Programimi në C# me Microsoft .NET

Përpiluar nga :
Fatbardha Abazi

Përmbajtja

Moduli 1: Hyrje në C# *3*
1.1. Çka është .NET Platform-a? 4
1.2. Çka është .NET Framework? 5
1.3. Puna me Programet zhvillimore 6
1.4. Karakteristikat e programimit në Visual Studio.Net 7
Moduli 2: Elementet e gjuhës C# *8*
2.1. Struktura në C# 8
2.2. Tipet e të dhënave (Data Type) 9
2.3. Deklarimi dhe inicializimi i variablave 10
2.4. Enumeration types (grupimi i variablave) 11
2.5. Konvertimi i tipeve të të dhënave 12
2.6. Operatorët dhe shprehjet 12
2.6.1. Tipet e operatorëve 12
2.6.2. Operatorët logjikë 13
2.7. Komanda për degëzim 13
2.7.1. Struktura e komandës if else 13
2.7.2. Struktura e komandës if else if else : 14
2.8. Unazat (ang. loops) 15
2.8.1 Unaza for 15
2.8.2. Unaza while 17
2.8.3.Unaza do while 17
Moduli 3: Definimi dhe përdorimi i Klasave *19*
3.1. Klasat dhe objektet 20
3.2. Definimi i klasave dhe krijimi i objektetve 20
3.3. Organizimi i klasave duke përdorur Namespaces 21
3.4. Metodat 22
3.5. Konstruktorët 24
3.6. Çfarë janë Properties 25
Moduli 4: Vektorët (Arrays) *26*
4.1. Vektorët (Arrays) 27
4.2. Krijimi i vektorëve 28
4.3. Përdorimi i ArrayList 30
Moduli 5: Ndërlidhja me databazë *31*
5.1. Teknologjia ADO.NET 32
5.2. Klasa DataSet 32
5.3. Krijimi i objekteve DataAdapter-ave 32
5.4 Dizajnimi i Formës 33
Përfundimi dhe Literatura *45*

Moduli 1: Hyrje në C#

Ky modul prezenton konceptet kryesore për Microsoft .NET Framework dhe .NET Platformën si dhe *Microsoft Visual Studio .NET Integrated Development Environment*. Ky modul gjithashtu shpjegon edhe se si ta përdorim Visual Studio .NET-in që të krijojmë aplikacione të bazuara në Microsoft .NET –in.

Megjithëse mund ta përdorim edhe Microsoft Notepad-in që të shkruajmë aplikacione të ndryshme dhe t'i kompajlojmë ato parcialisht duke i përdorur teknikën e programimit *command line (një komandë një rresht)*, programet për zhvillim (siç janë programet e Visual Studios) e ngritin produktivitetin dhe efektshmërinë e programimit duke i centralizuar të gjitha veglat që përdoren për ta dizajnuar programin, të cilat ofrojnë disa mundësi dhe funksione të ndryshme për funksionalitet sa më të lartë të programit të shkruar, gjithashtu ofron edhe mundësinë e ndihmës (*Microsoft IntelliSense dhe Ndihma Dinamike).*

Objektivat

Pas kompletimit të këtij moduli, ju do të jeni në gjendje të:

- Identifikoni komponentet e *.NET Platformës* dhe *.NET Framework-ut*
- Exploroni Visual Studion
- Krijoni një aplikacion të bazuar në *Microsoft Windows.*

1.1. Çka është .NET Platforma?

.NET Platforma ofron teknologji dhe shërbime të ndryshme të cilat e lehtësojnë zhvillimin e Web-aplikacioneve.
Komponentet e .NET Platformës janë:

- Veglat për zhvillim:
  Visual Studio .NET ofron një mjedis të përshtatshëm për zhvillim të aplikacioneve sepse ofron vegla të ndryshme që e lehtësojnë krijimin, shpërndarjen, sigurimin dhe mundësojnë mundësi (aftësi) sa më të mëdha të Web aplikacioneve dhe XML Web shërbimeve.

- Pajisjet:
  Pajisjet në të cilat mund të ekzekutohet një program i zhvilluar në .NET Platformë janë : kompjuteri personal, laptopi, telefoni, pajisjet për lojëra, etj. *Smart Devices (Pajisjet e mençura)* lejojnë qasjen në të dhënat e XML Web shërbimeve pavarësisht nga lokacioni, tipi, numri telefonik (për celular) që pajisja e përdor etj.

- User experiences (Eksperiencat e shfrytëzuesit)
  .NET experiences janë aplikacione të cilat i përdorin XML Web shërbimet që t'i mundësoj shfrytëzuesve të qasen në informata përmes Internetit dhe përmes një programi të vetëm në një mënyrë efikase.

- Serverët:
  .NET Enterprise Server e përforcojnë integritetin e sistemit, aplikacioneve, dhe partnerëve duke mbështetur XML Web shërbimet. Mbështetja e XML lejon që të shkruajmë aplikacione në sisteme më të hershme dhe t'i përdorim ato në sisteme më të reja, pa pasur nevojë t'i rishkruajmë ato.

- XML Web services (XML Web shërbimet) :
  Duke e përdorur XML Web shërbimet aplikacionet mund t'i shkëmbejnë të dhënat dhe të komunikojnë në mes vete me një aplikacion tjetër, pa marrë parasysh atë se si janë shkruar ato aplikacione, në çfarë sistemi operativ ose platforme punojnë ato, dhe çfarë pajisje i shfrytëzon ato aplikacione.

.NET Platforma ofron disa përparësi dhe përfitime për zhvilluesit e aplikacioneve (programerëve), siç janë:

- Shpejtësia e zhvillimit (programimit) të aplikacioneve
- Fleksibilitet
- Standarde të bazuara në Web (XML Web services), etj.

1.2. Çka është .NET Framework?

Net.Framework ofron bazën për krijimin e aplikacionit të bazuar në Platformën *.Net*.
Net.Framework konsiston në dy komponente:
CLR (*Common language runtime*) dhe në librari të klasave të cilat ekzekutohen në sistemin operativ. Çdo gjuhë programuese që mbështetet në CLS mund të ekzekutohet në CLR. *Net.Framework Microsoft o*fron mbështetjen për këto programe siç janë: .Net Framework, Microsoft Visual C#, dhe Microsoft Jscript.
CLR (Common language rutine) menaxhon ekzekutimin e kodit dhe ofron shërbime për lehtësimin e zhvillimit të aplikacioneve.

Libraria e klasave .Net.Framework
Libraria e klasave shfaqin karakteristikat e programit gjatë kohës së ekzekutimit dhe ofrojnë klasa të cilat janë të qasshme nga të gjitha aplikacionet të bazuara në web, windows, apo në *xml web* shërbime.

Libraritë e klasave *.Net.Framework* janë:

- ***ADO.Net***
- ***ASP.Net***

ADO.Net është gjeneratë e *Microsoft Active Data Objekt*. ADO.Net ofron mbështetjen e programimit të shkyçur. Kjo gjithashtu përkrah edhe XML.

ASP.Net përdoret për të ndërtuar web aplikacione. ASP.Net web format mbështetin një mënyrë të lehtë dhe të fuqishme për krijimin e web-ndërfaqeve dinamike për përdoruesin.

USER interfaces (ndërfaqja me përdorues)
Net. Framework ofron tri tipe të ndërfaqeve:

- ***Web Format***
- ***Windows Format***
- ***Console Aplikacionet***

Web format punohen duke përdorur ASP.Net.
Windows Format janë forma të cilat ekzekutohen në Win32 (Kompjuterëve të klientit) dhe puna e tyre bazohet në dritare, siç është p.sh. Kalkulatori.
Console aplikacionet janë aplikacione që bazohen në konzollë (console) si p.sh. Command Prompt.

1.3. Puna me Programet zhvillimore

Puna me programet zhvillimore përfshinë disa hapa të cilët duhet ndjekur:

1. Hapja Visual.Net dhe përdorimi i faqes startuese
   Kliko në ***Start-All-Microsoft,***
   Kliko në ***Online Community,***
   Kliko ***Search Online,***
   Kliko ***Get Started.***
2. Krijimi i një projekti të ri
   Kliko ***File-New-Projekt***
   Zgjidh në dritaren për dialog opsionin ***Windows Application,***
   Në fund në fushën ***Name*** shkruaj një emër dhe kliko ***OK.***
3. Ekzekutimi i ***Solution Explorer***
   Tek ***Solution Explorer*** kliko formën,
   Kliko me të djathtën dhe zgjidh formën të cilën dëshiron të hapet.
4. Shtimi i një klase të re në projekt
   Mbi Projekt në të djathtë kliko ***Add-class,***
   Në dritaren për dialog zgjedhet opsioni ***class,***
   Në kornizën ***Name***:shkruaj emrin e klasës,
   Mbylle ***Code editor*** për klasën e shtuar.
5. Ndryshimi i madhësisë së formës
   Selektohet forma dhe ndryshimi i madhësisë së formës bëhet nga ana e djathtë, nga ajo e poshtme ose nga këndi i poshtëm-majtas.
6. Shtimi i butonave në formë:
   Në ***toolbox*** kliko dy herë ***button*** dhe kështu bëhet shtimi i butoni.
7. Shkruarja e kodit në buton.
   Kliko dy herë në ***button1*** dhe hapet ***editor code*** (editori i kodit)
   Në ***editor code*** në mes të **private void Button Code** dhe ***End Sub***, shkruajmë kodin `MessageBox.Show("tung");`
8. Ekzekutimi dhe testimi i aplikacionit:
   Në toolbar, kliko butonin ***Start,***
   Kliko në ***button 1***,
   Shfaqet mesazhi "tung" kliko ***Ok,***
   Mbylle formën.
9. Ruajtja e projektit
   Kliko tek menyja kryesor ***File-Save All.***
10. Mbyllja e projektit
    Kliko tek menyja kryesore ***File*** dhe zgjidhe opsionin ***Close Solution.***
11. Mbyllja e ***Visual Studio,***
    Kliko ***File-Exit***.

1.4. Karakteristikat e programimit në Visual Studio.Net

Karakteristikat e programimit në *Visual Studio.Net* përfshinë disa hapa e ata janë:

- ***Dizajni***
- ***Zhvillimi***
- ***Kompajllimi***
- ***Deploy (lansimi)***

Dizajnimi ka të bëjë më atë se si do të duket programi , forma, pozicionimi, ngjyrën dhe madhësinë e komponentave të programit (button-at, checkbox-at, textbox-at, etj).

Zhvillimi ka të bëjë me shkrimin e kodit të programit.

Debug (kompajllimi), ka të bëjë me testimin e aplikacionit.

Deploy (vendosja) ruajtja e programit në disk tek klientët.

Moduli 2: Elementet e gjuhës C#

Ky modul ka të bëjë me hyrjen në sintaksën dhe strukturën bazë të C#.
Ky modul shpjegon tipet e të dhënave, duke përfshirë variablat dhe konstantet si dhe shpjegon se si të bëjmë grupimin e variablave konstante.

Objektivat
Pas kompletimit të këtij moduli, ju do të jeni në gjendje të:

- Kuptoni fundamentet e programimit në C#.
- Përdorni tipet e paradefinuara të C# .
- Shkruani shprehje matematikore.
- Krijoni urdhra për kushtëzim.
- Krijoni urdhra për përsëritje.

2.1. Struktura në C#

Struktura në C# duket kështu:

```
using System;
class HelloWorld
{
      static void Main ( )
      {
                  Console.WriteLine("hello, World");
      }
}
```

Fjalia *using* i referohet resurseve të librarive të klasave.
Fjalia *System* është *namespace* që ofron qasje në të gjitha funksionalitet e sistemit në të cilin shkruhet aplikacioni.
Fjalia *class* është definim i një klase prej të cilës mund të formohen objektet. Çdo gjuhë programuese e orientuar në objekte konsiston në shkrimin dhe përdorimin e klasave.
Main method është funksioni kryesor i cili ekzekutohet i pari, posa të ekzekutohet aplikacioni në C#, duhet të ketë metodën *main* në njërën prej klasave.
Urdhrat janë instruksione të cilat kryejnë veprime në aplikacionin e C#, ato ndahen në mes veti me presje. Urdhrat mund të shkruhen një rresht një urdhër apo në një rresht dy e më shumë urdhra, gjithashtu mund të shkruhen një urdhër në disa rreshta. Është praktikë e mirë që të shkruhen një rresht një urdhër.
Kllapat janë për të identifikuar fillimin dhe mbarimin e grupit të urdhrave.
Po ashtu në program mund të japim sqarime edhe për atë se çfarë kemi shkruar në program, e kjo bëhet me anë të komenteve. Komentet mund t'i bëjmë për një rresht apo edhe për më shumë rreshta. Komenti për një rresht bëhet duke shkruar slash (thyesë) (//) në fillim të rreshtit. Por ne mund të bëjmë edhe komentin për më shumë rreshta në këtë mënyrë /* programi */.

2.2. Tipet e të dhënave (Data Type)

Të dhënat mund të jenë të ndryshme, ato mund të jenë numra (numra me pikë dhjetore, numra shumë të mëdhenj, numra me parashenjë, numra pa parashenjë), fjalë, fjali. Sikurse në çdo gjuhë tjetër programuese edhe në C# ekziston sintaksa se si duhet të deklarohen dhe inicializohen tipet e të dhënave. Të dhënat mund të jenë të tipit integer, float, double, etj.

Definimi i Tipit	Përkufizimi
Byte	Numra të plotë ndërmjet 0 dhe 255
Sbyte	Numra të plotë ndërmjet -128 dhe 127.
Short	Numra të plotë ndërmjet -2768 dhe 2767.
Ushort	Numra të plotë ndërmjet 0 dhe 65525.
Int	Numra të plotë ndërmjet -214748647 dhe 214748647.
Uint	Numra të plotë ndërmjet 0 dhe 4294967295.
Long	Numra të plotë ndërmjet 9227206854775807 dhe 9227206854775807.
Unlog	Numra të plotë ndërmjet 0 dhe 1844674407709551615.
Bool	Vlera : të sakta ose të pasakta.
Float	Numra me pikë dhjetore .
Double	Numra të mëdhenj me pikë dhjetore.
Decimal	Numrat decimal.
Object	Tipi bazë i të gjitha tipeve të tjera.
Char	Përfshinë kodin UNICODE të karaktereve në mes të 0 dhe 65535.
String	Një sekuencë e karaktereve.

Storing Data- Mënyra se si ruhen të dhënat në memorien e kompjuterit. Varësisht se çfarë tipi të të dhënash deklarojmë atëherë kompjuteri rezervon hapësirën për atë tip të të dhënash.

Choosing a type- Ka të bëj me zgjedhjen e tipit të të dhënës varësisht prej asaj se çka do të përmbajë ajo e dhënë.

2.3. Deklarimi dhe inicializimi i variablave

Tek të dhënat e tipit integer (numra të plotë),mund të bëhet së pari deklarimi e më pas inicializimi:

```
int x;  //deklarimi
x=10; //inicializimi
```

Mirëpo mund të bëhet edhe deklarimi dhe inicializimi në të njëjtën kohë:

```
int x=10; //deklarimi dhe inicializimi
```

Kur dëshirojmë të bëjmë deklarimin dhe inicializimin e numrave decimal atëherë në fund të numrit decimal duhet të vendosim shkronjën M.

```
Decimal bankBalance=3433.20M
```

Visual Studio ofron mundësin e testimit të aplikacioneve urdhër për urdhër.
Ka disa metoda për realizimin e këtij testimi:

1. Kjo bëhet duke vendosur *Breakpoint* në reshtin në të cilin duam të testojmë. Vendosja e *Breakpoint* bëhet duke klikuar tek pjesa e hirtë në të majtë të urdhrit prej ku dëshirojmë ta testojmë dhe monitorojmë.
Një metodë tjetër është edhe përmes menysë kryesore *Debug* ku zgjedhim opsionin *New Breakpoint-Break at Function*, ose nëpërmjet tasteve shift+B.

2. Ekzekutoni aplikacionin.

3. Aplikacioni do të ekzekutohet deri tek urdhri në *Breakpoint* pastaj do të pauzohet dhe do të hijezohet urdhri që do të ekzekutohet.
Në dritaren *Autos* e cila ndodhet në fund të ekranit mund të shohim efektin e urdhrit.

2.4. Enumeration types (grupimi i variablave)

Shpeshherë kemi të bëjmë më të dhëna të tipin të njëjtë me vlerë fikse, atëherë C# na ofron mundësin që të dhënat e tipit të njëjtë t'i grupojmë në këtë formë:

```
Enum Planetet  // deklarimi i planetëve
{
   Merkuri,
   Venera,
   Toka,
   Marsi
}
```

Në këtë rast kompjuteri i jep vet vlera anëtarëve (në këtë rast *planeteve*). Nëse për llogaritje na nevojitet ndonjë e dhënë për tokën p.sh. masa e tokës atëherë ajo do të ketë një vlerë të cilën kompjuteri i ka dhënë automatikisht, mirëpo ne mund që të bëjmë edhe inicializimin duke i dhënë vlera të cilat dëshirojmë.

```
Enum Planet // inicialicimi i planetëve
{
   Merkuri=2434,
   Venera=5426,
   Toka=2464,
   Marsi=6475
}
```

Si dhe ne mund edhe t'ia caktojmë edhe tipin e të dhënave:

```
Enum Planet : unit // Përcaktimi i tipit të të dhënës
{
   Merkuri=2434,
   Venera=5426,
   Toka=2464,
   Marsi=6475
}
```

2.5. Konvertimi i tipeve të të dhënave

Kemi dy lloje shndërrimesh (konvertimesh) të të dhënave e ato janë:

- ***implicite dhe***
- ***eksplicite***.

Shndërrimi implicit është automatik dhe bëhet nga vet programi po ashtu nuk kërkon informacione nga jashtë. Shndërrimi eksplicit është shndërrim i cili kërkon që në mënyrë eksplicite të bëhet shndërrimi nga një tip në tipin tjetër.

Shndërrimi implicit:

```
int     x=1234566;  // deklarimi dhe  inicializimi i x-it
long   y=x; // konvertimi implicit
```

Shndërrimi eksplicit:

```
int  x=55;            //x=55
short z=(short) x;  //x-i konvertohet në short dhe i
                   //shoqërohet z.(eng. quhet CAST CONVERT)
```

2.6. Operatorët dhe shprehjet

Qëllimi i shkrimit të shprehjeve aritmetiko-logjike është që të kryejmë ndonjë veprim dhe të kthejmë një vlerë. Për shembull ato mund të jenë kalkulime matematikore, shoqërim të ndonjë vlere, apo krahasim i dy vlerave.
Shprehja është sekuencë e operandit dhe operatorit.
Operatori është një simbol që tregon veprimin që duhet të kryhet.
Operandi është vlera mbi të cilin kryhet veprimi.

2.6.1. Tipet e operatorëve

Operatorët mund të jenë matematikorë dhe logjikë.
Operatorët matematik kanë të bëjnë me llogaritjen e funksioneve të ndryshme matematikore.

Operatorët	Domethënia e tyre
x++	Përdore x-in për llogaritje, më pas rrite x=x+1.
++x	Rrite x-in për një, më pas përdore për llogaritje.
x--	Përdore x-in për llogaritje, më pas zbrite për një.
--x	Zbrite x-in për një, më pas përdore për llogaritje.
x+=y x-=y	x=x+y x=x-y
x*=y	x=x*y
x/=y	x=x/y
X%=y	x=x%y
x>>=y	x=x>>y
x&=y	x=x&y
x|=y x^=y	x=x|y x=x^y

2.6.2. Operatorët logjik

Operatorët logjik janë ata operatorë që tregojnë një vlerë të saktë ose të pasaktë. Operatorët logjik janë:

- **Operatori logjik "*ose*" (||),**
- **Operatori logjik "*dhe*" *(&&).***

Tek operatori logjik "***ose*" (||)** mjafton që njëra nga variablat të jetë e saktë dhe funksioni do të jetë ***true*** *(i v*ërtetë). Ndërsa tek operatori logjikë "***dhe" (&&)*** mjafton që njëra nga variablat të jetë e pasaktë dhe funksioni logjikë do të jetë ***false***.
Në C# variablat e tipit string (tekst) në mes veti lidhen me plus (+).
Për shembull:

```
String a = "mirë";
String b = "dita";
String c = a+b;
```

Rezultati:
mirëdita (pra kemi një rezultat i cili është bashkimi i dy stringjeve, pra mirë + dita)

2.7. Komanda për degëzim

Komanda e cila përdoret për degëzim është komanda **if** e cila ka këto forma:

1. *if else*
2. *if elseif else*

Komandë tjetër e cila përdoret për degëzim është edhe komanda **switch**.
Struktura e kësaj komande duket kështu:

```
switch (kushti)
 {
   case:
    urdhrat;
    break;
 }
```

2.7.1. Struktura e komandës if else

```
If (kushti == true OSE false)
{
  Urdhrat;
}
else
{
  Urdhrat;
}
```

Shembull. Të shkruhet programi që lexon orën e dhënë nga shfrytëzuesi dhe nëse ora e dhënë është më e madha se 12:00, të shtypet mesazhi “koha është pasdite”, përndryshe të shtypet mesazhi “koha është paradite”.

```
class Program
{
  static void Main(string ( ) args)
  {
    string Oras;
    double ora;

    Console.Write("te jepet ora ne formatin hh.mm:");

    Oras=Console.ReadLine();//lexohet ora si string
    ora = double.Parse(Oras);//konvertohet ora si string ne
                             // tipin double
    if (ora > 12.00) //nëse ora është më e madhe se 12 atëherë
    {                // ekzekuto këtë urdhër
      Console.WriteLine("koha është  pasdite");
    }
    Else            // përndryshe ekzekuto këtë urdhër
    {
      Console.WriteLine("koha është paradite");
    }

      Console.ReadKey();

    }
  }
}
```

2.7.2. Struktura e komandës if else if else :

```
if (kushti )
{
  Urdhri;
}
else if  (kushti tjetër)
{
  Urdhri;
}
else
{
  Urdhri;
}
```

2.8. Unazat (ang. loops)

Unazat (ang. loop) quhen strukturat kontrolluese përmes së cilave përsëriten pjesë të caktuara të programeve. Për realizimin e unazave në gjuhët programuese shfrytëzohen komandat for, while dhe do-while. Kurse unazat përkatëse zakonisht emërohen në bazë të këtyre komandave. Sikurse në gjuhët e tjera programuese edhe në C# përdoren këto unaza:

- ***for dhe for each***
- ***while***
- ***do while***

2.8.1 Unaza for

Unazat të cilat realizohen duke e shfrytëzuar komandën **for**, shkurt njihen si **unaza for**. Gjatë realizimit të unazave **for** mund të paraqiten disa raste karakteristike, të cilat njihen si: unaza të zakonshme, unaza të përbëra dhe unaza të ndërthurura.

Unazat for të zakonshme

Unazat brenda të cilave përfshihet vetëm një komandë mund të quhen unaza të zakonshme.

Struktura e kësaj unaze duket kështu:

```
for  (i=f; i<=p; i=i+h)
{
   u;
}
```

i- variabla e unazës.
f - vlera fillestare e variablës i.
p - vlera përfundimtare e variablës i.
h - hapi me të cilin ndryshohen vlerat e variablës i.
u - komandat të cilat ekzekutohen brenda unazës.

Unazat for të përbëra

Brenda unazës, e cila realizohet përmes komandës **for**, mund të përfshihen edhe më shumë komanda. Në këtë rast struktura e kësaj komande do të duket kështu:

```
for (i=f; i<=p; i=i+h)
{
  u1;
  u2;
  un;
}
```

Kllapat tregojnë se brenda trupit të unazës **for** ka një grumbull të komandave.

Shembull. Të shkruhet programi i cili na mundëson shkruarjen e fjalës *Cacttus* dhjetë herë.

```
class Program
{
  static void Main(string ( ) args)
  {
     int i;
     int n=10;
     string emri="Cacttus";
     for (i = 0; i <= n; i++)
     {
        Console.WriteLine(" {0} ",emri);
     }

     Console.ReadKey();

  }
}
```

Shembull.
Të shkruhet program i cili mundëson llogaritjen e tabelës së shumëzimit.

```
class Program
{
  static void Main(string ( ) args)
  {
    int i, j;
    Console.Write("deri sa dëshironi te llogaritni”);
    Console.Write(" tabelën e shumëzimit:");
    int n = int.Parse(Console.ReadLine());
    for (i = 1; i <= n; i++)
    {
      Console.WriteLine("tabela e shumëzimit me {0}\n", i);
      for (j = 1; j <= 10; j++)
       {
        Console.WriteLine("{0} * {1} = {2}", i, j, i * j);
       }
      Console.WriteLine(“\n”);
    }

    Console.ReadKey();
  }
}
```

2.8.2. Unaza while

Përveç unazës **for,** një unazë tjetër e cila përdoret është edhe unaza **while**. Struktura e kësaj unaze do të duket kështu:

```
i=f;
while(i<=p)
{
  a;
  i=i+h;
}
```

i - variabla e unazës.
f - vlera fillestare e variablës së unazës.
p - vlera përfundimtare e variablës së unazës.
h - hapi me të cilin ndryshohen vlerat e variablës i.
a - komandat e përfshira brenda unazës.

Shembull. Të shkruhet program i cili mundëson llogaritje e shumës së numrave nga 0 deri në 10.

```
class Program
{
  static void Main(string ( ) args)
  {
    double s;
    int i;
    s = 0;
    i=0;

    while (i<=10)
    {
      s = s + i;
      i++;
    }

    Console.WriteLine("Shuma është {0}",s);
    Console.ReadKey();
  }
}
```

2.8.3.Unaza do while

Struktura e unazës **do while** është kjo:

```
i=f;
do
{
 a;
 i=i+h;
}
while(i<=p);
```

Ku variablat paraqesin d.m.th. të njëjtë sikurse tek unazat e mëparshme.

Shembull. Të shkruhet program i cili mundëson llogaritjen e tabelës së shumëzimit duke përdorur komandën **do while**.

```
class Program
{
  static void Main(string ( ) args)
  {

    int i, int j;
    i=1;
    do
    {
      Console.WriteLine("Tabela e shumëzimit me {0} \ni  j  i*j", i);
      j = 1;
      do
      {
        Console.WriteLine("i={0}    j={1}   {2}", i, j, i * j);
        j++;
      }
      while (j <= 10);
      i++;
      Console.WriteLine("\n");
    }
    while (i <= 10);


    Console.ReadKey();

  }
}
```

Moduli 3: Definimi dhe përdorimi i Klasave

Ky modul ka të bëjë me kuptimin e koncepteve të programimit të orientuar në objekte, po ashtu edhe kuptimin e koncepteve objekt, klasë dhe metodë. Më pas do të mësohet se si do të definohen klasat dhe se si duhet të krijohen objektet, po ashtu edhe organizimin e klasave duke përdorur *namespace-at*.

Objektivat
Pas kompletimit të këtij moduli, ju do të jeni në gjendje të:

- definoni klasat
- deklaroni metoda
- përdorni konstruktorët

3.1. Klasat dhe objektet

Kur flitet për *programimin e orientuar në objekte* (*programimin me objekte)*, gjithnjë mendohet në *klasat* si dhe në *objektet* që deklarohen dhe në shfrytëzimin e tyre.
Klasat janë fundamentet për definimin e tipit të të dhënave në C#. Së pari duhet të definoni klasat para se të krijojmë objektet. Klasa është tërësia e një projekti, prej të cilës ju mund të krijoni objekte.
Në klasë duhet definuar karakteristikat e një objekti, që përmban cilësinë e definimit të tipit të të dhënave tek objekti, dhe metoda që duhet të përdoren në atë klasë.
Klasa është një përshkrim për një artikull, objekti është një artikull. Një shembull tjetër: Projekti i shtëpisë në letër është klasa ndërsa shtëpia është objekti.
Përmes klasave jepet mundësia e shkruarjes së programeve, të cilët sipas nevojës, lehtë ndryshohen, duke i ndryshuar vetëm klasat. Gjatë kësaj, problemi që zgjidhet copëtohet në klasa dhe në deklarimin e objekteve përkatëse, gjë që ka një rëndësi të veçantë kur kemi të bëjmë me programe komplekse, me çka zvogëlohet mundësia e gabimeve.

3.2. Definimi i klasave dhe krijimi i objekteve

Me qëllim të lehtësimit të punës me grumbuj të të dhënash të tipit të njëjtë, të cilat njihen edhe si të dhëna *homogjene*, shfrytëzohen fushat (vektorët, matricat ose fushat shumëdimensionale). Por, në gjuhën C# mund të grupohen edhe të dhëna të tipeve të ndryshme, përkatësisht të dhëna *heterogjene*, duke krijuar tipe të reja të të dhënave, të cilat njihen si klasa.
Forma e përgjithshme e definimit të klasave është:

```
public class Customer
{
    public string name;
    public decimal creditLomit;
    public int customerID;
}
```

Ne mund që klasat t'i deklarojmë private ose publike. Klasave publike mund t'i qasemi edhe nga klasat e tjera, ndërsa atyre private nuk mund t'i qasemi nga klasat e tjera.
Këtu e kemi të deklaruar një klasë me emrin *Customer* me pas brenda klasës kemi dhënë informata për konsumatorin ose siç njihen si komponente të klasës. Ndërsa para komponenteve të klasës deklarohen tipet e të dhënave në komponentet e klasës.
Me fjalën **public** të shënuar para komponenteve të klasës, atyre mund t'u qasemi dhe t'i shfrytëzojmë në program. Fjala **public**, e cila njihet si *specifikuesit e qasjes*. Nëse te klasa nuk shfrytëzohet specifikuesin **public**, ai do të nënkuptohet nga kompjuteri si **private**, dhe qasja direkte nga jashtë është e pamundshme.

Pas definimit të një klasë kompjuteri nuk rezervon vende në memorie për komponentet që përfshihen brenda klasës, pavarësisht se deklarohen tipet e variablave përkatëse. Por, me klasën krijohet një tip i ri, i cili pastaj mund të shfrytëzohet për deklarimin e objekteve të asaj klase. Definimi i klasës tregon vetëm se si objekti duket, kurse pas deklarimit në program, krijohet ***objekti i klasës***, ose siç thuhet krijohet ***instanca e klasës,*** e atëherë bëhet rezervimi i hapësirës memoruese**.**

3.3. Organizimi i klasave duke përdorur Namespaces

Namespace përdoren për organizimin e klasave brenda një hierarkie të lidhur logjike. Funksioni namespace po ashtu përdoret si sistem i brendshëm për organizimin e aplikacionit tuaj dhe po ashtu si rrugë e jashtme për shmangien e *name clashes (konflikteve)* ndërmjet kodit tuaj dhe aplikacionit tjetër.
Namespace na ofron mundësinë e krijimit të klasave me emër të njëjtë. Për shembull një kompani mund të krijojë klasa me të njëjtin emër. Namespace është sistem i organizuar i cili mund të përdoret për identifikimin e klasave të lidhura grup.
Për të krijuar një namespace, thjeshtë duhet të shtypni një hapësirë, *namespace* pasuar me një emër.
Ne mund që të përdorim më shumë se një *namespace* dhe të bëjmë ndërlidhjen në mes tyre. Në shembullin në vijmë mund të shohim organizimin e një klase duke përdoru namespace.

```
namespace CompanyName.Sales
{
//definimi i klasës brenda këtij namespace

        public class Customer( )
        {

        }
}
```

Ndërsa në shembullin tjetër kemi përdorur dy namespace.

```
namespace CompanyName
 {
    namespace Sales
    {
        public class Customer()
        {

        }
    }
 }
```

Që të dy shembujt, mund të përdorin kodin si në vijim:
CompanyName.Sales
Ky është saktësisht emri i komponentës për klasën *Customer.*

Për klasën *customer* mund të përdorni plotësisht emra të komponentës që ju referohet specifikave të klasës *customer* dhe t'iu shmangeni *name collisions* (konflikteve) me klasat *customer* të tjera.
Microsoft .Net Framework është formuar për më shumë namespace për të cilin me më rëndësi është emri **System.** Namespace **System** përmban klasa me shumë aplikacione të cilat përdoren kur bashkëveprojmë me sistemin operativ.
Për shembull namespace **System** përmbajnë klasat **Console**, të cilat ofrojnë disa metoda, përfshirë **WriteLine**, e cila është një komandë që jep mundësinë e shkruarjes së kodit në një ekran **Console** si më poshtë:
system.Console.WriteLine("Hello, World");

3.4. Metodat

Sikurse edhe gjuhë e tjera programuese edhe C# ofron mundësin e përdorimit të bibliotekave të gatshme, të përcaktuara nga kompajlleri i gjuhës programuese C#. Mirëpo edhe vet shfrytëzuesi mund të përcaktoj funksione të ndryshme të cilat i përdor sipas nevojës. Shpesh ndodh që brenda programit mund të përsëriten një grumbull i caktuar i komandave. Me qëllim të eliminimit të përsëritjes së këtyre komandave shumë herë brenda programit, bëhet grumbullimi i komandave që përsëriten dhe deklarohen si pjesë e veçantë e cila quhet metodë. Më pas kjo metodë emërtohet me një emër dhe thirret sa herë që ne kemi nevojë nga funksioni përkatës.

Forma e përgjithshme e metodës duket kështu:

```
public tipi emri_i_metodës(tipi i të dhënës parametri1,
                        tipi i të dhënës parametri2)
{
  string veprimi =(parametri1+parametri2).ToString();
}
```

Metoda për dallim nga funksioni nuk kthen vlerë.

Shembull. Të shkruhet program i cili mundëson llogaritjen e funksionit që pranon tre parametra dhe që kthen vlerën string.
Funksioni ka këta parametra:
double vlera1,
double vlera2, dhe
string operatori.
Varësisht prej operatorit të kryhen funksionet themelore aritmetike.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Matematika
    {
        public static string Aritmetika(double v1, double v2,
                                         string operatori)
        {
            double rezultati = 0;
            switch (operatori)
            {
                case "+":
                    rezultati = v1 + v2;
                    break;
                 case "-":
                        rezultati = v1 - v2;
                        break;

                case "*":
                    rezultati = v1 * v2;
                    break;

                case "/":
                    rezultati = v1 / v2;
                    break;
                default:
                    rezultati = 0;
                    break;
            }
            return rezultati.ToString();

        }

        public static void Mbledhja(double v1, double v2,
                                    string operatori)
        {
            string mbledhja = Matematika.Aritmetika(v1, v2,
                                                    operatori)
            Console.WriteLine("Mbledhja = " + mbledhja);

        }
        public static void Zbritja(double v1, double v2,
                                   operatori)
        {
            string mbledhja = Matematika.Aritmetika(v1, v2,
                                                    operatori)
            Console.WriteLine("Zbritja = " + mbledhja);

        }

    }
}
```

Mirëpo ne këtë klasë qe e kemi krijuar mund ta përdorim edhe në ndonjë program tjetër, për shembull në rastin kur na nevojitet këto veprime aritmetikore atëherë ne vetëm e thërrasim emrin e klasës dhe emrin e metodës si në shembullin e më poshtë.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication2
{
    class Program
    {

      static void Main(string() args)
        {
            Matematika.Mbledhja(12.3, 4.6, "+");
            Matematika.Zbritja(1, 2, "-");

            Console.ReadLine();
        }
     }
}
```

3.5. Konstruktorët

Me qëllim të inicializimit të variablave të cilat përfshihen në komponentet e klasës, brenda saj definohen funksione të veçanta, të cilat njihen si *konstruktorë*.
Këto funksione ekzekutohen automatikisht, kur deklarohen objektet përkatëse të klasave në fjalë. Konstruktorët dallohen nga funksionet e zakonshme brenda klasës, sepse *kanë emra të njëjtë me klasat* dhe para emrave të tyre si dhe te komandat e fundit **return** nuk shënohet asgjë. Konstruktorët mund të kenë ose mund të mos kenë parametra formalë.
Konstruktorët pa parametra formalë - nëse përcaktimi i vlerave të variablave në komponentet private bëhet brenda konstruktorëve, ato mund të mos kenë parametra formalë.
Konstruktorët me parametra formalë - përmes konstruktorëve, si edhe përmes funksioneve të zakonshme, variablave brenda klasës mund t'u shoqërohen edhe vlera të çfarëdoshme.
Përveç inicializimit të variablave me vlera, brenda konstruktorëve mund të kryhen edhe llogaritje të ndryshme. Por, rezultatet e llogaritjeve duhet të përcillen te variablat e përfshira në komponentet e klasave. Brenda një klase mund të përfshihen edhe disa konstruktorë njëkohësisht. Kjo lidhet me krijimin e disa objekteve të klasës në fjalë, ashtu që objektet e ndryshme të mund ta shfrytëzojnë njërin nga konstruktorët e definuar.
Nëse konstruktorët e përfshirë në definicionin e klasës dallohen mes vete për nga numri i parametrave, përfshirë edhe konstruktorin pa parametra, kompjuteri do ta thërrasë automatikisht konstruktorin i cili ka aq parametra.

3.6. Çfarë janë Properties

Që të kemi një menaxhim sa më të mirë të qasjes së anëtarëve të klasës nga objektet e asaj klase përdorim *properties.*
Sintaksa për të definuar një *properties* konsiston në modifikatorin e qasjes (publike, private), tipin e emrit, si në kodin e më poshtëm:

```
get
{
}
set
{
      // Karakteristikat e kodit get
}
```

Get dhe **Set** quhen qasës (accessor).
Get kthen (return) vlerë me tip të të dhënave të njëjtë me tipin e *properties*.
Set është ekuivalent me një metodë që e ka një parametër implicit (brendshëm) të quajtur ***value***.
Në kodin në vijim është ilustruar shembull i përdorimit të kësaj komande:

```
using system;
namespace LearningCSharp
{
    class Elephant
    {
        public decimal DailyFoodInake
        {
            get
            {
                return dailyCosumptionRate;
            }
            set
            {
                if (value < dailyCosumptionRate - 25)
                {
                    //notify medical center
                }
                else
                {
                    dailyCosumptionRate = value;
                }
            }
        }
    }
    class Zoo
    {
        static void Main(string[] args)
        {
            Elephant e = new Elephant();
            e.dailyCosumptionRate = 300M;
        }
    }
}
```

Moduli 4: Vektorët (Arrays)

Në këtë modul do të paraqesim disa struktura të të dhënave që përfshin vektorët (klasat ***System.Array***), koleksionet (klasat në namespace ***System.Collections***) si dhe shembuj se si do t'i përdorim vektorët në aplikacion.
Fushë numerike quhet grumbulli i numrave të vendosur në një hapësirë në bazë të parimeve të caktuara. Kur pozicionet e numrave në hapësirën e fushës numerike përcaktohen nga një madhësi, për fushën thuhet se është njëdimensionale dhe quhet vektor. Por për pos kësaj për përcaktimin e pozicioneve të numrave mund të përdoren dy madhësi, e kur fusha numerike është dydimensionale kemi të bëjmë me vektor multidimensional apo matricë.

Objektivat
Pas kompletimit të këtij moduli, ju do të jeni në gjendje të:

- Krijoni dhe t'i përdorni vektorët
- Përdorni klasat në namespace-in ***System.Collection***
- Përdorni klasat ***ArrayList***

4.1. Vektorët (Arrays)

Vektorët janë objekte dhe një vektor mund ta paradefinojmë si një listë.
Duke përdorur vektorët mund të bëjmë një grup të elementeve që kanë të dhëna të llojit të njëjtë, me një emër të një variable. Një vektor mund të jetë një zgjidhje e mirë kur dëshirojmë të krijojmë një listë parametrash. Vektori është një strukturë e të dhënash që përmban një numër të variablave që janë të njohura si elemente të vektorit. Për t'iu referuar elementeve specifike në këtë seri duhet të përdorim një indeks tregues. Në C# pozita e parë definohet me indeks zero ndërsa pozitat tjera rriten për një .

[0]	[1]	[2]	[3]	[4]	[5]	[6]

Indeks 0 *Indeks 6*

Vektori i cili ka një listë të vetme është i njohur si vektor njëdimensional. Ndërsa një vektor i cili ka më shumë se një dimension quhet vektor multidimensional, këta vektor multidimensional në gjuhën C++ njihen si matrica. Vektori multidimensional paraqet më shumë se një vlerë. Vektorët multidimensionalë të madhësive specifike shpesh i referohen madhësive si vektor dydimensional dhe tre-dimensional. Ne mund të mendojmë vektorin dy-dimensonal si bosht koordinativ. Për shembull ne mund të paraqesim vektorin dy-dimensional në boshtin koordinatave me koordinata x dhe y.
C# ofron ndihmën për vektorët duke u bazuar në klasat ***System.Array*** të Microsoft.Net Framework. ***System.Array*** është një klasë abstrakte e cila ofron shumë metoda të cilat mund të përdoren gjatë punës me vektor. Tabela më poshtë tregon metodat të cilat përdoren zakonisht gjatë punës me vektor.

Metoda	**Përshkrimi i metodës**
Sort	Renditja e elementeve brenda një vektori.
Clear	Cakton rangun e elementeve që nuk janë zero ose **null.**
Clone	Krijon kopjen tek vektorët.
GetLength	Rikthen madhësitë e elementeve të dhëna.
IndexOF	Indeksimi i anëtarëve të vektorit
Length	Ofron numrin e elementeve më dimensione specifike të vektorit.

4.2. Krijimi i vektorëve

Para se t'i përdorim vektorët, ne duhet t'i krijojmë dhe t'i deklarojmë ata.
Forma e përgjithshme e deklarimit të vektorëve është:

```
Tipi [] emri_i_ vektorit = new DataTypes {elementet e vektorit};
```

Ndërsa tek vektorët multidimensional :

```
Tipi[][] emri_i_vektorit =
new Tipi {el.e vektorit 1},{el.e vektorit 2};
```

Për shembull një vektor i cili përmban dy numra, atë e deklarojmë si integjer me një emër të caktuar në rastin tonë *numri*.

```
int [ ] numri = new int {1,2};
```

Mirëpo deklarimin dhe inicializimin e vektorit mund të bëjmë edhe në një mënyrë tjetër duke ia caktuar numrin e anëtarëve të cilët marrin pjesë në vektor, si në shembulli e vijim.

```
int [ ] numri = new int [5] {1,2,4,5};
```

Nga shembujt e deklarimit dhe inicializimit të vektorëve mund të shohim se anëtarët e vektorit në mes veti ndahen me presje. Vektori mund të jetë edhe i tipit *string,* andaj edhe deklarohet si string. Kur kemi vektor të tipit *string* anëtarët brenda vektorit futen brenda thonjëzave të dyfishta. Si shembull mund të marrim numërimin e stinëve të vitit, e në këtë rast e deklarojmë një vektor të tipit string me një emër *stina*.

```
string[] stina =
new string {"pranvera","vera","vjeshta", "dimri"};
```

Unazë (*loop-a*) e cila përdoret tek vektorët është unaza ***foreach.***
Shembull. Të shkruhet programi i cili mundëson leximin e anëtarëve të një vektori.

```
class Program
    {
        static void Main(string [ ]   args)
        {
           int  [ ] vektori=new int [ ] {0,1,2,3,4,5};

            foreach(int i in vektori)
            {
                Console.WriteLine(i);
            }
         Console.ReadKey();

        }
     }
```

Shembull. Të shkruhet programi i cili mundëson leximin e lëndëve dhe notave të një nxënësi:

```
class Program
{
  static void Main(string  [ ] args)
  {
    string [ ] lendet = new string  [ ]
    {"Matematike","Shqip","Anglisht","Fizike","Kimi"};

    int  [ ]Notat=new int [ ]  {3,5,4,2,4};
    int j = 0;
    foreach(string i in lendet)
    {
      Console.WriteLine("{0}      {1}",i, Notat [j ]);
      j++;
    }
    Console.ReadKey();

  }
}
```

Shembull. Të shkruhet programi i cili mundëson ndërtimin e një vektori C, nga shuma e vektorit A dhe B.

```
class Program
{
  static void Main(string  [ ] args)
  {
    int  [ ] vektori_A = new int [ ] {1,2,3};
    int  [ ]vektori_B= new int [ ]  {4,2,4};
    int [3] vektori_C;
    for (int  i=0; i<vektori_A.lenght; i++ )
    {
      vektori_C[i]= vektori_A[i] + vektori_B[i];
      Console.WriteLine(" shuma e kërkuar është= {0}  ", (
      int.Parse( vektori_A[i].ToString( )) +
      int.Parse(vektori_B[i].ToString())).ToString()   );

    }
    Console.ReadKey();

  }
}
```

4.3. Përdorimi i ArrayList

Një problem i cilin na paraqitet tek vektorët është se kapaciteti i strukturës së të dhënave duhet të dihet paraprakisht para se t'i japim vlera anëtarëve të atij vektori. ***ArrayList*** zgjedh këtë problem. Ky problem zgjedhet duke ofruar një strukturë të të dhënave sikurse në një vektor, mirëpo kapaciteti i së cilës mund të ndryshoj sipas nevojës. Sa herë që shtohen elemente të reja në një objekt të ***ArrayList*** atëherë kapaciteti rritet automatikisht. Fillimisht objekti zë 16 elemente, kur shtohet një elementi i 17 ***ArrayList*** zgjerohet në 32 elemente e kështu me radhë.
Qasja në elementet e ArrayList - Qasja në elementet e ***ArrayList*** bëhet sikurse qasja në elementeve të një vektori. Gjithashtu mund të përdorim edhe metodat e ***ArrayList*** për të shtuar ose për të fshirë (larguar) ndonjë element prej ***ArrayList***. Për ta zvogëluar kapacitetin e ArrayList duhet të thirret metoda ***Trim ToSize.*** Gjithashtu përdoret unaza ***foreach*** për të manipuluar me elemente e ***ArrayList***.

Vërejtje: Elementet e ***ArrayList*** janë të tipit objekt, prandaj kur dëshirojmë që t'i paraqesim këto elemente prej listës duhet patjetër ta konvertojmë tipin e tyre. Disa nga metodat të cilat përdoren tek ***ArrayList*** janë:

Metoda	Përdorimi
Add	Shtimi i objekteve në ***ArrayList***.
Remove	Largimin e objekteve nga ***ArrayList***.
Clear	Fshirjen e objekteve nga ***ArrayList***.
Insert	Insertimin e objekteve me indeks të specifikuar.
Trim ToSize	Vendos kapacitet me numrin aktual të elementeve.
Sort	Renditjen e elementeve në ***ArrayList***.
Reverse	Ndërrimi i pozitave të parë me pozitën e fundit dhe e kundërta, të elementeve në ***ArrayList***.

Moduli 5: Ndërlidhja me databazë

Ky modul shpjegon se si ta përdorim *Microsoft ADO.NET* dhe objektet e namespace *System.Data* si dhe si t'i përdorim të dhënat në databazë. Këtu përshkruhen aplikimet bazike në Microsoft Windows duke përdorur *ADO.Net*. Ky model gjithashtu përshkruan se si të përdorim aplikacionin duke e lidhur me databazë, se si të krijojmë procedura, dhe t'i kontrollojmë të dhënat duke përdorur objektet *DataSet*, të lidhim të dhënat dhe të kontrollojmë ato, t'i shtojmë, bashkojmë, t'i largojmë, t'i regjistrojmë të dhënat në databazë etj.

Objektivat

Pas kompletimit të këtij moduli, ju do të jeni në gjendje të:

- Krijoni aplikacione bazë në Windows duke përdorur *ADO.Net*.
- Lidhni C# me ndonjë databazë.
- Krijoni procedura.
- Kontrollojmë të dhëna duke përdorur objektet e *DataSe*t.
- Lidhni objektin *DataGrid* me data source.
- Shtoni, bashkoni, fshini dhe të regjistroni të dhënat në databazë.

5.1. Teknologjia ADO.NET

ADO.Net është një teknologji e cila na ofron shumë mundësi siç janë:
Kompletimin e klasave, ndërlidhjet, strukturat dhe grumbullimin si dhe kontrollimin e të dhënave.
Përparësia e modeleve të programimit në përkrahjen e XML. *ADO.Net* lejon që XML të ketë përdorim më të gjerë.
Ado.Net dhe *Windows Forma* i ofrojnë klientit komponente me të dhëna që ju mund t'i përdorni për menaxhimin e të dhënave tuaja. Këto komponente përfshijnë kontrolla të tilla siç është *DataGrid*, ku ju mund të shfaqni të dhënat me anë të *TextBox*-a*ve*, *Label-ave*, *ControlBox-ave* dhe kontrollave të tjera.
Pjesët më të rëndësishme të ADO.Net janë:

- ***Klasa .NET data provider***
- ***Klasa Data.set.***

.NET data provider ofron klasa që janë specifike për *data source*. Prandaj *.NET data* ofron shkrime specifike që duhet të gjenden në *data source* dhe do të punonin direkt me data source. .NET data ofron mundësinë që klasat të lidhen me data source.

ADO.Net përfshin modelet siç janë:
SQL Server.NET Data Provider.
OLE DB.NET Data Provider.

5.2. Klasa DataSet

ADO.Net DataSet klasa është bërthama e komponenteve për veçoritë e arkitekturës *ADO.Net*. **DataSet** objekti i *namespace System.Data* është një server virtual që është një vend i fshehtë në databazë, përfshinë jo vetëm të dhëna por edhe skema, tabela me të dhëna, etj.
Shpesh ne kemi nevojë që t'i shfrytëzojmë vetëm disa të dhëna të caktuara në databazë, e këtë na ofron DataSet. DataSet lejon që ju të merrni vetëm të dhënat që juve ju nevojiten në kohë të caktuara. DataSet ofron një koleksion të tabelave dhe po ashtu edhe lidhjeve në mes të tyre. Pra të dhënat e një tabele mund t'i lidhim me të dhënat e një tabelës tjetër. Mirëpo të dhënat në një tabelë mund të shkëmbehen edhe në më shumë se një tabele jo vetëm me një tabele.

5.3. Krijimi i objekteve DataAdapter-ave

Kur dëshirojmë që nga C# të krijojmë lidhje me një databazë, databazë kjo që mund të jetë në ndonjë program tjetër siç është për shembull një databazë e punuar në *Microsoft SQL Server* apo Microsoft Office Access atëherë nevojitet një ndërmjetësues në mënyrë që të dhënat të vendosen në të, e më pas në *Data source*. E ky ndërmjetësues është emërtuar si *DataAdapter*, i cili shërben si urë ndërmjet *DataSet* dhe *data source*. *DataAdapter* përfaqëson komplet lidhjen e databazës, si dhe databazës që ju duhet të përdorni për t'u lidhur me *DataSet* dhe përzgjedhjen e *data source*. *DataAdapter* është pjesë e shërbimeve të *.NetData*, gjithashtu përfshinë lidhjen e objektit me objektin *DataSet*. Një mundësi tjetër është po ashtu edhe shkëmbimi i të dhënave ndërmjet një objekti *DataTable* me

DataSet. Pasi që të krijojmë lidhjen me databazë, atëherë ne mund që t'i insertojmë (***insert***) ato të dhëna, më pas t'i fshijmë (***delete***) disa të dhëna të caktuara pasi që t'i kemi selektuar (zgjedhur), por të dhënat që fshihen në C# do të shkojnë në *DataAdapter* e më pas edhe në databazë, pra me fshirjen apo ndryshimin e ndonjë të dhëne, atëherë e dhëna e ndryshuar automatikisht do të vendoset në databazë përmes *DataAdapter - it*.

5.4 Dizajnimi i Formës

Në kuadër të një projekti puna e parë e cila duhet të realizohet është dizajnimi i windows formave. Për dizajnimin e windows formave duhet të përdorim *label*-a, *textbox*-a, *combobox*-a, *listbox*-a, DataGridView e cila përdoret për paraqitjen e shënimeve të cilat ndodhen në databazë, po ashtu edhe të dhënat të cila ne i regjistrojmë, dhe veglat tjera që i ofron Visual Studio. Të gjitha këto elemente gjenden në dritaren **ToolBox** e cila gjendet në të majtë të dritares kryesore të C#.
Kontrollat (controls)-janë objekte grafike si p.sh. kutitë e tekstit (text box), butonat etj, të cilët vendosen në formë për të paraqitur të dhëna, apo për të realizuar ndonjë veprim, ose për të lehtësuar përdorimin e formës. Ne do të tregojmë funksionin e kontrollave kryesore.
TextBox Control - Shpesh quhet fushë editimi, paraqitje ose kontrolli i botimit (edit control). Ky kontroll paraqet informacionin që mund të futet gjatë kohës kur po krijohet aplikacioni (design time), apo gjatë kohës së ekzekutimit të programit (run time). Gjithashtu në text box mund të paraqitet informacioni që mund të përcaktohet nga kodi i programit.
Label Control- është një kontrollë grafike që përdoret për të vendosur tekst në ndërfaqen tuaj. Ky tekst nuk mund të ndërrohet nga përdoruesi i programit në run time (gjatë kohës së ekzekutimit).
ListBox Control- Përdoret për të afishuar një liste me artikuj, nga e cila përdoruesi mund të zgjedhë një ose më tepër prej tyre. Nëse numri i elementeve të listës është më i madh se mund të paraqes lista, atëherë automatikisht do të shtohet.
ComboBox Control- është një kombinim i karakteristikave të një text box dhe një list box. Një përdorues, nëpërmjet combo box mund të zgjedhë të dhënat nga një listë dhe t'i vendos në text box. Këto kontrolla janë treguara në figurën e më poshtme.

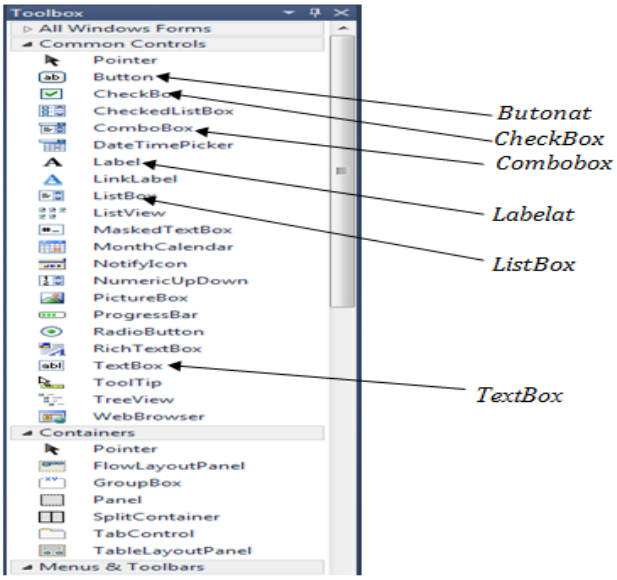

Fig 7.1. Pamja e dritares ToolBox

Tani do të shqyrtojmë rastin e lidhjes së C# me një databazë e cila gjendet në SQL të cilën më parë e kemi krijuar dhe e kemi emërtuar me emrin Fakulteti. Forma startuese do të duket kështu si më poshtë.

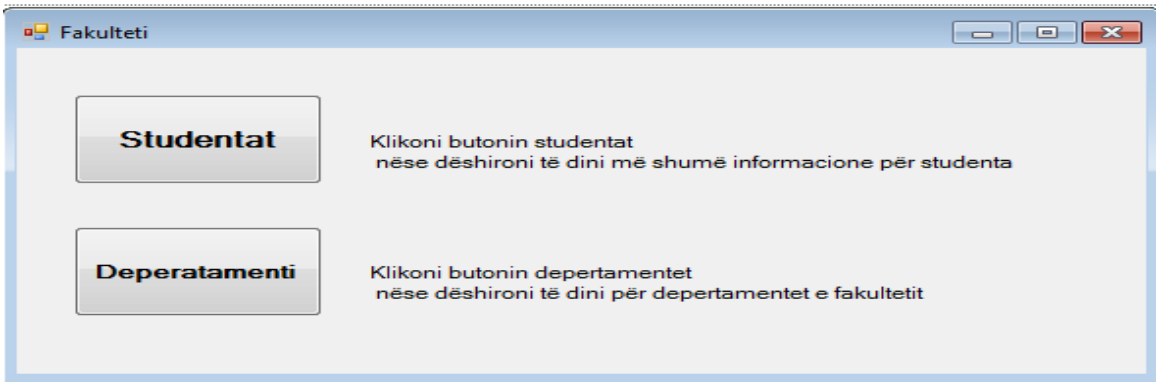

Fig 7.1. Pamja e dizajnit të windows formës kryesore

Detyra jonë është që kur të klikojmë për shembull tek butoni **studentët** atëherë në DataGrid të na shfaqen tabela me emrat e studentëve, që ne e kemi krijuar ne SQL, përpos kësaj ne kur klikojmë tek butoni studentët të na jepet mundësia që të regjistrojmë studentë të ri dhe këta studentë të regjistrohen automatikisht në databazë. Po ashtu edhe për butonat e tjerë kur të klikojmë të na shfaqen të dhënat adekuate. Po ashtu ne duhet që të krijojmë lidhje në mes të tabelave. Së pari ne duhet të krijojmë klasa si dhe procedura të cilat më pas i thërrasim varësisht se për çfarë kemi nevojë.
Kur klikojmë butonin studentët do të na hapet kjo dritare të cilën e kemi emërtuar me emrin studentët. Këtë dritare mund ta dizajnojmë në forma të ndryshme por tani do të dizajnojmë këtë formë.

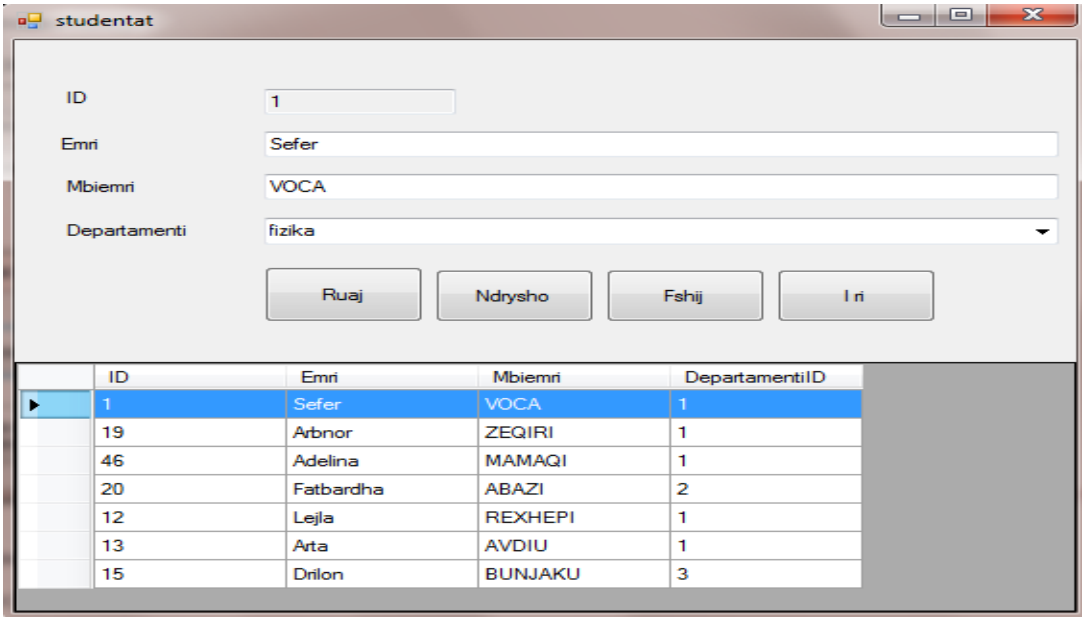

*Fig 7.3. Pamja e Windows Formës pasi të shtypim butonin **studentët**.*

Ndërsa pasi të klikojmë tek butoni **departamenti** do të na hapet kjo dritare si në figurën e mëposhtme.

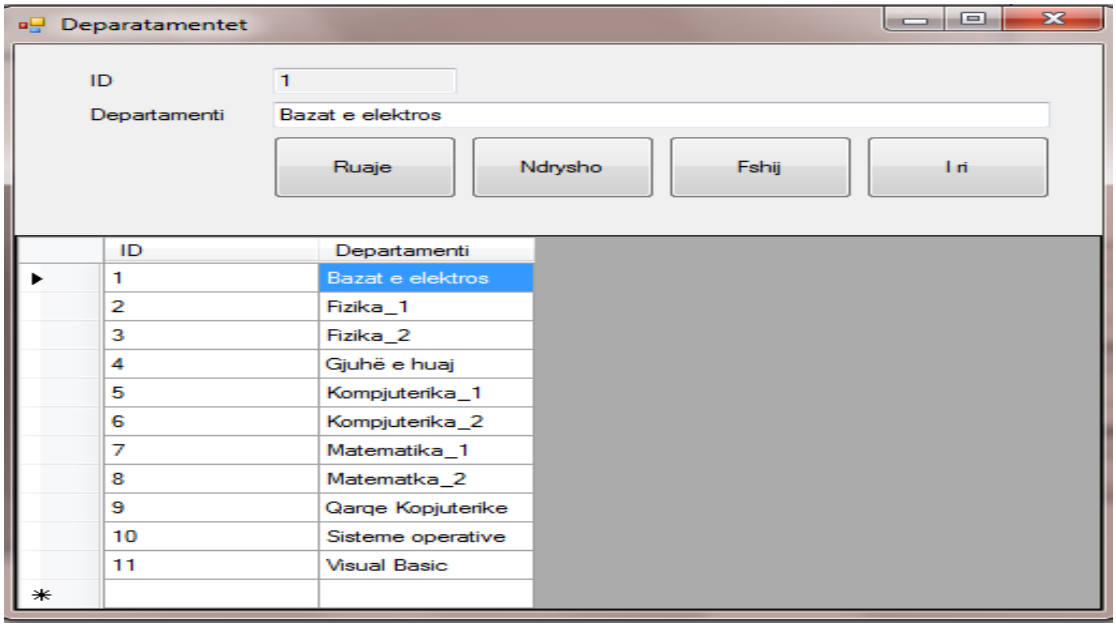

*Fig 7.4. Pamja e Windows Formës pasi të shtypim butonin **Departamenti**.*

Nga figurat mund të shohim se kemi përdorur butona, labela, comobox dhe dataGrid. Tek dritarja *properties* e cila gjendet në anën djathtë të ekranit tek opsioni *Text* shkruajmë tekstin të cilin dëshirojmë të na shfaqet kur klikojmë në butonin përkatës, ndërsa tek opsioni *Name* shkruaj emrin të cilin do ta përdorim gjatë shkruarjes së kodit.

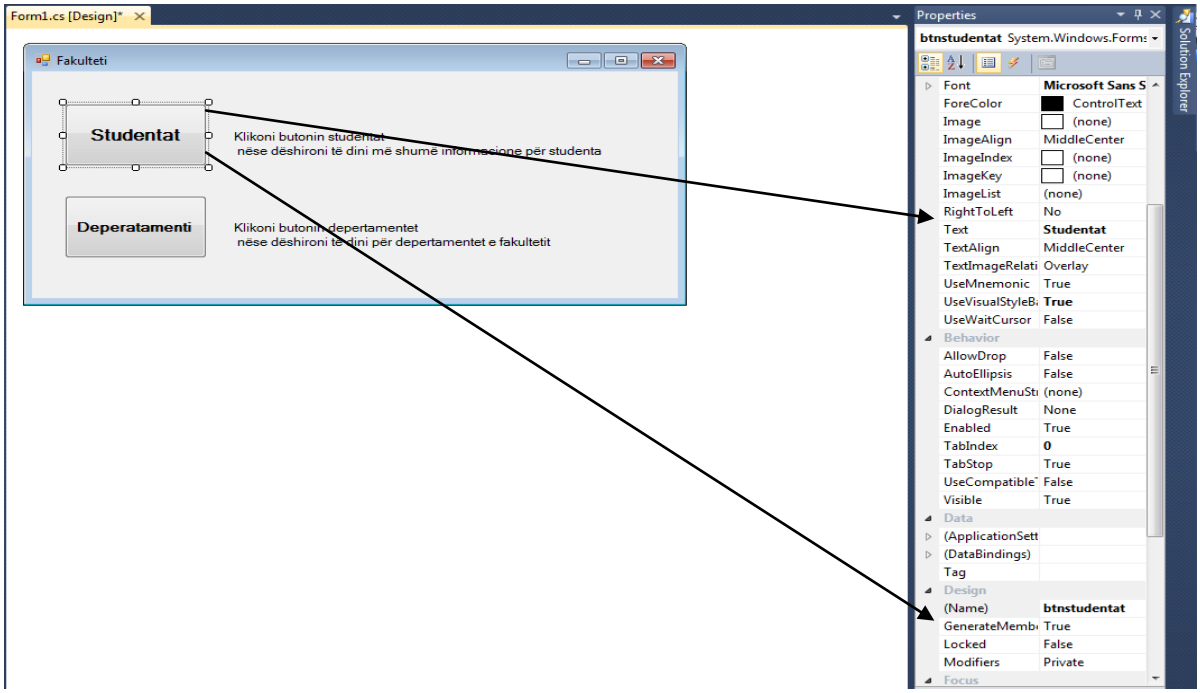

Fig 7.5. Pamja e dritares Properties.

Pra këto janë dizajnet të cilat do të na paraqiten kur klikojmë butonat përkatës. Mirëpo tani e kemi dizajnuar vetëm formën se si do të shfaqen të dhënat. Për t'u shfaqur këto të dhëna në dataGrid duhet që të shkruajmë kod. E për të realizuar këtë duhet të përdorim klasat, metodat si dhe procedurat. Klasat dhe metodat i krijojmë në C# ndërsa procedurat në SQL, më pas varësisht se kur kemi nevojë t'i përdorim ato i thërrasim në bazë të emrit që i kemi emërtuar më parë. Gjatë krijimit të procedurave ne SQL duhet ditur këto komanda, përdorimi i të cilave është përshkruar në tabelën e më poshtme.

Komanda	**Përdorimi**
Insert	Përdoret për të futur të dhëna në databazë.
Select	Përdoret për selektimin e të dhënave në databazë.
Delete	Përdoret për fshirjen e të dhënave në databazë.
Update	Përdoret për ndryshimin e të dhënave të caktuara në databazë.

Procedura për t'i ruajtur të dhënat e studentëve:

```
create PROCEDURE [dbo].[Ruaj_Studentin]
@Emri AS VARCHAR(100),
@mbiemri AS VARCHAR(100),
@DepartamentiID AS INT
AS
BEGIN
INSERT INTO Studentat
(emri,mbiemri,DepartamentiID) VALUES
(@Emri,@mbiemri,@DepartamentiID)
END
```

Procedura për t'i shfaqur të dhënat e studentëve:

```
create procedure [dbo].[shfaq_studente]
as
begin
select * from Studentat order by emri
end
```

Procedura për t'i fshirë të dhënat e studentëve:

```
create procedure [dbo].[fshij_studentat]
@ID as int
as
begin
          delete from ParaqitjaProvimeve
          where StudentId=@ID
end
```

Procedura për t'i ndryshuar të dhënat e studentëve:

```
create procedure Ndryshostudentin
@ID as int,
@emri as varchar(50),
@Mbiemri as varchar (50),
@DepartamentiID as int
as
begin
     update Studentat
     set emri =@emri,
     Mbiemri=@Mbiemri,
     DepartamentiID=@DepartamentiID
     where id=@ID
end
```

Një komandë tjetër e cila përdoret në SQL është komanda ***inner join,*** komandë kjo e cila përdoret për bashkimin e tabelave të caktuara. Për shembull nëse kemi një tabelë e cila ka të dhëna për profesorat ndërsa në një tabelë tjetër e cila ka të dhëna për lëndët, në këtë rast kemi të bëjmë me lidhjen e këtyre tabelave dhe si rezultat do të dimë lëndën të cilën e jep profesori i caktuar.

```
create procedure [dbo].[shfaq_emrin]
as
begin
select  p.*,l.lenda
from profesorat as p
inner join lendet as l on l.id=p.lendaid
end
```

Tek dritarja studentët duhet të krijojmë klasën të cilën e kemi emërtuar me emrin *Clstudenti*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;                                    (A)
using System.Data;
using System.Windows.Forms;

namespace Fakulteti.Klasat
{
   public static class clstudenti
    {
private static SqlConnection lidhja_me_DB = new SqlConnection(@"data                  (B)
source=.\sqlexpress;initial catalog=Fakulteti;integrated security=SSPI");

public static string RuajeStudentin(string Emri_I_Procedures, string[] Vlerat)
{
   string mesazhi = "";
   try
   {
     SqlCommand cmdInsertKomanda = new SqlCommand();
     cmdInsertKomanda.Connection = lidhja_me_DB;
     cmdInsertKomanda.CommandType = CommandType.StoredProcedure;
     cmdInsertKomanda.CommandText = Emri_I_Procedures;
     SqlParameter Studentat= new SqlParameter();
     cmdInsertKomanda.Parameters.Add(new SqlParameter("Studentat",
     Vlerat[0].ToString()));
     if (lidhja_me_DB.State == ConnectionState.Closed)
     {
       lidhja_me_DB.Open();
     }                                                          (C)
     SqlDataReader drEkzekuto = cmdInsertKomanda.ExecuteReader();
   }
   catch (Exception exp)
   {
     mesazhi = exp.Message;
   }
   finally
   {
     lidhja_me_DB.Close();
   }
   return mesazhi;
 }


// vazhdon në faqen tjeter
```

```
public static string Ndrysho_studenti(string Emri_I_Procedures, string[]Vlerat)
        {
        string mesazhi = "";
            try
            {
               SqlCommand cmdInsertKomanda = new SqlCommand();
                cmdInsertKomanda.Connection = lidhja_me_DB;
                cmdInsertKomanda.CommandType = CommandType.StoredProcedure;
                cmdInsertKomanda.CommandText = Emri_I_Procedures;
                SqlParameter lendet = new SqlParameter();
                cmdInsertKomanda.Parameters.Add(new SqlParameter("ID",
Vlerat[0].ToString()));
                cmdInsertKomanda.Parameters.Add(new SqlParameter("Studenti",
Vlerat[1].ToString()));
              if (lidhja_me_DB.State == ConnectionState.Closed)
              {

                    lidhja_me_DB.Open();
               }
               SqlDataReader drEkzekuto = cmdInsertKomanda.ExecuteReader();
             }
            catch (Exception exp)
            {
                mesazhi = exp.Message;
            }
            finally
            {
              lidhja_me_DB.Close();
            }
            return mesazhi;
        }

public static string Fshij_studentin(string Emri_I_Procedures, string[] Vlerat)
        {
            string mesazhi = "";
            try
            {   SqlCommand cmdInsertKomanda = new SqlCommand();
                cmdInsertKomanda.Connection = lidhja_me_DB;
                cmdInsertKomanda.CommandType = CommandType.StoredProcedure;
                cmdInsertKomanda.CommandText = Emri_I_Procedures;
                SqlParameter studentat = new SqlParameter();
                cmdInsertKomanda.Parameters.Add(new SqlParameter("ID",
Vlerat[0].ToString()));
              if (lidhja_me_DB.State == ConnectionState.Closed)
                {
                    lidhja_me_DB.Open();

                }
                SqlDataReader drEkzekuto = cmdInsertKomanda.ExecuteReader();
            }
            catch (Exception exp)
            {
                mesazhi = exp.Message;
            }
            finally
            {
                lidhja_me_DB.Close();
}
            return mesazhi;
 }
 // vazhdon në faqen tjeter
```

(D)

(E)

```
        public static void PastroFushat(TextBox[] txtFushat)
         {
             foreach (TextBox txt in txtFushat)
             {
                 txt.Text = "";
             }
         }
         public static DataTable ShfaqShenimet(string Emri_I_Procedures)
         {
             DataTable shenimet = new DataTable();
             try
             {
                 SqlDataAdapter dataAdapter = new
                 SqlDataAdapter(Emri_I_Procedures, lidhja_me_DB);
                 DataSet myDataSet = new DataSet();
                 dataAdapter.Fill(myDataSet);

                 shenimet = myDataSet.Tables[0];
             }
             catch
             {
             }
             finally
             {
                 lidhja_me_DB.Close();
             }
             return shenimet;
         }
    }

}
```

(F)

(A)- Së pari duhet të caktojmë *using System(-et)*, për shembull siç është rasti tek lidhja me databazë, që duhet të përdorim *using System.Data.SqlClient*. Po ashtu programin na njofton edhe për *using system-et* të cilat janë të nevojshëm t'i përdorim gjatë shkruarjes së kodit. Si dhe e caktojmë *namespacen* e klasës.

(B)- Më pas e deklarojmë klasën ose publike ose private. Pasi të deklarojmë klasën atëherë e shkruajmë kodin për t'u lidhur me databazën, duhet t'ia caktojmë emrin e serverit SQL si dhe emrin e databazës që në rastin tonë është Fakulteti.

(C),(D),(F) - Tani krijojmë metoda të cilat i thërrasim në formën kryesore komanda. Një komandë tjetër që e përdorim për parashikimin e gabimeve është edhe komanda *try, catch dhe finally*.

Try-catch-finally – *Trajtimi i gabimeve* është një sekuencë e kodit në të cilën mund të haset ndonjë gabim, ky kod vendoset në bllokun *try,* pjesa tjetër të sekuencës së kodit që e trajton gabimin e ndodhur në bllokun *try* duhet të shkruhet në bllokun *catch.* Pjesa e kodit e cila duhet gjithsesi të ekzekutohet, pa marrë parasysh se ka pasur ndonjë gabim apo jo, duhet të shkruhet në bllokun *finally*.

Në dizajn të formës klikojmë më tastin e djathtë dhe zgjedhim opsioni *view code* dhe shkruajmë këtë kod.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;
using Fakulteti.Klasat;


namespace Fakulteti
{
  public partial class frmStudentat : Form
  {
    public frmStudentat()
    {
      InitializeComponent();
    }

    private void Departamentet()
    {
      DataTable depertamentet =
      CLDepartamentet.ShfaqShenimet("ShfaqShenimet_Departamentet");
      cmbDepartamentet.DataSource = depertamentet;
      Validimet.ff(depertamentet, "ID", "Departamenti");
      cmbDepartamentet.DisplayMember = "id";
      cmbDepartamentet.ValueMember = "Departamenti";
      cmbDepartamentet.SelectedValue = "0";

    }

    private void shfaq_emrat()
    {
      dataGrid2.DataSource = clstudenti.ShfaqShenimet("shfaq_emrat");
      clstudenti.PastroFushat(new TextBox[]{txtEmri, txtID,
                                            txtMbiemri });

      cmbDepartamentet.SelectedValue = "0";

    }
    private void Ndrysho_Studentin()
     {
      string[] vlerat =
      new string[]
      {
        txtID.Text,txtEmri.Text,txtMbiemri.Text,
        cmbDepartamentet.SelectedValue.ToString(), cmbklasa.Text,
        cmbparalelja.Text
      };

       string mesazhi =
       clstudenti.Ndrysho_studenti("Ndryshostudentin", vlerat);
       if (mesazhi.Length > 0)
       {
         MessageBox.Show("Nuk eshte ndryshuar shenimi ne databaze" +
         System.Environment.NewLine + mesazhi);
       }
```

```
    }
      private void Form2_Load(object sender, EventArgs e)
      {
        btnIri_Click(null, null);
        Departamentet();
      }

      private void dataGrid2_CellClick(object sender,
      DataGridViewCellEventArgs e)
      {
        txtID.Text =
dataGrid2.Rows[dataGrid2.CurrentRow.Index].Cells["ID"].Value.ToString();
        txtEmri.Text =
dataGrid2.Rows[dataGrid2.CurrentRow.Index].Cells["Emri"].Value.ToString();
        txtMbiemri.Text =
dataGrid2.Rows[dataGrid2.CurrentRow.Index].Cells["Mbiemri"].Value.ToString()
      }

      private void btnNdrysho_Click(object sender, EventArgs e)
      {
        Ndrysho_Studentin();
        shfaq_emrat();
      }

      private void btnFshij_Click(object sender, EventArgs e)
      {
        if (MessageBox.Show("Deshironi te fshini departamentin e
        zgjedhur?", "", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question) == DialogResult.Yes)
        {
          string[] vlerat = new string[] { txtID.Text };
          string mesazhi =
          clstudenti.Fshij_studentin("fshij_studentat",
          vlerat);

          if (mesazhi.Length > 0)
          {
            MessageBox.Show("Nuk eshte fshire nga databaza" +
            System.Environment.NewLine + mesazhi);
          }
          else
          {
            shfaq_emrat ();
          }
        }
      }
      private void btnIri_Click(object sender, EventArgs e)
      {
        shfaq_emrat();
      }
      private void btnRuaj_Click(object sender, EventArgs e)
      {
        if ((Validimet.txt(new TextBox[]{txtEmri,txtMbiemri}))==false)
        {
          string mesazhi = clstudenti.RuajeStudentin("Ruaj_Studentin",
          new string[] { txtEmri.Text, txtMbiemri.Text,
          cmbDepartamentet.SelectedValue.ToString(), cmbklasa.Text,
          cmbparalelja.Text });
          if (mesazhi.Length > 0)
          {
```

```
            MessageBox.Show(mesazhi);
          }
          else
          {
            shfaq_emrat();
          }
        }
        else
        {
          MessageBox.Show("duhet te plotesohen te gjitha fushat");
        }

      }
    }
}
```

Për paraqitjen e departamenteve në combobox kemi përdorur të dhënat të cilat ndodhen në një klasë tjetër, kjo na e lehtëson punën pasi që nuk kemi nevojë të shkruajmë të njëjtin kod përsëri.
Një klasë tjetër të cilën e kemi përdorur është klasa *Validimet*, në këtë klasë kemi përdorur *datatable*, në mënyrë që kur të klikojmë në ndonjë combobox së pari të na paraqitet fjalia "zgjedh.." e jo emri i ndonjë departamenti, po ashtu edhe metoda pastro fushat , etj.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data;

namespace Fakulteti.Klasat
{
    public static class Validimet
    {
        public static bool txt(TextBox[] txtbox)
        {
            bool val = false;
            for (int i = 0; i <= txtbox.Length - 1; i++)
            {
                if (txtbox[i].Text == "")
                {
                    val = true;
                    break;
                }
            }
            return val;
        }
        public static void ff (DataTable tblSource, string ID, string
         Vlera)
        {

            DataRow drow = tblSource.NewRow();
            drow[ID] = "0";
            drow[Vlera] = "<Zgjedh ...>";
            tblSource.Rows.Add(drow);

        }
```

```
        public static void pastro_fushat(Control fushat)
         {
             for (int i = 0; i <= fushat.Controls.Count - 1; i++)
             {

                 if (fushat.Controls[i].GetType().ToString() ==
                   "System.Windows.Forms.TextBox")
                 {
                     TextBox txtctrl = (TextBox)fushat.Controls[i];
                     txtctrl.Text = "";
                 }

                 if (fushat.Controls[i].GetType().ToString() ==
                   "System.Windows.Forms.ComboBox")
                 {
                     ComboBox cmbctrl = (ComboBox)fushat.Controls[i];
                     cmbctrl.SelectedValue = "0";
                 }

             }

         }
    }
}
```

Përfundimi dhe Literatura

Në këtë punim përfshihet përdorimi i disa komandave me rëndësi të cilat na mundësojnë ruajtjen e të dhënave, ndryshimin e tyre dhe fshirjen e të dhënave. C# na ofron mundësinë që t'i fshijmë këto të dhëna pa krijuar procedura në SQL mirëpo më mirë është që të krijohen procedura në SQL për shkak të sigurisë së të dhënave.
SQL na ofron mundësi që të manipulojmë me të dhëna, siç është për shembull mundësia e renditjes së të dhënave ashtu siç ne dëshirojmë, nëse kemi një tabele me emra dhe mbiemra atëherë ne kemi mundësinë që kur të na shfaqen këto të dhëna t'i rendisim sipas emrit, si dhe shumë mundësi të tjera.

Literatura:

1. Introduction to C# programming with Microsoft.NET, Microsoft Official Curriculum

2. Bill Sempf – Csharp 2010 All in One For.Dummies

3. Bart de Smet – Csharp 4.0 Unleashed

Konsulent:

1. Sefer Voca – Zhvillues softuerik

Ky publikim mund të gjendet online në

http://sites.google.com/site/softashqiptare/