**İşaretçiler (Pointers)**

[**► Detaylı anlatım**](https://www.bilgigunlugum.net/prog/cprog/2c_isaretci)

İşaretçiler, bilgisayar belleğindeki belli bir adresin, genellikle bir değişkene ait, kaydedildiği ve kaydedilen bellek adresine doğrudan erişim sağlayan değişkenlerdir.

* [İşaretçi bildirimi](https://www.bilgigunlugum.net/prog/cprog/c_isaretci#c_isa01)
* [İşaretçilere değer atama](https://www.bilgigunlugum.net/prog/cprog/c_isaretci#c_isa02)
* [İşaretçilere ilk değer atama](https://www.bilgigunlugum.net/prog/cprog/c_isaretci#c_isa03)
* [İşaretçiler yoluyla değişkenlere değer atama](https://www.bilgigunlugum.net/prog/cprog/c_isaretci#c_isa04)
* [Diğer işaretçi işlemcileri ve işlemci aritmetiği](https://www.bilgigunlugum.net/prog/cprog/c_isaretci#c_isa05)
* [İşaretçiler yoluyla değişken değerlerini değiştirme](https://www.bilgigunlugum.net/prog/cprog/c_isaretci#c_isa06)

**İşaretçi bildirimi**

İşaretçilerin diğer değişkenlerden tek farkı başka bir değişkenin bellek adresini içeriyor olmasıdır. Aşağıdaki satır işaretçi bildiriminin genel yapısını göstermektedir:

**veri-türü** **\*değişken-adı**;

Yukarıdaki satırda yer alan **veri-türü** ifadesi işaretçinin adresini gösterdiği değerin veri türünü, **değişken-adı** ifadesi ise işaretçinin adını göstermektedir. \* işareti işaretçi değişken bildiriminde kullanılmaktadır. Aşağıdaki işlem satırı ip adlı int bir işaretçi değişkeni oluşturmaktadır:

int \*ip;

İşaretçiler, bildirimi yapıldıktan sonra doğrudan kullanılamazlar. Kullanmadan önce mutlaka bir bellek adresi atanması gerekir.

**İşaretçilere değer atanması**

Bir işaretçiyi kullanmadan önce, bir bellek adresinin mutlaka işaretçiye atanması gerekir. Çünkü, işaretçi bellek adresleri ile işlem yapar. C'de, işaretçilere bellek adresi atama işlemi, **&** işlemcisi ile yapılır.

C'de işaretçilerle kullanılan iki farklı işlemci vardır. Bunlardan biri \* , diğeri ise & işlemcisidir. Bu iki işlemcinin görevlerinin anlaşılması işaretçi kavramını anlamak açısından çok önemlidir:

**\*** : Hangi işaretçi değişken adından önce kullanılırsa, o işaretçi değişkene adresi atanan değişkenin **değerini** verir.

**&** : Hangi değişken adından önce kullanılırsa, o değişkenin **adresini** verir.

İşaretçi tanımlamalarında veri türünü belirlerken, daha sonra adresini işaretçiye atanacak değişkenin veri türünü dikkate almamız gerekir. Başka bir ifade ile, işaretçi ile işaretçiye bellek adresi atanan değişkenin veri türü aynı olmalıdır.

Konunun daha kolay anlaşılması için işaretçi kullanılması ile \* ve & işlemcilerini örnekler üzerinde incelemeye çalışalım:

Örnek

#include <stdio.h>

int main(void)

{

int \*ip, id; // int bir işaretçi ve değişken bildirimi

id = 21;

ip = &id; // id değişken adresini ip işaretçisine atar.

// İşaretçi kullanarak id değişken değerini ekrana yazar.

printf("id değişken değeri: %d", \*ip);

return 0;

}

Yukarıdaki programı derleyip çalıştırdığımızda, aşağıdaki ifadeleri ekrana yazar:

id değişken değeri: 21

Program önce id adlı bir int değişken ve ip adlı bir işaretçi değişkeni oluşturur. id değişkenine 21 değerini atar. id değişkeninin adresini ip işaretçisine atar. id değişken değerini işaretçi kullanarak ekrana yazar.

Örnek

#include <stdio.h>

int main(void)

{

int \*ip, id; // int bir işaretçi ve değişken bildirimi

id = 21;

ip = &id; // id değişken adresini ip işaretçisine atar.

// Değişken adını kullanarak id değişken değerini ekrana yazar.

printf("id değişken değeri: %d\n", id);

// İşaretçi kullanarak id değişken değerini ekrana yazar.

printf("id değişken değeri: %d\n", \*ip);

// Değişken referansını kullanarak id değişken adresini ekrana yazar.

printf("id değişken bellek adresi: %p\n", &id);

// İşaretçi adını kullanarak id değişken adresini ekrana yazar.

printf( "id değişken bellek adresi: %p", ip);

return 0;

}

Yukarıdaki örnekte, program ekrana aşağıdaki satırları yazar:

id değişken değeri: 21

id değişken değeri: 21

id değişken bellek adresi: 006dfee8

id değişken bellek adresi: 006dfee8

Program önce id adlı bir int değişken ve ip adlı bir işaretçi değişkeni oluşturur. id değişkenine 21 değerini atar. id değişkeninin adresini ip işaretçisine atar. id değişken değerini önce değişken adını sonra işaretçi kullanarak, iki kez ekrana yazar. id değişkeninin bellek adresini önce değişken referans değerini sonra işaretçi adını kullanarak ekrana yazar. Bellek adresi her bilgisayarda farklı bir değer alabilir.

Konumuza devam etmeden önce, **0x6dfee8** ifadesini kısaca inceleyelim. Bu ifade bilgisayar belleğinin belirli bir adresini 16'lık (Hexadecimal) sayı sisteminde göstermektedir. Yukarıdaki örnek programı çalıştırdığımızda farklı bir bellek adres değeri alabiliriz. Burada gösterilen bütün adres değerleri, her bilgisayar için farklı olabilir.

**İşaretçilere ilk değer atama**

Global veya statik olarak tanımlanan işaretçilere, tanımlanır tanımlanmaz otomatik olarak NULL bir değer ilk değer olarak atanır.

Statik olmayan ve yerel bir işaretçi bildirimi yaptığımızda ise, bu işaretçiye bir bellek adresi atamanın iki farklı yöntemi vardır:

1. Bir değişkenin bellek adresini & işlemcisi ile işaretçiye atamak.
2. malloc() fonksiyonu yoluyla tahsis edilen bir bellek bölümünün başlangıç adresini işaretçiye atamak.

Bu iki yöntemle işaretçiye bir bellek adresi atamadan önce, işaretçi bilinmeyen belirsiz bir değer taşır. Eğer işaretçiye bir değer atamadan kullanırsak program beklenmeyen sonuçlar verir.

Programlarda oluşabilecek muhtemel sorunları çözmek için, bir bellek adresi atanmamış statik olmayan ve yerel işaretçilerde ilk değer olarak 0 veya NULL değeri ilk değer olarak verilir.

char \*p = 0;

veya

char \*p = NULL;

Ancak, bu uygulama sorunun tamamen çözülmesi için yeterli değildir. Bu işlem satırından sonra da işaretçiye atanan değerlerin bir bellek adresi olmasına dikkat edilmelidir.

C'de, bir işaretçiye karakter dizisi sabitlerini ilk değer olarak atayabiliriz.

char \*cp = "Bilgisayar";

Burada, cp bir işaretçi olduğundan sadece bir bellek adresini gösterdiği kabul edilir. Ancak, C'de derleyiciler program tarafından kullanılan karakter dizilerini yükledikleri karakter dizisi tablosu adlı bir tablo oluştururlar. Böylece, program bir işaretçi değişkeni ile gösterilen bir karakter dizisini karakter dizisi tablosundan okuyarak işlem yapar.

**İşaretçiler yoluyla değişkenlere değer atama**

İşaretçileri kullanarak, işaretçinin adresini içerdiği değişkene bir değer atayabiliriz. Bu özelliği bir örnek üzerinde incelemeye çalışalım:

Örnek

#include <stdio.h>

int main(void)

{

int \*ip1, \*ip2, id1, id2;

ip1 = &id1; // id1 değişkeninin adresini ip1 işaretçisine atama

ip2 = &id2; // id2 değişkeninin adresini ip2 işaretçisine atama

id1 = 42; // id1 değişkenine değer atama

\*ip2 = 67; // id2 değişkenine ip2 işaretçisi yoluyla değer atama

printf("id1 değişkeninin değeri: %d\n", \*ip1);

printf("id2 değişkeninin değeri: %d\n", \*ip2);

printf("id1 değişkeninin bellek adresi: %p\n", ip1);

printf("id2 değişkeninin bellek adresi: %p", ip2);

printf("\n\n");

printf("ip1 işaretçisinin bellek adresi: %p\n", &ip1);

printf("ip2 işaretçisinin bellek adresi: %p", &ip2);

return 0;

}

Yukarıdaki programı derleyip çalıştırdığımızda, aşağıdaki ifadeleri ekrana yazar:

id1 değişkeninin değeri: 42

id2 değişkeninin değeri: 67

id1 değişkeninin bellek adresi: 006dfee4

id2 değişkeninin bellek adresi: 006dfee0

ip1 işaretçisinin bellek adresi: 006dfeec

ip2 işaretçisinin bellek adresi: 006dfee8

Program önce ip1 ve ip2 adlı 2 adet işaretçi değişkeni ile id1 ve id2 adlı 2 adet int değişken tanımlar. id1 değişkeninin bellek adresini ip1 işaretçisine id2 değişkeninin bellek adresini ip2 işaretçisine atar. id1 değişkenine direk olarak 42 sayısını atar. id2 değişkenine ise, ip2 işaretçisini kullanarak, 67 sayısını dolaylı yöntemle atar. Sonra id1 ve id2 değişkenlerinin değerlerini ve bellek adreslerini ekrana yazar. Programın her işlem satırından sonraki bellek durumu ise aşağıda gösterilmektedir:

Bir işaretçi kullanmadan önce, bir bellek adresi mutlaka işaretçiye atanmalıdır. Eğer bir bellek adresi atamadan işaretçi kullanılırsa, istenen netice elde edilemez. Çünkü, bir işaretçi tanımladığımızda sadece bir bellek adresini atayabilecek bir değişken oluşturmuş oluruz. Ancak, işaretçi değişkenine atanmış herhangi bir bellek adres değeri yoktur. Aynı şekilde eğer bir işaretçi değişken bildiriminden sonra, bir değişkenin bellek adresini işaretçiye atamadan, bir int değeri işaretçi değişkenine atarsak, yine istediğimiz neticeyi elde edemeyiz.

Aşağıda yer alan işlem satırlarından 1 ve 2 sayıları ile gösterilen satırlar, C dili kurallarına göre geçersizdir:

int \*ip;

\*ip = 357; // 1

printf() << \*ip; // 2

Şimdi, işaretçi kullanılan farklı bir örneği incelemeye çalışalım:

Örnek

#include <stdio.h>

int main(void)

{

char \*cp, cd;

cp = &cd;

for (cd='A'; cd<'L'; cd++) printf("%c ", \*cp);

return 0;

}

Yukarıdaki programı derleyip çalıştırdığımızda, aşağıdaki ifadeleri ekrana yazar:

A B C D E F G H I J K

Program büyük A harfinden başlamak üzere ASCII karakterlerin alfabenin ilk 11 tanesini ekrana yazar. Program önce cp adlı işaretçi bir değişken ve cd adlı bir char değişken tanımlar. cd değişkeninin bellek adresini cp işaretçisine atar. Bir for döngüsü ve cp işaretçini kullanarak sıra ile harflerin atandığı cd değişken değerlerini ekrana yazar. Burada, program for döngüsü yoluyla, cd değişken değerini 10 kez değiştirir ve her defasında cp işaretçisini kullanarak değişken değerini ekrana yazar.

**Diğer işaretçi işlemcileri ve işlemci aritmetiği**

Daha önce incelediğimiz \* ve & işlemcilerine ek olarak, işaretçilerle kullanabileceğiniz 4 işlemci daha vardır:

**+ - ++ --**

Artırma ve azaltma işlemcilerinin işaretçilerle kullanılmasında özel bir durum vardır. Herhangi bir veri türünden tanımlanmış olan bir işaretçi değerini bir değer artırmak veya azaltmak istediğimizde, işaretçi değere sadece **tanımlandığı veri türünün boyutu** kadar bir değer ekler veya çıkarabiliriz.

int \*ip, id;

ip = &id;

id = 21;

ip = ip + 1; // ip işaretçisinin gösterdiği bellek adresini int veri boyutu kadar ileri taşır.

ip = ip + (1 \* sizeof (int)); // ip işaretçisinin gösterdiği bellek adresini int veri boyutu kadar ileri taşır.

Yukarıdaki son iki işlem satırının her ikisinde de, ip işaretçisinin gösterdiği bellek adresi int veri türü boyutu kadar (4 byte) ileri taşınır.

Şimdi, bu özelliği bir örnek üzerinde incelemeye çalışalım:

Örnek

#include <stdio.h>

int main(void)

{

char cdizi[] = { 'A', 'B', 'C', 'D', 'E', 'F' };

char \*cp;

int len, id;

cp = cdizi; // cp = &cdizi[0];

len = sizeof(cdizi) / sizeof(char);

for (id=0; id<len; id++) printf("%c ", \*cp++);

return 0;

}

Yukarıdaki programı derleyip çalıştırdığımızda, aşağıdaki ifadeleri ekrana yazar:

A B C D E F

Program, cdizi adlı bir diziye ilk değer atama yöntemi ile A-F arasındaki harfleri atar. Sonra, cp adlı bir işaretçi ile len ve id adlı iki adet int değişken bildirimi yapar. cdizi dizisinin bellek adresini cp işaretçisine atar. Dizi eleman sayısını, dizi boyutunu char veri türü boyutuna bölerek elde eder ve len değişkenine atar. Bir for döngüsü, dizi eleman değerlerini ekrana yazar. for döngüsünün her tekrarında cp işaretçi değeri bir byte (char veri türü için) kadar artırılarak, bir sonraki dizi elemanının bellek adresini göstermesi sağlanır.

Örnek

#include <stdio.h>

int main(void)

{

int \*ip;

int idizi[5] = { 5, 17, 21, 34, 46 };

int id;

printf("İşaretçi bellek adresi: %p\n\n", &ip);

ip = idizi; // ip = &idizi[0];

for (id=0; id<5; id++, ip++) {

printf("İşaretçinin içerdiği bellek adresi: %p %p\n", ip, &idizi[id]);

printf("İşaretçinin gösterdiği değişken değeri: %d %d\n\n", \*ip, idizi[id]);

}

return 0;

}

Yukarıdaki programı derleyip çalıştırdığımızda, aşağıdaki ifadeleri ekrana yazar:

İşaretçi bellek adresi: 006dfed8

İşaretçinin içerdiği bellek adresi: 006dfec4 006dfec4

İşaretçinin gösterdiği değişken değeri: 5 5

İşaretçinin içerdiği bellek adresi: 006dfec8 006dfec8

İşaretçinin gösterdiği değişken değeri: 17 17

İşaretçinin içerdiği bellek adresi: 006dfecc 006dfecc

İşaretçinin gösterdiği değişken değeri: 21 21

İşaretçinin içerdiği bellek adresi: 006dfed0 006dfed0

İşaretçinin gösterdiği değişken değeri: 34 34

İşaretçinin içerdiği bellek adresi: 006dfed4 006dfed4

İşaretçinin gösterdiği değişken değeri: 46 46

Program, önce ip adlı int bir işaretçi ile idizi adlı 5 elemanlı int bir dizi tanımlar. İşaretçi bellek adresini ekrana yazdıktan sonra, dizinin ilk elemanının adresini işaretçiye atar. Bir for döngüsü içinde, her defasında işaretçi değerini int boyutu kadar artırmak suretiyle, işaretçinin bir sonraki dizi elemanının adresini göstermesini sağlayarak işaretçinin içerdiği adresi ve gösterdiği dizi değişken değerini hem işaretçi adı hem de dizi adını kullanarak ikişer defa ekrana yazar.

for döngüsüne ilk girdiğinde bellek durumu aşağıdaki ilk şekildeki gibidir. Döngü içinde ip++ ifadesi ilk çalıştığında ise, ip int bir işaretçi olduğundan, int boyutu kadar (32 bit bilgisayarlarda 4 byte) yukarıdaki bellek adresinin değerini alır. Yani, daha önce idizi[0] değişken adresini içeren ip işaretçisi artık idizi[1] değişkeninin adresini içermektedir. Aslında, işlem satırındaki ip++ ifadesinin gerçek değeri aşağıdaki gibidir.

ip = ip + (1\*sizeof(int));

Sonuç olarak, for döngüsünün her tekrarında ip++ işlemi ile ip işaretçisinin içerdiği adres int boyutu kadar artarak bir sonraki dizi değerinin adresini gösterir.

\* Bir işaretçiye eklenen veya çıkarılan değer mutlaka int bir değer olmalıdır.

\* İşaretçilerle toplama veya çıkarma dışında bir işlem yapılamaz.

Örnek

#include <stdio.h>

int main(void)

{

int \*ip;

int idizi[5] = { 15, 72, 154, 298, 345 };

printf("İşaretçi bellek adresi: %p\n\n", &ip);

ip = &idizi[4];

printf("İşaretçinin içerdiği adres: %p\n", ip);

printf("İşaretçinin gösterdiği değişken değeri: %d\n\n", \*ip);

ip-=2; // ip -= (4 \* sizeof (int));

printf("İşaretçinin içerdiği adres: %p\n", ip);

printf("İşaretçinin gösterdiği değişken değeri: %d\n\n", \*ip);

return 0;

}

Yukarıdaki programı derleyip çalıştırdığımızda, aşağıdaki ifadeleri ekrana yazar:

İşaretçi bellek adresi: 006dfeec

İşaretçinin içerdiği adres: 006dfee8

İşaretçinin gösterdiği değişken değeri: 345

İşaretçinin içerdiği adres: 006dfee0

İşaretçinin gösterdiği değişken değeri: 154

Program, önce ip adlı int bir işaretçi ile idizi adlı 5 elemanlı int bir dizi oluşturur. İşaretçi bellek adresini ekrana yazdıktan sonra, dizinin son elemanının adresini işaretçiye atar. İşaretçinin içerdiği adresi ve gösterdiği dizi değişken değerini ekrana yazar. Sonra, işaretçiden 4 int boyutu kadar değer çıkararak işaretçinin ilk dizi elemanının adresini göstermesini sağlayarak, işaretçinin gösterdiği adresi ve bu adreste bulunan dizi değerini ekrana yazar.

Programın çalışma süresince bellek içeriğindeki değişim aşağıda gösterilmektedir:

**İşaretçiler yoluyla değişken değerlerinin değiştirilmesi**

++ ve -- operatörleri ile normal toplama ve çıkarma işlemlerini kullanarak işaretçinin adresini gösterdiği değişken değerini dolaylı yöntemle artırabiliriz:

Örnek

#include <stdio.h>

int main(void)

{

int \*ip, id;

id = 274;

ip = &id;

printf("id değişken değeri: %d\n", id);

// ip işaretçisinin adresini gösterdiği değişken değerini artırma

(\*ip)++;

printf("id değişken değeri: %d", id);

return 0;

}

Yukarıdaki programı derleyip çalıştırdığımızda, aşağıdaki ifadeleri ekrana yazar:

id değişken değeri: 274

id değişken değeri: 275

Program, ip adlı int bir işaretçi ve id adlı int bir değişken oluşturur. id değişkenine 274 sayısını atar. id değişkeninin adresini ip işaretçisine atar. Değişken değerini ekrana yazdıktan sonra ip işaretçisini kullanarak, id değişken değerini artırır ve değişken değerini tekrar ekrana yazar.