C
Struct Union Enum Typedef

Dr. Öğr. Üyesi M. Ozan AKI

Rev 1.0

Neden

C Dilinde ön tanımlı veri tipleri birçok basit uygulama kodlaması için yeterli olmakla birlikte çoğu zaman özel veri tiplerine ihtiyaç duyulur.

C dilinde, struct, union, enum ve typedef ile ön tanımlı veri tipleri kullanılarak farklı veri türleri tanımlanabilmektedir.

Birçok dil de benzer yapıları içermekte ve özel veri tipleri tanımlamayı mümkün kılmaktadır.

struct (yapı)

struct(ure), birbirleriyle mantıksal bir bütün oluşturan farklı tipteki verilerin bir arada oluşturduğu topluluk ya da koleksiyonlardır.

Bu topluluklara tek bir isim ile referans verilir. Yapıyı meydana getiren değişkenlere ise üye adı verilir.

Üyelere, değişken adından sonra bir (.) nokta operatörü konularak erişilebilir.

Eğer struct yapı bir pointer tarafından gösteriliyorsa üyelere erişmek için (->) operatörü kullanılır.

struct – deklarasyon ve tanım

Tipik bir struct deklarasyonu ve tanımı şu şekildedir;

```
struct tarih_t {
    unsigned char gun;
    unsigned char ay;
    unsigned int yil;
};

struct tarih_t dogum_tarihi;
```

struct – deklarasyon ve tanım

Ya da, struct deklarasyonu ile aynı anda değişkenler de tanımlanabilir;

```
struct tarih_t {
    unsigned char gun;
    unsigned char ay;
    unsigned int yil;
} dogum_tarihi, kayit_tarihi;
```

struct – deklarasyon ve tanım

Eğer program boyunca sadece bir struct yapı değişkeni kullanılacaksa tip adı belirtilmeden doğrudan değişken tanımlanabilir.;

```
struct {
    unsigned char gun;
    unsigned char ay;
    unsigned int yil;
} dogum_tarihi, kayit_tarihi;
```

struct – üyelere erişim

struct yapı değişkeni adını takiben (.) nokta konularak yapı üyelerine erişilir.

```
struct {
    unsigned char gun;
    unsigned char ay;
    unsigned int yil;
} dogum_tarihi, kayit_tarihi;
```

```
dogum_tarihi.gun = 18;
dogum_tarihi.ay = 6;
dogum_tarihi.yil = 1986;
```

struct – bellekteki durum

Tanımlanan her bir struct yapısı, üyelerinin tek tek kapladığı alanın toplamı kadar yer bellekte yer kaplar.

```
struct {
    unsigned char gun;
    unsigned char ay;
    unsigned int yil;
} dogum_tarihi, kayit_tarihi;
```

sizeof(dogum_tarihi.gun) == 1; // unsigned char
sizeof(dogum_tarihi.ay) == 1; // unsigned char
sizeof(dogum_tarihi.yil) == 4; // unsigned int

sizeof(dogum_tarihi) == 6; // struct dogum_tarihi

union (birleşim)

union, struct yapısına benzer. Aynı şekilde deklare edilip tanımlanabilir.

Ancak union birleşiminin farkı, **tüm üye değişkenlerin aynı bellek alanını paylaşmalarıdır.**

Bu demektir ki, bir union birleşiminde farklı üyelerde farklı bilgileri saklamak olanaksızdır. Zaten amaç bu değildir.

union

union birleşiminin kullanım amacı, bir veri tipine ait bit dizisini, farklı bir veri tipi şeklinde yorumlayabilmektir.

bir union birleşimi tanımlandığında, bellekten sadece üye değişkenlerden en büyüğü (en çok bit alan kaplayanı) kadar yer ayrılır ve diğer üyeler LSB bitlerine hizalanarak aynı alanı kullanırlar.

union

Tipik bir union deklarasyonu ve tanımlaması şu şekildedir;

```
union birlesim_t {
   int i;
   char c;
};
union birlesim_t birlesim;
```

union

```
union {
   int i;
   char c;
} birlesim;

birlesim.i = 260;
printf("%d", birlesim.c); // 4 ? neden
```

union ve struct

```
union {
    int butun;
    struct {
        char low;
        char high;
    } parca;
} birlesim;

birlesim.butun = 260;
printf("%d", birlesim.parca.low); // 4
printf("%d", birlesim.parca.high); // 256
```

union ve struct

```
union {
    int butun;
    struct {
        char r;
        char g;
        char b;
    } parca;
} renk;

renk.r = 120;
renk.g = 40;
renk.b = 210;
printf("%d", renk.butun);
```

union ve üye bit sayısı

```
union {
   int butun;
   struct {
        unsigned zero:1;
        unsigned carry:1;
        unsigned prescalar:3;
    } bitler;
} renk;
```

enum (küme)

enum(aration) kümeleri, programcıya kod yazarken kolaylık sağlamak amacıyla bazı int tipinde sabitlere isim verilmesine izin veren yapılardır.

Bu sayede;

Kaynak kodun okunabilirliği artar,

Kod yazarken yapılabilecek hatalar en az seviyeye düşürülmüş olur.

enum - tanımlama

Tipik bir enum deklarasyonu ve tanımlaması şu şekildedir;

enum gunler_t **{** pazartesi, sali, carsamba, persembe, cuma, cumartesi, pazar **};**

enum gunler_t gun;

gun = carsamba;

enum - tanımlama

Farklı bir enum deklarasyonu ve tanımlaması şu şekilde olabilir;

enum gunler_t **{** pazartesi, sali, carsamba, persembe, cuma, cumartesi, pazar **}** gun;

gun = pazar;

if(gun == pazartesi) printf(“Pazartesi”);

enum – değer atama

enum gunler_t **{** pazartesi, sali, carsamba, persembe, cuma, cumartesi, pazar **}** gun;

... gibi tipik bir enum tanımında, kümenin tüm elemanları derleyici tarafından sıfırdan başlayan ardaşık tam sayılar atanır.

enum – değer atama

enum gunler_t **{** ocak=1, subat=2,
mart=3, nisan=4, mayis=5, haziran=6,
temmuz=7, agustos=8, eylul=9,
ekim=10, kasim=11, aralik=12 **}** gun;

Derleyici tarafından atanan tam sayı değerleri programcı tarafından değiştirilebilir.

enum – değer atama

enum gunler_t **{** ocak=1, subat, mart, nisan, mayis, haziran, temmuz, agustos, eylul, ekim, kasim, aralik **}** gun;

Ancak her bir üyeye değer atamak zorunlu değildir.

Derleyici, eğer değer atanmamış bir üyeyle karşılaşırsa, en son değer atanan üyeden itibaren ardaşık sayıları atamaya devam eder.

enum – bitmap değerler

enum yonler_t **{** kuzey=0x01, guney=0x02, dogu = 0x04, bati = 0x08 **}** yon;

enum kümelerinin en çok kullanıldığı alanlardan biri, çoklu seçenekler oluşturabilen yapılardır.

Burada, kümenin her bir üyesi, bir tamsayının faklı basamaklardaki bitlerini temsil eder.

enum – bitmap değerler

```
enum yonler_t { kuzey=0x01,
  guney=0x02, dogu = 0x04, bati = 0x08 }
  yon;
```

Böylece, & ve | bitsel operatörleri ile birden çok seçenek birleştirilebilir ya da test edilebilir.

enum – bitmap değerler

```
enum yonler_t { kuzey=0x01,
   guney=0x02, dogu = 0x04, bati = 0x08 }
   yon;
yon = kuzey;
yon = guney | bati;
if(yon == dogu) ...
if(yon == (kuzey|dogu)) ...
```

typedef

typedef anahtar kelimesi, mevcut bir veri tipinin faklı bir isimle kullanılabilmesine olanak tanır.

typedef, yeni bir veri tipi tanımlamaz.

Örneğin, kaynak kod içerisinde çok sayıda unsigned in tanımlaması yapılıyorsa;

typedef unsigned int uint;

şeklinde tanımlayarak artık bu tipteki değişkenleri kısaca;

uint a,b,c; // şeklinde tanımlayabiliriz.

typedef

Ya da struct, union ve enum veri tipi tanımlamalarını typedef ile yaparak her değişken tanımında struct, union ve enum anahtar kelimelerini kullanmaktan tasarruf edebiliriz;

```
typedef struct tarih_t {
    unsigned char gun;
    unsigned char ay;
    unsigned int yil;
};

tarih_t dogum_tarihi;
```