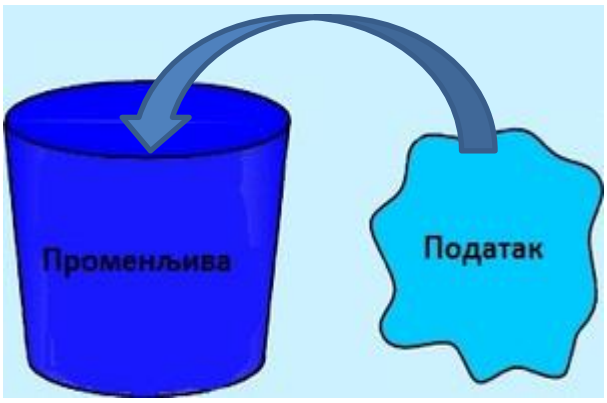
Променљива

Пре него што кажемо нешто више о типовима података, потребно је одговорити на следеће битно питање.

Шта је променљива ? **Променљиве** су места у меморији у којима подаци могу привремено да се чувају да би програм могао да их користи. Оне имају **ТИП**, **ИМЕ** и **ВРЕДНОСТ**. Вредност променљиве може да се мења за време извршавања програма, због чега је и добила назив – променљива. Да би уопште могла да се употреби, променљива најпре мора да се декларише : њен тип мора да се наведе и мора јој се дати име. Тип променљиве дефинише дозвољени распон вредности које тај тип може да садржи, као и операције које могу да се обављају на датој променљивој.



Типом података дефинисан је :
- начин регистровања података у меморији
- скуп могућих вредности тих података
- скуп могућих акција над подацима

Име променљиве гради се на основу следећег правила :
Име је низ слова (великих и малих) енглеске abecede, знака за подвлачење (_) и цифара. Име **мора** да почне словом или знаком за подвлачење. Према томе, примери имена променљивих су NazivPredmeta, ucenik1, Rezultat, radni_dan, _ime, ... док godisnje-doba, 1dan, ... нису и не могу бити имена променљивих.

Напомена: Важно је напоменути да C# (попут програмског језика C) разликује велика и мала слова тако да имена променљивих GodisnjeDoba и godisnjeDoba нису иста.

Именовање података у апликацијама и дефинисање типа података којем припадају постижемо декларацијом променљивих на следећи начин:

<ime tipa> <ime promenljive1>, …, <ime promenljiveN> ;

Декларација променљиве у програмском језику C# изгледа овако :

```
int a , b;
char p , q;
```

У наредним поглављима ћемо видети које типове података означавају речи int и char.

Подаци су објекти који се обрађују у програмима. Основно обележје сваког податка је његов **тип** . Типови података одређују могуће вредности података и могуће операције које могу да се изводе над тим подацима. C# је строго типизиран језик, што значи да морамо додати тип података сваком меморијском елементу који користимо у програму. С обзиром да је C# модеран језик, он има велики број уграђених типова који долазе спремни за употребу у нашим сопственим програмима. На наредним страницама видећемо који све типови података постоје и које се све операције могу вршити над њима

Оператори

У овом поглављу ћемо се упознати са одређеним врстама оператора и видети чему они служе. То су оператори доодељивања, аритметички оператори и оператори поређења. Поред тога, видећемо и какав приоритет важи међу њима.

Оператор додељивања. Оператор додељивања поставља вредност промењиве.

Оператор додељивања

Оператор	Опис
=	Додељивање вредности промењивој

Сада ћемо показати како се некој нашој промељивој додељује вредност коришћењем оператора додељивања = :

```
//Navodimo tip i ime promenljivih koje hocemo da koristimo
int a ;
int b ;
//Nasoj promenljivoj a dodeljujemo vrednost 5, a promenljivoj b vrednost 13
a = 5;
b = 13;
```

Аритметички оператори. Аритметички оператори извршавају аритметичке операције. Постоји пет аритметичких оператора (видети табелу која следи).

Аритметички оператори

Оператор	Опис
+	Сабирање
-	Одузимање
*	Множење
/	Дељење
%	Остатак при дељењу

У следећим изразима приказана је употреба оператора сабирања(+), одузимања(-), множења(*) и дељења (/), респективно :

```
1 + 2
3 - 2
2 * 5
10 / 3
```

Оператори поређења. Оператори поређења упоређују однос између вредности и враћају Boolean вредност (true ili false). Постоји шест оператора поређења (види следећу табелу).

Оператори поређења

Оператор	Опис
==	једнако
!=	неједнако
>	веће од
<	мање од
>=	веће од или једнако
<=	мање од или једнако

Напомена: Водите рачуна о разлици између оператора (==), који пореди две вредности и оператора доделе (=)који врши додељивање вредности.

Шта враћају следећи изрази као повратну вредност ?

```
//Pitamo se, da li je 10 manje od 3 ? Kako ovo nije ispunjeno ovaj izraz vraća false
10  < 3
//Pitamo se, da li je 10 manje od ili jednako 3 ?
//Kako ovo nije ispunjeno ovaj izraz vraća false
10 <= 3
//Pitamo se, da li je 10 različito od 3 ? Kako je ovo ispunjeno ovaj izraz vraća true
10 != 3
//Pitamo se, da li je 10 veće od 3 ? Kako je ovo ispunjeno ovaj izraz vraća true
10  > 3
//Pitamo se, da li je 10 veće od ili jednako 3 ?
//Kako je ovo ispunjeno ovaj izraz vraća true
10 >= 3
```

Приоритет оператора.

Када пишемо израз који садржи више од једног оператора, компајлер користи скуп правила која одређују који оператор треба применити први, други, трећи и тако даље док се не израчуна цео израз. Сваки оператор има уграђени приоритет, или првенство, које компајлер користи да би одредио који оператор треба следеће применити. Погледајмо један пример који додељује вредност променљивој типа int, која се зове mojBroj :

```
int mojBroj = 2 + 5 * 10 ;
```

Иницијално, можемо мислити да је вредност која се додељује променљивој mojBroj 70, јер је 2 + 5=7, а 7 * 10 = 70, зар не? Погрешно! Оператор множења (*) има већи приоритет, или веће првенство, од оператора сабирања (+), па се десна страна израза израчунава на следећи начин: 5 * 10 = 50, 2 + 50 = 52. Због тога се променљивој mojBroj додељује вредност 52. Међутим, приоритет оператора се може прегазити коришћењем заграда. на пример :

```
int mojBroj = (2 + 5) * 10 ;
```

С обзиром да је 2 + 5 стављено у заграде, оно се прво израчунава. Тако је, 2 + 5 = 7, а 7 * 10 = 70. Због тога се променљивој mojBroj додељује вредност 70. Погледајмо други пример :

```
int mojBroj = 2 * 20 / 5 ;
```

Оператор множења (*) и дељења (/) имају исти приоритет, па се оператори извршавају слева надесно. Тако се, прво израчунава 2 * 20 = 40, а затим 40 / 5 = 8. Због тога се, променљивој mojBroj додељује вредност 8.

У C# има велики број оператора и ови оператори се могу груписати у категорије које имају исти приоритет. Следећа табела нам говори о редоследу првенства оператора - прво је показана категорија са највећим приоритетом.У овој табели приказане су категорије оператора и оператори за сваку категорију по редоследу првенства.

Приоритет оператора

Категорија	Оператори
Основна	Група : (x)
	Приступ пољу : x.y
	Позив методе : f(x)
	Приступ индексу : a[x]
	Инкрементирање са суфиксом : x++
	Декрементирање са суфиксом : x--
	Нови објекат : new
	Get type : typeof
	Get size : sizeof коришћење граничне провере :checked
	Без франичне провере : unchecked
Унарна	Позитивна вредност: +
	Негативна вредност: -
	Логичко Boolean НЕ : !
	НЕ на нивоу бита: -
	Инкрементирање са префиксом :++x
	Декрементирање са префиксом : --x
	Конверзија: (type)

Вишеструки	Множење : *
	Дељење : /
	Остатак при дељењу : %
Додатни	Сабирање : +
	Одузумње: -
Померање на нивоу бита	Лево померање :
	Десно померање
Релациони	Мење од : <
	Веће од : >
	Мање од или једнако : <=
	Веће од или једнако : >=
	Компатибилност типа : is
	Компатибилно са оператором за конверзију : as
Једнакост	Једнако : =
	Неједнако : !=

Када се операнд стави између два оператора са истим приоритетом, редослед операције која ће се извршити одређује асоцијативност оператора. Са изузетком додељувања оператора, оператори који користе два операнда (нпр. оператор сабирања) су лево асоцијативни - што значи да се операције извршавају слева надесно. Оператори додељивања и троструки оператори су десно асоцијативни - што значи да се операције извршавају с десна на лево.

Још објашњења и конкретних примера видећете у поглављима о целобројном и реалном типу. Биће посебних примера везаних за целобројни тип и посебних везаних за реални тип.

Целобројни тип

Већ смо у математици рекли да једначина x + a = b нема увек решење у скупу **N** (прецизно: када је a ≥ b). Скуп N се зато проширује у скуп целих бројева **Z** = {... , -2, -1,0,1,2,3, ...}. Он је неограничен здесна и слева јер не постоји ни број од којега су сви цели бројеви мањи, ни број од којега су сви цели бројеви већи. Као и природни бројеви, скуп **Z** је затворен за операције сабирања и множења. То значи да је збир и производ било која два цела броја опет цео број. Међутим, за разлику од природних

бројева, скуп целих бројева је затворен и за одузимање. Ово не важи и за дељење, јер количник два цела броја не мора да буде цео број (на пример, 1 подељено са 2).

Као и у математици, и у C# целобројни тип ће имати нека ограничења. Постоји осам целобројних типиво које можемо користити да бисмо представили целе бројеве. У **Табели 1** приказани су ови целебројни типови. У њој су приказани C# типови, број бајтова и опсег вредности које се могу меморисати за тај тип.

Табела 1

C# типови	Коришћено бајтова	Вредности
sbyte	1	-128 до 127 (означен)
byte	1	0 до 255 (неозначен)
short	2	-32,768 до 32,767 (означен)
ushort	2	0 до 65,535 (неозначен)
int	4	-2,147,483 ,647 до 2,147,483,647 (означен)
uint	4	0 до 4,294,967,295 (неозначен)
long	8	-9,223,372,036,854,775,808 до 9,223,372,036,854,775,808 (означен)
ulong	8	0 до 18,446,744,073,709,551,615 (неозначен)

Напомена: Постарајте се да изаберете одговарајући тип зависно од тога да ли је ваша целобројна вредност означена или неозначена и у зависности од опсега који треба да меморишете.

Са операторима смо се упознали у посебном одељку, али ево још неких конкретних примера везаних за целобројни тип.

Што се тиче оператора додељивања показали смо како се некој нашој промељивој додељује вредност коришћењем оператора додељивања = :

```
//Navodimo tip i ime promenljivih koje hocemo da koristimo
int a;
int b;
//Nasoj promenljivoj a dodeljujemo vrednost 5, a promenljivoj b vrednost 13
a = 5;
b = 13;
```

Вредности са десне стране оператора додељивања може и сама бити израз. На пример:

```
//Navodimo tip i ime promenljive koje hocemo da koristimo
short moj broj;
//Ovom naredbom promenljivoj moj_broj je dodeljena vrednost 4
moj_broj = 1+5-2;
```

Упознали смо се и са аритметичким операторима. У следећим изразима приказали смо и употрбу оператора сабирања(+), одузимања(-), множења(*) и дељења (/), респективно :

```
1 + 2
3 - 2
2 * 5
10 / 3
```

Када се са аритметичким операторима користе цели бројеви, враћа се цео број и након дељења одбацује се сваки остатак. У примеру 10/3, цео број 10 је подељен са 3 а вредност овог израза је 3 са остатком 1 који ес одбацује. Ако желимо да добијемо остатак при дељењу онда користимо оператор остатка при дељењу (%); У следећем примеру, након целобројног дељења враћа се остатак 1 :

```
//Navodimo tip i ime promenljivih koje hocemo da koristimo
int a;
int b;
int ostatak;
//Nasoj promenljivoj ostatak dodeljujemo vrednost izraza a % b
//Time nasa promenljiva ostatak dobija vrednost 1
ostatak = a % b;
```

За **операторе поређења** рекли смо да упоређују однос између вредности и враћају Boolean вредност (true ili false). Наравно, са операторима поређења могу се користити и променљиве :

```
//Navodimo tip i ime promenljivih koje hocemo da koristimo
//Imamo 3 promenljive,  od toga su dve tipa int i jedna tipa bool
int a;
int b;
bool rezultat;
    //Promenljivoj a dodeljujemo vrednost 10, a promenljivoj b dodeljujemo 1
    a = 10;
    b = 1;
//Kao i malopre, nasa promenljiva rezultat ce imati vrednost true ili false
//Njenu vrednost dobijamo ako odgovorimo na pitanje, da li je 10 razlicito od 1 ?
//Kako je tacno, nasa promenljiva dobija vrednost true
rezultat = a != b;
```

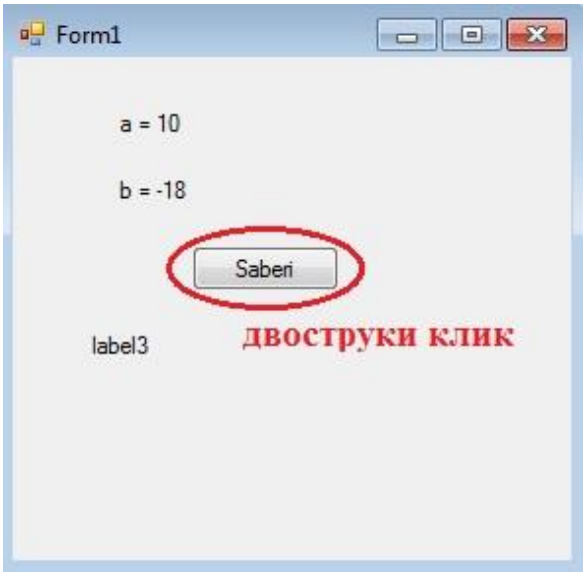
Сада када смо видели како вршимо доделу вредности нашој променљивој, које аритметичке операције смемо да над њом вршимо и који оператори поређења постоје и које су протитета покушаћемо да направимо прве сложеније програмчиће.

Пример 1. Написати програм који сабира два цела броја и њихов збир исписује у лабели.

Решење :

Нека наша два броја a и b (променљиве) буду типа int . Треба да креирамо и трећу променљиву (rezultat) у којој ћемо да смештамо збир та два броја.

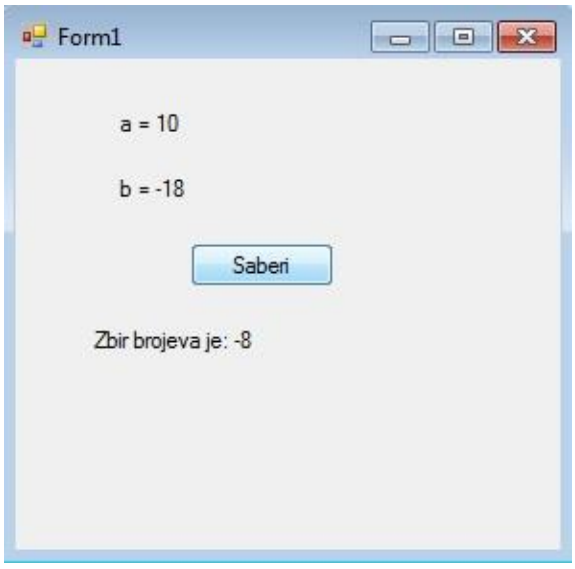
Изглед форме пре покретања програма је :



Слика 4.1. Изглед форме пре покретања програма

```
private void button1_Click(object sender,  EventArgs e)
{
    //Promenljive koje koristimo
    int a, b;
    //Promenljiva u koju cemo smestiti rezultat
    int rezultat;
    a = 10;
    b = -18;
    rezultat = a + b;
    label3.Text = "Zbir je : "  + rezultat;
}
```

Изглед форме након покретања програма је :



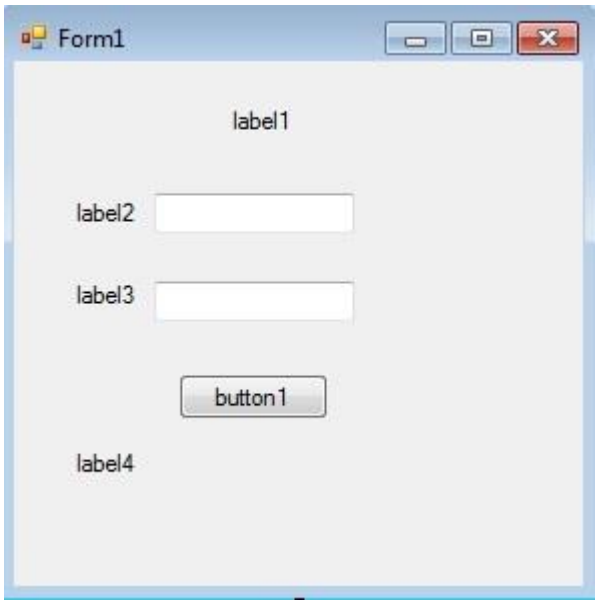
Слика 4.2. Изглед форме након покретања програма

Пример 2. Написати програм који сабира два цела броја која се уносе преко textBox-ова и чији се збир исписује у labele-и.

Решење :

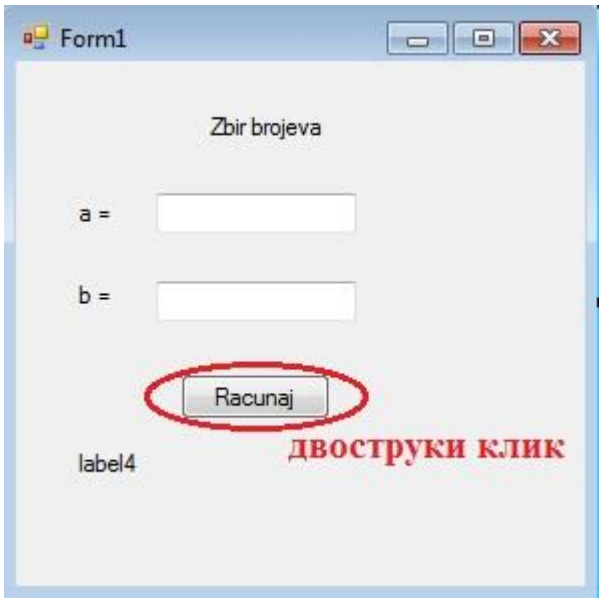
Нека наша два броја a и b (променљиве) буду типа int . Треба да креирамо и трећу променљиву (rezultat) у којој ћемо да смештамо збир та два броја.

Правимо форму облика:



Слика 4.3. Почетни изглед форме

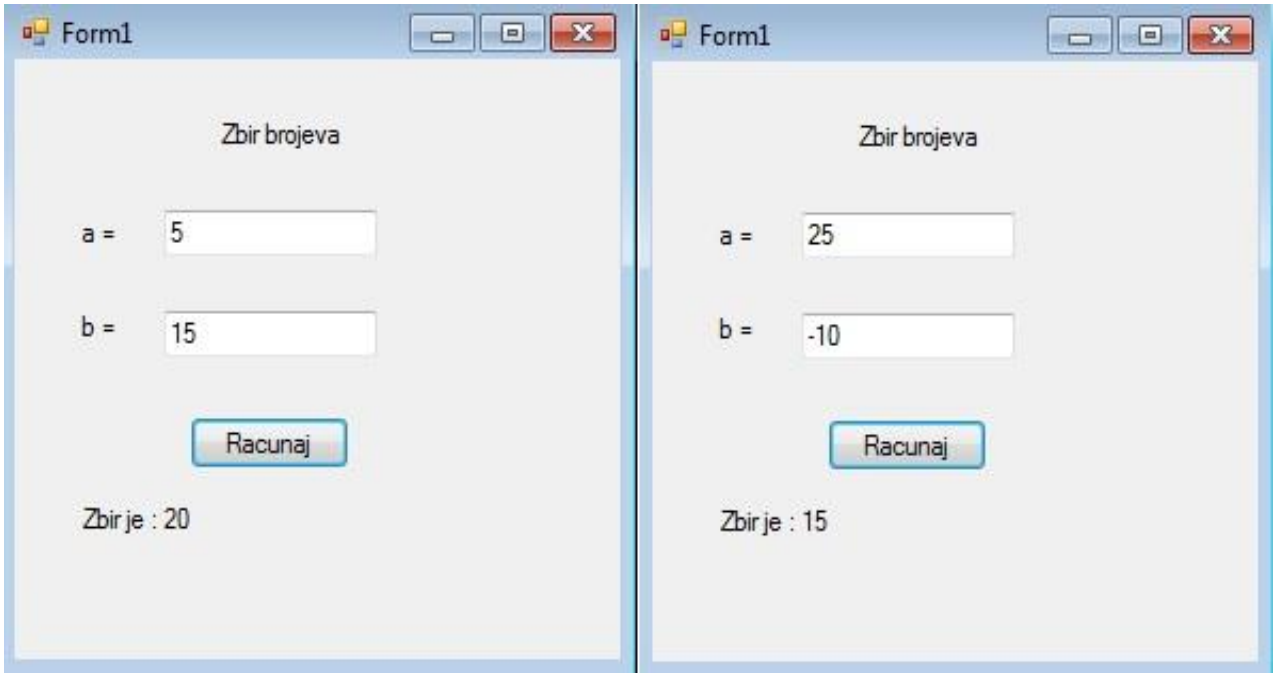
Преименујемо label-е и button1



Слика 4.4. Изглед форме након преименовања компоненти

```
private void button1_Click(object sender,  EventArgs e)
{
    //Promenljive koje koristimo
    int a, b;
    //Promenljiva u koju cemo smestiti rezultat
    int rezultat;
    a = int.Parse(textBox1.Text);
    b = int.Parse(textBox2.Text);
    rezultat = a + b;
    label4.Text = "Zbir je : "  + rezultat;
}
```

Покренемо програм и проверимо (за конкретне вредности бројева) :



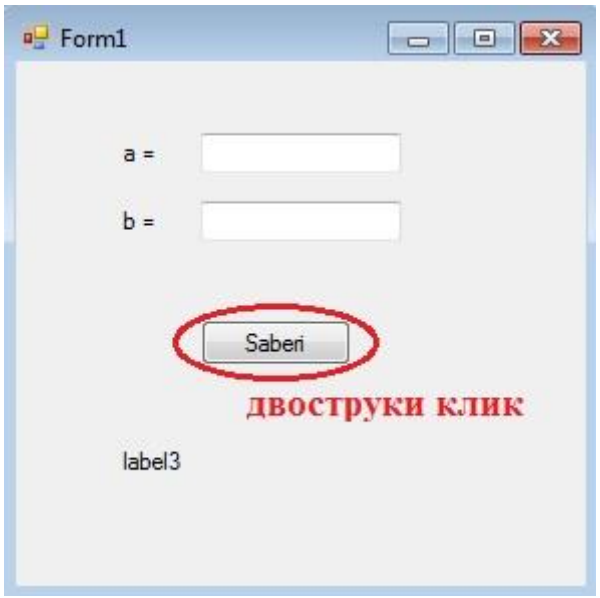
Слика 4.5. Изглед форме након покретања програма

Пример 3. Написати програм који сабира два броја из сегмента [-128,127] која корисник унесе преко textBox-ова.

Решење :

Направићемо две променљиве (a,b) у које ћемо да смештамо оно што корисник унесе преко textBox-ова. Као и у претходном примеру направићемо и трећу променљиву (rezultat) у коју ћемо сместити њихов збир. Ради боље читљивости програма за имена променљибих треба користити осмишљена имена, односно имена која асоцирају на врсту информације која се у њима чува. Поред овога треба променљивој увек додељивати најпогоднији тип. Пошто се овде ради о бројевима који припадају одговарајућем сегменту, типови наших променљивих a и b ће бити sbyte . Остало је још да видимо којег ће типа бити променљива rezultat . Пошто се ради о збиру два броја као резултат ће се добити број који не мора бити из сегмента датог сегмента, зато ћемо за њен тип узети int.

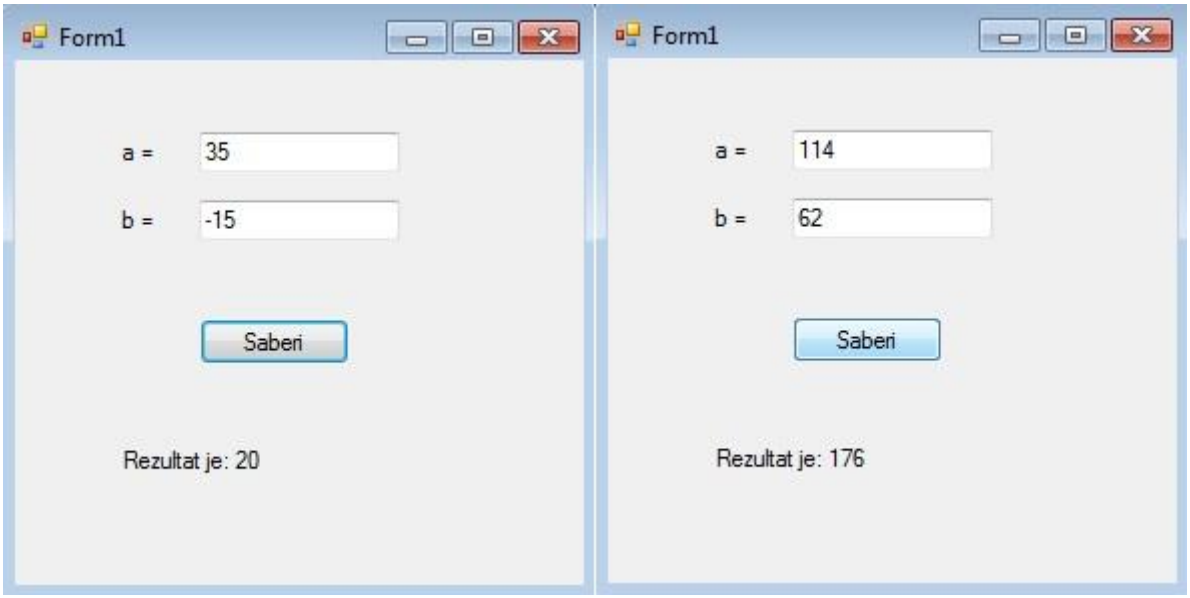
Изглед форме пре покретања програма је :



Слика 4.6. Изглед форме пре покретања програма

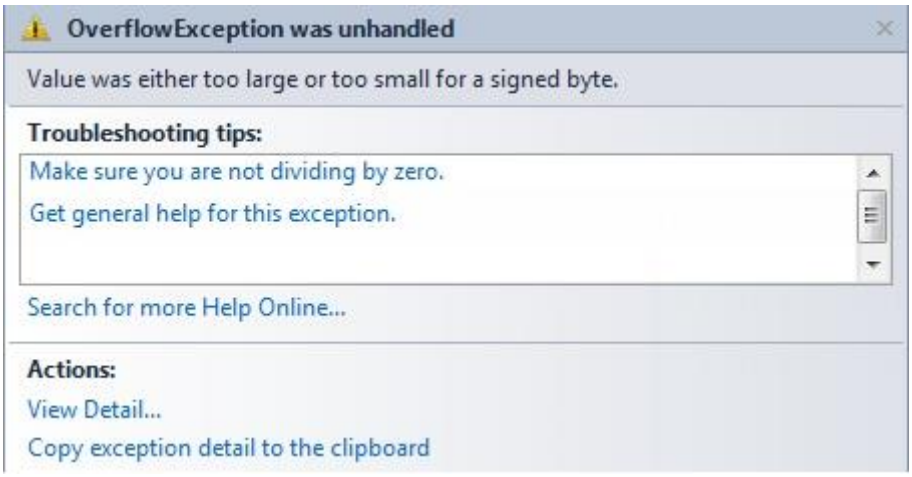
```
private void button1_Click(object sender,  EventArgs e)
{
    //Promenljivoj a dodeljujemo tip sbyte kao i promeljivoj b
    sbyte a;
    sbyte b;
    //Pravimo i trecu promenljivu rezultat u koju cemo da smestimo zbir prve dve i
    //na osnovu prethodnog teksta dodeljujemo joj tip int
    int rezultat;
    //Vrednost koju je korisnik uneo preko textBox-a u promenljivu a,
    a = sbyte.Parse(textBoz1.Text);
    //Isti postupak primenimo za promenljivu b
    b = sbyte.Parse(textBox2.Text);
    //U promenljivoj rezultat unosimo vrednost zbira a i b
    rezultat = a + b ;
    //Ispisujemo rezultat u labelu
    label3.Text = "Zbir je : "  + rezultat;
}
```

Изглед форме након покретања програма :



Слика 4.7. Изглед форме након покретања програма

Поставља се питање шта би се десило да смо за неку промењиву изабрали вредност који излази из опсега који одговара типу sbyte а то је [-128,127] ? Нека је то нпр. вредност 154. У том случају наш програм би се побунио и избацио нам грешку (види слику) која нам управо говори да је унесена вредност за нашу промењиву изван одговарајућег интервала за тип byte.



Слика 4.8. Изглед прозора који се појављује када настане грешка

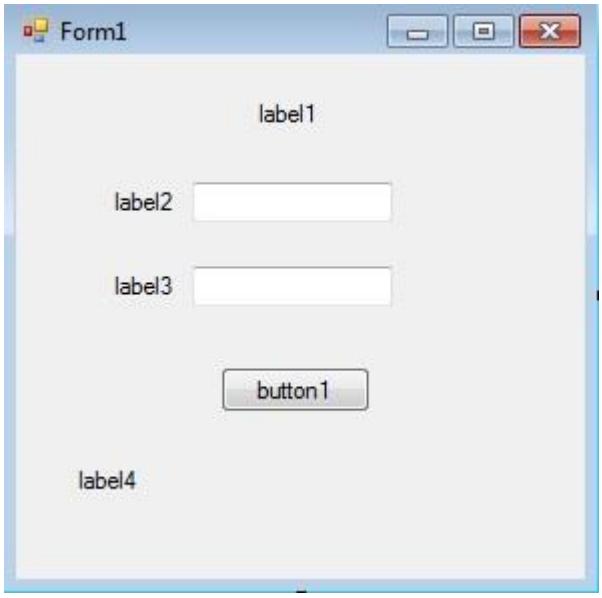
Као што смо на почетку рекли треба се постарати да изаберемо одговарајући тип зависно од тога да ли је наша целобројна вредност означена или неозначена и у зависности од опсега који треба да меморишемо .

Пример 4. Саставити програм који одређује количник и остатак при целобројном дељењу два цела броја унета преко textBox-ова.

Решење :

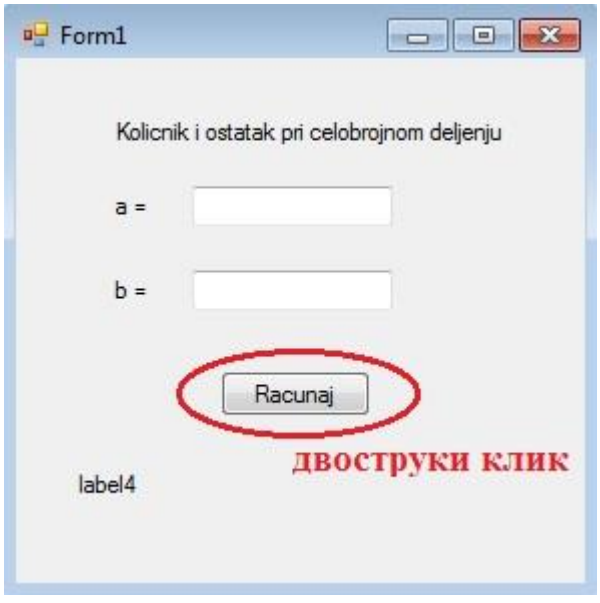
За решавање овог задатке биће нам потребне четири лабеле, два textBox-а и једно дугме. У променљиве a и b смештамо оно што корисник унесе у textBox1 и у textBox2, али тај унос конвертујемо у одговарајући тип што је у нашем случају int. Креирамо још две променљиве kolicnik и ostatak. Користећи знак % добићемо остатак при целобројном дељењу а знаком / добићемо количник.

Правимо форму облика:



Слика 4.9. Почетни изглед форме

Преименујемо label-е и button1

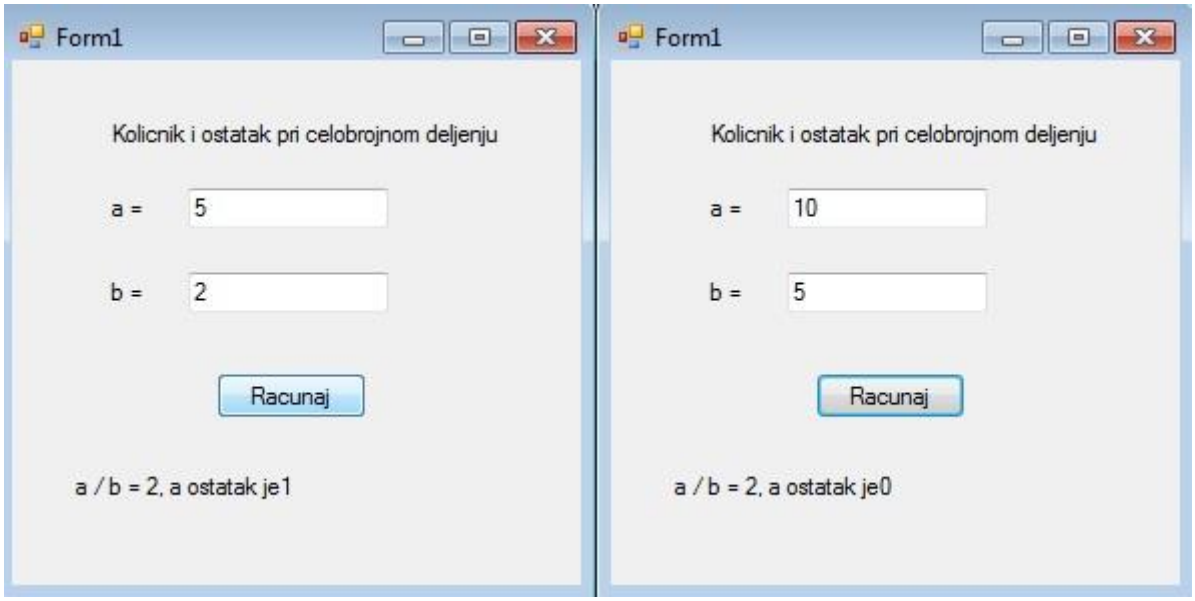


Слика 4.10. Изглед форме након преименовања компоненти

```
private void button1_Click(object sender, EventArgs e)
{
    //Uzimamo vrednost za promenljivu a koju je korisnik uneo
    int a = int.Parse(textBox1.Text);
    //Uzimamo vrednost za promenljivu b koju je korisnik uneo
    int b = int.Parse(textBox2.Text);
```

```
    //Deklarisemo promenljivu ostatak
    int ostatak;
    //Racuna ostatak
    ostatak = a % b;
    //Deklarisemo promenljivu kolicnik
    int kolicnik;
    //Racuna kolicnik
    kolicnik = a / b;
    //Ispisuje resenje
    label3.Text = "a / b = " + kolicnik +  ", a ostatak je" + ostatak;
}
```

Покренемо програм и проверимо (за конкретне вредности бројева) :



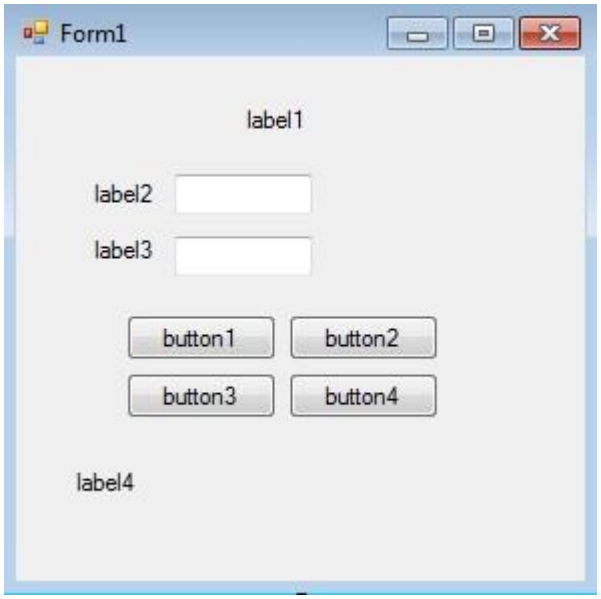
Слика 4.11. Изглед форме након покретања програма

Пример 5. Саставити програм који за унета два броја преко textBox-ова, рачуна збир, разлику, производ или количник, у зависности од тога коју операцију корисник одабере кликом на дугме (за сваку операцију је одговарајуће дугме).

Решење :

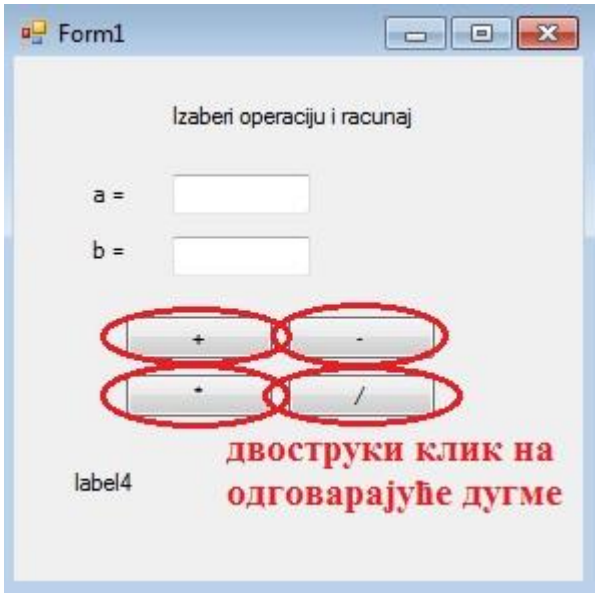
За решавање овог задатка биће нам потребне четири лабеле, четири дугмета и два textBox-а. Креираћемо две променљиве a и b у које ћемо користећи конвертовање сместити оно што корисник унесе у textBox1 и textBox2. Креирацемо прменљиву rezultat у коју ћемо у зависноти од тога на које смо дугме кликнули смештати збир,разлику, призвод или количних унетих бројева.

Правимо форму облика:



Слика 4.12. Почетни изглед форме

Преименујемо label-е и button-ове



Слика 4.13. Изглед форме након преименовања компоненти

```
private void button1_Click(object sender,  EventArgs e)
{
    //Promenljive koje koristimo
    int a, b;
    //Promenljiva u koju cemo smestiti rezultat
    int rezultat;
    a = int.Parse(textBox1.Text);
    b = int.Parse(textBox2.Text);
    rezultat = a + b;
    label4.Text = "Zbir je : "  + rezultat;
}
```

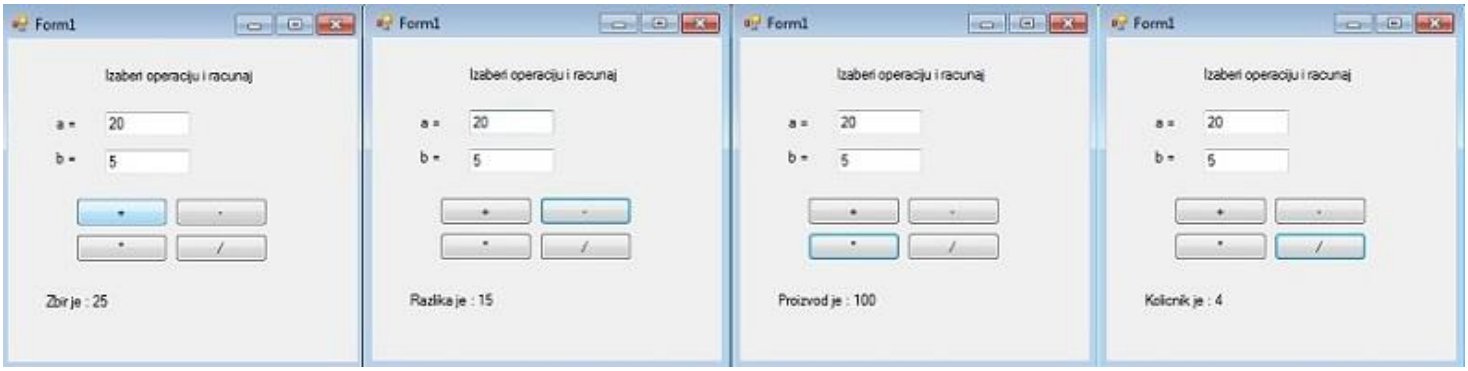
```
private void button1_Click(object sender,  EventArgs e)
{
    //Promenljive koje koristimo
    int a, b;
    //Promenljiva u koju cemo smestiti rezultat
    int rezultat;
    a = int.Parse(textBox1.Text);
    b = int.Parse(textBox2.Text);
    rezultat = a - b;
    label4.Text = "Razlika je : "  + rezultat;
}
```

```
private void button1 Click(object sender,  EventArgs e)
{
    //Promenljive koje koristimo
    int a, b;
```

```
    //Promenljiva u koju cemo smestiti rezultat
    int rezultat;
    a = int.Parse(textBox1.Text);
    b = int.Parse(textBox2.Text);
    rezultat = a * b;
    label4.Text = "Proizvod je : "  + rezultat;
}
```

```
private void button1_Click(object sender,  EventArgs e)
{
    //Promenljive koje koristimo
    int a, b;
    //Promenljiva u koju cemo smestiti rezultat
    int rezultat;
    a = int.Parse(textBox1.Text);
    b = int.Parse(textBox2.Text);
    rezultat = a / b;
    label4.Text = "Kolicnik je : "  + rezultat;
}
```

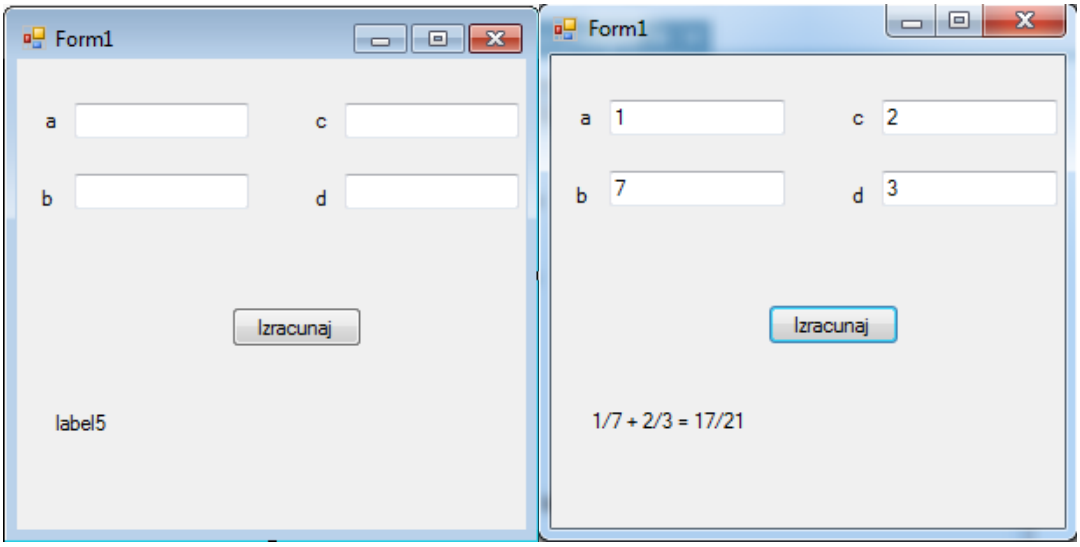
Покренемо програм и проверимо (за конкретне вредности бројева) :



Слика 4.14. Изглед форме након покретања програма

Пример 6. Написати програм који сабира два разломка и исписује њихов збир.

Решење:



```
private void button1_Click(object sender, EventArgs e)
{
    int a, b, c, d, imenilac, brojilac;
    a = int.Parse(textBox1.Text);
    b = int.Parse(textBox2.Text);
    c = int.Parse(textBox3.Text);
    d = int.Parse(textBox4.Text);
    //Kako jos nismo naucili if i for petlju, ovaj razlomak sada necemo skracivati
    brojilac = (a * d + b * c);
    imenilac = b * d;
    //Ispisujemo rezultat
    label5.Text = a + "/" + b + " + " + c + "/" + d + " = " + brojilac + "/" +
imenilac;
}
```

Реални тип

Као што смо у математици научили реални бројеви су сви рационални и ирационалани бројеви.Скуп реалних бројева означавамо са R и знамо да је скуп реалних бројева је бесконачан и непребројив. Као и у математици (реални бројеви су настали због потреба да се прошире цели бројеви) тако и у C# постало је потребно увести тип података који ће представљати нешто што је паралелно са реалним бројевима у математици. Због тога у C# - у постоје типови података ца покретним зарезом које можемо користити да бисмо представили такве бројеве. У C# - у постоје три таква типа (видети тебелу). Избор одговарајућег типа зависи од вредности и броја значајних цифара који је потребан да бисмо меморисали број у покретном зарезу.

Типови за покретни зарез

C# тип	Коришћено бајтова	Вредности
float	4	Приближно +/- $1.5 * 10^{-45}$ до приближно +/- $3.4 * 10^{38}$ са седам значајних цифара
double	8	Приближно +/- $5 * 10^{-324}$ до приближно +/- $1.7 * 10^{308}$ са 15 до 16 значајних цифара
decimal	12	Приближно +/- $1.0 * 10^{-28}$ до приближно +/- $7.9 * 10^{28}$ са 28 значајних цифара

Иако тип decimal подржава мањи опсег бројева, он је у неким случајевима пожељнији, јер никада неће доћи до грешака у заокруживању које се могу јавити код типова float и double. Број типа decimal може да меморише број са тачношћу од 28 цифара.

Напомена: Када се конкретан број додељује типу float, на крају се мора додати знак f или F.

На пример:

```
float mojBroj = 1.2f;
```

Када се конкретан број додељује типу double, на крају се може додати знак d или D, али ово није неопходно. На пример, исто је написати :

```
double mojBroj = 1234.5678d;
double mojBroj = 1234.5678;
```

Обратити пажњу! При додељивању конкретног броја типу double или float може се користити и експоненцијална нотација. На пример :

```
double mojBroj1 = 3.6e+5;  // = 360000
double mojBroj2 = 1.2e-2;  // = 0,012
```

Број 3.6e+5 користи експоненцијалну нотацију да би представио број 3.6 * 10 на пети и исти је као и 360 000. Слично, број 1.2e-2 је 1.2 * 10 на минус други што је исто као и 0,012.

Напомена: При додељивању конкретног броја типу decimal, на крају се мора додати знак m или M.

На пример:

```
decimal  mojBroj3 = 34356.234567m ;
```

Као што и код целобројног типа важе одређене аритметичке операције, операције поређења и приоритети тих операција и код релних типова важи исто, зато ћемо те ствари поновити.

Са операторима смо се упознали у посебном одељку, али ево још неких конкретних примера везаних за реалан тип. Што се тиче оператора додељивања показали смо како се некој нашој промељивој додељује вредност коришћењем оператора додељивања = :

```
//Navodimo tip i ime promenljivih koje hocemo da koristimo
int a ;
int b ;
//Nasoj promenljivoj a dodeljujemo vrednost 5, a promenljivoj b vrednost 13
a = 5;
b = 13;
```

Упознали смо се и са **аритметичким операторима**. У следећим изразима приказали смо и употрбу оператора сабирања(+), одузимања(-), множења(*) и дељења (/), респективно :

```
1 + 2
3 - 2
2 * 5
10 / 3
```

Када се са аритметичким операторима користе бројеви у покретном зарезу, враћа се број у покретном зарезу. На пример, у следећем примеру враћа се 3,333333 :

```
//Navodimo tip i ime promenljivih koje hocemo da koristimo
flaot a;
float b;
float rezultat;
a = 10f;
b = 3f;
//Nasa promenljiva rezultat imace vrednost 3,333333
rezultat = a / b;
```

Можемо користити и више оператора заједно, као што је приказано у следећем примеру :

```
3 * 4 / 2
```

За **операторе поређења** рекли смо да упоређују однос између вредности и враћају Boolean вредност (true ili false).

Сада када смо видели како вршимо доделу вредности нашој променљивој, које аритметичке операције смемо да над њом вршимо и који оператори поређења постоје и које су протитета покушаћемо да направимо прве сложеније програмчиће.

Пример 1. Написати програм који одузима два реална броја која се уносе преко textBox-ова и чија се разлика исписује у labele-и.

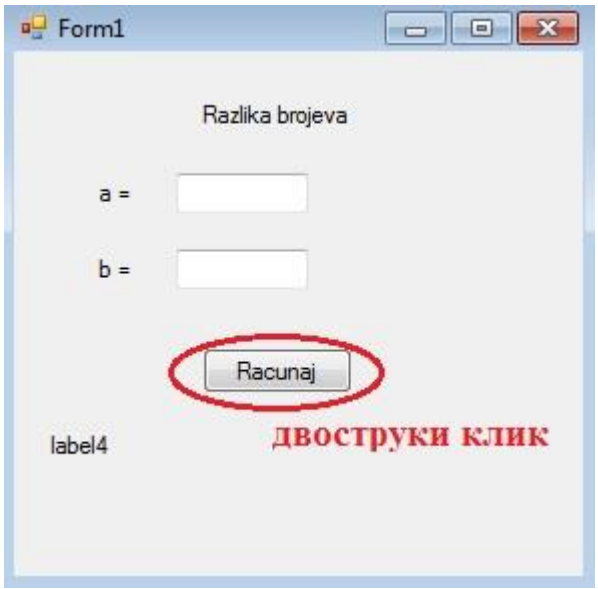
Решење :

Нека наша два броја a и b (промењиве) буду типа double. Треба да креирамо и трећу промењиву (razlika) у којој ћемо да смештамо разлику та два броја.

Правимо форму облика:

Слика 4.15. Почетни изглед форме

Преименујемо label-е и button1

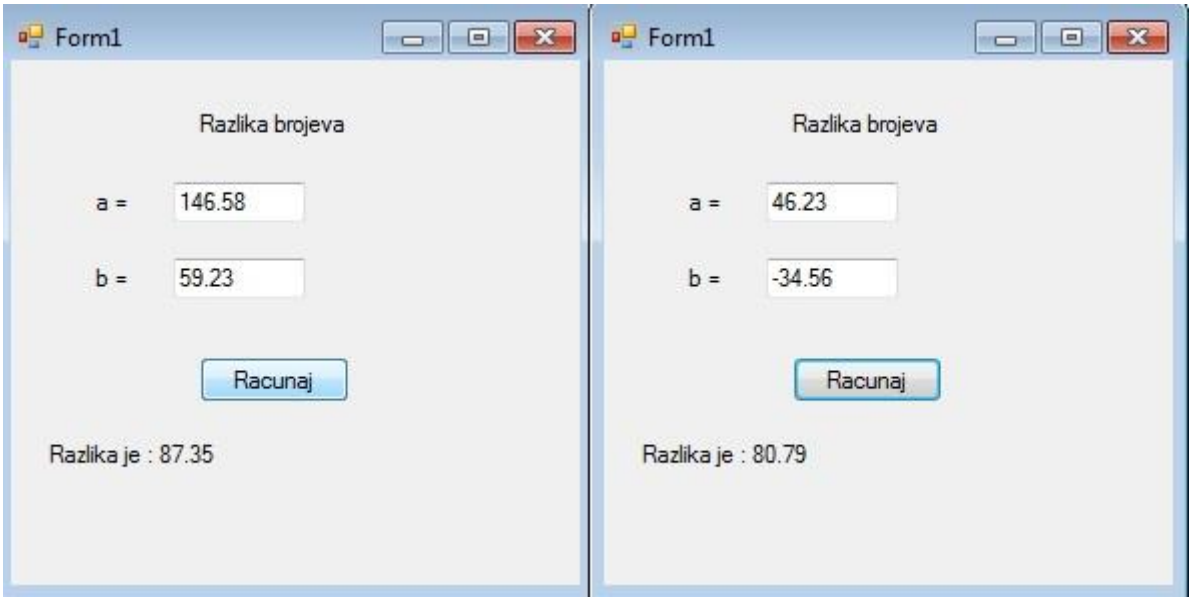


Слика 4.16. Изглед форме након преименовања компоненти

Нека наша два броја a и b (промењиве) буду типа double . Треба да креирамо и трећу промењиву (rezultat) у којој ћемо да смештамо разлику та два броја.

```
private void button1_Click(object sender,  EventArgs e)
{
    //Promenljive koje koristimo
    double a, b;
    //Promenljiva u koju cemo smestiti rezultat
    double rezultat;
    a = double.Parse(textBox1.Text);
    b = double.Parse(textBox2.Text);
    rezultat = a - b;
    label4.Text = "Razlika je : "  + rezultat;
}
```

Покренемо програм и проверимо (за конкретне вредности бројева) :

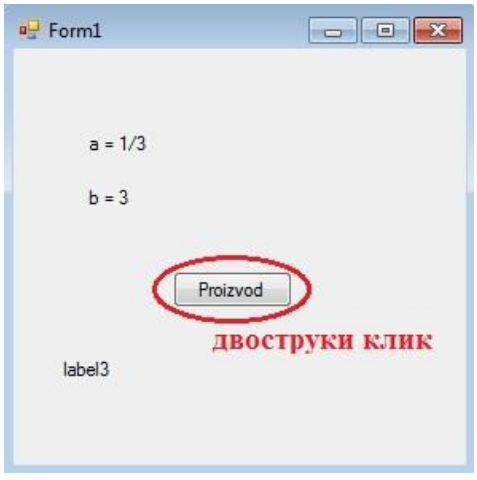


Слика 4.17. Изглед форме након покретања програма

Пример 2. Овај пример ће нам показати да оно што је тачно у математици не мора бити тачно и у програмирању, на пример (1/3) * 3 није једнако 1 .

Решење: :

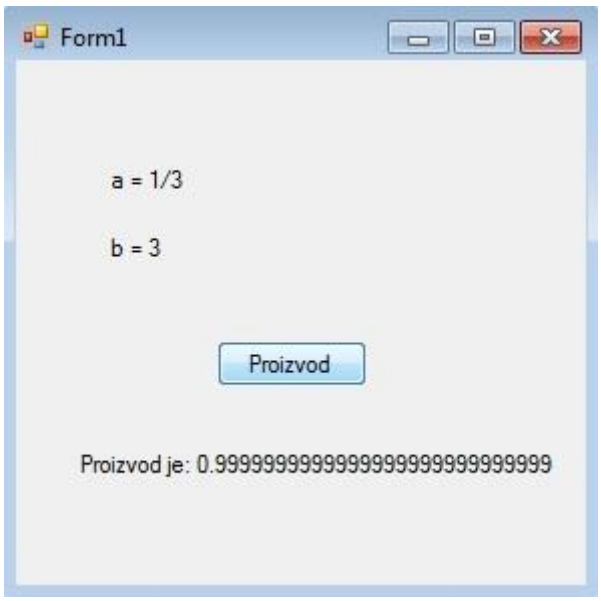
При решавању овог примера, користићемо 3 променљиве. Нека су то a, b и proizvod . Којег су типа ове променљиве ? Узећемо да је свако од њих типа decimal . Треба водити рачуна о томе када се користи тип decimal, на крају се мора додати m или M . Изглед форме пре покретања програма је :



Слика 4.18. Изглед форме пре покретања програма

```
private void button1_Click(object sender,  EventArgs e)
{
    //Promenljivoj a dodeljujemo tip float kao i promeljivoj b
    decimal a;
    decimal b;
    //Pravimo i trecu promenljivu proizvod u koju cemo da smestimo proizvod prve dve
    decimal proizvod;
    //Vrednost promenljive a postavljamo na 1/3
    a = 1/3m;
    //Vrednost promenljive b postavljamo na 1
    b = 3m;
    //U promenljivoj proizvod  unosimo proizvod promenljivih a i b
    proizvod = a * b;
    //Ispisujemo rezultat u labelu
    label3.Text = "Proizvod je : "  + proizvod;
}
```

Изглед форме након покретања програма :



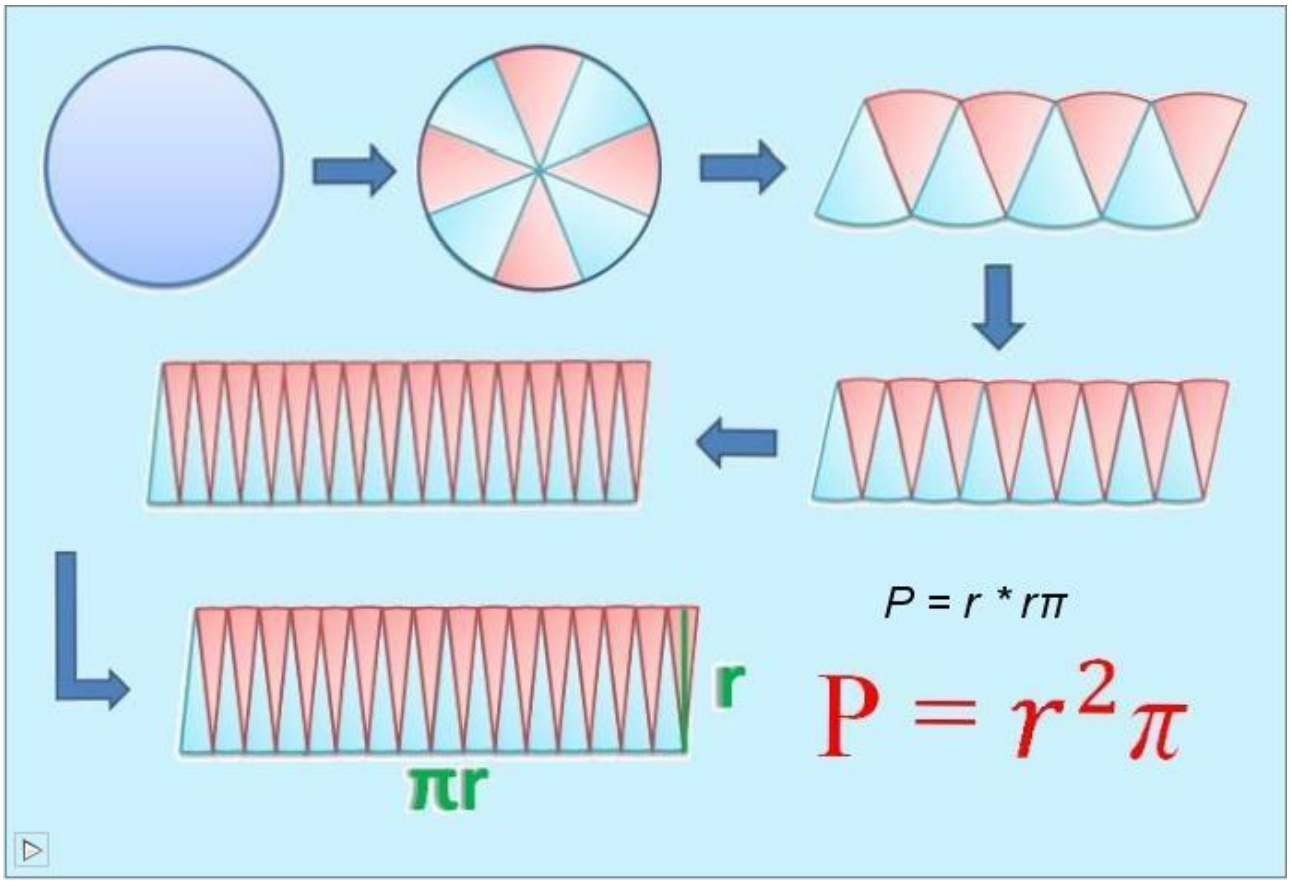
Слика 4.19. Изглед форме након покретања програма

Као решење добили смо број који после зареза има 28 цифара броја 9 . Зашто ? Па ако погледамо у табелу Типови са покретним зарезом видећемо да тип decimal има 28 значајних цифара .

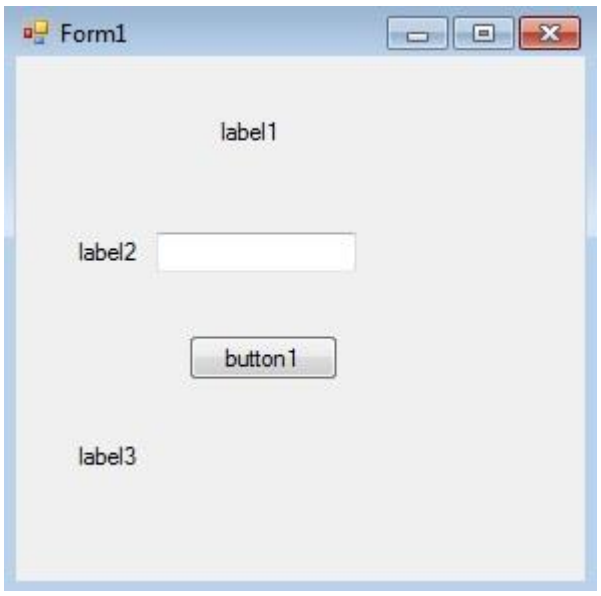
Пример 3. Написати програм који рачуна површину круга за полупречник који се задаје из textBox-а, при чему је Pi~3.14.

Решење :

За решавање овог задатка биће нам потребне три лабеле, једно дугме и један textBox. Креираћемо променљиву r у коју ћемо након конвертовања сместити оно што корисни унесе у textBox1 а променљиву Pi ћемо поставити на вредност 3,14. Кликом на дугме у лабели исписујемо тражени резултат.

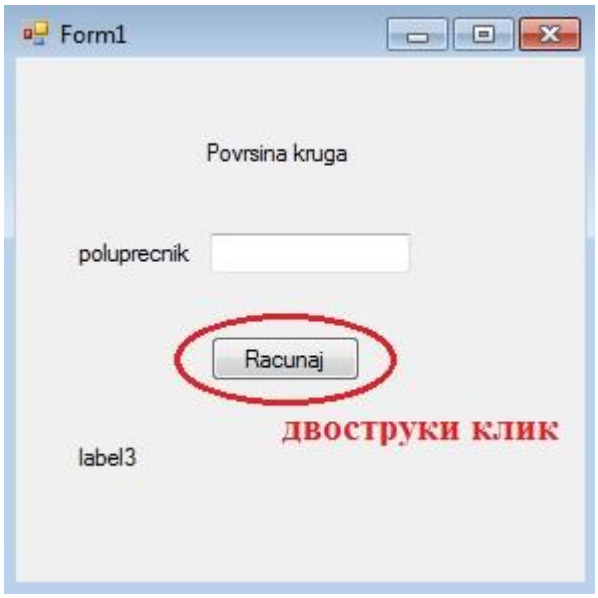


Правимо форму облика:



Слика 4.20. Почетни изглед форме

Преименујемо label-е и button1

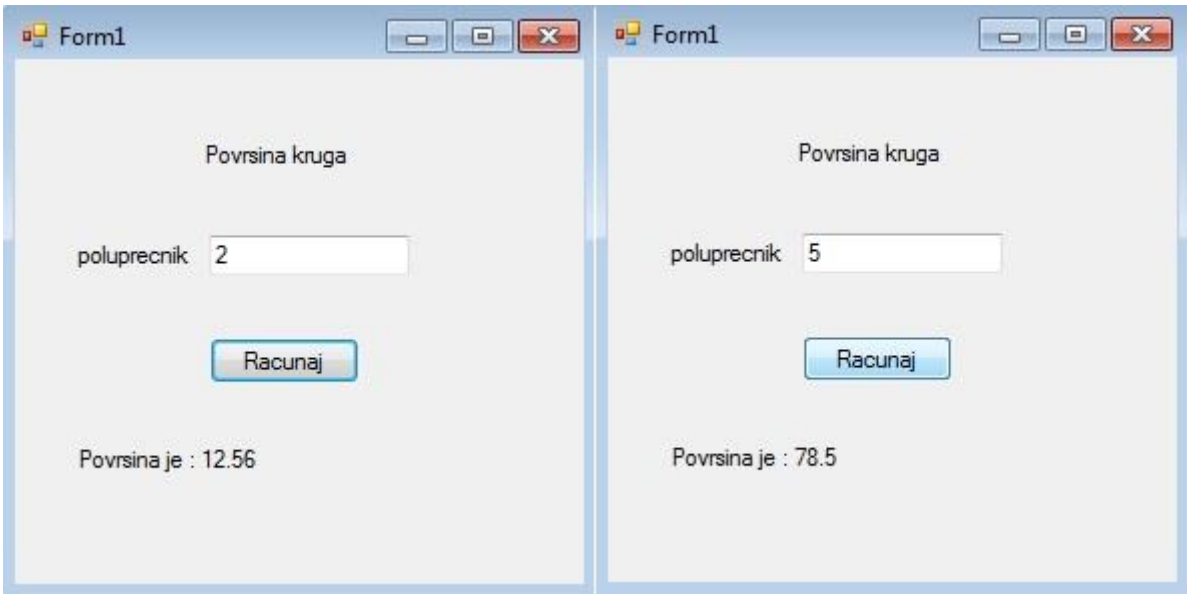


Слика 4.21. Изглед форме након преименовања компоненти

```
private void button1_Click(object sender,  EventArgs e)
{
    //Promenljivu koje koristimo
    double r;
    double Pi = 3.14;
    //Promenljiva u koju cemo smestiti rezultat
    double povrsina;
```

```
    r = double.Parse(textBox1.Text);
    povrsina = r * r * Pi;
    label3.Text = "Povrsina je : "  + povrsina;
}
```

Покренемо програм и проверимо (за конкретну вредност броја) :

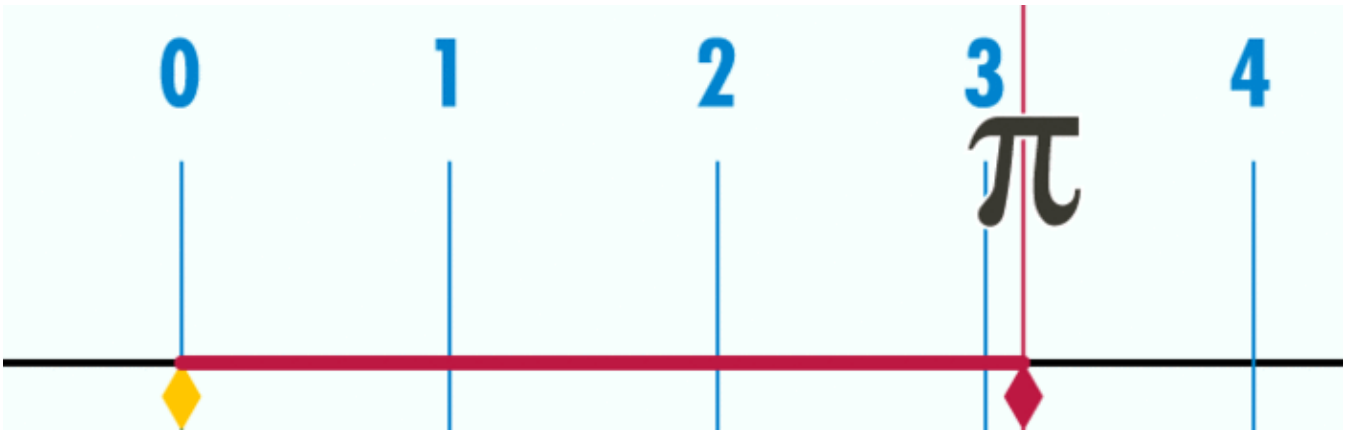


Слика 4.22. Изглед форме након покретања програма

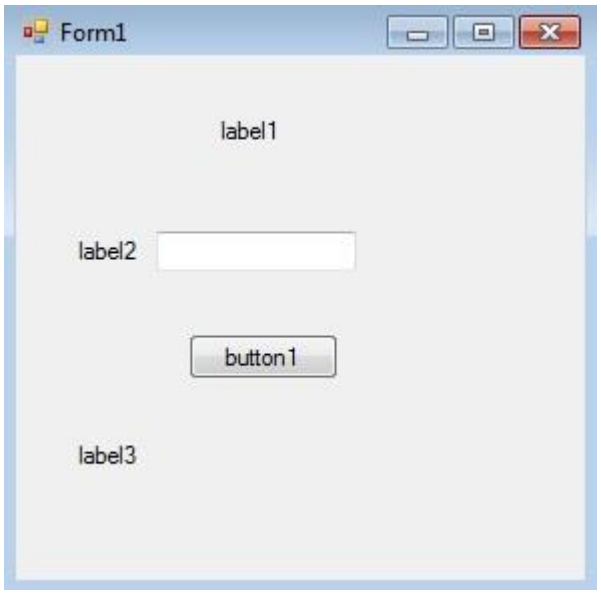
Пример 4. Написати програм који рачуна обим круга за полупречник који се задаје из textBox-а, при чему је Pi~3.14.

Решење :

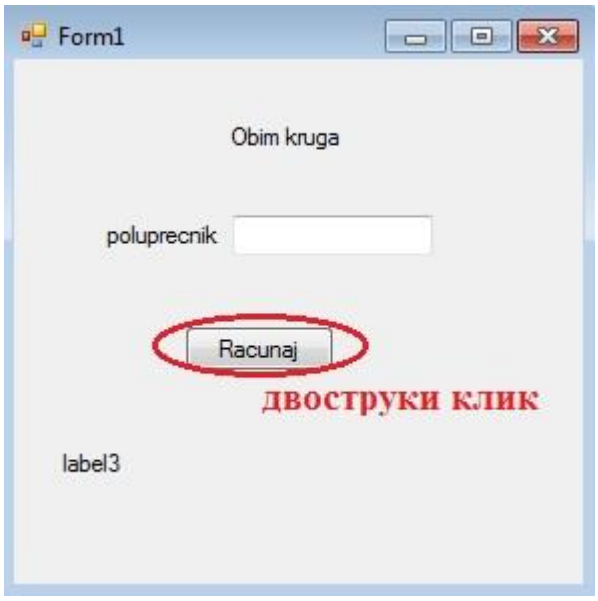
За решавање овог задатке биће нам потребне три лабеле, једно дугме и један textBox. Као и у претходном задатку креирамо променљиву r у коју смештамо оно што корисник унесе у textBox а вредност променљиве Pi постављамо на 3,14. Креирамо и променљиву obim у коју кликом на дугме смештамо израчунат обим круга за задато r.



Правимо форму облика: Преименујемо label-е и button1



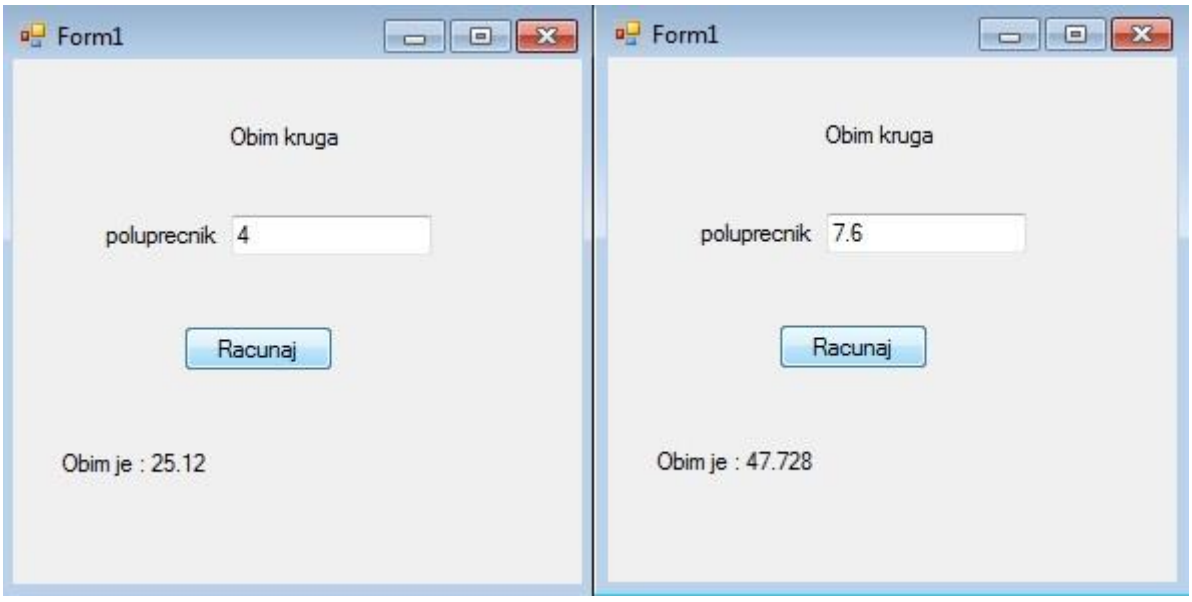
Слика 4.23. Почетни изглед форме



Слика 4.24. Изглед форме након преименовања компоненти

```
private void button1_Click(object sender,  EventArgs e)
{
    //Promenljivu koje koristimo
    double r;
    double Pi = 3.14;
    //Promenljiva u koju cemo smestiti rezultat
    double obim;
    r = double.Parse(textBox1.Text);
    obim = 2 * r * Pi;
    label3.Text = "Obim je : "  + obim;
}
```

Покренемо програм и проверимо (за конкретну вредност броја) :



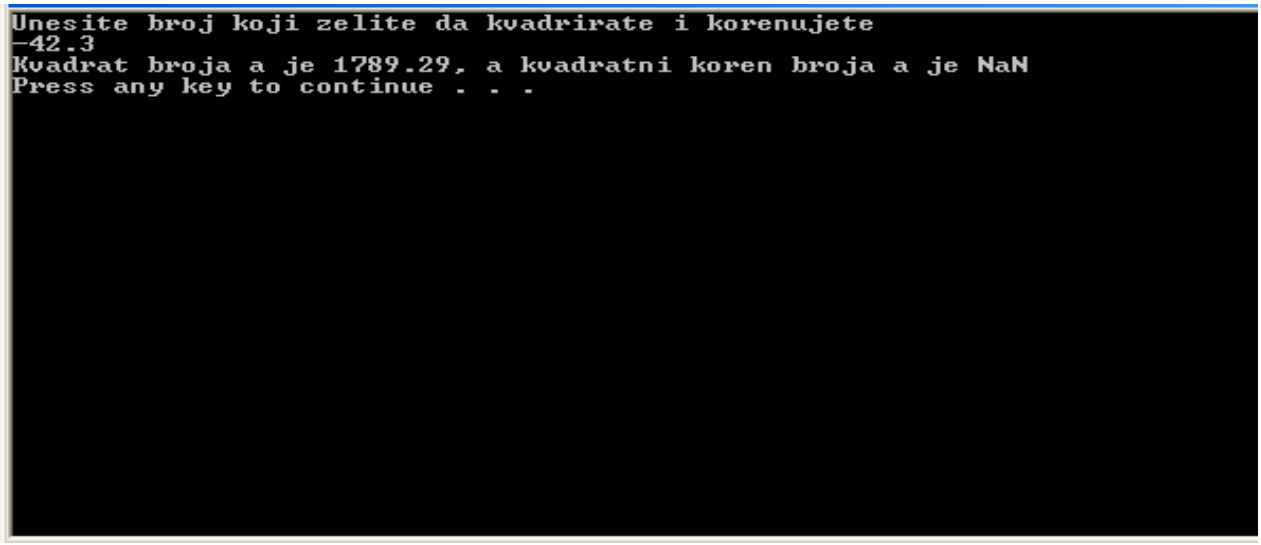
Слика 4.25. Изглед форме након покретања програма

Пример 5. Написати програм који за унети број рачуна његов квадрат и квадратни корен и резултат исписује на конзоли.

Решење :

```
static void Main(string[] args)
{
    double a, kvadrat, koren;
    //Unosimo broj koji zelimo da kvadriramo i korenujemo
    System.Console.WriteLine("Unesite broj koji zelite da kvadrirate i korenujete");
    a = double.Parse(System.Console.ReadLine());
    kvadrat = a * a;
    //Ugradjena funkcija Math.Sqrt koja racuna kvadratni koren prosledjenog broja
    koren =  Math.Sqrt(a);
    //Ispisujemo rezultat na konzoli
    System. Console.WriteLine("Kvadrat broja a je {0}, a kvadratni koren
        broja a je {1}", kvadrat, koren);
}
```

Као што знамо, можемо кореновати само позитивне бројеве. Шта ће се догодити ако у претходном примеру унесемо негативан број? Као што видите на следећој слици, програм је вратио NaN (Not a Number).



Пример 6. Написати програм који за унети број рачуна његов квадрат и квадратни корен и резултат исписује на конзоли.

Решење :

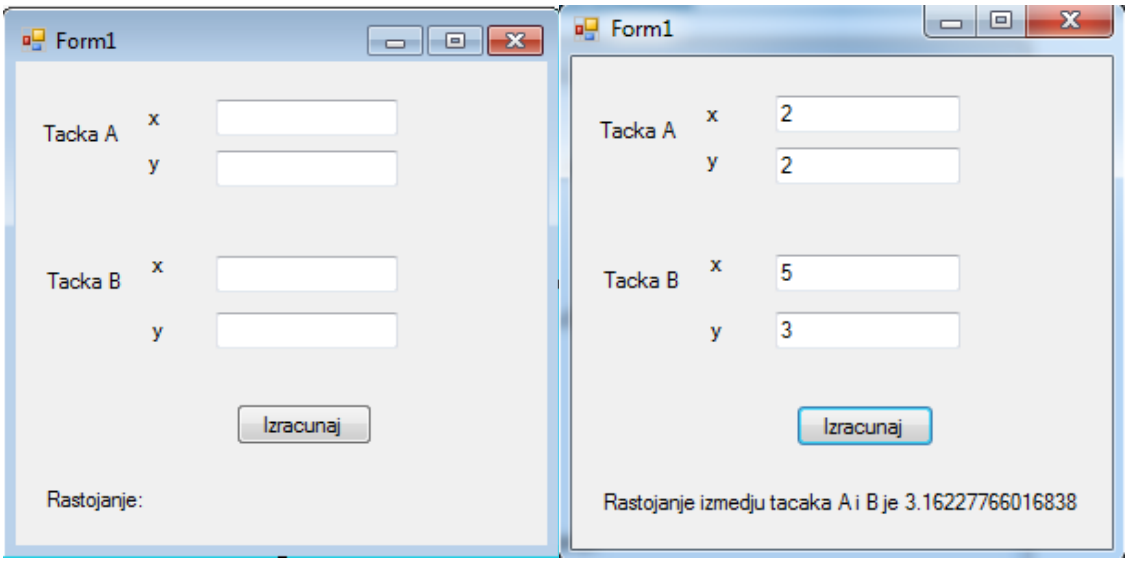
```
namespace ConsoleApplication1
```

```
{
    class Program
    {
        static void Main( string[] args)
        {
            double a, b, c, d, sredina;
            //Unosimo podatke
            System.Console.WriteLine("Unesite prvi broj:");
            a = double.Parse(System.Console.ReadLine());
            System.Console.WriteLine("Unesite drugi broj:");
            b = double.Parse(System.Console.ReadLine());
            System.Console.WriteLine("Unesite treci broj:");
            c = double.Parse(System.Console.ReadLine());
            System.Console.WriteLine("Unesite cetvrti broj:");
            d = double.Parse(System.Console.ReadLine());
            //Racunamo aritmeticku sredinu ovih brojeva
            sredina = (a + b + c + d) / 4;
            //Rezultat ispisujemo na konzoli
            System.Console.WriteLine("Aritmeticka sredina unetih brojeva jednaka je: "
+sredina);
        }
    }
}
```

Пример 7. Написати програм који рачуна растојање између две тачке.

Решење :

```
private void button1 Click(object sender, EventArgs e)
{
    //Deklarisemo promenljive
    int x1, y1, x2, y2;
    double rastojanje;
    //Ucitavamo promenljive iz textBox-ova
    x1 = int.Parse(textBox1.Text);
    y1 = int.Parse(textBox2.Text);
    x2 = int.Parse(textBox3.Text);
    y2 = int.Parse(textBox4.Text);
    //Pozivamo ugradjenu funkciju Math.Sqrt koja racuna kvadratni koren
    rastojanje = Math.Sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
    //Ispisujemo resenje
    label7.Text = "Rastojanje izmedju tacaka A i B je " + rastojanje;
}
```



Логички тип

Логички типови података служе за представљање резултата логичких исказа који могу да буду тачни или нетачни. Логички тип је bool и може да има само две вредности: true(тачно) и false(нетачно) И заузима један бајт.

Логички тип

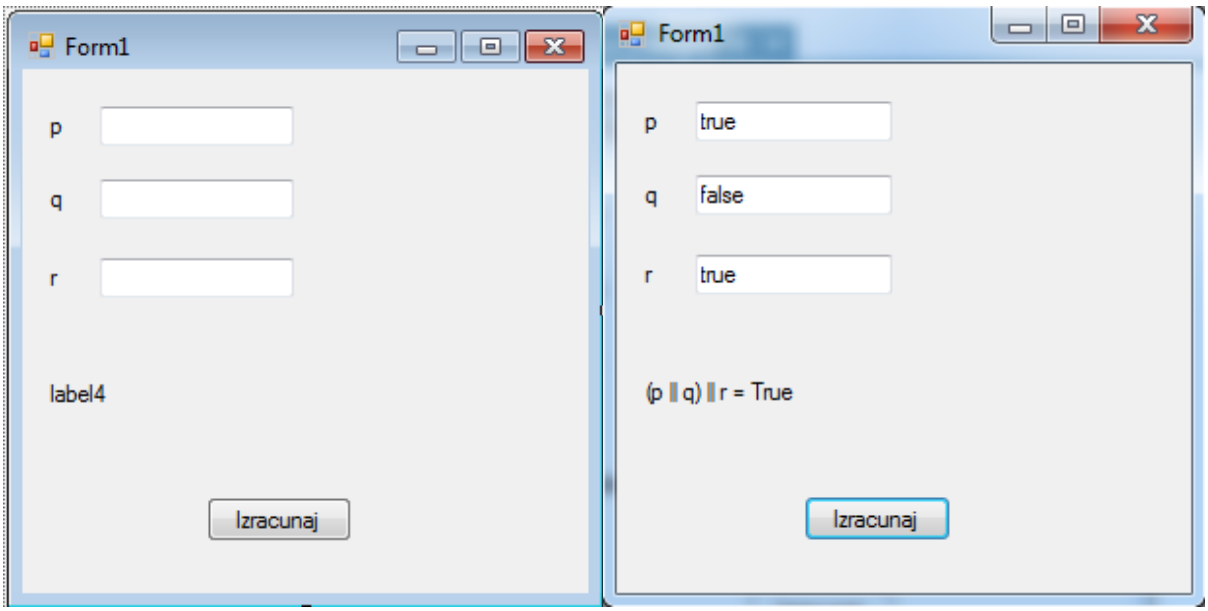
C# тип	Коришћено бајтова	Вредности
bool	1	true или false

Следећи пример показује како да користимо тип bool :

```
bool iskaz = true ;
```

Пример 1. Саставити програм који одређује истиносну вредност формуле (p || q) || r, где су p, q и r искази дате истиносне вредности.

Решење:



```
private void button1_Click(object sender, EventArgs e)
{
    //Uzimamo vrednost za promenljivu p koju je korisnik uneo
    bool p = bool.Parse(textBox1.Text);
    //Uzimamo vrednost za promenljivu q koju je korisnik uneo
    bool q = bool.Parse(textBox2.Text);
    //Uzimamo vrednost za promenljivu r koju je korisnik uneo
    bool r = bool.Parse(textBox3.Text);
    //Po formuli racuna i ispisuje resenje
    label4.Text = "(p || q) || r = " + ((p || q) || r);
}
```

Пример 2. Проверити да ли је израз (p || q) && (q || r) && (p || r) таутологија, тј да ли је за произвољне вредности исказних слова p, q, r, наведени исказ тачан.

Решење:

```
private void button1 Click( object sender, EventArgs e)
{
    bool p =  bool.Parse(textBox1.Text);
    bool q =  bool.Parse(textBox2.Text);
    bool r = bool.Parse(textBox3.Text);
    //Po formuli racuna i ispisuje resenje
    label4.Text= "(p || q) && (q || r) && (p || r)= " +((p || q) && (q || r) && (p ||
r));
}
```

Знаковни тип

Знаковни типови података служе за представљање слова, цифара и специјалних знакова који се могу јавити у разним текстовима. Знаковни типови података char представља 16-битне UNICODE знакове.

Напомена : UNICODE је стандард за електронско кодирање већине светских писаних језика .

Знаковни тип

C# тип	Коришћено бајтова	Вредности
char	2	16 - битни UNICODE знак

Типу char може се доделити знак између једноструких навода. На пример :

```
char mojZnak = 'A' ;
```

Типу char може се доделити и escape знак. Escape знак је знак са специјалним значењем. У наредној табели приказани су escape знакови које дозвољава C#, као и њихова значења.

Escape знакови

Escape знак	Опис
\'	једноструки наводник
\ "	двоструки наводник
\\	усправна коса црта (backslash)
\ 0	нула
\ a	упозорење
\ b	померање за једно место уназад (backspace)
\ f	Команда која даје инструкције штампачу да отпусти тренутни лист (Form feed)
\ n	нова линија
\ r	Знак који враћа курсор или главу стампача на почетак линије (Carriage return)
\ t	хоризонтално увлачење (tab)
\ v	вертикално увлачење

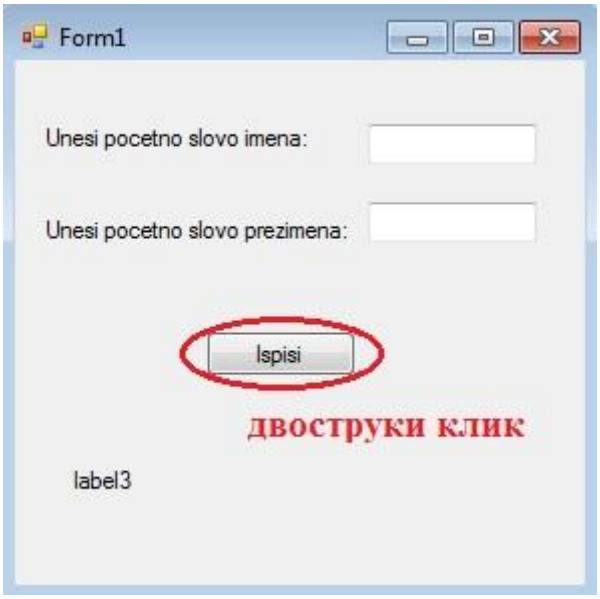
Следећи пример додељује знак за хоризонтално увлачење променљивој mojZnak :

```
char mojZnak = '\ t' ;
```

Пример 1. Стаставити програм који када корисник унесе почетна слова свога имена и презимена враћа његове иницијале .

Решење:

Изглед форме пре покретања програма :



Слика 4.26. Изглед форме пре покретања програма

Направићемо две променљиве ime и prezime у које ћемо да смештамо слова која корисник унесе. Како се ради о словима , њихов тип ће бити char.

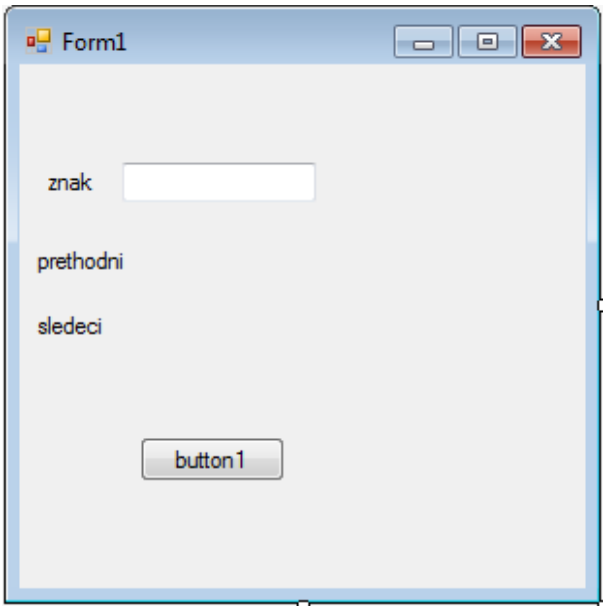
```
private void button1 Click(object sender,  EventArgs e)
{
    //Promenljivoj ime dodeljujemo tip char kao i promeljivoj prezime
    char ime;
    char prezime;
    //Vrednost koju je korisnik uneo preko textBoxa-a smestamo u promenljivu ime koja
je tipa char
    // ali pre toga moramo izvrsiti konvertovanje. Isto ponovimo za promenljivu
prezime .
    ime = char.Parse(textBox1.Text);
    prezime = char.Parse(textBox2.Text);
    //Ispisujemo u labeli slova koja smo smestili u promenljivim ime i prezime
    label3.Text = "Vasi inicijali su: "  + ime + " " + prezime;
}
```

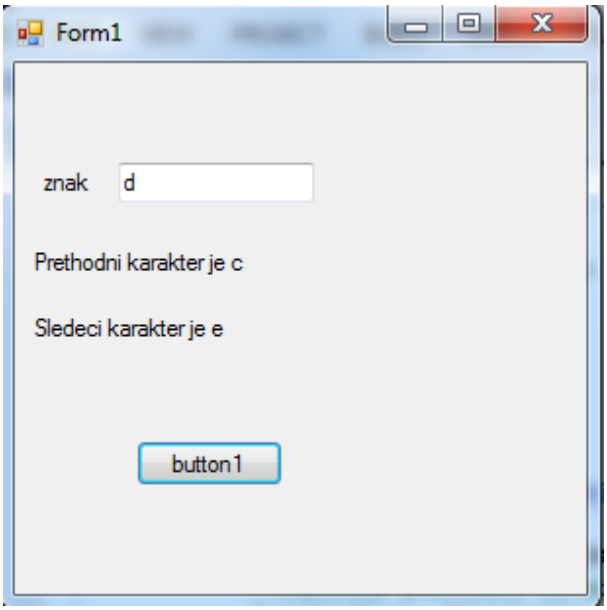
Изглед форме након покретања програма :

Слика 4.27. Изглед форме након покретања програма

Пример 2. Саставити програм који за унети карактер исписује претходни и следећи.

Решење:



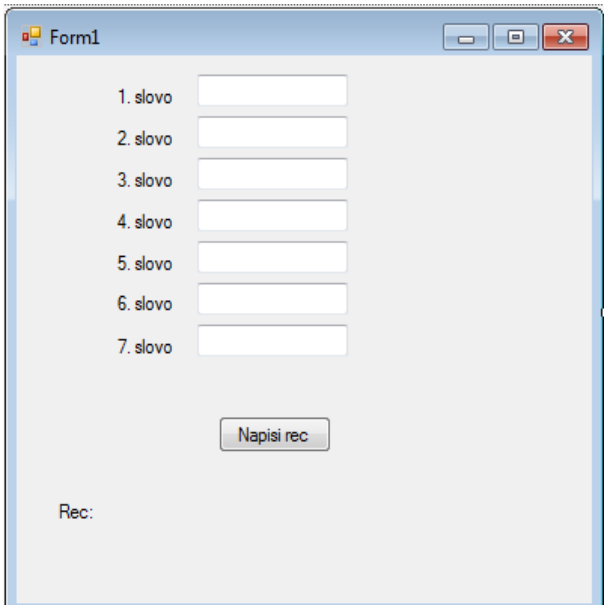


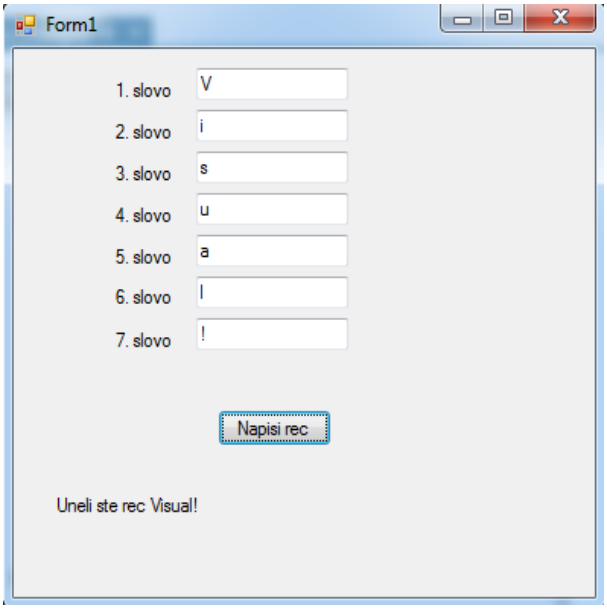

```
private void button1 Click(object sender, EventArgs e)
{
    //Uzimamo vrednost za promenljivu ch koju je korisnik uneo
    char ch = char.Parse(textBox1.Text);
    //Ispisuje resenje prethodnog karaktera
    label2.Text = " Prethodni karakter je : "+ (char)(ch - 1);
    //Ispisuje resenje sledeceg karaktera
    label3.Text = " Sledeci karakter je : " + (char)(ch + 1);
}
```

Пример 3. Саставити програм који унете карактере спаја у реч.

Решење:

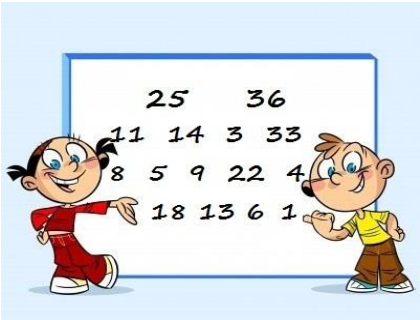
```
private void button1 Click(object sender, EventArgs e)
{
    /*Promenljivima slovo1, slovo2,...slovo7 koje su tipa char dodeljujemo
    vrednosti iz textBox-ova */
    char slovo1 = char.Parse(textBox1.Text);
    char slovo2 = char.Parse(textBox2.Text);
    char slovo3 = char.Parse(textBox3.Text);
    char slovo4 = char.Parse(textBox4.Text);
    char slovo5 = char.Parse(textBox5.Text);
    char slovo6 = char.Parse(textBox6.Text);
    char slovo7 = char.Parse(textBox7.Text);
    //Ispisujemo rezultat u label-u
    label8.Text = "Uneli ste rec " + slovo1 + "" + slovo2 + "" + slovo3 + "" + slovo4
+
    "" + slovo5 + "" + slovo6 + "" + slovo7;
}
```





Низовни тип

Огроман је број проблема који би се употребом простих променљивих врло тешко решио, или се уопште не би могао решити. Често је потребно у програму декларисати и више хиљада променљивих. Увођење толико променљивих на уобичајен начин практично је неизводљив. Због тога се у програмским језицима уводи појам **низа**, тј. **низовни тип**.

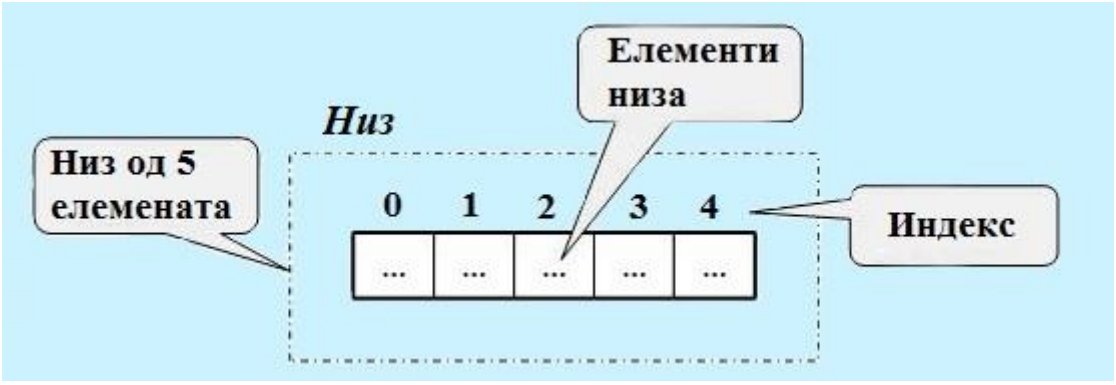


Низовни тип описује ограничен уређен скуп променљивих истог типа, које се називају компоненте (низа).

Ево једног таквог низа, који представља скуп променљивих истог типа...

Компоненте овог низа су бомбоне!

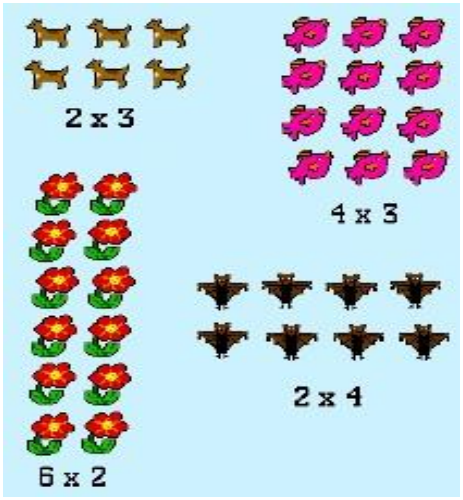
Сваки податак у низу се назива његовим елементом, а сваки елемент има свој *индекс*, односно објекат преко којег прилазимо том елементу у низу.

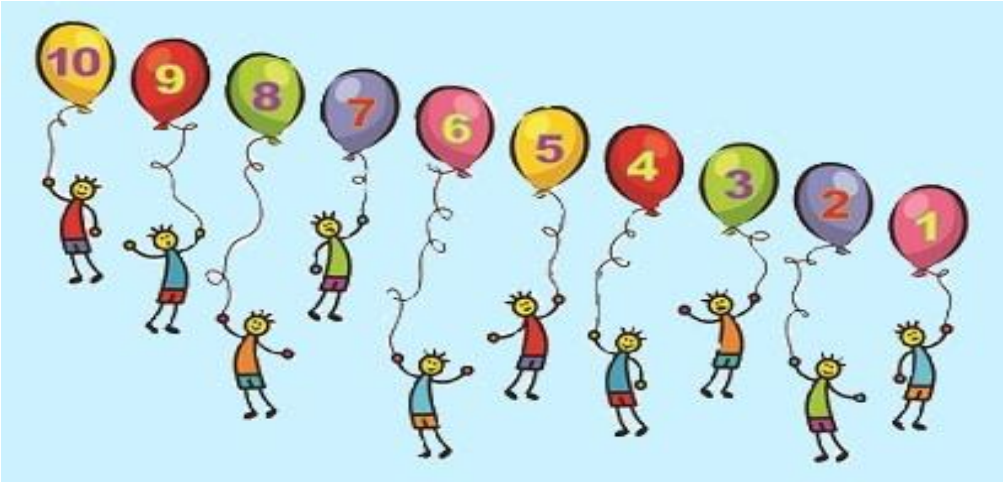


Слика 4.26. Изглед низа

Број елемената у низу представља његову дужину. Тако је дужина овог низа једнака 5. Елементи низа нумерисани су са 0,1,2,3,4. Ови бројеви представљају индексе елемената низа.

Низ може бити једнодимензионалан (када га зовемо једноставно низ), дводимензионалан (када га називамо матрицом због аналогије са истоименим математичким појмом), и вишедимензионалан. Код једнодимензионалног низа, димензија се поистовећује са дужином низа, нпр. кажемо да је низ димензије n. Код више димензионалних низова, међутим, не постоји појам дужине, него се увијек каже да је низ димензија m x n x ... x z или (m,n,...,z). За низ је битно познавати његове димензије да бисмо могли исправно индексирати односно дохватати његове елементе.

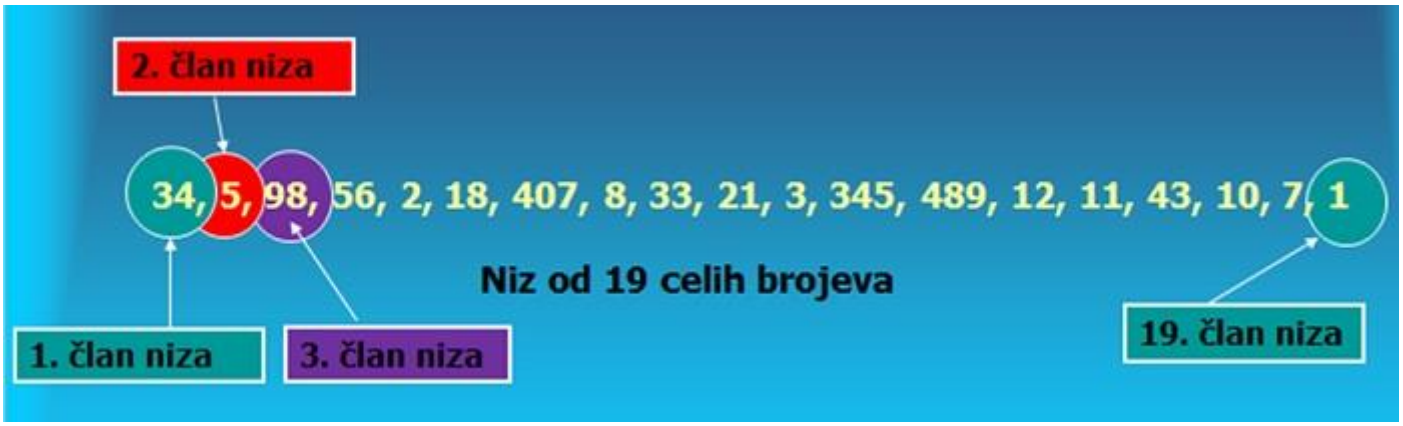




Елементима низа се приступа преко заједничког имена (назив низовне променљиве) и фиксног броја индекса. Стога се низ може посматрати као група индексираних променљивих истог типа са заједничким именом. Сваки елемент низа има онолико индекса колико сам низ има димензија. Тако, елемент једнодимензионалног низа има један индекс, а n-димензионални низ има n индекса.

У случају једнодимензионалних низова величина низа се поистовећује са бројем његових елемената, а индекс елемента се своди на редни број елемента у низу. Притом први индекс је нула.

Ево једног једнодимензионог низа чији су елементи цели бројеви...



Слика 4.27. Чланови низа

Низови се у C#-у декларишу на следећи начин:

```
tip_elementa naziv_promenljive[velicina_niza];
```

Тако на пример,ако бисмо декларисали низ mojNIz од 6 целобројних елемената, то би изгедало овако:

```
int mojNiz[6];
```

У примеру променљива мојНиз је низ који је типа (int []), односно низ целих бројева. У C#-у креирање низа односно алокација меморије врши помоћу речи new.

```
int[] mojNiz = new int[6];
```

У примеру се креира низ од 6 елемената који су цели бројеви. Пре него што почнемо користити елементе низа њима је потребно доделити неку вредност. Крерирањем низа те вредности биће једнака нули. Почетне вредности елемената можемо доделити на различите начине. Ево једног.

```
int[] mojNiz = { 1, 2, 3, 4, 5, 6 };
```

У овом примеру креирање и иницијализација низа је извршена истовремено, па је тако:

```
mojNiz[0]=1;
mojNiz[1]=2;
    .
    .
    .
```

Дводимензионалне низове декларишемо на сличан начин.

```
int[,] intMatrica;
float[,] floatMatrica;
String[,,] strNiz;
```

У датом примеру прва два низа су дводимензионална док је трећи тродимензионални. Креирање вишедимензионланих низова вршимо такође помоћу речи new.

```
int[,] intMatrica =
{
    //Red 0 vrednosti
    {1, 2, 3, 4},
    //Red 1 vrednosti
    {5, 6, 7, 8},
    //Red 2 vrednosti
    {9, 10, 11, 12},
};
//Matrica je dimenzija 3 x 4 (3 red, 4 kolone)
```

Елементима низа приступамо користећи сличну синтаксу као и код декларације, користећи оператор средње заграде:

```
//Deklarisemo jedan niz
int niz[20];
//Deklarisemo jednu pomoćnu promenljivu
int x;
//Dodeljujemo broj 17 prvom elementu niza
niz[0] = 17;
//Dodeljujemo promenljivoj x vrednost prvog elementa niza
```

```
x = niz[0];
```

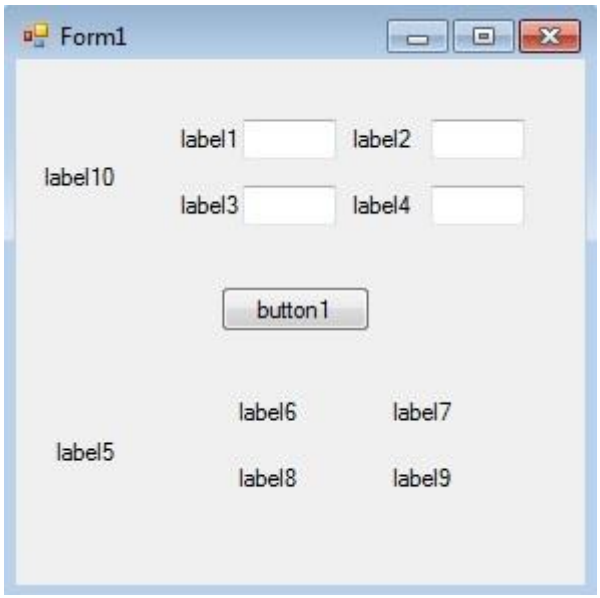
Пример 1. Направити програм који за унете елементе матрице А димензије 2x2 кликом на дугме израчунава А транспоновано .

Решење:

За решавање овог задатка биће нам потребно 10 лабела, једно дугме и 4 textBoxa-а. Како матрица А има четири елемента направићемо низ реалних броје у који ћемо те елементе да сместимо.

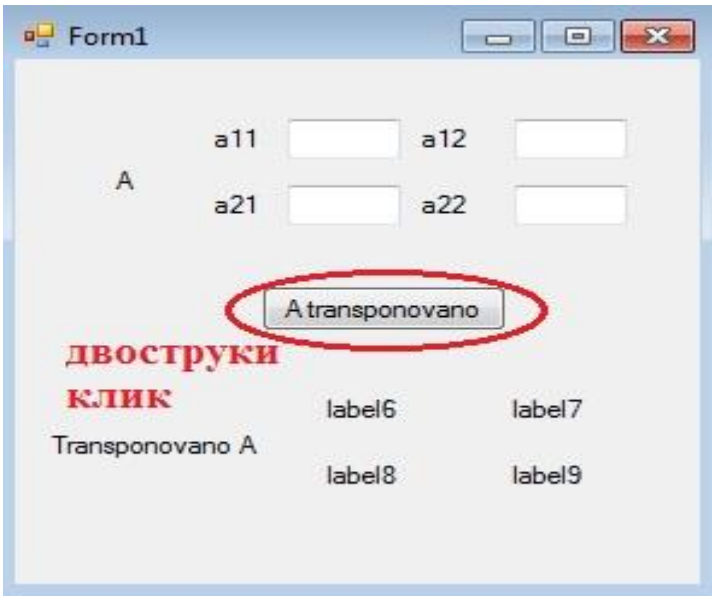
$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Правимо форму облика:



Слика 4.28. Почетни изглед форме

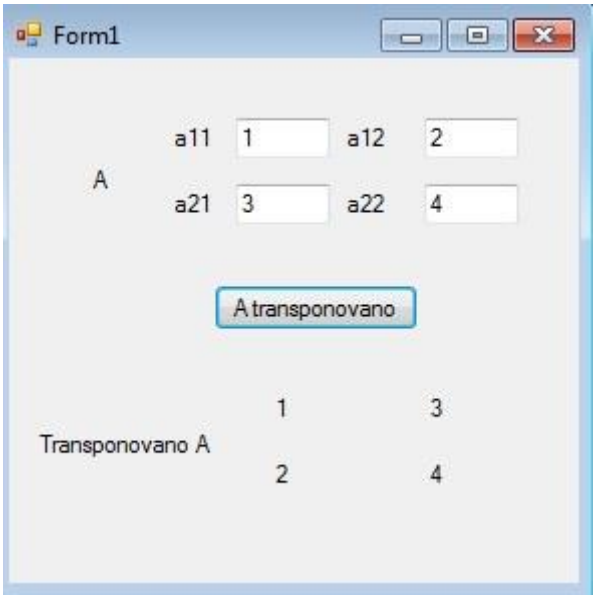
Преименујемо label-е и button1



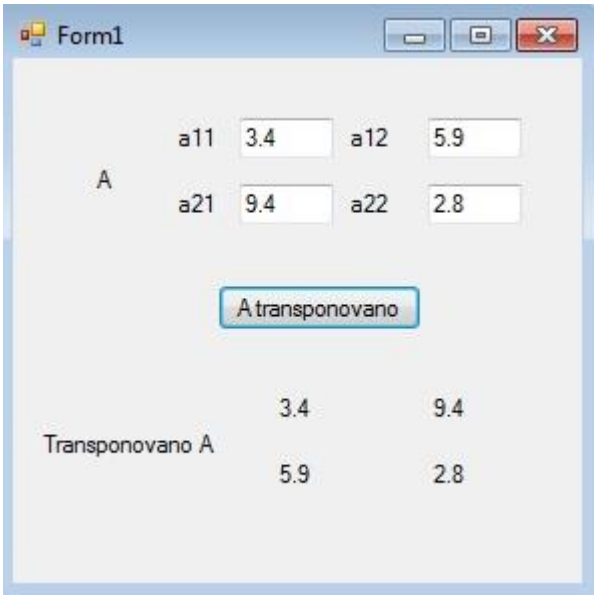
Слика 4.29. Изглед форме након преименовања компоненти

```
float[] clanovi = new float[4];
private void button1_Click(object sender, EventArgs e)
{
    clanovi[0] = float.Parse(textBox1.Text);
    clanovi[1] = float.Parse(textBox2.Text);
    clanovi[2] = float.Parse(textBox3.Text);
    clanovi[3] = float.Parse(textBox4.Text);
    label6.Text = " "  + clanovi[0];
    label7.Text = " "  + clanovi[2];
    label8.Text = " "  + clanovi[1];
    label9.Text = " "  + clanovi[3];
}
```

Изглед форме након покретања програма :



Слика 4.30. Изглед форме након покретања програма



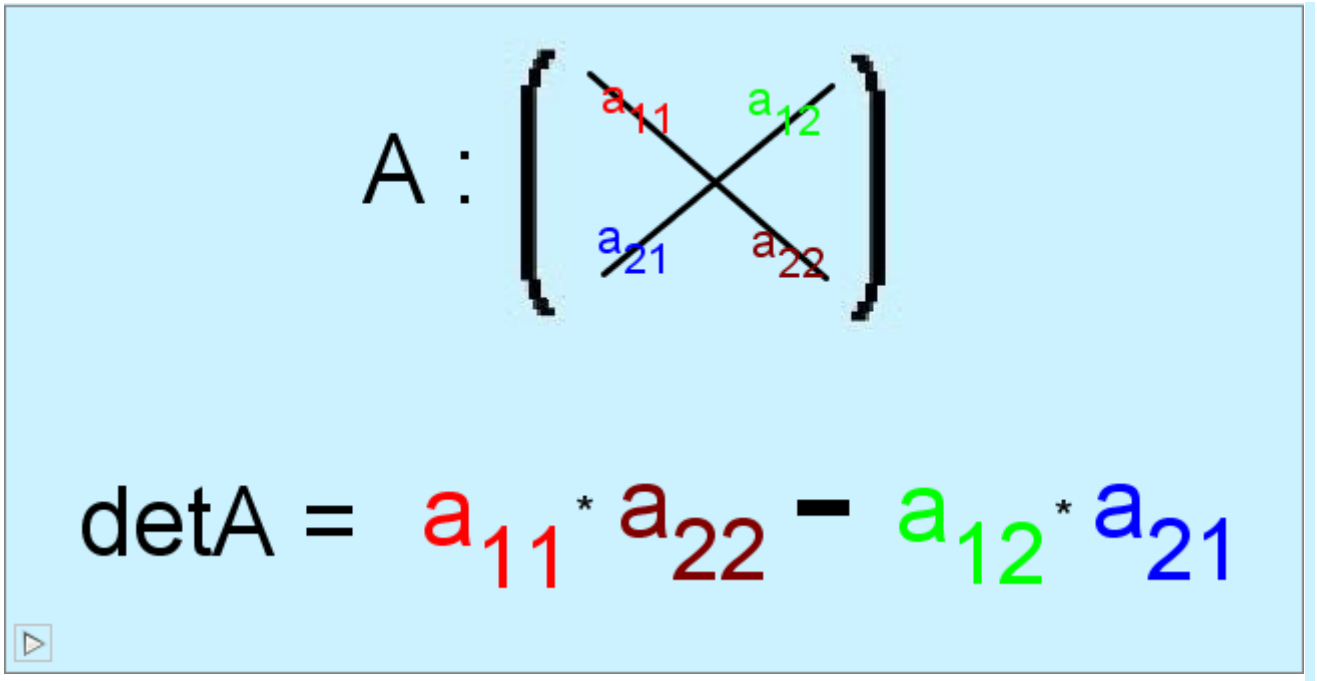
Слика 4.31. Изглед форме након покретања програма

Пример 2. Направити програм који за унете елементе матрице A димензије 2x2 кликом на дугме израчунава детерминантз матрице A .

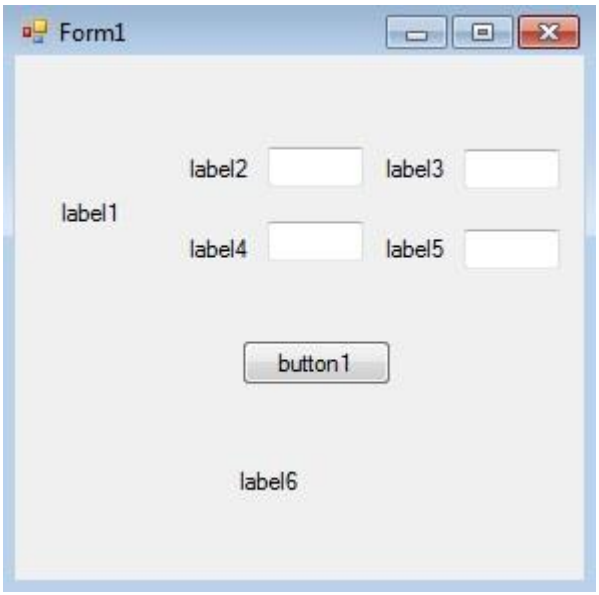
Решење:

За решавање овог задатка биће нам потребно 6 лабела, једно дугме и 4 textBoxa-а. Како матрица A има четири елемента направићемо низ реалних броје у који ћемо те елементе да сместимо.

4.2. Рачунање детерминанте

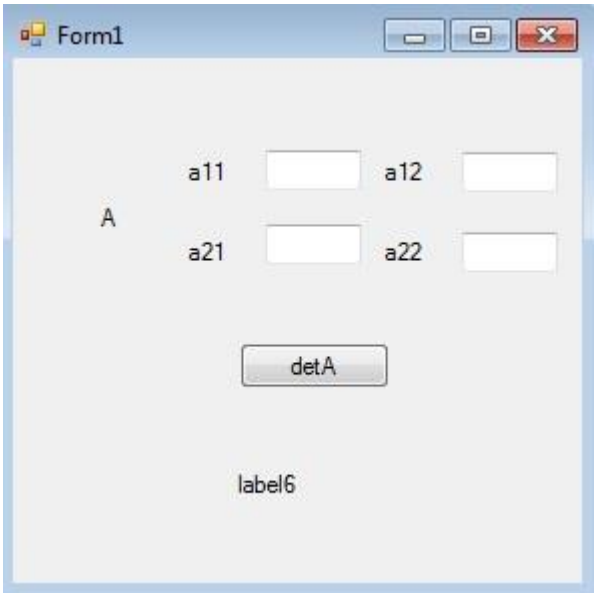


Правимо форму облика:



Слика 4.32. Почетни изглед форме

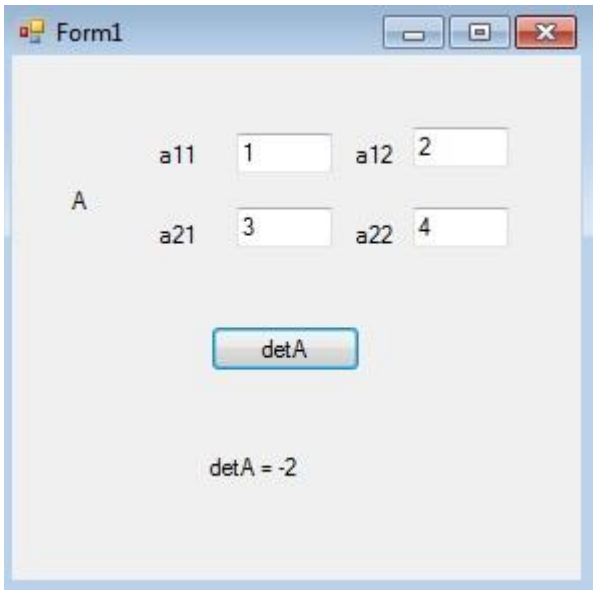
Преименујемо label-е и button1



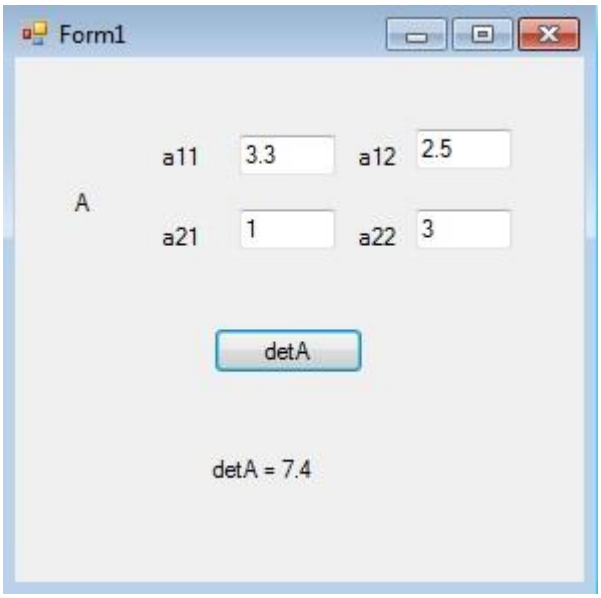
Слика 4.33. Изглед форме након преименовања компоненти

```
float[] clanovi = new float[4];
float det;
private void button1_Click(object sender, EventArgs e)
{
    clanovi[0] = float.Parse(textBox1.Text);
    clanovi[1] = float.Parse(textBox2.Text);
    clanovi[2] = float.Parse(textBox3.Text);
    clanovi[3] = float.Parse(textBox4.Text);
    det = clanovi[0]*clanovi[3] - clanovi[1]*clanovi[2];
    label6.Text = "detA = "  + det;
}
```

Изглед форме након покретања програма :



Слика 4.34. Изглед форме након покретања програма

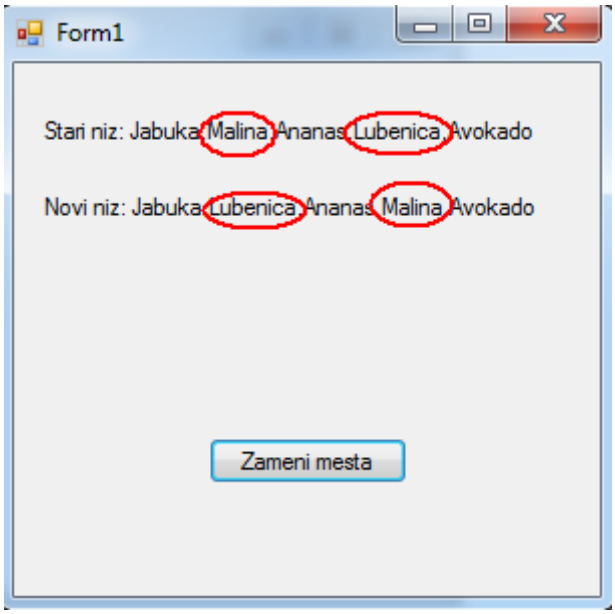


Слика 4.35. Изглед форме након покретања програма

Пример 3. Написати програм који у датом низу члановима на позицијама 2 и 4 замењује места.

Решење:

```
//Deklarisemo niz koji ima 5 clanova
string[] a ={"Jabuka", "Malina", "Ananas", "Lubenica", "Avokado" };
//Deklarisemo pomocnu promenljivu pom
string pom;
private void button1_Click(object sender, EventArgs e)
{
    //Nas niz pre zamene mesta clanova na pozicijama 2 i 4
    label1.Text = "Stari niz: " + a[0] + ", " + a[1] + ", " + a[2] + ", " + a[3] +
    ", " + a[4];
    //Clanovi niza a[] na pozicijama 2 i 4 menjaju mesta
    pom = a[1];
    a[1] = a[3];
    a[3]=pom;
    //Nas niz posle zamene mesta
    label2.Text = "Novi niz: " +a[0]+ ", " + a[1] + ", " + a[2] + ", " + a[3] + ", " +
a[4];
}
```



Пример 4. Написати програм који за унети први члан и разлику аритметичког низа, исписује првих 6 чланова тог низа.

Решење:

```
private void button1_Click(object sender, EventArgs e)
{
    //Deklarisemo niz koji ima 6 clanova
    float[] a = new float[6];
    //Ucitavamo prvi clan niza
    a[0] = float.Parse(textBox1.Text);
    //Razlika dva uzastopna clana
    int d = int.Parse(textBox2.Text);
```

```
    //Ispisujemo niz u labeli
    label1.Text = "Prvih 6 clanova niza a: " + a[0] + ", " + (a[0] + d) + ", " +
    (a[0] + 2*d) + ", " + (a[0] + 3*d) + ", " + (a[0] + 4*d) + ", " + (a[0] + 5*d);
}
```

Стринг тип

Превод речи string на наш језик био би ниска. У програмирању и другим гранама математике, ниска (string) је уређени низ симбола. Стрингови су дакле, низови UNICODE знакова и декларише их коришћење типа string. Стринг је погодан за карактер-по-карактер обрађивање и по томе се разликује од класичних низова. У овом контексту, стринг не мора репрезентовати текст. За промењљиву која је декларисана да је типа ниске, обично се алоцира довољно меморије да се ускладишти одређена количина симбола. Технички, ниске су представљене као низови карактера на чији десни крај се дописује карактер ’\0’ (тзв. терминална нула (енг. null terminator). Из овога разлога, ниске у C#-у се називају ниске терминисане нулом . Последица овога је да не постоји ограничење за дужину ниске у C#-у, али да је неопходно проћи кроз целу ниску како би се одредила њена дужина. Неке ниске су већ унапред дефинисане, као и њихова вредност и значење. Ниске у програмском језику C# се наводе између двоструких наводника. String niska се може затим исписати у конзоли коришћењем следећег позива методе који прихвата string као улаз

```
string niska = " Zdravo svete ! " ;
System.Console.WriteLine( niska );
```

String се може креирати и од других мањих стрингова, коришћењем оператора + . На пример :

```
string niska1 = " Zdravo svete ! " ;
string niska2 = " iz C# ! " ;
string niska3 ;
niska3 = niska1 + niska1 ;
System.Console.WriteLine( niska3 );
```

У промењљиву типа стринг се могу уградити и escape знакови . Escape знак је знак са специјалним значењем. У наредној табели приказани су escape знакови које дозвољава C#, као и њихова значења.

Escape знакови

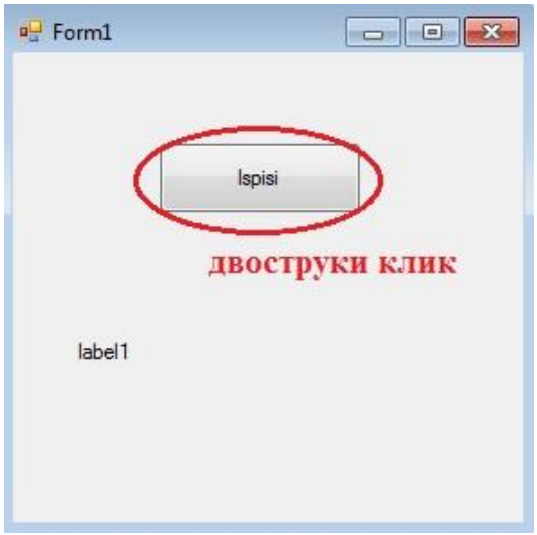
Escape знак	Опис
\'	једноструки наводник
\ "	двоструки наводник
\\	усправна коса црта (backslash)
\ 0	нула
\ a	упозорење
\ b	померање за једно место уназад (backspace)
\ f	Команда која даје инструкције штампачу да отпусти тренутни лист (Form feed)
\ n	нова линија
\ r	Знак који враћа курсор или главу стампача на почетак линије (Carriage return)
\ t	хоризонтално увлачење (tab)
\ v	вертикално увлачење

На пример:

```
string niska = " Zdravo svete ! " + " \n iz C# ! "  ;
System.Console.WriteLine( niska );
```

Пример 1. Написати програм који притиском на дугме исписује у лабелу текст из ниске.

Решење: Изглед форме пре покретања програма је :



Слика 4.35. Изглед форме пре покретања програма

```
private void button1_Click(object sender,  EventArgs e)
{
    string niska = " Zdravo svete  " + " \n iz C# ! "  ;
    label1.Text = niska;
}
```

Изглед форме након покретања програма је :

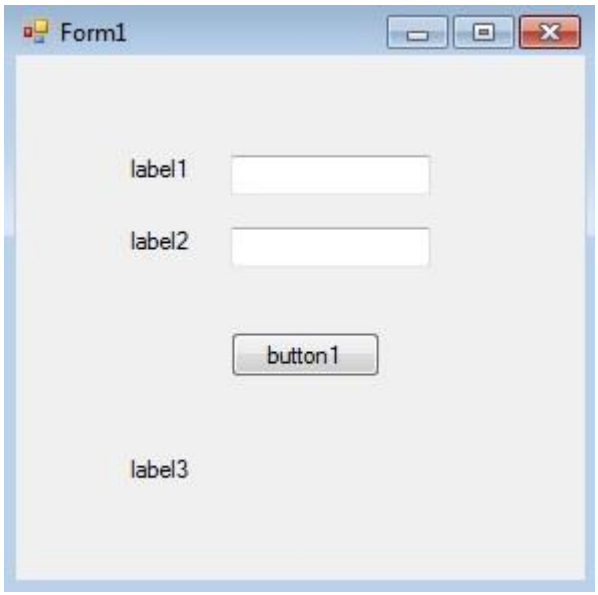
Слика 4.36. Изглед форме након покретања програма

Пример 2. Направити програм који када кликнемо на дугме исписује име и презиме у label - и које корисник унесе са тастатуре у textBox - ове.

Решење:

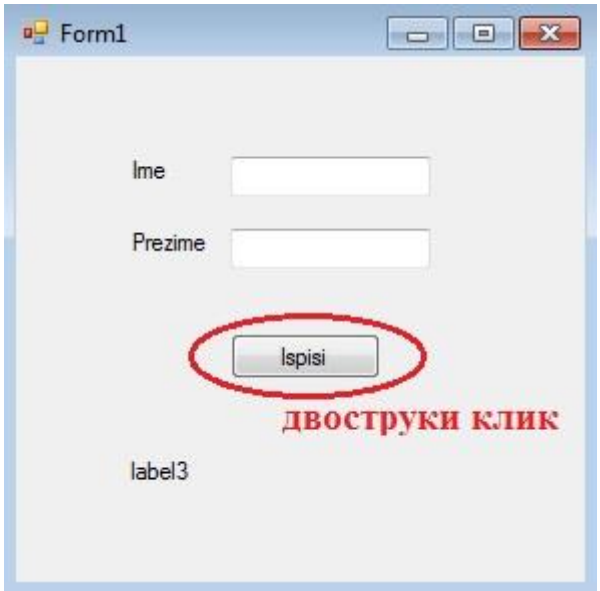
За решавање овог задатка биће нам потребне две променљиве типа string. Једну ћемо назвати ime а другу prezime. Правимо форму кода ће садржати следеће :

Правимо форму облика:



Слика 4.37. Почетни изглед форме

Преименујемо label-е и button1

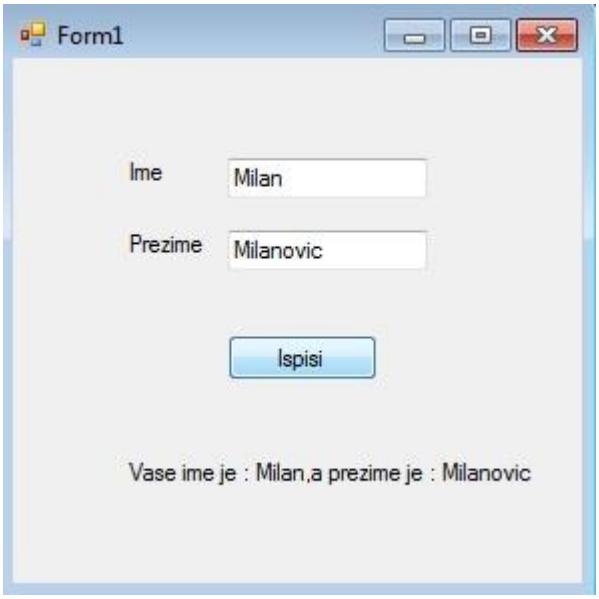


Слика 4.38. Изглед форме након преименовања компоненти

```
//Kreiramo promenljive ime i prezima tipa String
String ime ;
String prezime ;
private void button1_Click(object sender, EventArgs e)
{
    //U promenljive ime i string smestamo ono sto je korisnik uneo u
    //textBox1 i textBox2
    ime = textBox1.Text ;
    prezime = textBox2.Text ;
    label3.Text ="Vase ime je : " + ime +",a prezime je : " + prezime;
}
```

Izgled forme nakon pokretanja programa :

Слика 4.39. Изглед форме након покретања програма



Слика 4.40. Изглед форме након покретања програма

C# има богат скуп функционалности за манипулисањем стринговима тј. у C#-у су уграђене методе за рад са типом стринг. У следећим примерима илустроваћемо нека својства и неке од метода за рад са стринговима.

Коришћење својства lenght :

Својство lenght се користи да би се добио број знакоа у стрингу. Својство lenght враћа вредност типа int. У следећем примеру приказујемо на који начин можемо да видимо колики број знакова у нашем стрингу . Направићемо једну промењиву receniса у којој ћемо исписати Бити или не бити ! . Хоћемо да нам се у конзоли испише колико та реченица садржи знакова.

```
static void Main(string[] args)
{
    //Promenljivu recenica koja je tipa string postavljamo na
    //zeljenu recenicu Biti ili ne biti !
    string recenica = "Biti ili ne biti!";

    //Sada hocemo da nam se u konzoli ispise broj znakova u toj recenici.
    //To postizemo koriscenjem svojstva lengthi o na sledeci nacin :
    System.Console.WriteLine(recenica.Length);
```

```
}
```

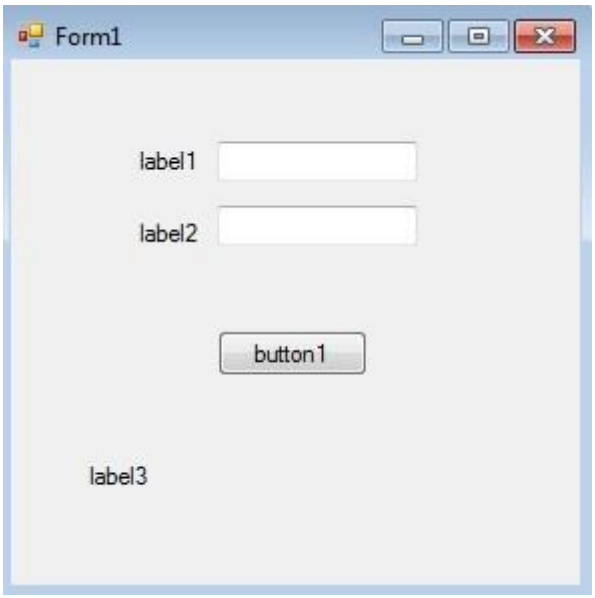
Покретањем овог кода , у конзоли ће се појавити исписан број 17. Зашто ? Наш програм је избројa све знакове у реченици "Бити или не бити !" у којој се поред слова као знакови рачунају и белине и знаци интерпукције . Зато је 17 број занкова у нашој реченици.

Пример 3. Направити програм који када кликнемо на дугме израчунава број слова у имену и презимену које корисник унесе са тастатуре у textBox - ове и исписује га у лабели.

Решење:

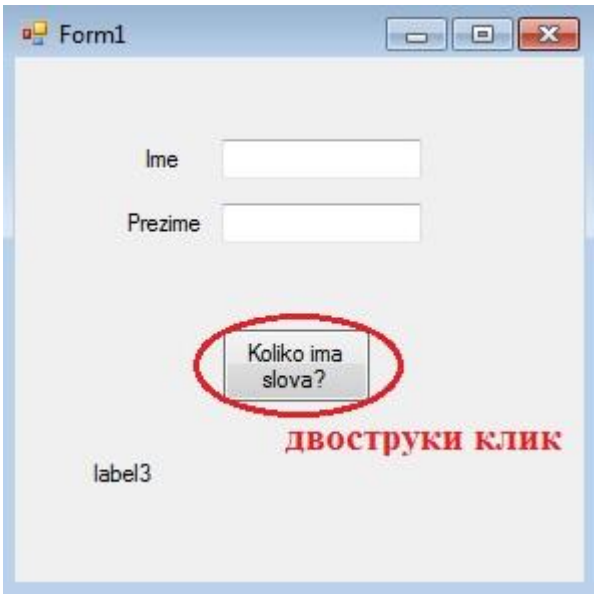
За решавање овог задатка биће нам потребне две променљиве типа string. Једну ћемо назвати ime а другу prezime.У трећу променљиву duzina сместићемо број слова у имену и презимену који је корисник унео користећи својство length . Правимо форму кода ће садржати следеће :

Правимо форму облика:



Слика 4.41. Почетни изглед форме

Преименујемо label-е и button1

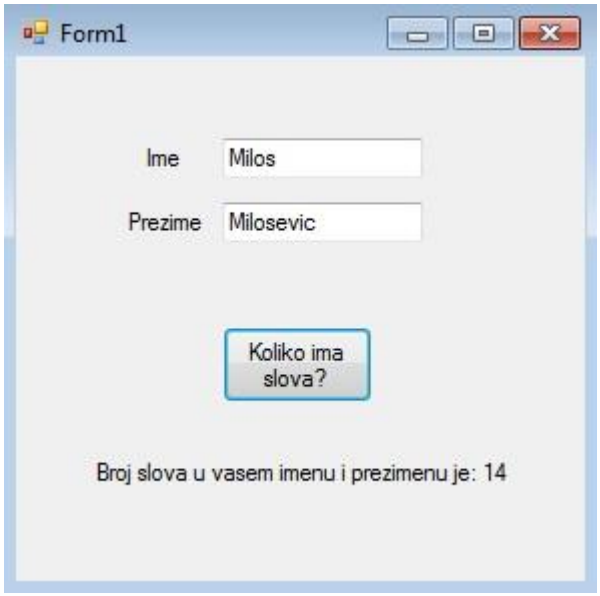


Слика 4.42. Изглед форме након преименовања компоненти

```
//Kreiramo promenljive ime i prezima tipa String i promenljivu duzina tipa int
String ime ;
String prezime ;
int duzina ;
private void button1_Click(object sender, EventArgs e)
{
    //U promenljive ime i prezime smestamo ono sto je korisnik uneo u
    //textBox1 i textBox2
    ime = textBox1.Text ;
    prezime = textBox2.Text ;
    //U promenljivu duzina smestamo broj karaktera u promenljivoj ime i broj karaktera
    //u promenljivoj prezime koristeci svojstvo Length
    duzina = duzina = ime.Length + prezime.Length;
    label3.Text ="Broj slova u vasem imenu i prezimenu je: " + duzina;
}
```

Изглед форме након покретања програма :

Слика 4.43. Изглед форме након покретања програма



Слика 4.44. Изглед форме након покретања програма

Поређење два стринга коришћењем методе Comapre()

Методу Comapre() можемо користити за поређење занкова смештених у све стрига. Метода Comapre() враћа вредност типа int који указује на то да ли је први стринг већи од, једнак или мањи од другог стринга.
Поређење се врши алфабетски над словима и нумерички над било којим бројем који се појави у стрингу. Метода Comapre() користи следеће правила за одређивање вредности, типа int, која се враћа :

1) Ако је први стринг већи од другог стринга, враћа се 1

2) Ако је први стринг једнак другом стринга, враћа се 0

3) Ако је први стринг мањи од другог стринга, враћа се -1

Најједноставнија верзија методе Comapre() прихвата два параметра типа string и користи следећу синтаксу :

String.Compare(niska1 , niska 2);

где су niska1 и niska2 стрингови које желимо да упоредимо. У слдећем примеру показано је коришћење методе Comapre().

```
static void Main(string[] args)
{
    //Hocemo da uporedimo dva stringa, string "bbc" sa stringom "abc"
    //S ozirom da je "bbc" alfabetski veci od stringa "abc", metoda Comapre() vraca
vrednost 1
    //koja se zatim smesta u promenljivu rezultat koja je tipa int
    int rezultat;
    rezultat = String.Compare("abc", "bbc" );
}
```

Коришћење малих и великих слова при поређењу стрингова

Још једна верзија методе Compare() поред две промењљиве типа String, прихвата параметар типа bool који специфицира да ли у поређењу желите да користите и величину слова. Ова верзија методе Compare() користи следећу синтаксу :

String.Compare(niska1 , niska 2 , ignorisiSlova);

gde je ignorisiSlova променљива, типа bool, којом се назначује да ли поређењем желимо да користимо и величину слова стрингова niska1 и niska 2 . Ако поставимо вредност променљиве ignorisiSlova на true, онда се величина слова два стринга не разматра при поређењу (ово је подразумевана вредност). Ако поставимо ignorisiSlova на false, онда се при поређењу разматра величина слова.

У следећем примеру метода Compare() враћа 0, јер се није узимала величина слова при поређењу стрингова "bbc" и "BBC" :

```
static void Main(string[] args)
{
    //Hocemo da uporedimo dva stringa, string "BBC" sa stringom "bbc"
    //ali da pri tom ne vodimo racuna o velicini slova. Koristicemo metodu Comare()
koja
    //prima 3 parematrea , od koga su prva dva tipa string, a trci je tipa bool i
odredjuje
    //da li ce se velicina slova uzimati u obzir ili ne
    //Vrednost koju metoda Compare() vraca, smestamo u promenljivu rezultat tipa int
    int rezultat;
    rezultat = String.Compare("bbc", "BBC" ,true);
    //Vrednost promenljive rezultata je 0 jer smo odlucili da u poredjenje ne
ukljucemo
    //velicinu slova
}
```

А у следећем примеру метода Compare() враћа -1, јер када се не игнорише величина слова "bbc" је мање од "BBC" :

```
static void Main(string[] args)
{
    int rezultat;
    rezultat = String.Compare("bbc", "BBC" ,false);
    //Vrednost promenljive rezultata je -1 jer smo odlucili da u poredjenje ukljucimo
    //velicinu slova
}
```

Конверзија величине слова стринга коришћењем метода ToUpper() и ToLower()

Методе ToUpper() и ToLower() могу се користити да би се конвертовала величина слова у стрингу. Метода ToUpper() враћа стринг са свим великим словаима, а метода ToLower() враћа стринг са свим малим словима. Обе методе враћају нови стринг. Најједноставније верзије ових метода користе следећу синтаксу :

niska1.ToUpper()

niska2.ToLower()

Следећо примери користе методе ToUpper( ) и ToLower( ):

```
//U promenljivu novaNiska1 smestamo sve iz niska1 samo napisano VELIKIM slovima
String novaNiska1 = niska1.ToUpper() ;
//U promenljivu novaNiska2 smestamo sve iz niska2 samo napisano malim slovima
String novaNiska2 = niska1.ToLower() ;
```

Ако је промељива niska1 била постављена на:

String niska1= "Biti ili ne biti";

Онда предходне линије кода дају нове променљиве novaNiska1 и novaNiska2 које су постављене на следеће стрингове, респективно:

"BITI ILI NE BITI"

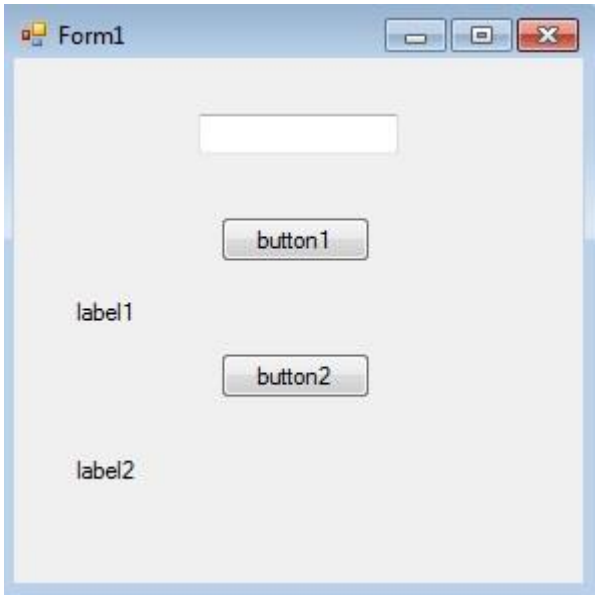
"biti ili ne biti"

Пример 4. Направити програм који када кликнемо на дугме реченицу која је унета у textBox исисује малим словима у лабели (креирати и друго дугме које кликом на њефа исписује реченицу у којој су сва слова велика).

Решење:

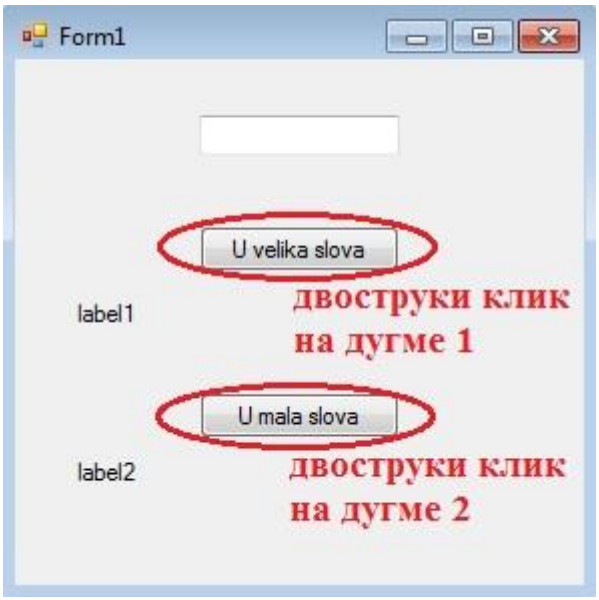
За решавање овог задатка биће нам потребна променљиве типа string у коју ћемо сместити реченицу коју корисник унесе у textBox. Конвертовање слова у мала или у велика постижемо коришћенем метода ToUpper( ) и ToLower( ).

Правимо форму облика:



Слика 4.45. Почетни изглед форме

Преименујемо button1 и button2



Слика 4.46. Изглед форме након преименовања компоненти

```
//Kreiramo promenljivu recenica tipa String
String recenica ;
private void button1_Click(object sender, EventArgs e)
{
    //U promenljive recenica smestamo ono sto je korisnik uneo u textBox1
    recenica = textBox1.Text;
    //U label1 ispisujemo recenicu velikim slovima
    label1.Text = recenica.ToUpper();
```

```
}
private void button2_Click(object sender, EventArgs e)
{
    //U label2 ispisujemo recenicu malim slovima
    label2.Text = recenica.ToLower();
}
```

Изглед форме након покретања програма :

Слика 4.47. Изглед форме након покретања програма

Слика 4.48. Изглед форме након покретања програма

Повезивање сринговa коришћењем методе Concat( )

Методу Concat () можемо користити за повезивање стрингова. Метода Concat () враћа нови стринг који се добија тако што се на крају предгодног стринга додаје сваки нови обезбеђени стринг. Њена најједноставнија верзија прихвата два стринга коришћењем следеће синтаксе:

String.Concat(niska1, niska2)

где су niska1 и niska2 стрингови које желимо да повежемо. У следећем примеру коришћена је ова верзија методе Concat( ) како би повезали два стринга, "Dobar" и "dan" , при чему се стринг који се враћа смешта у променљиву novaNiska :

```
//Pravimo novu promenljivu novaNiska u koju smestamo ono sto zelimo da povezemo
string novaNiska = String.Concat("Dobar ", "dan") ;
```

Сада ће наш стринг novaNiska бити постављен на "Dobar dan" .

Методи Concat() може се проследити било који број стирнгова које желимо да повежемо и следећој верзији методе Concat( ) прослеђена су јој четири стринга :

```
string novaNiska1 = String.Concat("Dobar ", "dan", " svima", " !") ;
```

Сада ће наш стринг novaNiska1 бити постављен на "Dobar dan svima !" .

Повезивање сринговa коришћењем оператора сабирања

За повезивање стрингова може се користити оператор сабирања (+). На пример :

```
string novaNiska = "Dobar" + " dan" + " svima" + " !";
```

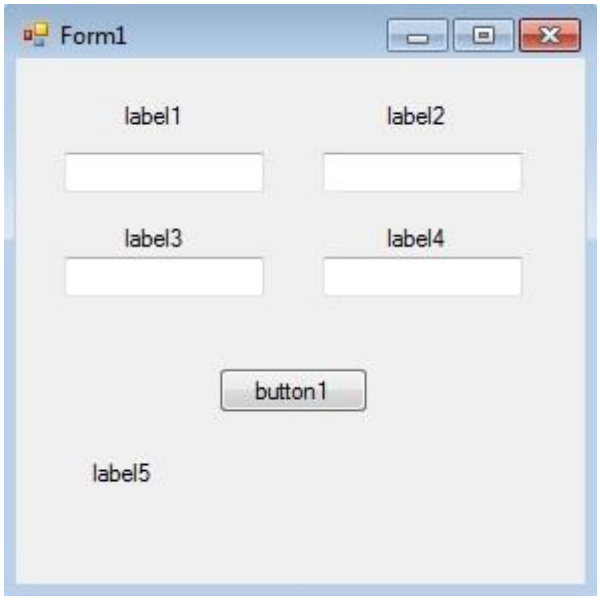
Сада ће наш стринг novaNiska бити постављен на "Dobar dan svima !" .

Пример 5. Направити програм који када кликнемо на дугме повезује 4 речи у реченицу које корисник унесе са тастатуре у textBox - ове а затим ту речрницу исписује у лабели.

Решење:

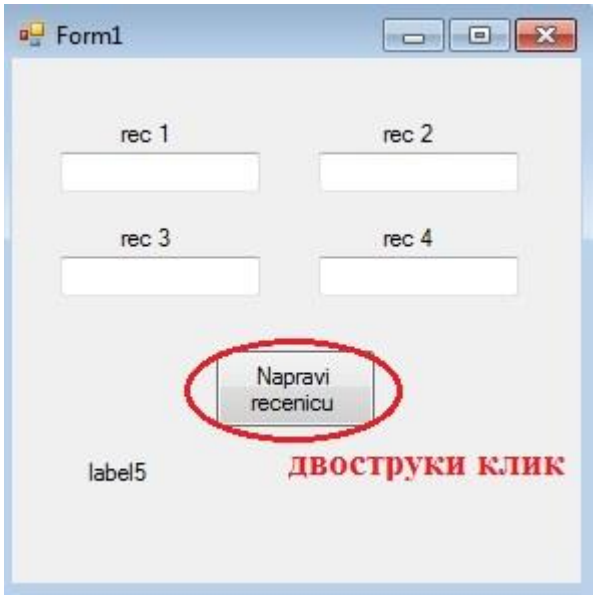
За решавање овог задатка биће нам потребна 4 textBox - а, 5 лабела и једно дугме. Направићемо 5 променљивих (rec1, rec2, rec3, rec4 и recenica) типа String у које ћемо сместити оно што корисник унеде у textBox - ове а у променљиву recenica ћемо сместити реченицу направљену од rec1, rec2, rec3 и rec4. За повезивање променљивих rec1, rec2, rec3 и rec4 користићемо метод Concat( ).

Правимо форму облика:



Слика 4.49. Почетни изглед форме

Преименујемо label-е и button1



Слика 4.50. Изглед форме након преименовања компоненти

```
//Kreiramo promenljive rec1, rec2, rec3, rec4 i recenica tipa String
String rec1, rec2, rec3, rec4, recenica;
private void button1_Click(object sender, EventArgs e)
{
    //U promenljive rec1, rec2, rec3, rec4 i recenica smestamo ono
    //sto korisnik unese u odgovarajuce textBox - ove
    rec1 = textBox1.Text;
    rec2 = textBox2.Text;
    rec3 = textBox3.Text;
    rec4 = textBox4.Text;;
    //U promenljivu recenica,koristeci metod Concat(), rec1, rec2, rec3 i rec4
    recenica = String.Concat(rec1, rec2, rec3, rec4);
    label5.Text = recenica;
}
```

Изглед форме након покретања програма :

Слика 4.51. Изглед форме након покретања програма

Слика 4.52. Изглед форме након покретања програма

Копирње стрингова коришћењем методе Copy()

Методу Copy() можемо користити да бисмо ископирали неки жељени стринг. Метода Copy() користи следећу синтаксу :

String.Copy(niska1)

где је niska1 стринг који желимо да ископирамо.

У следећем примеру приказан је начин коришћења методе Copy( ) за копирање стринга niska1 у стринг niska2

```
string niska2 = String.Copy( niska1 ) ;
```

Копирње стрингова коришћењем оператора додељивања (=)

Стрингови можемо копирати коришћењем оператора додељивања (=) којим се копира један стринг у други. На пример :

```
string niska1 = "Zdravo svete !" ;
string niska2 = niska1 ;
```

Сада ће наш стринг niska2 такође бити постављен на "Zdravo svete !" .

Провера да ли су два стригнга једнака коришћењем методе Equals()

Методу Equals() можемо користити да бисмо проверили да ли су два стринга једнака. Метода Equals() враћа вредност типа bool и има има статичку верзију и верзију истанце. Статична верзија методе Equals() може се позвати преко класе String и она прихвата два стринг параметра, за које се проверава да ли су исти. Синтакса статичне методе Equals() је :

String.Equals(niska1, niska2)

где су niska1 и niska2 два стринга које желимо да поредимо .

У слдећем примеру метода Equals() враћа true јер су два стринга која желимо да упоредимо једнака :

```
//Proveravamo da li string "abc" jednak stringu "abc"
bool rezulataPoredjenja = String.Equals("abc","abc") ;
```

Верзија инстанце методе Equals() позива се коришћењем стварног стринга и она пореди стринг са обезбеђеним стринг параметром. Синтакса ове верзије Equals() је :

niska1.Equals(niska2)

где су niska1 и niska2 стрингови које желимо да упоредимо.

У следећем примеру, метода Equals() враћа false, јер је садржај промењивих niska1 и niska2 различит :

```
//Proveravamo da li string niska1 jednak stringu niska2
bool rezulataPoredjenja = niska1.Equals( niska2 ) ;
```

Провера да ли су два стринга једнака коришћењем једнака оператора (=)

Да бисмо проверили да ли су два стринга једнака можемо користити оператор једнакости (=). У следећем примеру промењива rezulataPoredjenja добија вредност false јер је садржај промењивих niska1 и niska2 различит :

```
//Proveravamo da li string niska1 jednak stringu niska2
bool rezulataPoredjenja = niska1 == niska2 ;
```

Извлачење подстринга из стринга коришћењем методе Substring()

Методу Substring() може се користити да би се из стринга извукао подстринг. Метода Substring() враћа стринг. Најједноставнија верзија методе Substring() враћа подстринг који почиње од специфичног индекса и користи следећу синтаксу:

niska1.Substring(index)

где је index вредност типа int која специфира позицију знака од које треба почети са читањем стринга niska1. У следећем примеру користи се верзија методе Substring(), која враћа подстринг стринга niska1, почев од индекса 5.

```
string niska2 = niska1.Substring(5);
```

С обзиром да је промењива niska1 постављена на "Biti ili ne biti", промењива niska2 се поставља на "ili ne biti".

Друга верзија методе Substring(), за специфицирање беоја знакова које треба прочитати користи још један параметар типа int и користи следећу синтаксу:

niska1.Substring(index, brojKaraktera)

У следећем примеру користи се ова верзија Substring():

```
string niska3 = niska1.Substring(5,3);
```

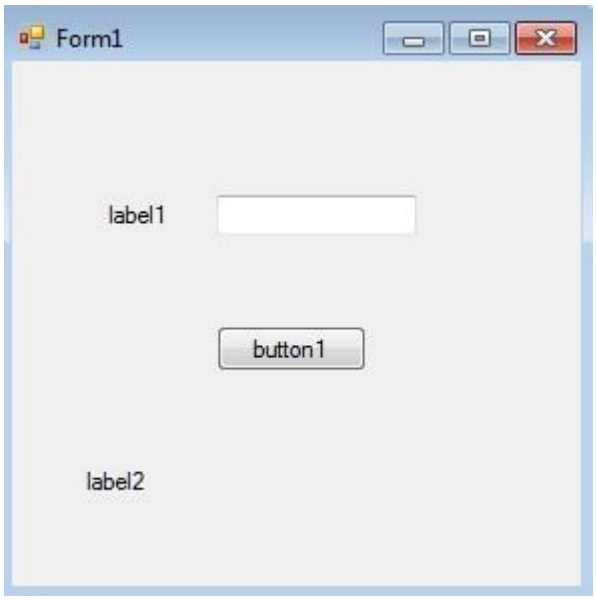
Овде је, промењива niska3 постављена на "ili" .

Пример 6. Направити програм који када кликнемо на дугме извлачи из задате ниске све преостале карактера почевши од оног са индексом 4.

Решење:

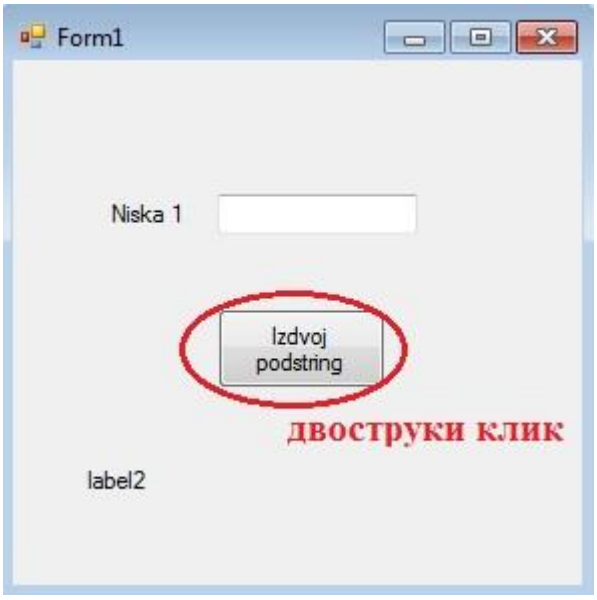
За решавање овог задатка биће нам потребан један textBox и 2 лабеле и једно дугме. Направићемо промењиве niska1 типа String у коју ћемо сместити оно што корисник унесе у textBox и промењиву niska2 у коју ћемо сместити подстринг од промењиве niska1. За извлачење подстринга из стринга niska1 користићемо метод Substring( ).

Правимо форму облика:



Слика 4.53. Почетни изглед форме

Преименујемо label-е и button1



Слика 4.54. Изглед форме након преименовања компоненти

```
//Kreiramo promenljive niska1 i niska2 tipa String
String niska1, niska2;
private void button1 Click(object sender, EventArgs e)
{
    //U promenljivu niska1 smestamo ono sto korisnik unese u textBox
    niska1 = textBox1.Text;
    //U promenljivu niska2 smestamo deo niske niska1 metodom Substring()
    niska2 = niska1.Substring(4);
    label2.Text = niska2;
}
```

Изглед форме након покретања програма :

Слика 4.55. Изглед форме након покретања програма

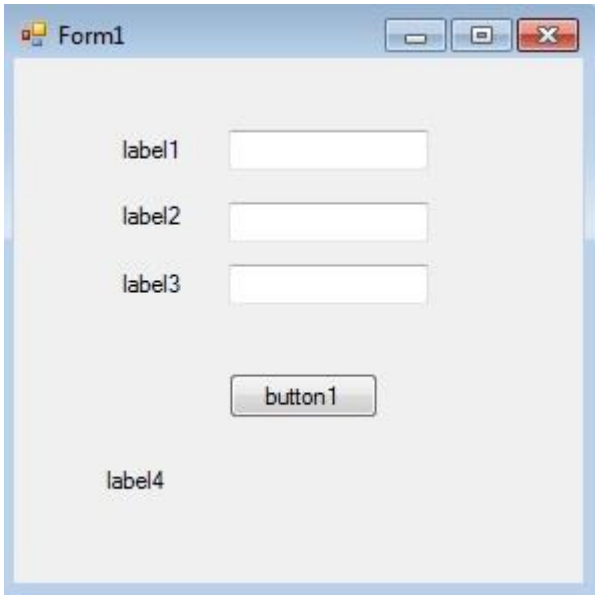
Слика 4.56. Изглед форме након покретања програма

Пример 7. Направити програм који када кликнемо на дугме извлачи из задате ниске карактере почевши од оног са индексом који корисник унесе у textBox, при чему корисник такође уноси и број знакова које треба издвојити из задатог стринга.

Решење:

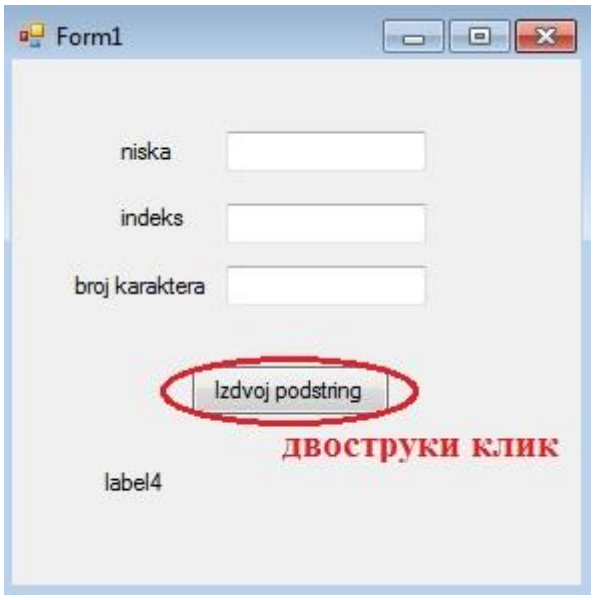
За решавање овог задатка биће нам потребна 3 textBox-а,4 лабеле и једно дугме. Направићемо променљиве niska1 типа String у коју ћемо сместити оно што корисник унесе у textBox1 и променљиву index у коју ћемо сместити индекс који корисник унесе у textBox2, и променљиву brojKaraktera у коју ћемо сместити оно што корисник унесе у textBox3. Кликом на дугме, ново добијена ниска ће се исписати у лабелу.

Правимо форму облика:



Слика 4.57. Почетни изглед форме

Преименујемо label-е и button1



Слика 4.58. Изглед форме након преименовања компоненти

```
//Kreiramo promenljive niska i podstring tipa String i
//promenljive index i brojKaraktera tipa int
String niska;
String podstring;
int index;
int brojKaraktera;
private void button1_Click(object sender, EventArgs e)
{
    //U promenljivu niska smestamo ono sto korisnik unese u texbox1
    niska = textBox1.Text;
    //U promenljivu index smestamo ono sto korisnik unese u textBox2 ali
    //to prvo moramo konvertovati u odgovarajuci tip sto je u ovom slucaju int
    index = int.Parse(textBox2.Text);
    // Slicno uradimo i za promenljivu brojKaraktera
    brojKaraktera = int.Parse(textBox3.Text);
    //Koristeci metod Substring() u promenljivu podstring smestamo
    //string koji se dobija izvlacenjem dela stringa iz promenljive niska
    podstring = niska.Substring(index, brojKaraktera);
```

```
    //Ono sto smo dobili u promenljivoj podstring, ispisujemo u lebeli
    label4.Text = podstring;
}
```

Изглед форме након покретања програма :

Слика 4.59. Изглед форме након покретања програма

Слика 4.60. Изглед форме након покретања програма

Примери

Пример 1. Саставити програм који са тастатуре прихвата једну реч и једно слово, а затим одређује дужину уписане речи и редни број уписаног слова ако је у речи.

Решење:

```
private void button1_Click(object sender, EventArgs e)
{
    //Uzimamo vrednost za promenljivu rec koju je korisnik uneo
    string rec = textBox1.Text;
    //Uzimamo vrednost za promenljivu slovo koju je korisnik uneo
    string slovo =  textBox2.Text;
    //Ispisuje duzinu reci
    label3.Text = "Duzina reci: " + rec.Length;
    //Ispisuje poziciju slova u reci
    label4.Text = "Pozicija slova u reci: " + (rec.IndexOf(slovo) + 1);
}
```

Пример 2. Саставити програм који у уписаној речи налази број појављивања уписаног слова у тој речи.

Решење:

```
private void button1_Click(object sender, EventArgs e)
{
    //Uzimamo vrednost za promenljivu rec koju je korisnik uneo
    string rec = textBox1.Text;
```

```
    //Uzimamo vrednost za promenljivu broj koju je korisnik uneo
    char broj = char.Parse(textBox2.Text);
    //Deklarisemo brojac
    int i;
    //Deklarisemo promenljivu broj_pojavljivanja
    int broj pojavljivanja;
    broj_pojavljivanja = 0;
    //Prolazimo kroz petlju
    for (i = 0; i < rec.Length; i++)
    //Proveravamo da li je rec[i] == slovo
    if (rec[i] == slovo)
        broj_pojavljivanja++;
    //Rezultat ispisujemo u labeli 3.
    label3.Text = "Broj pojavljivanja slova u reci: " + broj pojavljivanja;
}
```

Набројиви тип

Набрајање нам омогућава да креирамо скуп константи на које можемо да референцирамо у програму. Набрајање је вредносни тип. Рецимо да пишемо програм који ће се користити у класи астрономије и да програм треба да се референцира на редослед планета у соларном систему у односу на Сунце. Редослед је: Меркур, Венера, Земља, Марс, Јупитер, Сатурн, Нептун и Плутон.

Набрајање се декларише коришћењем кључне речи . У следећој поједностављеној синтакси приказано је како се креира набрајање:

enum ime_nabrajanja { lista-konstanti }

ime_nabrajanja : Име које се додељује набрајању. По конвенцији, прво слово имена набрајања је велико.

lista_konstanti : Листа константи које се налазе у набрајању.

У следећем примеру декларисано је набрајање названо Planete:

```
enum Planete

{
    Merkur,
    Venera,
    Zemlja,
    Mars,
    Jupiter,
    Saturn,
    Uran,
    Neptun,
    Pluton

}
```

Као што можемо видети, набрајањем Planete дефинише се девет константи, при чему свака константа представља позицију планете у односу на Сунце. Прва константа, Merkur, има подразумевану (default) вредност 0. Остале константе имају за 1 већу вредност у односу на предходну вредност, тако да је константа Venera постављена на 2, константа Zemlja на 3, и тако даље, до константе Pluton, која је постављена на 8. Подразумевани тип за константе у набрајању је int.

Константе у набрајању можемо и иницијализовати, као што је приказано у следећем примеру:

```
enum Planete

{
    Merkur = 1,
    Venera = 2,
    Zemlja = 3,
    Mars = 4,
    Jupiter = 5,
    Saturn = 6,
    Uran = 7,
    Neptun = 8,
    Pluton = 9

 }
```

Први упис поставља константу Merkur на вредност 1. Као и раније, друге константе се постављају на вредност за 1 већу у односу на предходну вредност, тако да је константа Venera постављена на 2, константа Zemlja на 3, и тако даље све до константе Pluton, која је постављена на 9.

Да бисмо приступили елементу, користимо нотацију тачке. На пример, да бисмо приступили елементу Zemlja, користили бисмо Planete.Zemlja. У следећем примеру приказана је позиција Земље у односу на Сунце:

```
System.Console.WriteLine( "Pozicija Zemlje = " + (int)Planete.Zemlja);
```

Обратимо пажњу на употребу оператора за конверзију, да би се добила вредност на коју је Земља постављена у набрајању. Овом наредбом приказан је следећи израз:

Pozicija Zemlje = 3;

Ако не одрадимо конверзију оператором за конверзију, онда ће Planete.Zemlja из претходне наредбе вратити стринг "Zemlja" .

Спецификирање вредности у набрајању :

Можемо спецификирати вредност константи у набрајању. На пример, следећа енумерација, названа PeriodicnostPlaneta, дефинише орбиталне периоде (време које је потребно планети да обиђе Сунце), за прве четири планете; орбитални периоди су у изражени у данима.

```
enum PeriodicnostPlaneta

{
    Merkur = 88,
    Venera = 225,
    Zemlja = 365,
    Mars = 687,
}
```

На следећи начин можемо приказати орбитални период Марса:

```
System.Console.WriteLine( " Orbitalni period za Mars je : " +
(int)PeriodicnostPlaneta.Mars+ " dana "  );
```

Спецификирање основног типа у набрајању :

Као што смо раније поменули, подразумевана вредност sa тип енумерације је int . Тип који користи енумерација познат је као основни тип *основни тип*. У суштини, као основни тип енумерације можемо користити било који целобројни тип. Да бисмо поставили основни тип.ставимо тип након имена енумерације и додајмо две тачке испред типа. На пример, у следећој енумерацији коришћен је тип long као њен основни тип:

```
enum PeriodicnostPlaneta: long

{
    Merkur = 88,
    Venera = 225,
    Zemlja = 365,
    Mars = 687,
}
```

Напомена: Иста константа се не може користити у декларацијама различитих набројивих типова података. На пример, дефиниције типова :

enum radnidani { pon, uto, sre, cet, pet };

enum vikend { pet, sub, ned };

су НЕКОРЕКТНЕ, јер се константа pet појављује у дефиницији оба типа .

Ако желимо да добијемо назив вредности нашег enuma можемо да користимо следећу линију кода :

```
(( PeriodicnostPlaneta)88).ToString();
```

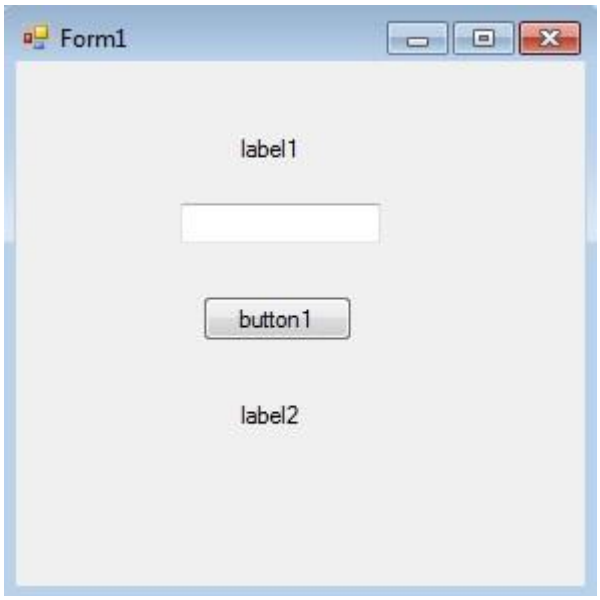
Ова линија кода би за повратну вредност имала стринг који би био : "Merkur" , уколико бисмо користили претходну енумерацију. На следећем примеру видећемо како можемо користити набројиви тип.

Пример 1. Направити програм који за задати редни број дана у недељи исписује њег ово име .

Решење:

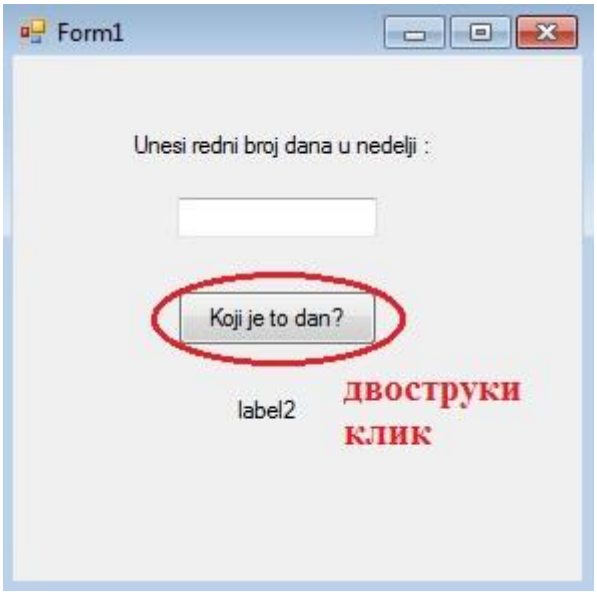
Направићемо, користећи набројиви тип, енумерацију Dani која ће садржати елементе : ponedeljak, utorak, sreda, cetvrtak, petak , subota, nedelja који ће редом имати вредности 1, 2, 3, 4, 5, 6, 7 . Пре тога направимо форму:

Правимо форму облика:



Слика 4.61. Почетни изглед форме

Преименујемо label-е и button1



Слика 4.62. Изглед форме након преименовања компоненти

```
//Kreiramo promenljivu k tipa int
int k ;
//Kreiramo enumeraciju Dani i svakom elementu dodajemo vrednost
enum Dani
{
    Ponedeljak = 1,
    Utorak = 2,
    Sreda = 3,
    Cetvrtak = 4,
    Petak = 5,
    Subota = 6,
    Nedelja = 7
}
private void button1_Click(object sender, EventArgs e)
{
```

```
    //Uzimamo vrednost za promenljivu k koju je korisnik uneo u textBox1
    k = int.Parse(textBox1.Text) ;
    //U lebeli2 ispisujemo resenje ispisivanjem naziva vrednosti naseg nabrojivog tipa
Dani
    label2.Text =k + ". dan u nedelji je : " + ((Dani)k).ToString();
}
```

Pokrenemo program i proverimo (za konkretne vrednosti brojeva) :

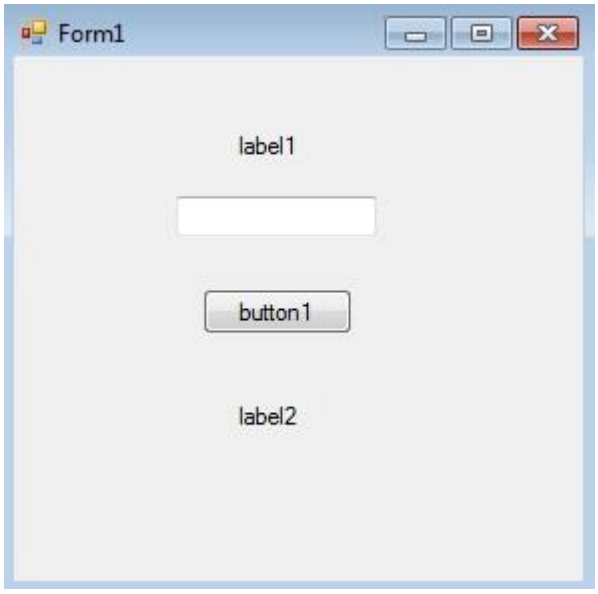
Слика 4.63. Изглед форме након покретања програма

Пример 2. Направити програм који за задати редни број месеца у години исписује његово име у лабели .

Решење:

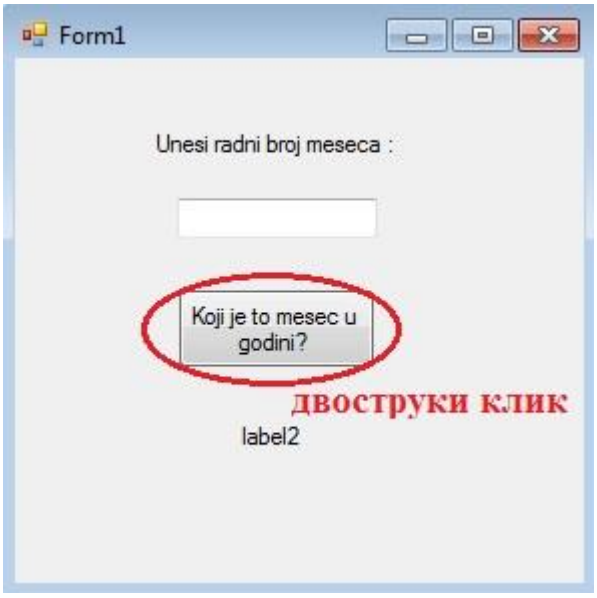
Направићемо, користећи набројиви тип, енумерацију Mesec која ће садржати елементе : Januar, Februar, Mart, April, Maj, Jun, Jul, Avgust, Septembar, Oktobar, Novembar, Decembar који ће редом имати вредности 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 . Пре тога направимо форму:

Правимо форму облика:



Слика 4.64. Почетни изглед форме

Преименујемо label-е и button1



Слика 4.65. Изглед форме након преименовања компоненти

```
//Kreiramo promenljivu k tipa int
int k ;
//Kreiramo enumeraciju Mesec i svakom elementu dodajemo vrednost
enum Mesec
{
    Januar = 1,
    Februar = 2,
    Mart = 3,
    April = 4,
    Maj = 5,
    Jun = 6,
    Jul = 7,
    Avgust = 8,
    Septembar = 9,
    Oktobar = 10,
    Novembar = 11,
    Decembar = 12
}
private void button1_Click(object sender, EventArgs e)
{
    //Uzimamo vrednost za promenljivu k koju je korisnik unese preko textBox1
    k = int.Parse(textBox1.Text) ;
    //U lebeli2 ispisujemo resenje ispisivanjem naziva vrednosti naseg nabrojivog tipa
Mesec
    label2.Text =k + ". mesec u godini je : " + ((Mesec)k).ToString();
}
```

Покренемо програм и проверимо (за конкретне вредности бројева) :

Слика 4.66. Изглед форме након покретања програма

Пример 3. Направити програм који за унету оцену из информатике исписује успех ученика (одличан, врло добар, добар, довољан, недовољан) .

Решење :

```
//Deklarisemo pomocnu promenljivu
int a;
//Kreiramo enumeraciju Ocene, ciji su elementi ocene iz informatike
enum Ocene
{
    nedovoljan = 1,
    dovoljan = 2,
    dobar = 3,
    vrlo_dobar = 4,
    odlican = 5
}
private void button1_Click(object sender, EventArgs e)
{
    //Prihvatamo unetu ocenu iz textBox-a
    a = int.Parse(textBox1.Text);
    //Ispisujemo uspeh u labeli
    label2.Text = "Vas uspeh iz informatike je " + ((Ocene)a).ToString()+ "." ;
}
```

Класе и методе класе

До класа долазимо полазећи од појединачних објеката. Посматрањем објеката уочавамо њихова заједничка својства која зовемо **атрибутима**. Такође, објектима можемо придружити исте „акције“ које у оквиру класе називамо **методима класе**. Њима описујемо функционалности објеката класе. **Класу** дефинишемо навођењем резервисане речи class иза које следи идентификатор класе (име класе). Затим у витичастим заградама { } дефинишемо чланове класе (атрибути и методи).Погледајмо на примеру

class ime_klase

{

opis / definicija clanova klase

}

На пример, класу ученика можемо интуитивно дефинисати на следећи начин:

```
public class Ucenik
{
    //Navodimo atribute koje ce nasa klasa Ucenik da ima
    String ime, prezime ;
    char odeljenje ;
    int razred ;
}
```

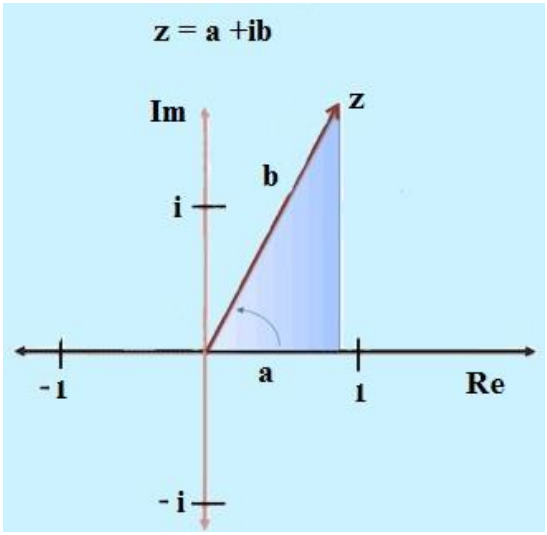
Напомена: Атрибутима описујемо одређену особину објекта (име, презиме, разред, одељење ...). Најчешће, различити објекти исте класе имају различите вредности атрибута. Често кажемо да вредности атрибута дефинишу стање објекта. При опису атрибута морамо навести тип коме тај атрибут припада (целобројни, реални, знаковни...) и име атрибута. При томе, наведени тип је претходно дефинисан (уграђен у систем или дефинисан од стране програмера).

Приступ атрибутима објекта у програмском језику C# реализујемо навођењем имена објекта, знака тачка (‘.’), па имена атрибута. На пример, ако је **X** објекат класе **Ucenik**, атрибуту разред приступамо са **X.razred**.

Пример 1. Направити класу која описује комплексне бројеве .

Решење:

Како правимо класу која описује реалне броје, морамо прво видети које ће атрибуте наша класа имати. Пошто знамо да комлексни бројеви имају свој имагинарни и реални део, наша класа ће имати два атрибута. Први ће бити атрибут који описује реални део комплксног број и он ће бити типа float истог типа ће бити атрибут који описује имагинарни део комплексног броја.



Слика 4.68. Представљање комплексног броја

```
public class KompleksniBroja
{
    //Navodimo atribute koje ce nasa klasa KompleksniBroj da ima
    float re ;
    float im ;
}
```

Методом описујемо понашање објекта у одређеној ситуацији и под одређеним условима , али и одређујемо нове вредности на основу особина које објекат поседује (нпр. метод prosek). На тај начин описујемо функционалност објекта.

Пре позива метода везаног за класу неопходно је помоћу конструктора направити одређени објекат на који ће се тај метод применити. Дакле, прво правимо наш комплексан број, то ће бити објекат типа KompleksniBroj. Уколико у класи нисмо дефинисали конструктор(а то ћемо научити нешто касније), користимо подразумевани конструктор који увек постоји, а потом додељујемо вредности одређеним атрибутима на следећи начин : На пример,x ако је објекат класе KompleksniBroj, атрибуту Re приступамо са x.re . Погледајмо :

```
//Pomocu podrazumevanog konstruktora pravimo kompleksan broj
 KompleksniBroj   a = new KompleksniBroj();
//Realnom delu dodeljujemo vrednost 5
a.re = 5;
//Imaginarnom delu dodeljujemo vrednost 3
a.im = 3;
```

Метод класе је именовани блок наредби који се састоји из заглавља и тела метода. У заглављу наводимо повратни тип (ако метод не производи вредност коју враћа, наводимо резервисану реч void), затим име метода за којим следи у малим заградама списак параметара метода. За сваки параметар наводи се тип коме тај параметар припада као и име параметра. После заглавља у витичастим заградама наводимо тело метода које се састоји из одговарајућих наредби програмског језика C#.Наш метод би имао следећу синтаксу:

povratni tip **ime_metode** (lista parametara koje metoda koristi)

{

telo metode

}

На пример, направићемо методу која исписује име, презиме и разред ученика. Прво морамо одредиди тип наше методе. У овом случају то ће бити String јер желимо да вратимо низ карактера који исписују име, презиме и разред ученика. Затим морамо одредити које параметре наша метода приме . То су три параметра , параметар ime типа string , параметар prezime типа string и параметар razred типа int .

```
//Navodimo tip metode i parametre koje metoda koristi
String ispisiIme( String ime, String prezime, int razred )
{
    //Povratna vrednost nase metode je string i njome ispisujemo zelenu recenicu
    return  " Ucenik " + ime +  " "  + prezime +  "je " + razred +  .razred .
}
```

Сви атрибути конкретног објекта доступни су и видљиви свим методима тог објекта и трају док траје објекат. За разлику од њих, промењљиве декларисане у оквиру неког од метода (локалне промењљиве), настају извршавањем тог метода, видљиве су само у оквиру њега, и гасе се завршетком тог метода. Зато, ако је потребно да користимо вредност у оквиру више метода једне класе (или у више извршавања једног метода), можемо је дефинисати као атрибут класе.

Позив метода објекта у програмском језику C# реализујемо на следећи начин:

imeObjekta.imeMetoda(lista vrednosti parametara)

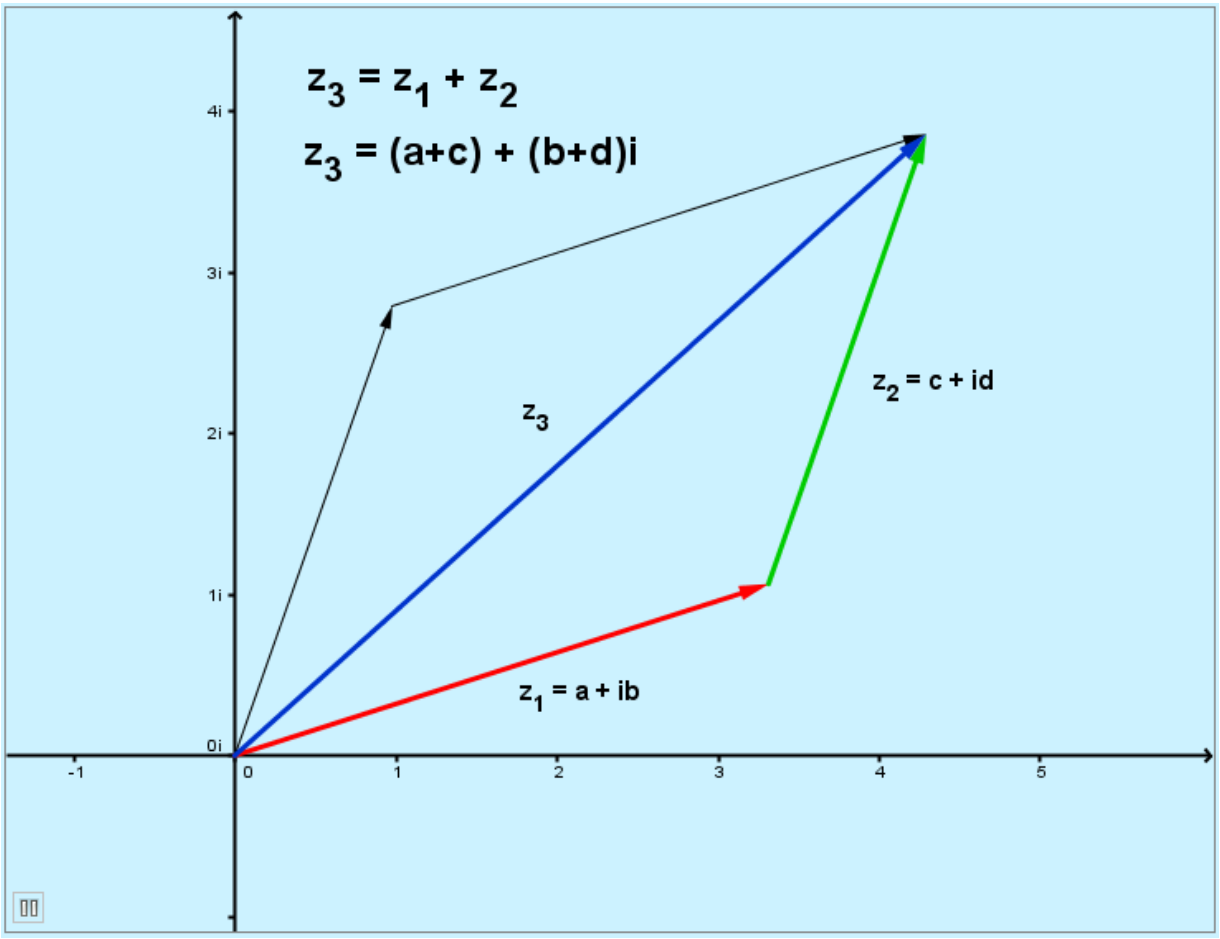
На пример, ако је x објекат класе Ucenik, метод ispisiIme позивамо

x.ispisiIme(ime,prezime,razred)

Више приче о класама следиће након неколико поглавља. Покушајмо сада да направимо једну методу класе KompleksniBroj коју смо претходно направили:

Пример 2. Направити метод који сабира два комплексна броја.

Представљање комплексног броја



Решење:

Прво морамо одредити којег је типа наша метода. Како правимо методу која сабира два комплексна броја, онда знамо да као резултат добијамо комплексан број па је тип наше методе KompleksniBroj. Следеће што треба да одредимо јесу параметри које наша метода прима.Па како сабирамо два комплексна броја, следи да наша метода прима два параметра типа KompleksniBroj .

```
KompleksniBroj Saberi(KompleksniBroj a, KompleksniBroj b)
{
    //Sada treba da odredimo kako sabiramo dva kompleksna broja. Pa sabiramo ih tako
sto
    //realan deo prvog broja saberemo sa realnim delom drugog i imaginarni deo prvog
broja
```

```
    //saberemo sa imaginarnim delom drugog
    //Prvo kreiramo novi kompleksan broj c pomocu podrazumevanog konstruktora
     KompleksniBroj  c = new KompleksniBroj();
    c.re = a.re + b.re ;
    c.im = a.im + b.im ;
    return c;
}
```

У C# имамо и већ посојеће, тј. дефинисане класе, чије методе можемо одмах користити.

Класа System.Math

System.Math класа нуди многе статичке методе које можемо да користитимо при израчунавању тригонометријских, логаритамских и других математичких прорачуна. На пример, **Pow** метод System.Math класе може да се користи да се подигне број на неки степен. Такође, System.Math класа садржи и бројне већ дефинисане константе, које можемо директно користити, као на пример константу **PI.**

На пример, желимо да израчунамо површину круга полупречника p. Како ћемо то урадити уз помоћ System.Math класе и њених већ постојећих метода и константи.

```
double povrsina = Math.Pow(p,2)*Math.PI;
```

Дакле, израчунали смо површину круга користећи методу Pow, чији је први аргумент баш полупречник, тј. оно што се степенује, а други, број на који се степенује полупречник. Константа PI се такође налази дефинисана у класи Math и можемо је слободно користити позивањем Math.PI.

Следећи метод који се често користи јесте метод **Sqrt**. То је метод који се користи када желимо да рачунамо корен неког броја или неког израза.

На пример, желимо да израчунамо хипотенузу правоуглог троугла, ако су нам већ познате његове катете a и b.

```
double hipotenuza = Math.Sqrt(a*a+b*b);
```

У овом задатку могли смо користити и већ од малопре познату методу Pow.

```
double hipotenuza = Math.Sqrt(Math.Pow(a,2)+Math.Pow(b,2));
```

Abs метод враћа апсолутну вредност броја (позитивну вредност броја независно од његовог знак). На пример, следећи код враћа апсолутну вредност броја -3, што је 3.

```
double apsolutna_vrednost = Math.Abs(-3);
```

Метод **Round** заокружује број на најближи цео број. На пример, следећи код се користи да заокружимо број 5,49999 на најближи цео број. Резултат ће бити 5.

```
double ceo_broj = Math.Round(-3);
```

Ако користимо Round да заокружимо број који је тачно на пола између два броја, као што је 5.5, Round увек враћа паран број најблизи броју. Дакле, Math.Round(5.5) ће вратити 6, али Math.Round(8.5) ће вратити 8.

Math.Max и **Math.Min** враћају већи и мањи од два броја, респективно. Они подржавају скоро све типове бројева као параметре.

```
//Maksimum ce biti 5
int maksimum = Math.Max(5,2);
//Minimum ce biti 2.3
float minimum = Math.Min(7.6,2.3);
```

Пример 3. Направити класу круг и методе које рачунају обим и површину круга.

Решење :

```
//Definisemo klasu krug
 public class Krug
{
    double r;
    //Metoda koja racuna povrsinu kruga
    public double Povrsina(double r)
    {
        return r * r * Math.PI;
    }
    //Metoda koja racuna obim kruga
    public double Obim( double r)
    {
        return 2 * r * Math.PI;
    }
}
private void button1_Click( object sender, EventArgs e)
{
    double r = double.Parse(textBox1.Text);
    //Kreiramo primerak klase
    Krug k = new Krug();
    //Pozivamo metodu Obim
    double obim = k.Obim(r);
    label2.Text = "Obim kruga je: " +obim;
}
private void button2 Click(object sender, EventArgs e)
{
    double r = double.Parse(textBox1.Text);
    Krug k1 = new Krug();
    label3.Text = "Povrsina kruga je :" + k1.Povrsina(r);
}
```