Programlama Dillerinin Prensipleri

Lab Notları – 5

1. Döngüler

Bir program yazıldığı vakit bazı durumlarda bir satırın birden çok kez çalıştırılması düşünülebilir. Örneğin ekrana 1'den 100'e kadar sayılar yazılmak isteniyor. Bu durumda hepsini printf kullanarak yazmaya kalkışmak 100 satırı sadece bu işlem için doldurmak anlamına gelir. İşte bu durumlarda döngüler kullanılmalıdır. Farklı yapılarda birçok döngü çeşidi bulunmaktadır. Java ve C dili hemen hemen aynı döngü yapılarını kullanır. Arada ufak farklılıklar bulunmaktadır. Döngüler kontrol bakımından iki türlüdür. Önce test (**pre-test**) ve sonra test (**post-test**) döngüleri, for ve while döngüleri önce test döngüleridir. Do-while ise sonra test döngüsüne girer.

for Döngüsü

```
int main(){  // Kullanışsız ve anlamsız
   printf("1\n");
   printf("2\n");
   printf("3\n");
   printf("4\n");
   …
   …
   return 0;
}
```

```
int main(){ // Kullanışlı ve doğru olanı
   for(int i=1;i<=100;i++) printf("%d\n",i);
   return 0;
}
```

Döngünün kaç kez döneceği belli ise bu durumlarda for döngüsü kullanmak daha uygundur. for döngüsü 3 bölümden oluşur ve 3 bölümünde girilmesi zorunlu değildir. Örneğin aşağıdaki Java kodunda sadece ilk bölüm girilmiştir.

```
public static void main(String[] args) {
      for(int i=0 ; ; ){
         if(i++ == 100) break;
         if(i % 10 == 0) System.out.println(i);
      }
   }
```

for döngüsünün içerdiği 3 bölümün görevi aşağıdaki gibi özetlenebilir.

for (ilklenme yeri bir kez çalışır ; Kontrol yeri her döngüde bakılır ; Güncelleme yeri her döngüde)
Bazı programlama dillerinde for döngüsünün özel bir hali olan foreach döngüsü kullanılır (C#). Bu döngü aralık tabanlı bir döngüdür. Ve bir serideki elemanları sıra sıra dolaşmayı sağlar. C++'ın bazı versiyonlarında foreach olarak yazılmasa da aralık tabanlı döngü yapısı oluşturulabilmektedir. Java'da da aynı şekilde foreach kelimesi desteklenmez fakat aralık tabalı döngü desteklenir. Örneğin aşağıdaki tamsayılar dizisinde kullanıcının girdiği sayı aranmaktadır.

```
public static void main(String[] args) {
      int []SayiDizisi = {15, 22, 41, 65, 35, 54, 100 };
      System.out.print("Aradığınız Sayı:");
      Scanner girdi = new Scanner(System.in);
      int sayi = girdi.nextInt();
      for(int i : SayiDizisi){
         if(i == sayi){
            System.out.println("Sayı Var.");
            break;
         }
      }
   }
```

Aynı yapı C dilinde normal for döngüsü kullanılarak aranmış olsaydı aşağıdaki gibi yazılmış olacaktı.

```
int main(){
          int SayiDizisi[] = {15, 22, 41, 65, 35, 54, 100 };
          int sayi;
          printf("Aranan Sayi:");
          scanf("%d",&sayi);
          for(int index=0;index<7;index++){
                    if(SayiDizisi[index] == sayi){
                              printf("Sayi Var.");
                              break;
                    }
          }
          return 0;
}
```

while Döngüsü

Bu döngü for döngüsüne benzer şekilde kontrol işlemini başta yapar. Dolayısıyla kontrol yanlış dönerse döngü çalışmaz. C dilinde ve Java'da yapısı aynıdır. Örneğin aşağıdaki program kodunda girilen sayıya kadar tam sayıların toplamı ekrana yazdırılıyor.

```
#include "stdio.h"

int main(){
          int sayi,toplam=0;
          printf("Sayi:");
          scanf("%d",&sayi);
          while(sayi != 0) toplam+=sayi--;
          printf("Toplam:%d\n",toplam);
          return 0;
}
```

do-while Döngüsü

Bu döngüyü diğer döngülerden ayıran özellik, koşul ne olursa olsun mutlaka bir kez çalışacaktır. Bunun nedeni kontrol kısmının döngünün sonunda olmasıdır. Aşağıda asal olmayan bir sayı girilene kadar yapılan kontrolde do-while döngüsü kullanılmıştır.

```
public static void main(String[] args) {
      int sayi;
      Scanner girdi = new Scanner(System.in);
      do{
        System.out.print("Sayı:");
        sayi = girdi.nextInt();
      }while(!new String(new char[sayi]).matches(".?|(..+?)\\1+"));
      System.out.println("Girilen sayı asal değildir.");
   }
```

Yukarıdaki kod bloğunda while içindeki kontrol başta biraz karmaşık gelebilir. Bir sayının asal olup olmadığının kontrolü çok farklı şekillerde yapılabilir. Burada regex kullanılarak yapılmıştır. Girilen sayı uzunluğunda boş karakterler dizisi oluşturulup bir String içerisine atılıyor. Daha sonra bu String içerisinde regex kullanılarak bir karşılaştırma yapılıyor. İlk soru işareti sıfır sayısının girilmiş mi kontrol eder. Burada nokta herhangi bir karakter ile eşleşme demektir. + işareti ise bir önceki ifadenin 1 veya daha fazla tekrarlanıp tekrarlanmadığını kontrol eder. 1+ ifadesi ise kendinden önce gelen parantezdeki kısmın 2 veya daha fazla tekrarlanıp tekrarlanmadığına bakar.

(2 veya daha fazla tekrarlanma) x (2 veya daha fazla tekrarlanma) = Asal Olmaz

break ve continue İfadeleri

Döngülerde genel kontrolün dışında bazı durumlar oluşması halinde de döngüden tamamen çıkılmak ya da bir turu es geçmek düşünülebilir. break ifadesi döngüyü koşulsuz bir şekilde sonlandırmayı sağlar. Örneğin aşağıdaki Java kodunda girilen ağırlıklar toplanıyor fakat olurda negatif bir ağırlık girerse döngü sonlandırılıyor.

```
public static void main(String[] args) {
      double ToplamAgirlik=0, agirlik;
      Scanner girdi = new Scanner(System.in);
      do{
        System.out.print("Agirlik:");
        agirlik = girdi.nextDouble();
        if(agirlik < 0) break;
        ToplamAgirlik += agirlik;
      }while(ToplamAgirlik <= 100);
      System.out.println("Girilen Toplam Ağırlık: "+ToplamAgirlik);
}
```

continue ifadesi ise o anki turu es geçmeyi sağlar. Örneğin 3'e bölünenlerin ekrana yazdırıldığı bir programda continue aşağıdaki gibi kullanılabilir. Burada 3'e tam bölünemeyenler es geçilmiştir.

```
int main(){
        for(int i=1;i<=100;i++){
                if(i%3 != 0) continue;
                printf("%d ",i);
        }
        return 0;
}
```

Örneğin bileşik faiz probleminde günlük faiz oranı %0.02 olan bir kredide 20000 TL çekilmek isteniyor fakat 25000 TL'den fazla geri toplam ödeme olması istenmiyor ise kredi kaç günde geri ödenmesi gerekir. Bu problemi döngü kullanarak çözebiliriz.

Bileşik faiz olduğu için her gün faiz hesabına giren miktar değişecektir. Basit formülü

Faiz = (A x n x t) / 3600

```
public static void main(String[] args) {
      double miktar=20000,faiz_Orani=0.02;
      int t;
      for(t=1;miktar<=25000;t++){
         double faiz = (miktar * faiz_Orani * t)/3600;
         miktar += faiz;
      }
      System.out.println("En fazla 25000 geri ödemek için "+t+" günlük alınabilir.");
   }
```

Sonsuz Döngüler

Döngülerdeki mantık doğru olduğu sürece sonlanması hiç gerçekleşmeyecek bir kontrolde döngü sonsuz defa dönecektir. for döngüsü düşünüldüğünde 3 bölümden oluştuğu ve bu bölümlerin girilmesi zorunlu olmadığı için boş bırakılırsa sonsuz döngü olur. Aşağıdaki ilk örnek Java'da diğeri C dilinde verilmiştir.

```
while(true){
        System.out.println(":)");
      }
```

```
for( ; ; ){
          printf(":) ");
}
```

Sonsuz döngü kurup içinde break ifadesi ile bu döngüden çıkılabilir. Bunun sıklıkla kullanım örnekleri vardır.

Önemli: Döngü ifadesinin sonuna ; işareti konulmaz. Konulursa bu bir derlenme hatası değildir. Ama bağlı bulunduğu bloğu çalıştırmaz. Örneğin aşağıdaki for döngüsü ekrana 0'dan 9'a kadar yazması beklenirken ekrana sadece 10 yazacaktır. for döngüsü çalışmış fakat noktalı virgül kullanımı nedeniyle bir alt satır döngüye bağlanmamıştır.

```
int main(){
          int i;
          for(i=0;i<10;i++);
                    printf("%d ",i);

          return 0;
}
```

Hangi Döngü Kullanılmalı

```
while(kontrol_ifadesi){
   // Döngü gövdesi
}
```

```
for( ; kontrol_ifadesi ; ){
   // Döngü gövdesi
}
```

```
for(ilkleme ; kontrol_ifadesi ; güncelleme) {
   // Döngü gövdesi
}
```

```
ilkleme
while( kontrol_ifadesi ){
    // Döngü gövdesi
    güncelleme
}
```

Hangi döngü kullanımında programcıya rahatlık sağlıyorsa o döngü kullanılmalıdır. Kaç kere döneceği bilinen bir döngüde for kullanımı beklenir. Bir değer girildiğinde çıkılacak bir döngüde mesela while veya do-while mantıklıdır.

İç içe Döngüler

İç içe döngüler genelde bir dış döngü ve bir veya birden fazla iç döngü içerirler. Her dış döngü bir iterasyon ilerlediğinde iç döngüler baştan başlayarak tekrarlanırlar. En dış döngüde aynı iterasyon tekrarı olmayacaktır. Bu tarz döngüler birden çok dizi boyutundan sıklıkla rastlanırlar. Aşağıdaki kod bloğunda çarpım tablosu ekrana yazdırılmıştır.

```
public static void main(String[] args) {
      // TODO code application logic here
      System.out.println("                  Çarpım Tablosu");
      System.out.println("-------------------------------------------------------------");
      // sayı başlıklarını yaz
      System.out.print("#  | ");
      for(int i=1;i<=9;i++){
         System.out.print("    " + i);
      }
      System.out.println("\n-------------------------------------------------------------");

      for(int i=1;i<=9;i++){
         System.out.print(i + "  | ");
         for(int j=1; j<=9; j++){
            if(i*j <10) System.out.print("    " + (i*j));
            else System.out.print("   " + (i*j));
         }
         System.out.println();
      }
   }
```

Durum Etiketleri

İç içe döngülerde break ve continue ifadeleri bağlı olduğu döngü için geçerli olup o döngüye etki ederler. Java dilinde durum etiketleri kullanılarak iç içe döngülerde break ve continue ifadelerinde istenilen döngüye etki edilebilir. Bunu C dilinde gerçekleştirmenin tek yolu **goto** ifadesini kullanmaktır.

```
public static void main(String[] args) {
      outer:
      for(int i=1;i<=9;i++){
         for(int j=1;j<=9;j++){
            if(i*j >= 10) break outer;
            System.out.print("  " + i*j);
         }
      }
      System.out.println();
   }
```

```
public static void main(String[] args) {
      outer:
      for(int i=1;i<=9;i++){
         for(int j=1;j<=9;j++){
            if(i*j >= 10) continue outer;
            System.out.print("  " + i*j);
         }
      }
      System.out.println();
   }
```

C dili için

```
#include "stdio.h"
int main(){
   for(int i=1;i<=9;i++){
      for(int j=1;j<=9;j++){
         if(i*j >= 10) goto outer;
         printf("  %d",i*j);
      }
   }
   outer:
   printf("\n");
   return 0;
}
```

Hazırlayan
Yrd. Doç. Dr. M. Fatih ADAK