Programlama Dilleri (315)

Dr. Öğr. Üyesi Ahmet Arif AYDIN

Baglanma (Bidings) , Kapsam (Scopes)

Özet

- İsimler (Names)
- Değişkenler (variables)
  - İsim, adres, tip , deger
  - l-value, r-value

Atama (assignment)

```
void main()
{
int a; // Hafızada 4 byte lık alan kaplar . İlk değer ataması yapılmamıştır
rasgele bir değer tutulur(garbage value)
int b;

a = 2; // 2 değeri a değişkeni için ayrılan alanda saklanır. Atama (assignment)
b=a; // a değeri için ayrılan hafızaki tutulan degeri b değişkeni için ayılan
alana kopyalar. (Now there are 2 memory slots, each 4 bytes in size that has
value 2 stored in them)

int c=2;  // ilk değer atama (Initialization)
}
```

https://www.quora.com/What-is-the-difference-between-initialisation-and-assignment-of-variable-in-c++-programming

Bağlanma (Bindings)

Bir program içerisinde bulunan bir varlık ile niteliğin ilişkilendirilmesine **bağlanma (binding)** denir.

Bağlanmanın gerçekleştiği zaman **bağlanma zamanı** (binding time) olarak tanımlanır ve programlamada çok önemli olan bir kavramdır.

Bağlanma Zamanı (binding time)

Bağlanmanın gerçekleştiği zaman dilimleri aşağıda verilmiştir:

language design time

language implementation time

compile time

load time

link time

run time

Baglanma:Örnek

```
int sayı;
sayı = sayı * 3
```

- **<u>Dil tasarımı zamanında</u> (language design time)**
  - sayı değişkeninin alabileceği muhtemel tipler
  - = ve * sembollerin alabileceği anlamlar bağlanır
- **<u>Derleme tasarım zamanı</u> (compiler design time)**
  - 3 sayısının anlamı bağlanır
  - sayı nın alabileceği muhtemel değerler bağlanır

Baglanma: Örnek

```
int sayı;
sayı = sayı * 3
```

- Derleme zamanı (compile time)
  - sayı değişkeninin tip bağlanması yapılır
  - * işaretinin manası bağlanır
- Çalışma Zamanı (run time)
  - sayı değişkeninin işlem sonucuna bağlanır

Niteliklerin Değişkenlere Bağlanması

Statik Binding: Bir bağlanma program çalıştırılmadan önce tanımlanmışsa ve programın çalıştırılması (execution) boyunca değişmemiş ise statik bağlanma olarak ifade edilir.

Dynamic Binding: Eğer bağlanma program çalıştırıldıktan sonra değişiyorsa dinamik bağlama olarak adlandırılır.

Tip Bağlanması (Type Bindings)

Bir çok programlama dilinde bir değişken tanımlanmadan önce tipinin bağlanması gerekmektedir.

- Statik Tip Bağlanması
- Dinamik Tip Bağlanması
- Tip Çıkarımı (Type Inference)

Statik Tip Bağlanması (explicit-açık)

Açık-Belirtilen Tanımlama (Explicit Decleration)

- Program içerisinde değişkenlerin ve tiplerinin açık bir biçimde yazıldığı tanımlamadır (int değer; (Java , C , C++).)

```
int a, b, c; // degişkenler tanımlanmıs (declaration)
int a = 10, b = 10; // ilk atama (initialization)
byte B = 22; // degişken tanımlanmış ve değer atama
double pi = 3.14159;
char a = 'a';
```

https://www.tutorialspoint.com/java/java_variable_types.htm

Statik Tip Baglanması (implicit-örtülü)

Örtülü Tanımlama (Implicit Decleration)

- Değişken tipleri **varsayılan kurallara** (default conventions) göre belirlenir.
- BASIC
  - Son karakteri $ ile biten tanımlayıcılar karakter tipi ile bağlanırlar
- Fortran
  - Bir tanımlayıcı I,J,K,L,M ,N veya ı,j,k,l,m,n ile başlarsa o tanımlayıcı integer olarak kabul edilir veya real olarak tanımlanır.

Tip Çıkarımı (type inference)

Örtülü olarak (implicit) ve kullanılan değere göre değişkenin tip çıkarımı yapılır.

C#

```
var toplam  = 0 ;
var total   = 0.3 ;
var isim    = "İstanbul"
;
```

İnt
Float
String

Statik değişkenler

Tip Çıkarımı (type inference)

ML (1998)

- Fun daireninalanı (r) = 3.14159 *r*r;
- Fun çarpma (k) = 10*k;
- Fun kare(x)= x * x ;

kare(3.4); HATA

ML de bulunan varsayilan sayısal deger integer oldugundan real deger ıstenen fonksıyonların tanımlanması gerekmektedir.

Fun kare(x): real = x * x ;

Dinamik Tip Bağlanması (Dynamic Type Bindings)

- Değişken tipi bir tanımlama ifadesiyle veya isminin yazılması ile belirlenmez. Atama işlemi sırasında ataması yapılan değere göre bağlanır

- Bu durum programlama esnekliği sağlar.
  - Javascript
  - PHP
  - Python
  - Ruby
  - Perl

Dinamik Tip Bağlanması (Dynamic Type Bindings)

Javascript

- Liste = [3.5, 2.3] liste değişkeni 2 elemandan oluşan bir dizi olarak bağlanır
- Liste = 55 önceki tanımına bakılmadan liste sayısal bir değişkene dönüşür.

- **Ruby**
  - Değişkenlerin tipi tanımlanmaz
  - Bütün veriler ve değişkenler nesne olarak kabul edilir
  - Herhangi bir değişken herhangi bir nesneyi tanımlamada kullanılabilir

Dinamik Tip Bağlanması: Dezavantajlar

- Programın güvenilirliğini azaltır çünkü **derleyicinin hata bulma özelliği kullanılmamış olur**.
- Derleyici (Compiler) yerine Yorumlayıcılar (interpreter) kullanılır.
- Çalışma anında tip bağlanmasının belirlenmesi zaman alan maliyetli bir iştir.
- Herhangi bir değişkenin herhangi bir veri tipine bağlanmasını sağlar
  - Atama işlemlerindeki hatalı atamalar tespit edilemez
  - i=5
  - Y=[1,2,3]
  - i=y; işlemi için hata vermez!

Bellek Bağlanması (Storage Bindings)

Bir degişkenin kullanacağı bellek hücre veya hücrelerinin boş ve kullanıma açık olan bellekten alınması gerekmektedir. Bu işleme **allocation** denmektedir.

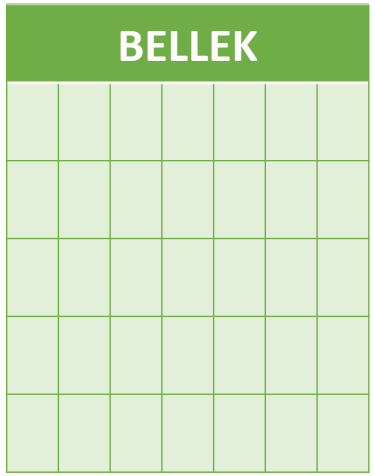
Bellek Bağlanması (Storage Bindings)

Bir degişkenin kullandıgı bellek hucre veya hücrelerinin boş ve kullanıma açık hale getirilmesine de**allocation** denmektedir.

Bellek yeri bırakılması

Yaşam Süresi (Lifetime)

Bir bellek hücre veya hücrelerinin bir değişkene başlanmasından ayrılmasına kadar geçen süre lifetime olarak tanımlanır



Bellek Bağlama: Statik Değişkenler-Avantajlar

Bir programın **çalışmasından önce** belirlenen bellek hücrelerine bağlanan ve programın çalışması süresince aynı bellek hücrelerini kullanan değişkenlere statik değişkenler denir.

Avantajlar

- Global olarak tanımlanan degişkenler statik olması uygun görülmüştür
- Statik değişkenlerin hafıza alanları direkt olarak tanımlıdır.
- Hızlı ve verimli çalışmayı arttırılar. Bellek alanın allocation ve deallocatıon işlemleri zaman alır.

Bellek Bağlama: Statik Değişkenler-Dezavantajlar

- Hafıza alanının seçimindeki esneklik azalır.
- Sadece statik değişken tanımlayan dillerde **recursive işlem yapamazsınız**
- Bellek degişkenler arasında paylaştırılamaz
  - iki alt program farklı zamanlarda çalıştırılmış olmalarına rağmen sadece kendi değişkenlerine tanımlanan alanı kullanır.

Bellek Bağlama: Yığıt Dinamik Değişkenler

(Stack-Dynamic variables)

- Bellek bağlanma işlemi değişken tanımlamalarının yapıldığı kod kısmının çalışma anında tanımlanır (elaboration).
- Değişkenlerin tanımlandığı blok aktif olduğu müddetçe değişkenler yaşar.
- C++ ve Java metotları içerisinde tanımlanan değişkenler bu kategoridedir.

Bellek Bağlama: Yığıt Dinamik Değişkenler

- Avantaj
  - Recursive yapıların oluşturulmasını sağlarlar
  - Az bellek harcanmasını sağlarlar
- Dezavantaj
  - Bellekten yer ayrılması (allocation) ve bellek biriminin serbest bırakılması (deallocation) zaman alır.
  - Alt programlar geçmişi unutur
  - Dolaylı adresleme yapılır

Bellek Bağlama: Açık Yığın Dinamik Değişkenler

(Explicit heap-dynamic variables)

- Programcılar tarafından tanımlanan istenildiğinde bellek üzerinde oluşturulup sonrasında istenildiğinde bellekten silinebilen değişkenlerdir.
- Tip bağlanması derleme zamanında , bellek yer ayrılması ise programın çalışma anında tanımlanır.

Bellek Bağlama: Açık Yığın Dinamik Değişkenler

Açık yığın dinamik değişkenler (Explicit heap-dynamıc variables)

```
int *ornekpointer; //  pointer oluştur
...
 ornekpointer = new int; // yıgın dınamik değişken oluştur
...
delete ornekpointer; // yıgın dınamik değişkeni siler
```

- Dinamik bellek yönetimi sağlar
- Yönetimi zor oldugundan güvenilir değildir

Bellek Bağlama:Örtülü dinamik Değişkenler

Örtülü Dinamik Değişkenler (Implicit heap-dynamıc varıables)

- Sadece değer aldıklarında belleğe bağlanırlar
- Tip ve bellek başlanması her değer aldığında değişirler

Bellek Bağlama: Örtülü dinamik Değişkenler

(Implicit heap-dynamıc varıables)

- Javascript
  - Yükseklik = [74, 84, 86, 90, 71];
  - Önce tanımlı olup olmamasından bağımsız olarak şuan yükseklik değişkeni 5 elemanlı bir dizi olarak bağlanmıştır

Bellek Bağlama: Örtülü dinamik Değişkenler

Avantaj

- Esnek kod yazımını sağlarlar
- Kısa zamanda uygulama geliştirme imkanı

Dezavantaj

- Dinamik özellikler çalışma anında izlendiğinden performans kaybına yol açarlar
- Derleyicinin hata yakalama özelliğini kullanılamaz

Kapsam (Scope)

Bir değişkenin bir program içerisinde

erişilebilir ve görünür olduğu alana kapsam (scope) denir.

Kapsam: static scope

- Statik scope kavramı ALGOL 60 ile ortaya çıkmıştır.
- Değişkenin kapsamı statik olarak tanımlanabilmektedir
- Program çalışmadan önce tanımlanır.

Kapsam: static scope

- Statik kapsamlı diller **alt programların iç içe tanımlanmasına** imkan sağlar
  - Ada, JavaScript, Common LISP, Scheme, Fortran 2003+, F#, and Python

- C tabanlı diller alt programların iç içe tanımlanmasına izin vermez. Bu dillerde statik scope sınıf tanımlamaları ve blok yardımıyla sağlanır.

Kapsam: static scope

```
function anaprogram()
{
    function altprogram1()
    {
        var a = 5;
        altprogram2();
    }

    function altprogram2()
    {
        var b = a;
    }

    var a = 3;
    altprogram1();
}
```

1. altprogram2 içerisinde a değişkeninin tanımı aranır
2. Eğer altprogram2 içerisinde a değişkeninin tanımı bulunmazsa bir üst (ancestor fonksiyona) bakılır.
3. Bir üst seviyedeki anaprogram() içerisinde bulunmazsa hata verir.
4. altprogram1 de tanımlanan a değeri altprogram2 içerisinde tanımlanan a değişkeninden farklıdır.

Kapsam: Blok (Blocks)

- Bir programlama dilinin deyimlerinin (if, for, while) yardımıyla bir araya getirilerek oluşturulan yapıya blok adı verilir.

- Blok içindeki değişkenler blok için lokal değişken olarak tanımlanır.

Kapsam: Blok (Blocks)

```
void program()
{
    int sayı;
    ...
    while (...)
    {
        int sayı;
        sayı++;
    ...
    }
    ...
}
```

- C de yazılan bu kod C++ için dogru iken Java ve C# için dogru değildir.
- While bloğu içerisinde tanımlanan
  - sayı değişkeni while bloğuna ait olan local bir değişkendir

Kapsam:Tanımlama Sırası (Decleration Order)

- Bir çok programla dilinde değişkenlerin tanımlama sırası (decleration order) diğer işlemlerden önce gelir. Önce kullanılacak değişkenler tanımlanır sonra değişkenler üzerinde işlemler yapılır.
- Java ve C++ da bir değişken tanımlandığı yerden itibaren bloğun sonuna kadar geçerlidir.
- C# da tanımlandığı bloğun hepsi için geçerlidir.

```
void döngü()
{
...
    for (int i= 0; i < 10; i++)
    { ... }
    ...
}
```

Global Kapsam (global scope)

```
a = 1
def fonk1():
    print ' fonk1: ', a

def fonk2():
    a = 2
     print ' fonk2: ', a
def fonk3():
    global a
    a = 3
     print ' fonk3: ', a

fonk1()
fonk2()
fonk3()
```

Bir çok programlama dili global değişken tanımlamasına izin vermektedir.

- C, C++, PHP, JavaScript, and Python

Dinamik Kapsam (Dynamic Scope)

```
int b = 5;
int f1(){
    int a = b + 5;
    return a;
}
int f2() {
   int b = 2;
    return f1();
}
int main(){
    f1();
    f2();
    return 0;
}
```

- Alt programların çalışma anında yazıldıkları sıraya göre değil karışık olarak çağrılmasıdır.
- Dinamik kapsamda çağırma stack inde bulunan en yakın değer alınır.

f1()?
f2()?

Dinamik Kapsam (Dynamic Scope)

```
int b = 5;
int f1(){
    int a = b + 5;
    return a;
}
int f2() {
   int b = 2;
    return f1();
}
int main(){
    f1();
    f2();
    return 0;
}
```

f1() 10 değerini gönderir
f2() 7 değerini gönderir

Kapsam (scope) ve yaşam süresi (lifetime)

- Bir değişkenin kapsamı tanımlandığı blok içindedir
- Değişkenin yaşam süresi programın çalışması bitinceye kadardır.

```
void işlem()
{
    int[] intList = new int[100];
    String[] strList = new String[100];
    ...
    for (index = 0; index < 100; index++) {
    ...
    }
    ...
    for (index = 0; index < 100; index++) {
    ...
    }
    ...
}
```

Kapsam (scope) ve yaşam süresi (lifetime)

```
void yazdır()
{
    ...
} /* yazdır son */

void hesapla()
{
    int toplam;
    ...
    yazdır();
    ...
} /* hesapla son*/
```

- Toplam değişkeninin kapsamı hesapla fonksiyonu ile sınırlıdır.
- Yazdır fonksiyonu hesapla içinden çağrılır ve hesapla fonksiyonu tamamlanıncaya kadar toplam değişkeni bellekte tutulur.

Referencing Environments

Belirlenen noktalarda tanımlı olan bütün değişkenlerin listesine ortam referansları (referencing environments) denir.

```
k = 3; #global tanımlama
def ap1():
    x = 5; # local x
    y = 7; # local y
    ... (1)
def ap2():
    global k; # Global k atama yapılabilir
    c = 9; # yeni local
    ... (2)
def ap3():
    nonlocal c: # local olmayan c oluşturulur
    k = 11; # yeni local oluşturur
    ... (3)
```

Referencing Environments

Belirlenen noktalarda tanımlı olan bütün değişkenlerin listesine ortam referansları (referencing environments) denir.

```
k = 3; #global tanımlama
def ap1():
    x = 5; # local x
    y = 7; # local y
    ... (1)
def ap2():
    global k; # Global k atama yapılabilir
    c = 9; # yeni local
    ... (2)
def ap3():
    nonlocal c: # local olmayan c oluşturulur
    k = 11; # yeni local oluşturur
    ... (3)
```

(1) Local x ve y , ve global k

(2) Local c ve global k erişilebilir ve global k ' nin değeri değiştirilebilir

(3) Local olamayan c local k

Sabitler (named constants)

- Program içerisinde bulunan bir değişkenin değerinin bir değere sabitlenmesidir.
- Her seferinde 3.14159 kullanmak yerine Pi sabiti tanımlanabilir.
  - #define M_PI 3.14159265358979323846