**Void Göstericiler(Void Pointers)**

Void göstericiler pointerların türden bağımsız olanıdır. Bilinmesi gereken ilk şey C’ de void diye bir tür olmadığıdır. Örneğin aşağıdaki gibi bir tanımlama yapılması sentaks hatasına neden olur.

#include <stdio.h>

int main(void)

{

void a;//error

return 0;

}

Ancak bizim bu kısımda öğreneceğimiz void türü değil, void\* türüdür. Zaten teknik olarak void bir tür de değildir. Bunu bir fonksiyonun geri dönüş değeri olmaması durumunda fonksiyonun geri dönüş değer türü kısmına void yazılması ile açıklanabilir aslında. Önceden hatırlayabileceğimiz gibi bu durumda fonksiyon geri dönüş değerine sahip olmamaktaydı. Ancak void\* türü ilerde de ele alacağımız dinamik bellek yönetiminde kullanılan malloc, calloc ve realloc gibi fonksiyonların da geri dönüş değeridir.

Dikkat edilmesi gereken bir kısım void göstericilerin sizeof değerleridir. Bu aynı diğer göstericiler gibi derleyiciye bağlı olarak değişmesinin yanı sıra 64 bitlik derleyicilerde 8 byte uzunluğa sahiptir. Örneğin:

#include <stdio.h>

int main(void)

{

printf("%zu", sizeof(void\*)); //8

return 0;

}

Anahtar Notlar: void göstericiler içerik operatörünün operandı yapılamaz. Çünkü bu durumda bu ifade bir incomplete type olarak değerlendirilecektir. Örneğin aşağıdaki örnekte bu durumdan kaynaklı bir sentaks hatası oluşucaktır.

#include <stdio.h>

int main(void)

{

int val = 10;

void\* vp1 = &val;

printf("%d",\*vp1); //error

return 0;

}

void göstericiler aslında genellikle diğer göstericilere aracılık etmek için kullanılır. Çünkü herhangi türden bir değişkenin adresi void göstericilere atanabilir. Örneğin aşağıdaki örnekteki kod legaldir.

int main(void)

{

int val = 10;

double dVal = 12.56;

long lVal = 1891990;

void\* vp1 = &val; //legal

vp1 = &dVal; //legal

vp1 = &lVal; //legal

return 0;

}

İşte void göstericilerin bu özelliği aslında void\* türünden herhangi bir pointerının tür dönüştürme operatörüne bile gerek duyulmadan herhangi başka türden göstericiye atanmasına olanak sağlar. Örneğin:

int main(void)

{

int val = 10;

void\* vp1 = &val; //legal

int\* ptr = vp1; //legal

return 0;

}

Dikkat edilmesi gereken bir kısım ise void\* türünden burada olduğu gibi int\* türüne doğrudan C++ dilinde legal değildir. Ancak C dili için burda bir warning dahi oluşmaz, oldukça normal bir durumdur. Ancak şüphesiz tür dönüştürme operatörünün kullanımı da saçma değildir.

Anahtar Notlar: void göstericilerde pointer aritmetiği kullanılamaz.

Bildiğimiz üzere göstericiler konusunda göstericilerin en önemli özelliklerinden biri olan pointer aritmetiğinden bahsetmiştik. Bu özellik dizilerin bellekte ardışıl yer almasından kaynaklanan ve diziyi dolaşmak için oldukça efektif bir yöntemdi. Ancak void göstericiler için böyle bir durum söz konusu değil. Daha önceden de belirttiğimiz gibi pointer ++ operatörünün operandı yapıldıklarında değerleri gösterdikleri değişkenin türünün sizeof değeri kadar artmaktadır. Örneğin bu int türü için 4 byte olacaktır. (64 bitlik derleyiciler için) Ancak void göstericilerde gösterilen değişkenin sizeof değeri belli olmadığından dolayı ne kadar arttırılacağı tam olarak kestirilemez. Aslında bu nedenle pointer aritmetiğin void göstericiler için çalışmadığını söyleyebiliriz.

**Void Göstericiler Ne İşe Yarıyor?**

Bu pointerlar aslında çoğu programcının düşündüğü önemsiz araçlar değil. Özellikle C++ dilinde jenerik programlamanın(Türden Bağımsız Programlama) bu kadar ileriye gitmesinin temelini oluşturduğunu da söyleyebiliriz. Bu göstericiler kullanılarak türden bağımsız bir şekilde kod yazılabilmektedir. Örneğin daha önceden yazmış olduğumuz takas fonksiyonunu jenerik hale getirebiliriz.

Soru: Parametresiyle aldığı iki değişkeni türden bağımsız olarak takas eden gSwap isimli fonksiyonu yazınız ve hem bir gerçek sayı türüyle hem de bir tamsayı türüyle test ediniz.

#include <stdio.h>

void gSwap(void\* vp1,void\*vp2,int size)

{

char\* cp1 = vp1;

char\* cp2 = vp2;

while (size--)

{

char temp = \*cp1;

\*cp1 = \*cp2;

\*cp2 = temp;

++cp1;

++cp2;

}

}

void gSwapTest(void)

{

int iVal1 = 12;

int iVal2 = 78;

double dVal1 = 23.45;

double dVal2 = 9.81;

printf("iVal1 = %d\niVal2 = %d\n",iVal1,iVal2);

printf("dVal1 = %lf\ndVal2 = %lf\n",dVal1,dVal2);

gSwap(&iVal1,&iVal2, sizeof(iVal1));

gSwap(&dVal1,&dVal2, sizeof(dVal1));

printf("iVal1 = %d\niVal2 = %d\n",iVal1,iVal2);

printf("dVal1 = %lf\ndVal2 = %lf\n",dVal1,dVal2);

}

int main(void)

{

gSwapTest();

return 0;

}

Bu örnekte void pointerın char pointera dönüştürülmesinin sebebi char türünün sizeof değerinin 1 olmasından kaynaklanmaktadır. Bu sayede tutulan elemanların değerleri sırasıyla 1 byte 1 byte değiştirilecektir. Ta ki size değeri bitene kadar işte bu sebepten de size kadar dönen bir döngüye ihtiyacımız vardı.