

C - Fonksiyonlar

22/07/2008 15:12 by Coder

FONKSIYONLAR Fonksiyonlar C dilinin temeldiri ve tüm islemin gerçekleştiği yerlerdir. 6.1 Fonksiyon tanımı Fonksiyon blogunun yapısı; dönüş_tipi fonksiyon_adi(parametre listesi)

```
{
  işlem blogu
} dönüş_tipi, fonksiyonun sonucu, fonksiyonun çağırıldığı noktaya gidecek olan degerin tipini belirtir. Fonksiyonlar dizi hariç her tür veri tipi döndürebilirler. Parametre listesi ise, fonksiyon çağırılırken kullanılması gereken degiskenler ve onların veri tipleridir. Fonksiyon parametre kullanmadan da çağırılabilir -bos bir parametre listesi için void veri tipinden faydalanılır.
```

Degisken tanımlamada aynı veri tipine sahip degiskenler, birbirinden virgöl ile ayrılarak tanımlanabiliyordu. Fakat fonksiyonlara ait parametre tanımlamasında bu yöntem kullanılmaz. Parametre tanımlamalarında her parametre, kendi veri tipi ile tanımlanır. Yani; f(int i, int k, float f) doğru iken f(int i, k, float f) yanlıştır. İkinci örnekte k degiskeni kendi veri tipine sahip olmalıdır. 6.2 Fonksiyonun tanım alanı Her fonksiyon, kendi basına bir kod blogudur. Bu yüzden fonksiyonlar kendi tanım alanlarını oluştururlar. Yani fonksiyonun kodu kendine özeldir ve program içinde yer alan herhangi bir ifade tarafından bu kod parçasına erişilemez. Örneğin bir fonksiyon içinde goto kullanılarak başka bir fonksiyonun kodunun herhangi bir yerine atlama yapılamaz. Fonksiyonu oluşturan kod parçası programdan saklanır ve bu kısım global degisken kullanmadıkça ne programın bir kısmı tarafından etkilenir ne de programın başka bir kısmını etkileyebilir. Bir fonksiyonun içinde tanımlanmış degiskenlere local (yerel) degiskenler adı verilir. Local degiskenler, tanımlandığı fonksiyon çağırıldığında devreye girer ve fonksiyonun çalışması sona erince kullanımdan kaldırılır. Bu yüzden local bir degisken fonksiyonlar arasında deger tasılamak için kullanılamaz. Değer tasılamak için parametreler ve dönüş degerleri kullanılır. 6.3 Fonksiyon argümanları Eger fonksiyon argüman alacaksa bunlar parametre listesinde tanımlanmalıdır.

Örneğin; int is_in(char *s, char c) { while(*s) if(*s==c) return 1; else s++; return 0; } is_in fonksiyonunun s ve c olmak üzere 2 parametresi vardır. Bu fonksiyon verilen c karakterinin, s stringinde olup olmadığını kontrol eder.

Parametreler fonksiyonlar arasında deger tasıma görevini yerine getiriyor olsalar da, fonksiyon içinde local degisken gibi davranırlar. Parametreler'e deger atanabilir veya bir ifade içinde kullanılabilirler. C'de fonksiyona parametre gönderme işlemi 2 ayrı şekilde yapılır: Değer ile ve referans ile çağırma. 6.3.1 Değer ile çağırma Değer ile çağırmada, fonksiyondan kullanılan degiskenlere argümanın degeri atanır. Bu durumda, fonksiyon içinde argüman üzerinde yapılan işlemlerin argümanın çağırılan yerdeki degerini üzerinde bir etkisi yoktur. Örneğin; #include \sqrt{x} int sqr(int x) { x = x*x; return(x); } void main(void) { int t=10; printf("%d %d", sqr(t), t); } kod parçası gözönüne alındığında, sqr fonksiyonuna parametre olarak verilen t degiskeninin degeri, fonksiyon içinde x'e atanmıştır. x=x*x işlemi yapıldığında degisken fonksiyon içinde local degisken olarak kullanılan x'in degeridir. Yani ana programdaki printf satırının sonucu "100 10" dur.

6.3.2 Referans ile çağırma Referans ile çağırma, deger yerine, gönderilecek parametrenin adresini gönderme yöntemidir. Fonksiyona gönderilen argüman bir deger değil bir adrestir - yani fonksiyon içinde bu adres kullanılarak, gönderilen argümanların degerleri degistirilebilir. Pointer'ların fonksiyona argüman olarak gönderilmesi, diğer tipler ile aynıdır. Örnek program, adresleri verilen 2 sayının degerlerini degistirir; #include \sqrt{x} void swap(int *x, int *y) { int temp; temp = *x; /* save the value at address x */ *x = *y; /* put y into x */ *y = temp; /* put x into y */ } void main(void) { int i, j; i = 10; j = 20; printf("i and j before swapping: %d %d", i, j); swap(&i, &j); /* pass the addresses of i and j */ printf("i and j after swapping: %d %d", i, j); } Bu örnekte swap() fonksiyonu, degiskenlerin degerlerini degistirebilir çünkü degiskenler fonksiyona deger ile değil referans ile, yani pointer ile gönderilmiştir. Adres göndermek için & operatörünün kullanıldığına dikkat edilmelidir. 6.3.3 Fonksiyona dizi gönderme

Fonksiyona parametre olarak dizi gönderilmek istendiğinde yapılan şey fonksiyona dizinin adresini göndermektir. Bunun sebebi, dizi adlarının, dizinin hafızadaki başlangıça adresini vermesidir. Fonksiyon çağırılırken bu bir deger ile çağırılma gibi görünse de, aslında referans ile çağırma ve fonksiyon içinde, gönderilen parametre üzerinde degisiklik yapılabilir. Örneğin; #include \sqrt{x} #include \sqrt{x} void print_upper(char *string) { register int t; for(t=0; string[t]; ++t) { string[t] = toupper(string[t]); putchar(string[t]); } } void main(void) { char s[80]; printf("Enter a string: "); gets(s); print_upper(s); printf("ns is now uppercase: %s", s); return 0; } print_upper() fonksiyonu, gönderilen stringin

karakterleriin teker teker denk gelen büyük harfe çevirir ve ekrana yazar. Burada parametre olarak gönderilen string'in üzerinde degisiklik yapılmaktadır. Eger stringin içeriğinin degismesi istenmiyorsa, ekrana yazdırma islemi void print_upper(char *string) { register int t; for(t=0; string[t]; ++t) putchar(toupper(string[t])); } fonksiyonu ile yapılabilir. 6.4 argc ve argv - main() fonksiyonunun parametreleri Genelde yazılan programlara deger göndermek gerekebilir. Bu işlem komut satirindan, programin adi ile deger göndererek yapılır. Program adi ile birlikte gönderilen degerler, main() ana fonksiyonuna gönderilen parametreler olarak algılanir. Komut satiri argümanlarına , UNIX'ın C kodu derelyicisinin kullanimi örnek verilebilir; cc c_program_adi C'de, programa argüman göndermek için iki özel degisken tanımlanmıştır: argc ve argv. Bu iki degisken komut satirindan argüman almak için kullanilir. argc degiskeni, programa gönderilen argüman sayisini tutar. Bu deger her zaman 1'den büyüktür çünkü program adinin kendisi de bir argüman olarak kabul edilir. argv ise, karakter pointer'ları tutan bir diziyi gösteren pointer'dir. Bu dizideki her pointer, komut satirindan girilen argümanlardan birini göstermektedir. Komut satirindan girilen her argüman string olarak degerlendirilir. Eger bir sayi kullanılacaksa, bu önce uygun string formatına çevrilmelidir. Örnek program 'Merhaba' dan sonra verilen ismi ekrana yazdırir; #include <stdio.h> void main(int argc, char *argv[]) { if(argc!=2) { printf("You forgot to type your name.n"); exit(1); } printf("Hello %s", argv[1]); } Koda dikkat edilecek olunursa, ana fonksiyonun diger örneklerdeki gibi void main(void) olarak degil, void main(int argc, char *argv[]) olarak tanımlandığı görülür. Bu standart fonksiyon tanımlama sekline uygundur ve burada main fonksiyonu int tipinde argc ve char* tipinde bir argv dizisini argüman olarak alır. Programa girildiginde yapılan kontrol, argüman sayisinin dogru olup olmadiginin kontrolüdür. Bu programa test adi verilip derlenirse, programi çağirma sekli, test isim seklinde olacaktır. Komut satirinda her argüman bosluk (space) karakteri ile ayrilir. Örneğin; test ad1 ad2 2 argüman aliyorken, bu satir; test ad1,ad2 olarak yazılırsa bir argüman alır. argv argümaninin, char *argv[] olarak tanımlanması, argüman olarak verilen dizinin eleman sayisinin belli olmadığını ifade eder. Bu dizinin ilk elemanı, yani argv[0], programin adini verir. Yani programa verilen gerçek argümanlar, dizinin 2. gözünden itibaren baslar. Bir diger örnek de, verilen bir sayidan geriye dogru sayan bir program; #include <stdio.h> #include <math.h> int main(int argc, char *argv[]) { int disp, count; if(argc Bu programa sinir sayinin yaninda, geri sayimi ekranda gösterip göstermeyecegini belirten bir de komut verilir. Eger bu komut 'display' ise, geri sayim ekranda gösterilir. argc ve argv parametreleri, genelde bir programin ilk çalıştırılma anında, bazı opsiyonlar veya kullanılacak dosyaların adlarını vermek için kullanilir. argc ve argv isimlerini kullanma zorunluluğu yoktur ama bu kullanım geneldir. Bunlar yerine programci kendi istediği herhangi bir ismi kullanabilir. 6.5 return ifadesi return ifadesinin 2 ayrı görevi vardır: Biri, daha önce akis kontrolünde de bahsedilen bir fonksiyondan çıkma, diğeri de bulunduğu fonksiyondan bir deger döndürme. Aslında, her iki görevi de aynıdır çünkü return ifadesi bir deger döndürmeden fonksiyondan çıkma işlemini gerçekleştiremez. 6.5.1 Fonksiyondan dönüş Kontrolün, fonksiyondan çağırıldığı yere dönmesi 2 şekilde olur: birincisi fonksiyonun çalışmasının sona ermesi, ikincisi ise return ifadesi ile. Aşağıdaki örnekte, kontrol fonksiyonun isini bittigi için ana programa döner; void pr_reverse(char *s) { register int t; for(t=strlen(s)-1; t>=0; t--) putchar(s[t]); } void main(void) { pr_reverse("I like C"); } Burada fonksiyon, parametre olarak verilen stringi tersten yazdırdıktan sonra, main fonksiyonuna döner. Genelle fonksiyonlar bu şekilde kodlanmazlar. Hatta bir fonksiyon farklı durumlara göre farklı degerler döndürebilecek şekilde kodlanır. Örneğin; int find_substr(char *s1, char *s2) { register int t; char *p, *p2; for(t=0; s1[t]; t++) { p = &s1[t]; p2 = s2; while(*p2 && *p2==*p) { p++; p2++; } if(!*p2) return t; /* 1st return */ } return -1; /* 2nd return */ } void main(void) { if(find_substr("C is fun", "is") != -1) printf("Substring is found."); } Bu programdaki find_substr, verilen bir string içinde, isetlenen ikinci bir string geçiyorsa bunun başlangıç adresini, eger geçmiyor ise -1 döndürür. Görüldüğü gibi 2 ayrı return ifadesi kullanılmıştır ve bu ifadelerden biri her defasındafarklı deger döndürebilecek şekilde kodlanmıştır. 6.5.2 Deger döndürme void olarak tanımlanmamış her fonksiyon return ifadesi ile bir deger döndürmek zorundadır. Eger fonksiyon void tanımlanmadığı halde return ile bir deger döndürmüyorsa, garbage value döndürüyor denir. Fonksiyonlar deger döndürebildikleri için, dönüş degerinin tipine göre, program içindeki ifadelerde kullanılabilirler. Bu yüzden aşağıdaki yazımlar doğrudur; x = power(y); if(max(x,y) > 100) printf("greater"); for(ch=getchar(); isdigit(ch);) ... ; Fonksiyonlar çağırıldıkları noktalarda bir deger ifade ettikleri için, atama işlemlerinde sol tarafta bulunmazlar. Örneğin; swap(x,y) = 100; /* incorrect statement */ yanlış bir yazımdır. Program yazarken genel anlamda 3 çeşit fonksiyon kullanilir: Birinci tip fonksiyonlar hesap işlemleri yaparlar. Fonksiyon parametre olarak verilen degerler üzerinde bir işlem yapar ve sonucu ana programa döndürür. sqrt() veya sin() gibi aritmetik fonksiyonlar bunlara örnektir. İkinci tip fonksiyonlar kendilerine verilen bilgiyi degerlendirir ve bu bilgi sonucunda yapılan işlemin başarıyla mi başarısızlık mi

sonuçlandığını döndürür. Dosya üzerinde işlem yapan fclose() fonksiyonu bu tip fonksiyonlara bir örnektir. Parametre olarak verilen dosya düzgün kapatılabilirse 0, aksi halde EOF döndürür. Son fonksiyon tipi, herhangi bir dönüş değeri olmayan ve tamamen prosedürel yapıda çalışan fonksiyonları içerir. Bu tip fonksiyonlar çağırıldıktan sonra, tanımlı olan işlem kümesini gerçekleştirir ve ana programa dönerler. Değer döndüren fonksiyonlarda, Pascal'da olduğu gibi, dönüş değerini herhangi bir değişkene atama zorunluluğu yoktur. Fonksiyon, değişken kullanmadan da, bir işlemin sonucunu veya direkt olarak bir değeri kullanarak, programa değer döndürebilir. Örneğin; `int mul(int a, int b) { return a*b; }` `int main(void) { int x, y, z; x = 10; y = 20; z = mul(x, y); /* 1 */ printf("%d", mul(x,y)); /* 2 */ mul(x, y); /* 3 */ return 0; }` Bu programda mul() fonksiyonu, verilen 2 parametrenin çarpım sonucunu, yedek bir değişken kullanmadan döndürmektedir.

6.5.3 Pointer döndürme Pointer döndüren fonksiyonlar, diğer veri tiplerinden değer döndüren fonksiyonlarla aynı görevi yaparlar. Bu tip fonksiyonları tanımlarken dikkat edilmesi gereken şey dönerken pointer'ın ne tip bir değişkeni gösterdiği bilgisidir. Çünkü dönüş değeri aritmetik işlemde kullanıldığında tipi sorun çıkartabilir. Bazı durumlarda, hangi tip pointer döneri bilinmediğinden, genel pointer tipi olan void * kullanılabilir.

```
#include <string.h>
char *match(char c, char *s) { while(c!=*s && *s) s++; return(s); }
void main(void) { char s[80], *p, ch;
gets(s); ch = getchar(); p = match(ch, s); if(*p) /* there is a match */ printf("%s ", p); else printf("No match found."); }
```

6.5.4 void tipinde fonksiyonlar void veri tipinin kullanım alanlarından biri, herhangi bir değer döndürmeyen fonksiyonların dönüş tipi olarak kullanılmasıdır. Bu tip fonksiyonlar prosedürel fonksiyonlardır. Örneğin; `#include <string.h>
void print_vertical(char *str) { while(*str) printf("%c\n", *str++); }` `void main(int argc, char *argv[]) { if(argc > 1) print_vertical(argv[1]); }` programındaki print_vertical() fonksiyonu, verilen bir stringin her karakterini bir satıra yazdır ve ana programa döner. Herhangi bir dönüş değeri yoktur.

6.6 main() fonksiyonu C, fonksiyon temelli bir programlama dili olduğundan, bir programın ana fonksiyonu olarak main() tanımlanır. main() in dönüş tipi, işletim sistemine göre değişir. Çoğu C compiler'ın main'ın default dönüş tipini void olarak kabul ederken, UNIX'te main dönüş tipi int'tir.

6.7 Fonksiyon prototipleri C'de fonksiyon kodu yazılmadan önce fonksiyona ait prototip yazılmalı ve fonksiyon tanımlanmalıdır. Bu teknik olarak gerekli olmasa da kullanımı genelde desteklenmektedir. Prototip kullanımı fonksiyona tip kontrolünde kolaylık sağlamaktadır. Prototip kullanımı ayrıca derleyiciye argüman sayısını ve tipini farklı noktalarda kontrol etme imkanı da sağlar. Fonksiyon prototipinin genel yazımı; `d tipi fonk_adi (argüman listesi);` Argüman listesinde argümanların isimlerini kullanma zorunluluğu yoktur. Fakat veri tiplerini verem zorunluluğu vardır. Aşağıdaki kod örneği, prototip kullanmanın gerekliliğini gösterir. `sqr_it()` fonksiyonu int * yerine int ile çağırılmaya çalışıldığından, derleyici programın çalışmasından önce derleme anında hata verecektir; `void sqr_it(int *i); /* prototype */
int main(void) { int x; x = 10; sqr_it(x); /* type mismatch */ return 0; }` `void sqr_it(int *i) { *i = *i * *i; }`

Fonksiyona ait prototip, fonksiyonun ilk kullanıldığı yerden önce tanımlanmalıdır. Önceki örneklerdeki gibi fonksiyon kodu, fonksiyon kullanılmadan önce yazılırsa, bu da o fonksiyona ait prototip görevini görür. Örneğin yazacağımız fonksiyon, main() fonksiyon blogundan çağırılıyorsa, fonksiyonumuza ait prototipin main()'de önce yazılması gerekir; `void f(int a, int b) { printf("%d ", a % b); }` `int main(void) { f(10,3); return 0; }`

Kaynak: SELAM.tk