C VERİ TÜRLERİ

BASİT VERİ TÜRLERİ

Değişkenler, program içinde üzerinde işlem yapılan, veri saklanan ve durumlarına göre programın akışı sağlanan nesnelerdir. C'de bir değişken kullanılmadan önce tanımlanmalıdır. Değişkenler alfabe harfleri (letter) veya alt çizgi (_) ile başlar.

Değişkenler tanımlanırken onlara ne tür veriler atanabileceği de belirtilir. Temel veri türleri oldukça azdır:

void: Belirsiz tür.

char: Tek bir byte. Bir karakter taşıyabilir.

int: Tamsayı. İşlemci mimarisine göre boyu değişir.

float: Kayan noktalı gerçek sayı.

double: Çift duyarlılıkta kayan noktalı gerçek sayı.

C ile çıkış, formatlı olarak üretilir. % işareti ile format belirtilir. Her türün format belirticisi vardır.

%c Karakter basmak için

%d Tamsayı basmak için

%f Gerçek sayı basmak için

```
printf("%c %d %f", letter, number, avarage);
```

TÜR NİTELEYİCİLER

Türlere değişik özellikler kazandırmak için niteleyiciler kullanılır.

short ve long

Bu iki niteleyici, int veri türünün uzunluğunu belirlemek için kullanılır. Tamsayı veri türü için işlemci tarafından sınırlanan bir veri uzunluğu vardır. 'short' ve 'long' niteleyicilerinin bu sınır içerisinde uzunluğu ne kadar yapacağı derleyiciye bağlıdır.

32-bit bir işlemcide short genellikle 16 bit, long ise 32 bit uzunlukta tamsayı belirmektedir. int veri türü derleyiciye göre 16 bit veya 32 seçilebilir. Ancak uzunluklar arasında aşağıdaki sıralama mevcuttur:

short < int < long

```
#include <stdio.h>

int main() {
        int i;

        printf("Integer: %d byte, short: %d byte.\n",
           sizeof(int), sizeof(short int));

        return 0;
}
```

```
$ ./tamsayi
Integer: 4 byte, short: 2 byte.
```

signed ve unsigned

char ve int veri türleri üzerinde uygulanabilir. Verinin işaretli veya işaretsiz olduğunu belirler. ‘signed’ bir değişken 0 veya pozitiftir. Bu durumda n bit bir değişken 0...2^n – 1 arasında bir değere sahip olabilir. ‘unsigned’ bir veri türü negatif değere sahip olabilir.

Örneğin 8 bit olan char veri türü ‘unsigned’ olarak 0..255, ‘signed’ olarak –128...127 arasında değerlere sahip olabilir.

long

double veri türüne uygulanır. Daha duyarlı bir gerçek sayı elde etmek için kullanılır.

float < double < long double

volatile

Bu veri türü niteleyicisi gömülü sistemlerde gerekmektedir. Aşağıdaki kod parçasını inceleyelim:

unsigned short CTL_Data_Ready;

while ((rx->ctrl & CTL_Data_Ready) == 0) ;

Akıllı bir derleyici while döngüsü içerisinde CTL_Data_Ready değişkeninin hiçbir zaman değişmeyeceğini görüp optimizasyon yapar. Mevcut değerini bir kere hesap eder ve döngüye girmez. Normalde bu mantıklı bir yaklaşımdır. Ancak yukarıdaki kod seri porttan okumak için yazılmıştır. Bu döngü, verinin hazır olması için burada bekler. Hazır olduğunda ise CTL_Data_Ready değişkeni 1 olacak ve program döngüden kurtulacaktır. CTL_Data_Ready değişkeni programdan bağımsız ve asenkron olarak seri portta veri olup olmamasına bağlı olarak değişir. Dolayısıyla derleyicinin yapacağı iyileştirme (optimizasyon) aslında programın yanlış çalışmasına sebep olacaktır.

Bu problemi ortadan kaldırmak için volatile kullanılır. Volatile anahtar kelimesi, derleyiciye burada optimizasyon yapmaması gerektiğini söyler.

```
unsigned short volatile CTL_Data_Ready;
```

Sabit Değişken, const

Bir değişkenin program boyunca değiştirilemez olmasını sağlar.

const float PI = 3.14;

Sabitler daha sonra değiştirilemediği için tanımlanırken ilk değer ataması yapılmalıdır.

Aşağıdaki tanımlamada cp değiştirilemez bir işaretçidir.

```
char c;
char *const cp = &c;
```

Aşağıdaki tanım ise normal bir işaretçi tanımlar ancak gösterdiği karakter değiştirilemez.

```
const char *cp;
```

0 ve 1 dışındaki rakamları sabit olarak tanımlayarak rakam yerine isim kullanmak daha okunaklı kod olmasını sağlayacaktır.

Sabiti program içerisinde değiştirmeye kalkışmak derleme aşamasındaki aşağıdaki gibi bir hata ile karşılaşılmasına neden olur:

```
#include <stdio.h>

int main() {
        const int THRESHOLD = 10;

        THRESHOLD = 20;

        printf("Threshold: %d\n", THRESHOLD);

        return 0;
}
```

```
$ cc sabit.c  -o sabit
sabit.c: In function 'main':
sabit.c:6: error: assignment of read-only variable 'THRESHOLD'
*** Error code 1
```

GELİŞMİŞ VERİ TÜRLERİ

Burada bahsedilecek veri türleri temel veri türlerinden oluşturulur.

Yapılar

Yapılar, daha gelişmiş veri saklama nesneleri oluşturmaya yarar. Örneğin her öğrenci okul numarası ve ismi ile kaydedilir. Aşağıdaki yapı basit bir öğrenci yapisi (OOP – Nesneye Dayalı Programlama dillerinde nesne olarak adlandırılır) oluşturur:

```
struct ogrenci_nesnesi {
      int numara;
      char isim[50];
};

struct ogrenci_nesnesi buse;
```

Yapılar, tek bir değişken ile bir nesneye ait tüm özellikleri takip etmeyi sağlar. Yukarıdaki örnekte önce öğrenci nesnesi struct kullanılarak tanımlanmıştır. Daha sonra buse isimli bir öğrenci değişkeni oluşturulmuştur.

Yapıların ilk değerleri aşağıdaki şekilde atanır:

```
struct ogrenci_nesnesi buse = { 1015, “Buse Eyüpoğlu“ };
```

Bu nesnenin alt elemanlarina buse.numara ve buse.isim şeklinde erişilir.

Yeni Veri Türü Tanımlama, typedef

typedef, bir veri yapısını veya veri türünü isimlendirmek, tanımlamak için kullanılır.

```
typedef int tamsayi;
typedef struct ogrenci_nesnesi ogrenci;
```

Bu tanımlamalardan sonra int veri türü yerine tamsayi, “struct ogrenci_nesnesi” yerine ise kisaca ogrenci kullanilabilir.

union

Union, değişik tür ve boyutta verileri tek bir alanda saklayabilen veri yapısıdır.

```
union number {
      short total;
      doubgle avarage;
} number_x;
```

Bu farklı veri türlerine şu şekilde erişilir: number_x.total, number_x.avarage.

C derleyici union bir nesne için bellek tahsis ederken, elemanlarından en fazla belleğe gereksinim duyana göre boyut belirler. Yukarıdaki örnekte number_x bellekte double bir veri türü kadar yer kaplar.

Sıralı Türler, enum

Sıralı türler, sabit değerler listesidir ve tamsayı olarak adreslenirler. Haftanın günleri veya yılın ayları tipik örneklerdir.

```
enum gunler { paz, pts, sal, car, per, cum, cts };
enum gunler ilkhafta;
```

İlk eleman 0'dan başlamak üzere ardışıl sırada index numarası atanır. Hangi değerden başlaması gerektiğini programcı belirleyebilir.

```
enum yazaylari { haziran=6, temmuz, agustos };
```

İlk eleman haziranın index numarası 6, temmuz 7, ağustos 8'dir.

static

Statik olarak tanımlanan bir değişken tanımlı olduğu fonksiyona özeldir ve bir defa ilk değeri atanır. Bundan sonraki fonksiyon çağırmalarında değerini korur. Yani bir sonraki çağrılmada, bir önceki çağrılmadaki değere sahiptir.

```
main()
{
      int i;

      for (i=0;i<3;++i)
      stat();

      return 0;
}

stat() {
      int auto_var = 0;
      static int static_var = 0;

      printf("auto = %d, static = %d \n", auto_var, static_var);
      ++auto_var;
      ++static_var;
      return 0;
}
```

Program çıktısı aşağıdaki gibi olacaktır:

```
auto_var = 0, static_var= 0
auto_var = 0, static_var = 1
auto_var = 0, static_var = 2
```

Harici Değişkenler

Harici değişkenler programın tamamı tarafından erişebilir olan değişkenlerdir. Tersi, yerel değişkenler yalnızca fonksiyon içerisinde geçerlidir. Harici değişkenler program boyunca kalıcıdır; yerel değişkenler ise geçicidir, tanımlı oldukları fonksiyonla birlikte silinirler.

Harici değişkenler extern anahtar kelimesi kullanılarak tanımlanır. Aşağıda uid ve gid fonksiyonların dışında tanımlanmış olup extern ile fonksiyon içinde kullanılmıştır. Programın en başındaki tanımlama değişkenler için bellekte yer ayrılması içindir. Sonraki extern ifadesi yapılan tanımlamalar ise sadece kullamak içindir.

Örnek:

```
#include <stdio.h>

int uid, gid;

int main() {
        extern int uid, gid;

        gid = 1001;
        setuid(101);
        printf("Gid: %d, Uid: %d\n", gid, uid);

        return 0;
}

int setuid(int nuid) {
        extern uid;

        uid = nuid;

        return 0;
}
```

```
$ ./external
Gid: 1001, Uid: 101
```

Eğer harici deişken kullanılmadan önce tanımlanmış ise extern edilmesine gerek yoktur. Fakat garantiye almak için extern ifadesini her zaman kullanmak yararlıdır. Böylece include

ile başka kodları kendi kodunuza dahil ettiğinizde tanım mı kod mu daha önce gelecek diye düşünmenize gerek kalmaz.

Extern ile tanım yaparkan dizilerin boyutunu vermeye gerek yoktur. Çünkü yer ayrılması daha önceki tanımlama ile olmuştur. Örneğin `char line[1024]` olarak tanımlanmış bir değişken extern edilirken `extern line[]` demek yeterlidir.

Uyarı: Eğer bir fonksiyon içinde harici değişkenle aynı isme sahip yerel (local) değişken var ise değişken ismi ile yerel değişkene ulaşılır.

Harici değişkenler daha çok fonksiyonlar arasında parametre aktarmaya karşı alternatif olarak kullanılır. Ancak çok fazla sayıda harici değişken kullanmak programı debug etme ve yönetmeyi zorlaştırır. Çünkü fonksiyonlar genelde “kara kutu” olarak kullanılır. Yani işlevi iyi tanımlanmış, verilen girdiye göre üreteceği çıktı bellidir ve içinde nelerin döndüğü diğer fonksiyonları ilgilendirmez. Ancak harici değişkenler vasıtasıyla bu kara kutu artık başka fonksiyonların da eriştiği ve kullandığı değişkenlere müdahale edeceğinden programın yönetilmesi daha zor olacaktır.

Register Değişkenler

İşlemci veri saklama birimleri olan kaydediciler (register) sistem belleğine göre çok daha hızlı işlenir. Normalde işlemci kaydedicileri genel kullanıma açıktır ve derleyici hangi kaydediciyi ne zaman kullanacağına kendisi karar verir. Fakat C dili, derleyiciye istenen değişkenleri eğer mümkünse `register` üzerinde saklamasını tavsiye edebilir. Bu sayede programın hızına etki sağlayacak önemli bir değişken veya sürekli erişilen bir değişken kaydedici üzerinde saklanarak programın verimi artırılabilir.

Register değişken şu şekilde tanımlanabilir:

```
register float input;
```

Son Söz

Veri türleri programlama dilinin temelidir ve ilk öğrenilmesi gereken konulardandır. Bu belge C veri türleri için ortalama bir programcının ihtiyaçlarını karşılayacak niteliktedir. Bu konu anlaşılmadan veri yapılarına geçilmemelidir.

Barış Şimşek
http://www.enderunix.org/simsek

25 Ekim 2005, Beylerbeyi/Üsküdar