**Tür Belirleyiciler ve Yer Belirleyiciler(Type qualifiers and Storage class specifiers)**

1-) Tür Belirleyiciler

C programlama dilinde 5 tane tür belirleyici anahtar sözcük bulunmaktadır. Bunlardan ikisi kullanımdan oldukça düşmüş anahtar sözcüklerdir. Bu kullanımdan düşmüş olanlardan bir tanesi C++ dilinde bambaşka bir anlama gelmekte olup orda sıkça kullanılmaktadır.

C programlama dilinde bulunan tür belirleyiciler:

a-) auto: Bu anahtar sözcük ile C++ dilinde bulunan auto anahtar sözcüğüyle karıştırılmamalıdır. C++’ta auto anahtar sözcüğü sıkça kullanılırken C’deki auto C’de neredeyse hiç kullanılmamaktadır. Bu anahtar sözcük kısaca bir değişkenin otomatik ömürlü olduğunu göstermek için kullanılır. Bunun sebebi yerel değişkenlerin defaultta zaten otomatik ömürlü olması global değişkenlerin otomatik ömürlü olamayışı ve parametre değişkenlerinin otomatik ömürlü olmaktan başka seçeneğinin bulunmayışı gibi durumlar sebep gösterilebilir.

Anahtar Notlar: Global değişkenler auto anahtar sözcüğü ile kullanılamaz bunun neden global değişkenlerin static ömürlü olmasından kaynaklanmaktadır. Örneğin aşağıdaki örnekte bundan kaynaklı bir sentaks hatası oluşur.

#include <stdio.h>

auto int a=10;//error

int main(void)

{

//code

}

Anahtar Notlar: auto anahtar sözcüğü C++ dilinde C dilinin yanı sıra C++11 güncellemesiyle birlikte bu anahtar sözcüğe artık farklı anlamlar yüklenmeye başladı. Bu anahtar sözcük artık auto type deduction olarak adlandırılmaya başladı. Kısaca C’deki auto ile C++ dilindeki auto birbirinden son derece farklıdır.

C’deki auto anahtar sözcüğünün kullanımı:

int main(void)

{

auto int b=20;

return 0;

}

b-) register: register anahtar sözcüğü C'de gene auto gibi çok kullanımdan düşmüş bir anahtar sözcüktür. Ancak bazı eski sistemlerde kod yazıyorsanız gene kullanılabilir ancak günümüz derleyicileri oldukça gelişmiş olduğundan bu anahtar sözcüktür. Bu anahtar sözcüğün görevi kısaca kodu optimize et talimatı derleyiciye vermektir. Ancak günümüz derleyicileri oldukça gelişmiş olduğundan dolayı artık derleyiciler buna kullanıcının karar veremeyecek durumda olduğunu söyler kullanımdan düşmesinin asıl nedeni de aslında budur. Örnek bir register anahtar sözcüğü kullanımı aşağıdaki gibidir.

#include <stdio.h>

int main()

{

register char x = 'S';

register int a = 10;

auto int b = 8;

printf("The value of register variable b : %c\n",x);

printf("The sum of auto and register variable : %d",(a+b));

return 0;

}

c-) extern: extern kod yazarken sıkça kullanılan bir anahtar sözcüktür. Bu anahtar sözcük aslında bize bir dosyada bir yerlerle örneğin x diye bir değişken var o değişkeni bul ve o dosyada kullanılır hale getir demektir. Eğer o değişken bulunamazsa elbette kullanılamaz.

Anahtar Notlar: Bir değişkenin extern ile yazılması bir bildirim gibi düşünülmemelidir. Bunun en büyük kanıtı extern ile tanımlanan bir değişkene ilk değer verilememesidir. Burada aslında programda örneğin bir x değişkeni olduğu söylenmektedir. Şüphesiz bu değişkenin bulunması kadar bulunmaması da doğal bir durumdur. Ancak bulunamaması durumuna karşı dikkatli olunca gerektiği de muhakkaktır. İlerde ele alacağımız static anahtar sözcüğü kullanılarak değişkenler başka dosyalardan erişime kapatılabilirler.

d-) typedef: C dilinde typedef anahtar sözcüğü var olan bir veri türünü veya veri yapısını farklı bir isimle yeniden adlandırılmanızı sağlar. Bu özellik, kodunuzu daha okunaklı hale getirmenize ve karmaiık veri yapıları için daha anlaşılır isimler kullanmanıza yardımcı olmaktadır. Kısaca kodun okunabilirliği ve anlaşılabilirliğini oldukça yükseltiği senaryolar mevcuttur. Ancak en çok kullanıldığı alan özellikle kullanıcının oluşturduğu türlerle birlikte fonksiyon göstericileridir. Ancak biz bu doküman içerisinde bunun dışındaki kullanımlarına değineceğiniz. Örneğin en basit şekilde aşağıdaki gibi bir tanımlama yapılabilir.

typedef int Tamsayi;

typedef float Ondalik;

Bu ifade aslında kısaca şu anlama gelmektedir. Bundan sonra Tamsayi gördüğün yer aslında int demek anlamına gelmektedir. Kısaca Tamsayi demek int demek diyebiliriz. Örneğin aşağıdaki gibi bir tanımlama geçerlidir.

Tamsayi x = 10;

Ondalik y = 3.14;

Typedef bildirimlerinin kullanıcının oluşturduğu türler ile birlikte kullanımı ilerde detaylı bir şekilde ele alınacaktır. Bu sentaks özellikle üretimde sıklıkla kullanıldığından dolayı oldukça önemli bir yere sahiptir.

e-) static: static anahtar sözcüğü ile bildirilen değişkenler fonksiyon içerisinde bildirilseler dahi fonksiyon bloğu bittiği zaman yok edilmezler. Ancak çalışma zamanında yok edilirler. Aşağıdaki örnekte yapılan 3 foo çağrısından sonra görüldüğü gibi x otomatik ömürlü bir nesne olduğundan değeri her foo çağrısında sabit kalacaktır. Diğer taraftan y değişkeni static anahtar sözcüğü ile bildirildiği için her foo çağrısında bu değişken artmaya devam edicektir.

#include <stdio.h>

void foo(void)

{

int x = 10;

static int y = 10;

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

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

}

int main()

{

foo();

foo();

foo();

}

Anahtar Notlar: static anahtar sözcüğü ile bildirim global değişkenler için de yapılabilir. Ancak static global değişkenler ile normal global değişkenler arasında çok önemli bir fark vardır. Ancak bunu anlamak için öncelikle C’de linkage yani bağlam kavramını anlamak gerekir.

C’de 3 farklı linkage türü vardır. Bunlar:

1-) External Linkage(Dışsal Bağlama): Dışsal bağlama kavramı değişkenler için kullanılan bir terimdir. Çünkü C’de fonksiyonların herhangi bir şekilde bağlam kavramına sahip olmaları söz konusu değildir. Dışsal bağlama adından da anlaşılabileceği gibi değişkenlere başka dosyadan da erişebileceği anlamına gelir. Normal global değişkenler static global değişkenlerin aksine dışsal bağlamaya sahiptir.

Örneğin aşağıdaki örnekte iki tane dosya gösterilmiştir. Bu dosyalardan nutility.h başlık dosyası içerisinde bildirilen x değişkeni extern bildirimi kullanılarak başka dosyalarda da kullanılabilir. Ancak bu durum sadece normal global değişkenler için geçerlidir.

//main.c

#include “nutility.h”

extern int x;

int main(void)

{

int y = x;

}

//nutility.h

int x = 10;

//code

Bu örnekte x değişkeni doğrudan extern bildirimi yardımıyla kullanılmaktadır. Unutulmamalıdır ki buradaki extern bildirimi normal bir değişken bildirimi gibi değildir. Bu aslında bir yerlerde bir x değişkeni var ve ben o x değişkenini kullanmak istiyorum anlamına gelen bir bildirimdir. Yani bunu normal bir değişken bildirimiyle karıştırmamak gerekir.

2-) Internal Linkage(İçsel Bağlama): İçsel bağlama değişkenlere sadece programcının çalıştığı dosyadan erişim hakkının bulunduğunu göstermektedir. C’de static global değişkenler internal linkage özelliği gösterirler. Aslında static global ile normal global değişkenler arası tek ve en önemli fark budur.

Örneğin aşağıdaki örnekte nutility.h başlık dosyasında bulunan x değişkeni bizim main.c kaynak dosyamızda kullanılamaz. Çünkü burada x değişkeni içsel bağlamaya sahip bir değişkendir.

//main.c

#include “nutility.h”

extern int x;

int main(void)

{

int y = x;

}

//nutility.h

static int x = 10;

//code

3-) No-Linkage(Bağlamasız): C’de herhangi bağlamaya sahip olmayan değişkenler Bağlamasız değişkenlerdir. Bu değişkenler parametre değişkenleri ve yerel değişkenlerdir. Bu değişkenler daha önce belirtmiş olduğumuz diğer kategorilerden ayrı tutulmalıdır.

Örneğin aşağıdaki örnekte kullanılan yerel değişkenler(local variables) ve parametre değişkenleri(parameter variables) buna bir örnektir.

#include <stdio.h>

İnt add(int x,int y)

{

return x + y;

}

int main(void)

{

int x = 10,y = 34;

printf(“sum : %d\n”,add(x,y));

}

2-) Yer Belirleyiciler

a-) const: const anahtar sözcüğü C dilinde oldukça sık kullanılan bir anahtar sözcüktür. Bu anahtar sözcük ile bildirilen değişkenler değiştirilemez. Ancak pointer değişkenlerde constluk durumları üretimde sık kullanıldığı için doğru bilinmesi oldukça önemlidir. Fonksiyonların parametre değişkenleri de yerel değişkenler de const anahtar sözcüğü ile kullanılabilir. Örneğin aşağıdaki örnekte x değişkenini değiştirme girişimi sentaks hatasına neden olur. Diğer taraftan const anahtar sözcüğü kullanılarak bildirilen değişkenler ancak salt okuma amaçlı kullanılabilirler.

#include <stdio.h>

int square(const int x)

{

return x \* x;

}

İnt main(void)

{

int x;

printf(“Bir sayı giriniz?\n”);

scanf(“%d”, &x);

printf(“square of %d is %d\n”,x, square(x))

}

Elbette programcıların bazıları const anahtar sözcüğünün gereksiz olduğunu düşünmektedir. Oysa bu anahtar sözcük üretimde en çok kullanılan anahtar sözcüklerden biridir. Bir değişken eğer herhangi bir şekilde değiştirilmiyorsa bu durumda const yapılması oldukça önemlidir.

Göstericilerle de const anahtar sözcüğü birlikte kullanılabilir. Ancak const anahtar sözcüğünün göstericilerle birlikte kullanımı biraz daha farklıdır. Örneğin aşağıdaki iki satırdaki bildirimler pek çok programcının düşündüğü gibi aynı şeyi ifade etmez.

const int\* p1;

int \* const p2;

Bu ifadelerin isimleri bile birbirinden farklıdır. p1 değişkeni low level const pointer olarak isimlendirilirken p2 değişkeni top level const olarak isimlendirilir. Peki bunlar arasında ne fark vardır? Bunların arasındaki en önemli fark ilk ifadede p1 değişkeninin değil \*p1’in const olmasından bahsedilebilir. Ancak p2 değişkeni için durum tam tersidir. Sonuç olarak aşağıdaki örnekte aralarında fark net bir şekilde gösterilmektedir.

int main(void)

{

int x = 15;

int y = 34;

const int\* p1 = &x;

int \* const p2 = &y;

p1 = &y; //legal

\*p1 = 56; //error

p2 = &x; //error

\*p2 = 78; //legal

}

Ancak şüphesiz bir göstericinin tüm özellikleri de const yapılabilir. Ancak bunun için iki defa const anahtar sözcüğü kullanılması gerekmektedir. Örneğin aşağıdaki örnekte hep p1 const hem de \*p1 const olacaktır.

int main(void)

{

int x = 15;

int y = 34;

const int\* const p1 = &x;

p1 = &y; //error

\*p1 = 56; //error

}

İkisini birden const yapmanın da başka bir yolu olmadığını unutmamak gerekir.

const anahtar sözcüğü değişkenlerin sadece salt okuma amaçlı kullanımları durumunda değişkenlerin const anahtar sözcüğüyle bildirilmesi gerekmektedir. Bu değişkenlerin bildirildiği gibi göstericiler de daha önceden gösterdiğimiz gibi const olmaktadır. Örneğin aşağıdaki örnekteki fonksiyonda arr göstericisi yalnızca ekrana basma yapacağından ve sadece salt okuma amaçlı bir kullanım olacağında burda int\* değil const int\* türünden bildirilmiştir.

#include <stdio.h>

void printArray(const int\* arr, int size)

{

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

{

printf(“%d ”,\*(arr + i));

}

printf(“\n”);

}

b-) volatile: Bu anahtar sözcük C ve C++ gibi dillerde sıkça kullanılan bir anahtar sözcüktür. Genellikle bellek-mapped donanım veya kesmeler(interrupt) gibi kesinti tabanlı olaylarla ilişkilendirilen değişkenlerin bellek yönetimi ve derleyici optimizasyonlarıyla ilgili davranışını kontrol etmek için kullanılır. volatile anahtar sözcüğü, derleyicinin değişkenin değerini veya bellek yerini önceden tahmin etmesini engeller ve her zaman bellekten gerçek değeri okur veya yazmaya zorlar. Kısaca o değişkenin değeri için derleyicinin geçici bellek(register bellek) kullanma anlamına gelmektedir. Bu değişkenlerin değerleri doğrudan bilgisayarın RAM’inden alınır.

Örneğin aşağıdaki örnekte bir interrupt olduğunu kontrol eden bir while döngüsü vardır. Buradaki interrupt kontrol değişkeninin volatile anahtar sözcüğüyle bildirilmesi gerekmektedir. Bu sayede derleyici, kesme oluştuğunda bu değişkenin her zaman güncel olduğunu bilir.

volatile int kesmeGeldi = 0; // Kesme işareti geldiğinde değişkeni güncelle

void kesmeIsr() {

kesmeGeldi = 1;

}

int main() {

while (!kesmeGeldi) {

// Kesme işareti gelene kadar bekleyin.

}

// İşleme devam edin.

}

Bir başka volatile anahtar sözcüğü kullanımı da donanım erişimi ile ilgili bir senaryoda da kullanılabilir. Örneğin:

volatile unsigned int \*donanımReg = (volatile unsigned int \*)0x40000000; // Donanım registerına erişim

void yazDonanimReg(unsigned int deger) {

\*donanımReg = deger;

}

unsigned int okuDonanimReg() {

return \*donanımReg;

}

Bu örnekte “donanımReg” bir bellek-mapped donanım registerına işaret eder ve bu register “volatile” olarak işaretlenir. Bu, derleyicinin bu registgerın değerini her zaman bellekten okumasınını veya yazmasını gerektirir, böylece derleyici beklenmedik optimize edilmiş bir kod üretilmez.

Volatile anahtar sözcüğü, derleyeicinin belirli durumlarda optimize etmesini engeller ve programın kesmeler(interrupts), donanım erişimi veya diğer dışsal etkenlerle doğru şekilde çalışmasını sağlar. Ancak, gerektiği gibi kullanılmalıdır ve her tür değişken volatile anahtar sözcüğü ile birlikte kullanılmamalıdır. Aksi takdirde performan kaybına neden olabilir.

Kısaca tür ve yer belirleyici anahtar sözcükler gerektiği yerde kullanılmalıdır. Gereksiz yere kullanımları anlamlı değildir. register ve auto gibi anahtar sözcükler kullanımdan düşmüştür ancak bu kategorideki diğer anahtar sözcükler C dilinde sıkça kullanılmaktadır.