Programlama Dilllerinin Prensipleri

Lab Notları – 3

Veri Türleri - 2

Diziler

Homojen verilerin bir araya gelerek oluşturdukları yapı. Bir dizi içerisinde aynı tür veri bulunur. Dizi indeksi sıfırdan başladığı için son indeks "elemansayısı – 1" olarak ifade edilir. C ve Java'da dizi tanımlamaları birbirine yakındır. Aşağıdaki örnek C dilinde çalışırken Java'da ilklenmeden kullanılmaya çalışılıyor hatası verecektir. Burada aslında ileride anlatılacak olan bellek ile alakalı bir durum söz konusudur. Java'da diziler tanımlandıkları yerde değerleri verilmeli ya da heap bellek bölgesinde oluşturulmalıdırlar.

```
#include "stdio.h"
int main(){
   int x[5];
   x[0]=100;
   printf("%d",x[0]);
   return 0;
}
```

```
int x[5];                    // Derlenme zamanı hatası verir.
x[0]=100;
System.out.println(x[0]);
```

Java için doğru tanımlama aşağıdaki iki şekilde olabilir.

```
int[]x={100,200,300};
System.out.println(x[0]);
```

```
int[]x = new int[3];
x[0]=100;
System.out.println(x[0]);
```

İki boyutlu dizilerde tanımlama yukarıdakine benzer koşullarda aynıdır. Burada dikkat edilmesi gereken dizilerin arka planda aslında bir gösterici şeklinde tutulduklarıdır. Dolayısıyla aşağıdaki iki boyutlu dizi tanımlamasında ekrana adres yazacaktır.

```
int x[3][3];
x[0][0]=100;
printf("%d",x[0]);
```

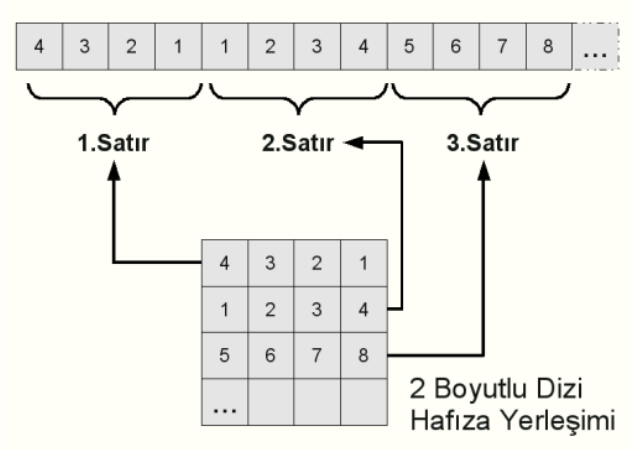
```
int [][]x = new int[3][3];
x[0][0]=100;
System.out.println(x[0]);
```

Dizilerin bellekte tutulma şekilleri bir göstericinin bellekte tutulma şekli ile aynıdır. Yapılan iş sadece ilk elemanın adresini tutmaktır. Derleyici dizinin ilk elemanının adresini tutmakla yetineceği için diziler tanımlandıkları yerde boyutları belirtilmelidir ki derleyici adresi nereye kadar arttırabileceğini bilsin. Siz sayilar[3]'teki elemanı getir dediğinizde derleyici arka tarafta aslında *(sayilar+3) adresindeki değeri getir demektedir.

Programda sayıların tutulduğu adresleri ekrana yazmaya kalkarsanız. Adreslerin ardışık olduğunu göreceksiniz.

2686696
2686700
2686704

Adreslere bakıldığında 4 farkla ilerlediği görülür. 96+00=04 hexadecimal olarak. Bunun nedeni int bellekte 4 byte olarak tutulduğundan kaynaklanmaktadır (C Dili).



Pointers (Göstericiler)

Göstericilerin içinde tuttukları değer adres değerleridir. Göstericilerin geliştirilme amacı dolaylı (indirect) adreslemenin gücünden faydalanmak (daha çok makineye yakın dillerde kullanılır.) ve dinamik bellek yönetimini sağlamak içindir. Bu bellek bölgesine heap bellek bölgesi adı verilir. Java gibi üst düzey dillerde belleğe doğrudan erişime izin verilmez. Fakat C dili ile yapılan pointer işlemleri daha kısıtlanmış hali Java'da yapılır. Aslında Java'da kullanılan nesnelere erişim şekilleri referanslar yardımıyla yapılmaktadır. Ama Java'da C dilindeki gibi * ile erişim yapılmamakta heap bellek bölgesini gösteren zaten bir referans olmaktadır. Aşağıdaki örneği inceleyelim. Her iki kod bloğunda da aslında p ve r birer referanstır.

#include "stdio.h"
int main(){
  int x=100,y=50;
  int *p = &x;
  int *r = &y;
  int *tmp = p;
  p=r;
  r=tmp;
  printf("p:%d\n",*p);
  printf("r:%d\n",*r);
  return 0;
}
	C Dili Örneği

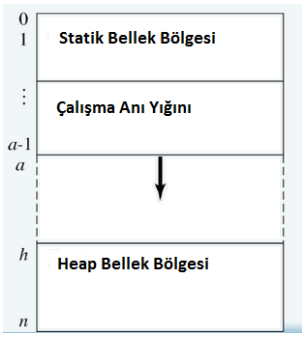
public class Sayi {
  public int deger;
  public Sayi(int dgr){
  deger=dgr;
  }
}
public class IlkProje {
 	
  /**
  * @param args the command line arguments
  */
	Java Örneği

```
    public static void main(String[] args) {
        Sayi p = new Sayi(100);
        Sayi r = new Sayi(50);
        Sayi tmp = p;
        p=r;
        r=tmp;
        System.out.println("p:"+p.deger);
        System.out.println("r:"+r.deger);

    }
}
```

Çalışan herhangi bir programdaki değişkenler, sınıflar, metotlar mutlaka bellekte bir yerde tutulurlar. Bellekte tutuldukları yerler bakımından 3 farklı bölge bulunmaktadır.

- Statik Bellek Bölgesi
- Çalışma Anı Yığını
- Heap Bellek Bölgesi



Statik Bellek Bölgesi

Bu bölgede yer alacak değişkenler hangileri olduğu, daha program başlamadan bellidir. Bu bölgede Global değişkenler, sabitler, static olarak tanımlanmış lokal değişkenler tutulurlar. Statik bellek bölgesinde tutulan değişkenler program çalıştığı sürece var olurlar, program sonlandığında bellekten silinirler.

```
#include "stdio.h"
int kontrol;
const float pi=3.14;
int Arttir(){
    static int sayim = 0;
    sayim++;
    return sayim;
}
int main(){
    printf("%d\n",Arttir());
    printf("%d\n",Arttir());
    printf("%d\n",Arttir());
}
```

Yukarıdaki C kod örneğinde kontrol değişkeni global değişkendir. pi sayısı const ifadesi olduğu için sabittir ve Arttir metodunun içerisindeki sayim değişkeni de başında static olduğu için statik lokal değişkendir. Bu ismi anılan 3 değişkende statik bellek bölgesinde tutulur. Statik bellek bölgesinde

tutulduğu için program sonlanıncaya kadar bellekte tutulurlar. Bundan dolayı Arttir metodu her çağrıldığında sayim değişkeni sıfırdan başlamak yerine kalmış olduğu değerden devam edecektir. Java'da metot içerisinde static kullanımına izin yoktur. Sadece sınıf içerisindeki elemanlar static olarak tanımlanabilir. Bununda anlamı bu eleman sınıftan türetilecek bütün nesneler için ortak ve aynıdır. Aşağıdaki örnek incelendiğinde ekrana her zaman aynı sayıyı yazacaktır. Bunun da nedeni basittir, static olarak tanımlanmış Sayi sınıfının deger elemanı tüm nesneler için ortak olacak ve üstünde yapılmış en son değişikliği koruyacaktır.

```
public class Sayi {
   public static int deger;
   public Sayi(int dgr){
      deger=dgr;
   }
}
public static void main(String[] args) {
      // TODO code application logic here
      Sayi p = new Sayi(100);
      Sayi r = new Sayi(50);
      Sayi tmp = p;
      p=r;
      r=tmp;
      System.out.println("p:"+p.deger);
      System.out.println("r:"+r.deger);
   }
```

Global değişkenler statik bellek bölgesinde tutuldukları için program sonlanıncaya kadar bellekte tutulacaktır ve programın herhangi bir satırından bu değişkenlere erişilebilecektir. İşte bu yüzden global değişkenlerin kullanımı (değişip değişmediklerinin kontrolü zor olduğu için) tavsiye edilmemektedir.

Çalışma Anı Yığını (RTS)

En aktif bellek bölgesidir denilebilir. İsmini de oradan aldığı bu bellek bölgesi bir yığın (stack) şeklindedir ve bu yapıda çalışır. Bu yapıya ilk giren en son çıkar. Bir program çalıştığı sürece genişleyip daralan bitişik bir yapıya sahiptir. Bu bellek bölgesinde fonksiyon ve metot çağrımları ve bu fonksiyon ve metotların barındırdığı lokal değişkenler bulunur. Bir fonksiyon veya metot çağrıldığında bu fonksiyon veya metoda ait parametreler değişkenler ve dönüş değerleri bu bellek bölgesinde tutulur. Çalışma anı yığın bölgesi genişlemiş olur. Fonksiyon veya metot çağrılan yere döndüğünde bu fonksiyon veya metodun çalışma anı yığınında ayırmış olduğu yer geri döndürülür. Dolayısıyla geri döndürülen bu değişkenlere çağrım bittikten sonra erişim olamayacaktır.

```
#include "stdio.h"
int DegerArttir(){
   static int sayac=0; // Statik bellek bölgesinde
   return ++sayac;
}
int topla(int a,int b){
     int sonuc = a+b;  // Çalışma anı yığınında
     return sonuc;
}
int main(){
          printf("%d\n",DegerArttir());
```

```
        printf("%d\n",DegerArttir());
        printf("%d\n",topla(21,10));
        printf("%d\n",topla(5,7));
    return 0;
}
```

Yukarıdaki kod örneğine bakıldığında, iki metot ve bir main metodu yer almaktadır. Main metodunda iki kere DegerArttir metodu çağrılmış ve daha sonra topla metodu çağrılmıştır. DegerArttir metodunun içerisindeki sayaç değişkeni statik lokal değişken olduğu için statik bellek bölgesinde diğer bütün değişkenler, çalışma anı yığınında oluşturulur. Topla metodunun çağrımı bittikten sonra, çalışma anı yığınında oluşturulmuş olan a, b ve sonuc değişkenleri bellekten yok edilirler.

Heap Bellek Bölgesi

Bu bellek bölgesi C ve C++ gibi programlama dillerinde dikkat edilmesi gereken çok önemli bir bölgedir. Çünkü C ve C++ gibi dillerde bu bölgenin kontrolü programcıya bırakılmıştır. Bu da demek oluyor ki eğer bu bölgenin kontrolü iyi sağlanmaz ise bellek taşması ya da yanlış değerlere erişim gibi problemler ile karşı karşıya kalınabilir. Bu bölgeye duyulan ihtiyacın nedeni, dinamik oluşturulan yapıların boyutları değişken olacak ve çalışma anında belirlenecektir. Dolayısıyla bu yapılar heap bellek bölgesinde tutulmalı, bu yapılara ve değişkenlere göstericiler yardımıyla erişilmelidir. Bu bellek bölgesinde tutulan bir değerin adı yoktur yani anonimdir ve ancak değerin bulunduğu adresi gösterecek bir gösterici yardımıyla erişilebilir.

Heap bellek bölgesinde C programlama dilinde bir yer ayırmak için malloc Java'da ise new operatörü kullanılır. C dilinde malloc kullanmadan tanımlanan göstericiler heap bellek bölgesinden yer ayıramazlar. Aşağıdaki C ile yazılmış örneği inceleyelim.

```
#include "stdio.h"
#include "stdlib.h"
int main(){
        int *yas = malloc(sizeof(int)); // Heap bellek bölgesi
        *yas = 30;
        printf("%d\n",*yas);
        int *p;                         // adresi yok
        free(yas);
   return 0;
}
```

malloc metodu kullanıldığında "stdlib.h" kütüphanesi programa eklenmelidir. Heap bellek bölgesinde ayrılan yer C dilinde, işi bittiğinde belleğe geri verilmelidir. Bu bölgenin kontrolü programcıda olduğu için eğer geri döndürülmez ise çöp dediğimiz durum oluşur. Hatırlanırsa bu bölgedeki adreslere çalışma anı yığınındaki göstericiler yardımıyla erişiliyordu. Dolayısıyla çalışma anı yığınındaki gösterici kaybedilmeden önce heap bellek bölgesinde ayrılan yer geri döndürülmelidir. Yoksa o bölge bilgisayar kapanıncaya kadar kullanılamaz duruma gelir. Java'da ise bu durumda yani ayrılan yer belleğe geri verilmediği durumda yine çöp oluşur fakat Java'da çöp toplayıcı mekanizması vardır. Dolayısıyla belli aralıklarla çöp toplayıcılar devreye girerek göstericisi olmayan bellek bölgesini geri döndürürler. C dilinde geri döndürme işi free metodu ile yapılır. Fakat Java'da buna benzer bir yapı yoktur. İllaki Java'da geri döndürülmek isteniyorsa bunun en güzel yolu null'a eşitlemektir. Böylelikle Java Sanal makinesinin çöp toplayıcısı çalıştığında geri döndürülecektir.

```
Sayi s = new Sayi(100);          // Java
s=null;
```

Burada tekrar geri dönüp bir deneme yapılırsa, hatırlarsanız Sayi sınıfının değer elemanı static olarak tanımlanmıştı. Yukarıdaki kod bloğunda s, Sayi nesnesini gösteren bir göstericidir. null'a eşitlenmiş ve geri döndürülmüştür. Fakat bir alt satırına değer yazdırılmaya çalışılırsa hata vermediğini ve 100'ü ekrana yazdığı görülecektir.

```
Sayi s = new Sayi(100);
s=null;
System.out.println(s.deger);   // ekrana 100 yazar
```

void Göstericisi

C dilinde, türü olmayan bir göstericidir. Dolayısıyla yeri geldiğinde bir tamsayıyı gösterebileceği gibi yeri geldiğinde bir ondalık sayıyı da gösterebilir. Sadece göstericinin gösterdiği yer kullanılacağı zaman, derleyicinin o anki hangi tür olduğu bilmesi açısından dönüştürme işlemi uygulanmalıdır. Bunun örneği aşağıdaki kod parçasında görülebilir.

```
#include "stdio.h"
#include "stdlib.h"
int main(){
        int x=100;
        float a=12.5;
        void* obj;
        obj=&x;
        printf("%d\n",*(int*)(obj));
        obj=&a;
        printf("%.2f\n",*(float *)(obj));
   return 0;
}
```

Bu şekilde bir nevi object türüne benzetilebilir. Fakat Java'da buna gerek yoktur. Çünkü Java'da Object türü bulunmaktadır. Aşağıdaki örneği inceleyelim. Şablon sınıflar Java diline daha sonra eklenmiştir.

```
Object x=100;
System.out.println(x);
x="Sakarya";
System.out.println(x);
x=28.12;
System.out.println(x);
```

İşlemler

Java ve C dili dört işlemi desteklerler ve bunlar için özel operatörleri vardır. Bunlar dışında arttırma ve azaltma gibi operatörleri de desteklerler. İşlemlerde dikkat edilmesi gereken işleme giren operandlardan büyük tür, sonucunda türüdür. Örneğin aşağıdaki C kodunda ekrana 3 yazacaktır. Sebebi işleme giren x ve y değişkenleri tam sayıdır ve sonucunda tam sayı olması gerekir. Bundan dolayı 3.5 **yazmamıştır**.

```
int main(){
   int x=7,y=2;
   float z=x/y;
```

```
   printf("%f\n",z);
   return 0;
}
```

Java programlama dilinde de durum aynıdır. Sonucun bir double'a atanması da durumu değiştirmeyecektir. Aşağıdaki Java programı çalıştırıldığında ekrana 3.0 yazacaktır.

```
int x=7,y=2;
double z= x/y;
System.out.println(z);
```

Doğru sonuç elde edilmesi için bir sayının daha büyük türe dönüştürülmesi gerekir. Örneğin aşağıdaki gibi yazılırsa sonuç 3.5 çıkacaktır.

```
int x=7;
double y=2;
double z= x/y;
System.out.println(z);
```

Programlama dillerinde işlem öncelikleri vardır. İlk önceliği parantezler belirler.

Öncelik Sırası:

+, - Sayıların işaretleri

++ , -- Arttırma, azaltma

. * işaretçiler (pointer)

* , / , % Çarpma, bölme, modüler

+ , - Toplama, çıkarma

-= , += , %= , /= , *= Bileşik atama işlemleri

= Atama işlemi

Örneğin aşağıdaki kod ekrana 50 yazar.

```
System.out.println(5+3*15);
```

Örneğin aşağıdaki C kodu ekrana 24 yazar çünkü arttırmanın önceliği çarpmadan daha yüksektir.

```
int main(){
   int x=1,y;
   y=++x*12;
   printf("%d\n",y);
   return 0;
}
```

Ama aynı kod aşağıdaki gibi yazıldığında ekrana 12 yazar sebebi arttırma işleminin daha sonra yapılmasıdır. Arttırmanın önceliği daha yüksek olabilir fakat ++ işareti x değişkeninden daha sonra geldiği için x önce işleme girer daha sonra x'in değeri arttırılır. Fakat sonuç bundan etkilenmeyecektir ve 12 olacaktır.

```
int x=1,y;
y=x++*12;
printf("%d\n",y);
```

X=X+10; ifadesinde normalde soldan sağa ve yukarıdan aşağıya işleme sokulur. Fakat X'in değerinin hesaplanması için sağ tarafa ihtiyaç vardır. Dolayısıyla X'in eski değeri ile 10 toplanıp, X yeni değerini alacaktır. Tabi ki bu ifadenin kısaltmasını da programlama dilleri desteklemektedir.

X += 10; Bütün operatörlerde bu geçerlidir.

Atamaların her zaman sol tarafı değer alan kısım olmalıdır. Yani aşağıdaki tanımlama anlamsız ve geçersizdir. Atama sonrasında sol taraf değer kazanır.

100 = x;

Hazırlayan
Yrd. Doç. Dr. M. Fatih ADAK