Досег промењивих у другим структурама

Утицај досега промењивих постоји и у другим приступима коду, у свим применама блокова кодова и нестованих блокова кода.

Пример:

```
int i;
for(i = 0; i < 10; i++)
{
     string tekst = $"Linija {Convert.ToString(i)}";
     Console.WriteLine($"{tekst}");
}
Console.WriteLine($"Poslednji tekstualni izlaz u petlji: {tekst}");
```

У коду, промењива tekst је локална за петљу, па код се неће компајлирати пошто позив методе WriteLine() се изводи изван петље и намерава да користи промењиву tekst, а то је изван опсега петље.

Модификовани код:

```
int i;
string tekst;
for(i = 0; i < 10; i++)
{
     tekst = $"Linija {Convert.ToString(i)}";
     Console.WriteLine($"{tekst}");
}
Console.WriteLine($"Poslednji tekstualni izlaz u petlji: {tekst}");
```

И овај код се неће компајлирати пошто промењива мора бити декларисана и иницијализована пре употребе, а промењива tekst је иницијализована само унутар петље.

Вредност додељена промењивој tekst је изгубљена када се заврши петља пошто није иницијализована изван петље.

Модификација кода:

```
int i;
string tekst = "";
for(i = 0; i < 10; i++)
{
     tekst = $"Linija {Convert.ToString(i)}";
     Console.WriteLine($"{tekst}");
}
Console.WriteLine($"Poslednji tekstualni izlaz u petlji: {tekst}");
```

Последња вредност додељена промењивој tekst у петљи је доступна и изван петље.

Види се да tekst не садржи празан стринг када изађе из петље већ вредност из петље.

Објашњење оваквог понашања је повезано са алокацијом меморије промењиве tekst.

Сама декларације промењиве не мења значајно ствари, тек када се додели вредност промењивој долази до алокације меморијског простора.

Када се алокација деси током рада петље, вредност је суштински дефинисана као локална вредност и досег је до краја петље.
Иако сама промењива није локализована у петљи, вредост коју она садржи, јесте.
Ипак, доделом вредности изван петље се осигурава да је вредност локална главном делу кода а да је и даље у досегу унутар петље.
То значи да промењива не излази из опсега пре него се заврши главни део кода, па има приступ својој вредности изван петље.

Однос параметара и враћене вредности према глобалним подацима

Ако се упореде следећа два програма:

```
using System;
namespace Proba
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 5;
            Console.WriteLine($"x je {x}");
            PrikaziDuplo(ref x);
            Console.WriteLine($"x je {x}");
        }
        static void PrikaziDuplo(ref int x)
        {
            x *= 2;
            Console.WriteLine($"duplirano x = {x}");
        }
    }
}
```

и

```
using System;
namespace Proba
{
    class Program
    {
        static int x;
        static void Main(string[] args)
        {
            x = 5;
            Console.WriteLine($"x je {x}");
            PrikaziDuplo(ref x);
            Console.WriteLine($"x je {x}");
        }
        static void PrikaziDuplo(ref int x)
        {
```

```
            x *= 2;
            Console.WriteLine($"duplirano x = {x}");
        }
    }
}
```

оба кода дају исти резултат:
x је 5
duplirano x = 10
x је 10

Пример који користи глобалну промењиву x чини да је употреба функције донекле лимитирана пошто се мора стално копирати вредност у друге промењиве ако је потребно стално чувати ту вредност.
Такође, глобална вредност се може модификовати било где у апликацији, што може довести до непредвиђених резултата.
Ако би се дефинисала глобална промењива као X, онда нико не би могао да користи исто име у коду а дошло би и до конфузије на који се промењиву мисли.
Са друге стране, експлицитно специфицирање параметара омогућава бољу контролу над процесима у коду.
Нпр, функција imeFunkcije(x1, out x2), указује да су x1 и x2 важне промењиве а да ће x2 добити нову вредност када се функција заврши.
Савет је да се радије користе параметри него глобалне промењиве.