Pointer 101

Pointer; C/C++ dillerinde önemli bir yere sahip ve uzmanlaşmamız için biraz efor sarfetmemiz gereken bir konudur. Ama temel mantığını anladığımız sürece bize oldukça kolaylıklar sağlayacaktır. Bu yazımızda, birlikte pointer ile ilgili önemli işlevleri ve o işlevleri kavramamıza yardımcı olacak örnekleri gözden geçireceğiz.   
  
Pointer'ın Temel İşlevi  
  
Bilgisayar ortamındaki her bir veri değeri, hafızada belli bir yer kaplar: float ise 4 byte, char ise 1 byte, double ise 8 byte... Bir de değerlerin kapladığı alanı temsil eden adresler vardır. Adresin temsil ettiği bit değeri, mimariye göre değişiklik gösterir. Günümüzdeki bilgisayarların çoğunda ise her bir adres, 1 byte'lık (8 bit'lik) bir alanı temsil eder. Adresler onaltılık (hexadecimal) sisteme göre numaralandırılır.

Nasıl ki temelde gördüğümüz her bir değişken bir değeri ifade eder, pointer da o değişkenlerin adreslerini ifade eder. Bunun daha iyi anlaşılması için aşağıdaki basit kodu ve tabloyu inceleyelim.



Burada 42 sayısını atadığımız number değişkeninin adresine işaret edecek bir pointer tanımlandığını görüyoruz. Tanımlama için öncelikle pointer’ın veri tipi ile işaret edeceği adresteki değişkenin tipi aynı olmalı. Bu, ileriki aşamalarda bahsedeceğimiz pointer aritmetiği ile ilgili işlemlerde de bellek hatalarının olmaması için önemlidir. Yıldız işareti (\*), işlemciye bir pointer atanacağını ifade eder. &number, number değişkenin adres değeridir. Ampersand (&), isminin başına geldiği değişkenin adresini belirtir. nPtr isimli pointer’a, number’ın adresini atadığımıza göre printf ile ekrana çıktı gönderebiliriz:



Gördüğümüz üzere nPtr pointer’ının bulunduğu adres, işaret ettiği adres ve o adresin içeriğindeki değeri ekrana bastırdık. Evet, pointer da aslında adresi olan bir değişkendir: Diğer bildiğimiz değişkenlerle arasındaki fark ise değerinin bir adresi ifade etmesidir. Bunu daha iyi kavramak için yukarıdaki kodda yaptığımızı basit bir tabloya aktarmak mümkün:



Tablo, kodda 61fe1c adresindeki pointer’a, number’ın bulunduğu 61fe10 adresinin atandığını gösteriyor. Yani pointer, adres depolayan bir değişken görevi görüyor.

Pointer ile Call by Reference (Referans ile Çağrı)

Call by value (değer ile çağrı) işleminde fonksiyonlar, main’den aldığı değişkenlerin değerini değiştiremez. Değişkenleri klonlayıp kopyaları üzerinde işlemler yapar. Ama orijinal değişkenler üzerinde hiçbir etkisi olmaz. Örnek olarak bir sayının faktöriyelini hesaplayan koda bakalım.

number değişkeninin bir kopyası fonksiyonda parametre olarak kullanıldı. Görüldüğü üzere fonksiyon içindeki komutlar, verilen parametrenin değerini değiştiriyor. Ama number değişkeni üzerinde hiçbir etkisi olmuyor. Kodumuzu çalıştırıp sonucun nasıl olacağını birlikte görelim:



Evet, number değişkeni üzerinde hiçbir değişiklik yok. Peki ya değişkenin adresini kopyalayıp parametre olarak kullansaydık? İşte bu, doğrudan adresin içindeki bağımsız değişkene erişmemize olanak tanır. Aynı kod üzerinde oynamalar yapalım:



Gördüğünüz gibi buradaki fark, number’ın adres değerinin kopyasının fonksiyondaki pointer’a atanmasıdır.



Kodumuzu çalıştırdığımızda number değerinin değiştiğini fark ediyoruz. Call by value ile call by reference arasındaki bu farkın nedeni, birden fazla değişkenin “aynı” içeriğe sahip olabilmesi ama her birinin “farklı ve tek” bir adreste bulunmasında gizli.

Pointer, Dizi ve Pointer Aritmetiği

Diziler, birden fazla değeri temsil eden değişkenlerdir. Belirtilen değer sayısı kadar statik bellek tahsisi yapılır. Örneğin double tipinde dört değer tutan dizinin bellekte kapladığı alan 4xsizeof(double) = 32 byte olacaktır. Pointer ve dizinin belli başlı ortak yönleri vardır. Beraber inceleyelim.



Diziyi tanımladıktan sonra dizinin ismi olan arr2, bir adresi ifade eder. Yukarıdaki a değişkenine atadığımız basit mantıksal işleme bakacak olursak sonuç, 1 olacaktır. Çünkü arr2, birinci elemanın adresine işaret eden bir pointer görevi görür. Bu yüzden birinci elemanın adresinin içeriğini ifade eden \*arr2 ile arr2[0], aynı değeri ifade eder. Son olarak arr2’nin tuttuğu adres değeri hexadecimal biçimde ekrana yazdırılır. Ekran çıktısı:



Bir de pointer aritmetiğine göz atalım. Hatırlarsanız, pointer’ın veri tipinin bu hususta önemli olduğunu belirtmiştim. Evet, başlarda da belirttiğimiz gibi günümüzdeki çoğu bilgisayarda adreslerin her biri 1 byte’lık bir alanı temsil eder. İnt tipinde tek bir değişken tanımlayacak olursak 4 byte’lık bir alan kaplanacağı için değişken dört adrese depolanır. Eğer int tipinde 6 elemanlık bir dizi tanımlayacak olursak dizinin kapladığı alan 24 byte olacağı için bu bellek bloğunun adres sayısı da 24 olacaktır. Her bir elemanı ise 4 adreslik alan kaplayacaktır ve başlangıç adresleri içerikteki değeri temsil edecektir. Pointer aritmetiği adresler arasında geçişler yapmamızı mümkün kılar. Mesela 6 elemanlık bu dizide birinci elemanın adres değeri a ise, ikinci elemanın adres değeri a+sizeof(int) olur. Bunun nedeni, birinci elemanın adresine işaret eden pointer, int cinsinde tanımlı. Bu da birim artış değerini int boyutuna çeviriyor. Aşağıdaki kod ve tabloyla daha iyi anlaşılacak:



Dizinin ilk elemanının adresini belirten arr2’yi, aPtr isimli pointer’a atadık (Bu atamayı yapmadan da dizinin ilk elemanın pointerı ile geçişler yapabileceğinizi unutmayın. Ben daha anlaşılır olması açısından bir pointer kullandım. Ama dizi ismi olan arr2, bir constant pointer olduğu için sonradan adres değerinin değiştirilemeyeceğini ve bu yüzden “arr2++” değil de sadece “arr2+(sayı)” şeklinde geçişler için kullanabileceğimizi de hatırlatmakta fayda var) Burada her bir dizi elemanının adres ve değerini sırasıyla ekrana yazdıracağız. Kodumuzu çalıştıralım:



Tablomuz yardımıyla bellekteki işlemleri gözden geçirelim:



Evet, aPtr int türünde olduğu için “aPtr + (sayı)” ifadesi, “aPtr + sizeof(int) \* (sayı)” ifadesine eş değer oluyor.

Pointer ile Dinamik Bellek Tahsisi (Dynamic Memory Allocation)

Önceden de belirtiğim gibi, bir dizi tanımladığınızda eğer dizinin uzunluğu belliyse o dizi için statik bellek tahsisi yapılır. Mesela “int dizi[20];” ifadesini girdiğimizde bellekte 20\*sizeof(int)=80 byte’lık bir bellek bloğu tahsis edilir. Peki, önceden tanımlanmış dizi uzunluğundan daha az ya da daha fazla alana gerek duyarsak? Statik bellek tahsisiyle program çalışırken önceden tanımlı olan uzunluğu değiştiremezsiniz. Ya herhangi bir değer girmeden yalnızca kullanıcıdan istesek? Bu da mümkün değil. Çünkü program, dizinin uzunluğu belirtilmediği sürece çalışmayacaktır. Ama eğer pointer yardımıyla dinamik bellek tahsisi prosedüründen yararlanırsak, yukarıdaki bu sorunlar ortadan kalkacaktır. Burada yapacağımız uygulamalar pointer’ın işlevini anlamak amacını gütmekte. O yüzden dinamik bellek tahsisi gibi uzun bir konunun detaylarına inmemiz uygun değil. Farklı kaynaklardan bu konunun detaylarını araştırabilirsiniz. Öncelikle statik bellek tahsisi ile ilgili bahsettiğim iki durumu inceleyelim.



Gördüğünüz gibi bu durumda programın çalışması için öncelikle arr dizisinin uzunluğu belirtilmeli.



Bu sefer arr dizisinin uzunluğu kod çalıştırılmadan önce belirtildi. Bu durumda ise size’a verilen değer 5’ten büyükse bellek ihlali meydana gelecek. Genellikle bu tür bellek ihlallerini C derleyicileri algılamaz. O yüzden tahmin edilemeyecek türden sonuçlar meydana gelebilir. Programı çalıştıracak olursak:



Görüldüğü gibi tanımladığımız dizinin uzunluğu 5 olmasına rağmen bir eleman fazla ekrana yazdırılmış. Farklı uzunlukta tanımlı dizilerde bu fazlalık, farklı sonuçlar verecektir. Şimdi de dinamik bellek ayırma ile ilgili uygulamalarımızı yapalım.



Bu programda ben dinamik bellek tahsisi için kullanılan dört fonksiyondan ikisi olan malloc() ve free()’yi kullandım. Malloc() (memory allocation), bellek bloğunun (bu bellek bloğu bir array, structure vb. için ayrılmış olabilir) boyutunun değerini ifade eden bir parametreye sahip olan ve dönüş değeri olarak bu ayrılan bellek bloğunun başlangıç adresini döndüren bir fonksiyondur. Alternatifi olan calloc()’a (contiguous allocation) göre hız açısından daha avantajlıdır. Ama calloc()’un aksine, ayrılan bloklara otomatik olarak herhangi bir başlangıç değeri atanmadığı için atama yapılmadan okuma yapılırsa ekrana rastgele çöp değerleri dönecektir. O yüzden değerler okunmadan önce atama yapılması gerekir. Free() fonksiyonu ise program çalıştırıldıktan sonra bellekteki ayrılan bloğu serbest bırakır. Dinamik bellek tahsisi manuel olarak kontrol edildiği için bu konseptte bellekteki ayrılan alanı serbest bırakma işleminin programcı tarafından yapılması gerekir. Aksi taktirde programda istem dışı sonuçlar meydana gelebilir, bellek sızıntıları oluşabilir. Bu bilgiler ışığında programımızı incelersek ilk başta istediğimiz “size” değeri ile int’in boyut değerinin çarpımı uzunluğunda bir bellek bloğu ayrılıyor. Malloc, (void\*) tipi olduğu için ptarr’e atanırken eşitliğin sağında başta (int\*) tipine dönüştürüldü. Bu tipteki dönüşüm bloğun int boyutuna göre içeriklere ayrılmasını sağlıyor. Böylelikle bellek bloğumuz, int tipinde bir diziyi temsil etmiş oluyor. Bu dizinin her bir elemanını, diğer bir deyişle bellek bloğunun her bir içeriğini, birinci for döngüsündeki komutlarla tanımlayabiliyoruz. İkinci for döngüsüyle de girilen değerleri ekrana yazdırıyoruz. Son olarak free(ptarr) fonksiyonu ile ptarr için ayrılan bellek bloğunu serbest bırakıyoruz. Programımızı çalıştıralım:



İşte dinamik bellek ayırma işlemiyle farklı boyutta verileri işlememiz mümkün. Ayrıca ilerleyen bölümlerde struct kullanacağımız pointer uygulamalarında da dinamik bellek ayırmanın önemine bir kez daha değinmiş olacağız.

Fonksiyon İşaretçisi (Function Pointer)

Fonksiyon işaretçisi, adı üstünde fonksiyona işaret eder. Kendisine fonksiyonun adresi atanan bir pointer, o fonksiyondaki talimatlara erişim sağlar.



Görmüş olduğumuz örnekte iki int tipinde parametreye sahip ve void tipinde extra fonksiyonu, kendisiyle özdeş bir fonksiyon pointer’ına atandı. Bu atama sayesinde pointer’ı kullanarak istediğimiz parametre değerleriyle extra fonksiyonundaki talimatlara erişebiliriz.



Kodu çalıştırdığımızda fonksiyon isminin bir adres değeri döndürdüğünü görüyoruz. Bu değer, fonksiyonun başlangıç adresidir. Nasıl ki bir dizinin ismi, ilk elemanının adres değerini ifade ediyordu. Fonksiyon ismi de içindeki kodların başlangıç adresini ifade ediyor.

Fonksiyon pointer’ını tanımlarken dikkat edilmesi gereken noktalardan biri, parantez kullanılarak tanımlanmasıdır. int \*ptr(int,int) ifadesinde parantez olmadığı için derleyici bunu int tipinde pointer dönüşlü ve iki int tipinde parametreye sahip bir fonksiyon olarak algılar. Eğer int (\*ptr)(int,int) olarak kullanılırsa bir fonksiyon işaretçisi tanımlanmış olur. Pointer dönüşlü fonksiyonları da ilerleyen bölümlerde inceleyeceğiz.

Dizilere değinmişken, bir fonksiyon işaretçisini birden fazla fonksiyona sahip bir dizi şeklinde kullanabiliriz. Bir örnek üzerinden inceleyelim.



Bu programda iki sayıyı karşılaştıran, ortalama değerlerini ve birinci sayının ikinci sayıya bölümünden kalanını veren ve bu bilgileri ekrana yazdıran üç void tipi fonksiyonu, void tipi bir fonksiyon pointer’ına görülen sırayla atadık. Bu sayede kullanıcıdan yapmak istediği işlemi ve kullanmak istediği iki sayıyı alıp değeri (\*ptr[c])(a,b) ifadesiyle ekrana yazdırabiliyoruz.







Yukarıdaki görseller, üç işlevin ayrı ayrı çıktılarını ifade ediyor.

Bir de fonksiyon işaretçisi sayesinde herhangi bir fonksiyonu farklı bir fonksiyonda parametre olarak kullanabiliyor ve sonuç olarak döndürebiliyoruz.





Bu örnekte kullanıcının girmiş olduğu rakamın rastgele rakam barındıran b’nin değeriyle eşit olup olmadığını belirliyoruz. “esitlikdurumu” fonksiyonu, doğru cevap verildiğinde “esit”, yanlış cevap verildiğinde “esitdegil” fonksiyonunu parametre olarak alır. Bu işlem fonksiyon pointer’ı sayesinde mümkün olur.

Double Pointer

Pointer’lar ötelenebilme özelliğine sahiptirler. Bir a değişkeninin pointer’ı a’ya işaret ettiği gibi a değişkenin pointer’ının pointer’ı da a değişkeninin pointer’ına işaret eder.



C dilinde pointer boyutunda belli bir sınır yoktur yalnız ne kadar fazla boyutta pointer kullanırsak bir o kadar da programımız karmaşık ve hataya müsait olur. Biz burada örnek ve yaygın olması açısından double pointer’ları ele alacağız.

Double pointer’ın önemli uygulamalarından biri bellek ayırarak matris (iki boyutlu dizi) oluşturmaktır. Bir örnek üzerinden inceleyelim.



Burada size\_1 dizimizin satır sayısını, size\_2 ise sütun sayısını ifade ediyor. malloc() fonksiyonunu kullanarak öncelikle satır sayısı kadar her bir sütundaki elemanlara işaret edecek pointer’ı, dptr ismindeki double pointer için ayırdık. Sonra bu ayrılan satır sayısı kadar pointer’a sütun sayısı kadar integer boyutunda bellek bloğu ayırdık. Sonra da random algoritmamızın başlangıç değerini ifade eden srand() fonksiyonunu kullanıp programın her başlatıldığı anda farklı bir değerin ekrana yazdırılması için mevcut POSIX zamanını (UNIX olarak da bilinir, başlangıç tarihi 1 Ocak 1970’tir) saniye cinsinde tutan time(NULL) ifadesini seed olarak kullandık. Sonra iç içe for döngüsü yardımıyla her bir matris (iki boyutlu dizi) elemanına sırasıyla random değerler atadık. Son olarak da dinamik bellek ayırma sürecinde asla unutmamamız gerek fonksiyon olan free() ile dptr için ayrılan bellek bloklarını serbest bıraktık. Programı çalıştırırsak:



Bu bellek ayırma işlemini char tipi bir iki boyutlu dizi için de yapabilir ve istediğimiz uzunlukta cümleler oluşturabiliriz.

Bir diğer önemli uygulamalarından biri de call by reference’dır. Call by reference, önceden de tanımladığımız gibi, fonksiyon aracılığıyla bir değişkenin içeriğini değiştirmek için adresini parametre olarak kullanmaktır.



Bu örnekte c’nin adres değerini tutan cPtr ile d’nin adres değerini tutan dPtr’nin tuttukları adres değerlerini değiştirdik. Bunu yapmak için degisim ismindeki void tipi ve iki double pointer parametreli fonksiyonu kullandık.

