**Fonksiyon Göstericileri (Function Pointers)**

Fonksiyonlar şimdiye çok sık kullandığımız bizim bazı işleri gerçekleştirdiğimiz ve programın parçalara ayrılmasından oldukça önemli bir yere sahip olan bir iş yapan alt programlardır. Şimdiye kadar temel türlere sahip nesneleri gösteren ve her türden nesneyi gösterebilen göstericilerle tanıştık. Ancak bu dokümanla birlikte fonksiyonları gösteren göstericilerle de tanışacağız. Fonksiyon göstericilerinin genel biçimi:

<geri dönüş değer türü>(<asteriks atomu> <fonksiyon göstericisi ismi>) (<parametre değişkenleri>)

Bir fonksiyon göstericine illaki bir fonksiyon adresi atanmasına gerek yoktur. Ancak unutulmamalıdır ki fonksiyon adresi atanmasa bile bu zaten bir fonksiyon adresine dönüştürülür. Örneğin aşağıdaki örnekte fptr nesnesi foo fonksiyonun gösteren bir fonksiyon göstericidir.

void foo(void);

int main(void)

{

void (\*fptr)(void) = foo;//\*\*

return 0;

}

Buradaki durumda \*\* ile gösterilen ifadede foo doğrudan bir fonksiyon adresine dönüştürülür. Yani burda address of operatörünü kullanmak ile doğrudan yazmak arasında bir fark yoktur. Unutulmaması gereken diğer bir kısım ise burada fptr değişkeninin türüdür. Burada fptr nesnesinin türü void (\*) (void) türüdür.

Fonksiyon göstericileri aslında normalde göstericilerin sahip oldukları tüm özelliklere sahiptir. Örneğin aşağıdaki örnekte fptr fonksiyon göstericisi ilk durumda foo fonksiyonunu gösterirken ikinci durumda func fonksiyonunu göstermektedir.

void foo(void);

void func(void);

int main(void)

{

void (\*fptr)(void) = foo; //\*

fptr = func;//\*\*

return 0;

}

Burada \* ile belirtilen ifadeden \*\* ifadesine kadar fptr fonksiyonu foo fonksiyonunu \*\* ifadesinden sonra ise func fonksiyonunu göstermektedir.

Anahtar Notlar: Fonksiyon göstericileri ancak belli türdeki fonksiyonları gösterebilir. Örneğin fonksiyon göstericisinin türü void (\*) (void) ise bu göstericiye ancak void (void) türünden fonksiyon adresleri atanabilir. Bu bir sentaks hatası olmasa da bu durum tanımsız davranışa sebep olur.

Peki bu fonksiyon göstericileri kullanılarak fonksiyon çağrısı nasıl yapılabilir?

Fonksiyon göstericileri kullanılarak doğrudan bir fonksiyon çağırıyormuş gibi fonksiyon göstericileri kullanılabilir. Örneğin:

#include <stdio.h>

void foo(void)

{

printf("foo called\n");

}

void func(void)

{

printf("func called\n");

}

int main(void)

{

void (\*fptr)(void) = foo; //\*

fptr();

fptr = func;

fptr();

return 0;

}

Bu örnekte tek bir pointer nesnesi kullanılarak birden fazla fonksiyon çağrısı yapılabilir.

Aşağıdaki örnekte func fonksiyonun adresinin fp fonksiyon göstericisine atanması sonucu fp func fonksiyonunu çağırabilir.

#include <stdio.h>

void func(int\*x)

{

\*x=(\*x)\*(\*x);

}

int main()

{

int x=10;

void (\*fp)(int\*);

fp=&func;

fp(&x);

printf("%d",x);

}

Aşağıdaki örnekte foo fonksiyonunun hangi fonksiyonu çağıracağına onu çağıran fonksiyon karar vermektedir çünkü fonksiyonun argümandan parametreye aktarımı söz konusudur.

#include <stdio.h>

void f1(void)

{

printf("f1 cagrildi.");

}

void f2(void)

{

printf("f2 cagrildi.");

}

void f3(void)

{

printf("f3 cagrildi.");

}

void foo(void(\*fp) (void))

{

printf("foo!\n");

fp();

}

int main() {

foo(&f2);

}

Burada fonksiyon hangi fonksiyon adresiyle çağrılırsa ona bağlı olarak o fonksiyon çağrılacaktır.

Aşağıdaki örnekte fp'nin 3 defa çağrılması ilk görünüşte çok mantıklı gelmese de her çağrıda farklı bir fonksiyon çağrısı söz konusudur. Bu da fonksiyon göstericilerinin adres değişikliği ile göstericinin birden çok fonksiyonu aynı sentaks ile çağırmasına olanak sağlamaktadır.

#include <stdio.h>

typedef void (\*FPTR) (void);

void f1(void){

printf("f1 cagrildi.\n");

}

void f2(void){

printf("f2 cagrildi.\n");

}

void f3(void){

printf("f3 cagrildi.\n");

}

int main() {

FPTR fp=&f1;

fp();

fp=&f2;

fp();

fp=&f3;

fp();

}

**Standart Kütüphanenin qsort() Fonksiyonu**

Standart kütüphanelerden stdlib.h da bulunan qsort fonksiyonu dizileri tür bağımsız(generic) şekilde gönderilen fonksiyon adresine bağlı sıralayan bir fonksiyondur. Aslında qsort Quick Sort algoritmasından gelmektedir ve bu algoritma O n log n karmaşıklığına sahip olduğundan dolayı Bubble sort(O n^2 karmaşıklığı) algoritmasına göre daha verimli çalışmaktadır. Standart qsort fonksiyonu 4 adet parametre değişkenine sahip bir standart fonksiyondur. qsort algoritmasına gönderilmesi istenen argümanlar:

qsort(<dizinin ilk elemanının adresi veya kendisi>,<dizinin uzunluğu>,<dizinin bir elemanının size of u>,<dizinin sıralama kriterini belirleyen fonksiyonun adresi>)

Aşağıdaki örnekte int türden değerlere sahip bir dizinin büyükten küçüğe sıralanması için yapılması gereken qsort çağrısı gösterilmiştir. Örnek rastgele bir dizi üreten bir fonksiyon ile test edilmiştir.

#include <time.h>

#include <stdio.h>

#include <stdlib.h>

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

void displayArray(const int \*a, int size) {

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

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

}

void setRandomArray(int \*a, int size) {

randomize();

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

a[i] = rand() % 100 + 1;

}

int cmp(void\*vp1,void\*vp2){

int\*p1=vp1;

int\*p2=vp2;

if(\*p1>\*p2)

return 1;

if (\*p1<\*p2)

return -1;

return 0;

}

int main() {

int size;

printf("Kac elemanli bir dizi yaratilsin?");

scanf("%d",&size);

int\*arr=(int\*)malloc(size\* sizeof(int));

setRandomArray(arr,size);

displayArray(arr,size);

printf("\n");

qsort(arr, size, sizeof(int), (int (\*)(const void \*, const void \*)) &cmp);

displayArray(arr,size);

free(arr);

}

Soru: sıralanmamış olan ve her elemanı const char\* türünden olan bir diziyi sıralayan qsort çağrısını yapınız.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

const char \*names[] = {"Mehmet", "Necati", "Berke", "Murat",

"Burak", "Harun", "Engin", "Batuhan", "Ahmet", "Ali", "Mustafa",

"Abdullah", "Barbaros", "Berk", "Berkay","Leyla", "Ece", "Zeynep",

"Berrak","Gamze", "Gizem", "Nergiz","Derya", "Jale", "Kezban",

"Fadime", "Selma", "Berna", "Ender", "Sevil"};

void displayStringArray(const char\* sArr[],int size)

{

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

{

printf("%s ",sArr[i]);

}

}

int compareStrings(char\*\*s1,char\*\*s2)

{

return strcmp(\*s1,\*s2);

}

int main()

{

displayStringArray(names, sizeof(names)/sizeof(names[0]));

qsort(names,sizeof(names)/sizeof(names[0]),sizeof(names[0]),&compareStrings);

displayStringArray(names, sizeof(names)/sizeof(names[0]));

}

Normal göstericilerimizin ve void göstericilerimizin dizileri olabileceği gibi fonksiyon gösterici dizileri de olabilir. Hatta bu sentaks üretimde sıkça kullanılan bir sentakstır. Ancak burada normal gösterici dizilerinden biraz daha farklı bir şekilde tanımlanırlar. Fonksiyon gösterici dizilerinin genel biçimi aşağıdaki gibidir:

<geri dönüş değer türü>(<asteriks atomu><değişken ismi>[<eleman sayısı>])(<parametre değişkenleri>);

Mesela Aşağıdaki örnekteki gibi bir fonksiyon gösterici dizisi oluşturulabilir.

char\*f1(const char\*,const char\*);

char\*f2(const char\*,const char\*);

char\*f3(const char\*,const char\*);

int main() {

char\*(\*fp[3])(const char \*,const char \*)={&f1,&f2,&f3};

}

Anahtar Notlar: Bir fonksiyon dizisinin herhangi bir elemanının dizinin geri dönüş değer türünden farklı olması anlamlı değildir. Bu sentaks hatası olmasa dahi asla yazılmaması gereken bir koddur çünkü burda bir tanımsız davranış(undefined behaviour) oluşur. Aynı durum parametre değişkenlerinin farklı olması veya sayısının fazla olması durumunda da geçerlidir. Örneğin:

char\*f1(const char\*,const char\*);

char\*f2(const char\*,const char\*);

double\*f3(const char\*,const char\*);

int main() {

char\*(\*fp[3])(const char \*,const char \*)={&f1,&f2,&f3}; //ub

}

Anahtar Notlar: Fonksiyon göstericilerinin türleri oldukça uzun türler olduğundan dolayı genellikle typedef bildirimi kullanılarak tanımlanırlar. Aşağıdaki kod da typedef bildirimi ile fonksiyon gösterici dizilerin nasıl kolay yoldan tanımlanabileceğini gösteren önemli bir örnektir. typedef konusunda da belirttiğimiz gibi typedef bildirimleri fonksiyon göstericilerinde ve fonksiyon gösterici dizilerinde çok önemli bir yere sahiptir. Örneğin:

typedef char\* (\*FPTR) (const char\*,const char\*);

char\*f1(const char\*,const char\*);

char\*f2(const char\*,const char\*);

char\*f3(const char\*,const char\*);

int main()

{

FPTR fp[3]={&f1,&f2,&f3};

FPTR fp1=f1;

}

ctype standart kütüphanesinin bazı fonksiyonların getchar fonksiyonu ile alınan bir karaktere göre for döngüsü ile fonksiyon gösterici dizisini dolaşmasını anlatan güzel bir örnek:

#include <stdio.h>

#include <ctype.h>

#define asize(a) sizeof(a)/sizeof(a[0])

typedef int (\*FPTR) (int);

int main() {

FPTR fp[]={&islower,&isupper,&isalpha,&isdigit,&isalnum,&isxdigit,

&isspace,&isblank,&iscntrl,&isgraph,&ispunct};

const char \*p[]={"islower","isupper","isalpha","isdigit","isalnum","isxdigit",

"isspace","isblank","iscntrl","isgraph","ispunct"};

int ch;

printf("Bir karakter giriniz?");

ch=getchar();

for (int i=0;i<asize(fp);++i){

if(fp[i](ch))

printf("%s test icin Ok ",p[i]);

else

printf("%s testi icin not Ok ",p[i]);

}

}

Fonksiyon göstericileriyle ilgili detaylar ileride ele alınacaktır.