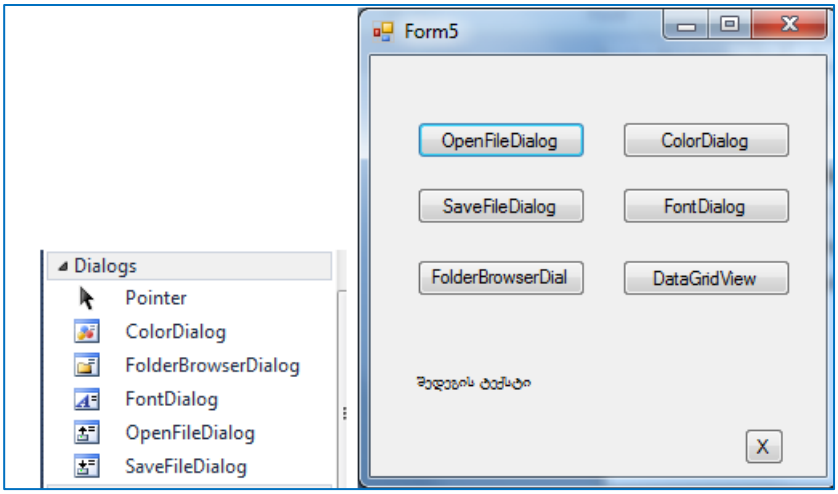
ამოცანა_3: Form2-ზე მთავარი მენიუს Edit პუნქტში ჩავდოთ კომბობოქსი შრიფტების არჩევის მიზნით, კერძოდ AcadNusx და AcadMtavr მნიშვნელობებით. ამ ფონტების არჩევით უნდა შეიცვალოს label9-ში ჩაწერილი სტრიქონის შრიფტი.

3.4. C# ენის ვიზუალური სტანდარტული დიალოგური საშუალებები

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.3.18).

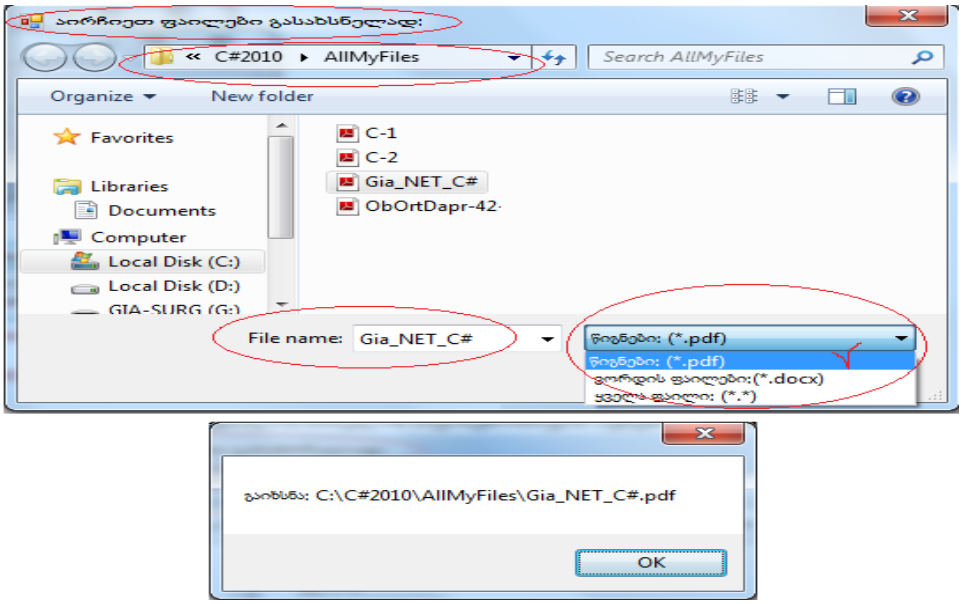


ნახ.3.18

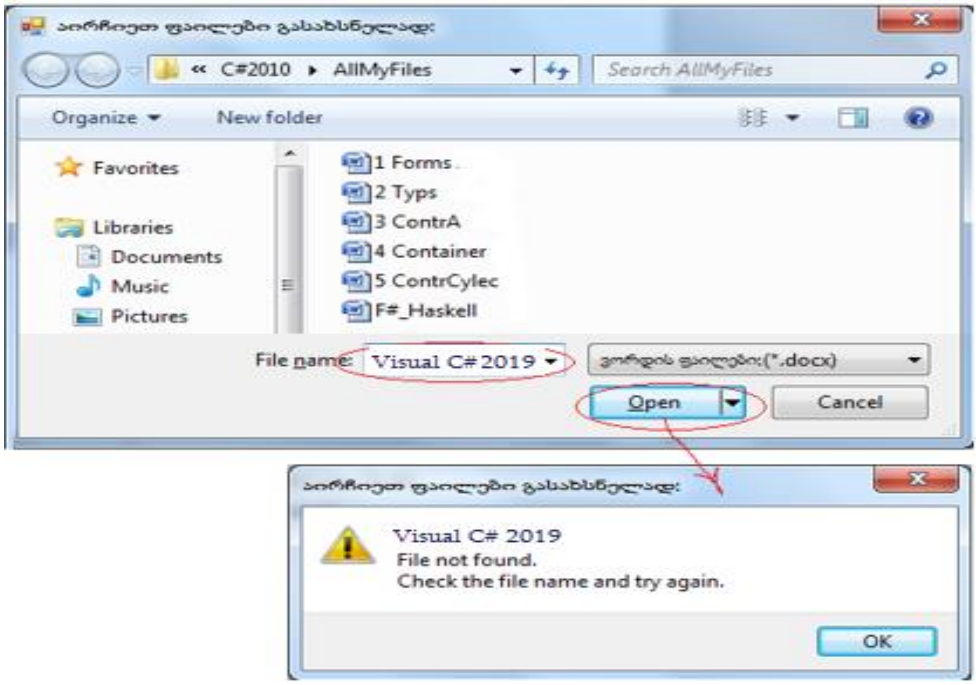
3.4.1. OpenFileDialog

- **OpenFileDialog** კლასის ობიექტის დანიშნულებაა გასახსნელი ფაილის ამორჩევა, მითითებული ფოლდერის (InitialDirectory), ფილტრის (Filter - მაგალითად, ფაილის ტიპით და დიალოგური ველის სათაურის (Title) მიხედვით (ნახ.3.19 და 3.20).

დიალოგის შედეგი ფაილის სახელის თვისებისთვის იქნება - FileName.



ნახ.3.19. დადებითი შედეგით



ნახ.3.20. უშედეგოდ დასრულება

```
// ლისტინგი --- OpenFileDialog -------
     private void button1_Click(object sender, EventArgs e) //
fileOpenDialog
        {
            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("უშედეგო დასასრული !");
        }
```

3.4.2. SaveFileDialog

- **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 --- 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 !");
        }
```

3.4.3. FolderBrowserDialog

- **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.

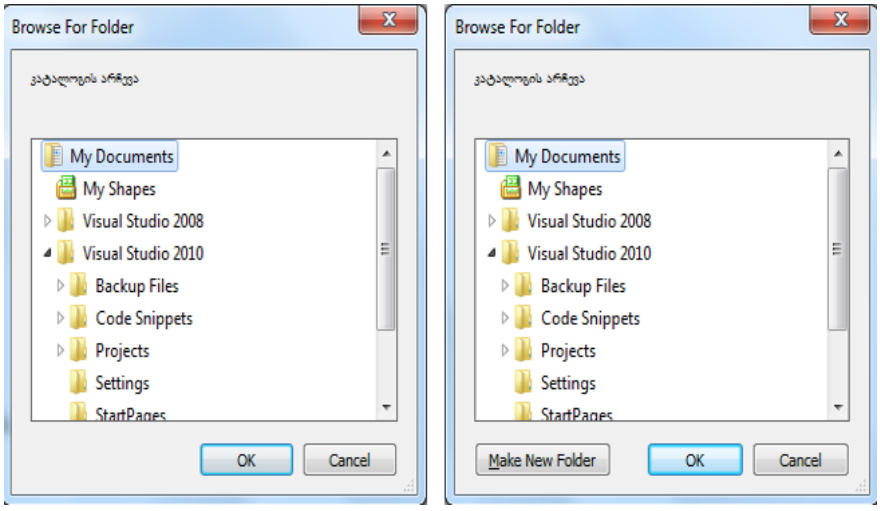
```
// ლისტინგი --- FolderBrowserDialog -------
private void button3_Click(object sender, EventArgs e)
    {
        FolderBrowserDialog fb = new FolderBrowserDialog();
        fb.RootFolder = Environment.SpecialFolder.MyDocuments;
        fb.ShowNewFolderButton = false;
        fb.Description = "კატალოგის არჩევა";

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

სტრიქონით:

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

ფესვურ 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.3.21).

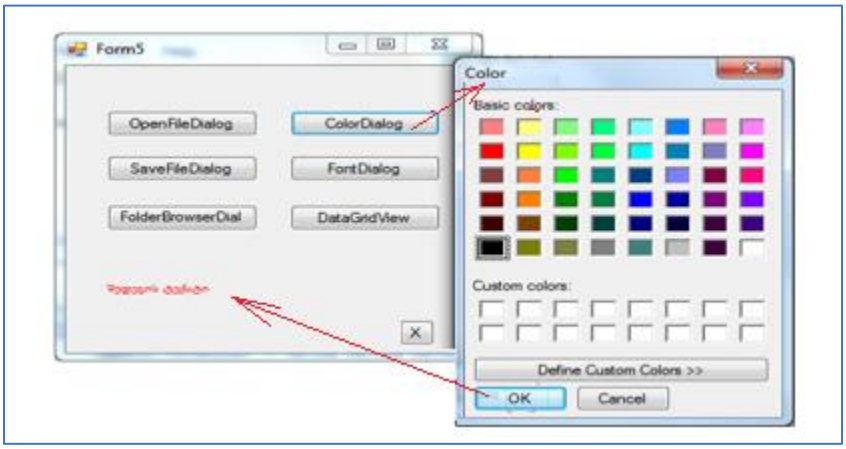


ნახ.15.4

3.4.4. ColorDialog

- **ColorDialog** კლასის ობიექტის დანიშნულებაა ფერის არჩევა ფორმაზე მოთავსებული მონიშნული ელემენტისთვის (იხ. ლისტინგი და ნახ.3.22).

```
// ლისტინგი --- 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");
   }
```



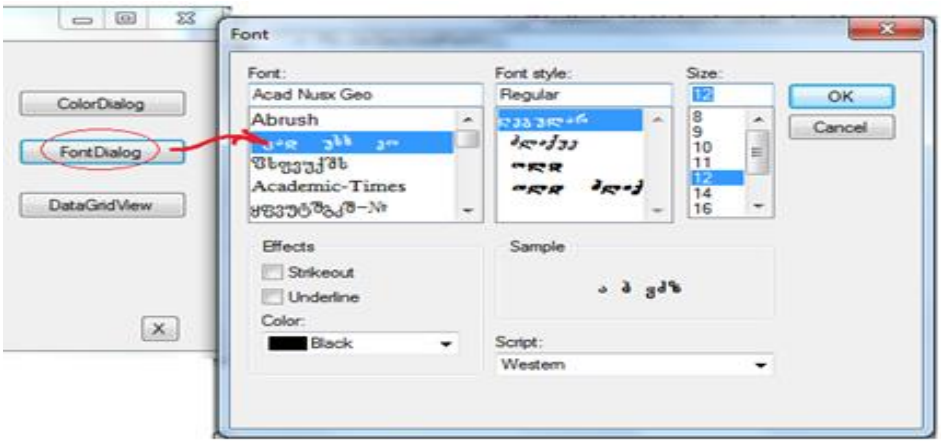
ნახ.3.22

3.4.5. FontDialog

- **FontDialog** კლასის ობიექტის დანიშნულებაა შრიფტის (ფონტის) არჩევა ფორმაზე მოთავსებული მონიშნული ელემენტისთვის (იხ. ლისტინგი და ნახ.3.23).

```
// ლისტინგი --- FontDalog  -------
 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)
         {
```

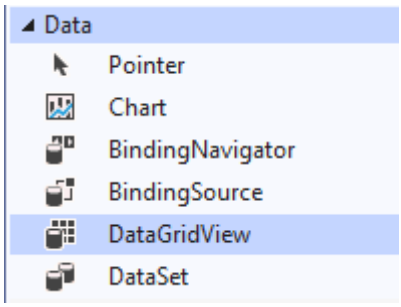
```
            label1.Font = fd.Font;
            label1.ForeColor = fd.Color;
        }
    else
        MessageBox.Show("უშედეგო დასასრული");
}
```



ნახ.3.23

C# ენის მონაცემთა ბაზასთან მუშაობის ვიზუალური საშუალებანი

Visual Studio.NET Framework 8.0 პლატფორმაზე აგებული აპლიკაციებისთვის, რომლებიც C# ენის საფუძველზეა დაწერილი, აქვს მონაცემებთან წვდომის სტანდარტული ვიზუალური ელემენტები (ნახ.4.1).

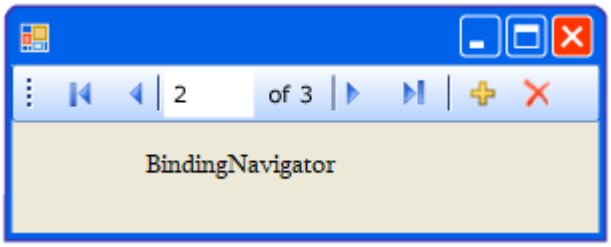


ნახ.4.1. Toolbox Data კომპონენტები

პროგრამის დასაკავშირებლად მონაცემთა ბაზებთან (SQL Server, Oracle ან სხვ.) გამოიყენება ADO.NET დრაივერი (ActiveX Data Object). ჩვენ 1.7.3 პარაგრაფში განვიხილეთ .NET Entity Framework. განსხვავება მათ შორის ისაა, რომ Entity Framework პლატფორმა არის მონაცემებთან მიმართვის ობიექტ-ორიენტირებული ტექნოლოგია, ORM (Object-Relational Mapping) მოდელი და მონაცემთა ბაზასთან წვდომისთვის იყენებს LINQ-ს (SQL-ენის მსგავსი). კოდი გენერირედება ავტომატურად. ADO.NET კი უფრო დიდი და სწრაფი ინსტრუმენტია, რომელიც გამოიყენება როგორც Web-სერვისების ასაგებად (ADO.NET Data Services), ასევე მაიკროსოფთის WPF და WCF ტექნოლოგიებში [5,6,14,15].

განვიხილოთ მოკლედ Toolbox-ის Data ელემენტები [39-41]:

- BindingNavigator - მონაცემთა მოძებნის და ცვლილების სტანდარტული საშუალებების შექმნა Windows Forms-ზე (ნახ.4.2).



ნახ.4.2.

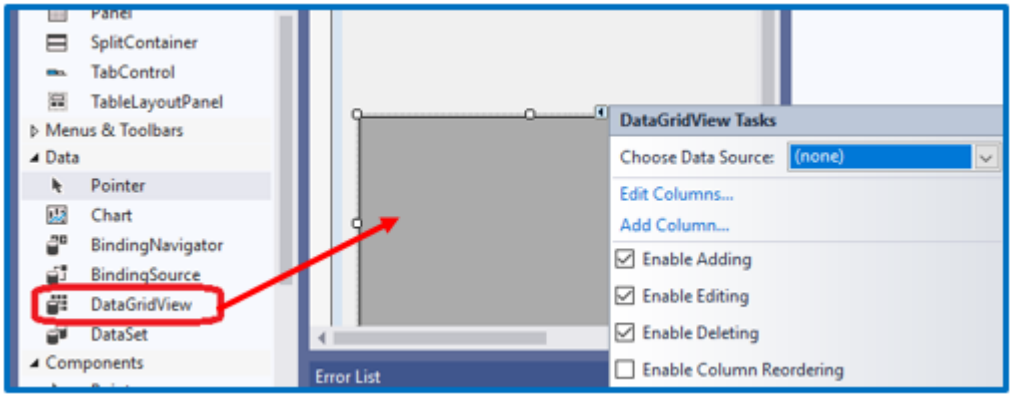
- BindingSource - მონაცემებს წყაროდან მიამაგრებს მართვის ელემენტს ფორმაზე. შემდგომში ყველა ოპერაცია, როგორიცაა გადაადგილება, მოწესრიგება, ფილტრაცია და განახელება, ხორციელდება ამ BindingSource კომპონენტის გამოძახებით;
- DataGridView - ცხრილური მონაცემების ასახვა და რედაქტირება;
- DataSet - არის ADO.NET-ის ძირითადი ობიექტი. იგი იღებს მონაცემებს ბაზის ცხრილებიდან და ათავსებს კეს- მეხსიერებაში შემდგომი დამუსავების მიზნით.

4.1. ცხრილების წარმოდგენის მართვის ელემენტი DataGridView

სამუშაოს მიზანი: ცხრილებთან (Tables) მუშაობის მართვის ელემენტების შესწავლა DataGridView საფუძველზე.

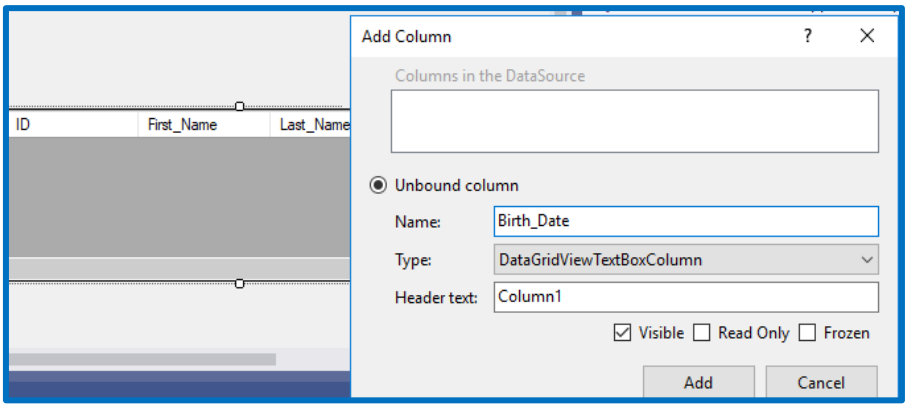
მარტივი სიებისა და მონაცემთა ერთგანზომილებიანი ველების ასახვის მიზნით გამოიყენება ListBox- და ComboBox-ები. ცხრილები (Tables), რომლებიც სტრიქონებისა და სვეტებისგან შდგება, საუკეთესო საშუალებაა მონაცემთა ორგანზომილებიანი ველების, მასივების წარმოსადგენად. C# ენაში ასეთი ობიექტების ასახვის მიზნით გამოიყენება მართვის ელემენტი, ტიპით DataGridView. ამ ელემენტს, პრაგმატული თვალსაზრისით, დიდი გამოყენება აქვს მონაცემთა ბაზების სისტემებში (ADO.NET), რასაც ჩვენ მომდევნო ლაბორატორიებში განვიხილავთ.

4.3 ნახაზზე მოცემულია Windows Forms გააქტიურების შემდეგ ToolBox-დან Form1-ზე გადმოტანილი DataGridView ელემენტი და საწყისი სიტუაცია.



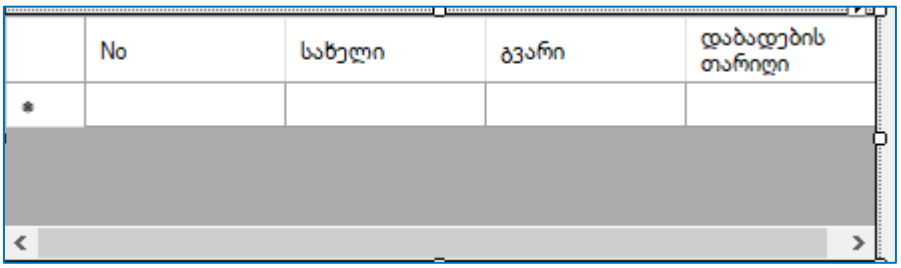
ნახ.4.3

დიალოგურ რეჟიმში ცხრილის ველების (სვეტების) შესატანად ვირჩევთ Add Columns და გადავდივართ 4.4 ნახაზზე მოცემულ ფანჯარაში. შვიტანოთ მიმდევრობით „სტუდენტები“-ს ატრიბუტები, მაგალითად, No, First_Name (სახელი), Last_Name (გვარი), Birth_data (დაბადების თარიღი) და ა.შ.



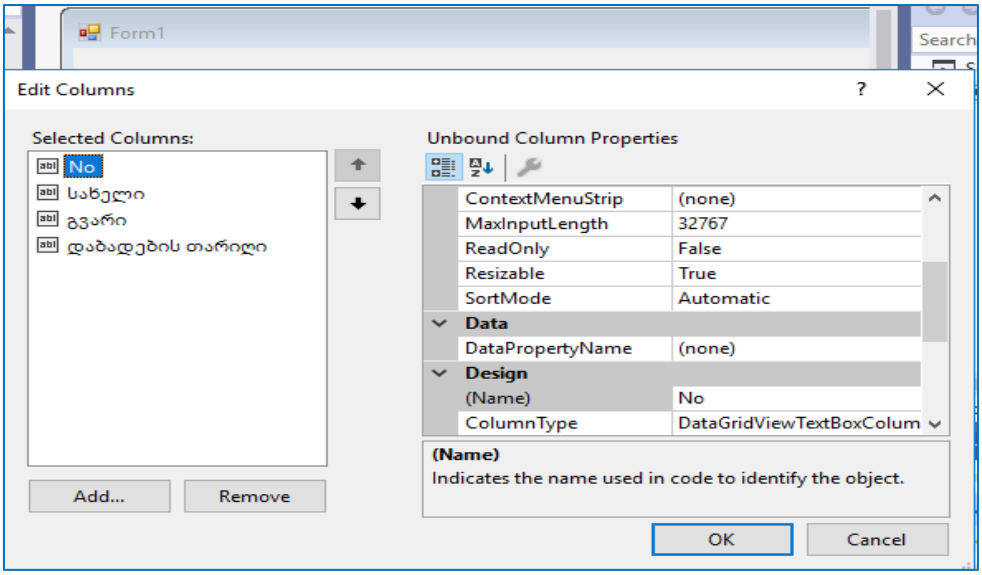
ნახ.4.4

პროგრამის ამუშავების შემდეგ მივიღებთ 4.5 ნახაზზე მოცემულ ცხრილს.

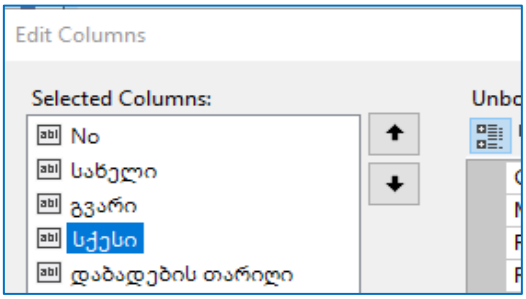


ნახ.4.5

შეტანილი ველების კორექტირებისათვის ვიყენებთ Edit Columns პუნქტს. ჩავამატოთ ველი Sex (სქესი) ან შვასწოროთ უკვე ჩაწერილი დასახელებები. მაგალითად, 4.5 ნახაზზე, Edit ფანჯარაში გვინდა ველი sqesi შევცვალოთ ქართული შრიფტით და ამასთანავე იგი გადავიტანოთ ველი „სახელი“-ს შემდეგ. 4.6. და 4.7 ნახაზებზე ნაჩვენებია ეს შემთხვევები.

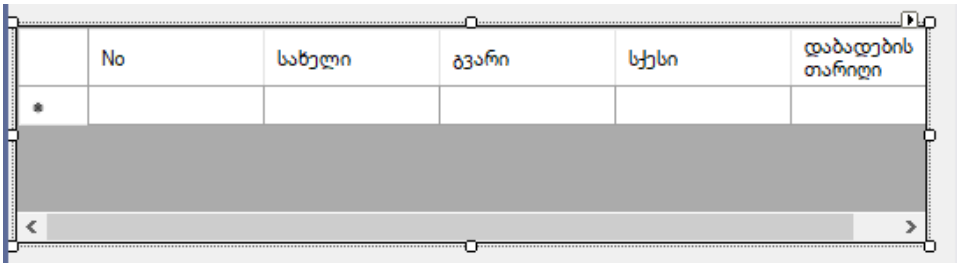


ნახ.4.6.



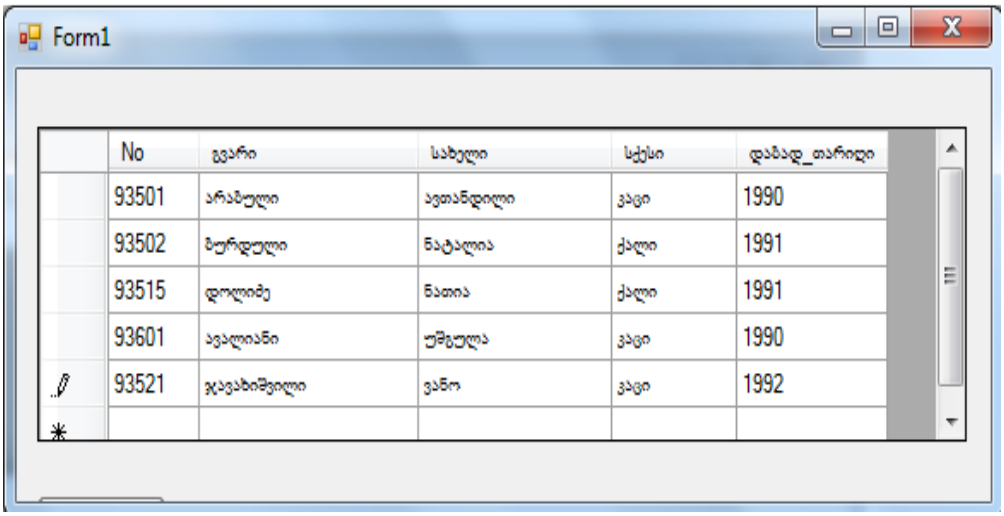
ნახ.4.7

კოდის მუშაობის ახალი შედეგი მოცემულია 4.8 ნახაზზე.



ნახ.4.8

აქვე შეიძლება მონაცემთა სტრიქონების (Rows) შეტანა ველების ქვეშ. მაგალითი ნაჩვენებია 4.9 ნახაზზე.

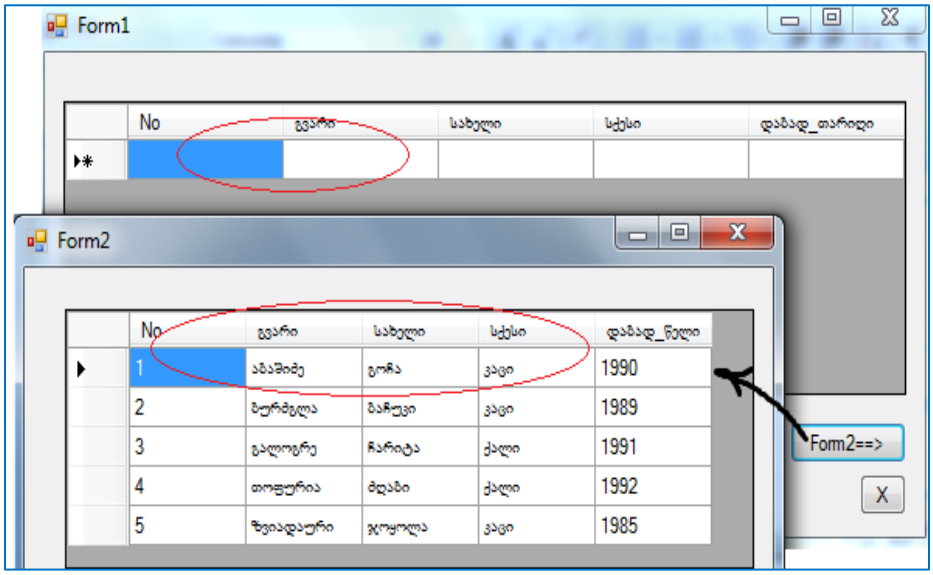


ნახ.4.9

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 ბაზასათან კავშირი (იხ. მომდევნო თავი), ან კოდის Form2_Load მეთოდში უნდა ჩავწეროთ შემდეგი სტრიქონები (იხ.ლისტინგი).

```
// ლისტინგი --- 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", "აბაშიძე", "გოჩა", "კაცი", "1990");
dataGridView1.Rows.Add("2", "ბურძგლა", "ბაჩუკი", "კაცი", "1989");
dataGridView1.Rows.Add("3", "გალოგრე", "ჩარიტა", "ქალი", "1991");
dataGridView1.Rows.Add("4", "თოფურია", "მღაბი", "ქალი", "1992");
dataGridView1.Rows.Add("5", "ზვიადაური", "ჯოყოლა", "კაცი", "1985");
   }
```

პროგრამის ამუშავებით მიიღება 4.10 ნახაზზე მოცემული სურათი. 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.



ნახ.4.10

შედეგიდან 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_1: ვიპოვოთ სახელები და გვარები ყველა 1990 წელს დაბადებული კაცი სტუდენტის.

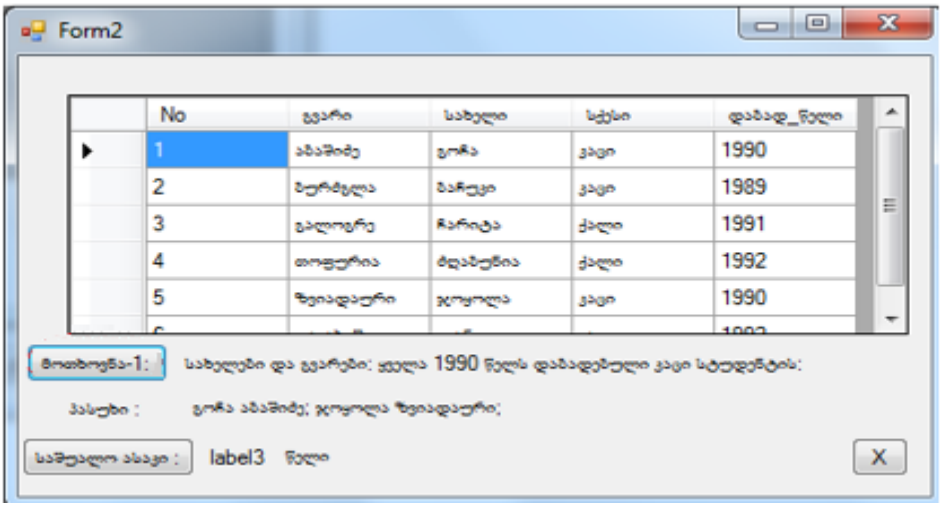
მოთხოვნის ფორმალური მხარე მდგომარეობს ველი „გვარი“-ს მნიშვნელობების გამობეჭდვაში, ველი ***სქესი*** = “კაცი“ მნიშვნელობისთვის.

საჭიროა Form2-ზე დავდოთ ბუტონი და მივაბათ მას მომდევნო ლისტინგში ასახული პროგრამული კოდი.

```
// ლისტინგი --- DataGridView-ში სტრიქონების ამორჩევა პირობით ------
 private void button1_Click(object sender, EventArgs e)
  {
    label1.Text = " ";
```

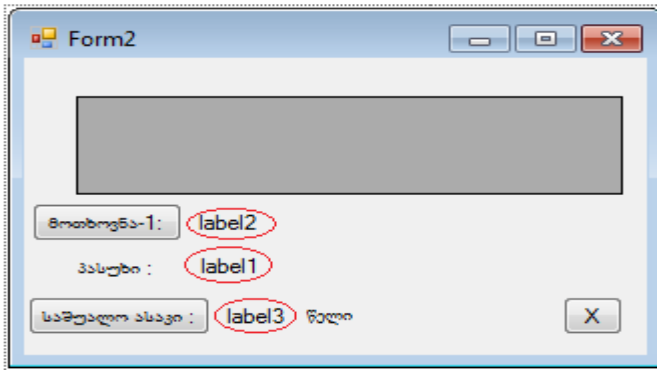
```
    label2.Text = "სახელები და გვარები: ყველა 1990 წელს დაბადებული
                   კაცი სტუდენტის:";
    for (int i = 0; i < dataGridView1.Rows.Count; i++)
     {
       if (dataGridView1.Rows[i].Cells[3].Value == "კაცი" &&
                   dataGridView1.Rows[i].Cells[4].Value == "1990")
         {
           label1.Text += dataGridView1.Rows[i].Cells[2].Value + " " +
                          dataGridView1.Rows[i].Cells[1].Value + "; ";
         }
     }
}
```

შედეგები გამოტანილია 4.11 ნახაზზე.



ნახ.4.11

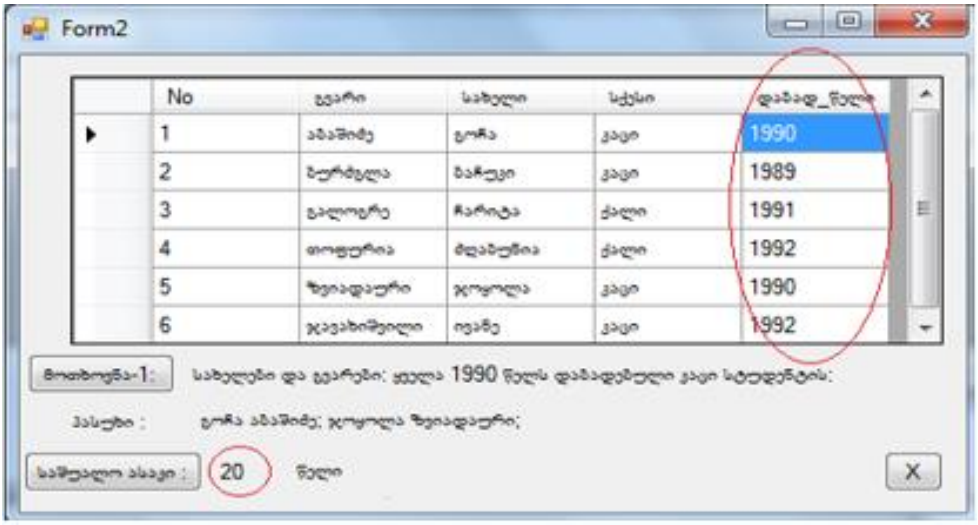
ამოცანა_2: შევადგინოთ კოდი ღილაკისთვის „საშუალო ასაკი“, რომელიც გამოიტანს label3-ში ყველა სტუდენტის საშუალო ასაკის მნიშვნელობას (ნახ. 4.12). მომდევნო ლისტინგში მოცემულია ეს კოდი.



ნახ.4.12. ფორმა ახალი ღილაკით „საშუალო ასაკი“

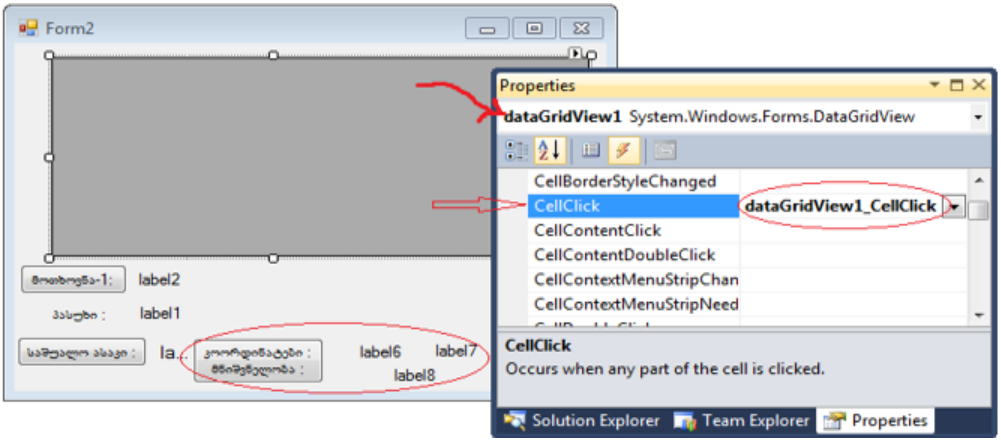
```
  // ლისტინგი --- DataGridView საშუალო ასაკის ანგარიში -------
private void button2_Click(object sender, EventArgs e)
  {
    int Birth_Year, Averag_Age, Sum=0;
    DateTime now = DateTime.Now;  // მიმდინარე თარიღი - სისტემური
    int age,a,b;
    for (int i = 0; i < dataGridView1.Rows.Count; i++)
        {
           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 4.13 ნახაზზე.



ნახ.4.13. მოთხოვნის შედეგი

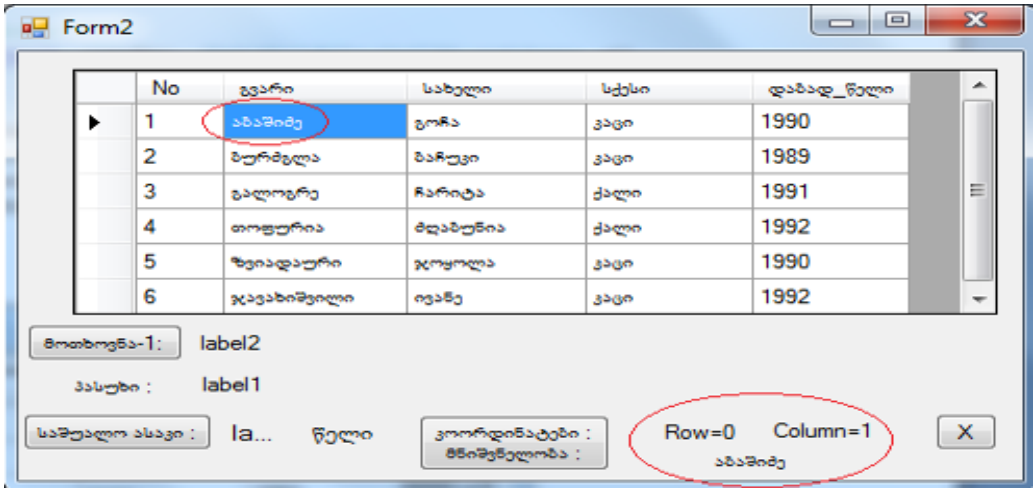
ამოცანა_3: ავაგოთ კოდი, რომელიც იმუშავებს ცხრილთან. კერძოდ, მაუსის კურსორის ცხრილის რომელიმე უჯრაზე დაწკაპუნებით, ეკრანზე გამოიტანს ამ უჯრის სვეტის და სტრიქონის კოორდინატებს და შიგ მოთავსებულ მნიშვნელობას. ***მოვლენის*** შექმნა მოცემულია 4.14 ნახაზზე.



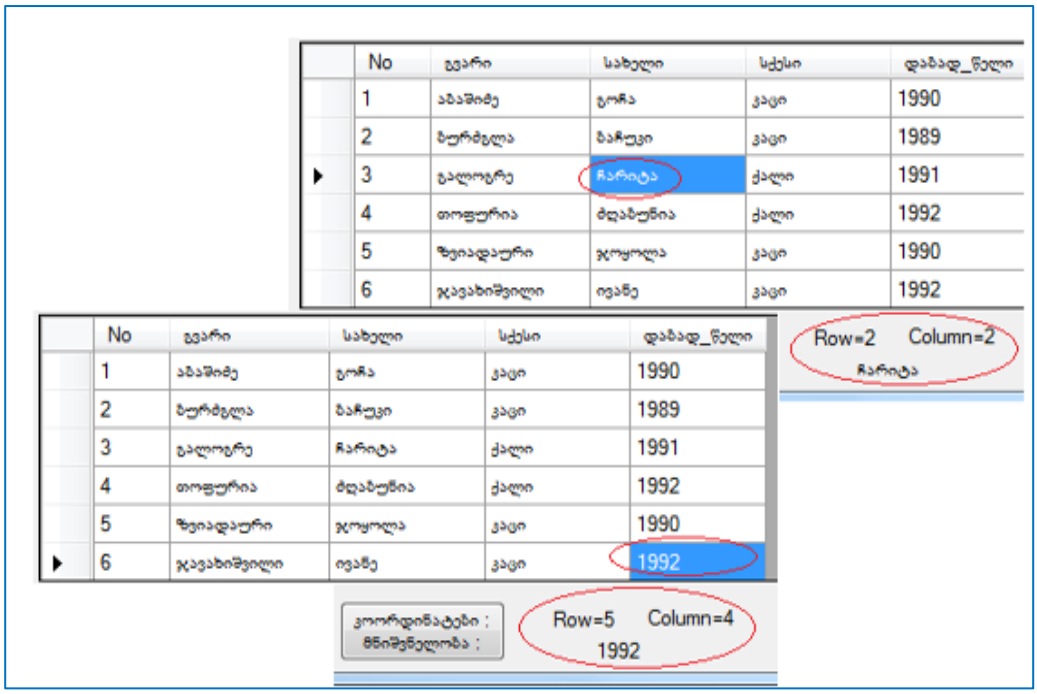
ნახ.4.14

```
// ლისტინგი ---- 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;
      }
```

შდეგები ასახული 4.15-ა,ბ ნახაზზე.



ნახ.4.15-ა



ნახ.4.15-ბ

4.2. C# ენისა და SQL Server ბაზის ერთობლივი გამოყენება ADO.NET დრაივერით და DataGridView კლასით

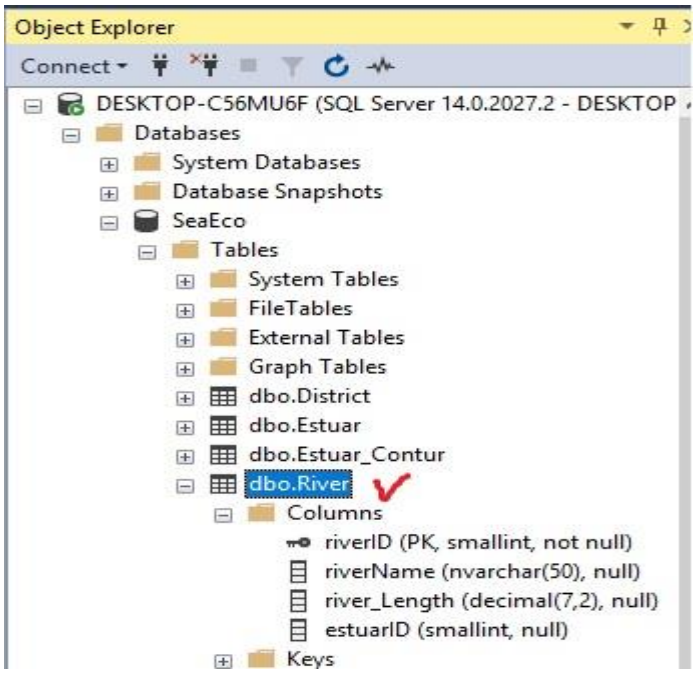
მიზანი: ვიზუალური დაპროგრამების C# ენის, Ms SQL Server მონაცემთა ბაზების პაკეტის და ADO.NET დრაივერის ერთობლივი გამოყენებით პროგრამული აპლიკაციების აგების შესწავლა.

თავის შესავალში ჩვენ მოკლედ შევეხეთ ADO.NET-ის დანიშნულებას და მის ზოგიერთ ობიექტს, რომელთა საშუალებითაც ხორციელდება კავშირი C# ენაზე დაწერილ ინტერფეისსა და მონაცემთა ბაზას შორის.

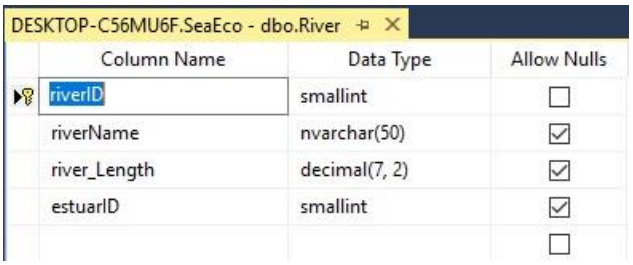
ამოცანა_1: მოცემულია Ms SQL Server მონაცემთა ბაზა (მაგალითად, შავი ზღვის ეკოლოგიური მონიტორინგის სისტემა [33]), რომლის ერთ-ერთი ცხრილი (Table) არის River.dbo (მდინარეები). საჭიროა ავაგოთ C# პროექტი (მომხმარებლის ინტერფეისი), რომელიც მონაცემთა ბაზიდან ამოიღებს მდინარეების მონაცემებს DataGridView ცხრილში, შეძლებს Insert, Update და Delete ოპერაციების განხორციელებას. გამოყენებულ უნდა იქნას ADO.NET დრაივერის საშუალებები.

4.2.1. ექსპერიმენტული მონაცემთა ბაზის მომზადება SQL Server პაკეტით

4.16 ნახაზზე ნაჩვენებია საწყისი მონაცემთა ბაზა, რომელიც Ms SQL Server პაკეტითაა რეალიზებული. (ა) შეესაბამება ბაზის ცხრილების იერარქიასა და აქვე ჩანს River ცხრილის სტრუქტურაც (ბ), მონაცემთა შესაბამისი ტიპებით. (გ)-ზე მოცემულია ჩანაწერები, რომელთა შეტანა მოხდა წინასწარ (თუმცა ჩვენი პროგრამისთვის არაა აუცილებელი მისი არსებობა. შესაძლებელია იგი შეივსოს ინტერფეისიდან).



ნახ.4.16-ა. მონაცემთა ბაზა Ms SQL Server-ში



ნახ.4.16-ბ. River (მდინარეების) ცხრილის სტრუქტურა Ms SQL Server-ში

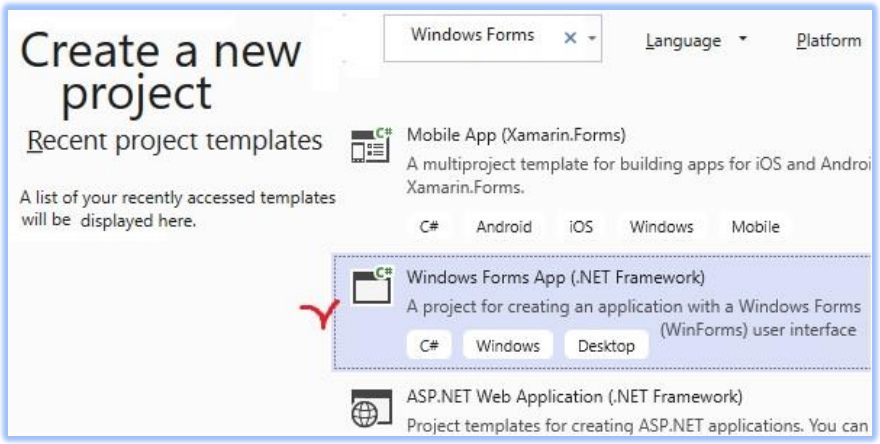
DESKTOP-C56MU6F.SeaEco - dbo.River

riverID	riverName	river_Length	estuarID
1	ჭოროხი	26,00	1
2	კინტრიში	25,20	2
3	ნატანები	60,00	3
4	სუფსა	108,00	4
5	რიონი (სამხრ...	327,00	5
6	რიონი (ჩრდი...	327,00	6
7	ხობისწყალი	150,00	7
8	ენგური	213,00	8

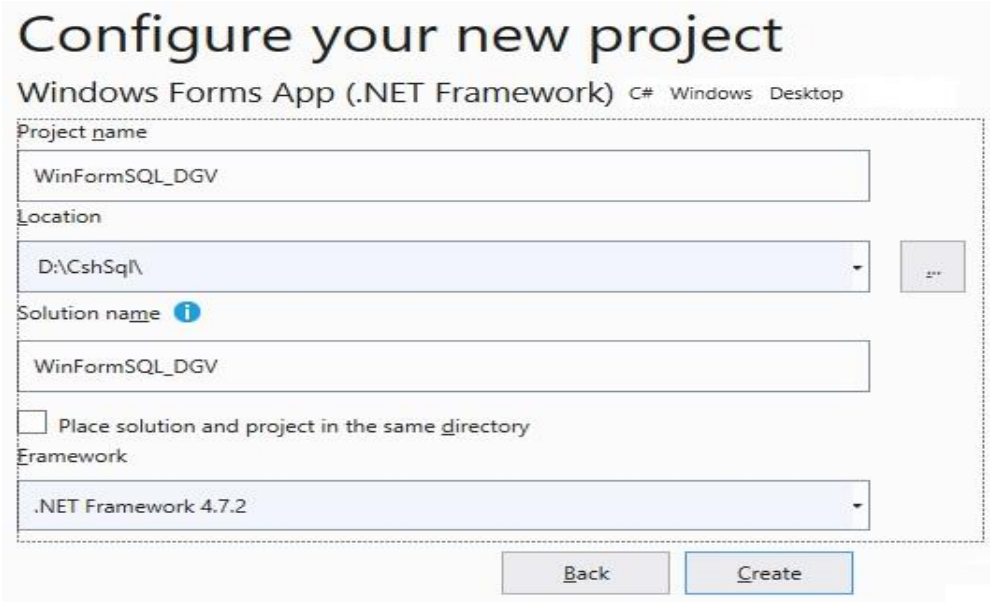
ნახ.4.16-გ. River (მდინარეების) საწყისი ცხრილი
Ms SQL Server-ში

4.2.2. ახალი პროექტის შექმნა .NET პლატფორმაზე C# ენის გამოყენებით

დავიწყოთ ახალი პროექტის აგება Ms Visual Studio.NET სამუშაო გარემოში. 4.17 ნახაზზე ნაჩვენებია პროექტის შექმნის პროცედურა.



ნახ.4.17-ა. WinFormSQL_DGV პროექტის შექმნა

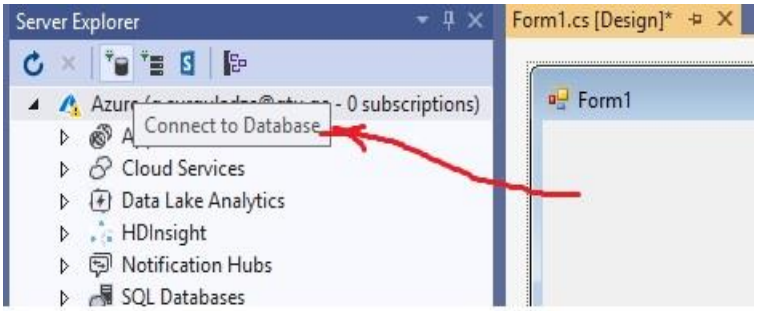


ნახ.4.17-ბ. WinFormSQL_DGV პროექტის შექმნა და განთავსება სისტემის კატალოგში

ვირჩევთ პროექტის სახელს (Name), მისი შენახვის ადგილს (Browse-ს დახმარებით) და Solution name-ს. შემდეგ OK.

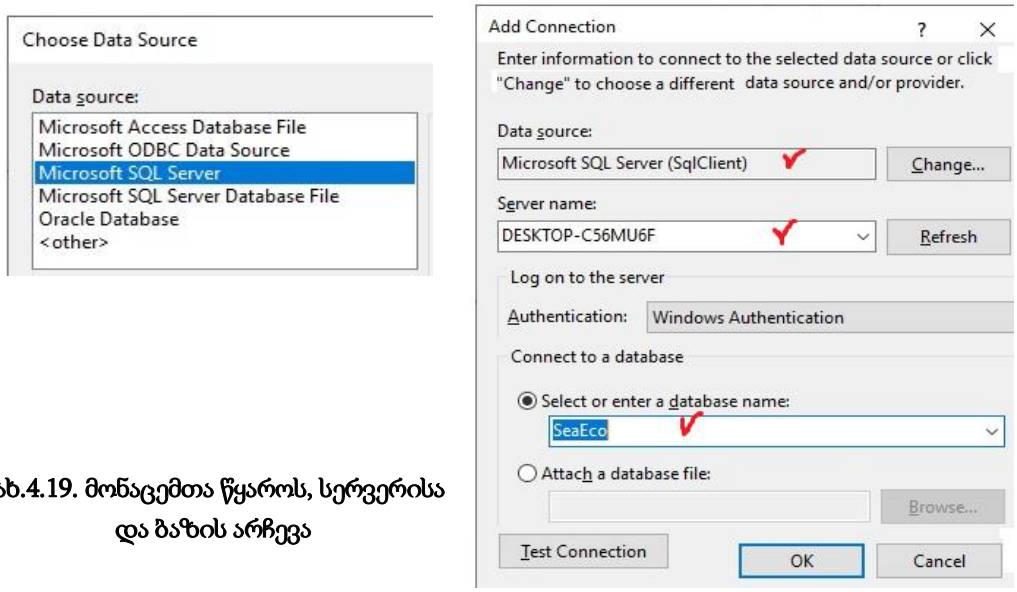
4.2.3. პროექტთან მონაცემთა ბაზის მიერთება

საჭიროა პროექტისთვის განვახორციელოთ მონაცემა ბაზის მიერთება (Connect to Database), რაც 4.18 ნახაზზეა ნაჩვენები.



ნახ.4.18. Connect Database ამოქმედება

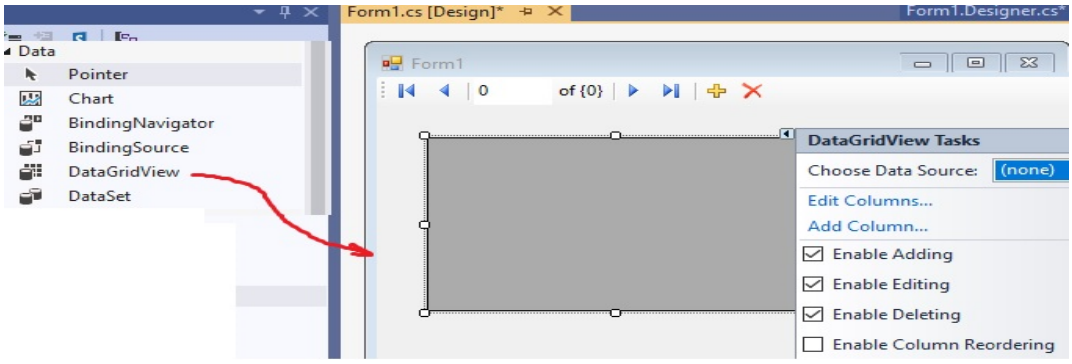
გამოიტანება ახალი ფანჯარა (ნახ.4.19), რომელშიც უნდა შეირჩეს შესაბამისი წყარო (Data Source), სერვერი (Server name) და მონაცემთა ბაზა (Database name).



ნახ.4.19. მონაცემთა წყაროს, სერვერისა და ბაზის არჩევა

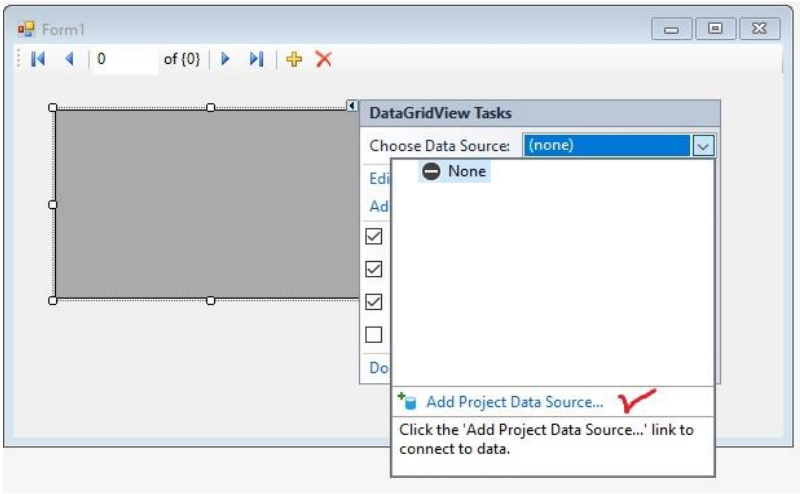
4.2.4. DataGridView ელემენტის გააქტიურება და ცხრილის პარამეტრების განსაზღვრა

ინსტრუმენტების პანელიდან ფორმაზე გადავიტანოთ DataGridView ელემენტი და ზედა მარჯვენა კუთხე პატარა ისრით ავამოქმედოთ. მივიღებთ 4.20 ნახაზს.



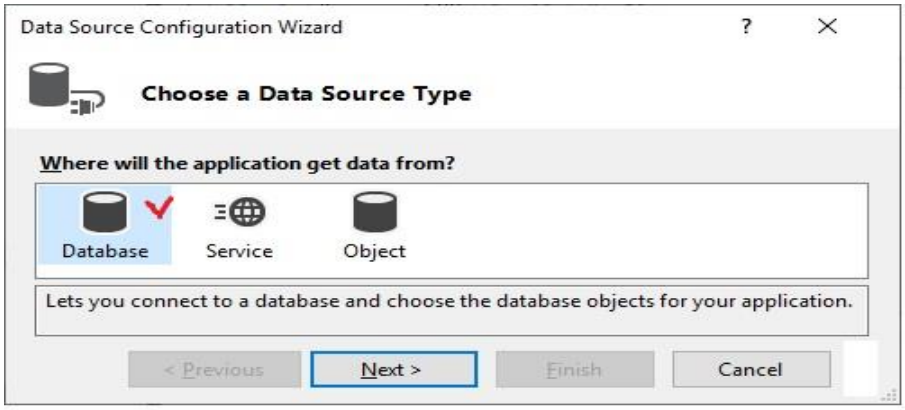
ნახ.4.20. პარამეტრების განსაზღვრა

ნახაზზე ჩანს, რომ ჩამატების, რედაქტირებისა და წაშლის ოპერაციები ნებადართულია (ჩეკბოქსები მონიშნულია). ავირჩიოთ Choose Data Source კომბობოქსის ღილაკი, მივიღებთ 4.21 ნახაზს.



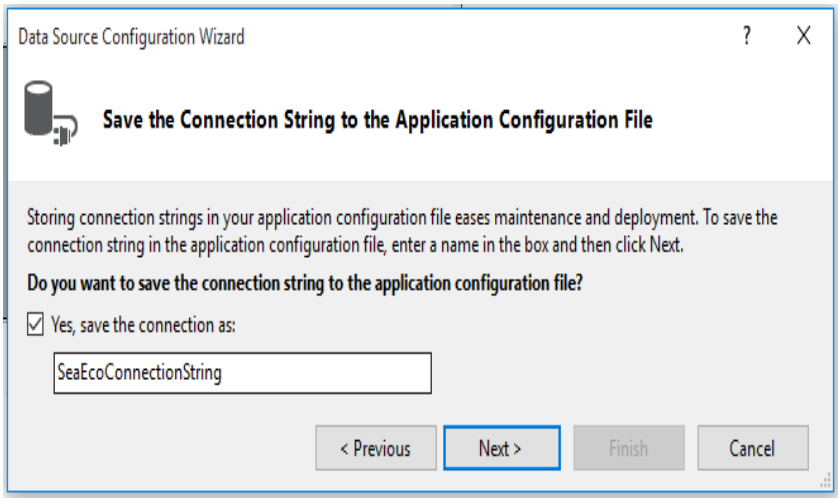
ნახ.4.21. მონაცემთა წყაროს პროექტის დამატება

ავამოქმედოთ Add Project Data Source და გადავიდეთ 4.22 ნახაზზე.



ნახ.4.22. მონაცემთა წყაროს ტიპის არჩევა

ვირჩევთ Database-სა და Next (ნახ.4.23).

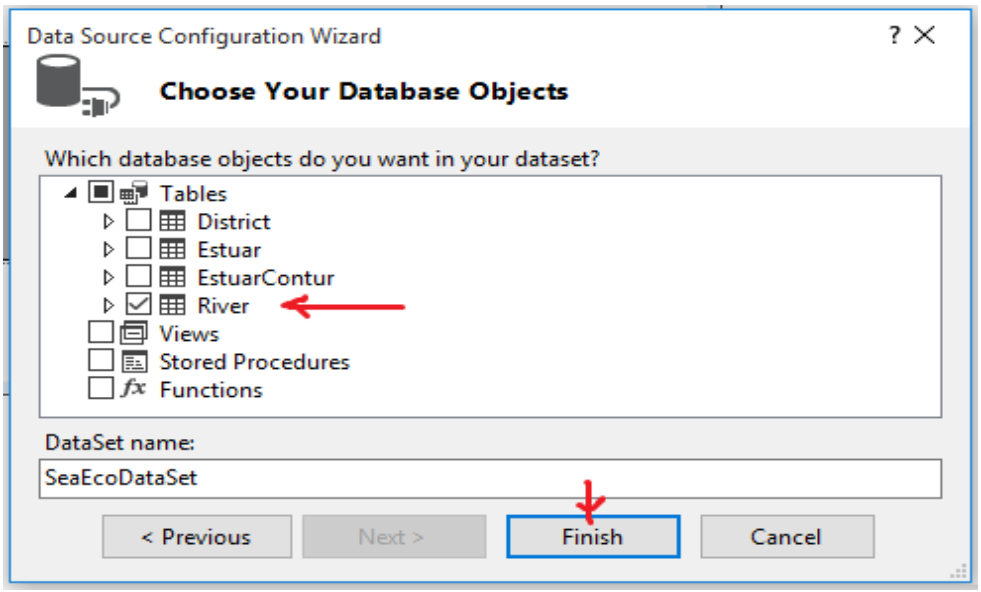


ნახ.4.23. მონაცემთა Connection (მიერთების) შერჩევა

Connection პარამეტრი ყველა კომპიუტერს ექნება თავისი. მისი განსაზღვრა შესაძლებელია Server Explorer-იდან (ჩვენ შემთხვევაში იგი არის: DESKTOP-C56MU6F.SeaEco.dbo). ბოლოს Next და გადავალთ 4.24 ნახაზზე.

ნახ.4.24. Connection String-ის შენახვა აპლიკაციის კონფიგურაციის ფაილში

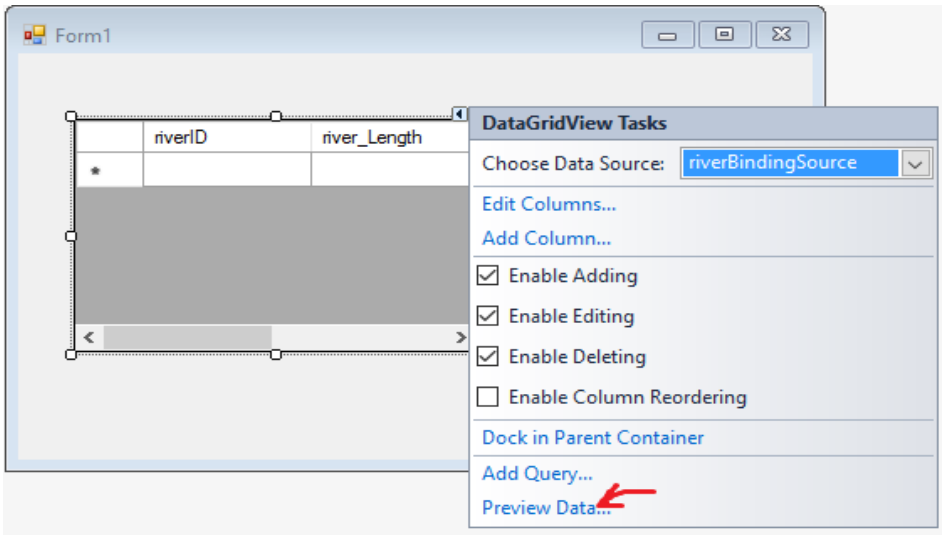
შემდეგ გამოიძახება მონაცემთა ბაზის ობიექტების არჩევის ფანჯარა (ნახ.4.25).



ნახ.4.25. ობიექტების მონიშვნა (მაგალითად, River)

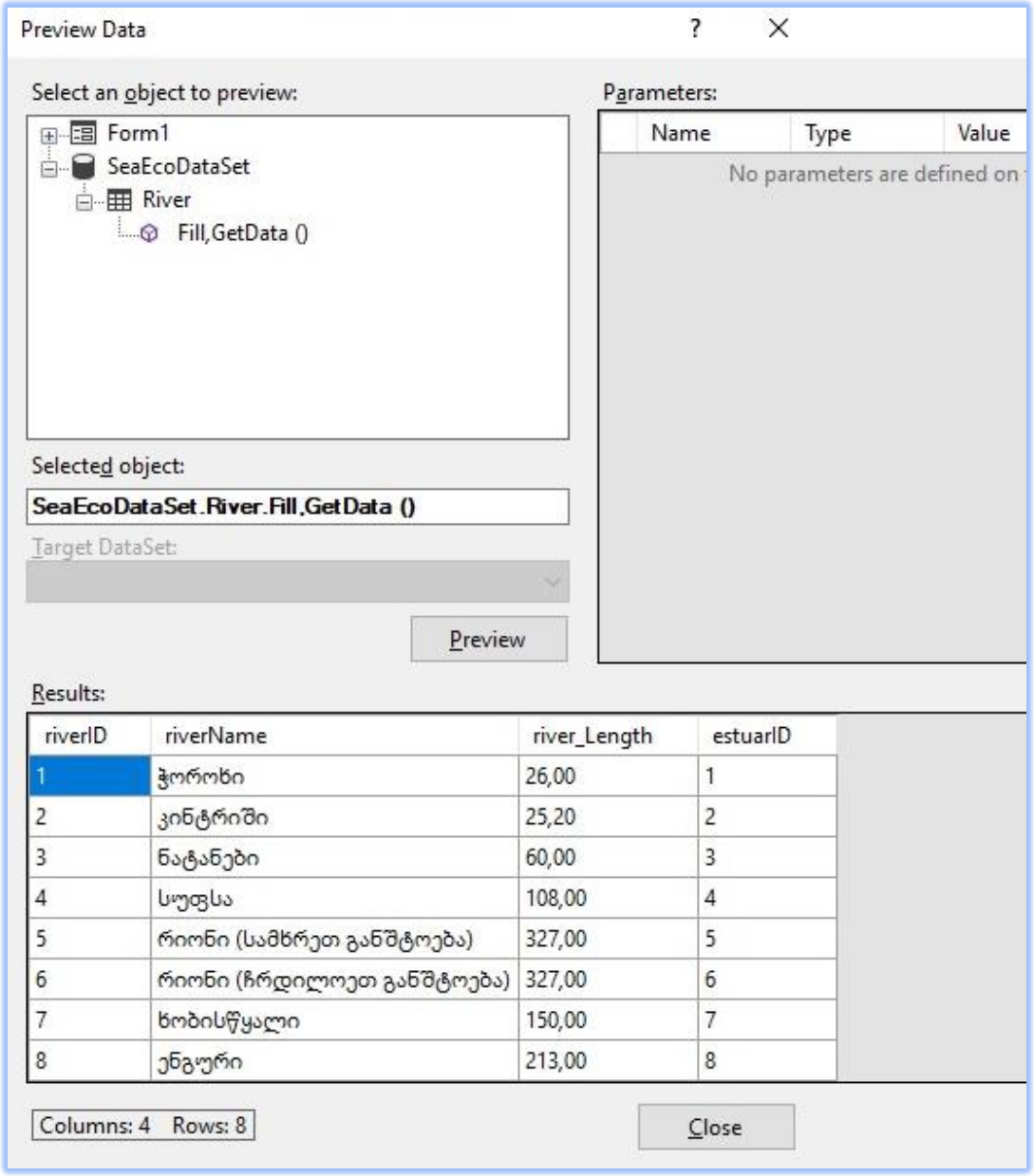
4.26 ნახაზზე ნაჩვენებია მონაცემთა წყაროს (Data Source) განსაზღვრის შედეგის მნიშვნელობა, ჩვენ შემთხვევაში იგი არის:

riverBindingSource.



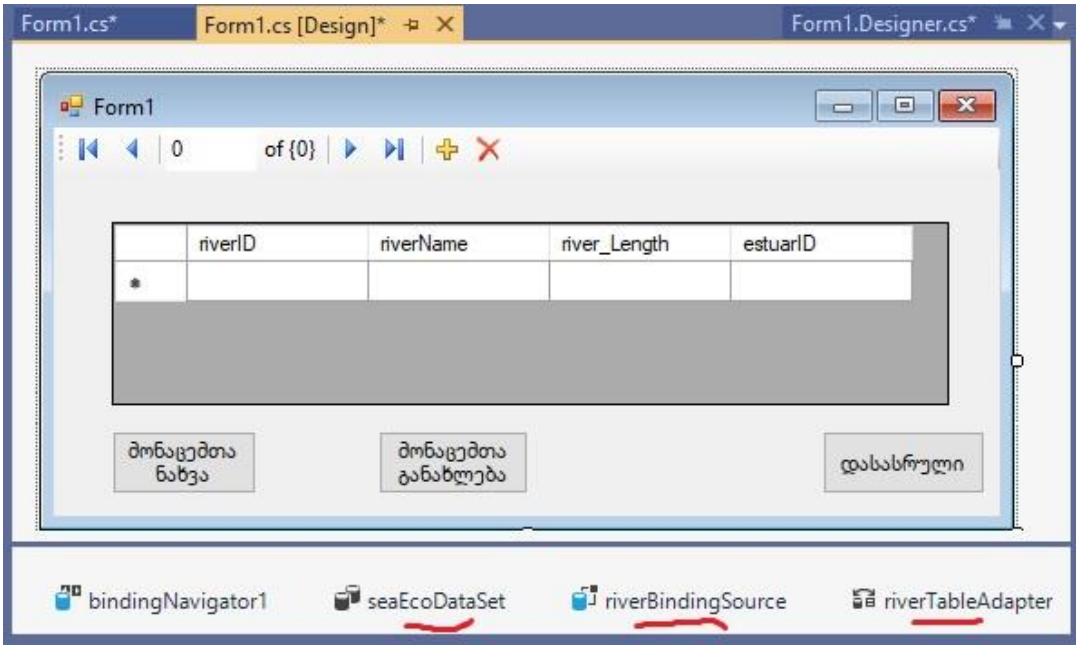
ნახ.4.26. Data Source შედეგი: "riverBindingSource"

აქვე შეიძლება გამოვიყენოთ Preview Data ლინკი და დავათვალიეროთ წინასწარ მიერთებული ბაზის ცხრილის ჩანაწერები (ნახ.4.27).



ნახ.4.27. Preview Data ცხრილი

ბოლოს, Form1-ს Properties-ში შევუცვალოთ სახელი „შავი ზღვის მდინარეები (საქართველოს საზღვრებში)“, ინსტრუმენტების პანელიდან გადმოვიტანოთ სამი ღილაკი (Button1,2.3), დავარქვათ სახელები. მიიღება 4.28 ნახაზი.



ნახ.4.28. პროექტის ძირითადი ინტერფეისი

4.2.5. პროგრამული რეალიზაციის საკითხები

ახლა გადავიდეთ სისტემის ინტერფეისის ღილაკების ფუნქციების დაპროგრამებაზე, ანუ უნდა განხორციელდეს SQL Server-ის შესაბამისი ბაზის ცხრილის ჩანაწერების დათვალიერება, ჩამატება, შეცვლა და წაშლა.

თავდაპირველად ჩავამატოთ სახელსივრცის სტრიქონი:

```
using System.Data.SqlClient;
```

შემდეგ გამოვაცხადოთ გლობალური ცვლადები:

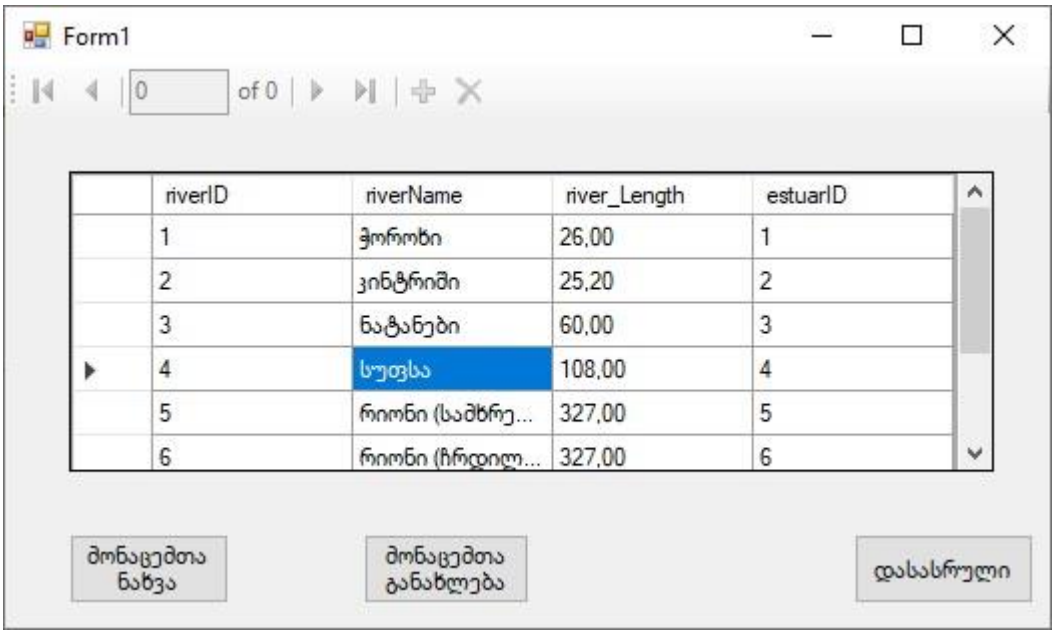
```
SqlDataAdapter sda;
SqlCommandBuilder scb;
DataTable dt;
```

„მონაცემთა ნახვის“ ღილაკისათვის (button1) გვექნება შემდეგი კოდი:

```
private void Button1_Click(object sender, EventArgs e)
  {
    SqlConnection con = new SqlConnection("Data Source=DESKTOP-C56MU6F;
              Initial Catalog = SeaEco; Integrated Security = True");
    sda = new SqlDataAdapter(@"SELECT riverID,
                            river_Length, riverName,
                            estuarID from River", con);
```

```
        dt = new DataTable();
        sda.Fill(dt);
        dataGridView1.DataSource = dt;   }
```

პროგრამის ამუშავებით, თუ მასში არაა შეცდომები, მიიღება 4.28 ნახაზი.



ნახ.4.28. „მონაცემთა ნახვის“ (DataShow) ღილაკის ამოქმედებით მიღებული შედეგი

„მონაცემთა განახლების“ ღილაკის კოდი (Insert, Update, Delete) ნაჩვენებია ქვემოთ:

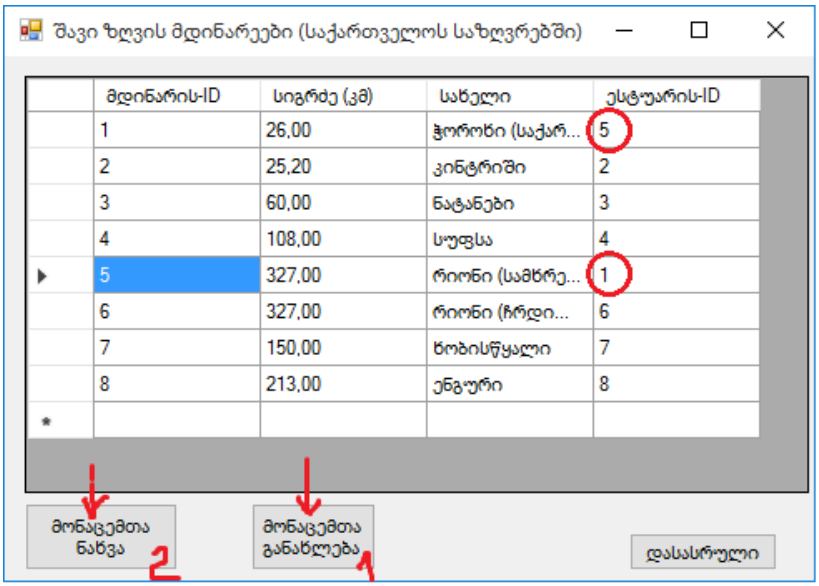
```
private void Button2_Click(object sender, EventArgs e)
  {
     scb = new SqlCommandBuilder(sda);
     sda.Update(dt);
  }
```

მთლიანი პროგრამის კოდი მოცემულია ლისტინგში.

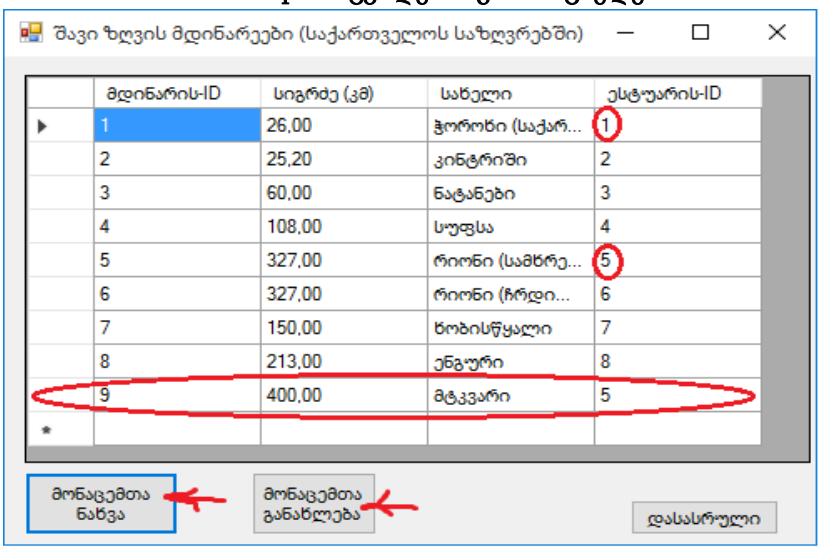
```
// --- ლისტინგი --- Insert, Update, Delete for DataGridView_SQL----
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```
using System.Data.SqlClient; // !!!

namespace WinFormSQL_DGV
{
    public partial class Form1 : Form
    {
        SqlDataAdapter sda;
        SqlCommandBuilder scb;
        DataTable dt;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // TODO: This line of code loads data into the
'seaEcoDataSet.River' table. You can move, or remove it, as needed.
            this.riverTableAdapter.Fill(this.seaEcoDataSet.River);

        }
        // monacemTa naxva
        private void Button1_Click(object sender, EventArgs e)
        {
            SqlConnection con = new SqlConnection("Data Source=DESKTOP-
C56MU6F; Initial Catalog = SeaEco; Integrated Security = True");
            sda = new SqlDataAdapter(@"SELECT riverID,
                                       river_Length, riverName,
                                       estuarID from River", con);
            dt = new DataTable();
            sda.Fill(dt);
            dataGridView1.DataSource = dt;

        }

        // ganaxleba
        private void Button2_Click(object sender, EventArgs e)
        {
            scb = new SqlCommandBuilder(sda);
            sda.Update(dt);

        }

        private void Button3_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

4.29 ნახაზზე მოცემულია არსებულ ცხრილში ორი მნიშვნელობის (ესტუარისID) შეცვლა (ახალი რიცხვები ნაჩვენებია წრეში), შემდეგ 1-ელი და მე-2 ღილაკების ამოქმედებით ბაზაში ჩაიწერება ეს შეცვლილი მნიშვნელობები (შეიძლება დავხუროთ პროგრამა და თავიდან ავამუშავოთ).



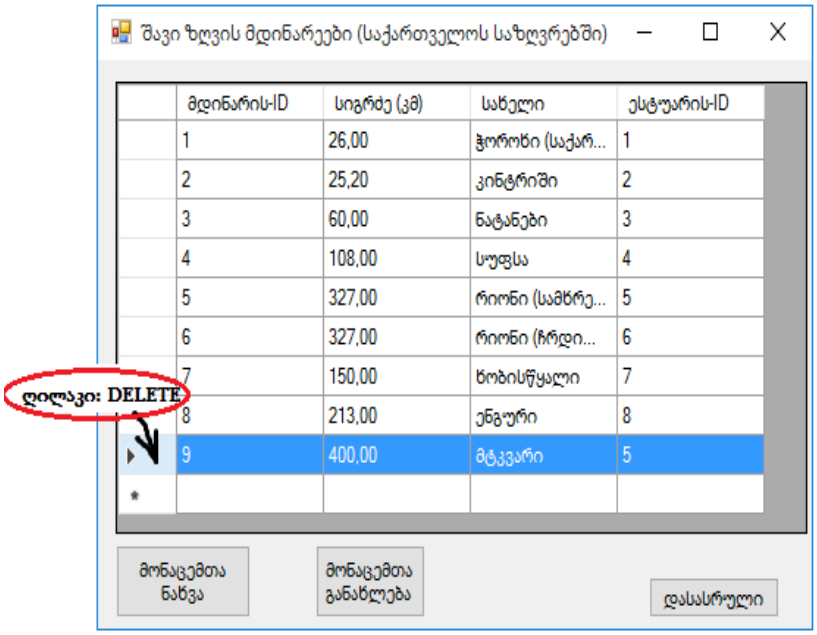
ნახ.4.29. Update ცვილების განხორციელება



ნახ.4.30. Insert ოპერაციის განხორციელება

ბოლოს ნაჩვენებია სტრიქონის წაშლის (Delete) ოპერაცია. იგი ხორციელდება მაგალითად, „**მდინარის-ID**“ შესაბამისი სტრიქონის მონიშვნით

და შემდეგ კომპიუტერის კლავიატურის Delete- ღილაკის ამოქმედებით (ნახ.4.31).



ნახ.4.31. Delete ოპერაციის განხორციელება

ამით დავასრულეთ ვიზუალური C# ელემენტების საფუძველზე მონაცემთა ბაზასთან დაკავშირების რეალიზაციის ამოცანის განხილვა, რომლის შედეგადაც აგებულ იქნა შავი ზღვის ეკომონიტორინგის ინფორმაციული სისტემის ფრაგმენტი.

დავალება: SQL Server მონაცემთა ბაზის ცხრილის ჩანაწერების დასამუშავებლად გამოიყენეთ BindingNavigator ინსტრუმენტი, გამოიკვლიეთ მისი ფუნქციები.

ააგეთ მომხმარებლის ინტერფეისი რომელიმე საპრობლემო სფეროსთვის (ამოცანა შეათანხმეთ მასწავლებელთან).

V თავი
.NET პლატფორმაზე ინფორმაციული სისტემების დაპროგრამება

5.1. აპლიკაციის აგება დაპროგრამების რამდენიმე ენის ერთად გამოყენების საფუძველზე (.dll ფაილების შექმნით)

სამუშაოს მიზანი: ობიექტ-ორიენტირებული დაპროგრამების ენების: C++, Visual Basic და C# გამოყენებით ერთი პროგრამული პროექტის აგების ტექნოლოგიის შესწავლა .NET პლატფორმაზე. ამ ენებზე .dll ფაილების შექმნის და მათი ერთად მუშაობის პროცესის გაცნობა.

1. თეორიული ნაწილი

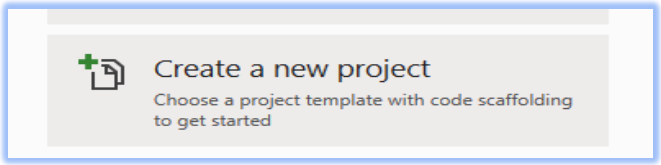
ილუსტრირებულია .NET ტექნოლოგიის ერთ-ერთი ***ძირითადი პრინციპი,*** რომ პროგრამული აპლიკაციის დამუშავება შესაძლებელია დეველოპერების გუნდში სხვადასხვა პროგრამული ენის მცოდნე სპეციალისტების მიერ, კერძოდ C++, Visual Basic და C# ენების საფუძველზე. პროექტის აგებისას გათვალისწინებულ უნდა იქნას საერთო მოთხოვნები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

2. პრაქტიკული ნაწილი

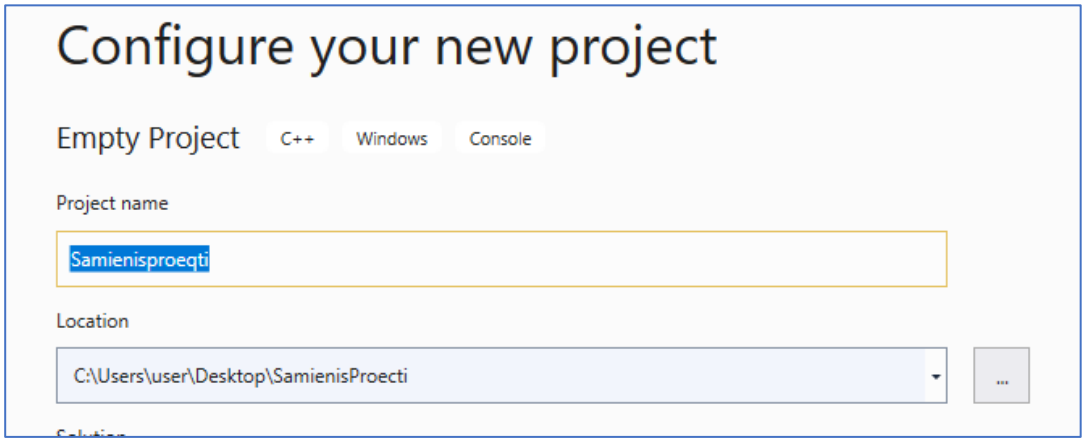
განვახორციელოთ პროექტის პროგრამული რეალიზაცია Visual Studio .NET Framework 4.7.2 სამუშაო გარემოში.

- შევარჩიოთ ახალი პროექტის განთავსების ადგილი
- მენიუდან გამოვიძახოთ ახალი ცარიელი პროექტის შექმნის პროგრამა: (Visual Studio 2019 .NET Framework 4.7.2 პროგრამის გააქტიურების შემდეგ,



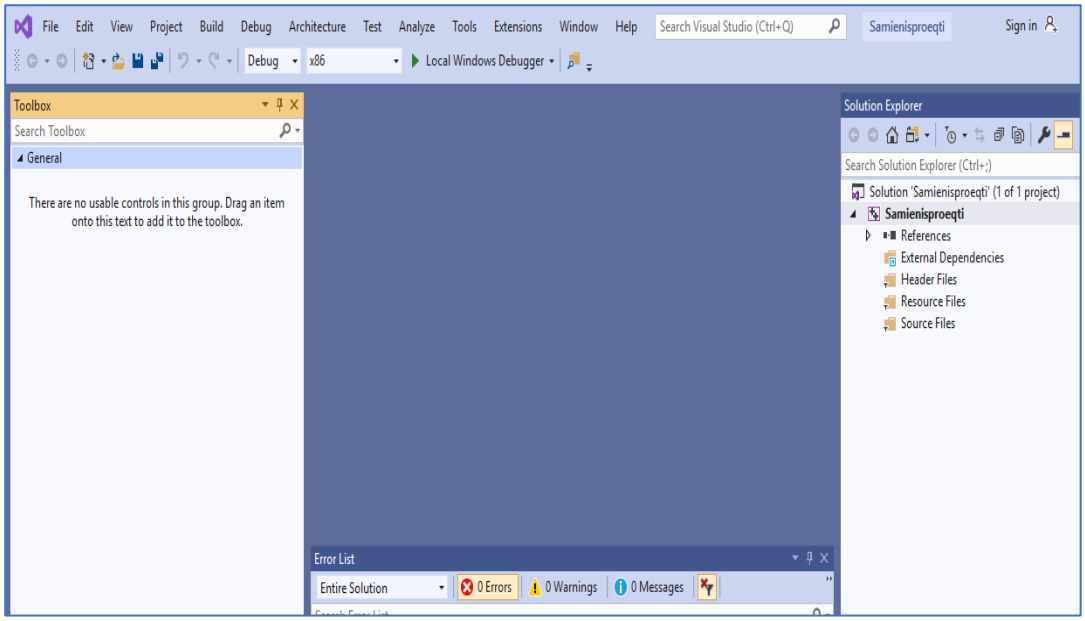
ჩამონათვალიდან Get started ვირჩევთ -> **Create a new Project -> Next**

მიიღება ფანჯარა, სადაც განისაზღვრება პროექტის სახელი, მაგალითად, Samieenisproeqti და შენახვის ადგილი - Location (ნახ.5.1).



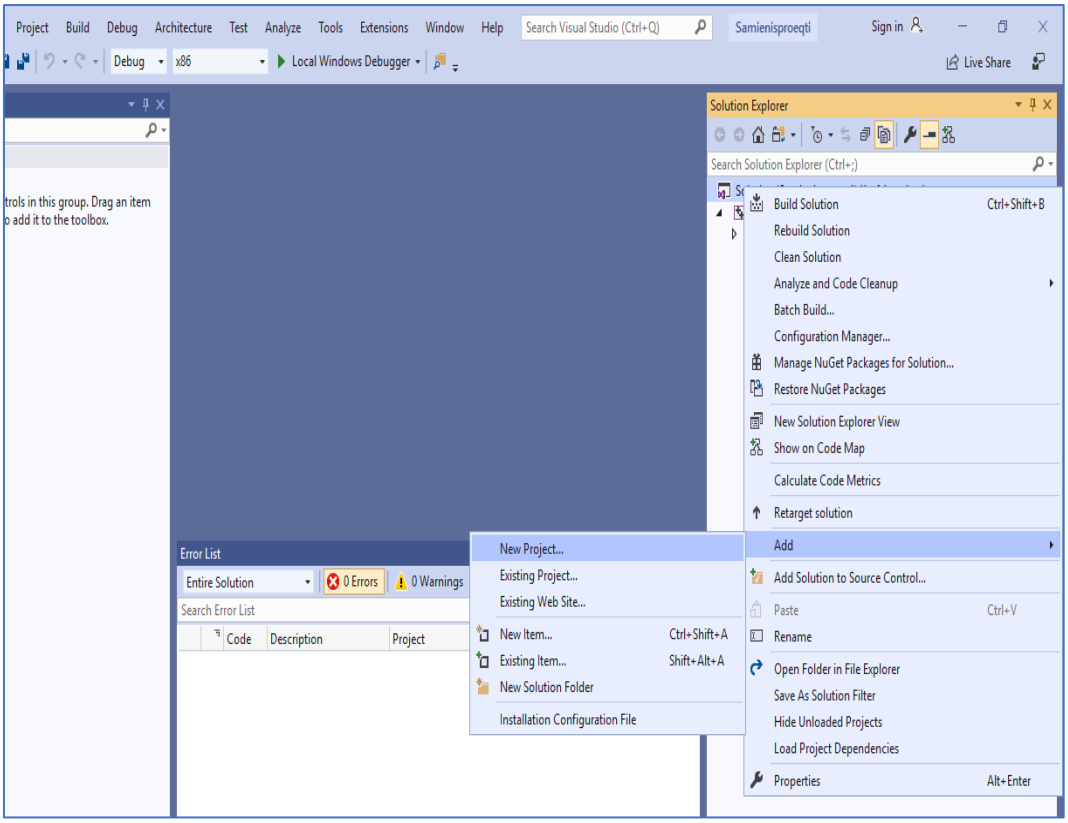
ნახ.5.1

ქვედა მარჯვენა კუთხეში მოთავსებული Create ღილაკის გააქტიურების შემდეგ მიიღება ფანჯარა (ნახ.5.2).



ნახ.5.2

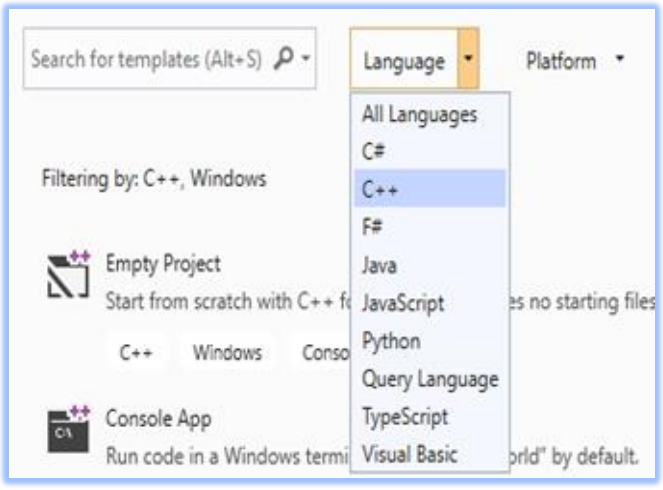
მიღებული ფანჯრის Solution Explorer - ის არეში ვამატებთ ახალ პროექტს. ვქმნით კლასს C++ ენის ბაზაზე. ამისათვის მოვნიშნავთ Solution SamienisProeqti -ს კონტექსტური მენიუდან ნიმუშის შესაბამისად ვამატებთ ახალ პროექტს:



ნახ.5.3

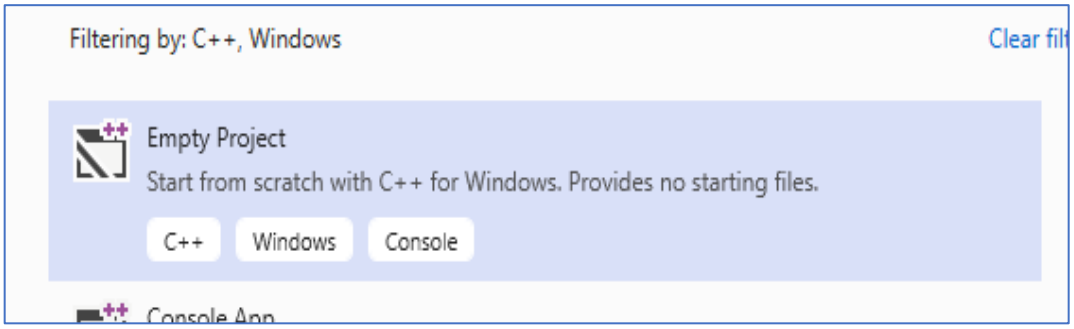
- **კლასის შექმნა ახალ პროექტში C++.NET ენაზე**

ცარიელ პროექტს (გადაწყვეტა - Solution 'SamienisProjecti') დავამატოთ ახალი (ქვე)პროექტი მასზე მარჯვენა ღილაკის დაწკაპუნებით და შემდეგ **Add -> New Project** არჩევით, მიიღება ფანჯარა, საიდანაც Lenguage ჩამონათვალიდან, ვირჩევთ C++ ენას (ნახ.5.4).



ნახ.5.4.

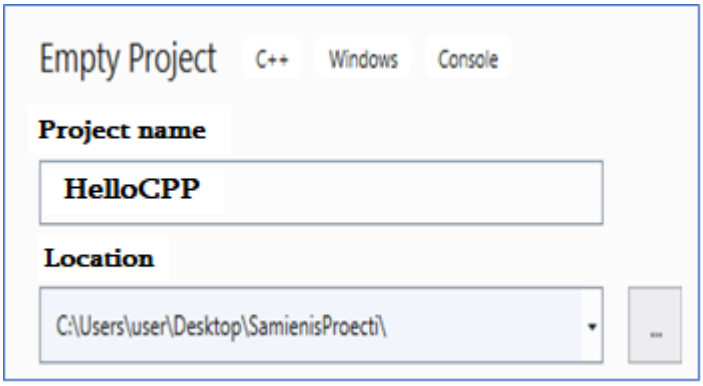
პროექტისთვის კი ვირჩევთ empty Project-ს (ნახ.5.5).



ნახ.5.5

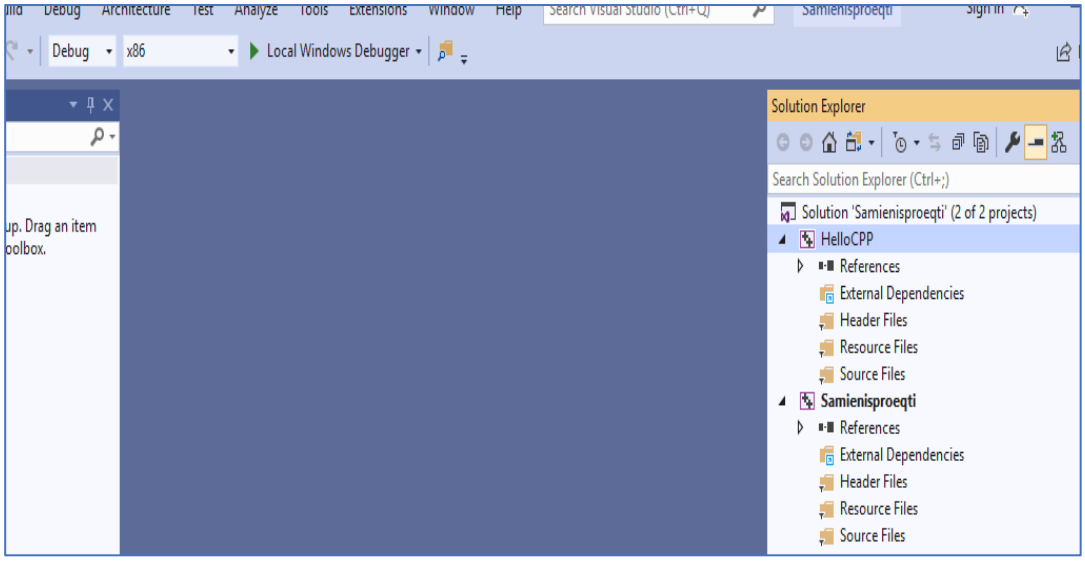
შენიშვნა! წინა ვერსიებისგან განსხვავებით Blank Solution-ის ნაცვლად VS2019-ში გვაქვს empty Project

Project Name - ში ვწერთ სახელს, ხოლო ლოკაცია გააქტიურდება ავტომატურად (რომლის შეცვლაც საჭიროების შემთხვევაში შესაძლებელია) (ნახ.5.6).



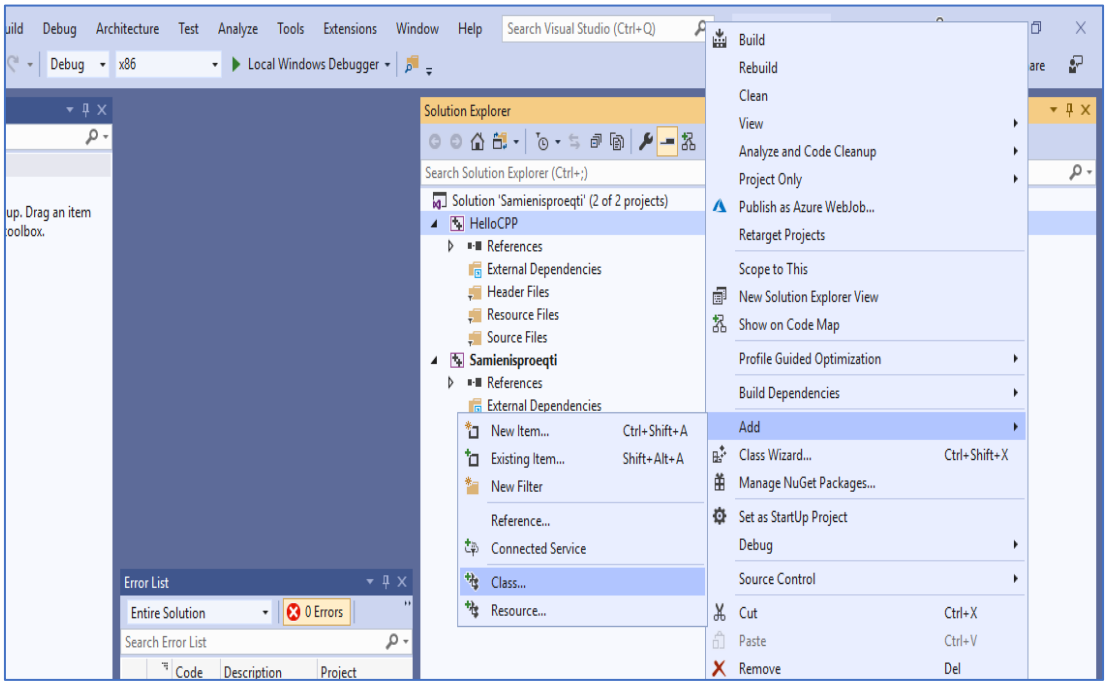
ნახ.5.6

Create ღილაკის გააქტიურების შემდეგ შეიქმნება აპლიკაცია, რომელიც ნეიტრალური იქნება დაპროგრამების ენებისადმი. შედეგად მივიღებთ 5.7 ნახაზზე ნაჩვენებ ფანჯარას.



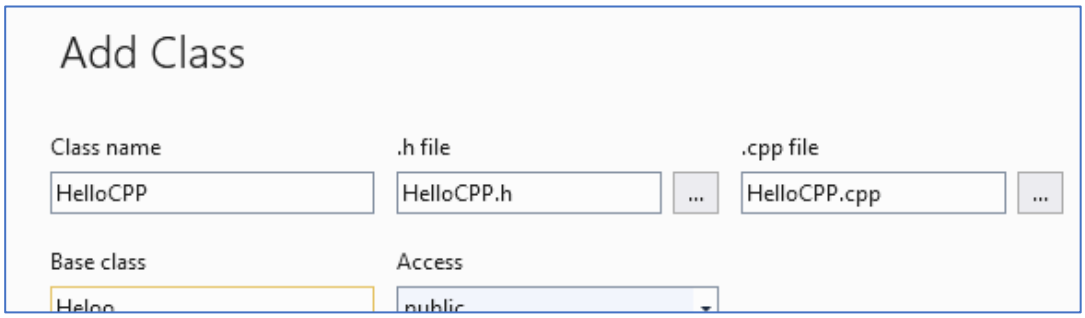
ნახ. 5.7

კლასის შესაქმნელად მოვნიშნავთ HelloCPP და ნიმუშის შესაბამისად (ნახ. 5.8), ვააქტიურებთ ბრძანებებს: Add → Class.

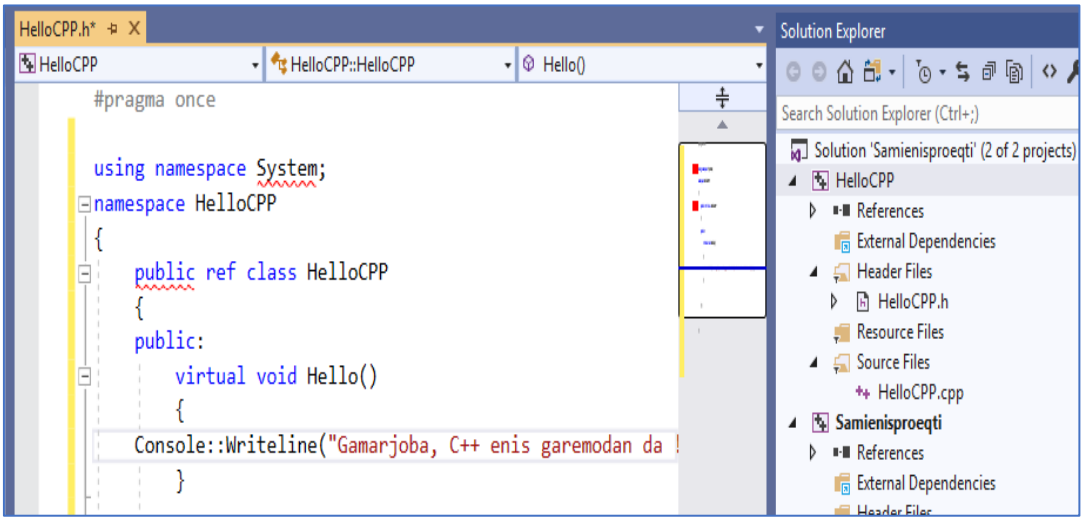


ნახ.17.3

შედეგად მიიღება ფანჯარა, სადაც Class name-ში უნდა ჩავწეროთ კლასის დასახელება (ნახ. 5.9).



ნახ.5.9



ნახ.5.10

გავხსნათ Hello.CPP.h ფაილი HelloCPP.h კოდი მოცემულია ლისტინგში.

```
// ---- ლისტინგი_17.1 -- შეცვლილი HelloCPP.h ----
#pragma once
using namespace System;
namespace HelloCPP
{
   public ref class HelloCPP
    {
        // --- აქ ჩაემატება კლასის მეთოდი ----
```

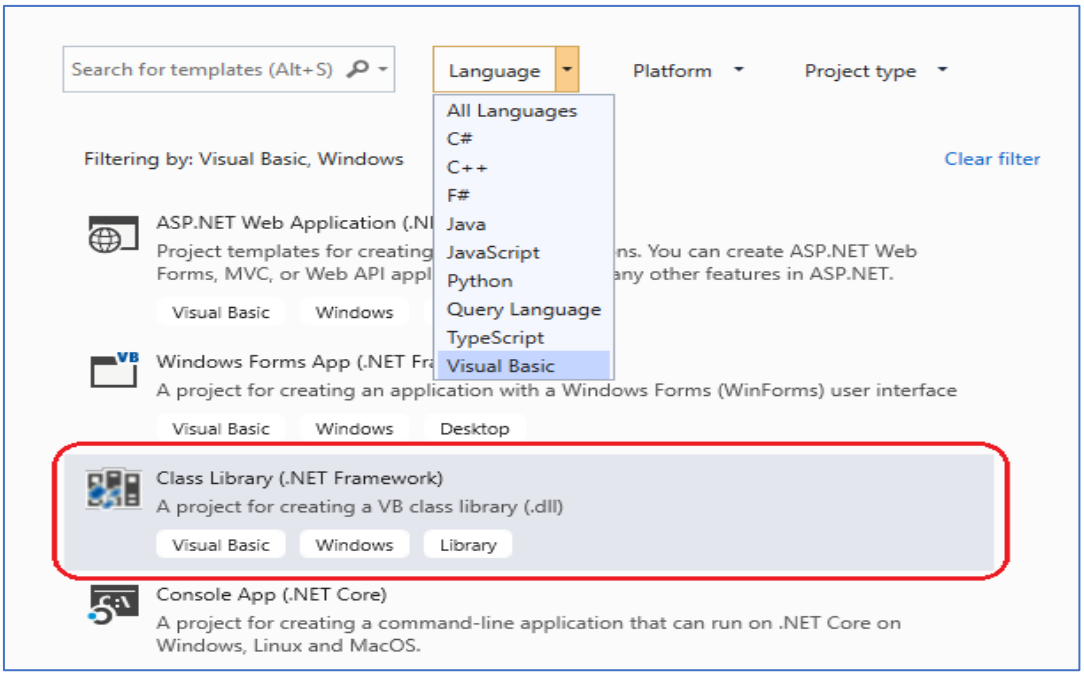
```
      public:
   virtual void Hello()
    {
       Console::WriteLine("Mogesalmebit C++ enis garemodan da !\n viZaxeb
Visual Basics !\n\n“);
      }
   };
}
```

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

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

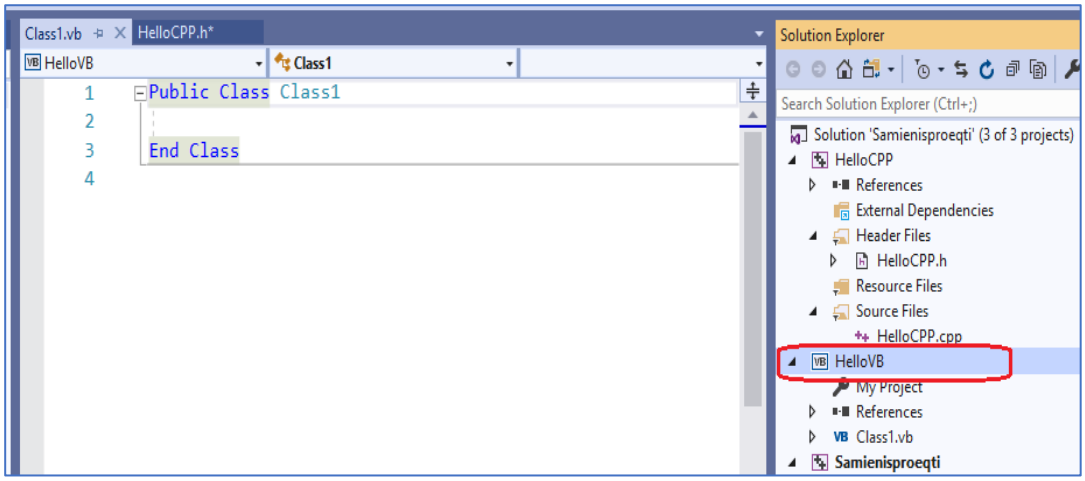
- **კლასის შექმნა ახალ პროექტში Visual Basic.NET ენაზე**

დავამატოთ Solution-ში ახალი პროექტი და მივუთითოთ, რომ იგი შეიქმნას Visual Basic .NET -ში, ავირიოთ Visual Basic ენა და ჩანართი: Class Library; პროექტი შევინახოთ სახელით Name: HelloVB (ნახ.5.11).



ნახ.5.11

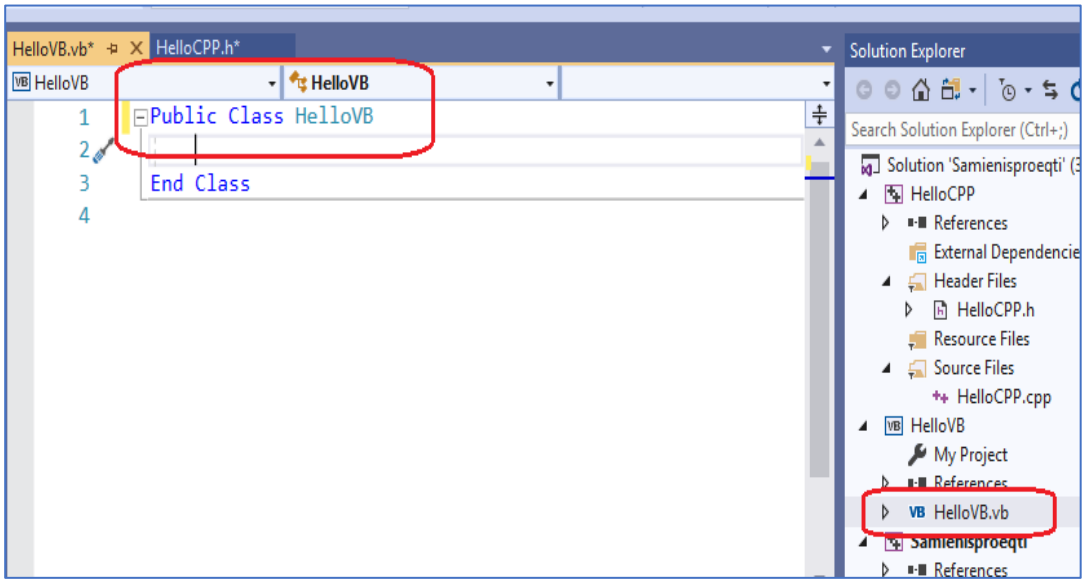
მივიღებთ ასეთ შედეგს (ნახ.5.12). ახალი პროექტი HelloVB დაემატება Solution Explorer პანელზე თავისი შემადგენელი ფაილებით.



ნახ.5.12

ახლა უკვე გვაქვს HelloCPP და HelloVB ორი პროექტი.

მეორეში ავირჩიოთ Class1.vb ფაილი და შევუცვალოთ სახელი შინაარსის გათვალისწინებით, მაგალითად, HelloVB.vb (ნახ. 5.13),



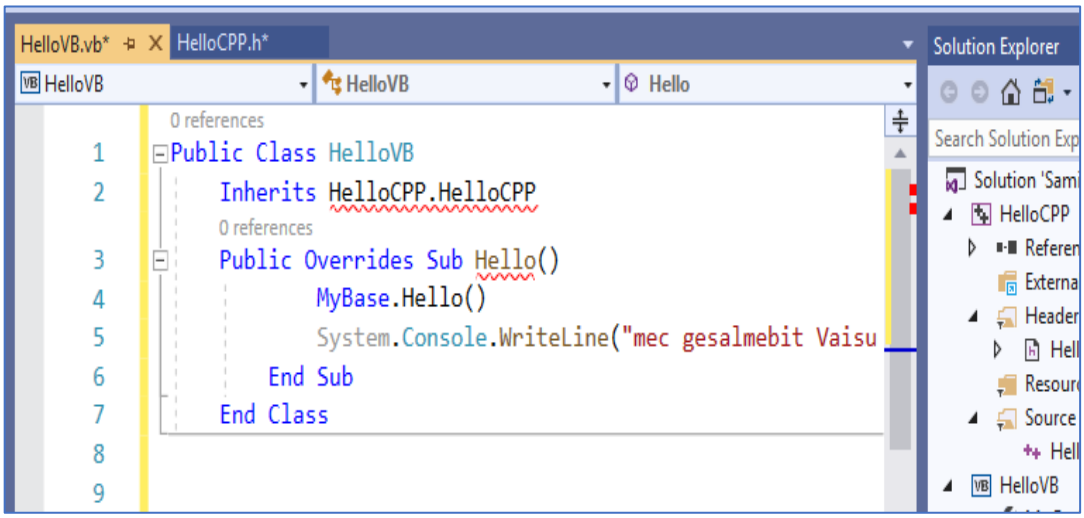
ნახ.5.13

შემდეგ გავხსნათ იგი და ჩავწეროთ ჩვენთვის საჭირო ტექსტი (იხ. ლისტინგი).

```
// ' --ლისტინგი---  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
```

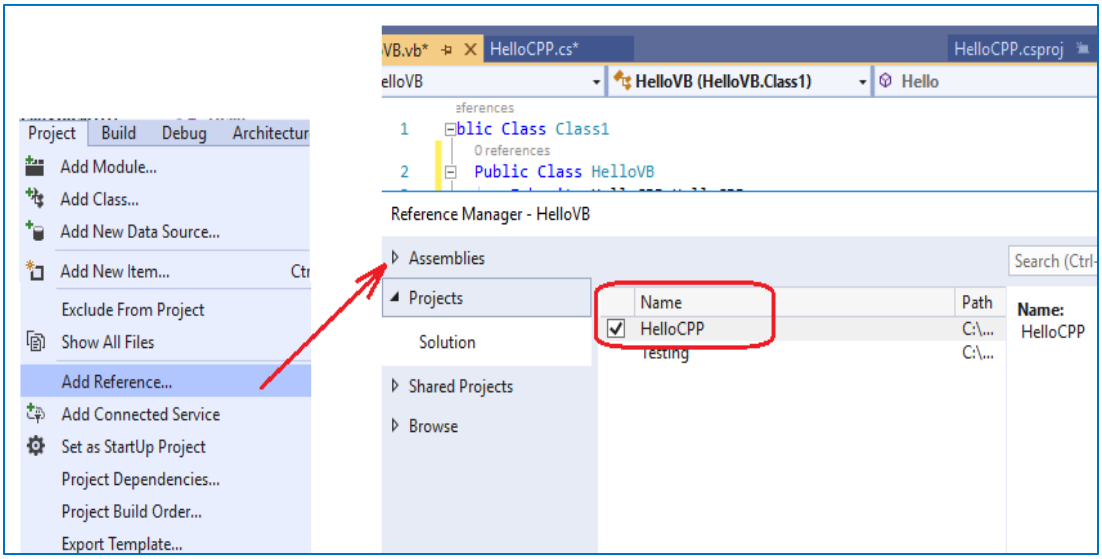
ეს კოდი განსაზღვრავს HelloVB კლასს, რომელიც მემკვიდრეობით იქნა მიღებული C++ ის მმართველი კლასიდან HelloCPP. ამგვარად, HelloVB კლასი ჩაანაცვლებს (Overrides) მშობელი HelloCPP კლასის ვირტუალურ Hello() მეთოდს, ოღონდ ჯერ გამოიძახებს Hello() მეთოდის ვერსიას მშობელი კლასიდან HelloCPP, შემდეკ კი გამოყავს ეკრანზე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

საყურადღებოა, რომ კოდის რეადაქტორში 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 ფრაგმეტზე, იგი მოგვცემს შეცდომის მნიშვნელობას (ნახ.5.14).



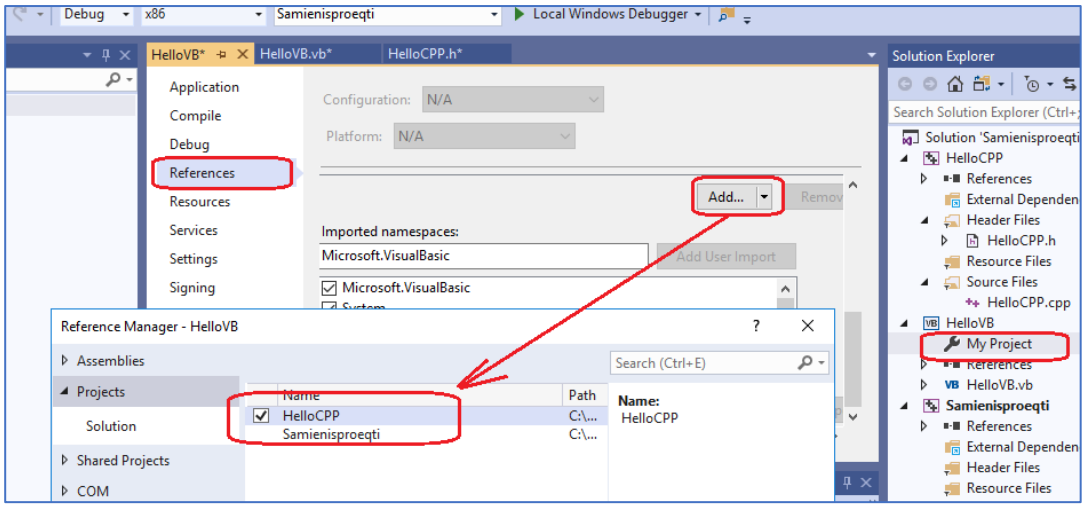
ნახ.5.14

აუცილებელია Visual Basic-ის კომპილატორს მიეთითოს თუ სად იმყოფება ეს ნაკრები (assembly), რომელიც განსაზღვრავს მოცემულ ტიპს. ამისათვის მთავარი მენიუს Project-> Add Reference->Projects-დან ავირჩევთ HelloCPP-ს და OK. (ნახ. 5.15).



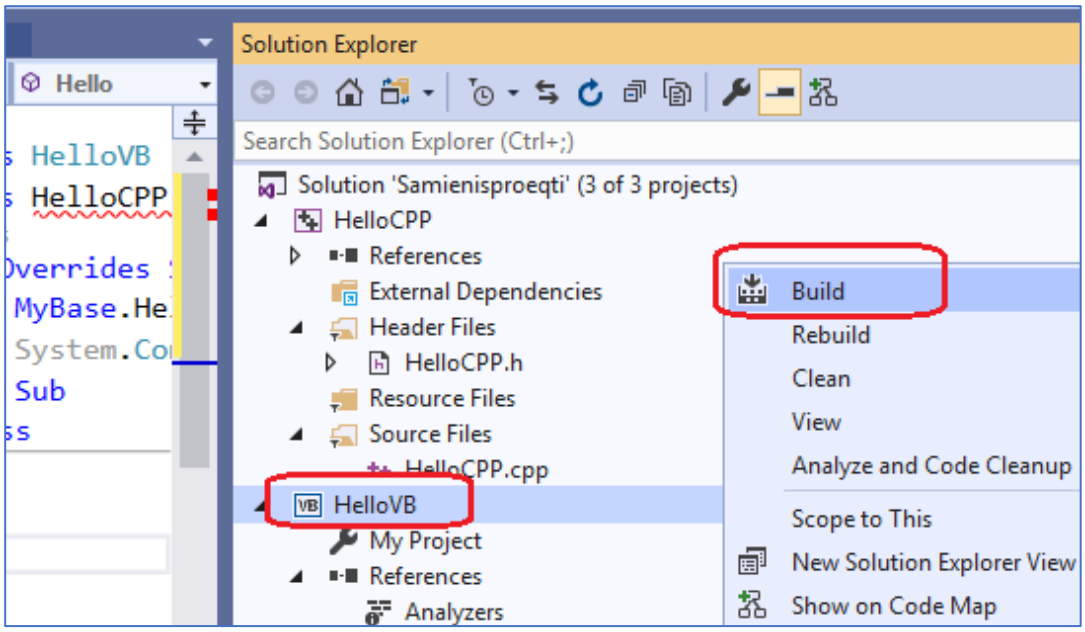
ნახ.5.15

ამის შემდეგ საჭიროა, ჯერ Solutio Explorer ის ფანჯრიდან მოინიშნოს ჩანართი My Project და გააქტიურდეს (ორჯერ დაქლიქვით) მიღებული ფანჯრის Referance - ით გაიხნება 5.16 ნახაზის მსგავსი ფანჯარა, რომელშიც Add→Reference ღილაკით ვამატებთ HelloCPP და ვადასტურებთ პროექტის დამატებას.



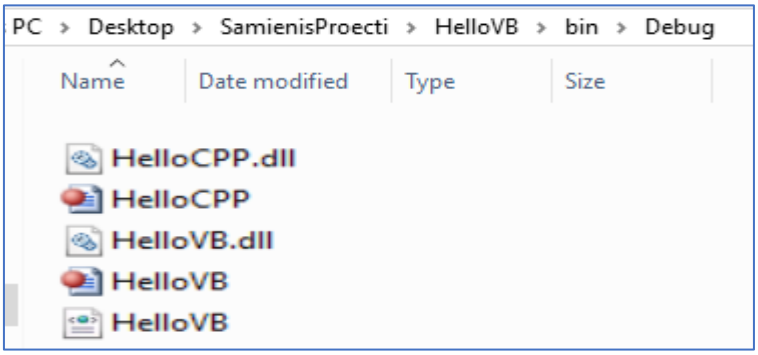
ნახ. 5.16

ამის შემდეგ HelloVB პროექტზე მარჯვენა ღილაკით გამოვიტანოთ (ნახ. 5.17) კონტექსტური მენიუ და ავირჩიოთ Build (ეს პროექტი ტრანსლატორით ამუშავდება და სინტაქსური გამართვის შედეგს მოგვცემს).



ნახ. 5.17

როგორც წესი, თუ შეცდომები არაა HelloVB პროექტში, მაშინ მის შესაბამის ფოლდერის bin->Debug -ში უნდა გამოჩნდეს დაკავშირებული Helli.CPP.dll კლასის ფაილი (ნახ.5.18).



ნახ.5.18

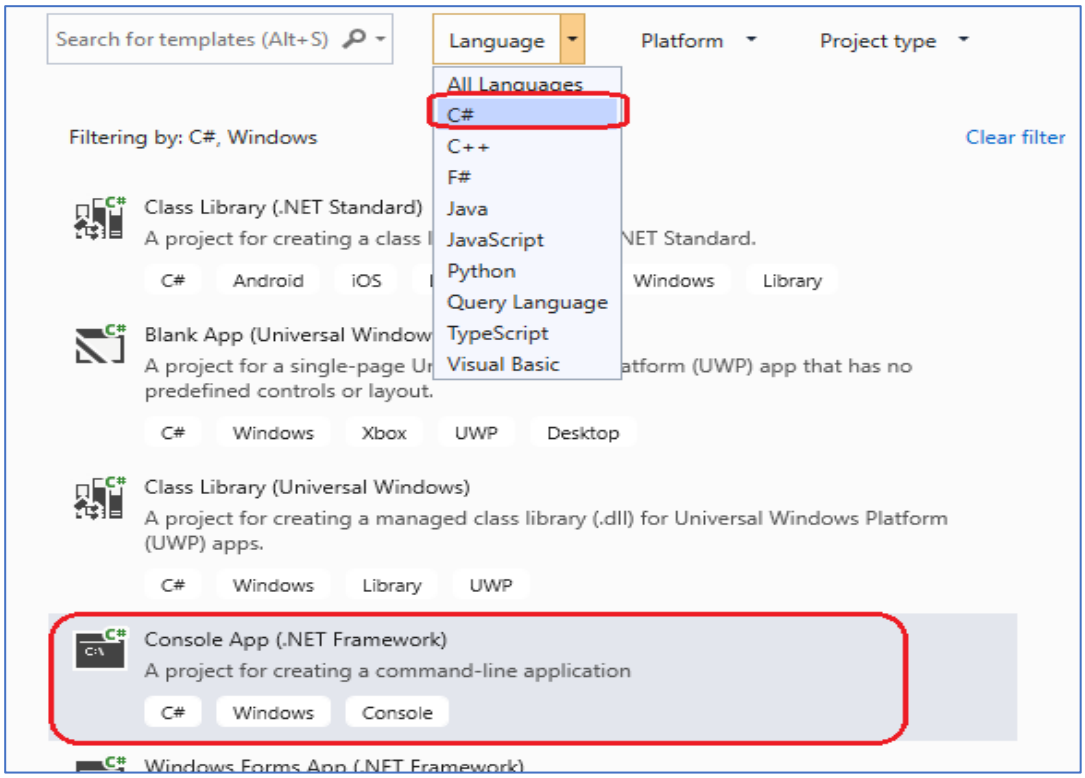
ამგვარად, Hello.CPP პროექტი მიეერთდება გადაწვეტას (Solution-ში) და გამოჩნდება მომავალში HelloVB-ში, ნახაზზე ნაჩვენები „ქვეშგახაზული“ შეცდომების აღნიშვნები HelloVB.vb პროგრამის ტექსტიდან გაქრა ! პროგრამის საბოლოო ტექსტი მოცემულია ლისტინგში:

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

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

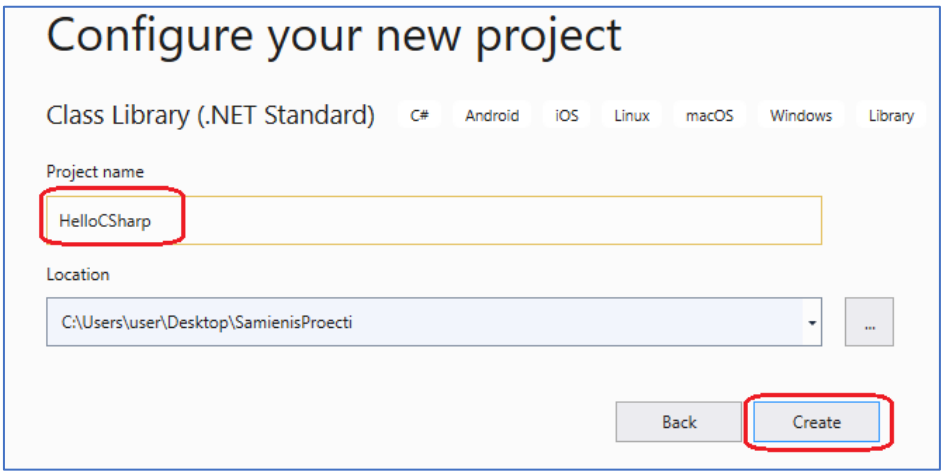
დასკვნით ფაზაში ჩვენი გადაწყვეტისათვის (Solution-ში) SamiEnisProjecti უნდა შევქმნათ მესამე, მთავარი სასტარტო პროექტი , რომელიც აგებული იქნება C#.NET ენაზე Console Application შაბლონის სახით. დავარქვათ მას HelloCSharp. ამგვარად, ვამატებთ პროექტს (ნახ.5.19-ა,ბ,გ) `HelloCSharp` სახელით.

წინა შემთხვევების მსგავსად აქაც ვირჩევთ ახალ პროექტს და ენის C# არჩევის შემდეგ კლასების ბიბლიოთეკა შეგვიძლია ავირჩიოთ როგორც უახლესი მოდელი - Universal Windows ასევე -Console Application (ჩვენს შემთხვევაში ვირჩევთ მეორეს - ა).



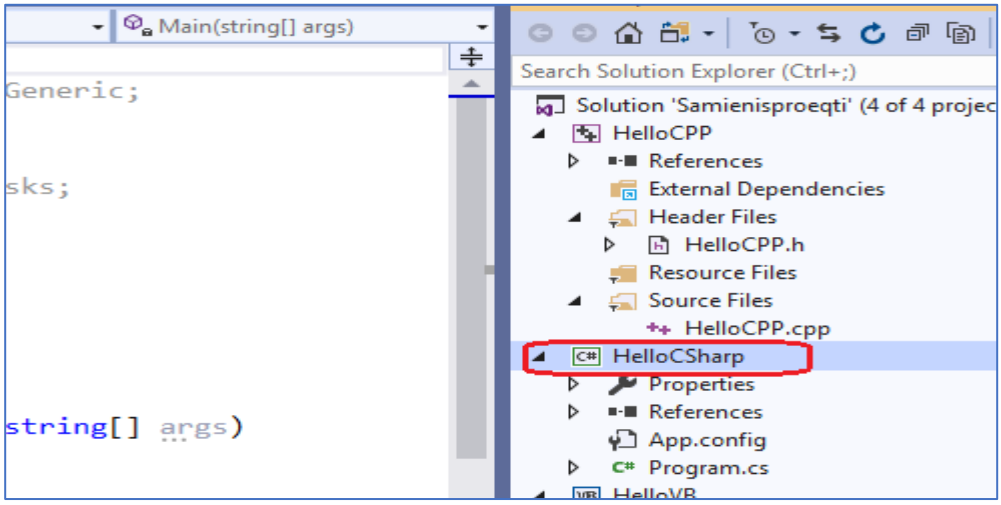
ნახ.5.19-ა

შედეგად მიიღება ფანჯარა, სადაც პროექტს ვარქმევთ სახელს HelloCSharp და ვადასტურებთ Create ღილაკით **ნახ.5.18-ბ:**



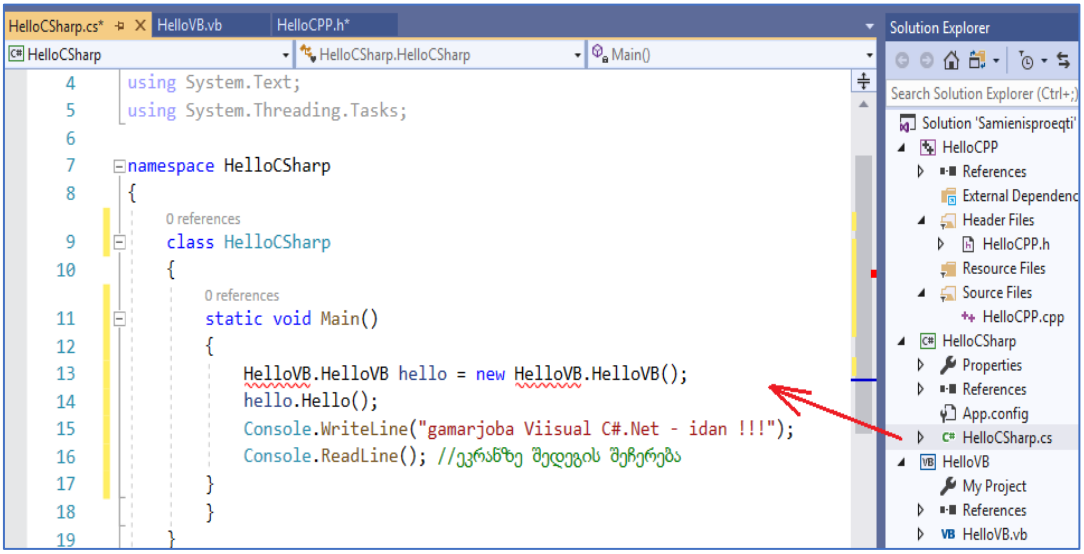
ნახ.5.19-ბ:

Solution - ფანჯარას დაემატება ახალი HelloCSharp პროექტი, თავისი შემადგენელი ყველა კომპონენტით. ნახ.5.19-გ.



ნახ.5.19-გ

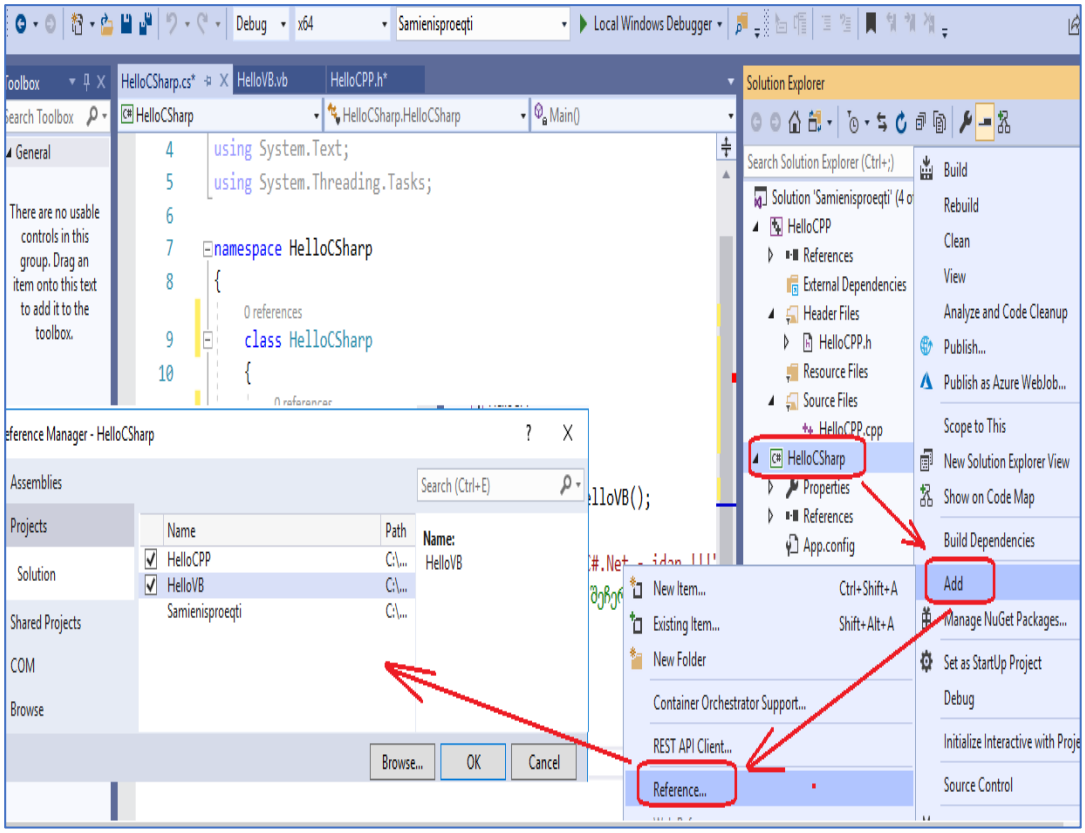
გადავარქვათ სახელი Program.cs ფაილს ჩვენი პროექტის შინაარსის შესაბამისად: HelloCSharp.cs. გამოვიტანოთ ეს ფაილი რედაქტირების ფანჯარაში და ჩავამატოთ Main()-ში შესაბამისი სტრიქონები. ასევე შეიძლება Main() -ის არგუმენტების წაშლა, აქ არ გვჭირდება. რედაქტირების შემდეგ ტექსტი მოცემულია 5.20 ნახაზზე.



ნახ.5.20

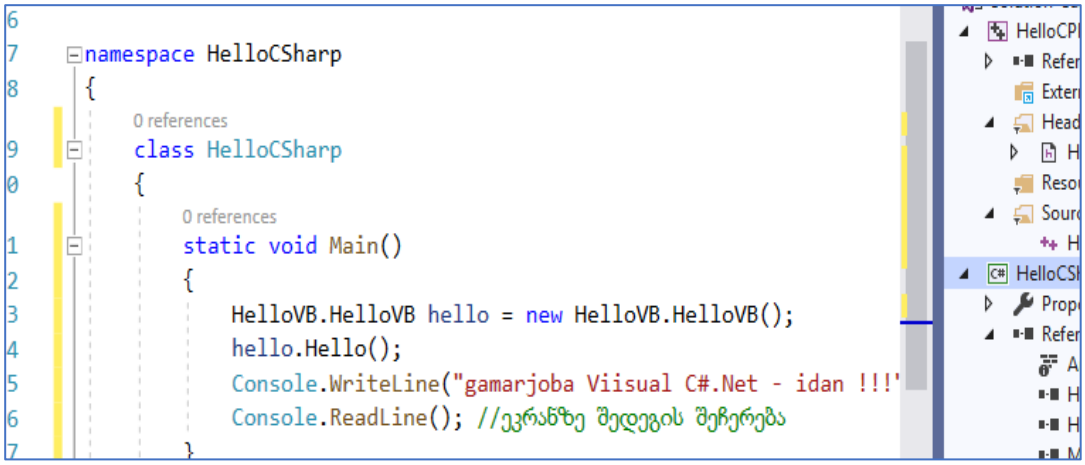
პროგრამაში Main() სტატიკური მეთოდი არის აპლიკაციაში შესვლის წერტილი. ეს მეთოდი ქმნის HelloVB კლასის ახალ ობიექტს hello და მისთვის იძახებს Hello() მეთოდს, რომელშიც ინახება ორი შეტყობინება. ერთი CPP.NET-იდან, მეორე VB.BET- დან, რომლებიც ადრე მოვამზადეთ. მესამე შეტყობინებას თვით C#.NET გამოიტანს, დამშვიდობებასთან ერთად. ახლა საჭიროა HelloCSharp პროექტში ჩავამატოთ მიმთითებლები (კავშირები) HelloCPP და HelloVB პროექტებზე. მაშინ 5.20 ნახაზზე ნაჩვენები შეცდომები (ქვეშხაზგასმული HelloVB) გასწორდება.

ამისათვის ვირჩევთ Reference…-ს და ნახაზზე მოყვანილი ნიმუშის შესაბამისად ვააქტიურებთ მითითებებს (ნახ.5.21).



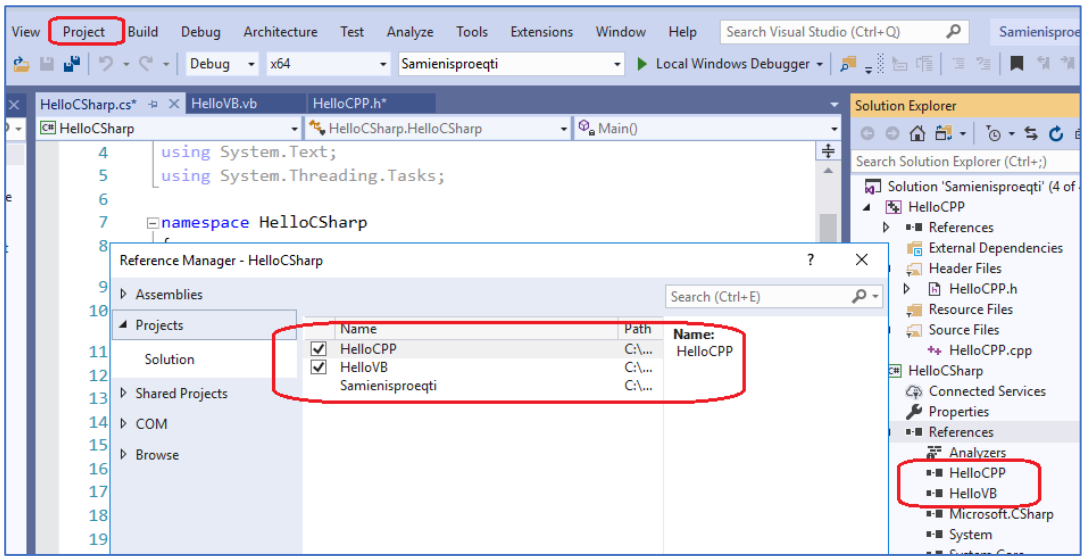
ნახ.5.21

HelloCPP და HelloVB პროექტებთან კავშირების შემდეგ, პროგრამულ კოდში სინტაქსური შეცდომის აღმნიშვნელი ინდიკატორები გაქრება ნახ.5.22.



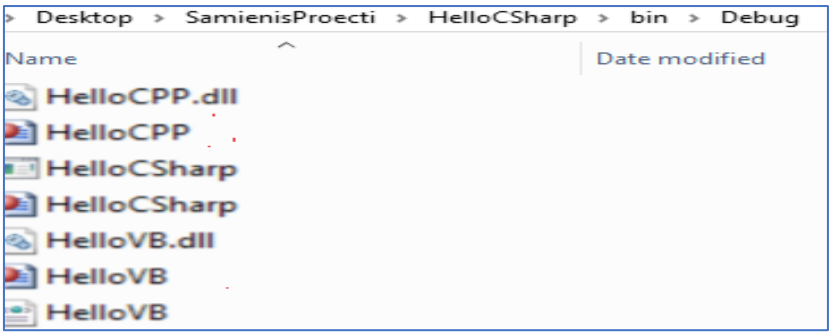
ნახ.5.22

Project-ჩანართიდან ,Add Reference ფანჯარაში გადასვლის შემდეგ მონიშნული HelloCPP და HelloVB პროექტები მიუთითებს იმაზე, რომ კავშირი პროექტებს შორის წარმატებით შესრულდა. ეს შედეგი აისახება Solution-ფანჯარაში HelloCSharp პროექტის References –ში (ნახ.5.23).



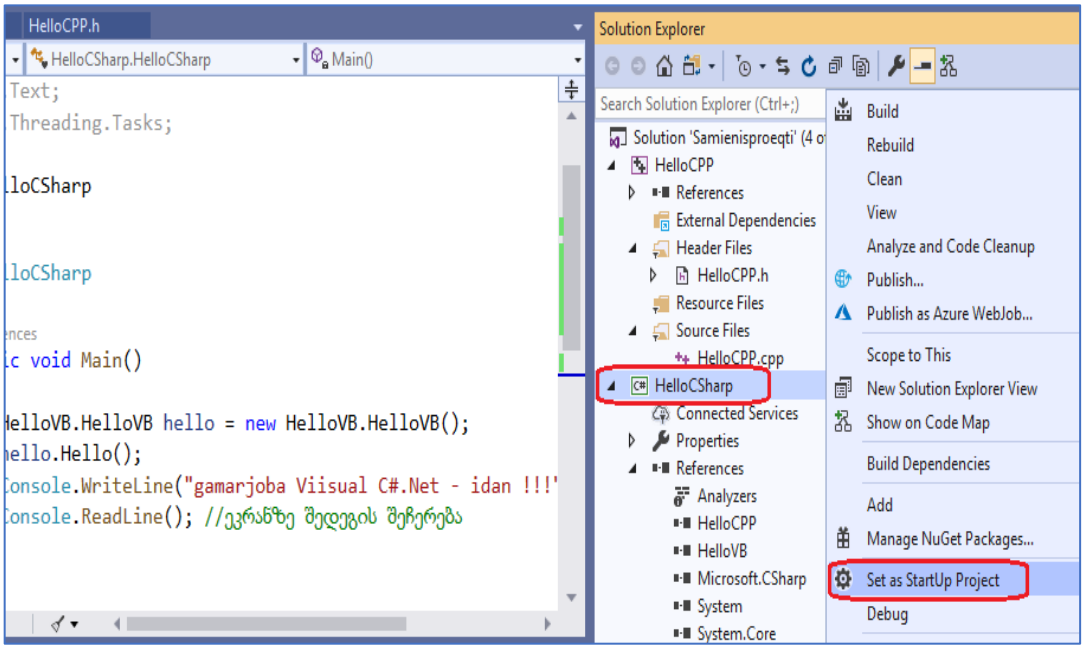
ნახ.5.23

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.5.24). აქვეა HelloCSharp.exe აპლიკაციის ფაილიც.



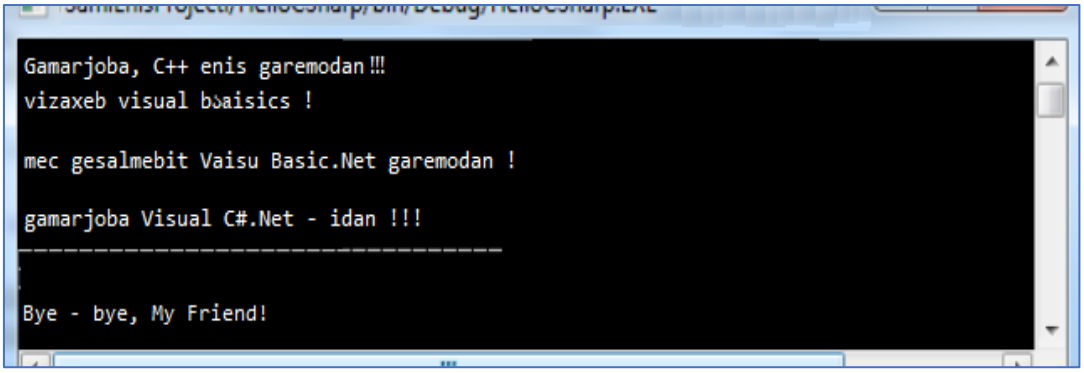
ნახ.5.24

Solution-ში HelloCSharp პროექტი გადავაკეთოთ სასტარტო ფაილად (ნახ.5.25).



ნახ.5.25

ბოლოს, ავამუშავოთ აპლიკაცია და მივიღებთ შედეგს (ნახ.5.26).



ნახ.5.26

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.

დავალება: ააგეთ განხილული პროგრამული პროექტის კოდი და გააანალიზეთ მისი მუშაობის შედეგები..

5.2. ინფორმაციული სისტემის კლასებისა და ობიექტების დაპროგრამება მართვის ამოცანის მაგალითზე

მიზანი: ობიექტორიენტირებული დაპროგრამების ინკაფსულაციის თვისების დეტალიზებული შესწავლა. კლასის განსაზღვრა როგორც მონაცემებისა და მეთოდების ერთობლიობა და მათი პროგრამული რეალიზაციის განხორციელება. როგორ შევქმნათ კლასები, მათი ობიექტები, მეთოდები და კლასთაშორის კავშირები კონსოლისა და ვინდოუს_ფორმების პროექტებში ?

განვიხილოთ აღნიშნული საკითხები პრაქტიკული მართვის ამოცანის საფუძველზე, მათი შემდგომი განზოგადების მიზნით.

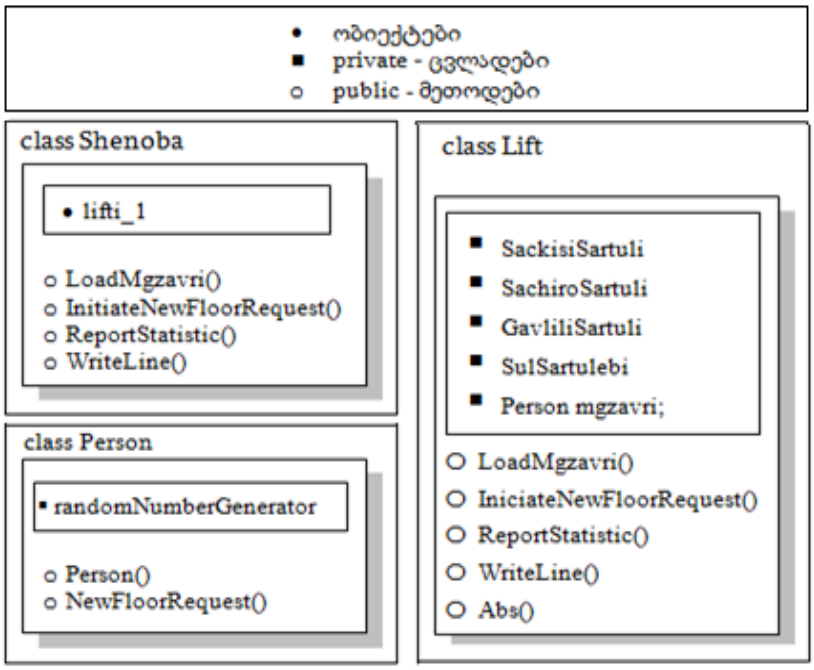
ამოცანა_1: ავაგოთ ობიექტორიენტირებული პროგრამის კოდი (კლასებისა და მეთოდების გამოყენებით) მაღლივ შენობაში ლიფტის გადაადგილების მოდელირებისათვის.

მაგალითად, შენობა N სართულიანია. მას აქვს ლიფტი. პერსონა, რომელიც შედის ლიფტში (ხდება მგზავრი), ირჩევს მისთვის საჭირო სართულსა და მიემგზავრება (ზევით ან ქვევით). საჭიროა ამ პროცესის მოდელირება და დაპროგრამება ისე, რომ დაფიქსირდეს ლიფტის საწყისი მდგომარეობა, ამუშავების შემდეგ მისი საბოლოო მდგომარეობა, გავლილი სართულების რაოდენობა (ერთი ამუშავებისას). საბოლოოდ გამოიცეს რეპორტი, თუ სულ რამდენი სართული გაიარა ლიფტმა ერთი სეანსის (სათანადო პერიოდის) განმავლობაში.

3.1) კლასთა მოდელი და მათი აგების პირობები:

ინკაფსულაციის სქემა მოცემულია 5.27 ნახაზზე.

- პროგრამაში სამი კლასია: Shenoba, Lift და Person;
- Shenoba კლასს აქვს Lift კლასის ერთი ობიექტი, სახელით lifti_1 ;
- Person კლასის ერთი ობიექტი იმყოფება mgzavri - ცვლადის შიგნით, რომელიც იყენებს lifti_1;
- Lift კლასის ობიექტს შეუძლია გადაადგილება ნებისმიერ სართულზე, ინტერვალში [1,N=60].



ნახ.5.27

- პროგრამაში სართულის არჩევა ხდება მგზავრის მიერ (გამოიყენება „შემთხვევით რიცხვთა გენერატორი“ Random-ფუნქციით);

- lifti_1 ლიფტი მოძრაობს საჭირო სართულისაკენ, რაც აისახება კონსოლზე;

- მგზავრ(ებ)ის ლიფტით მოძრაობის სეანსი შედგება რამდენიმე ეტაპისაგან, რომლბზეც შეირჩევა ახალი მიზნობრივი სართულები;

- სეანსის ბოლოს გამოიცემა ანგარიშის ტექსტი (რეპორტი), თუ ჯამში რამდენი სართული გაიარა ლიფტმა;

- შუალედური და საბოლოო შედეგები გამოიტანება ეკრანზე.

3.2) კლასთა კავშირების აღწერა UML ტექნოლოგიით:

მომხმარებლის სამი კლასი: Shenoba, Lift, Person და ერთი სისტემური კლასი System.Random ამოდელირებს ლიფტის მუშაობის პროცესს:

- Shenoba-კლასი შეიცავს Lift-ობიექტსა და იძახებს მის მეთოდებს;

- Lift-ობიექტი იყენებს Person-ობიექტს, რათა მართოს ლიფტის მოძრაობა;

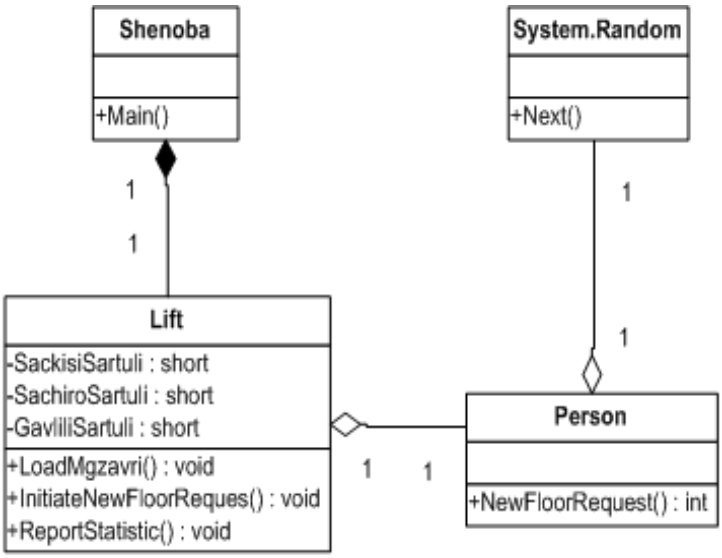
- Person-ობიექტი კი იყენებს System.Random-ობიექტს, რათა აირჩიოს საჭირო სართული შემდგომი გადაადგილებისათვის.

ობიექტ - ორიენტირებული პროგრამული სისტემების დაპროექტებისას კლასებმა მსგავსი ფუნქციები უნდა შეასრულოს. თითოეულს უნდა ჰქონდეს თავისი უნიკალური ფუნქციონალობა, და ყველა ერთად უნდა უზრუნველყოფდეს მთლიანი პროგრამის მუშაობას.

ნებისმიერი მაღლივი შენობა, როგორც „მთელი“ შედგება სხვადასხვა ნაწილისაგან: იატაკი, კედლები, ფანჯრები, ლიფტები და ა.შ. ამგვარად, შენობასა და ლიფტს შორის არსებობს დამოკიდებულება „მთელი-ნაწილი“ (აგრეგატული კავშირი UML ენაზე). ამიტომაცაა, რომ Lift-კლასის ობიექტის ცვლადი გამოცხადებულია Shenoba-კლასის შიგნით (ნახ.3.1, lifti_1). პროგრამულად იგი რეალიზებულია სტატიკური ცვლადის სახით (47-ე სტრიქონი, ლისტინგი_3.1): private static Lift lifti_1;

იგი ინახავს Lift კლასის ობიექტსა და შეუძლია მისი მეთოდების გამოძახება. გარდა ამისა, Shenoba-კლასს შეიძლება ჰქონდეს აგრეთვე სხვა ცვლადებიც, რომლებიც აგრეგატულად დაქვემდებარებული კლასების ობიექტების ცვლადები იქნება.

5.28 ნახაზზე მოცემულია ჩვენი მაგალითის კლასებს შორის კავშირების UML დიაგრამა, აგებული Ms Visio პაკეტის გარემოში [9,12].

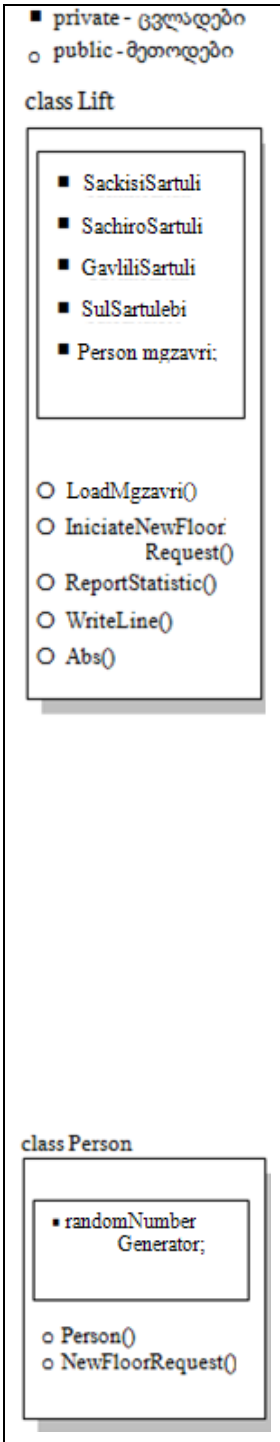


ნახ.5.28. UML-ის კლასების დიაგრამა

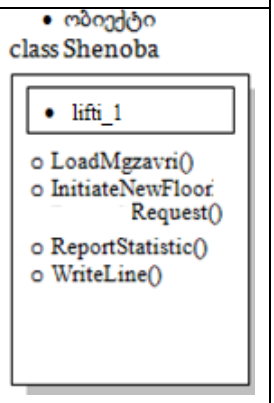
Shenoba და Lift კლასებს შორის აგრეგაციას უწოდებენ კომპოზიციასა და იგი შავი რომბიკით გამოისახება. „1“-ები ნიშნავს, რომ 1 შენობაში არის 1 ლიფტი (ჩვენს შემთხვევაში). Lift და Person, აგრეთვე Person და System.Random კლასებს შორის აგრეგატული დამოკიდებულებაა, მაგრამ არა-კომპოზიციური. ის პატარა რომბითაა ნაჩვენები. განსხვავება ისაა, რომ შენობას ლიფტი ყოველთვის აქვს. ლიფტში კი პერსონა შეიძლება იყოს ან არ იყოს, ლიფტი ისეც მუშაობს.

3.3) პროგრამული რეალიზაცია:

5.1 ლისტინგში მოცემულია განხილული კლასების პროგრამული რეალიზაციის C# კოდის ტექსტი..




```
1  using System;
2  namespace ConsoleApp_Lift
3  {
4    class Lift
5    {
6      private int SackisiSartuli = 1;
7      private int SachiroSartuli = 0;
8      private int GavliliSartuli = 0;
9      private int SulSartulebi = 0;
10     public  int i = 1;
11     private Person mgzavri;
12   public void LoadMgzavri()
13     {
14       mgzavri = new Person();
15     }
16   public void InitiateNewFloorRequest()
17   {
18     SachiroSartuli = mgzavri.NewFloorRequest();
19     GavliliSartuli = Math.Abs(SackisiSartuli -
                                    SachiroSartuli);
20     Console.WriteLine("{0,2}.| Sackisi: {1,2}|
         Sachiro: {2,2}| Gavlili: {3,2} ",  i.ToString(),
         SackisiSartuli.ToString(),
         SachiroSartuli.ToString(),
         GavliliSartuli.ToString());
21   // Math.Abs(SackisiSartuli - SachiroSartuli);
22     SulSartulebi += GavliliSartuli;
23      SackisiSartuli = SachiroSartuli;
24      i++;
25    }
26   public void ReportStatistic()
27   {   Console.WriteLine("\n===>>> Sul gavlili
28                          sartulebi: " + SulSartulebi);
29     }
30    }
31
32 class Person
33   {
       private System.Random
34                      randomNumberGenerator;
35     public Person()   // კონსტრუქტორი
36     {
         randomNumberGenerator = new
37                                System.Random();
38       }
39
40     public int NewFloorRequest()
41     {
```




```
42          // აბრუნებს არჩეულ შემთხვევით
43          //  რიცხვს 1-60 დიპაზონში
44          return randomNumberGenerator.Next(1,60);
            }
 }
45
46  class Shenoba   // კლასი შენობა
47  {
48    private static Lift lifti_1;
      private static void Main()
49     {
50       lifti_1 = new Lift();
51       lifti_1.LoadMgzavri();
52      Console.WriteLine("      samgzavro sartulebi \n
53        ============================\n");
54       lifti_1.InitiateNewFloorRequest();
55       lifti_1.InitiateNewFloorRequest();
56       lifti_1.InitiateNewFloorRequest();
57       lifti_1.InitiateNewFloorRequest();
58       lifti_1.InitiateNewFloorRequest();
59       lifti_1.ReportStatistic();
60     }
  }}
```

ლისტინგი_5.1: კონსოლის რეჟიმი

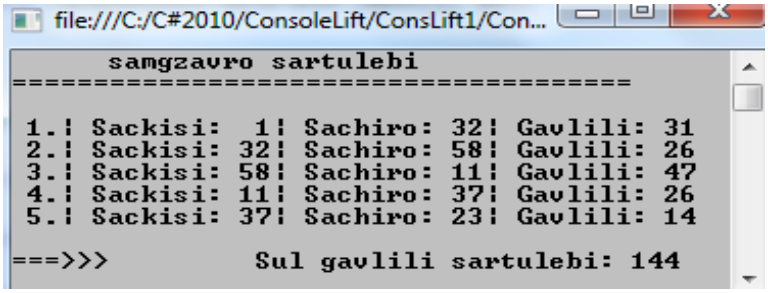
3.4) პროგრამის ლისტინგის ანალიზი

N	დანიშნულება
4	Lift - კლასის განსაზღვრების დასაწყისი
6	SackisiSartuli - ცვლადის გამოცხადება **int**-ტიპით, private-წვდომის სპეციფიკატორით და საწყისი მნიშვნელობით=1
11	Mgzavri - ცვლადის გამოცხადება, რომელიც შეიცავს Person- კლასის ობიექტს. Person- კლასი ასრულებს მგზავრის როლს Lift-კლასთან მიმართებით
12	LoadMgzavri() - მეთოდის განსაზღვრების დასაწყისი. ის არის Lift-კლასის ინტერფეისის ნაწილი და გამოცხადებულია როგორც public
14	Person-კლასის ახალი (**new**) ობიექტის შექმნა. ეს ობიექტი მიენიჭება ცვლადს - mgzavri
16	InitiateNewFloorRequest() - მეთოდის განსაზღვრების დასაწყისი. ის არის Lift-კლასის ინტერფეისის ნაწილი და გამოცხადებულია როგორც public
18	mgzavri-ობიექტისთვის NewFloorRequest() მეთოდის გამოძახება. ამ მეთოდით დაბრუნებული მნიშვნელობა მიენიჭება ცვლადს SachiroSartuli
19	ერთი მგზავრობის შემდეგ იანგარიშება გავლილი სართულების რაოდენობა

20	ეკრანზე გამოიტანება: საწყისი_სართული, საჭირო_სართული, გავლილი_სართულების_რაოდენობა
21	იანგარიშება სულ გავლილი სართულების რაოდენობა ლიფტის მუშაობის მთელი სეანსის დროს
22	ლიფტის საწყისი სართულის მნიშვნელობას ენიჭება მისი ბოლო გაჩერების სართულის მნიშვნელობა
23	ლიფტის ამუშავების მომდევნო ბიჯის ინკრემენტი
26	ReportStatistic() მეთოდის გამოძახებით ეკრანზე გამოიტანება სტატისტიკა SulSartulebi ცვლადით
31	Person კლასის განსაზღვრების დასაწყისი
33	randomNumberGenerator ცვლადის გამოცხადება System.Random კლასის ობიექტის შესანახად
34	სპეციალური მეთოდის (კონსტრუქტორის !) განსაზღვრების დასაწყისი, რომელიც გამოიძახება ავტომატურად Person კლასის ობიექტის შექმნის დროს
36	System.Random-კლასის ახალი ობიექტის შექმნა და მისი მინიჭება randomNumberGenerator - ცვლადზე
39	int ტიპის NewFloorRequest() მეთოდის განსაზღვრა. ის Person-კლასის ინტერფეისის ნაწილია
42	პერსონა (ვირტუალური მგზავრი) ლიფტში ირჩევს საჭირო სართულს (შემთხვევით რიცხვთა გენერატორი ასრულებს ამ ფუნქციას) დიაპაზონში [1-60]
45	Shenoba კლასის აღწერა
47	Shenoba კლასში გამოცხადებულია Lifti ტიპის ცვლადი Lifti_1. Shenoba კლასი კომპოზიციურ კავშირშია Lift კლასთან (ნახ.3.2)
48	Main( ) მეთოდის აღწერის დასაწყისი
49	Lifti კლასის ახალი ობიექტის შექმნა და მისი მინიჭება lifti_1 ცვლადზე
50	lifti_1 ობიექტისთვის LoadMgzavri( ) მეთოდის გამოძახება
51-56	lifti_1 ობიექტისთვის .IniciateNewFloorRequest( ) მეთოდის გამოძახება 5-ჯერ
57	lifti_1 ობიექტისთვის . ReportStatistic() მეთოდის გამოძახება (შედეგების გამოსაცემად ეკრანზე)

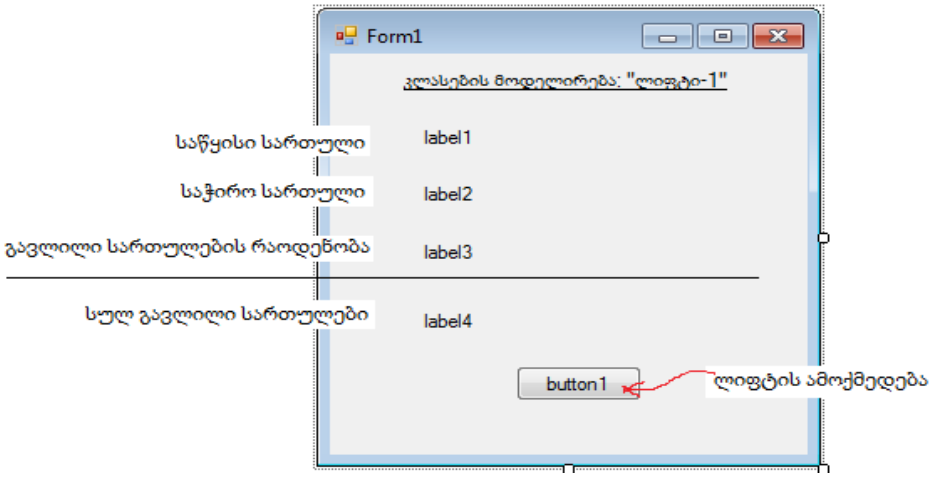
3.5) პროგრამის მუშაობის შედეგები:

5.3 ნახაზზე ნაჩვენებია განხილული პროგრამის შესრულების შედეგები კონსოლის რეჟიმში.



ნახ.5.30

ამოცანა_3.2: განხილული ამოცანისათვის ავაგოთ პროგრამული კოდი კლასების საფუძველზე ვინდოუსის ფორმის რეჟიმში. 5.31 ნახაზზე ნაჩვენებია სამუშაო ფანჯარა, რომელიც Form კლასის Form1 ობიექტია. მასზე განთავსებულია ოთხი label (1,2,3,4) და ერთი button1, რომლითაც მოდელირდება ლიფტის ამოქმედება (ლიფტის გამოძახება, როცა პერსონა გარეთაა ან სართულის არჩევა ლიფტის ღილაკით, როცა პერსონა ლიფტშია, ანუ მგზავრია).



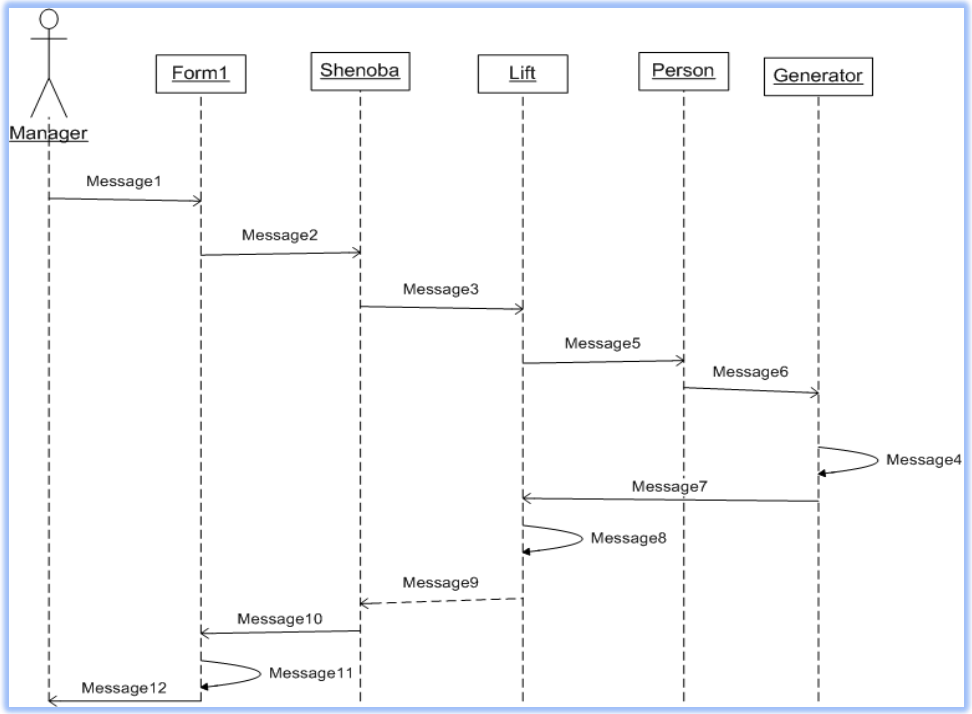
ნახ.5.31

```
// ლისტინგი_5.2--Form1 ელემენტებით და button1-ის კოდი ---
using System;
using System.Drawing;
using System.Windows.Forms;
```

```
namespace WinFormLift
{
   public partial class Form1 : Form
   {
      Shenoba shenoba_1;    // კლასი Shenoba  უნდა შეიქმნას
      public Form1() { InitializeComponent();   }
      private void button1_Click(object sender, EventArgs e)
      {   // shenoba_1  ობიექტის შექმნა
         shenoba_1 = new Shenoba(label1, label2, label3, label4);
      }
   }
}
```

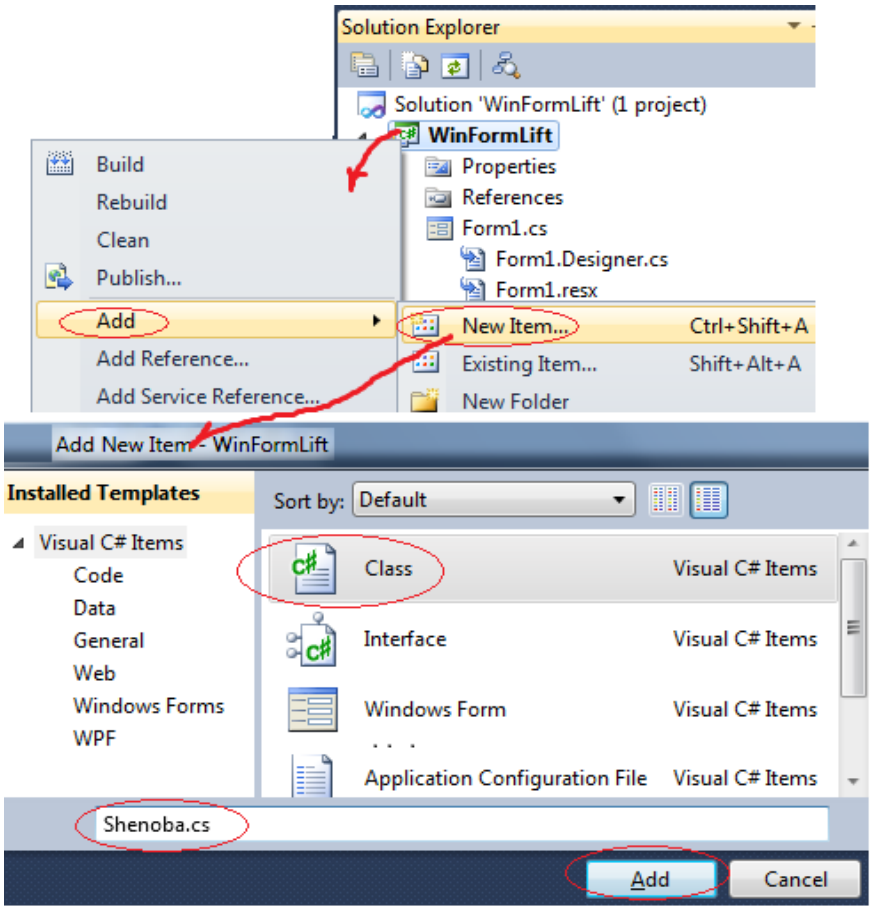
გარდა Form კლასისა (Form1 ობიექტით), რომელიც ასრულებს პროგრამის მომხმარებლის ინტერფეისის ფუნქციას, ლიფტის მუშაობის პროცესის ობიექტ - ორიენტირებული მოდელის ასაგებად საჭიროა კლასები: Shenoba, Lift და Person.

ამ კლასების თვისებები და ფუნქციობა ჩვენ ზემოთ აღვწერეთ. ახლა საჭიროა ავაგოთ სცენარი „ლიფტის მუშაობა“, რომლისთვისაც გამოვიყენებთ UML-ის მიმდევრობითობის (Sequence) დიაგრამას (ნახ.5.32).



ნახ.5.32. UML-ის მიმდევრობითობის დიაგრამა

ახლა შევქმნათ დანარჩენი კლასები და აღვწეროთ მათი ფუნქციობები. Solution Explorer-ში დავამატოთ (Add new Items) ახალი კლასები, როგორც ეს 5.33 ნახაზზეა ნაჩვენები.



ნახ.5.33

Shenoba კლასის კოდი მოცემულია 5.3_ ლისტინგში.

```
// ლისტინგი_5.3 --- კლასი Shenoba --------
using System;
using System.Windows.Forms;
namespace WinFormLift
{
  public class Shenoba
  {
```

```
   static Lift lift_1 = new Lift();

  public Shenoba(Label label1, Label label2, Label
                                                label3, Label label4)
 {
  lift_1.LoadMgzavri();
    lift_1.InitiateNewFloorRequest(label1,label2, label3);
  lift_1.ReportStatistic(label4);
 }
 }
}
```

Lift კლასის პროგრამული კოდი მოცემულია 5.4_ლისტინგში.

// ლისტინგი_5.4 --- კლასი Lift --------

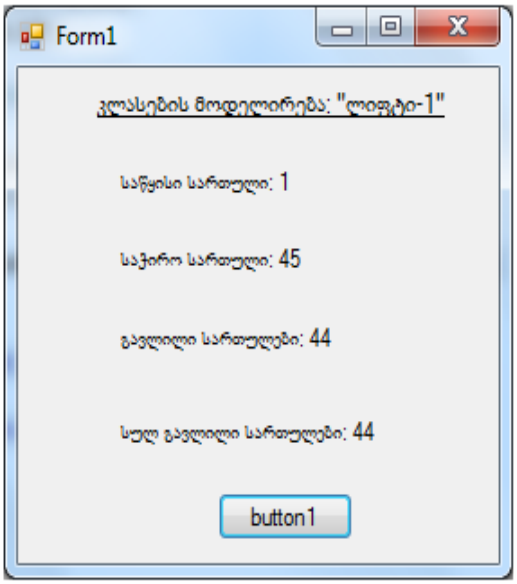
```
using System;
using System.Windows.Forms;
namespace WinFormLift
{
  public class Lift
   {
     private int SackisiSartuli = 1;
     private int SachiroSartuli = 0;
     private int GavlilSartulebi = 0;
     private int SulSartulebi = 0;
     public int i;
     private Person mgzavri;

   public void LoadMgzavri()
     {
        mgzavri = new Person();
     }
 public void InitiateNewFloorRequest(Label label1,
                                        Label label2, Label label3)
   {
     SachiroSartuli = mgzavri.NewFloorRequest();
     GavlilSartulebi = Math.Abs(SackisiSartuli - SachiroSartuli);
     label1.Text = "საწყისი სართული: " + SackisiSartuli.ToString();
     label2.Text = "საჭირო სართული: " + SachiroSartuli.ToString();
```

```
    label3.Text = "გავლილი სართულები: " + GavlilSartulebi.ToString();
    SulSartulebi += GavlilSartulebi;
    SackisiSartuli = SachiroSartuli;
    i++;
   }
 public void ReportStatistic(Label label4)
 {
   label4.Text = "სულ გავლილი სართულები: " + SulSartulebi;
} } }
```

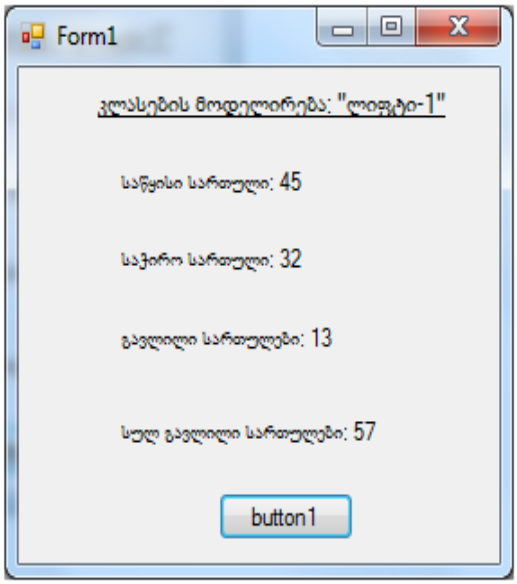
Person კლასის პროგრამა მოცემულია 5.5_ლისტინგში.

```
// ლისტინგი_5.5 --- კლასი Person  --------
using System;
using System.Windows.Forms;
namespace WinFormLift
{  public class Person
  {  private System.Random randomNumberGenerator;
     public Person()    // კონსტრუქტორი
      {
        randomNumberGenerator = new System.Random();  }
  public int NewFloorRequest()
   {  return randomNumberGenerator.Next(1, 60);
  }
 }
}
```

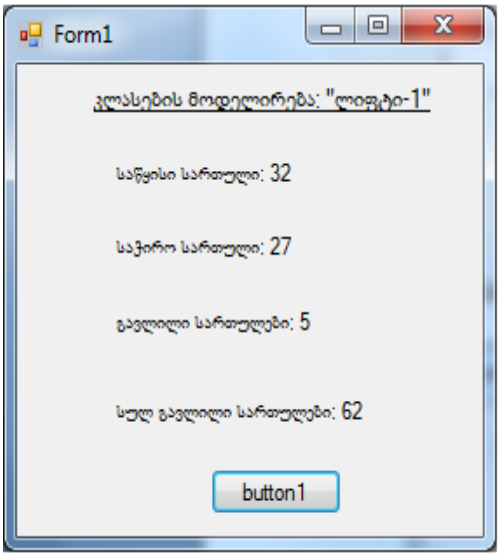
პროგრამის ამუშავების შემდეგ მიიღება 5.34 ნახაზზე ნაჩვენები შედეგები.



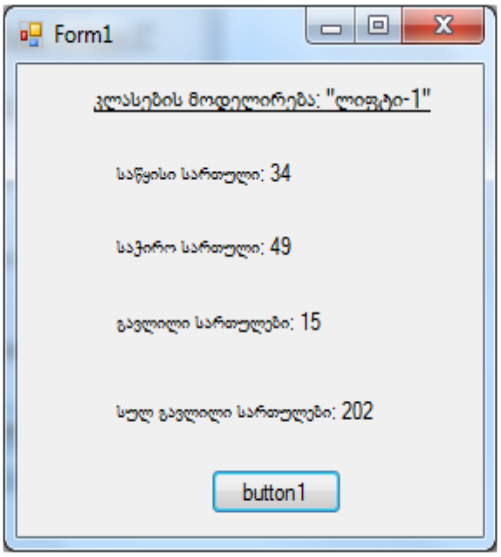
ნახ.5.34-ა. 1-ელი ბიჯი



ნახ.5.34-ბ. მე-2 ბიჯი



ნახ.5.34.-გ. მე-3 ბიჯი



ნახ.5.34-დ. მე-10 ბიჯი,

და ა.შ.

5.3. პროგრამული აპლიკაციების ტესტირება

მიზანი: პროგრამული კოდების ტესტირების პროცესის შესწავლა Visual Studio.NET ინტეგრირებულ გარემოში.

1. თეორიული ნაწილი

Unit testing და Coded UI არის Microsoft-ის ტესტირების ინსტრუმენტები, რომლებიც სრულდება Visual Studio.NET-გარემოში.

Unit testing - ანუ მოდულური ტესტირება დაპროგრამების პროცესია, რომლის საშუალებითაც მოწმდება საწყისი კოდის ცალკეული მოდულების კორექტულობა. ასეთი ტესტირების იდეა მდგომარეობს იმაში, რომ ყოველი არატრივიალური ფუნქციის ან მეთოდისათვის დაიწეროს ტესტი. ეს უზრუნველყოფს კოდის სწრაფად შემოწმებას, ხომ არ მიიყვანა კოდის ბოლო ცვლილებამ პროგრამა რეგრესიამდე ანუ შეცდომების გაჩენამდე პროგრამის უკვე ტესტირებულ ნაწილებში [11].

Coded UI ტესტი კი ავტომატურად იწერს, შესრულებაზე უშვებს და ამოწმებს ტესტ - ქეისებს.

ასეთი ტესტების წერა შესაძლებელია C# ან Visual Basic-ზე Visual Studio გარემოში. Unit testing ტესტირების ტექნოლოგია განვიხილოთ ვირტუალური ობიექტის, მაგალითად, ფინანსური ობიექტის, ბანკის მაგალითზე.

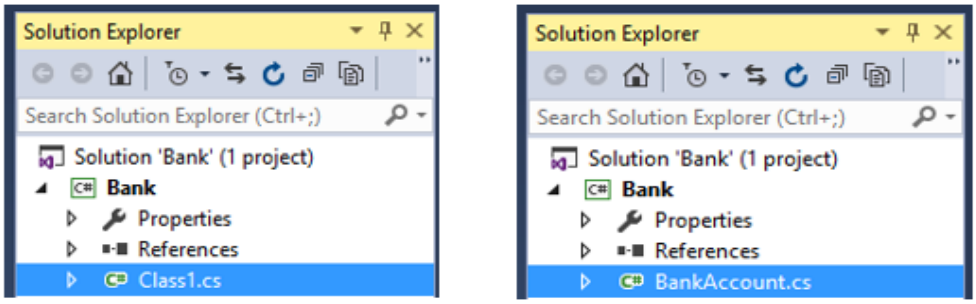
- დასატესტი პროგრამის პროექტის შექმნა: Visual Studio–ში საჭიროა ავირჩიოთ: **File -> New -> Project.**

შედეგად გამოჩნდება დიალოგური ფანჯარა, სადაც ავირჩევთ:

Visual C# => ClassLibrary

პროექტი სახელით Bank.

მიიღება 5.35 ნახაზზე ნაჩვენები Solution Ex[plorer ფანჯარა. აქ Class1.cs სახელი შევცვალოთ BankAccount.cs -ით.



ნახ.5.35. Class1 - > BankAccount

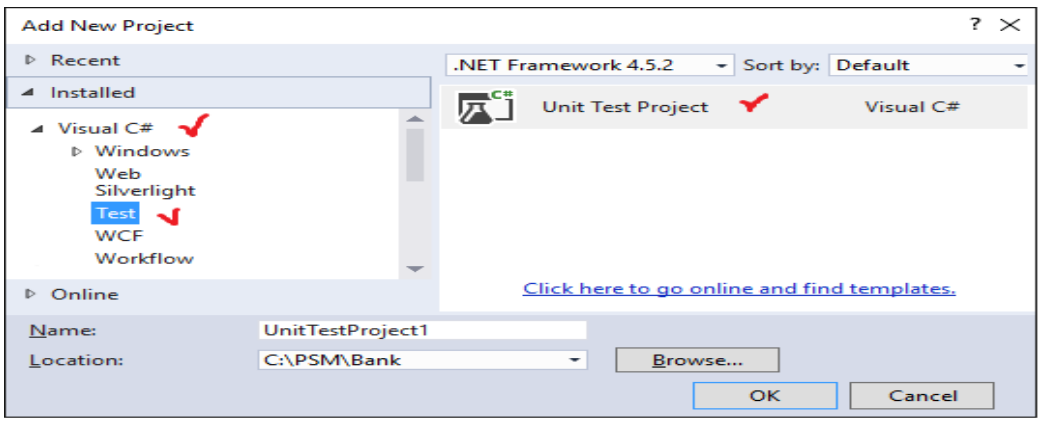
შემდეგ BankAccount.cs-ის ტექსტი რედაქტორის არეში შევცვალოთ ჩვენი დასატესტი პროგრამის კოდით.

ეს საწყისი ტექსტი, მაგალითად, მოცემულია 5.6 ლისტინგში.

```
//-- ლისტინგი_5.6 --- BankAccount.cs --------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BankAccountNS
{
 public class BankAccount
  {
    private string m_customerName;
    private double m_balance;
    private bool m_frozen = false;
   private BankAccount()
    {
    }
   public BankAccount(string customerName, double balance)
    {
      m_customerName = customerName;
      m_balance = balance;
    }
   public string CustomerName
    {
      get { return m_customerName; }
    }
   public double Balance
    {
      get { return m_balance; }
    }
   public void Debit(double amount)
    {
      if (m_frozen)
       {
         throw new Exception("Account frozen");
       }
      if (amount > m_balance)
       {
          throw new ArgumentOutOfRangeException("amount");
        }
       if (amount < 0)
        {
          throw new ArgumentOutOfRangeException("amount");
```

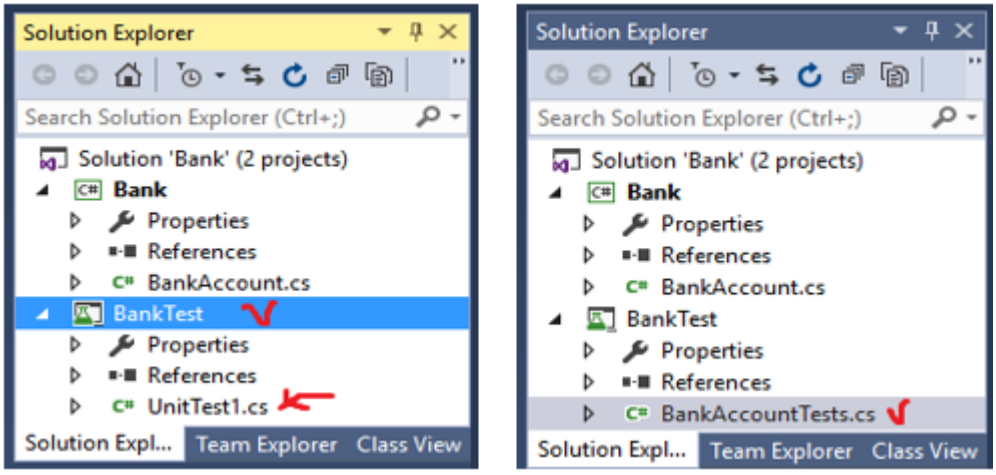
```
        }
        m_balance += amount; // განზრახ არასწორი კოდი
        // m_balance -= amount;   // გასწორებული
     }
    public void Credit(double amount)
     {
       if (m_frozen)
         {
          throw new Exception("Account frozen");
         }
       if (amount < 0)
        {
          throw new ArgumentOutOfRangeException("amount");
        }
        m_balance += amount;
     }
    private void FreezeAccount()
     {
        m_frozen = true;
     }
    private void UnfreezeAccount()
     {
        m_frozen = false;
     }
    public static void Main()
     {
      BankAccount ba = new BankAccount("Mr.Bryan Walton", 11.99);
      ba.Credit(5.77); ba.Debit(11.22);
      Console.WriteLine("Current balance is ${0}", ba.Balance);
     }
  }
}
```

- ტესტ-ფაილის პროექტის აგება (Unit Test Project):



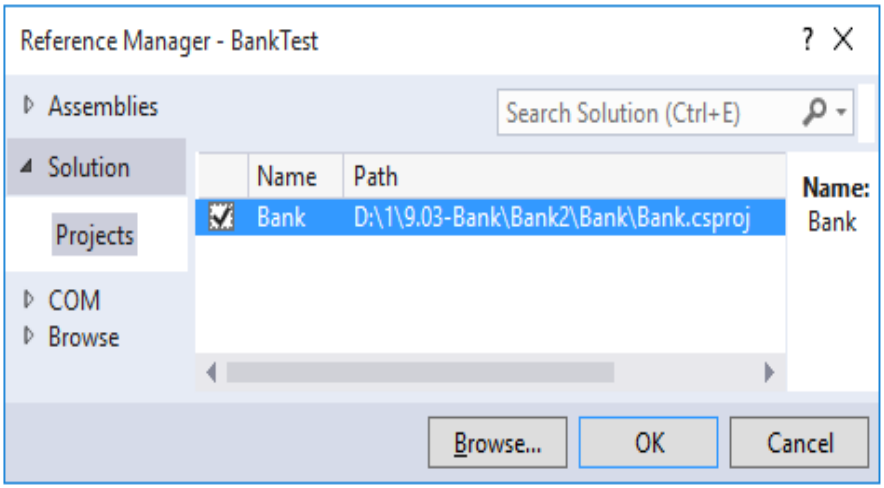
ნახ.5.36. Unit Test პროექტის შექმნა

მივიღებთ 5.37. ნახაზზე ნაჩვენებ სურათს BankTest პროექტით. მარჯვენა სურათზე შეცვლილია UnitTest კლასის სახელი BankAccountTests-ით.



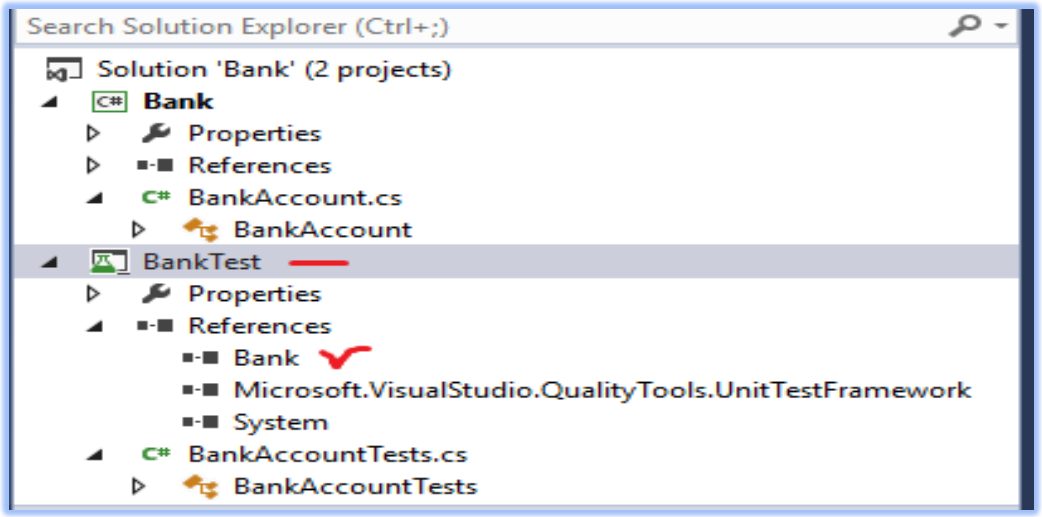
ნახ.5.37

BankTests პროექტში დავამატოთ reference **Bank** solution-იდან. ამისათვის **BankTests**-ზე მაუსის მარჯვენა ღილაკით ავირჩიოთ **Add Reference** და მივიღებთ 5.38 ნახაზზე ნაჩვენებ ფანჯარას. აქ Solution სტრიქონში ვირჩევთ Projects და Bank-ის ჩეკბოქსს მოვნიშნავთ.



ნახ.5.38

მივიღებთ 5.39 ნახაზზე ნაჩვენებ შედეგს.



ნახ.5.39

ჩავამატოთ BankAccountTests პროგრამაში სახელსივრცე Bank-ის პროექტიდან: `using BankAccountNS;`

ამგვარად, BankAccountTests.cs ფაილს ექნება 5.7 ლისტინგზე ნაჩვენები სახე.

```
//-- ლისტინგი_5.7 -----  BankAccountTests.cs -----
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;

namespace UnitTestProject1
{
   [TestClass]
   public class BankAccountTests
   {
      [TestMethod]
      public void TestMethod1()
      {
      }
   }
}
```

ახლა შევქმნათ პირველი ტესტ - მეთოდი. ამ პროცედურაში, დაიწერება უნიტ - ტესტის მეთოდები BankAccount class-ის Debit მეთოდის ქცევის ვერიფიკაციისათვის. ეს მეთოდები ზემოთაა ჩამოთვლილი.

დასატესტი მეთოდების ანალიზის გზით გაირკვა, რომ საჭიროა მინიმუმ სამი ქცევის შემოწმება:

1. მეთოდი ქმნის ArgumentOutOfRangeException - გამონაკლისს, თუ კრედიტის ჯამი გადააჭარბებს ბალანსს;
2. იგი ქმნის ArgumentOutOfRangeException - გამონაკლისს მაშინაც, როცა კრედიტის ზომა უარყოფითია;
3. თუ 1 და 2 პუნქტები წარმატებით დასრულდა, მაშინ მეთოდი ითვლის ჯამს ბალანსის ანგარიშიდან.

პირველ ტესტში შევამოწმოთ, რომ კრედიტის დასაშვები მნიშვნელობისათვის (როცა დადებითი მნიშვნელობისაა და ბალანსის ანგარიშზე ნაკლებია) ანგარიშიდან მოიხსნება საჭირო თანხა.

1. დავამატოთ BankAccountTests კლასს შემდეგი მეთოდი:

```
// unit test code -----
[TestMethod]
public void Debit_WithValidAmount_UpdatesBalance()
{
  // arrange
   double beginningBalance = 11.99;
   double debitAmount = 4.55;
   double expected = 7.44;
   BankAccount account = new BankAccount("Mr. Dito",
                    beginningBalance);
   // act
   account.Debit(debitAmount);
   // assert
   double actual = account.Balance;
   Assert.AreEqual(expected, actual, 0.001, "Account not debited
                correctly");
}
```

მეთოდი საკმაოდ მარტივია. ჩვენ ვქმნით ახალ BankAccount ობიექტს საწყისი ბალანსით და შემდეგ ვაკლებთ სწორ ოდენობას. ჩვენ ვიყენებთ Microsoft-ის unit-ტესტის ფრეიმვორკს მართვადი კოდის AreEqual მეთოდისათვის, რათა მოხდეს საბოლოო ბალანსის ვერიფიკაცია - არის ის, რასაც ჩვენ ველით.

ტესტ - მეთოდის მოთხოვნები ასეთია:

1. მეთოდი მონიშნული უნდა იყოს [TestMethod] ატრიბუტით;
2. მეთოდმა უნდა დააბრუნოს void;
3. მეთოდს არ შეიძლება ჰქონდეს პარამეტრები.

ტესტის აგება და ამუშავება:

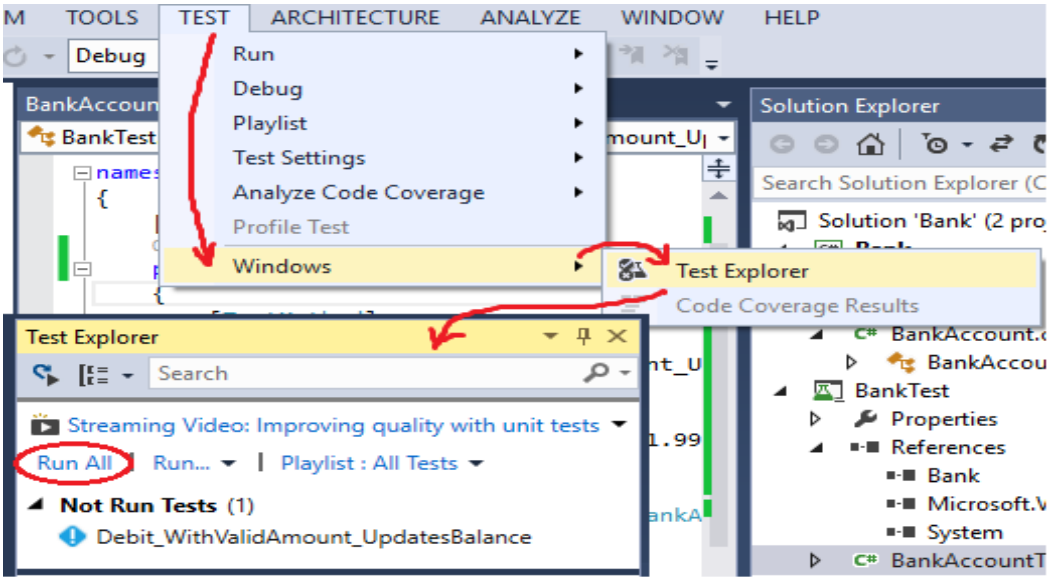
მთლიანი ტესტის კოდი მოცემულია 5.8 ლისტინგში.

//-- ლისტინგი_5.8 ----- BankAccountyTests.cs ----

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;

namespace UnitTestProject1
{
  [TestClass]
  public class BankAccountTests
  {
     [TestMethod]
     public void Debit_WithValidAmount_UpdatesBalance()
     {
       // arrange
       double beginningBalance = 11.99;
       double debitAmount = 4.55;
       double expected = 7.44;
       BankAccount account = new BankAccount("Mr. Dito",
                   beginningBalance);
       // act
       account.Debit(debitAmount);

       // assert
       double actual = account.Balance;
       Assert.AreEqual(expected, actual, 0.001, "Account
              not debited correctly");
     }
  }
}
```

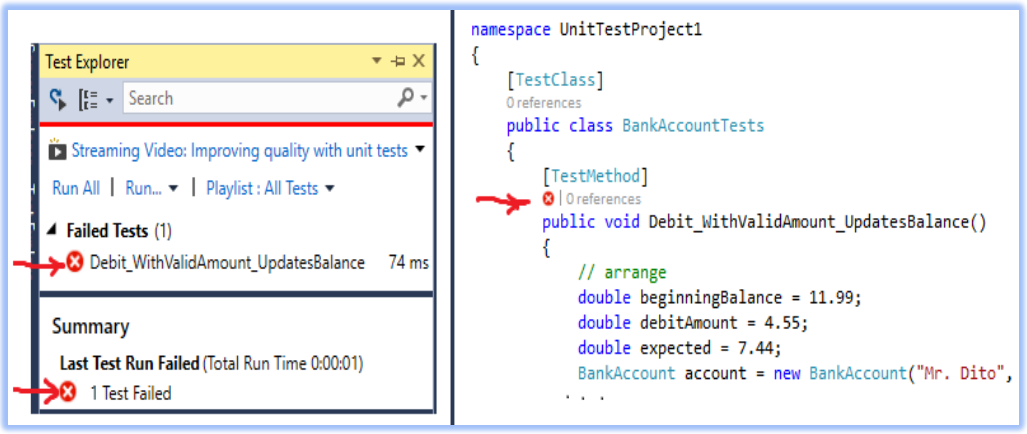
- BUILD მენიუდან ვირჩევთ Build Solution;

- TEST მენიუდან ვირჩევთ Windows და Test Explorer პუნქტებს. იხსნება Test Explorer ფანჯარა (ნახ.5.40).



ნახ.5.40.

აქ ვირჩევთ Run All - სა და ვიღებთ შედეგს (ნახ.5.41).



ნახ.5.41

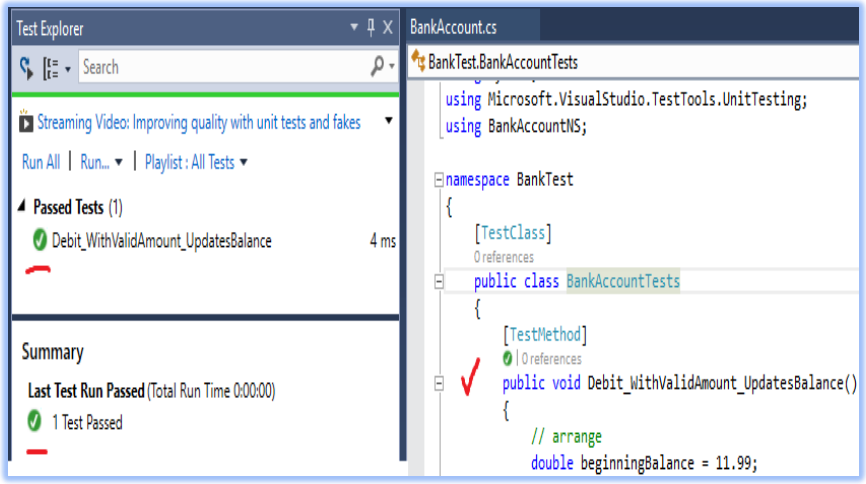
ნახაზზე მითითებული „x“-სიმბოლოები წითელ წრეშია მოთავსებული, ე.ი. ტესტირებამ აღმოაჩინა შეცდომები და კოდი წარმატებით ვერ შესრულდა.

თუ მეთოდი წარმატებით ჩაივლიდა, მაშინ მივიღებდით მწვანე ფერის x-სიმბოლოებს.

შემდეგი ეტაპი კოდის გასწორება და ხელახალი ტესტირებაა. დასატესტ პროგრამაში შევცვალოთ სტრიქონში „ + „ ნიშანი „ – „ -ით.

```
// m_balance += amount; // intentionally incorrect code
m_balance -= amount; // intentionally correct code
```

ტესტის თავიდან ამუშავებით ვიღებთ წარმატებულ შედეგს ანუ მიიღება მწვანე ფერის სიმბოლოები (ნახ.5.42).



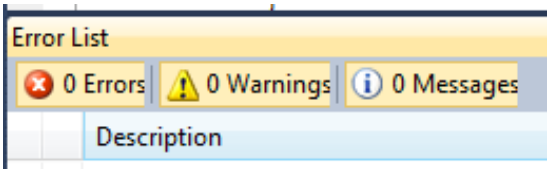
ნახ.5.42. სწორი შედეგი

5.4. გამონაკლის შემთხვევათა აღმოჩენისა და გამორიცხვის ვიზუალური საშუალებები

მიზანი: პროგრამებისა და აპლიკაციების გამართვის პროცესში გამონაკლისი შემთხვევებისა და შეცდომების აღმოჩენა და მათი გამორიცხვა.

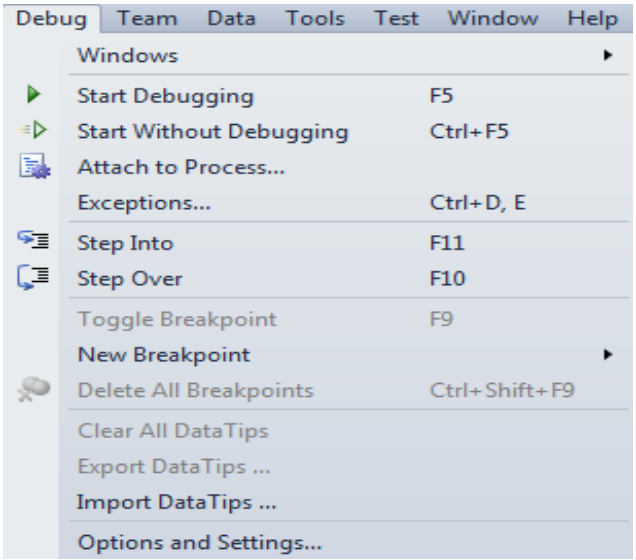
როგორც ცნობილია, პროგრამის გამართვისა და ტესტირების (შესრულების) პროცესში ადგილი აქვს პროგრამული შეცდომების გამოვლენას. ეს შცდომები სამი სახისაა: *სინტაქსური, პროცედურული და ლოგიკური.*

სინტაქსური შეცდომების აღმოჩენა ხდება C#-ენის კომპილატორის საშუალებით და გამოიტანება პროგრამული ტექსტების რედაქტორის ქვედა ნაწილში, Error List ფანჯარაში (ნახ.5.43). მათი პოვნა და შესწორება შედარებით ადვილია.



ნახ.5.43

პროცედურული შეცდომები ვლინდება პროგრამის შესრულების პროცესში, როცა პროგრამაში აღარაა სინტაქსური შეცდომები და ხდება მისი ამუშავება: Start Debugging ან F5 (ნახ.5.44), შესაძლებელია ისეთი შეცდომის გამოვლენა, რომელიც წყვეტს პროგრამის შესრულების პროცესს (ანუ სრულდება ავარიულად).

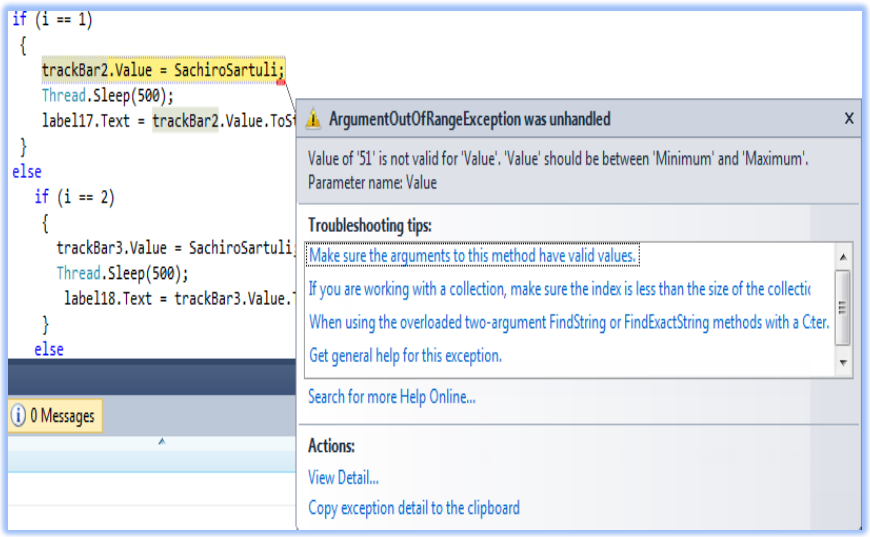


ნახ.5.44

დებაგერს გამოაქვს ამ დროს სათანადო შეტყობინება (ნახ.5.45), რომელიც პროგრამისტის მხრიდან მოითხოვს ანალიზსა და შეცდომის გამორიცხვას (Exception Handling).

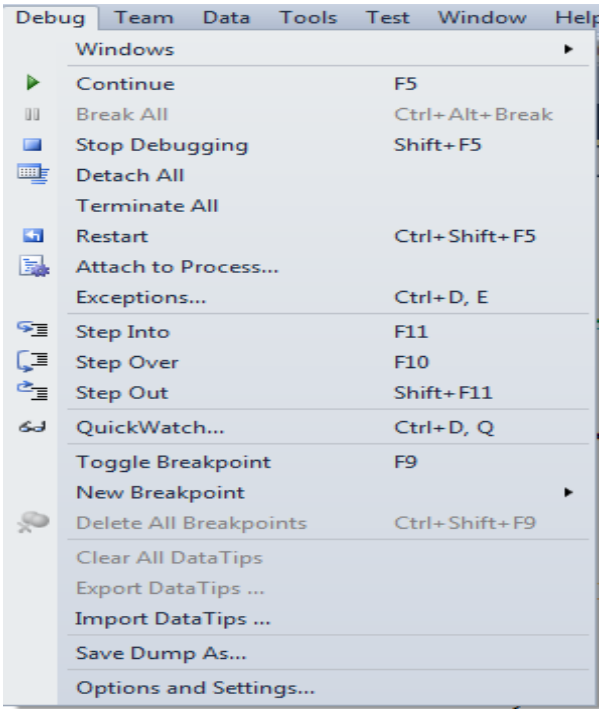
ლოგიკური შეცდომების აღმოჩენა შედარებით რთულია. პროგრამა ამ დროს მუშაობსა და სრულდება ნორმალურად (არაავარიულად), მაგრამ შედეგები „საეჭვოა“.

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



ნახ.5.45

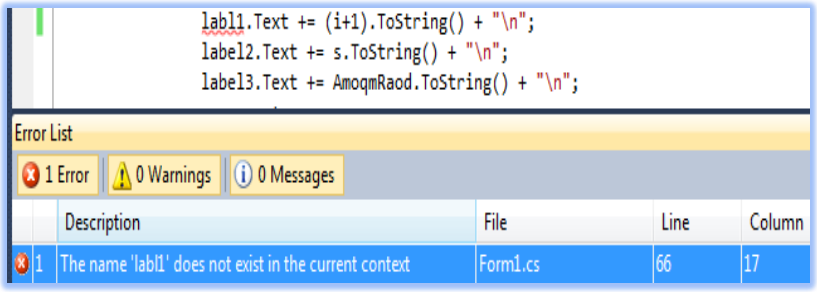
C# ენის რედაქტორს აქვს კარგი ინსტრუმენტული საშუალებები (Debugger) ამ ტიპის შეცდომების მოსაძებნად და აღმოსაფხვრელად (ნახ.5.46).



ნახ.5.46

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 შეცდომასა და მის ადგილმდებარეობას (ნახ.5.47).



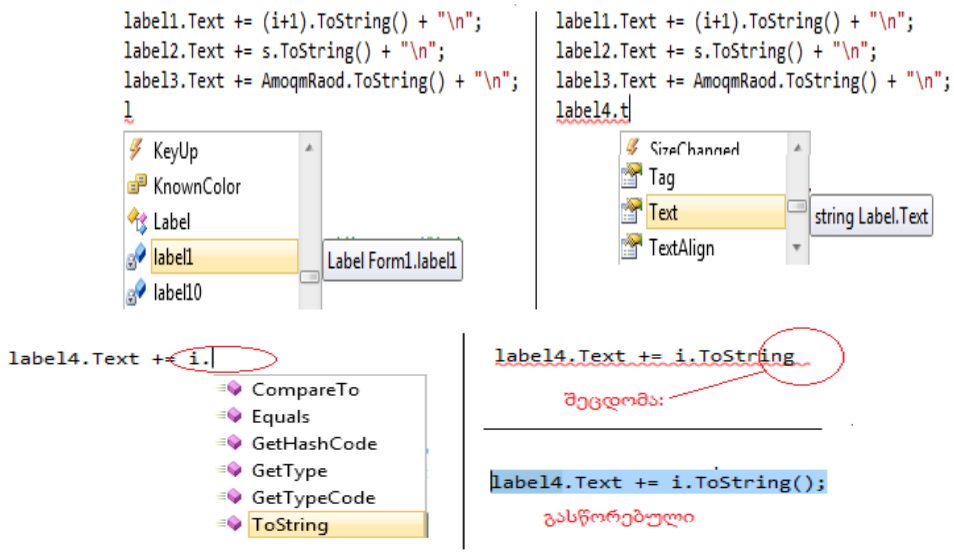
ნახ.5.47

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 საშუალება. მაგალითად, პროგრამაში ოპერატორების ტექსტის შეტანისას, ან ობიექტის მეთოდისა და მოვლენის არჩევისას (წერტილის „.“ დასმისას) ხდება ვიზუალური ბლოკის (Intellisense - ავტოდამატება) შემოთავაზება (ნახ.5.48), საიდანაც ამოირჩევა საჭირო სიტყვა და Enter-კლავიშით სწრაფად ჩაისმება მითითებულ ადგილას.

ეს გამორიცხავს როგორც ოპერატორის (ობიექტის თვისების, მეთოდისა და ა.შ.) არასწორ სინტაქსურ ჩაწერას, ისე 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 კონტროლისა და დამხმარე საშუალებები პროგრამისტის მუშაობას ეფექტურს ხდის.

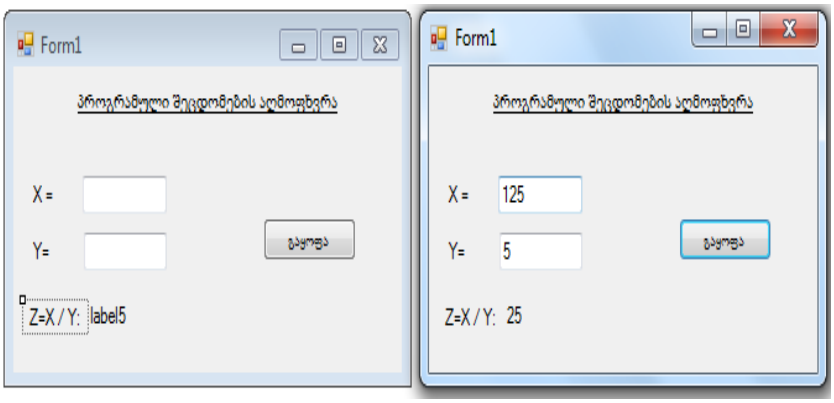


ნახ.5.48

2) განსაკუთრებული შემთხვევები: შეცდომები პროგრამის შესრულებისას და მათი აღმოფხვრა

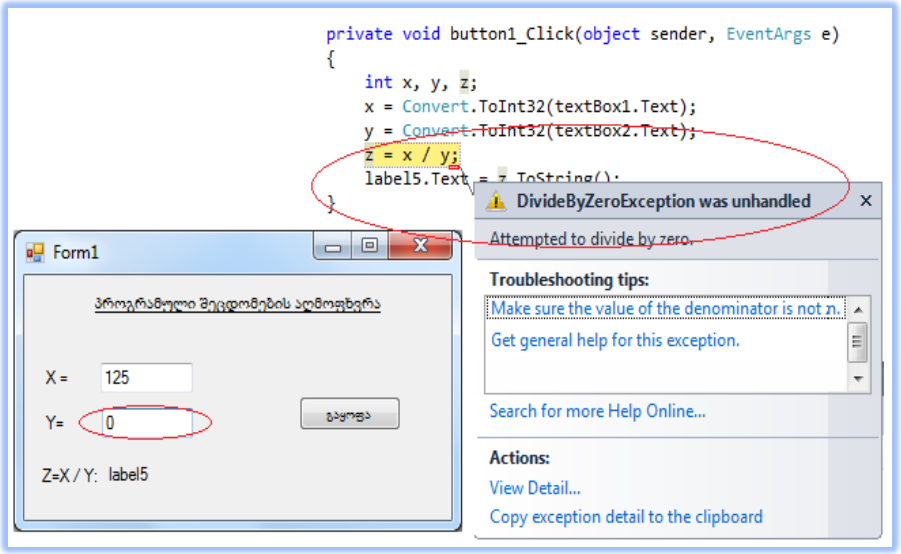
თუ პროგრამის ტექსტი სინტაქსური შეცდომებისაგან თავისუფალია, ის შეიძლება ამუშავდეს შესარულებლად. ამ დროს შესაძლებელია ისეთი შეცდომების გამოვლენა, რომლებიც პროგრამას ავარიულად დაასრულებს ან საერთოდ არ დაასრულებს („გაჭედავს“). მაგალითად, უსასრულო ციკლი ან სხვ. განსაკუთრებული შემთხვევა. განვიხილოთ ასეთი მაგალითები:

ამოცანა_1: ავაგოთ პროგრამის კოდი, რომელიც შეასრულებს მთელი რიცხვების შეტანასა და გაყოფის ოპერაციას. 5.49 ნახაზზე ნაჩვენებია ფორმა ორი ტექსტბოქსით (შესატანად), label5 შედეგის გამოსატანად და ღილაკი „გაყოფა“ პროგრამული კოდით (ნახ.5.50).



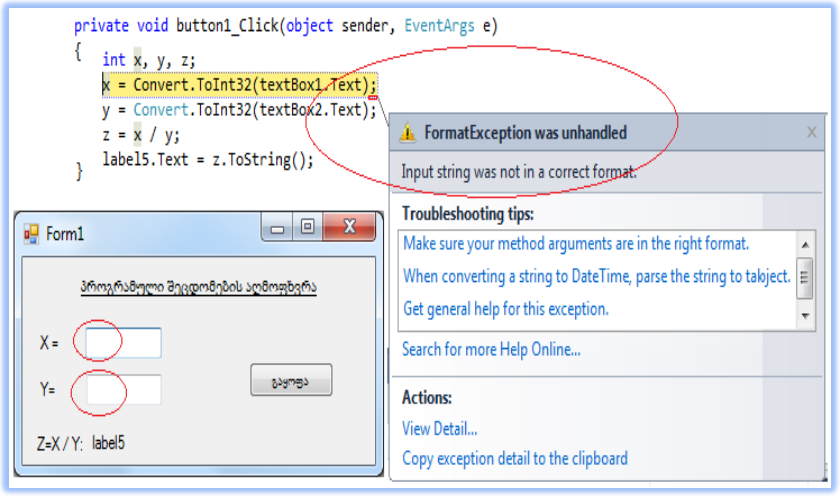
ნახ.5.49

როგორც ვხედავთ, პროგრამა მუშაობს თითქოს ნორმალურად, ასრულებს გაყოფას. ტესტირების პროცესში, რომელიც გულისხმობს კოდის ფუნქციის გამოკვლევას საწყისი მონაცემების სხვადასხვა მნიშვნელობისათვის, ვიღებთ „ნულზე გაყოფის“ შეცდომას (ნახ.5.50).



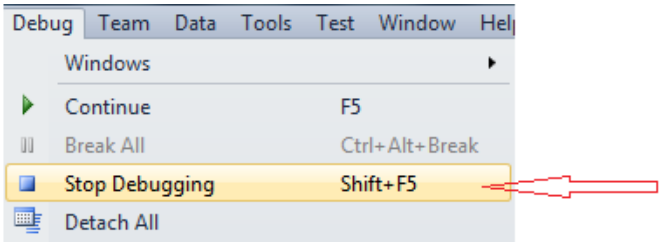
ნახ.5.50

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-საგან განსხვავებული) (ნახ.5.51).



ნახ.5.51

რედაქტირების რეჟიმში გადასასვლელად საჭიროა მენიუდან „დებაგერის შეჩერება“ (ნახ.5.52).

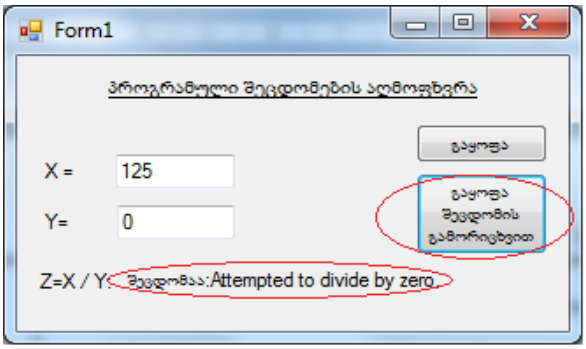


ნახ.5.52

ახლა შესაძლებელია ზემოაღწერილი ტიპის შეცდომებისა-თვის გამორიცხვის პროცედურის კოდის ფორმირება. მაგალითად, 5.8 ლისტინგზე მოცემულია ასეთი კოდი.

```
//ლისტინგი_5.8 --- 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 = "შეცდომაა:" + nul_Div.Message;
   }
}
```

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

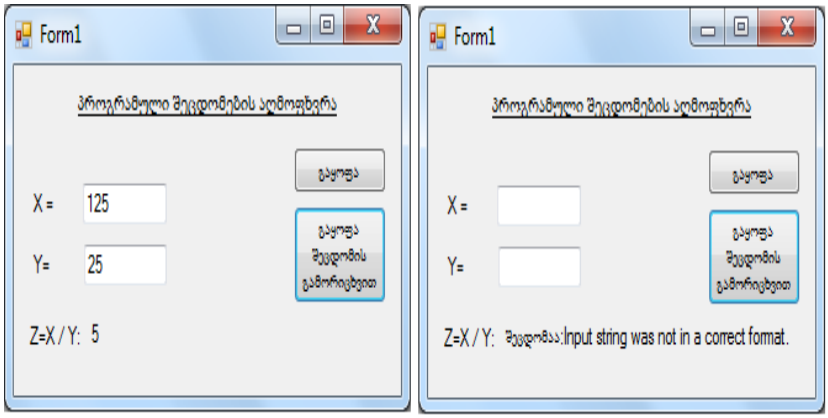


ნახ.5.53

საყურადღებო: გამოყენებულია კონსტრუქცია try { }… catch{ }. საკვანძო სიტყვა try ინიცერებას უკეთებს 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 არაავარიულად.

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



ნახ.5.54

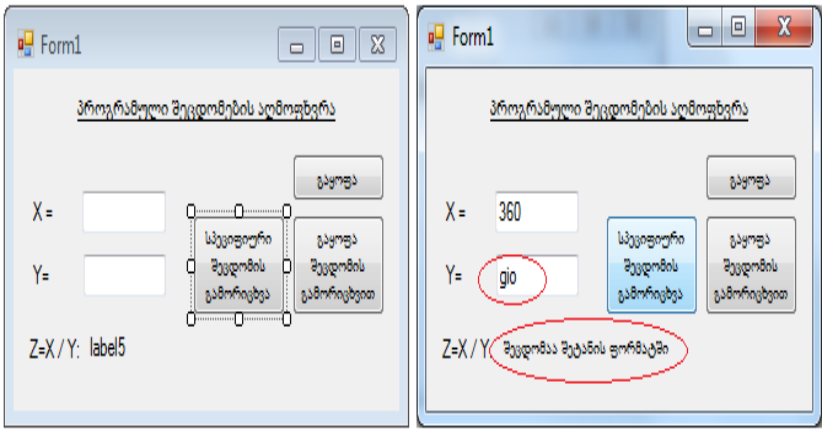
განსაკუთრებულ შემთხვევათა დამუშავების try…catch მექანიზმი შეიძლება გაფართოვდეს ბიბლიოთეკაში არსებული Exception-კლასის საფუძველზე. აქ იგულისხმება სპეციფიური შეცდომების აღმოჩენის შესაძლებლობა, მაგალითად, ტიპების გარდაქმნისას, ნულზე გაყოფისასა და ა.შ. თუ შეცდომის სახე წინასწარ არაა განსაზღვრული, მაშინ გამოიყენება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

5.9_ლისტინგში მოცემულია ღოლაკის „შეცდომის გამორიცხვა“ შესაბამისი კოდის ფრაგმენტი.

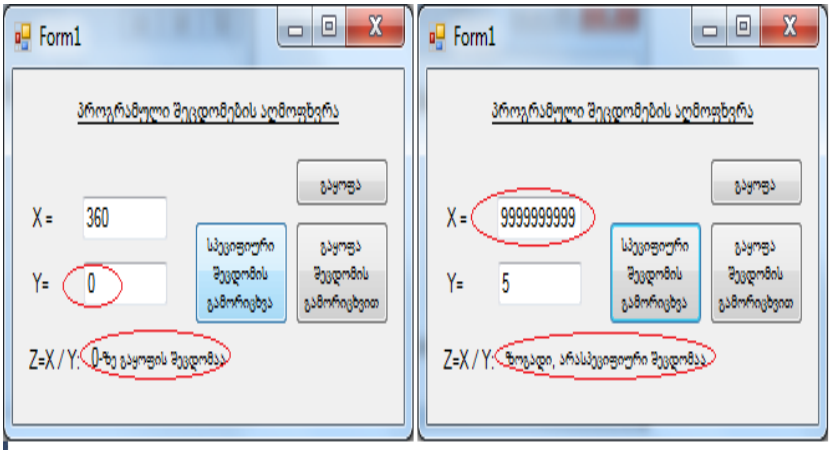
```
// ლისტინგი_5.9 ---- - სპეციფიური და ზოგადი შეცდომები ---------
private void button3_Click(object sender, EventArgs e)
 {
  int x, y, z;
  try
   {
     x = Convert.ToInt32(textBox1.Text);
     y = Convert.ToInt32(textBox2.Text);
     z = x / y;
     label5.Text = z.ToString();
   }

 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 შეცდომაა\n"+ nul_Div.Message;
   }
}
```

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

შედეგები ასახულია 5.55-ა,ბ ნახაზებზე.



ნახ.5.55-ა



ნახ.5.55-ბ

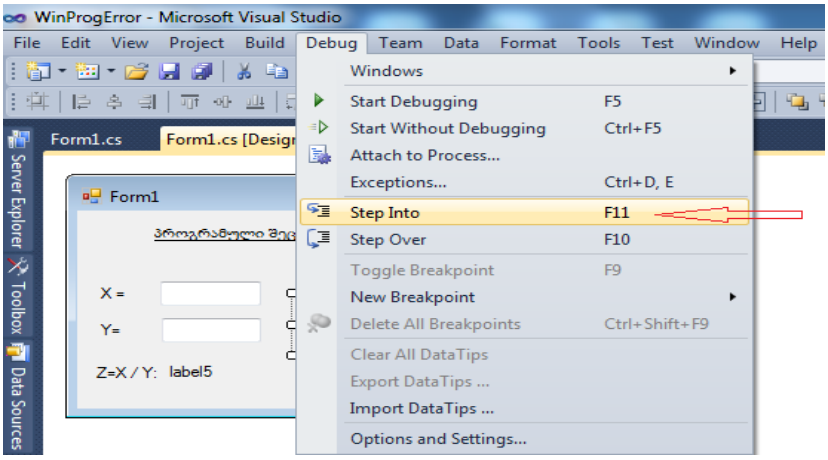
2.3) ლოგიკური შეცდომები და პროგრამის გამართვა (Debugging) ბიჯური შესრულების რეჟიმში

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!

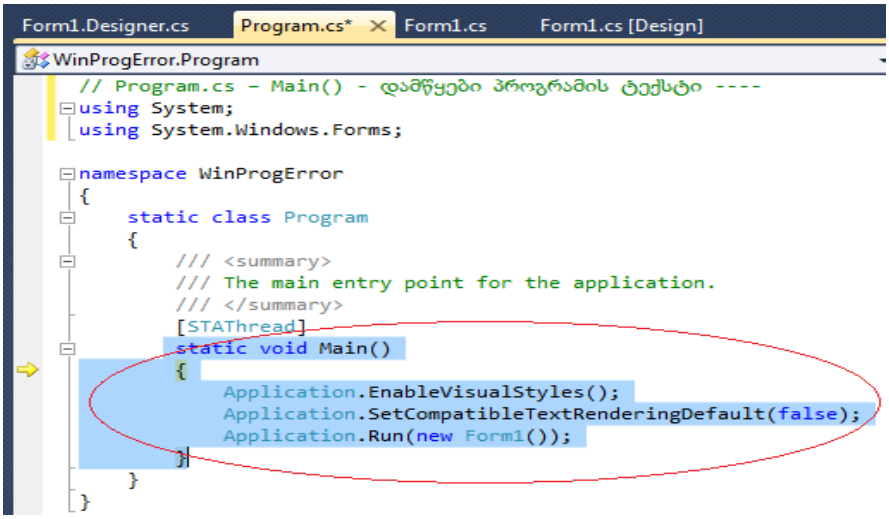
ასეთი შეცდომების აღმოჩენა საკმაოდ რთულია და მოითხოვს ტესტირების პროცესისა და პროგრამის შესრულების მიმდევრობის შედეგების ანალიზს.

ვიზუალური C# ენა ფლობს პროგრამის გამართვის (Debugging) კარგ დამხმარე საშუალებებს. განვიხილოთ ისინი ჩვენი WinProgError პროექტის მაგალითზე (ნახ.5.56). გავხსნათ პროექტი, მოვამზადოთ Form1 ფორმა და მენიუს Debug-ში ავირჩიოთ Step Into (ან F11 ღილაკი კლავიატურის ზედა რიგში).



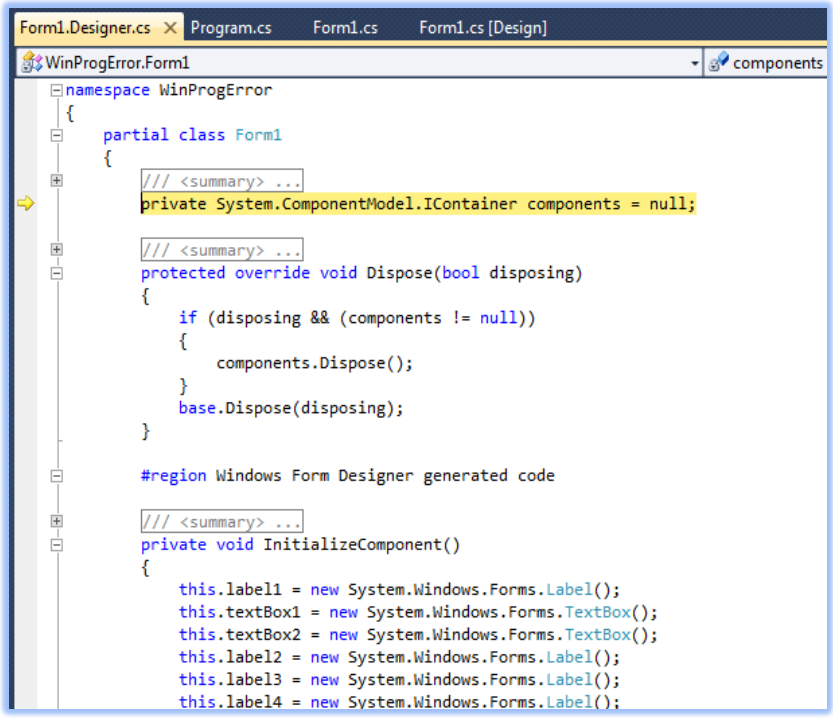
ნახ.5.56

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.5.57). მარცხნივ ჩანს ყვითელი ისარი, რომელიც F11-ით ბიჯურად გადაადგილდება იმ სტრიქონზე, რომელიც სრულდება მოცემულ მომენტში.



ნახ.5.57

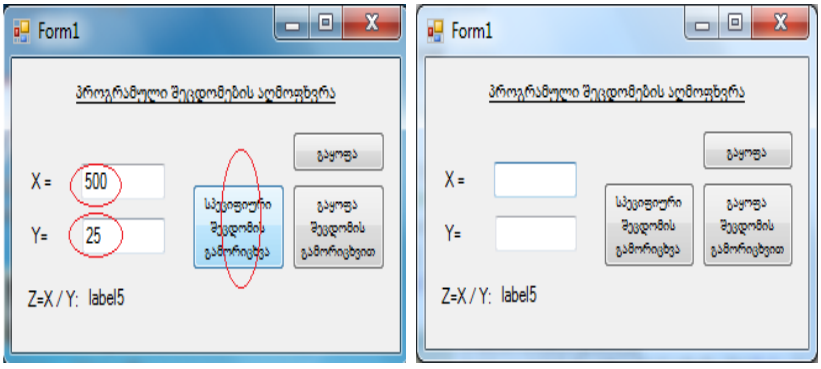
Application.Run (new Form1()); სტრიქონის ამუშავებით ისარი გადადის (იხ. Solution Explorer) Form1.Designer.cs პროგრამაში (ნახ.5.58). შემდეგ InitializeComponent()-ში გაივლის ფორმაზე დალაგებულ ყველა ელემენტს.



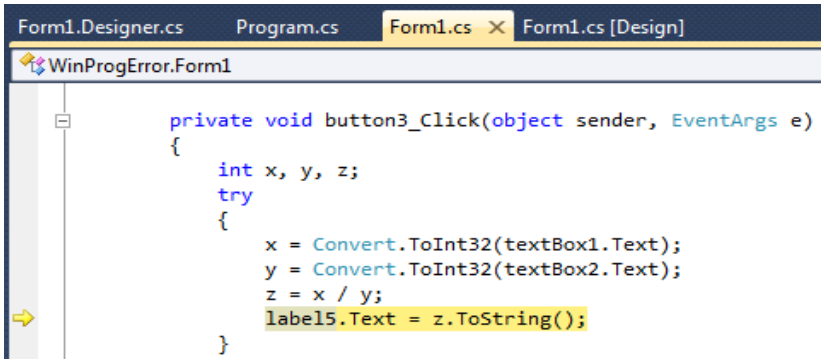
ნახ.5.58

Form1.Designer.cs პროგრამის ბიჯურად გავლის შემდეგ Main()-იდან ამუშავდება Run და ეკრანზე გამოვა Form1 (ნახ.5.59), სადაც უნდა შევიტანოთ X და Y მნიშვნელობები და ავამოქმედოთ ღილაკი „სპეციფიური შეცდომის გამორიცხვა“ (ნახ.5.60).

ბიჯის ისარი გადადის Form1.cs პროგრამის ტექსტზე (ნახ.5.60), სადაც button3_Click მოვლენის შესაბამის try {…} ბლოკში შედის.

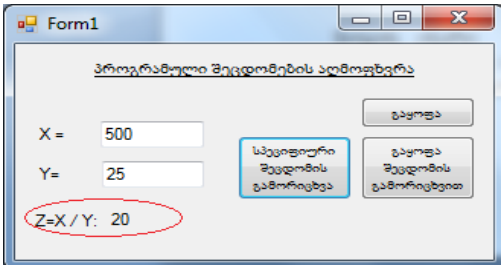


ნახ.5.59



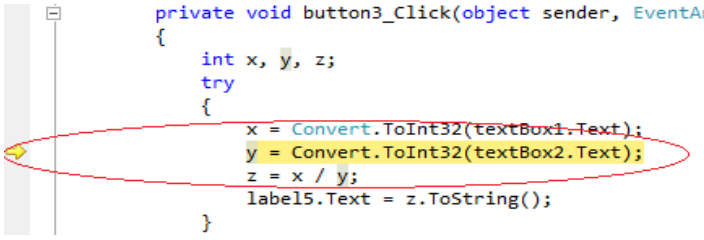
ნახ.5.60

ვინაიდან X და Y-ის შეტანილი მნიშვნელო-ბები დასაშვებია, შედეგში აისახება label5.Text=z.ToString() მნიშვნელობა, ანუ 20 (ნახ.5.61).



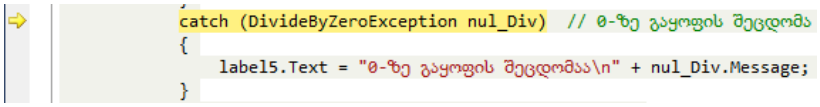
ნახ.5.61

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



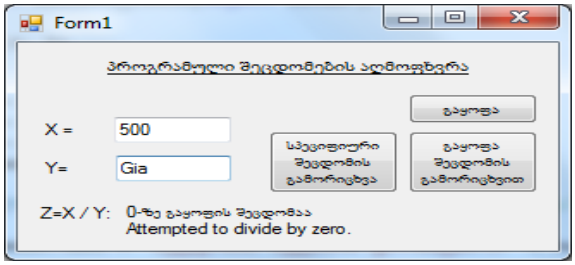
ნახ.5.62

მოქმედება გადაეცემა catch ბლოკს:



ნახ.5.63

დავუშვათ, რომ X ან Y არარიცხვითი სიმბოლო ან სტრიქონია, მაგალითად, “G, Gia,…” (ნახ.5.64).



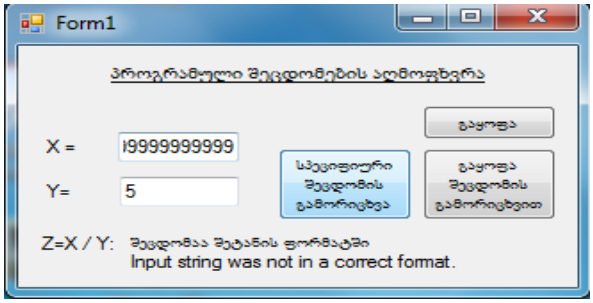
ნახ.5.64

F11-ით გადაადგილების შემდეგ პროგრამის ტექსტის აქტიური ფრაგმენტი იქნება შემდეგი (ნახ.5.65).

```
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;
    }
```

ნახ.5.65

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.2.24), მაშინ კოდის ფრაგმენტი ასე გამოიყურება (ნახ.5.66).



ნახ.5.66

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

ნახ.5.67

პროგრამასთან მუშაობის დასამთავრებლად დავხუროთ Form1. ამ დროს მართვა (ისარი) გადაეცემა Form1.Designers.cs (ნახ.5.68) და ბოლოს პროგრამას Form1.cs, რომელშიც არის Main(), და რომლითაც დაიწყო თავიდან ამ პროგრამის მუშაობა (ნახ.5.69).

Form1.Designer.cs | 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);
        }
```

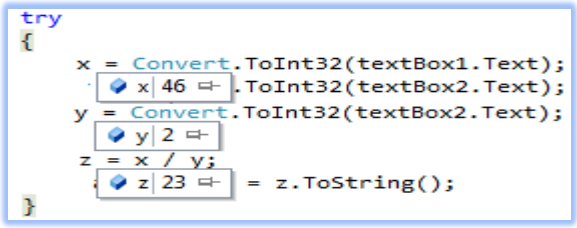
ნახ.5.68

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

ნახ.5.69

ამით დასრულდება დებაგერის მუშაობის პროცესი.

პროგრამის ანალიზის პროცესში შესაძლებელია ცვლადების მნიშვნელობათა ვიზუალური შემოწმება. ამისათვის მაუსის კურსორი უნდა მივიტანოთ ცვლადთან. 5.70 ნახაზზე ნაჩვენებია პროგრამაში X, Y და Z -ის მნიშვნელობები.

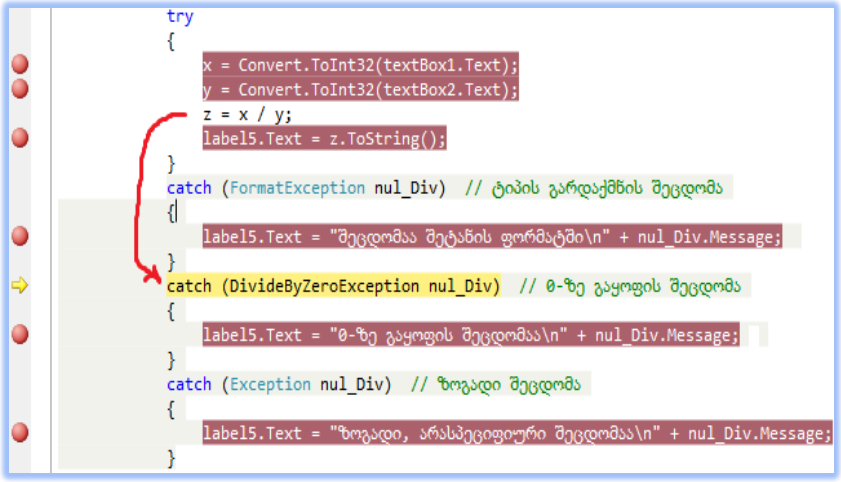


ნახ.5.70

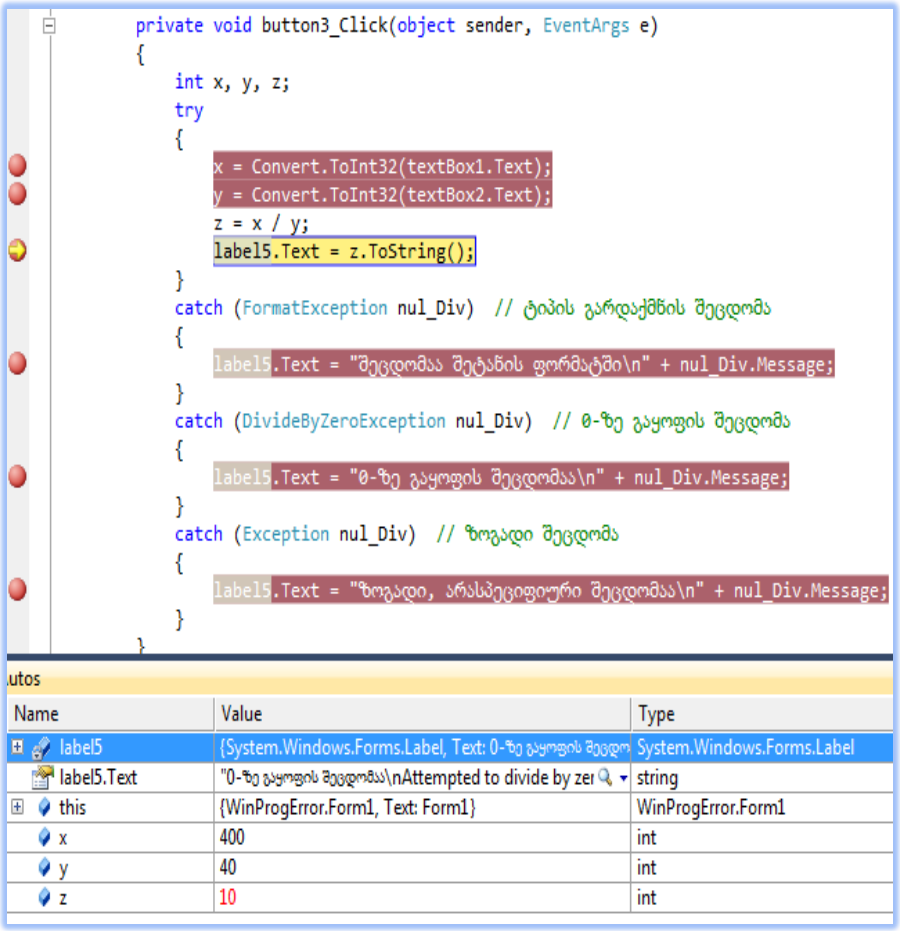
დიდი პროგრამების ანალიზის დროს ბიჯურ რეჟიმში მუშაობა არაეფექტურია, სჭირდება ხანგრძლივი დრო. უფრო მოსახერხებელია ვიზუალური კონტროლის ორგანიზების მეორე ხერხი, რომელიც წყვეტის წერტილების კონცეფციითაა ცნობილი.

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 თაგვის მარცხენა კლავიშს (ან F9 კლავიშს). გამოჩნდება შინდისფერი წრე. პროგრამის კოდში შეგვიძლია შევქმნათ წყვეტის რამდენიმე წერტილი (ნახ.5.71). მენიუდან Debug →Windows→Autos არჩევით რედაქტორის ქვედა ნაწილში გამოიტანება ცვლადების მონიტორინგის ფანჯარა, რომელშიც ერთდროულად ჩანს რამდენიმე ცვლადი მათი აქტუალური მნიშვნელობებით (ნახ.5.72). მენიუდან Debug→Windows→Locals -ის არჩევით მონიტორინგის ფანჯარაში გამოიტანება მხოლოდ ლოკალური ცვლადები და მათი მნიშვნელობები.



ნახ.5.71

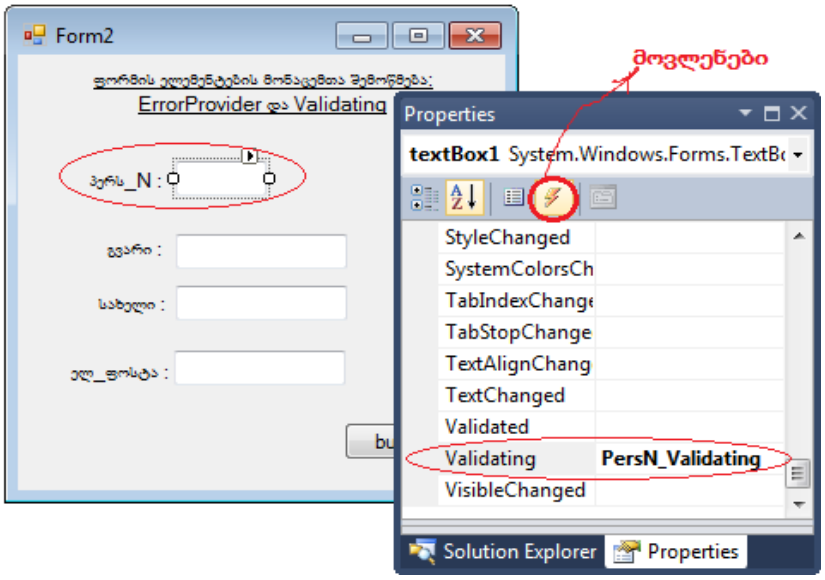


ნახ.5.72

4) შესატან მონაცემთა კონტროლი

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

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.5.73).



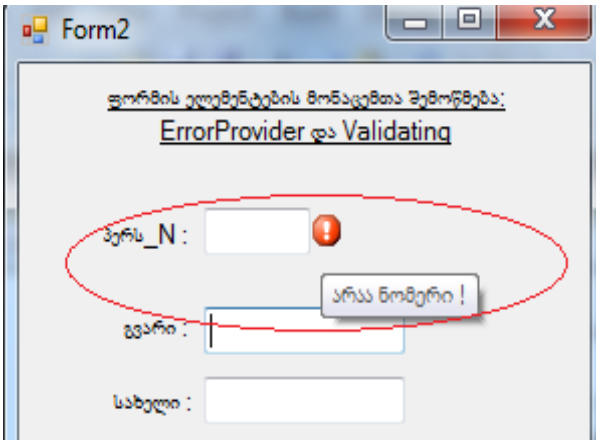
ნახ.5.73

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

ამგვარად ახლა საჭიროა მოვლენის დამმუშავებელის შექმნა. მაგალითად, ვდგებით „პერს_N“ ტექსტბოქსზე და Properties-ში Validating თვისებას ვაძლევთ სახელს: PersN_Validating. დამმუშავებლის მეთოდის კოდი მოცემულია 5_9 ლისტინგში.

```
// ლისტინგი_5.9 ---  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 5.74 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).

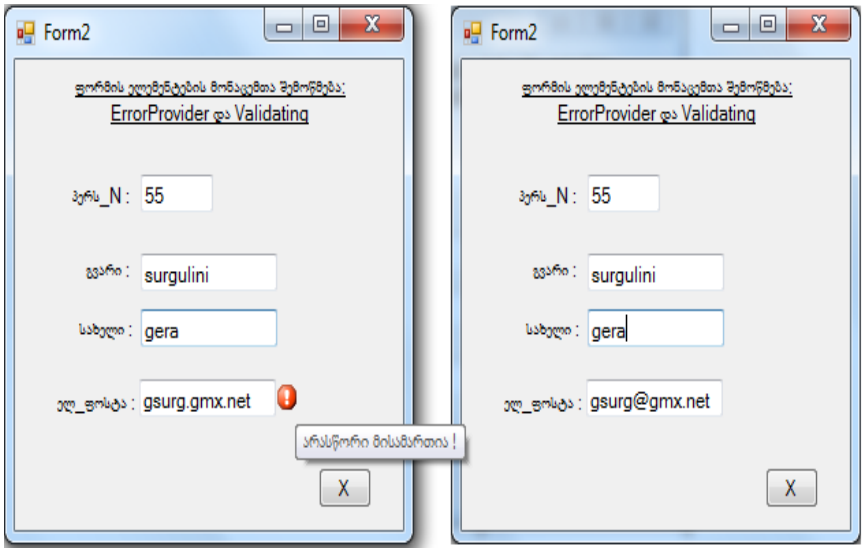


ნახ.5.74

მეოთხე ტქსტბოქსში 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 (5.10_ლისტინგი).

```
// ლისტინგი_5.10 --- 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 ნაჩვენებია 5.75 ნახაზზე.


Form2
ფორმის ელემენტების მონაცემთა შემოწმება:
ErrorProvider და Validating
პერს_N : 55
გვარი : surgulini
სახელი : gera
ელ_ფოსტა : gsurg.gmx.net
არასწორი მისამართია !
X
Form2
ფორმის ელემენტების მონაცემთა შემოწმება:
ErrorProvider და Validating
პერს_N : 55
გვარი : surgulini
სახელი : gera
ელ_ფოსტა : gsurg@gmx.net
X

ნახ.5.75

ლიტერატურა:

1. ჩოგოვაძე 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., კივილაძე 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 Fundation в .NET 3.5 с примерами на C# 2008 для профессионаюв. 2-е издание: Пер. с англ. - М. : ООО "И.Д. Вильяме". 2008

15. Petzold Ch. Applications=Code+Markup. A Guide to the MicroSoft Windows Presentation Foundation. St-Petersburg. 2008

16. Microsoft Visual Studio. https://en.wikipedia.org/wiki/Microsoft_Visual-_Studio

17. What is .NET Framework? Complete Architecture Tutorial. https://www.-guru99.com/net-framework.html

18. გოგიჩაიშვილი გ., სურგულაძე გ., შონია ო. დაპროგრამება C/C++ ენებზე. სტრუქტურული და ობიექტორიენტირებული დაპროგრამება. სტუ, ტექნიკური უნივერსიტეტი. თბ., 1997.

19. პეტრიაშვილი ლ., სურგულაძე გ. მონაცემთა მენეჯმენტის თანამედროვე ტექნოლოგიები (Oracle, MySQL, MongoDB, Hadoop). სტუ. „IT-კონსალტინგის ცენტრი“. თბ., 2017

20. Halpin T. ORM 2 Graphical Notation, Neumont University. 2005. http://www.orm. net/pdf/ORM2_TechReport1.pdf

21. Surguladze G., Turkia E., Topuria N., Gavardashvili A. Construction of the Multimedia Databases and Users Interfaces for Ecological System of Black Sea with Orm/Erm. “Information and Computer Technology, Modeling and Control”. Chapt.45. Nova Science Publishers. © Copyright, 3rd Quarter. 2017, USA, pp. 1-8. https://www.novapublishers.com/catalog/product_info.php? products_id=62232

22. Xamarin. https://en.wikipedia.org/wiki/Xamarin

23.https://docs.microsoft.com/en-us/visualstudio/releases/2019/release-notes]

24. https://docs.microsoft.com/en-us/visualstudio/ide/whats-new-visual-studio-2019?view=vs-2019]

25. Visual Studio Code. https://en.wikipedia.org/wiki/Visual_Studio_Code

26. .NET Core and .NET Standard: What Is the Difference? https://www.infoq-.com/news/2017/10/dotnet-core-standard-difference/

27. https://en.wikipedia.org/wiki/Visual_Studio_Code.

28. http://www.williamspublishing.com/PDF/978-5-9908910-4-3/part.pdf.

29. Python + Visual Studio Code = успешная разработка. ProgLib. Библиотека программиста. https://proglib.io/p/python-vscode/

30. https://www.amarinfotech.com/difference-between-net-core-2-0-vs-net-framework.html

31. https://www.infoq.com/news/2017/10/dotnet-core-standard-difference/

32. https://en.wikipedia.org/wiki/Application_programming_interface

33. სურგულაძე გ., თოფურია ნ., გავარდაშვილი ა. შავი ზღვის ეკოლოგიური მონიტორინგისა და კვლევის საინფორმაციო სისტემა. სტუ. „IT-კონსალტინგის ცენტრი“. თბ., 2018

34. მეიერ-ვეგენერი კ., სურგულაძე გ., ბასილაძე გ. საინფორმაციო სისტემების აგება მულტიმედიური მონაცემთა ბაზებით. ISBN 978-9941-20-468-5. სტუ, თბილისი, „ტექნიკური უნივერსიტეტი“. 2014.

35. ვედეკინდი ჰ. (გერმანია), სურგულაძე გ., თოფურია ნ. განაწილებული ოფის-სისტემების მონაცემთა ბაზების დაპროექტება და რეალიზაცია UML-ტექნოლოგიით. სტუ. თბ., 2006.

36. სურგულაძე გ., თოფურია ნ. მონაცემთა ბაზების მართვის სისტემები: ობიექტ-როლური მოდელირება (ORM/ERM, SQL Server). სტუ. თბ., 2007.

37. ჩოგოვაძე გ., ფრანგიშვილი ა., ჯაგოდნიშვილი თ., სურგულაძე გ. საინფორმაციო სისტემებიდან ინფორმაციული საზოგადოებისაკენ. სტუ შრ.კრებ..: „მას"-N1(23). 2017. გვ.7-16

38. ჩოგოვაძე გ., ფრანგიშვილი ა., სურგულაძე გ., პეტრიაშვილი ლ., ნარეშელაშვილი გ. ბიზნესის საინფორმაციო სისტემების ინტეგრირებული გამოყენება ინტერდისციპლინური განათლების სფეროში სტუ შრ.კრებ..: „მას"-N2(24). 2017. გვ.7-16

39. Windows Forms controls. msdn, Microsoft. 2017. https://docs.microsoft-.com/en-us/dotnet/framework/winforms/controls/

40. BindingNavigator Control Overview (Windows Forms). msdn, Microsoft. 2017. https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/binding-navigator-control-overview-windows-forms

41. BindingSource Component Overview. msdn, Microsoft. 2017. https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/bindingsource-component-overview

გადაეცა წარმოებას 10.10.2019. ხელმოწერილია დასაბეჭდად 15.10.2019. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 15. ტირაჟი 100 ეგზ.

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრი“
(თბილისი, მ.კოსტავას 77)

ISBN 978-9941-8-1708-3