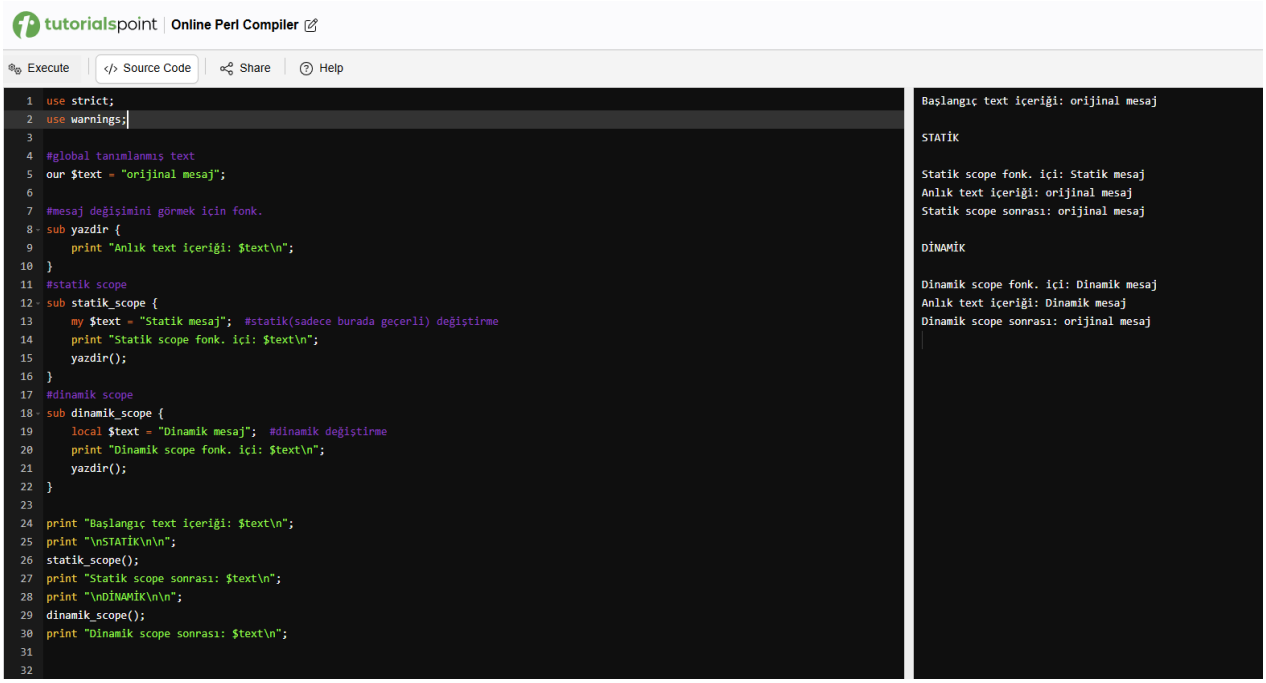
SORU1

```
use strict;
use warnings;

#global tanımlanmış text
our $text = "orijinal mesaj";

#mesaj değişimini görmek için fonk.
sub yazdir {
   print "Anlık text içeriği: $text\n";
}
#statik scope
sub statik_scope {
   my $text = "Statik mesaj";  #statik(sadece burada geçerli) değiştirme
   print "Statik scope fonk. içi: $text\n";
   yazdir();
}
#dinamik scope
sub dinamik_scope {
   local $text = "Dinamik mesaj";  #dinamik değiştirme
   print "Dinamik scope fonk. içi: $text\n";
   yazdir();
}

print "Başlangıç text içeriği: $text\n";
print "\nSTATİK\n\n";
statik_scope();
print "Statik scope sonrası: $text\n";
print "\nDİNAMİK\n\n";
dinamik_scope();
print "Dinamik scope sonrası: $text\n";
```



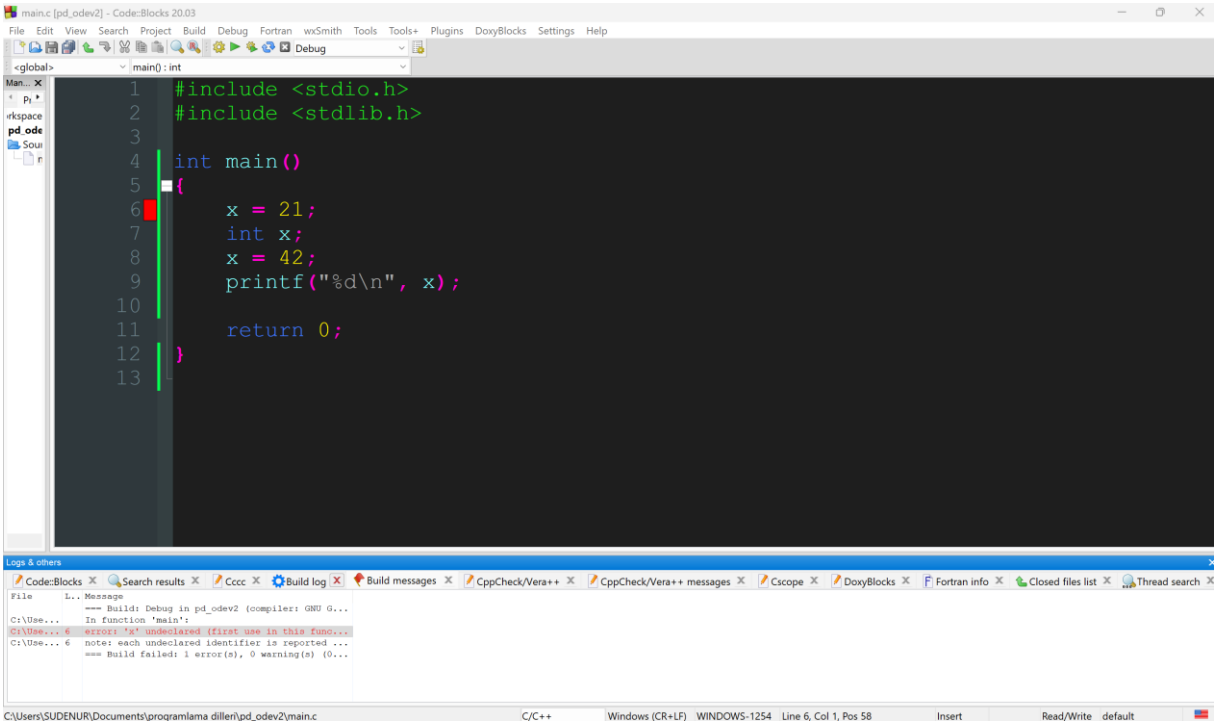
Kitapta anlatılan dinamik kapsamda; bir değişkenin değeri, o anda hangi fonksiyonların birbirini çağırdığına göre belirlenir. Yani değişkenin değeri, programın hangi sırayla çalıştığına göre değişir.

Ama Perl'deki local, bu dinamik kapsamla tamamen aynı değildir. Perl'de local, sadece global bir değişkenin değerini geçici olarak değiştirir. Bu geçici değer, local kullanıldığı andan itibaren o kapsamda ve çağrılan alt fonksiyonlarda geçerli olur. Ancak, yeni bir değişken oluşturulmaz; var olan global değişkenin değeri sadece geçici olarak değiştirilir.

SORU2

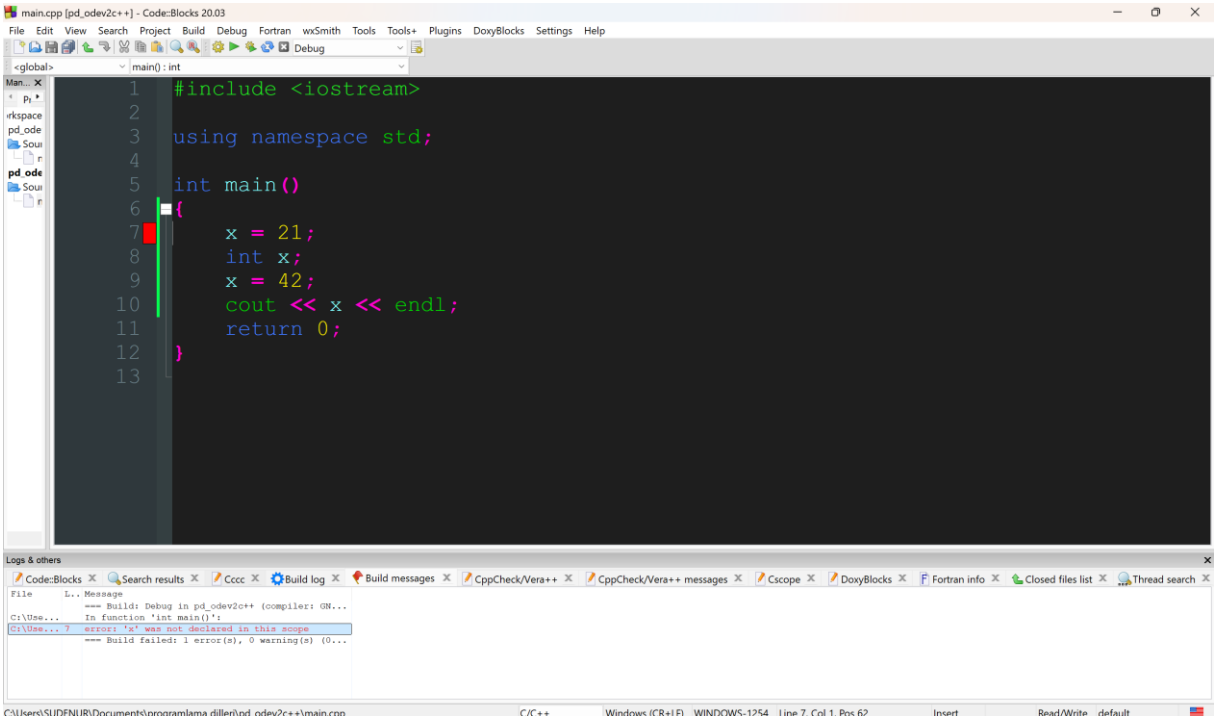
C99

```
#include <stdio.h>

#include <stdlib.h>

int main()

{

   x = 21;

   int x;

   x = 42;

   printf("%d\n", x);


   return 0;

}
```



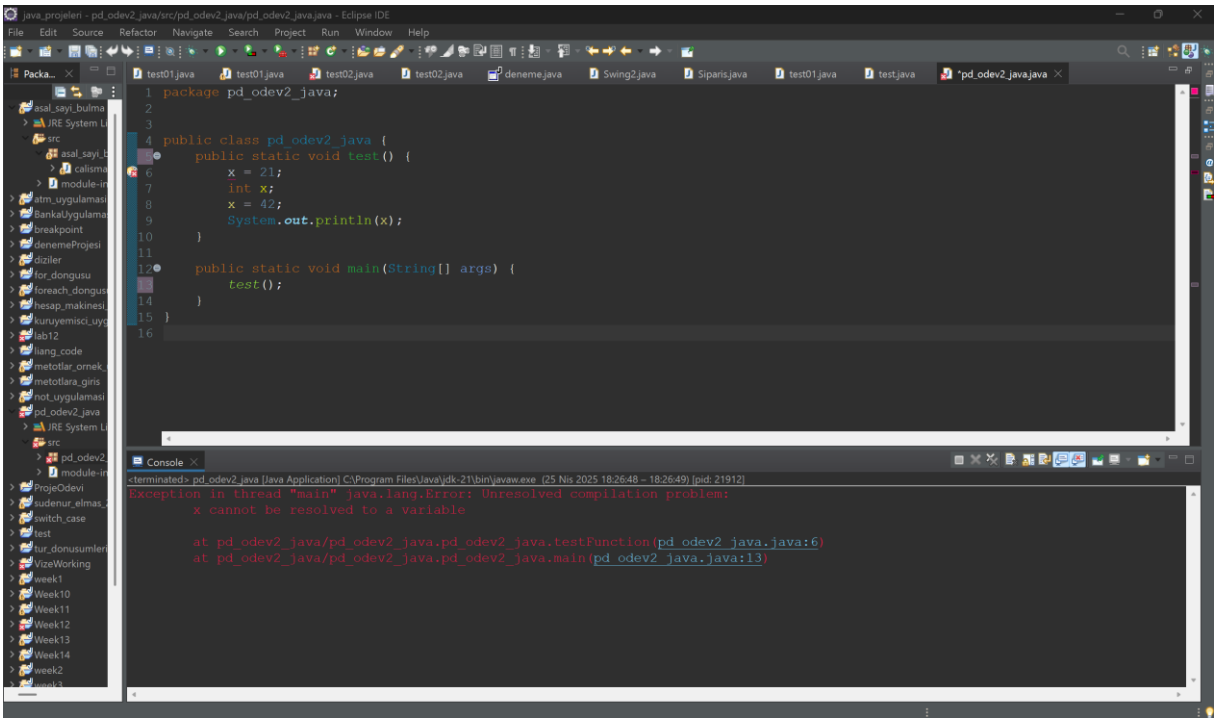
C++

```
#include <iostream>

using namespace std;

int main()

{

   x = 21;

   int x;

   x = 42;

   cout << x << endl;

   return 0;

}
```



JAVA

```
package pd_odev2_java;

public class pd_odev2_java {

   public static void test() {

      x = 21;

      int x;

      x = 42;

      System.out.println(x);

   }

   public static void main(String[] args) {

      test();

   }

}
```

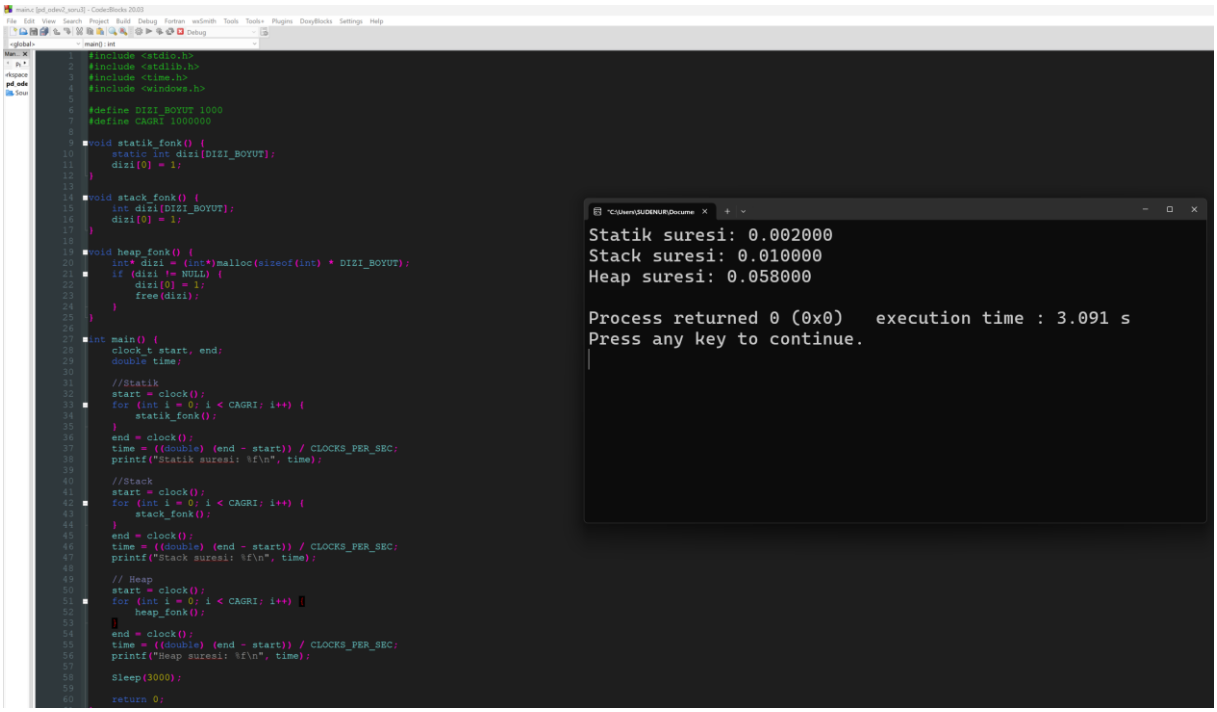


C99, C++ ve Java da bu kodlar çalışmaz çünkü bu dillerde bir değişkeni kullanmadan önce tanımlamak gereklidir. Bu kural, dilin tip güvenliğini korumasını ve hataların erken tespit edilmesini sağlar. Üç dilde de aynı kurala uyulmaması benzer şekilde derleme zamanı hatalarına sebep olur.

SORU3

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>

#define DIZI_BOYUT 1000
#define CAGRI 1000000

void statik_fonk() {
    static int dizi[DIZI_BOYUT];
    dizi[0] = 1;
}

void stack_fonk() {
    int dizi[DIZI_BOYUT];
    dizi[0] = 1;
}

void heap_fonk() {
    int* dizi = (int*)malloc(sizeof(int) * DIZI_BOYUT);
    if (dizi != NULL) {
        dizi[0] = 1;
        free(dizi);
    }
}

int main() {
    clock_t start, end;
    double time;

    //Statik
```

```
    start = clock();
    for (int i = 0; i < CAGRI; i++) {
        statik_fonk();
    }
    end = clock();
    time = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Statik suresi: %f\n", time);

    //Stack
    start = clock();
    for (int i = 0; i < CAGRI; i++) {
        stack_fonk();
    }
    end = clock();
    time = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Stack suresi: %f\n", time);

    // Heap
    start = clock();
    for (int i = 0; i < CAGRI; i++) {
        heap_fonk();
    }
    end = clock();
    time = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Heap suresi: %f\n", time);

    Sleep(3000);

    return 0;
}
```



Statik bellek, programın başında ayrılır ve program sonlanana kadar bellekte kalır. Her fonksiyon çağrısında yeni bellek ayrımı yapılmaz, sadece mevcut bellek kullanılır. Bu yüzden hızlıdır.

Stack, fonksiyon çağrılarıyla birlikte değişebilen bir bellek bölgesidir. Fonksiyon her çağrıldığında dizi stack'e yerleştirilir, fonksiyon sonlandığında stack temizlenir. Statik bellekten yavaş olabilir, çünkü her çağrıda stack'e yer ayırma işlemi yapılır.

Heap, dinamik bellek yönetimi için kullanılır. Her çağrıda heap'ten bellek istenir ve sonrasında serbest bırakılır (free). En yavaş yöntemdir, çünkü her seferinde bellek ayrımı ve serbest bırakma işlemi yapılır.

Süre sıralaması: heap > stack > statik