Algoritmalar ve Programlama I – BLM 103

Hafta 13: C Yapıları

Fenerbahçe Üniversitesi

13. Hafta İçeriği

- Yapılar (Structures)
- Yapı Dizisi
- Dinamik Bellek Tahsisi
  - Dinamik Boyutlu Diziler
- Bağlantılı Liste (Linked List)

Veri Yapıları

- Yeri yapıları, bellekte belirli bir organizasyonda verilerin tutulmasıdır.
  - İlişkili değişkenleri bir arada tutmak
  - Programın daha hızlı koşması

- Diziler bir çeşit veri yapılarıdır.
- Bu derste, 2 tür daha göreceğiz:
- Yapı (`struct`) – C dili tarafından desteklenmektedir.
- Bağlı liste (linked list) – `struct` ve dinamik bellek tahsisi ile birlikte oluşturulur.

C Dilinde Struct'lar

- Farklı veri tipindeki, ilişkili verileri bir arada tutmaya yarar.
  - Diziler sadece aynı veri tipindeki elemanları bir arada tutar.

- Örnek:
  Bir uçak yazılımı yapılacak:

```
char ucusNumarasi[7];
int rakim;
int boylam;
int enlem;
int ucakAdi;
double ucakHizi;
```

- Bu verileri gruplayıp her bir uçak için bu yapı kullanılabilir hale getirilebilir.

Struct Tanımlaması

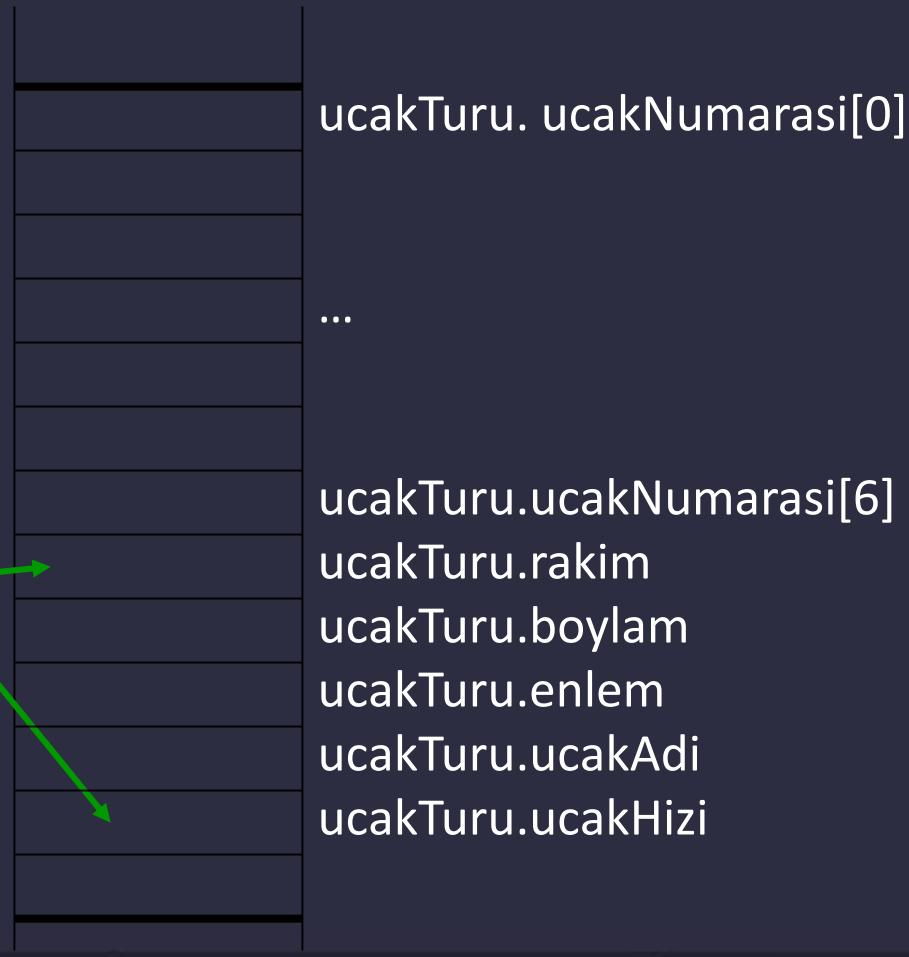
- Struct'ın nasıl olacağının derleyiciye ifade edilmesi aşağıdaki gibidir.

```
struct ucakTuru {
 char ucusNumarasi[7];
 int rakim;
 int boylam;
 int enlem;
 int ucakAdi;
double ucakHizi;
};
```

- Derleyiciye ucakTuru struct'ının elemanları ve bellekte ne kadar yer kaplayacağı hakkında bilgi verilmiş oldu.
- Ancak henüz bellekte bir yer tutulmadı.

Struct Tanımaması ve Kullanımı

- ucakTuru struct tipinde bir değişken tanımlandı ve bellekte yer ayrıldı
- `struct ucakTuru Boeing777;`
- Bu struct'ın elemanlarına tek tek erişim yapılabilir
- `ucakTuru.ucakHizi = 800.0;`
  `ucakTuru.rakim = 10000;`
- Struct'ın elemanları, stuct'ın içerisindeki tanımlanma sırasına göre bellekte yerleştirilirler



Aynı Anda Tanımlama ve Deklare Etmek

```
struct ucakTuru {
 char ucusNumarasi[7];
 int rakim;
 int boylam;
 int enlem;
 int ucakAdi;
double ucakHizi;
} boeing737;
```

- Aynı zamanda farklı struct'larda tanımlamabilir
- `struct ucakTuru a380;`

Veritipi Tanımlama - typedef

- C dili kendi veri tipinizi tanımlamanıza izin vermektedir.

- Yazım:
- `typedef <tip> <isim>;`
- Örnekler:
- `typedef int test;`
- `typedef struct ucakTuru airbus;`
-
```
typedef struct ogrenci {
  int a;
  double b;
} ornek1;
```

Typedef Kullanımı

- Kodun okunmasını kolaylaştırmaktadır. Uygulamaya özel veri tipleri tanımlamayı sağlar

```
ogrenci ogrenciler[500];
Flight plane1, plane2;
```

- Bu tür tanımlamalar programın header bölümüne konulabilir. Bu yaratılmış veritipleri .c uzantılı dosyada kullanılır. Veritiplerinde bir değişiklik yapılacağında .c uzantılı dosyada hiçbir değişiklik yapmaya gerek olmadan, sadece header değiştirilerek kullanılabilir.

Struct Dizileri

- Struct'lar diziler halinde tanımlanabilirler:
- ```
  KIMLIK insanlar[100];
  ```

- Dizideki her bir elemana ayrı indeks ile erişim yapılır.
- ```
  insanlar[34].kimlikNo = 41423159121O6;
  ```

- Aynı ifade
- ```
  (insanlar[34]). kimlikNo = 41423159121O6;
  ```
  ```

Struct'ların Fonksiyonlara Arguman Olarak Gönderilmesi

- Struct'lar fonksiyona gönderildiklerinde adresleri değil, kopyaları gönderilir. Bellekte yeni bir yer açılır.

- Kopyasının değil, kendisi ile çalışmak isteniyorsa, işaretçiler kullanılmalıdır.

```
int carpisma(Ucak *ucakA, Ucak *ucakB)
{
  if (ucakA->yukseklik == ucakA-> yukseklik) {
     ...
  }
  else
    return 0;
}
```

Union’lar

- Struct’lar gibi elemanları bir arada gruplamak için kullanılırlar, ancak sadece gruptaki bir elemanın aynı anda değeri olabilir.

- union {

  jet jetu;
  helikopter helikopteru;
  kargoucagi kargoucagiu;
  } hava;

Union’lar

```
#include <stdio.h>
#include <string.h>

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

int main( ) {

   union Data data;

   data.i = 10;
   data.f = 220.5;
   strcpy( data.str, "C Programming");

   printf( "data.i : %d\n", data.i);
   printf( "data.f : %f\n", data.f);
   printf( "data.str : %s\n", data.str);

   return 0;
}
```

Dinamik Bellek Tahsisi - malloc

- Bellekte ardışık olarak N adet byte yer ayırmak için kullanılır.

- ```
  void *malloc(int numBytes);
  ```

- Tahsis edilmiş olan yerin başlangıç adresini döndürür.
- Bellekte ayrılan alana "Heap" denmektedir.
  ```

Malloc Kullanımı

- Malloc fonksiyonunu kullanmak için kaç byte yer ayrılacağına karar verilmelidir. Sizeof operatörü, verilen veritipinin boyutunun ne kadar olduğunu geri döndürmektedir.

- ```
  ucaklar = malloc(n * sizeof(UCAK));
  ```

- Malloc'un dönecek olan işaretçi türünü, aşağıdaki gibi UCAK işaretçisine dönüştürülebilir. Bu işleme "casting" denmektedir.

- ```
  ucaklar = (UCAK *) malloc(n* sizeof(UCAK));
  ```

Örnek

```
int ucakSayisi;
UCAK *ucaklar;

printf("Havada kac uçak var?");
scanf("%d", & ucakSayisi);

ucaklar = (UCAK *) malloc(sizeof(UCAK) * ucakSayisi);
if (ucaklar == NULL) {
  printf("Yer tahsisinde hata.\n");
  ...
}
ucaklar[0].yukseklik = ...
```

Eğer tahsis işlemi başarısız olursa, Geriye NULL karakteri dönmektedir.

Not: Bir pointer'da dizi notasyonu da kullanılabilir.

Serbest Bırakma - free

- Veriye ihtiyaç kalmadığında, bellekten atılması için free fonksiyonu kullanılabilir

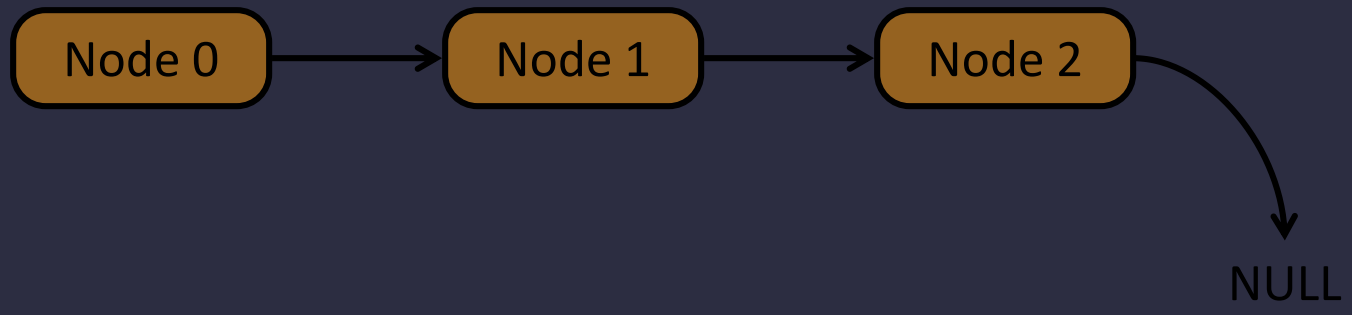
- Malloc ile tahsis edilmiş adresin işaretçisi, free fonksiyonuna verilmelidir.

- ```
  void free(void*);
  ```

- Free ile bellek boşaltılmadığında, bellekte yeterince yer kalmadığında uygulama kapanabilir.
  ```

Bağlı Liste (Linked List) Yapısı

- Bir bağlı liste, işaretçiler kullanılarak birbirlerini gösteren yapılardır.
  - Her bir elemana düğüm (node) denmektedir.
  - Her node bir sonrakini göstermektedir.
  - İlk nod baş, son nod ise kuyruk olarak ifade edilmektedir.



Bağlı Liste vs Diziler

- Bir bağlı listenin N. Elemanına gitmek için, başlangıçtan itibaren n adet düğümün geçilmesi gerekmektedir..
- Ancak diziler bellekte sıralı olarak tutuldukları için, dizinin başlangıç elemanı + N . Adrese doğrudan ulaşılabilir.
- Bağlı listelerin avantajları:
  - Dinamik boyutludur, listeden ekleme ve çıkartma çalışma zamanında yapılabilir
  - Listenin ortasından ekleme ve çıkartma işlemi kolaydır.
- Dizilerin avantajı:
  - Dizinin herhangi bir elemanına hızlı erişim

Örnek: Park Alanı

- Park alanına parketmiş araçlar için bir veri tabanı oluşturulacak
- Bu veri tabanında aşağıdaki işlemler desteklenmesi gerekmektedir.
  - Bir aracın aranması
  - Yeni araç eklenmesi
  - Mevcut bir aracın silinmesi

- Veritabanı araçları seri numaralarına göre sıralı olmalıdır.
- Kaç arabanın olacağı baştan belli olmadığı için, bağlı liste kullanımı uygundur.

Araba Veri Yapısı

- Her araba aşağıdaki bilgileri taşımaktadır:
  seriNo, model, yaş, kilometre, fiyat.

- `typedef struct aracTuru Araba;`

```
struct aracTuru {
  int seriNo;
  char model[20];
  int yas;
  int kilometre;
  double fiyat;
  Araba *sonraki; /* sonraki arabanın adresi*/
}
```

Listeyi Tarama

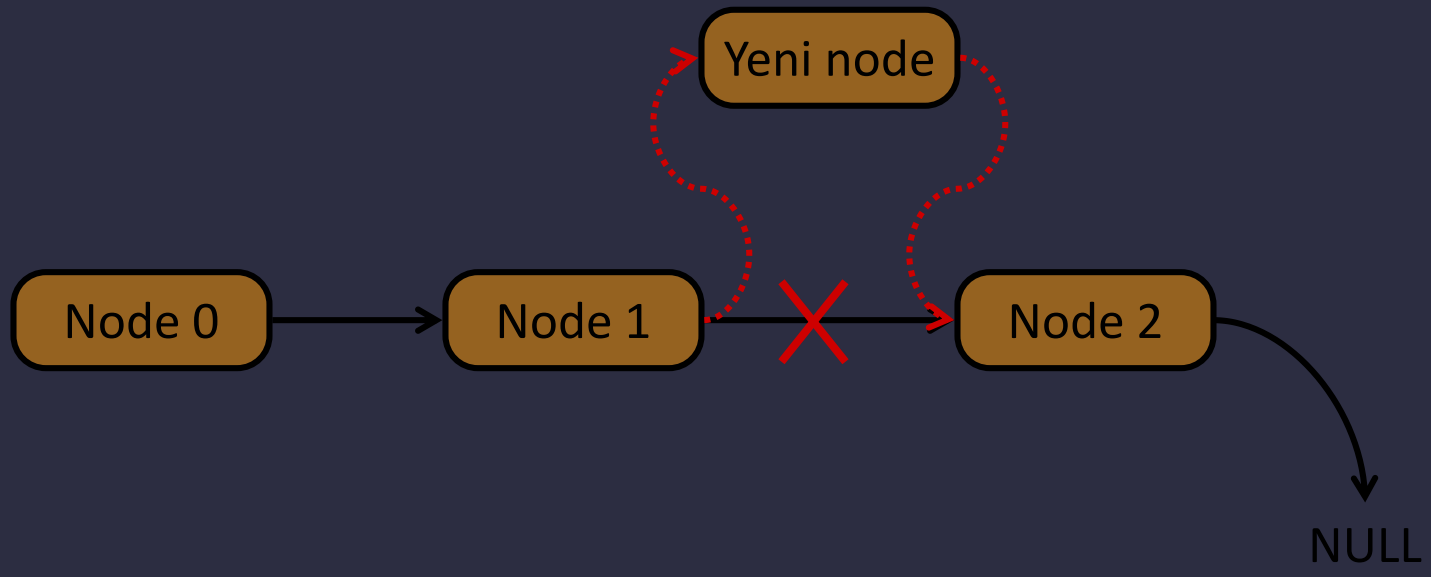
- Araba, ekleme ve silme gibi işlemler için mevcut düğüm (node)'in bulunmasını gerektirmektedir. Aradığımız seriNo'u bulana kadar listede devam edilmelidir.
-

```
Araba* ListeyiAra(Araba *ilkArac, int arananNo)
{
  Araba *onceki, *suanki;
 onceki = ilkArac;
 suanki = ilkArac->sonraki;

  while ((suanki!=NULL) && (suanki->seriNo != arananNo)) {
     onceki = suanki;
     suanki = suanki->sonraki;
  }
  return suanki;
}
```

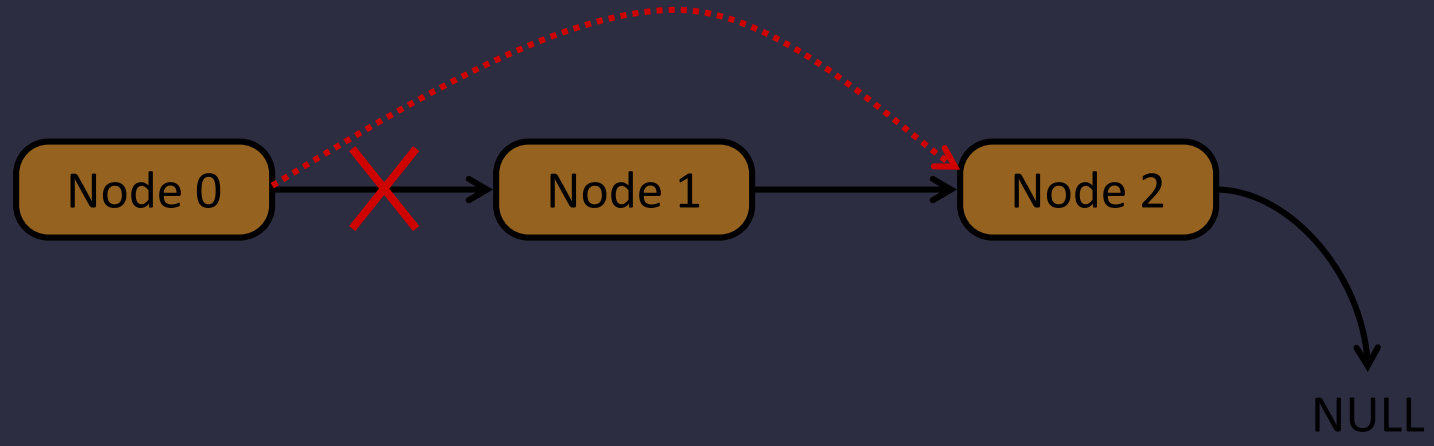
Düğüm Ekleme

- Araya yeni bir nod eklemek için, eklenecek nodların arasına gerekli bağlamalar yapılmalıdır. Bunun için node 1'in sonraki adresine, yeni node, yeni node'in sonraki adresine ise node 2 yazılmalıdır.



Düğüm Silme

- Silinecek olan nod'un öncesiki nodunun gösterdiği sonraki nod adresine, bir sonraki nod'un adresi verilmelidir.



Bağlı Listeler ile Kurulabilecek Yapılar

- Gelecek derslerde gösterilecek olan:

  - Ağaçlar
  - Hash tabloları
  - Graphlar

  yapılabilir.