YAPILAR VE BİRLİKLER

Yapılar

Yapı Tanımlaması

Yapısal programlamaya adını veren yapı (structure), türetilmiş (derived) veya kullanıcı tanımlı (user defined) bir veri tipidir. Farklı tiplerdeki elemanları bir arada gruplandıran özel bir veri tipi tanımlamak için `struct` anahtar kelimesini kullanırız.

Yapı bir veya birden fazla ilkel veri tipinin (`char`, `int`, `long`, `float`, `double`, ... ve bu tiplere ait diziler) bir araya gelmesiyle oluşturulan yeni veri tipleridir. Diziler, aynı tipte elemanlardan oluşmasına rağmen, yapılar farklı dipte elemanların bir araya gelmesiyle oluşabilir;

```
struct yapı-kimliği {
    veri-tipi yapı-elemanı-kimliği1;
    veri-tipi yapı-elemanı-kimliği1;
    ...
} değişken-kimliği1, değişken-kimliği2;
```

Örnek olarak Öğrenci; adı, soyadı, numarası, yaşı, cinsiyeti gibi farklı öğeler ile bir `ogrenci` yapısı tanımlanabilir;

```
struct ogrenci {
    char adi[50];
    char soyadi[0];
    int yas;
    int cinsiyet;
} ogrenci1;
```

Tanımlanan yapı kullanılarak değişkenler aşağıdaki gibi tanımlanabilir;

```
struct yapı-kimliği değişken-kimliği;
```

Yukarıda tanımlanan yapı kimliği kullanılarak birçok değişken kimliklendirilebilir;

```
struct ogrenci ogrenci2,ogrenci3;
```

Yapı Elemanlarına Erişim

Tanımlanan yapı değişkenleri üzerinden her bir alamana nokta işleci (dot operator) erişilir. Bu işleç de öncelik işleci parantez `()` ile aynı önceliktedir. Atama işleci (`=`) bir yapıyı (struct) doğrudan kopyalamak için kullanılabilir. Ayrıca, bir yapının üyesinin değerini başka birine atamak için atama işlecini de kullanabiliriz.

```
#include <stdio.h>
#include <string.h>
int main() {
    struct ogrenci {
        char adi[50];
        char soyadi[50];
        int yas;
        int cinsiyet;
    } ogrenci1;
    strcpy(ogrenci1.adi, "Ilhan");
    strcpy(ogrenci1.soyadi, "OZKAN");
    ogrenci1.yas=50;
    ogrenci1.cinsiyet=1;
    printf("Adı:%s\nSoyad:%s\nYaşı:%d\nCinsiyeti:%d\n",
          ogrenci1.adi, ogrenci1.soyadi,
          ogrenci1.yas, ogrenci1.cinsiyet);
```

```
    struct ogrenci ogrenci2={"Deniz","AK",35,2}; /* yapı değişkenine
                                                    ilk değer veridi */
    printf("Adı:%s\nSoyad:%s\nYaşı:%d\nCinsiyeti:%d\n",
          ogrenci2.adi, ogrenci2.soyadi,
          ogrenci2.yas, ogrenci2.cinsiyet);
    struct ogrenci ogrenci3=ogrenci2;
    printf("Adı:%s\nSoyad:%s\nYaşı:%d\nCinsiyeti:%d\n",
          ogrenci3.adi, ogrenci3.soyadi,
          ogrenci3.yas, ogrenci3.cinsiyet);
    return 0;
}
```

Yapı Göstericileri

Yapı göstericileri, tıpkı diğer değişkenlere gösterici tanımladığımız gibi tanımlayabiliriz. Gösterici üzerinden yapı değişkenlerine dolaylı işleç (indirection operator) yani (`->`) ile erişilir. Bu nokta işleci ile aynı önceliklidir.

Yapılar ve göstericileri; veri tabanları, dosya yönetim uygulamaları ve ağaç ve bağlı listeler gibi karmaşık veri yapılarını işlemek gibi farklı uygulamalarda kullanılır.

```
#include <stdio.h>
#include <string.h>
int main() {
  struct ogrenci {
    char adi[50];
    char soyadi[50];
    int yas;
    int cinsiyet;
  } ogrenci1;
  strcpy(ogrenci1.adi, "Ilhan");
  strcpy(ogrenci1.soyadi, "OZKAN");
  ogrenci1.yas=50;
  ogrenci1.cinsiyet=1;
  struct ogrenci* ogrenciGosterici;
  ogrenciGosterici=&ogrenci1;
  printf("Adı:%s\nSoyad:%s\nYaşı:%d\nCinsiyeti:%d",
    ogrenciGosterici->adi,
    ogrenciGosterici->soyadi,
    ogrenciGosterici->yas,
    ogrenciGosterici->cinsiyet);
  return 0;
}
```

Yapılarla İşlemler

Bir yapı (struct) değişkeni, ilkel tiplerden (`char`, `int`, `float`, ...) tanımlanan bir diziye benzer şekilde bir yapı dizisi tanımlayabiliriz. Ayrıca yapı değişkenini bir fonksiyona parametre olarak gönderebilir ve bir fonksiyondan bir yapı döndürebilirsiniz.

```
#include <stdio.h>
#include <string.h>
enum cinsiyet {BELIRTILMEMIS,KADIN,ERKEK};
struct ogrenci {
    char adi[50];
    char soyadi[50];
    int yas;
    int cinsiyet;
};

struct ogrenci yeniOgrenci();
void ogreciYaz(struct ogrenci);
```

```
int main() {
  struct ogrenci ogrenciler[30];
  struct ogrenci o=yeniOgrenci();
  ogreciYaz(o);
  return 0;
}
struct ogrenci yeniOgrenci() {
    struct ogrenci yeni={"Ilhan","Ozkan",50,ERKEK};
    return yeni;
}
void ogreciYaz(struct ogrenci pOgrenci) {
    printf("Adı:%s\nSoyad:%s\nYaşı:%d\nCinsiyeti:%d",
        pOgrenci.adi,
        pOgrenci.soyadi,
        pOgrenci.yas,
        pOgrenci.cinsiyet);
}
```

Yapılar değer tipler olduğundan, yapılara ait göstericileri parametre olarak kullanmak, olabilecek değişiklikleri aktarmak için üstünlük sağlar.

```
#include <stdio.h>
#include <string.h>
enum cinsiyet {BELIRTILMEMIS,KADIN,ERKEK};
struct ogrenci {
    char adi[50];
    char soyadi[50];
    int yas;
    int cinsiyet; //1:erkek,2:Kadın,0:Belirtmiyor
};

struct ogrenci yeniOgrenci();
void ogrenciOku(struct ogrenci*);

int main() {
  struct ogrenci o=yeniOgrenci();
  ogrenciOku(&o);
  return 0;
}

struct ogrenci yeniOgrenci() {
    struct ogrenci yeni={"Ilhan","Ozkan",50,ERKEK};
    return yeni;
}
void ogrenciOku(struct ogrenci* pOgrenci) {
    printf("Adı Giriniz:");
    scanf("%s",pOgrenci->adi);
    printf("Soyadı Giriniz:");
    scanf("%s",pOgrenci->soyadi);
    printf("Yaş Giriniz:");
    scanf("%d",&(pOgrenci->yas));
    printf("Cinsiyet Giriniz (0-1-2):");
    scanf("%d",&(pOgrenci->cinsiyet));
}
```

Anonim Yapılar

Anonim yapı, kimlik veya takma isim (`typedef`) ile tanımlanmayan bir yapı tanımıdır. Genellikle başka bir yapının içine yerleştirilir. 2011 C sürümü ile kullanılmaya başlanan bu özelliğin aşağıda sıralanan üstünlükleri vardır;

– Esneklik (flexibility): Anonim yapılar, verilerin nasıl temsil edildiği ve erişildiği konusunda esneklik sağlayarak daha dinamik ve çok yönlü veri yapılarına olanak tanır.

- Kolaylık (convenience): Bu özellik, farklı veri tiplerini tutabilen bir değişkenin kompakt bir şekilde temsil edilmesine olanak tanır.
- Başlatma Kolaylığı (easy of initialization): Yapı değişkenine ilişkin ek kimliklendirme yapılmadan ilk değer verilmeleri ve kullanılmaları daha kolay olabilir.

```
#include <stdio.h>
#include <string.h>
struct ogrenci {
    char adi[50];
    char soyadi[50];
    int yas;
    int cinsiyet; //1:erkek,2:Kadın,0:Belirtmiyor
    struct {
      int gun;
      int ay;
      int yil;
    };
};
struct ogrenci yeniOgrenci();
void ogrenciYaz(struct ogrenci);
int main() {
  struct ogrenci o=yeniOgrenci();
  ogrenciYaz(o);
  return 0;
}
struct ogrenci yeniOgrenci() {
    struct ogrenci yeni={"Ilhan","Ozkan",50,1, {1,1,1970}};
    return yeni;
}
void ogrenciYaz(struct ogrenci pOgrenci) {
    puts(pOgrenci.adi);
    puts(pOgrenci.soyadi);
    printf("Yas:%d\n",pOgrenci.yas);
    printf("Cinsiyet:%d\n",pOgrenci.cinsiyet);
    printf("Dogum Tarihi:%d-%d-%d\n",pOgrenci.gun,pOgrenci.ay,pOgrenci.yil);

}
```

Öz Referanslı Yapılar

Kendi kendine yani öz referanslı yapı (self-referetial struct), öğelerinden bir veya daha fazlası kendi türündeki göstericilerden oluşan bir yapıdır. Kendi kendine referanslı kullanıcı tanımlı yapılar, bağlantılı listeler ve ağaçlar gibi karmaşık ve dinamik veri yapıları (dynamic data structure) için yaygın olarak kullanılırlar.

```
#include <stdio.h>
struct ogrenci {
    char adi[10]; char soyadi[10]; int yas;
    struct ogrenci* sonrakiOgrenci;
};
void ogrecileriYaz(struct ogrenci*);
int main() {
  struct ogrenci ilk={"Ilhan","OZKAN",50,NULL};
  struct ogrenci sonraki={"Ayse","YILMAZ",40,NULL};
  ilk.sonrakiOgrenci=&sonraki;
  struct ogrenci birsonraki={"Tahsin","BULUT",35,NULL};
  sonraki.sonrakiOgrenci=&birsonraki;
  ogrecileriYaz(&ilk);
  return 0;
}
void ogrecileriYaz(struct ogrenci* pIlk) {
    int sayac=0;
```

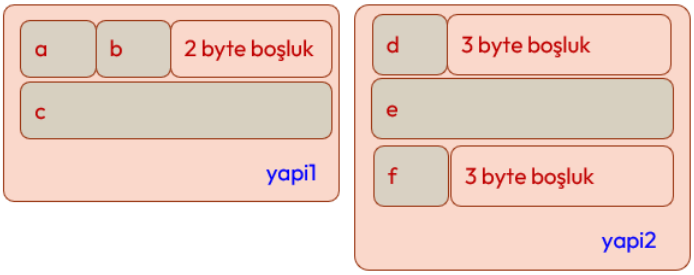
```
    while (pIlk!=NULL) {
        printf("--OGRENCI:%d--\nAdı:%s\nSoyad:%s\nYaşı:%d\n",
        ++sayac, pIlk->adi, pIlk->soyadi, pIlk->yas);
        pIlk=pIlk->sonrakiOgrenci;
    }
}
/*Program Çıktısı:
--OGRENCI:1--
Adı:Ilhan
Soyad:OZKAN
Yaşı:50
--OGRENCI:2--
Adı:Ayse
Soyad:YILMAZ
Yaşı:40
--OGRENCI:3--
Adı:Tahsin
Soyad:BULUT
Yaşı:35

...Program finished with exit code 0
*/
```

Yapı Dolgusu

C dilinde yapı dolgusu (structure padding), işlemci (CPU) mimarisi ile belirlenir. Dolgu (padding) işlemi ile üyelerinin bellekte doğal olarak hizalanması için belirli sayıda boş bayt eklenir. Bunun nedeni 32 veya 64 bitlik bir bilgisayarda işlemcinin tek seferde bellekten 4 bayt okumasından kaynaklanmaktadır.

```
#include <stdio.h>
struct yapi1 {
   char a;
   char b;
   int c;
};
struct yapi2 {
   char d;
   int e;
   char f;
};
int main() {
   printf("char bellek miktarı: %d\n", sizeof(char));
   // char bellek miktarı: 1
   printf("int  bellek miktarı: %d\n", sizeof(int));
   // int  bellek miktarı: 4
   printf("-----------------------\n");
   printf("yapilar için olmasi gereken bellek miktarı: %d\n",
      2*sizeof(char)+sizeof(int));
   // yapilar için olmasi gereken bellek miktarı: 6
   printf("yapi1 için bellekte ayrılan miktar: %d\n", sizeof(struct yapi1));
   // yapi1 için bellekte ayrılan miktar: 8
   printf("yapi2 için bellekte ayrılan miktar: %d\n", sizeof(struct yapi2));
   // yapi2 için bellekte ayrılan miktar: 12
   return 0;
}
```

Yukarıda verilen örnekte aynı bellek miktarına sahip elemanlar için yapının bütününe bakıldığında farklı miktarda bellek ayrılabileceği aşağıdaki şekilden anlaşılmaktadır;



Şekil 21. Yapı Dolgusu

Dolgu (padding) işlemi ile üyelerinin bellekte doğal olarak hizalanması için belirli sayıda boş bayt eklenir. Bunu tüm yapılar için engellemenin yolu;

```
#pragma pack(1)
```

Ön işlemci yönergesini (preprocessor directive) kaynak koda eklemektir. Eğer yalnızca belirlenen yapı için bunun yapılması isteniyorsa yapı tanımına `__attribute__((packed))` özelliği eklenir. Bu durumun engellendiği program aşağıdaki verilmiştir.

```
#include <stdio.h>
#pragma pack(1)
struct yapi1 {
   char a;
   char b;
   int c;
};
struct __attribute__((packed)) yapi2 {
   char a;
   int b;
   char c;
};
int main() {
   printf("char bellek miktarı: %d\n", sizeof(char));
   // char bellek miktarı: 1
   printf("int  bellek miktarı: %d\n", sizeof(int));
   // int  bellek miktarı: 4
   printf("------------------------\n");
   printf("yapılar için olması gereken bellek miktarı: %d\n",
      2*sizeof(char)+sizeof(int));
   // yapılar için olması gereken bellek miktarı: 6
   printf("yapi1 için bellekte ayrılan miktar: %d\n", sizeof(struct yapi1));
   // yapi1 için bellekte ayrılan miktar: 6
   printf("yapi2 için bellekte ayrılan miktar: %d\n", sizeof(struct yapi2));
   // yapi1 için bellekte ayrılan miktar: 6
   return 0;
}
```

Birlikler

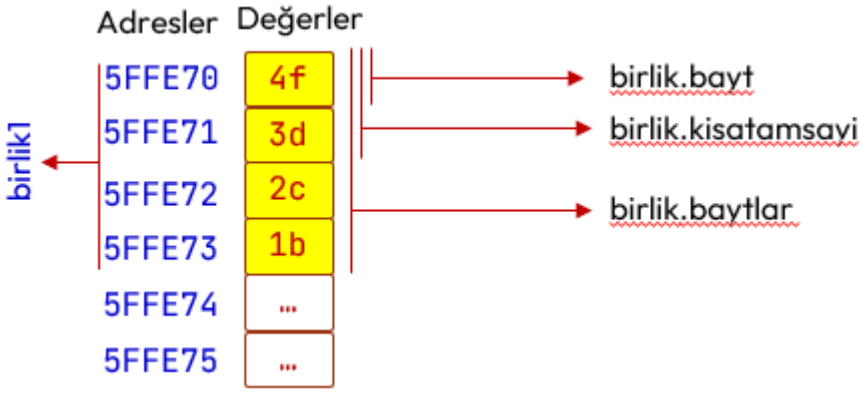
Birlikler (union), Pascal dilindeki `record case` talimatına (statement) benzer. Birlik (union), yapı (struct) gibi tanımlanır. Aralarındaki fark yapı elemanlarının her birine ayrı bellek bölgesi ayrılırken, birlik üyelerinin her biri aynı bellek bölgesini paylaşırlar.

```
union yapı-kimliği {
    veri-tipi yapı-elemanı-kimliği1;
    veri-tipi yapı-elemanı-kimliği1;
    ...
} değişken-kimliği1, değişken-kimliği2;
```

Birliğin belek boyutu, elemanlarından en fazla bellek kaplayanı kadardır. Aşağıda üyelerinin aynı bellek bölgesini paylaştığı görülmektedir.

```
#include <stdio.h>
int main() {
  union tamsayiBirligi {
    char baytlar[4];
    int  tamsayi;
    char bayt;
    short int kisatamsayi;
  } birlik1,birlik2;
  union tamsayiBirligi birlik3,birlik4;
  birlik1.tamsayi=0x1B2C3D4F;
          //16lık sayılarda her çift rakam bir bayt olur
  printf("sizeof tamsayiBirligi.baytlar: %d\n", sizeof birlik1.baytlar);
          //4
  printf("sizeof tamsayiBirligi.bayt: %d\n", sizeof birlik1.bayt);
          //1
  printf("sizeof tamsayiBirligi.tamsayi: %d\n", sizeof birlik1.tamsayi);
          //4
  printf("sizeof tamsayiBirligi.kisatamsayi: %d\n",
          sizeof birlik1.kisatamsayi);
          //2
  printf("---------------------------------------\n");
  printf("sizeof tamsayiBirligi: %d\n", sizeof birlik1);
          //4
  printf("---------------------------------------\n");
  printf("birlik1.tamsayi:%x\n",birlik1.tamsayi);         //1b2c3d4f
  printf("birlik1.baytlar:%x-%x-%x-%x\n",
         birlik1.baytlar[0],
         birlik1.baytlar[1],
         birlik1.baytlar[2],
         birlik1.baytlar[3]);                             //4f-3d-2c-1b
  printf("birlik1.bayte:%x\n",birlik1.bayt);               //4f
  printf("birlik1.kisatamsayi:%x\n",birlik1.kisatamsayi); //3d4f
  printf("---------------------------------------\n");
  birlik1.bayt=0x00;
  printf("birlik1.tamsayi:%x\n",birlik1.tamsayi);         //1b2c3d00
  return 0;
}
```

Şekil olarak da birlik1 değişkeninin bellek yerleşimi aşağıda verilmiştir;



Şekil 22. Birlik1 Değişkeninin Bellek Yerleşimi