BİLGİSAYAR BİLİMİ VE MÜHENDİSLİĞİNE GİRİŞ DERSİ: FONKSİYONEL BİR YAKLAŞIM

Halit OĞUZTÜZÜN **Göktürk ÜÇOLUK**
Orta Doğu Teknik Üniversitesi
Bilgisayar Mühendisliği Bölümü
06531 Ankara

ABSTRACT

Institutions adopt different approaches to teaching introductory courses in Computer Science and Engineering. There is an ongoing debate regarding the best way of introducing students, both majors and non-majors, to the fundamentals of the discipline. Middle East Technical University Department of Computer Engineering has recently redesigned its introductory course for the majors. The new approach aims at both developing computing literacy and improving skills for abstract procedural thinking. For the latter the functional programming paradigm provides a convenient vehicle. The experience gained from the 1996-97 Fall semester offering, although limited, is found encouraging.

1. Giriş

Bilgisayar Bilimi ve Mühendisliği (BBM) olgunlaşma sürecini yaşayan bir disiplindir. Müfredat programları sürekli bir değişim içindedir. Disiplinin gelişme sürecine ilişkin kapsamlı bir raporun özeti [1]'de, standart bir müfredat modeli oluşturma çabalarının yeni bir ürünü [2]'de yer almaktadır. Üniversiteye yeni başlayan öğrencilerin BBM alanında aldıkları ilk kredili dersin kapsamının ne olması ve nasıl bir yaklaşımla verilmesi gerektiği akademik çevrelerde yoğun olarak tartışılmaktadır. Bu bildiriye konu olan giriş dersi [2]'de önerilen modelle uyumludur.

Bu bildiride dile getirilen görüşleri somutlaştırmak amacıyla öğrencilerimizin ev ödevi olarak çözdükleri bir problemden yararlanacağız. Problem nxn'lik boş bir satranç tahtasını en çok k tane at kullanarak kaplamaktır. Tahta üzerine konulan bir atın bulunduğu ve tehdit ettiği kareler kaplanmış sayılmaktadır. Atların birbirini tehdit edip etmemesi önemli değildir. Bu problem için Scheme dilinde yazılmış bir program Şekil 1'de sunulmaktadır. Sunulan kod bir öğrencinin ödevinden bazı biçimsel değişiklikler yapılarak uyarlanmıştır. Kullanılan yöntem *backtracking* esasına dayanmaktadır: Başarısızlıkla sonuçlanan adım geri alınmakta ve başka bir alternatif denenmektedir.

2. Giriş Dersi Neleri Amaçlamalı? Kapsamı Ne Olmalı?

Öğrenciye kazandırmak istediğimiz yetenekler kısaca, verilen bir problem belirtimini anlama, problemi değişik soyutlama düzeylerinde analiz etme, alternatif çözümler üretme, bu çözümleri bilgisayar üzerinde sınama ve bu sınamanın sonuçlarını değerlendirme yetenekleridir. Dahası, öğrenci bu çalışma içinde yaratıcılık yönünü kullanmalı, zihinsel yeteneklerini zorlamalı, yaptığı işten zevk almalı ve eğitim hayatının bundan sonraki aşamaları için merak ve hevesi uyanmalıdır.

Öğrenci karşılaştığı bir problemi çok sayıda küçük adımlar atarak değil az sayıda büyük adımlar atarak çözmeye çalışmalıdır. Örneğin, tahtanın belirli bir kaplanma konfigürasyonundan yeni bir atın konulmasıyla başka bir konfigürasyona geçişi tek bir işlem adımı olarak düşünebilmeli ve bu fikri doğrudan ifade edebilmelidir. O halde öğrenciye vereceğimiz araç onun büyük adımları anlamlı alt-problemlerin çözümleri olarak ayrı ayrı ifade etmesini ve bunları yapay engellere maruz kalmadan akıcı bir şekilde birleştirmesini kolaylaştırmalıdır. Kısaca öğrencinin sentez yapma yeteneğini arttıran bir araç istenmektedir.

Giriş dersinin bir programlama dersine, bunun da bir programlama dilinde kodlama dersine indirgenmesine karşıyız. Öğrencilerimize günlük hayatlarında karşılaşacakları teknolojiyi -ileride alacakları derslere sıkça gönderme yaparak da olsa- tanıtmak durumundayız. Dersin bir kısmı bu nedenle öğrenciyi bizzat kullandığı veya sıkça bahsini duyduğu teknolojiler konusunda bilgilendiren bir "genel kültür" dersi halinde işlenmelidir.

Öğrencilere kazandırmayı amaçladığımız temel kavramlar şunlardır: algoritma, bir algoritmanın zaman verimliliği, özyinelemeyle fonksiyon tanımı, fonksiyonlar üzerinde işlem yapma, yordamsal soyutlama, veri tipi soyutlama, özyinelemeli veri yapıları. Bu kavramları verirken fonksiyonel programlama pratiğinden yararlanılmalı ve öğrencilerin bu kavramları hissetmeleri ve kullanmaları sağlanmalıdır. Yoksa öğrencilerden bu kavramlarla ilgili terminolojik tanımları ezberlemeleri istenmemelidir. Daha sonraki dersler bu kavramları teknik ayrıntılarıyla işleyerek pekiştirecektir.

3. Niçin Fonksiyonel Programlama?

Programlama dili, problemin dünyası ile bilgisayarın dünyasını birbirine bağlayan bir ortam sunar. Dil işlemcisi, bu bağlantının gerektirdiği çeviri ve/veya yorumlama işlevlerini yerine getirir. Dilin problem dünyasına yakın olması, bir yandan çözüm yöntemimizi ifade etmemizi ve bunu diğer insanlara da iletmemizi kolaylaştırırken öte yandan bilgisayar kaynaklarını en verimli şekilde kullanmamızı zorlaştırır. Dilin bilgisayarın dünyasına yakın olması ise tersine bir etki yaratır. Bu tartışma kapsamında, yelpazenin alt ucunda birleştirici dillerini, üst ucunda ise kısıt programlama dillerini düşünebiliriz. Bilgisayar kaynaklarını verimli şekilde kullanmak birçok endüstriyel uygulama için kritik öneme sahiptir ve mühendislik eğitiminin bunu gözardı etmesi düşünülemez. Ancak başlangıç düzeyinde verimlilik kavramının asimtotik anlamıyla sınırlı tutulması yeterlidir. İlk dilin yelpazenin alt ucuna yakın imperatif bir dil olması (Basic, Pascal, Fortran, C vb.) öğrencinin soyut düşünme gücünü kısıtlayıcı bir etki yapmaktadır, çünkü öğrenci problemin çözümünü düşük soyutlama düzeyindeki işlemler cinsinden ifade etmeye zorlanmaktadır. Yelpazenin üst ucuna yakın olan, örneğin Prolog gibi mantıksal dillerde ise programlama, problemin bilgisayarda işletilebilir bir belirtimini ifade etmeye yönelik deklaratif bir tarz almaktadır. Endüstride belirtimleme, prototipleme, doğrulama, otomatik kod üretme gibi uygulamalar açısından deklaratif programlamanın önemi açıktır. Ancak başlangıç düzeyindeki öğrencinin soyut düzeyde fakat yordamsal düşünme yeteneğini geliştirmek daha öncelikli bir hedeftir. Yelpazenin ortalarında gördüğümüz fonksiyonel programlama dilleri hem algoritmik süreçlerin ifadesine hem de deklaratif tarza yatkınlık göstermektedir. Fonsiyonel paradigmayı başlangıç düzeyi için cazip hale getiren diğer özellikler aşağıdaki paragraflarda açıklanmaktadır.

Fonksiyonel programlama fonksiyon tanımlama ve uygulamanın en temel işlemler olduğu bir programlama paradigmasıdır. Bu paradigmada değişkenler matematikteki değişkenlerden, fonksiyonlar matematikteki fonksiyonlardan farksız davranır. Fonksiyonlar yaygın bir deyimle “birinci sınıf vatandaş” statüsündedir; daha açık bir deyişle fonksiyonlar başka fonsiyonlara girdi olabilirler ve bir fonksiyon uygulamasının sonucu olarak üretilebilirler. Bir program, imperatif paradigmada olduğu gibi bellek içeriğini değiştiren adımların sıralanmasından değil, belirli bir referans ortamında yapılan tanımlamalar ve hesaplanan ifadelerden oluşmaktadır. Bir programın çalıştırılması, bir ifadenin mevcut referans ortamında hesaplanmasından ibarettir.

Örneğin, Şekil 1’deki program 7 adet fonksiyon tanımından ibarettir. En üst düzeyde `kapla` adı verilen fonksiyon bulunmaktadır. Diyelim ki 5x5’lik bir tahtanın 6 atla kaplanması probleminin çözümü (`kapla 5 6`) ifadesinin hesaplanması sonucunda elde edilmektedir. Bu ifadenin anlamı `kapla` fonksiyonunun 5 ve 6 argümanlarına uygulanmasıdır. Bu uygulama tanımlanmış diğer fonksiyonların uygulanmasını da içermektedir.

Dilin semantik kuralları, öğrencinin matematiksel sezgisine başvurularak inandırıcı bir tarzda açıklanabilmektedir. Ülkemizde merkezi sistemle BBM ile ilgili bölümlere yerleştirilen öğrencilerin matematik temelleri göreceli olarak iyi durumdadır. Öğrencilerin matematik kavramlarına yatkınlıkları fonksiyonel paradigma yaklaşımının yararlandığı bir kaldıraç noktası olmaktadır. Aynı zamanda öğrencilerin lise matematik bilgileri içinde yer alan fonksiyon tanımlama ve uygulama kavramlarının formelleştirilmesi, küme ve dizi kavramlarına, sembolik mantık işlemlerine ve tümevarım mekanizmasına operasyonel yorum kazandırılması öğrencilerin matematik yönünden olgunlaşmalarına katkıda bulunmaktadır. Şekil 1’deki programda, anılan matematiksel kavramların bir çoğu rol oynamaktadır.

Bir programın zaman verimliliği, girdi büyüklüğünün bir fonksiyonu olarak, çoğu zaman açıklıkla görülebilmektedir. Çünkü verimlilik analizi fonksiyon uygulamalarının sayılmasına indirgenmektedir.

4. Hangi Programlama Dili?

Giriş dersinin amacının herhangi bir programlama dilini öğretmek olmadığını savunuyoruz. Ancak bir problemi çözmek için bir yöntem tasarlayan öğrencinin bunu bilgisayarda çalıştırması için bir araca gereksinimi vardır. Bize göre bu iş için en uygun araç bir fonksiyonel programlama dilidir. Belirli bir programlama dilinin seçimi ise taktik düzeyde bir karardır. Birçok eğitim kurumunda başlangıç düzeyinde kullanılan fonksiyonel diller arasında Scheme, Miranda, ML ve Haskell sayılabilir. Bu dillerin her biri ders kapsamında vurgulanan kavramları doğrudan desteklemektedir. Bizim Scheme dilini seçmemize etken olan nedenleri iki grup halinde belirtebiliriz: dilin kendi özellikleri ve pragmatik ölçütler. Pragmatik ölçütler arasında şunları sayabiliriz:

a) dilin kullanımında, daha önemlisi fonksiyonel paradigmada, deneyimli öğretim üyeleri ve yardımcılarının mevcut olması,

b) dili esas alan üstün kaliteli bir ders kitabının ve başlangıç düzeyindeki diğer kaynakların bulunması,

c) dil işlemcisinin lisans sorunlarına yol açmadan elde edilebilir, kopyalanabilir ve öğrencilerin erişebilecekleri platformlara taşınabilir olması.

Modern bir Lisp türevi olan Scheme eğitim amacıyla bir çok kurumda kullanılmaktadır. Scheme dilinin en olumlu bulduğumuz, aslında diğer fonksiyonel dillerce de paylaşılan özelliğini kısaca şöyle belirtebiliriz: ifadede basitlik ve güçlülüğün bir arada olması.

Basit sözdizim: Dilin sentaksı tasarlanırken minimalist bir yaklaşım izlenmiştir. Bu nedenle

öğrenciler dilin yazım kurallarını zahmetsizce öğrenmektedirler. İlk ders saatinin sonunda basit matematiksel hesaplamaları yapar duruma gelmektedirler. Daha sonra dilin bir özelliği tanıtılırken buna ilişkin sözdizim yapısı öğrencileri şaşırtmamaktadır, çünkü sentaks dili tasarlayanların estetik anlayışına bağlı tercihlerle değil minimalist ilkeye göre belirlenmektedir. Veri tiplerinin sadeliği ve dinamik olarak denetlenmesi, dil işlemcisinin hız (gerek dil işlemcisinin üretiği kodun hızlılığı gerekse dil işlemcisinin kendi işini daha hızlı ve kolay yapması anlamında) kaygısından kaynaklanan ayrımlarla öğrencinin kafasının meşgul edilmemesini sağlamaktadır. Örneğin, öğrencinin tamsayı reel sayı ayrımını öğrenmesi, değişkenlerini buna göre tanımlaması ve bu farkı her an kafasında canlı tutması, sayısal hassasiyet söz konusu olmadıkça, gerekmemektedir. Öte yandan *prefix* notasyonu işleçlerin öncelik ve birleşme özelliklerini hatırda tutma sorununu ortadan kaldırmaktadır. Kısaca, zorunlu bir nedeni olmayan, sadece konvansiyon olarak öğrenilmesi gereken kurallar en aza indirgenmiştir. Böylece öğrenci tüm dikkatini çözüm yönteminin matematiksel özüne yordamsal bir bakış açısıyla yöneltebilmektedir.

Etkileşimli ortam: Dilin gerçekleştiriminde sıkça izlenen yol yorumlamadır. Yorumlayıcı ile etkileşimli olarak çalışma ortamı geribildirim almayı çabuklaştırdığından öğrenme süreci hızlanmaktadır. Etkileşimli ortam öğrenciyi değişik seçenekler üretmeye ve bunları hemen denemeye teşvik etmektedir. Hata mesajları tek bir kaynaktan çıkmaktadır. Derleme, bağlama gibi uğraşılan problemle ilgisi olmayan işlemler yoktur. Hatta ilk alıştırmalarda editör kullanmaya dahi gerek yoktur.

Scheme, Lisp ailesi içinde en çok Common Lisp'e yakındır. Scheme, bir endüstri standardı olan Common Lisp'ten çok daha sade olmakla birlikte eğitim amacıyla sınırlandırılmış bir oyuncak dil değil, aksine güçlü bir algoritmik programlama dilidir. Güncel tutulan bir referans dokümanı [3] ve buna dayalı bir resmi standardı (IEEE 1178-1990) mevcuttur. Bol miktarda örnek kod elde etmek mümkündür. Hızlı ilerleyen öğrenciler kendi başlarına Scheme dilinin özelliklerini keşfe çıkmaktadırlar. Ayrıca, öğretimde kullanılan dilin endüstriyel bağlantısı olan bir dil olması öğrenci motivasyonunu olumlu yönde etkilemektedir.

```
;;; nxn'lik satranc tahtasini
;;; en cok k atla kapla:
(define (kapla n k)
  (yerlestir k (bos-tahta n n)
               (bos-tahta n n)))

; k ati mumkun 'hamleler'le 'yerlestir'erek
;'aciklar'i kapat:
(define (yerlestir k aciklar hamleler)
  (cond
  ((null? aciklar) '())
  ((= k 0) 'basarisiz)
  ((> (length aciklar) (* 9 k)) 'basarisiz)
  ((null? hamleler) 'basarisiz)
  (else
    (let ((alt-cozum (yerlestir (- k 1)
            (kalan aciklar (car hamleler))
            (cdr hamleler))))
      (if (equal? alt-cozum 'basarisiz)
          (yerlestir k aciklar
                     (cdr hamleler))
          (cons (car hamleler)
                alt-cozum))))))

; 'konum' konumuna bir atin konulmasindan
; sonra acik 'kalan' kareler:
(define (kalan aciklar konum)
  (let ((kapali (kapat konum)))
    (filter (lambda (a)
              (not (member a kapali)))
            aciklar)))

; 'konum' karesindeki atin kapattigi
; kareler:
(define (kapat konum)
  (let ((sutun (car konum))
        (satir (cdr konum)))
    (list konum
          (cons (- sutun 1) (- satir 2))
          (cons (- sutun 1) (+ satir 2))
          (cons (+ sutun 1) (- satir 2))
          (cons (+ sutun 1) (+ satir 2))
          (cons (- sutun 2) (- satir 1))
          (cons (- sutun 2) (+ satir 1))
          (cons (+ sutun 2) (- satir 1))
          (cons (+ sutun 2) (+ satir 1)))))

;;; Destekleyici Fonksiyonlar
; l1 ve l2 kümelerinin kartezyen carpimi:
(define (carp l1 l2)
  (if (empty? l1) '()
      (append
        (map (lambda (s)
               (cons (car l1) s)) l2)
        (carp (cdr l1) l2))))

; [p,q] tamsayi araligini uret:
(define (aralik p q)
  (if (> p q) '()
      (cons p (aralik (+ p 1) q))))

; n1xn2 boyutlarinda bos bir tahta yarat:
(define (bos-tahta n1 n2)
  (carp (aralik 1 n1) (aralik 1 n2)))
```

ŞEKİL 1: Örnek bir Scheme "programı".

Bir programlama dilini eğitim amaçlı kullanırken yeni başlayan öğrenciye bazı üst düzey veri tipleri (örneğin doğal dillerle ilgili olarak kelime, cümle gibi veri tipleri) sunulması, öğrencinin dilin anlaşılması nispeten zor ayrıntılarına boğulmadan ilginç problemlerle uğraşmasını sağlamaktadır. Dersin ileri aşamalarında bu veri tiplerinin yine dilin kendi olanaklarıyla nasıl gerçekleştirildiğini göstermek hem öğrencinin teknik bilgisini pekiştirmekte hem de öğrenciye "uygulama alanına uygun özel notasyon yaratma" kavramını hissettirmektedir. (Standard dili bilen okuyucuların incelemesini kolaylaştırmak amacıyla, Şekil 1'deki programda öğrencilere sağlanan kullanımı daha kolay fonksiyonlar yerine bunların standard karşılıklarına yer verilmiştir.)

5. ODTÜ Deneyimi

Orta Doğu Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü 1996-97 öğretim yılının birinci döneminden başlayarak CENG111 kodlu "Bilgisayar Mühendisliği Kavramlarına Giriş" dersinin yeni yaklaşım ve içerikle verilmesini benimsedi. Bu bildirinin yazarları, üç araştırma görevlisinin yardımıyla dersi yürüttüler. 90 birinci sınıf öğrencisi, iki grup halinde haftada üç saat derse girdiler, beş grup halinde haftada iki saat iş istasyonu laboratuvarında çalıştılar. Bu laboratuvarın imkanları, 18 Sun iş istasyonu ve bir dosya sunumcusundan oluşmaktadır. Laboratuvar saatleri dışında öğrenciler bölümün ve üniversitenin diğer olanaklarından, özellikle PC laboratuvarlarından, yararlandılar.

Dönemin ilk dört haftasında bilgisayar mimarisi, işletim sistemi ve diğer sistem yazılımları ana hatlarıyla derslerde anlatıldı. Laboratuvar saatlerinde öğrencilere bilgisayarın temel parçaları tanıtıldı; Unix işletim sistemi, X pencereleme ortamı ve bundan sonra sıkça kullanacakları programlar (*vi* editörü, *tin, elm* vb. haberleşme yazılımları gibi) temel kullanım düzeyinde öğretildi.

Dönemin kalan on haftası boyunca fonksiyonel paradigmaya dayalı olarak temel programlama eğitimi yapıldı. Bu aşamada ders kitabı olarak [4], yardımcı kaynak olarak [5] kullanıldı. Öğrenciler derslerde kavramlarla tanıştılar; laboratuvarda bu kavramları gerektiren problemleri bilgisayar başında bireysel çalışarak asistanların gözetimi altında çözdüler. Problemlerde sayısal ve sembolik hesaplamaya eşit ağırlık verildi.

Öğrencilerin karne notları; üç ev ödevi, laboratuvar ödevleri, iki ara sınav ve bir final sınavının bileşkesi olarak ortaya çıktı. Orta-geçer anlamına gelen CC ve üzerinde not alanlar %74 oranındayken dersi tekrarlamak zorunda kalanların oranı %10'da kaldı.

Öğretim elemanları ve öğrenciler aralarında kesintisiz iletişim için bölümün ortak dosya sisteminde kurulu bir haberleşme grubundan, elektronik posta olanağından ve *world wide web*'den yararlandılar. (Ders ile ilgili *www* sayfasına *http://www.ceng.metu.edu.tr* URL'inden ulaşılabilir.)

Bölümde aynı dönem verilen CENG100 kodlu yönlendirme dersi, aynı öğrenci grubuna bilgi teknolojileri, bilgisayar mühendisliği disiplini ve mesleği, üniversite ve bölüm hakkında tanıtıcı bilgiler vermesinin yanısıra bilgisayar okur-yazarlığı, Internet kullanımı gibi beceriler de kazandırarak CENG111 dersini bütünledi.

Bir dönemlik deneyim kesin sonuçlardan söz etmek için elbette ki kısadır. İleri sınıflardaki öğrencilerin özümlemekte zorlandığı kavramları yeni başlayan öğrencilerin büyük bir doğallıkla benimseyip kullandıklarını gördük. Akademik başarılarının ötesinde, öğrencilerimizde dönem boyunca gördüğümüz şevk ve heyecan doğru yolda olduğumuza dair bizde bir kuşku bırakmamıştır.

TEŞEKKÜR

Dersin işlenişinde önemli katkıları olan araştırma görevlileri İlker Altıntaş, Selçuk Şenkul ve Hasan Ulusoy'a teşekkür ederiz.

REFERANSLAR

1. Hartmanis, J. (editor) *Computing the future*. Summary report of the Committee to Assess the Scope and Direction of Computer Science and Technology of the National Research Council, *Communications of the ACM 35*, 11 (November 1992), 30-40.
2. Walker, H.M. and Schneider, G.M. A revised curriculum for a liberal arts degree in computer science, *Communications of the ACM 39*, 12 (December 1996), 85-95.
3. Clinger, W. and Rees, J. (editors) *Revised^4 Report on the Algorithmic Language Scheme*. November 1991. Available from URL: http://www-swiss.ai.mit.edu/scheme-home.html.
4. Harvey, B. and Wright, M. *Simply Scheme: Introducing Computer Science.* MIT Press, Cambridge, Massachusetts, 1994. ISBN 0-262-08226-8.
5. Friedman, D.P. and Felleisen, M. *The Little Schemer.* MIT Press, Cambridge, Massachusets, 1996 (4th edition). ISBN 0-262-56099-2.