Struct & Union

Motivasyon

- Bir objenin birden fazla özelliği olabilir.
- Biz bu özellikleri saklamak istediğimizde birden fazla değişken kullanmamız gerekiyor.
- Örneğin, bir öğrencinin özelliklerini tanımlamamız için 5 adet farklı değişkene ihtiyacımız var.

```
char name[41];
long int id;
char department[16];
short int class;
float GPA;
```

- Yapılar, kodunuzu daha iyi organize etmenize yardımcı olur

Motivasyon

- Yapılar (Structures)
  - Bir isimle ilgili değişkenlerin koleksiyonları
  - Farklı veri tiplerinde değişkenler içerebilir
  - Yaygın olarak dosyalarda depolanacak kayıtları tanımlamak için kullanılır
  - İşaretçilerle birlikte, bağlantılı listeler, yığınlar, kuyruklar ve ağaçlar oluşturabilir
  - Türetilmiş veri türleridir; başka türdeki nesneler kullanılarak oluşturulmuştur.

Struct

- Bu yapının kullanımı:

```
struct TipAdı{
        tip deg_ismi;
        tip deg_ismi;
        ...
 }degisken_list;
```

- struct structure yapısının anahtar kelimesi
- TipAdı bir tanımlayıcıdır ve yapı tipinin değişkenlerini bildirmek için struct anahtar sözcüğüyle kullanılır.
- degisken_list opsiyoneldir.

Örnek

- İki öğrencinin tüm bilgilerini struct yapısını kullanarak topladık.
- stu1 ve stu2 değişken isimlerini kullandık.

```
struct stu_info {
   char name[41];
   long int id;
   char dept[16];
   short int class;
   float gpa;
} stu1, stu2;
```

struct Kullanımı

- Struct kullanıcılar tarafından tanımlanan veri türleridir.
- **`stu_info`** bu türün ismi, stu1 ise bu türden bir değişkenin adıdır.
  - **`stu_info`** int gibi bir türün adı
  - **`stu1`** bu tipteki bir değişkenin adı.

Geçerli Operatörler

- Aynı türden bir yapıya yapı atama
- Bir yapının adresini (&) almak
- Bir yapının üyelerine erişim
- Bir yapının boyutunu belirlemek için sizeof operatörünü kullanma

Struct yapısının öğelerine ulaşmak

- Yapı elemanlarına erişmek için iki operatör kullanılır:
  - yapı elemanı operatörü (.) - ayrıca nokta operatörü denir
  - Yapı işaretçisi operatörü (->) - ok operatörü de denir
- Bir yapının alanına aşağıdaki gibi bir nokta (.) kullanarak erişebilirsiniz:

  structure_variable_name.field_name

- Sadece alan adını kullanamazsınız. Bir alan adı sadece bir yapı değişkeni bağlamında anlamlıdır

yani,

id = 123; yanlış (kimlik tanımlanmamış)

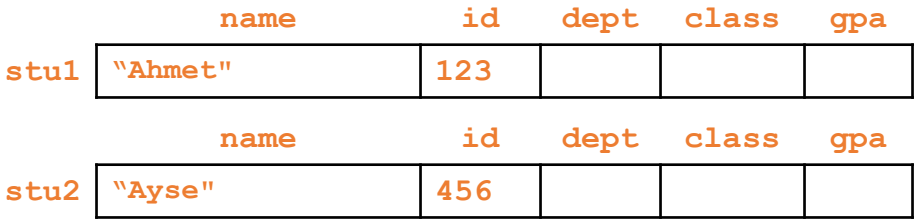
stu1.id = 123; doğru

fig10_02.c

Struct yapısının öğelerini değiştirme

- **stu1** ve **stu2**

- **strcpy(stu1.name, “Ahmet"); stu1.id=123;**
- **strcpy(stu2.name, “Ayse"); stu2.id=456;**



struct Değişkenleri Tanımlama

- İki farklı yolu vardır:

```
struct {
   char name[21];
   int credit;
}cse1142;

/*Define both type
and variables*/
```

```
struct course_type {
      char name[21];
      int credit;
};
/*Define only the type*/

struct course_type cse1142;
 /*Define variables*/
```

student_info.c

struct içinde struct

- Yapı türü alanına sahip bir yapınız olabilir.
- Örnek:

```
struct A_type {
   int m, n;
};
```

```
struct B_type {
   int f;
   struct A_type g;
} t;
```

- **t** iki alana sahiptir: f ve g. f tipi int, g ise struct A_type tipinde. t aşağıdaki alanlara ve alt alanlara sahiptir:

```
t.f
t.g
t.g.m
t.g.n
```

struct içinde struct

- Bir yapı kendini bir alan olarak alamaz ama işaretçi olarak alabilir
- Örnek

```
struct B_type {
   int f;
   struct B_type g;      //ERROR
   struct B_type *bPtr; // OK
} t;
```

Örnek – 2 boyutlu nokta

- 2 boyutlu uzayda bir nokta için bit struct tanımlayın

```
struct point_type {
   int x, y;
};
```

- Bu nokta tipinde A ve B değişkenleri tanımlayın.

```
struct point_type A, B;
```

- Bu değişkenlere değer atayın

```
A.x=2; A.y=3;
B.x=1; B.y=2;
```

Örnek – Dikdörtgen ve Üçgen

- Üçgen için struct tanımlayın

```
struct triangle_type {
   struct point_type A, B, C;
};
```

- Üçgen tipinde t değişkeni tanımlayın ve ilk değerlerini atayın.

```
struct triangle_type t={{1,3},{2,4},{1,6}};
```

- Dikdörtgen için struct tanımlayın

```
struct rectangle_type {
   struct point_type A, B, C, D;
};
```

struct değişkenlere ilk değer atama

- Tanımlama sırasında değer atayabilirsiniz:

```
struct A_type {
   int m, n;
} k={10,2};

struct B_type {
   int f;
   struct A_type g;
} t={5,{6,4}};
```

- Bu tarz bir atama yanlıştır

```
struct A_type {
   int m=10, n=2;
} k;
```

struct değişkenlere ilk değer atama

- Aşağıdaki kullanım da yanlıştır:

```
struct A_type {
   int m, n;
 } k;
 ...
 k={10,2};
```

Fonksiyon ile struct kullanma

- Parametre olarak struct
- Geri dönen değer olarak struct

Parametre olarak struct

- struct parametreniz olabilir.

```
void func1(struct A_type r)
{   struct A_type s;
    s=r;
    s.m++;
}
```

r->m
yazamayız

- pointer da olabilir:

```
void func2(struct A_type *p)
{   (*p).m=10;
    (*p).n=3;
}
```

(*p).m=10
yerine
p->m=10
kullanılabilir

Örnek– Struct Pointer

```
void func2(struct A_type *h)
{
    (*h).m=5; /* Equivalent of "h->m=5;"*/
}

int main()
{   struct A_type k={1,2};
    func2(&k);
    printf("%d  %d\n", k.m, k.n);
}
```

function_ex1.c

struct as the Return Type

- Geri dönen değer olarak struct

```
struct A_type func4()
{   struct A_type s={10,4};
    ...
    return s;
}
```

Örnek

```
struct complex {
    float real;
    float imaginary;
} c={5.2,6.7}, d={3,4};

struct complex add(struct complex n1, struct
 complex n2){
    struct complex r;
    r.real = n1.real + n2.real;
    r.imaginary = n1. imaginary + n2.
 imaginary;
    return r;
}
```

complex.c

Struct diziler

Diğer tiplerdeki diziler gibi.

Metotlarda call-by-reference

```
struct stu_info {
   char name[41];
   long int id;
   char dept[16];
   short int class;
   float gpa;
} stu1, stu2;

struct stu_info   class[100];
int    number[100];

number[3]     = 42;
class[3].id   = 42;
```

Örnek- Küp

- Nokta tanımlama

```
struct point_type {
   int x, y;
};
```

- Dikdörtgen tanımlama

```
struct triangle_type {
   struct point_type A, B, C;
};
```

- Küp tanımlama

```
struct cube_type {
   struct point_type corner[8];
};
```

Örnek

- 10 öğrencinin ortalamasını hesaplayan C kodunu yazınız

calculate_avg.c

struct boyutu

```
struct A {
    short int m;
    int n;
    char k;
};
```

- Sisteminize bağlı olarak

2+4+1 = 7

typedef

- typedef kullanarak mevcut tipler için yeni isimler tanımlayabilirsiniz.
- `typedef` yeni bir tip yaratmaz

```
typedef existing_type_name new_type_name(s);
```

- Örnek

```
typedef int tamsayi, int_arr[10];
```

- Şimdi, aşağıdakileri yazabiliriz:

```
tamsayi i, j, arr[50];
int_arr a;
i=10;   j=35;   arr[3]=17;
a[2]=15;
```

typedefEx.c

typedef

- Typedef yapı adının önünde "struct" kelimesini kullanmaktan kaçınmak için kullanılır.
- Örnek

```
typedef struct A_type  A_t;
A_t var1;

typedef struct {
   int x, y;
} nokta_t, noktalar_t[10];

nokta_t n;
noktalar_t N;
n.x = 5;
N[4].x = 8;
```

typedefEx2.c

union

- Bir programda veya fonksiyonda farklı türde değişkenlerin aynı bellek alanını paylaşması için ortaklık bildirimi union deyimi ile yapılır.
- Bu da belleğin daha verimli kullanılmasına imkan verir.
- Bu tipte bildirim yapılırken struct yerine union yazılır.

union

```
union paylas{
    float f;
    int   i;
    char  kr;
 };
```

union

- Hafıza en geniş türü tutacak şekilde ayrılır.

```
union Data
{
   int i;
   float f;
   char  str[20];
};

int main( )
{
   union Data data;

   printf( "Memory size occupied by data : %d\n", sizeof(data));

   return 0;
}
```

http://www.tutorialspoint.com/cprogramming/c_unions.htm

union x ve y nin aynı bellek alanını işgal ettiğinin kanıtı

```
/* union1 */
#include <stdio.h>

    union paylas{
           int x;
           int y;
    }z;
    main()
    {
    int *X,*Y;
    z.x = 10;
    X = &z.x;
    printf("\nTamsayı(x) : %d - bellek adresi %X",z.x,X);
    z.y = 20;
    Y = &z.y;
    printf("\nTamsayı(y) : %d - bellek adresi %X",z.y,Y);
}
```

Tamsayı(x) : 10 - bellek adresi DF23

Tamsayı(y) : 20 - bellek adresi DF23

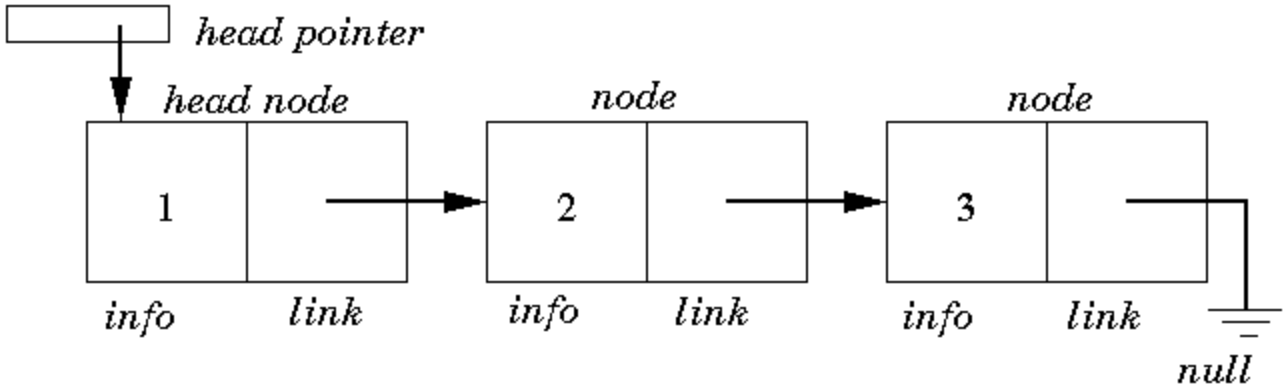
Örnek

```
union number {
    int x;
    double y;
};

int main(void){
    union number value; // define union variable
    value.x = 100; // put an integer into the union
    printf("int:%d  double:%f\n",value.x, value.y);

    value.y = 100.0; // put a double into the same union
    printf("int:%d  double:%f\n",value.x, value.y);
}
```

Örnek – Bağlı liste



A Linked List

Kendinden Referanslı Yapılar

- Kendinden referanslı bir yapı, aynı yapı tipindeki bir yapıya işaret eden bir işaretçi elemanı içerir.

```
struct node {
    int data;
    struct node *nextPtr;
};
```

Dinamik Bellek Ayırma

- Dinamik veri yapılarının oluşturulması ve sürdürülmesi, dinamik bellek ayırma işlemi gerektirir
  - bir programın, yeni düğümleri tutmak için yürütme zamanında daha fazla bellek alanı elde etme ve artık gerekmeyen alanı serbest bırakma yeteneği.
- Malloc ve free işlevler ve sizeof operatörü dinamik bellek ayırma için önemlidir.

Dinamik Bellek Ayırma

Bellek ayırma - Malloc kullanımı

```
newPtr = malloc(sizeof(struct node));
```

Geri bırakma

```
free(newPtr);
```

Bağlı Liste

- Bağlı liste herhangi bir tipten node'ların (düğümlerin) yine kendi tiplerinden düğümlere işaret etmesi (point) ile oluşan zincire verilen isimdir.
- Buna göre her düğümde kendi tipinden bir pointer olacak ve bu pointerlar ile düğümler birbirine aşağıdaki şekilde bağlanacaktır.

http://bilgisayarkavramlari.sadievrenseker.com/2007/05/03/linked-list-linkli-liste-veya-bagli-liste/

Bağlı Liste

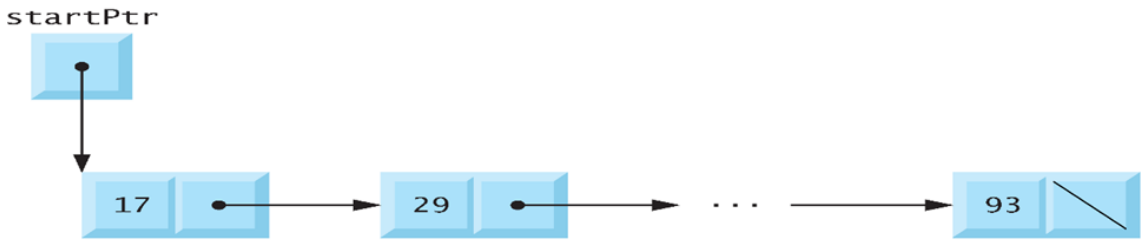


Fig. 12.2 | Linked-list graphical representation.