Programlama Dili Prensipleri

Lab Notları – 8 ve 9

Fonksiyonel Programlama

Fonksiyonel programlamada bir fonksiyon aynı parametreler aldığı sürece aynı sonucu üretecektir. Daha çok yapay zeka için kullanılan bu diller, benzetim (simülasyon) uygulamalarında da görülebilir. Bu diller sadece fonksiyonlar üzerine kurulmuş modelleri gerçekleştirebilirler. Fonksiyonlar birden çok parametre alabilir ama geriye sadece bir değer döndürürler. Fonksiyonlar kendilerini çağırabilecekleri gibi başka fonksiyonları da çağırabilir ve başka fonksiyonların gövdeleri olabilirler. Bu dillerde alt yordamlar fonksiyonlar kullanılarak alt parçalar bölünmelidir.

Lisp Programlama Dili

Lisp dili sembolik veri işleme amacı ile tasarlanmıştır. Karmaşık hesaplamaları daha basit ifadeler cinsinden yazarak kolaylıkla çözümlenmesini sağlar. Lisp programlama dili sadece iki veri yapısı içerir. Bunlar Atom ve Listedir. Atom denilen veri yapısı sembollerden oluşabileceği gibi sayısal değerlerden de oluşabilir. Lisp esas olarak yorumlayıcı kullanan bir dildir. Derleyici kullanan versiyonları da vardır. (Örnek: GNU Common Lisp)

İlk program: Ekrana merhaba yazmak.

```
1  (Defun MerhabaYaz()
2   "Merhaba"
3   )
```

Lisp programlama dilinde fonksiyon tanımlamaları Defun kelimesi ile yapılır. Ve her komut parantez içerisinde yazılmalıdır.

Nasıl Derlenir: Normalde yukarıdaki komutu direk GNU Common Lisp'in siyah ekranına yazıp enter tuşuna bastığınızda ekrana Merhaba yazdığını göreceksiniz. Fazla satırlı fonksiyonlar ile uğraşıldığında siyah ekran yazıyı bir yere kaydetmediği için tekrar tekrar yazmak zaman kaybına neden olacaktır. Bunun yerine uzantısı .lisp olan bir dosyaya kaynak kodu yazıp, dosyayı siyah ekrandan yüklemek daha doğru olacaktır.

Lisp komut satırını açıp (load "C:/Users/MFA/Desktop/Lab8ve9/Lisp/merhaba.lisp") komutunu giriyoruz. Böylelikle MerhabaYaz fonksiyonunu tanıtmış oluyoruz. Komut satırında Finish loading yazması gerekiyor. Denemek için komut satırına (MerhabaYaz) komutunu girdiğimizde bu MerhabaYaz fonksiyonunu çağır demektir. Sonuç ekranda çıkacaktır. Bu kısma kadar yaptığımız işlem yorumlama işlemi idi. İstersek yazdığımız kaynak kodu derleyip çıktı dosyası oluşturabiliriz. Bunun için yazılacak komut:

(compile-file "C:/Users/MFA/Desktop/Lab8ve9/Lisp/merhaba.lisp")

Yukarıdaki komut girildikten sonra eğer kaynak kodda bir hata yoksa merhaba.lisp ile aynı klasörde merhaba.o dosyası oluşacaktır. Bu dosya merhaba.lisp dosyası ile aynı yöntemle siyah ekrana okutulup içindeki fonksiyonlar tanıtılabilir. Bu dosyanın merhaba.lisp dosyasından farkı çıktı dosyası olup içeriğinin görüntülenemiyor oluşudur.

Lisp Programlama Dilinde Matematiksel İşlemler

Matematiksel işlemlerin gerçekleştirilmesi diğer programlama diline benzemektedir. Fakat Lisp programlama dilinde söz dizimsel ifade prefixtir. Yani 10+15 şeklinde diğer programlama dillerinde topladığınız ifadeyi Lisp programlama dilinde (+ 10 15) şeklinde yazmalısınız.

Örnek: Dört işlemi gerçekleştiren Lisp kodu

```
(Defun Hesap(x y)

;Sayıların toplamı
(print "Toplami")
(setq sonuc (Topla x y))
(print sonuc)

;Sayıların farkı
(print "Farki")
(setq sonuc (Cikar x y))
(print sonuc)

;Sayıların Çarpımı
(print "Carpimi")
(setq sonuc (Carp x y))
(print sonuc)

;Sayıların Bölümü
(print "Bolumu")
(Bol x y)
)

(Defun Topla(x y)
(+ x y)
)
(Defun Cikar(x y)
(- x y)
)

(Defun Carp(x y)
(* x y)
)

(Defun Bol(x y)
(/ x y)
)
```

Yukarıdaki kodda normalde print yazılmasa da ekrana sonucu yazar fakat arka arkaya fonksiyonlar çağrıldığı için sadece en sondaki ekrana yazmasın diye her çağrımdan sonra sonuç ekrana yazdırılmıştır.

Örnek: Kullanıcıdan değer alınması

Aşağıdaki kodda görüldüğü gibi bir değişkene değer atamak için kullandığımız setq ifadesinin sol tarafına read komutunu verirsek, değişkene kullanıcıdan aldığı değeri atayacaktır.

```
(Defun Kare()
(print "X:")
(setq x (read))
(print "Y:")
(setq y (read))
(print "Sonuc:")
(+ (* x x) (* y y))
)
```

setq ile birden çok değişkene değer atanabilir. Örneğin (setq x 10 y 32) gibi.

Bir değişkenin değerini bir arttırma veya bir azaltmak için diğer programlama dillerinde ++ ve – operatörleri kullanılır. Lisp programlama dilinde ise arttırmak için (incf x) kullanılır. Bu durumda x 'in değeri 1 artacaktır. Aynı şekilde (decf x) denildiğinde x'in değeri 1 azalacaktır.

(rotatef x y) komutunu verdiğinizde x ve y nin değerlerini değiştirir. Değiştirdikten sonra ekrana nil yazar.

(#b001) komutu yazılan 0 ve 1 lerin ikili bir ifade olduğunu belirtir ve ekrana onun 10 tabanındaki halini yazar örneğin (#b001) ekrana 3 yazacaktır.

Bölme işleminde diğer programlama dillerinde olduğu gibi işleme giren sayılar tam sayı ise sonuç ondalık çıkacaksa bile ekrana tam halini yazacaktır. Örneğin (/ 1 2) komutu ekrana 1/2 yazar ki kullanıcı açısından bir şey ifade etmez. Fakat sayılardan birini ondalık sayıya çevirirsek yani (/ 1 2.0) yaptığımızda sonuçta ondalık olacağı için ekrana 0.5 yazacaktır.

#c ifadesi yazılacak sayının karmaşık sayı olduğunu belirtir. Örneğin (+ #c(1 2) #c(5 2)) yazdığınızda ekrana iki karmaşık sayının toplamını yani #c(6 4) yazacaktır.

Lisp Programlama Dilinde Kontrol Blokları

Diğer programlama dillerine benzer olarak kontrol blokları if ifadesi ile gerçekleştirilir. Fakat yine bağlaç operatörleri (and ve or gibi) prefix şeklinde yerleşecektir.

Örnek: Girilen yaş değerinin 2 ile 18 arasında olup olmadığını kontrol ediyor.

```
(Defun Cocukmu()
( setq yas (read))
(if
(and (> yas 2) (< yas 18)) T
nil
)
)
```

Yukarıdaki kod bloğunda T doğruyu ifade ederken nil yanlışı ifade etmektedir.

Örnek: Verilen üç sayıdan en büyüğünü bulan fonksiyon.

```
(Defun EnBuyuk(x y z)
    (if (and (> x y) (> x z)) x
        (if (and (> y x) (> y z)) y
            (if (and (> z x) (> z y)) z
                "esitlik var"
            )
        )
    )
)
```

Lisp Programlama Dilinde Döngüler

- For Döngüsü: Aşağıya ya da yukarı doğru saydırılabilir.

Örnek: 1 den 10'a kadar ekrana tam sayıları yazmak.

```
(Defun Artan()
    (loop for i from 1 to 10 do
        (print i)
    )
)
```

Örnek: Girilen sayıya kadar tersten tek sayıları yazdırmak

```
(Defun Tek()
    (setq sayi (read))
    (loop for i from sayi downto 1 do
        (if (/= (mod i 2) 0) (print i))
    )
)
```

Yukarıdaki kod bloğunda /= ifadesi eşit değil operatörüdür. Yani diğer programlama dillerinde kullanılan != gibi bir operatördür. mod ifadesi ise pascalda olduğu gibi sayının modunu alır.

Örnek: Faktöriyel Lisp kodu

```
(Defun Fakt(x)
    (setq sonuc 1);
    (loop until (< x 1) do
        (setq sonuc (* sonuc x))
        (setq x (- x 1))
    )
    sonuc
)
```

Örnek: Fibonacci hesabını özyineleme olmadan hesaplama

```
(Defun Fib(x)
    (setq onceki -1)
    (setq sonuc 1)
    (setq toplam 0)
    (setq i 0)
    (loop while (<= i x) do
        (setq toplam (+ sonuc onceki))
        (setq onceki sonuc)
        (setq sonuc toplam)
        (setq i (+ i 1))
    )
    sonuc
)
```

- Fonksiyonlarda opsiyonel parametre tanımlama

```
(Defun fonk(x &optional y z)
 (list x y z)
)
```

Yukarıdaki kod bloğunda &optional ifadesinden sonra gelen tüm parametreler opsiyoneldir. Yukarıdaki fonksiyon çağrıldığında x y ve z elemanlarından oluşan bir liste oluşturur. Eğer y veya z veya her ikisi girilmedi mi girilen kadar liste oluşturacak gerisini nil yapacaktır. Örneğin fonksiyon (fonk 5) şeklinde çağrılabilir bu durumda ekranda (5 nil nil) yazacaktır.

Fakat yukarıdaki fonksiyonu y'yi opsiyonel bırakıp z'ye bir değer vererek çağırmak mümkün olmamaktadır. Bu durumda &key ifadesinden faydalanabiliriz.

```
(Defun fonk(&key x y z)
 (list x y z)
)
```

Yukarıdaki fonksiyonda şöyle bir çağrım yapıldığında (fonk :x 3 :z 5) ilk parametreye 3, ikinci parametre opsiyonel geçilir ve üçüncü parametreye 5 değeri verilir.

Aynı if içerisinde birden çok komut çalıştırılmak isteniyorsa hepsi aynı paranteze alınır ve başlarına progn komutu getirilir.

Örnek:

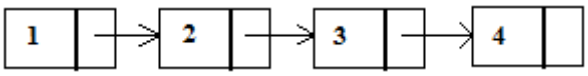
```
(Defun NickAta(&optional isim)
    (if isim
        (setq nick (concatenate 'string isim "123"))
        (progn (setq isim "")
               (setq nick (concatenate 'string isim "123"))
        )
    )
    nick
)
```

Yukarıdaki kod bloğunda NickAta metoduna isim parametre olarak gönderilmez ise boş bir isim oluşturulup "123" string'i ile birleştiriliyor. String birleştirme işleminde concatenate komutu kullanılmaktadır.

Listeler:

Listeler lisp programlama dilinde büyük rol oynar. Örneğin 4 elemanlı A isminde bir liste tanımlamak için,

(defparameter *A* (list 1 2 3 4))



(first *A*) komutu listenin ilk elemanını getirecektir, yani ekrana 1 yazacaktır.

(rest *A*) komutu listenin o andaki konumundan sonrasını getirecektir, yani ekrana (2 3 4) yazacaktır.

İki farklı liste append komutu ile birleştirilebilir.

(append *A* *B*) komutu A listesinin sonuna B listesini ekleyecektir.

Bir listeyi reverse komutu ile tersine çevirebilirsiniz.

(setq *A* (reverse *A*))

Listede bir eleman aramak için find komutu kullanılır. Eğer listede aranan eleman yoksa nil döndürecektir. Var ise elemanı döndürecektir.

(find 4 *A*)

Listedeki bir elemanın konumunu döndürmek için position komutu kullanılır. Yukarıdaki liste örneği düşünüldüğünde aşağıdaki kod ekrana 1 getirecektir. Çünkü 2 elemanı 1. index'te.

(position 2 *A*)

Listedeki bir elemanı silmek için delete komutu kullanılır. Yukarıdaki liste örneği düşünüldüğünde aşağıdaki kod 4'ü silecektir.

(delete 4 *A*)

İki listenin eşit olup olmadığını karşılaştırmak için eq komutu kullanılır. Eşit ise T, değil ise nil dönecektir.

(eq *a* *b*)

Bir liste düğümünde verinin olduğu bölüm lisp programlama dilinde car ile ifade edilir. Diğer düğümü gösteren bölüm ise cdr ile ifade edilir.

Örnek: Listenin belirli bir konumuna eleman ekleme

```
(Defun Ekle(liste index x)
    (push x (cdr (nthcdr index liste)))
 liste
)
```

Yukarıdaki ekle fonksiyonu parametre olarak bir liste, elemanı ekleyeceği konum ve elemanı almaktadır. Daha sonra push komutunu kullanarak elemanı eklemektedir.

Yukarıdaki metot lisp'e tanıtıldıktan sonra bir liste tanımı yapılırsa,

(defparameter *p* (list 1 2 3 4))

daha sonra,

(Ekle *p* 1 10)

1. indeks'e 10 sayısını ekleyecektir. Listenin yeni görünümü (1 2 10 3 4) olacaktır.

Örnek: Listenin belirli bir konumundan eleman çıkarma

```
(Defun Cikar(liste index)
    (delete (first (subseq liste index)) liste)
 liste
)
```

Yukarıdaki kod bloğunda subseq parametre olarak verilen listenin belirtilen index'inden başlayarak sona kadar alt bir liste oluşturur.

Hazırlayan
Arş. Gör. M. Fatih ADAK