YAPILAR VE BİRLİKLER

Yapı Tanımlaması

Yapısal programlamaya adını veren yapı (struct), 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 {
        unsigned yas;
        char cinsiyet;
        float kilo;
        unsigned boy;
    } 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;
```

Yapılar sınıflara benzer aralarındaki fark erişim belirleyicisinin halka açık (`public`) olmasıdır;

```
struct Vector {
    int x;
    int y;
    int z;
};
// Aşağıdakine eşdeğerdir:
class Vector {
public:
    int x;
    int y;
    int z;
};
```

Yapı Elemanlarına Erişim

Tanımlanan yapı değişkenleri üzerinden her bir alamana nokta işleci (dot operator) erişilir. Bu operatör de parantez öncelik işleci 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 <iostream>
using namespace std;
int main() {
    struct ogrenci {
```

```
        unsigned yas;
        char cinsiyet;
        float kilo;
        unsigned boy;
    } ogrenci1;

    ogrenci1.yas=19;
    ogrenci1.cinsiyet='E';
    ogrenci1.kilo=75.5;
    ogrenci1.boy=180;
    cout << "Öğrenci1'in:" << endl << "yaşı:" << ogrenci1.yas<< ", cinsiyeti: "
         << ogrenci1.cinsiyet << ", kilosu:"<< ogrenci1.kilo
         << ", boyu:" << ogrenci1.boy << endl;

    struct ogrenci ogrenci2={25,'K',55.0,'K'}; //yapı değişkenine ilk değer verme.
    cout << "Öğrenci 2'nin:" << endl << "yaşı:" << ogrenci2.yas<< ", cinsiyeti: "
         << ogrenci2.cinsiyet << ", kilosu:"<< ogrenci2.kilo << ", boyu:"
         << ogrenci2.boy << endl;

    struct ogrenci ogrenci3=ogrenci2;
    cout << "Öğrenci 3'ün:" << endl << "yaşı:" << ogrenci3.yas<< ", cinsiyeti: "
         << ogrenci3.cinsiyet << ", kilosu:"<< ogrenci3.kilo << ", boyu:"
         << ogrenci3.boy << endl;
}
```

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 <iostream>
using namespace std;
int main() {
    struct ogrenci {
        unsigned yas;
        char cinsiyet;
        float kilo;
        unsigned boy;
    } ogrenci1;

    ogrenci1.yas=19;
    ogrenci1.cinsiyet='E';
    ogrenci1.kilo=75.5;
    ogrenci1.boy=180;

    cout << "Öğrenci1'in:" << endl << "yaşı:" << ogrenci1.yas
         << ", cinsiyeti: " << ogrenci1.cinsiyet
         << ", kilosu:" << ogrenci1.kilo
         << ", boyu:" << ogrenci1.boy << endl;

    struct ogrenci* ogrenciGosterici;
    ogrenciGosterici=&ogrenci1;
    cout << "Öğrenci1'in:" << endl << "yaşı:" << ogrenciGosterici->yas
         << ", cinsiyeti: " << ogrenciGosterici->cinsiyet
         << ", kilosu:" << ogrenciGosterici->kilo
         << ", boyu:" << ogrenciGosterici->boy << endl;
}
```

Takma İsimler

Halihazırda mevcut veri tiplerinin adını yeniden tanımlamak için `typedef` anahtar kelimesi kullanılır. Takma isimler (typedef), yapı (struct) ve daha sonra göreceğimiz birlik (union) gibi veri tiplerinin çokça kullanıldığı kodlarda kodun boyunu oldukça kısaltır ve aşağıdaki şekilde tanımlanır;

```
typedef mevcutisim takmaisim;
```

Aşağıda takma isim verilmiş yeni veri tipleriyle yazılmış bir kod örneği verilmiştir;

```
#include <iostream>
using namespace std;

typedef char karakter;
typedef unsigned int pozitiftamsayi;
typedef float gerceksayi;

struct ogrenci {
    pozitiftamsayi yas;
    karakter cinsiyet;
    gerceksayi kilo;
    pozitiftamsayi boy;
};
typedef struct ogrenci Ogrenci;
int main() {
    Ogrenci ogrenci1;
    ogrenci1.yas=19;
    ogrenci1.cinsiyet='E';
    ogrenci1.kilo=75.5;
    ogrenci1.boy=180;
    cout << "Öğrenci1'in:" << endl << "yaşı:" << ogrenci1.yas
         << ", cinsiyeti: " << ogrenci1.cinsiyet
         << ", kilosu:" << ogrenci1.kilo
         << ", boyu:" << ogrenci1.boy << endl;
}
```

Fonksiyonlara da takma isim verilebilir;

```
typedef int Fonk(int); /* Fonk bir fonksiyona verilen bir takma isimdir;
Fonk, int tipinde parametre alan ve int geri döndüren her fonksiyon olabilir */

int faktoriyel(int n) { //faktoriyel fonksiyonu da Fonk tipindedir.
    return (n==1)? 1: n*faktoriyel(n-1);
}
int onKat(int i) {
    return i*10;
}
int main() {
    Fonk *fn; // fn bir Fonk tipinde olan fonksiyonlara olan göstericidir.
    fn = &faktoriyel; // faktoriyel fonksiyonunu göster
    fn(3); // 3! Hesaplanır
    fn = &onKat; // onKat fonksiyonunu göster
    fn(4); // 4*10 hesaplanır
}
```

Yapı Dizileri

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 <iostream>
```

```
using namespace std;
enum cinsiyet {BELIRTILMEMIS,KADIN,ERKEK};
struct ogrenci {
    unsigned yas;
    int cinsiyet;
    float kilo;
    unsigned boy;
};
typedef struct ogrenci Ogrenci;
int main() {
    Ogrenci ogrenciler[30];
    cout << "Yaşı Giriniz:";
    cin >> ogrenciler[0].yas;
    cout << "Cinsiyet Giriniz (0-1-2):";
    cin >> ogrenciler[0].cinsiyet;
    cout << "Kilo Giriniz:";
    cin >> ogrenciler[0].kilo;
    cout << "Boy Giriniz:";
    cin >>ogrenciler[0].boy;

    cout << "Öğrenci:" << endl << "yaşı:" << ogrenciler[0].yas
        << ", cinsiyeti: " << ogrenciler[0].cinsiyet
        << ", kilosu:" << ogrenciler[0].kilo
        << ", boyu:" << ogrenciler[0].boy << endl;
}
```

Parametre Olarak Yapılar

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

```
#include <iostream>
using namespace std;
enum cinsiyet {BELIRTILMEMIS,KADIN,ERKEK};
struct ogrenci {
    unsigned yas;
    int cinsiyet;
    float kilo;
    unsigned boy;
};
typedef struct ogrenci Ogrenci;
void ogreciYaz(Ogrenci);
void ogrenciOku(Ogrenci*);
int main() {
  Ogrenci ogrenciler[30];
  ogrenciOku(&ogrenciler[0]);
  ogreciYaz(ogrenciler[0]);
}
void ogreciYaz(Ogrenci pOgrenci) {
    cout << "Öğrenci:" << endl << "yaşı:" << pOgrenci.yas
         << ", cinsiyeti: " << pOgrenci.cinsiyet
         << ", kilosu:" << pOgrenci.kilo
         << ", boyu:" << pOgrenci.boy << endl;
}
void ogrenciOku(Ogrenci* pOgrenci) {
    cout << "Yaşı Giriniz:";
    cin >> pOgrenci->yas;
    cout << "Cinsiyet Giriniz (0-1-2):";
    cin >> pOgrenci->cinsiyet;
    cout << "Kilo Giriniz:";
    cin >> pOgrenci->kilo;
```

```
    cout << "Boy Giriniz:";
    cin >> pOgrenci->boy;
}
```

Anonim Yapılar

Anonim yapı, kimliği veya `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 <iostream>
using namespace std;
enum cinsiyet {BELIRTILMEMIS,KADIN,ERKEK};
struct ogrenci {
    unsigned yas;
    int cinsiyet;
    float kilo;
    unsigned boy;
    struct {
        int gun;
        int ay;
        int yil;
    };
};
typedef struct ogrenci Ogrenci;
void ogreciYaz(Ogrenci);
int main() {
  Ogrenci ogrenci={ 45,KADIN, 65.0, 180, {01, 02, 2000} };
  ogreciYaz(ogrenci);
}
void ogreciYaz(Ogrenci pOgrenci) {
    cout << "Öğrenci:" << endl << "yaşı:" << pOgrenci.yas
         << ", cinsiyeti: " << pOgrenci.cinsiyet
         << ", kilosu:" << pOgrenci.kilo
         << ", boyu:" << pOgrenci.boy
         << ", tarihi:" << pOgrenci.gun << "-" << pOgrenci.ay << "-" << pOgrenci.yil
         << endl;
}
```

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. Dinamik veri yapıları yapısal programlama sürecinde çokça geliştirilen ve artık standart hale gelen koleksiyonlardır. C++ dilinde koleksiyonlar, artık hazır bir kütüphanedir ve *Konteyner Şablonları* başlığında incelenecektir.

```
#include <iostream>
using namespace std;
enum cinsiyet {BELIRTILMEMIS,KADIN,ERKEK};
struct ogrenci {
    unsigned yas;
```

```
    int cinsiyet;
    float kilo;
    unsigned boy;
    struct ogrenci* sonrakiOgrenci;
};
typedef struct ogrenci Ogrenci;
void ogrecileriYaz(Ogrenci*);
int main() {
  Ogrenci ogrenci1={ 45,KADIN, 65.0, 165, nullptr };
  Ogrenci ogrenci2={ 55,ERKEK, 85.5, 170, nullptr };
  Ogrenci ogrenci3={ 25,ERKEK, 65.5, 155, nullptr };
  ogrenci1.sonrakiOgrenci=&ogrenci2;
  ogrenci2.sonrakiOgrenci=&ogrenci3;
  ogrecileriYaz(&ogrenci1);
}
void ogrecileriYaz(Ogrenci* pOgrenci) {
    while(pOgrenci!=nullptr) {
        cout << "Öğrenci:" << counter++ << endl << "yaşı:" << pOgrenci->yas
             << ", cinsiyeti: " << pOgrenci->cinsiyet
             << ", kilosu:" << pOgrenci->kilo
             << ", boyu:" << pOgrenci->boy << endl;
        pOgrenci=pOgrenci->sonrakiOgrenci;
    }
}
/* Program Çıktısı:
Öğrenci:0
yaşı:45, cinsiyeti: 1, kilosu:65, boyu:165
Öğrenci:1
yaşı:55, cinsiyeti: 2, kilosu:85.5, boyu:170
Öğrenci:2
yaşı:25, cinsiyeti: 2, kilosu:65.5, boyu:155

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

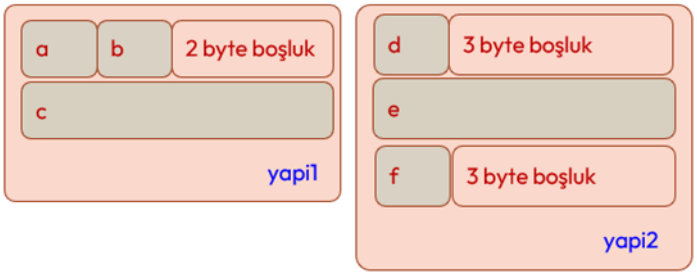
Yapı Dolgusu

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 <iostream>
using namespace std;
struct yapi1 {
   char a;
   char b;
   int c;
};
struct yapi2 {
   char d;
   int e;
   char f;
};
int main() {
   cout << "char bellek miktarı:" << sizeof(char) << endl;
        // char bellek miktarı: 1
   cout << "int  bellek miktarı:" << sizeof(int) << endl;
        // int  bellek miktarı: 4
   cout << "------------------------" << endl;
   cout << "Her iki yapı için olmasi gereken bellek miktarı:"
```

```
        << 2*sizeof(char)+sizeof(int) << endl;
        // Her iki yapı için olmasi gereken bellek miktarı: 6
   cout << "yapi1 için bellekte ayrılan miktar:" << sizeof(struct yapi1) << endl;
        // yapi1 için bellekte ayrılan miktar: 8
   cout << "yapi2 için bellekte ayrılan miktar:" << sizeof(struct yapi2) << endl;
        // yapi2 için bellekte ayrılan miktar: 12
}
```

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



Şekil 31. 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 kaynak koda eklemektir.

```
#include <iostream>
using namespace std;
#pragma pack(1)
struct yapi1 {
   char a;
   char b;
   int c;
};
struct yapi2 {
   char d;
   int e;
   char f;
};
int main() {
   cout << "char bellek miktarı:" << sizeof(char) << endl;
         // char bellek miktarı: 1
   cout << "int  bellek miktarı:" << sizeof(int) << endl;
         // int  bellek miktarı: 4
   cout << "------------------------" << endl;
   cout << "Her iki yapı için olmasi gereken bellek miktarı:"
        << 2*sizeof(char)+sizeof(int) << endl;
         // Her iki yapı için olmasi gereken bellek miktarı: 6
   cout << "yapi1 için bellekte ayrılan miktar:" << sizeof(struct yapi1) << endl;
         // yapi1 için bellekte ayrılan miktar: 6
   cout << "yapi2 için bellekte ayrılan miktar:" << sizeof(struct yapi2) << endl;
         // yapi2 için bellekte ayrılan miktar: 6
}
```

Eğer yalnızca belirlenen yapı için bunun yapılması isteniyorsa yapı tanımına `__attribute__((packed))` özelliği eklenir;

```
#include <iostream>
using namespace std;
struct yapi1 {
   char a;
   char b;
```

```
   int c;
};
struct __attribute__((packed))  yapi2 {
   char d;
   int e;
   char f;
};
int main() {
   cout << "char bellek miktarı:" << sizeof(char) << endl;
         // char bellek miktarı: 1
   cout << "int  bellek miktarı:" << sizeof(int) << endl;
         // int  bellek miktarı: 4
   cout << "------------------------" << endl;
   cout << "Her iki yapı için olmasi gereken bellek miktarı:"
        << 2*sizeof(char)+sizeof(int) << endl;
         // Her iki yapı için olmasi gereken bellek miktarı: 6
   cout << "yapi1 için bellekte ayrılan miktar:" << sizeof(struct yapi1) << endl;
         // yapi1 için bellekte ayrılan miktar: 8
   cout << "yapi2 için bellekte ayrılan miktar:" << sizeof(struct yapi2) << endl;
         // yapi2 için bellekte ayrılan miktar: 6
}
```

Birlikler

Birlikler (union), Pascal dilindeki record case talimatına (statement) benzer. Birlik (union), yapı (struct) gibi tanımlanır.

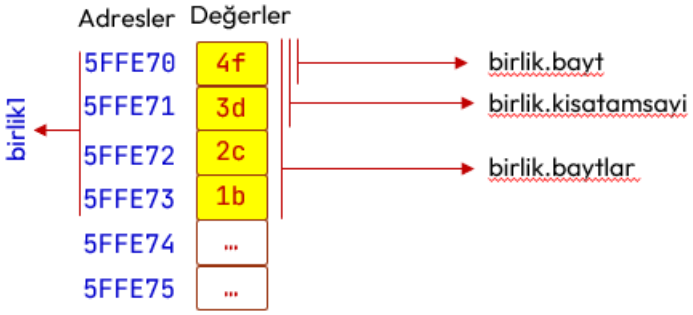
```
union yapı-kimliği {
    veri-tipi yapı-elemanı-kimliği1;
    veri-tipi yapı-elemanı-kimliği1;
    ...
} [değişken-kimliği1][,değişken-kimliği2];
```

Aralarındaki fark yapı elemanlarının her birine ayrı bellek bölgesi ayrılırken, birlik üyelerinin her biri aynı bellek bölgesini paylaşırlar. Birliğin belek boyutu, elemanlarından en fazla bellek kaplayanı kadardır. Aşağıda üyelerinin aynı bellek bölgesini paylaştığı görülmektedir.

```
#include <iostream>
#include <iomanip> //setbase()
using namespace std;
union tamsayiBirligi {
    char baytlar[4];
    int  tamsayi;
    char bayt;
    short int kisatamsayi;
} birlik1,birlik2;
int main() {
    union tamsayiBirligi birlik3,birlik4;
    birlik1.tamsayi=0x1B2C3D4F; //16lık sayılarda her çift rakam bir bayt olur
    cout << "sizeof tamsayiBirligi.baytlar:" << sizeof birlik1.baytlar << endl;//4
    cout << "sizeof tamsayiBirligi.bayt:" << sizeof birlik1.bayt << endl;//1
    cout << "sizeof tamsayiBirligi.tamsayi:" << sizeof birlik1.tamsayi
         << endl;//4
    cout << "sizeof tamsayiBirligi.kisatamsayi:" << sizeof birlik1.kisatamsayi
         << endl;//2
    cout << "---------------------------------------" << endl;
    cout << "sizeof tamsayiBirligi:" << sizeof birlik1 << endl;//4
    cout << "---------------------------------------" << endl;
    cout << "birlik1.tamsayi:" << setbase(16) << birlik1.tamsayi
         << endl; //1b2c3d4f
    cout << "birlik1.baytlar:" << setbase(16)
         << int(birlik1.baytlar[0]) << "-"
```

```
        << int(birlik1.baytlar[1]) << "-"
        << int(birlik1.baytlar[2]) << "-"
        << int(birlik1.baytlar[3]) << endl; //4f-3d-2c-1b
    cout << "birlik1.bayt:" << setbase(16) << int(birlik1.bayt) << endl;  //4f
    cout << "birlik1.kisatamsayi:" << setbase(16) << birlik1.kisatamsayi
        << endl; //3d4f
    cout << "-------------------------------------" << endl;
    birlik1.bayt=0x00;
    cout << "birlik1.tamsayi:" << setbase(16) << birlik1.tamsayi; //1b2c3d00
}
```

Şekil olarak da birlik1 değişkeninin bellek yerleşimi aşağıda verilmiştir;



Şekil 32. birlik1 Değişkeninin Bellek Yerleşimi