**Pointerlar (Göstericiler)**

Bilgisayarda tutulan her değişkenin bellekte bir adresi vardır. Bu değişkenlerin adresleri tutan değişkenlere pointer(gösterici) denilmektedir. Pointer bir değişkeni de gösterebilir bir dizinin ilk elemanını da gösterebilir. Pointerlarda kullanılan ve C ile programlama oldukça önemli bir yere sahip operatörler vardır. Bunlardan biri içerik operatörüdür. Bunlardan ilk öğrenmemiz gereken ampersan(address of operator) operatörüdür. Bu operatör ise adından da anlaşılabileceği gibi herhangi bir değişkenin adresinin elde edilmesini sağlamaktadır. Bu operatör operatör öncelik tablosunun ikinci öncelik seviyesinde bulunan ve unary kullanıma sahip olan bir operatördür. Bu operatörün de binary kullanıma sahip hali vardır. Bu da ilerde ele alıcağımız bitsel operatörlerden bitsel ve(&) operatöründen başkası değildir.

Ampersan operatörü ile elde edilen adres 16 sistemde(hexadecimal) bir adrestir. Bir pointer değişken aşağıdaki gibi tanımlanabilir:

int main(void)

{

int x = 10;

int\* ptr = &x;

}

Bu ifadenin ingilizce ifadesi pointer to x olarak isimlendirilir. Bu ptr’ın şekil ile gösterimi aşağıdaki gibi yapılabilir:

yazı tipi, çizgi, beyaz, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Anahtar Notlar:** Burada ptr değişkeninin türü int türünden değil int\* türündendir.

Burada x’in tutulduğu bellek bloğu 4 bytelık bir bellek bloğudur. Bu bildirimden sonra x değişkeninin adresi demek ile ptr demek aynı anlama gelmektedir. Printf fonksiyonu kullanılarak adres de basmak mümkündür. Bunun için %p ifadesi kullanılır. Örneğin:

#include <stdio.h>

int main(void)

{

int x = 34;

int\*ptr = &x;

printf(“%p\n”, &x);

printf(“%p\n”, ptr);

printf(“%p\n”,&ptr);

}

Bu kod herhangi bir derleyicide derlendiğinde x değişkeninin adresiyle ptr değişkeni aynı adrese sahip olduğu görülmektedir. Yani x değişkeninin adresi demek ptr demektir. Ancak ptr değişkeninin adresi x değişkeninin adresinden farklı olduğu da görülmektedir.

Bir pointer değişken default init. edilirken NULL macrosu kullanılabilir veya direkt olarak hiçbir şey de yazılmayabilir. Örneğin aşağıdaki örnekte görüldüğü gibi:

#include <stdio.h>

int main(void)

{

int\*p1 = NULL;

int\*p2;

}

Bunun yanı sıra pointerlar hep aynı adresi göstermek zorunda değildir. Örneğin aşağıdaki örnekte ptr değişkeni önce x değişkenini gösterirken sonradan y değişkenini göstermektedir. Yani kısaca en başta ptr, x değişkeninin adresidir. Sonrasında ise ptr, y değişkeninin adresidir. Buradaki printf çağrılarından bu durum net bir şekilde görülmektedir:  
#include <stdio.h>

int main(void)

{

int x = 10;

int y = 34;

int\* ptr = &x;

printf("&x : %p\n", &x);

printf("&y : %p\n", &y);

printf("ptr : %p\n",ptr);

ptr = &y;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("&x : %p\n", &x);

printf("&y : %p\n", &y);

printf("ptr : %p\n", ptr);

}

Bunun diyagram şekilde gösterimi de ilk başta şu şekilde:

metin, yazı tipi, çizgi, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

Son durumda ise şu şekilde olacaktır:

metin, yazı tipi, çizgi, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yani ilk adımda ptr pointer to x olarak isimlendirilirken ikinci durumda pointer to y olarak isimlendirilebilir.

Ampersan operatörü hakkında az çok bir fikrimiz olduğuna göre en önemli diğer bir operatör olan içerik operatörüyle konuya devam edebiliriz. Bu operatör, operatör öncelik tablosunun ikinci öncelik seviyesinde bulunan ve unary kullanıma sahip olan bir operatördür. Bu operatörünün ingilizce adı “dereferencing operator”’dür. İçerik operatörü “\*” ile ifade edilir. İlk aklımıza gelen çarpma operatörüyle aynı gösterime sahip olduğundan derleyici çarpım operatörüyle içerik operatörünü nasıl ayırt ettiği olacaktır. Bu aslında işaret artı(+) ve toplam artı(+) arasındaki fark ile aynıdır. Örneğin eğer bir operatör iki tane operand alıyorsa bu buna binary kullanım adı verilir. Ancak söz konusu operatör tek bir operand alıyorsa bu durumda unary kullanıma sahiptir. İşte bir artının(+) işaret artı olduğunu da unary kullanımdan anlarız. Benzer şekilde yıldızın(\*) içerik operatörü mü yoksa çarpım operatörü mü olduğunu unary veya binary kullanılmasından anlayabiliriz. Örneğin eğer tek bir operand almış ise, diğer bir adıyla unary ise, bu durumda içerik operatörüdür diyebiliriz. Ancak iki operand almış ise, diğer bir adıyla binary ise, bu durumda çarpım operatörüdür. Örneğin aşağıdaki örnekte çarpım operatörü kullanılmıştır:

#include <stdio.h>

int main(void)

{

int a,b;

printf(“iki sayi giriniz?\n”);

scanf(“%d%d”,&a,&b);

int result = a \* b; //çarpım operatörü

}

İçerik operatörü kullanılarak pointer değişkenin gösterdiği nesnenin kendisine erişilebilir. Örneğin aşağıdaki örnekte ptr dereference edildiği zaman elde edilen değer aslında x demektir:

#include <stdio.h>

int main(void)

{

int x = 10;

int\* ptr = &x;

printf("\*ptr : %d\n",\*ptr);

printf("x : %d\n",x);

}

Bu örnekteki kod çalıştırıldığında \*ptr ile x aynı sonucu elde edilecektir. Çünkü ptr pointer to x olduğu için onu reference etmek x’in değeri elde edilir.

**Anahtar Notlar:** içerik operatörü yalnızca pointer değişkenlerle kullanılabilir. Normal bir değişken ile kullanımı doğrudan sentaks hatasına neden olur örneğin:

#include <stdio.h>

int main(void)

{

int x = 10;

printf(“%d”,\*x);//error

}

İlerde ele alıcağımız pointer to pointerlar diğer bir değişle göstericiyi gösteren göstericiler konusunda iki defa dereferencing operatörünün nasıl kullanılacağını inceleyeceğiz.

**Anahtar Notlar:** dereferencing operatörlüyle ampersan operatörü birlikte de kullanılabilir. Bu durumda elde edilen değer sayının kendisidir. Örneğin:

#include <stdio.h>

int main(void)

{

int x = 10;

printf("x : %d\n",\*&x);//x'in kendisi

}

Saçma görünse ve üretimde pek kullanılmasa da bu sınırsız şekilde yapılabilir örneğin aşağıdaki gibi ifade geçerlidir:

#include <stdio.h>

int main(void)

{

int x = 10;

printf("x : %d\n",\*&\*&\*&\*&\*&x);//legal

}

Pointerlar değişkenleri göstericekleri gibi dizilerin ilk elemanlarını da gösterebilirler. Örneğin aşağıdaki örnekte olduğu gibi:

int main(void)

{

int arr[] = {1,2,3};

int\*ptr = arr;

}

**Anahtar Notlar :** bir pointer değişkenin dizinin ilk elemanının adresini göstermesiyle kendisini göstermesi arasında herhangi bir fark yoktur. Örneğin yukarıdaki örnekte arr yerine &arr[0] yazılsaydı da aynıdır. Bunun nedeni aslında index of operatörün başka bir anlama daha gelmesinden dolayıdır. Bu özellik arr[0] demek ile \*(arr +0) aynı anlama gelmektedir. Bu sebepten son durumda &arr[0] ifade &\*(arr + 0) aynı anlama gelir. Sonuç olarak ampersan ve dereferecing operatörü birbirini götürür son durumda arr kalır. Bu yüzden eskiden index of operatörüyle yapılan işler dereferencing operatörüyle de yapılabilir. Örneğin:

#include <stdio.h>

#include <stdlib.h>

#define randomize() srand(time(NULL))

#define SIZE 20

void setRandomArray(int arr[],int size)

{

for(int i = 0;i < size;++i)

{

\*(arr + i) = rand() % 100;

}

}

void printArray(int arr[],int size)

{

for (int i = 0; i < size; ++i) {

printf("%d ",\*(arr + i));

}

printf("\n");

}

int main(void)

{

randomize();

int arr[SIZE];

setRandomArray(arr,SIZE);

printArray(arr,SIZE);

}

**Pointer Aritmetiği**

Bir diziyi gösteren pointerlar ++ operatörünün operandı yapılarak dizide dolaşılabilir. Örneğin:

#include <stdio.h>

#include <stdlib.h>

#define randomize() srand(time(NULL))

#define SIZE 20

void setRandomArray(int arr[],int size)

{

while(size--)

{

\*arr++ = rand() % 100;

}

}

void printArray(int arr[],int size)

{

while(size--)

{

printf("%d ",\*arr++);

}

printf("\n");

}

int main(void)

{

randomize();

int arr[SIZE];

setRandomArray(arr,SIZE);

printArray(arr,SIZE);

}

Bu örnekte diziyi gösteren pointerlar başlangıçta dizinin ilk elemanını göstermektedir. O pointerı arttırmak demek aslında while döngüsünün ikinci adımında pointer dizinin bir indisli veya diğer bir deyişle ikinci elemanını göstermektedir. Bu çoğu açıdan efektif bir özelliktir. Kullanılan bu özelliğe pointer aritmetiği denilmektedir.

**Pointer Ne İşe Yarıyor?**

Pointerlar üretimde sıkça kullanılmaktadır. Temel amacı çoğunlukla bilinmemektedir. Pointerlar aslında fonksiyonlarla kullanıldığında efektif olmaktadır. Normalde fonksiyonların parametre değişkenleri örneğin bir temel tür olduğu zaman bu durumda kopyalama semantiği devreye girer. Ancak mesela bu parametre değişkeni int yerine int\* türünden olsaydı bu durumda kopyalama semantiği devreye girmeyecekti. Bu da örneğin 10 elemanlı bir int dizi için 40 bytelık bellek bloğuna kopyalama gerçekleşecekti ama bunun yerine sadece bir pointer sizeof değeri kadarlık bir bellek bloğu kullanılacak. Bu bellek tasarrufu için oldukça önemlidir.

Diğer bir efektif kullanımı ise değişkenlerin call by reference yoluyla bir fonksiyona gönderilmesini sağlamak. Örneğin bir değişken takas fonksiyonu yazılacaksa bu kopyalama semantiğiyle yapılamazdı. Örneğin aşağıdaki örnekte swap fonksiyonunu görmeden bu fonksiyon bu iki değişkeni takas edemez dememiz gerekir:

int main(void)

{

int x = 15;

int y = 9;

swap(x,y);

}

Bunun nedeni de bu fonksiyonun call by value olmasından kaynaklanmaktadır. Bu fonksiyon aşağıdaki gibi çağrılsa bu değişkenleri takas edebileceği söylenebilirdi. Örneğin:

int main(void)

{

int x = 15;

int y = 9;

swap(&x,&y);

}

Çünkü bu durumda fonksiyon call by referencedır. Bu yüzden bu değişkenlerin fonksiyon içerisinde değiştirilmesi onu çağıran kaynak dosyayı doğrudan etkiler. Bu da takas değişiklik işleminin yapılmasına olanak sağlanır. Örneğin aşağıdaki kod kopyalama semantiği kullanılarak yazılamazdı:

#include <stdio.h>

void swap(int\* p1,int\* p2)

{

int temp = \*p1;

\*p1 = \*p2;

\*p2 = temp;

}

int main(void)

{

int x = 10;

int y = 34;

printf("x : %d\ny : %d\n",x,y);

swap(&x,&y);

printf("x : %d\ny : %d\n",x,y);

}

Özetlemek gerekirse neden call by reference?

1-) Bellek tasarrufu için bazı durumlarda çok büyük byte değerlerine sahip diziler veya dizi dizilerine yerine sadece bir pointer kullanılarak kod çok daha efektif hale getirebilir.

2-) Call by value ile yapılamayan örneğin fonksiyona gönderilen değişkenin fonksiyon içinde değiştirilmesi durumundan o fonksiyonu çağıran kaynağın bu değişiklikten doğrudan etkilenmesi için

**Pointerlarla İşlemler**

Pointerlarla işlemlerde bazı kurallar vardır:

1-) Bir pointerla bir pointer toplanması geçersizdir.

2-) Bir pointerdan bir pointer çıkarılabilir. Sonuç tam sayıdır.

3-) Bir pointerla bir tam sayı toplanabilir. Sonuç pointer türünden çıkar. Bunu aslında biz bunu dizilerde sıkça kullandık.

4-) Bir pointerdan bir tam sayı çıkarılabilir. Sonuç pointer türünden çıkar.

5-) Bir tamsayıdan bir pointer çıkarılması geçersizdir.

Bu işlemler dizilerde sıkça kullanılır. Mesela aranan bir eleman bir elemanın kaçıncı eleman olduğu iki tane pointerın birbirinden çıkarılmasıyla elde edilebilir. Örneğin:

#include <stdio.h>

#include <stdlib.h>

#define randomize() srand(time(NULL))

#define SIZE 20

void setRandomArray(int arr[],int size)

{

while(size--)

{

\*arr++ = rand() % 100;

}

}

void printArray(int arr[],int size)

{

while(size--)

{

printf("%d ",\*arr++);

}

printf("\n");

}

int\* indexOf(int\* arr,int size,int searchingElement)

{

while(size--)

{

if(\*arr == searchingElement)

return arr;

arr++;

}

return NULL;

}

int main(void)

{

randomize();

int arr[SIZE];

setRandomArray(arr,SIZE);

printArray(arr,SIZE);

int x;

printf("Aranan Eleman?\n");

scanf("%d",&x);

int\* index = indexOf(arr,SIZE,x);

printf("idx : %lld",index - arr);

}

Pointerlarla ilgili detaylar ilerde ele alınacaktır.