Bölüm 10 - Structure, Union, Bit İşlemleri, ve Numaralandırma

İçerik

10.1 Giriş

- Structure (Yapı)
  - Tek isim altında toplanan ilişkili değişkenler topluluğu
    - Farklı veri tiplerinden değişkenler içerebilir
  - Çoğunlukla dosyalara yüklenecek kayıtları tanımlamak için kullanılır
  - Pointerlarla birlikte kullanılır, bağlantılı listeler, stoklar, kuyruklar ve branşlar (dallar) oluşturabilir

Örnek

```
struct kart {
        char *karo;
        char *kupa;
    };
```

`struct` kart yapısının tanımını verir

- **`card`** yapının adıdır ve yapı tipindeki değişkenleri tanımlamak için kullanılır
- **`card`** **`char *`** tipinde iki **üye** içerir
  - Bu üyeler **`karo`** ve **`kupa`** dır

10.2 Structure (Yapı) - Tanımı

- **`struct`** özellikleri
  - Bir **`struct`** kendini içermez
  - Aynı yapı türüne pointer olan bir üye içerebilir
  - Bir yapı tanımı bellekte yer kaplamaz
    - Bunun yerine yapı değişkenlerini tanımlayan yeni bir veri tipi oluşturur
- Tanımlanması
  - Diğer değişkenler gibi tanımlanır:
    - `kart birKart, deste[ 52 ], *kPtr;`
  - Virgülle ayrılan bir liste oluşturabilir:

```
struct kart {
    char *karo;
    char *kupa;
} birKart, deste[ 52 ], *kPtr;
```

Geçerli Operasyonlar

- Aynı tipten bir yapıya bir yapı atama
- Bir yapının adresini (**&**) alma
- Yapının bir üyesine ulaşma
- **`sizeof`** operatörü kullanarak yapının boyutunu belirleme

- Başlangıç Listesi
  - Örnek:

```
kart birKart= { "Üç", "Kupa" };
```

- Atama Deyimleri
  - Örnek:

```
kart ucKupa= birKart;
```

  - **ucKupa** şu şekilde de tanımlanabilir:

```
kart ucKupa;
ucKupa.yuz= "üç";
ucKupa.tip= "Kupa";
```

- Structure üyesine erişim
  - Structure değişkeni ile nokta operatörü (.) kullanılır

```
kart benimKartim;
printf( "%s", benimKartim.kupa );
```

  - Structure değişkenine pointer ile ok işareti (->) kullanılır

```
kart *benimKartPtr = &benimKartim;
printf( "%s", benimKartPtr -> kupa );
```

  - **benimKartPtr -> kupa**

    (* benimKartPtr).kupa ya denktir

10.5 Structure - Fonksiyonlar ile Kullanımı

- Yapıyı fonksiyona atama
  - Tüm yapıyı fonksiyona aktar
    - Veya belli üyeleri aktar
  - Her iki aktarım da değer ile çağrılır
- Referansla-çağırma ile yapıyı aktarma
  - Yapının adresini aktar
  - Referansı aktar
- Değer ile çağırmayla dizileri aktarma
  - Dizi üyesi olacak şekilde bir yapı oluştur
  - Yapıyı aktar

10.6 `typedef` Tip Tanımlaması

- **`typedef`**
  - Daha önce tanımlanmış veri tiplerine eşanlam (lakap) oluşturur
  - Daha kısa tip adları oluşturmak için kullanılır
  - Örnek:

```
typedef struct Kart *KartPtr;
```

  - **`struct Kart *`** tipi için bir lakap olarak **`KartPtr`** adlı yeni bir tip adı oluşturur
  - **`typedef`** yeni bir veri tipi oluşturmaz
    - Sadece bir lakap oluşturur

10.7 Örnek: Yüksek Performanslı Kart Karma ve Dağıtma Simulasyonu

- Önkod:
  - Kart yapısının bir dizisini oluştur
  - Kartları desteye koy
  - Desteyi kar
  - Kartları dağıt

```c
/* Fig. 10.3: fig10 03.c
   yapı kullanarak kart karma ve dağıtma simulasyonu */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

struct kart{
   const char *yuz;
   const char *takim;
};

typedef struct kart Kart;

void tamDeste( Kart * const, const char *[],
               const char *[] );
void kar( Kart * const );
void dagit( const Kart * const );

int main()
{
   Kart deste[ 52 ];
   const char *yuz[] = { "As", "İki", "Üç",
                         "Dört", "Beş",
                         "Altı", "Yedi", "Sekiz",
                         "Dokuz", "On",
                         "Bacak", "Kız", "Papaz"};
   const char *takim[] = { "Kupa", "Karo",
                         "Maça", "Sinek"};

   srand( time( NULL ) );
```

```
31
32    tamDeste( deste, yuz, takim );
33    kar( deste );
34    dagit( deste );
35    return 0;
36 }
37
38 void tamDeste( Kart * const cDeste, const char * cYuz[],
39                const char * cTakim[] )
40 {
41    int i;
42
43    for ( i = 0; i <= 51; i++ ) {
44       cDeste[ i ].yuz = cYuz[ i % 13 ];
45       cDeste[ i ].takim = cTakim[ i / 13 ];
46    }
47 }
48
49 void kar( Kart * const cDeste )
50 {
51    int i, j;
52    Kart temp;
53
54    for ( i = 0; i <= 51; i++ ) {
55       j = rand() % 52;
56       temp = cDeste[ i ];
57       cDeste[ i ] = cDeste[ j ];
58       cDeste[ j ] = temp;
59    }
60 }
```

52 kartı desteye koy. **yuz** ve **deste** modül(kalan) ile belirlenir

0 ve **51** arasında rasgele sayı seç. **i** elemanını onunla yer değiştir

10.7 Örnek: Yüksek Performanslı Kart Karma ve Dağıtma Simulasyonu

```
61
62 void dagit( const Kart * const cDeste )
63 {
64    int i;
65
66    for ( i = 0; i <= 51; i++ )
67       printf( "%5s - %-8s%c", cDeste[ i ].yuz,
68               cDeste[ i ].takim,
69               ( i + 1 ) % 2 ? '\t' : '\n' );
70 }
```

Diziyi tara ve veriyi yaz

10.7 Örnek: Yüksek Performanslı Kart Karma ve Dağıtma Simulasyonu

```
Sekiz - Karo
Sekiz - Maça
 Yedi - Kupa
   As - Maça
  İki - Sinek
 Yedi - Sinek
Bacak - Maça
Papaz - Kupa
   Üç - Kupa
   Üç - Maça
   On - Kupa
   On - Maça
  Altı - Maça
  Altı - Kupa
Dokuz - Karo
Bacak - Sinek
Papaz - Karo
Dokuz - Sinek
  Altı - Sinek
   Kız - Karo
    As - Sinek
 Papaz - Maça
 Papaz - Sinek
   Kız - Kupa
  Dört - Sinek
  Dört - Maça
```

```
As - Kupa
Beş - Sinek
İki - Karo
On - Karo
Altı - Karo
İki - Maça
On - Sinek
Bacak - Karo
Üç - Karo
Dokuz - Maça
İki - Kupa
Yedi - Karo
Kız - Sinek
Üç - Sinek
As - Karo
Beş - Maça
Yedi - Maça
Dört - Kupa
Sekiz - Sinek
Beş - Karo
Dokuz - Kupa
Beş - Kupa
Dört - Karo
Sekiz - Kupa
Bacak - Kupa
Kız - Maça
```

10.8 Union (Bileşim)

- **union**
  - Zaman süresince değişik nesneler içeren bellek
  - Bir zamanda sadece bir veri üyesi içerir
  - **union** üyeleri aynı yeri paylaşır
  - Yerden tasarruf sağlar
  - Sadece tanımlanan son veri üyesine erişim sağlanır
- **union** tanımlaması
  - Yapı ile aynıdır

```
union Sayi{
   int x;
   float y;
};
union Sayi deger;
```

10.8 Union (Bileşim)

- Geçerli **`union`** işlemleri
  - Aynı tipten **`union`** a atama: **`=`**
  - Adres alma: **`&`**
  - union üyesine erişim: **`.`**
  - Üyeye pointer ile erişim: **`->`**

```c
/* Fig. 10.5: fig10_05.c
   union a bir örnek*/
#include <stdio.h>

union sayi {
   int x;
   double y;
};

int main()
{
   union sayi deger;

   deger.x = 100;
   printf( "%s\n%s\n%s%d\n%s%f\n\n",
           "Tamsayı üyesine bir değer ver ve",
           "her iki üyeyi yaz.",
           "int:   ", deger.x,
           "double:\n", deger.y );

   deger.y = 100.0;
   printf( "%s\n%s\n%s%d\n%s%f\n",
           "Reel üyeye bir değer ver ve",
           "her iki üyeyi yaz.",
           "int:   ", deger.x,
           "double:\n", deger.y );
   return 0;
}
```

10.8 Union (Bileşim)

```
Tamsayı üyesine bir değer ver ve
Her iki üyeyi yaz.
int:   100
double:
-92559592117433136000000000000000000000000000000000000000000000000.00000

Reel üyeye bir değer ver ve
Her iki üyeyi yaz.
int:   0
double:
100.000000
```

10.9 Bit Operatörleri

- Her veri bit dizileri şeklinde temsil edilir
  - Her bit ya **0** veya **1** dir
  - 8 bitlik bir dizi bir byte olur

Operatör	Adı	Açıklama
&	bitişlem AND (VE)	İki operandaki karşılık gelen bitlerin her ikisi de **1** ise sonuçtaki bitler **1** olur
|	bitişlem OR (VEYA)	İki operandaki karşılık gelen bitlerin en az biri **1** ise sonuçtaki bitler **1** olur
^	Sadece bitişlem için OR (VEYA)	İki operandaki karşılık gelen bitlerin sadece biri **1** ise sonuçtaki bitler **1** olur
<<	Sola kaydır	İkinci operandda belirlenen bir sayısı kadar birinci operanddaki bitleri sola kaydırır. Sağda geriye kalanları **0** ile doldurur
>>	Sağa kaydır	İkinci operandda belirlenen bir sayısı kadar birinci operanddaki bitleri sağa kaydırır. Solda geriye kalanları doldurma işlemi makineye bağlıdır
~	tümleyen	Tüm **0** bitler **1** ve tüm **1** bitler **0** olur

```c
/* Fig. 10.9: fig10 09.c
   Bitişlem AND, genel  OR,
   sadece bitişlem OR ve tümleyen operatörleri */
#include <stdio.h>

void bitYaz( unsigned );

int main()
{
   unsigned say1, say2, maske, bitKur;

   say1 = 65535;
   maske = 1;
   printf( "Aşağıdakilerin birleşiminin sonucu:\n" );
   bitYaz( say1 );
   bitYaz( maske );
   printf( "bitişlem  AND operatörü & ile\n" );
   bitYaz( say1 & maske );

   say1 = 15;
   bitKur = 241;
   printf( "\nAşağıdakilerin birleşiminin sonucu:\n" );
   bitYaz( say1);
   bitYaz( bitKur );
   printf( "genel bitişlem OR operatörü | ile\n" );
   bitYaz( say1 | bitKur );

   say1 = 139;
   say2 = 199;
   printf( "\nAşağıdakilerin birleşiminin sonucu:\n" );
```

```
31    bitYaz( say1 );
32    bitYaz( number2 );
33    printf( "sadece bitişlem OR operatörü  ^ ile\n" );
34    bitYaz( say1 ^ say2 );
35
36    say1 = 21845;
37    printf( "\n Sayı:\n" );
38    bitYaz( say1 );
39    printf( "tümleyeni:\n" );
40    bitYaz( ~number1 );
41
42    return 0;
43 }
44
45 void bitYaz( unsigned deger )
46 {
47    unsigned c, maskeYaz = 1 << 31;
48
49    printf( "%7u = ", deger );
50
51    for ( c = 1; c <= 32; c++ ) {
52       putchar( deger & maskeYaz ? '1' : '0' );
53       deger <<= 1;
54
55       if ( c % 8 == 0 )
56          putchar( ' ' );
57    }
58
59    putchar( '\n' );
60 }
```

MASKE sadece bir bit seti ile oluşturuldu
yani (**10000000 00000000**)

MASKE sürekli olarak **deger** e eklenir.
MASKE sadece bir bit içerir, böylece eğer **AND** true(doğru) gönderirse bunun anlamı **deger** o bite sahip olur.
deger daha sonra bir sonraki biti test için kaydırılır.

```
Aşağıdakilerin birleşiminin sonucu:
  65535 = 00000000 00000000 11111111 11111111
      1 = 00000000 00000000 00000000 00000001
bitişlem AND operatörü & ile
      1 = 00000000 00000000 00000000 00000001

Aşağıdakilerin birleşiminin sonucu:
     15 = 00000000 00000000 00000000 00001111
    241 = 00000000 00000000 00000000 11110001
genel bitişlem OR operatörü | ile
    255 = 00000000 00000000 00000000 11111111

Aşağıdakilerin birleşiminin sonucu:
    139 = 00000000 00000000 00000000 10001011
    199 = 00000000 00000000 00000000 11000111
Sadece bitişlem OR operatörü ^ ile
     76 = 00000000 00000000 00000000 01001100

Sayı:
  21845 = 00000000 00000000 01010101 01010101
tümleyeni
4294945450 = 11111111 11111111 10101010 10101010
```

10.10 Bit Alanları

- Bit Alanları
  - Boyutu (bit olarak) belirlenmiş bir yapı üyesi
  - Daha iyi bellek kullanımı sağlar
  - **`int`** veya **`unsigned`** olarak tanımlanır
  - Bireysel bitlere erişim sağlanmaz
- Bit alanları tanımlama
  - **`unsigned`** veya **`int`** üyeli ve (**`:`**) ile alan uzunluğunu veren bir tamsayı ile tanımlanır
  - Örnek:

```
struct BitKart{
   unsigned yuz: 4;
   unsigned takim : 2;
   unsigned renk : 1;
};
```

10.10 Bit Alanları

- Adlandırılmamış bit alanı
  - Alan yapıda destek olarak kullanılır
  - Bitlere bir şey yüklenmeyebilir

```
struct Ornek{
    unsigned a : 13;
    unsigned   : 3;
    unsigned b : 4;
}
```

  - Sıfır uzunluklu adlandırılmamış bit alanı bir sonraki bit alanını yeni bir depolama birimi sınırına hizalar

10.11 Enumeration Sabitleri

- Numaralandırma
  - Belirleyiciler ile temsil edilen tamsayı sabitleri seti
  - Enumeration sabitleri değerleri otomatik belirlenen sembolik sabitler gibidir
    - Değerler **0** ile başlar ve birer artar
    - Değerler **=** ile kesin olarak verilebilir
    - Kesin sabit adları vardır
  - Örnek:

```
enum Aylar{ OCK= 1, SUB, MAR, NIS, MAY, HAZ,
  TEM, AGU, EYL, EKM, KAS, ARA};
```

    - Belirleyicilerin **1 den 12** ye değiştiği yeni tip bir enum Aylar oluşturur
  - Enumeration değişkenleri sadece kendi enumeration sabit değerlerini kabul eder (integer temsilini değil)

10.11 Enumeration Sabitleri

```
1  /* Fig. 10.18: fig10_18.c
2     enumeration tip kullanımı*/
3  #include <stdio.h>
4
5  enum aylar { OCK = 1, SUB, MAR, NIS, MAY, HAZ,
6               TEM, AGU, EYL, EKM, KAS, ARA };
7
8  int main()
9  {
10    enum aylar ay;
11    const char *ayAdi[] = { "", "Ocak", "Şubat",
12                            "Mart", "Nisan", "Mayıs",
13                            "Haziran", "Temmuz", "Ağustos",
14                            "Eylül", "Ekim",
15                            "Kasım", "Aralık" };
16
17    for ( ay = OCK; ay <= ARA; ay++ )
18       printf( "%2d%11s\n", ay, ayAdi[ ay ] );
19
20    return 0;
21 }
```

10.11 Enumeration Sabitleri

```
 1          Ocak
 2         Şubat
 3          Mart
 4         Nisan
 5         Mayıs
 6       Haziran
 7        Temmuz
 8       Ağustos
 9         Eylül
10         Ekim
11         Kasım
12        Aralık
```