6 Bağlı Listeler (Linked Lists)

Liste (*list*) sözcüğü aralarında bir biçimde öncelik-sonralık ya da altlık-üstlük ilişkisi bulunan veri öğeleri arasında kurulur. Doğrusal Liste (Linear List) yapısı yalnızca öncelik-sonralık ilişkisini yansıtabilecek yapıdadır. Liste yapısı daha karmaşık gösterimlere imkan sağlar. Listeler temel olarak tek bağlı ve çift bağlı olmak üzere ikiye ayrılabilir. Ayrıca listelerin dairesel veya doğrusal olmasına göre de bir gruplandırma yapılabilir. Tek bağlı listelerde node'lar sadece bir sonraki node ile bağlanarak bir liste oluştururlar. Çift bağlı (iki bağlı) listelerde ise bir node'da hem sonraki noda hem de önceki noda bir bağlantı vardır. Bu bağlantılar Forward Link (ileri bağ) ve Backward Link (geri bağ) olarak adlandırılırlar. Doğrusal listelerde listede bulunan en son node'un başka hiçbir noda bağlantısı yoktur. Bağ değeri olarak NULL alırlar. Dairesel listelerde ise en sondaki node, listenin başındaki node'a bağlanmıştır. Aşağıda buna göre yapılan sınıflandırma görülmektedir.

- Tek Bağlı Listeler (One Way Linked List)
    - Tek Bağlı Doğrusal Listeler (One Way Linear List)
    - Tek Bağlı Dairesel Listeler (One Way Circular List)
- Çift Bağlı listeler (Double Linked List)
    - Çift Bağlı Doğrusal Listeler (Double Linked Linear List)
    - Çift Bağlı Dairesel Listeler (Double Linked Circular List)

İlerleyen kısımlarda bu listeler ve bunlarla ilgili program parçaları anlatılacaktır. Şimdilik bilinmesi gereken ayrıntı, çift bağlı listelerde `previous` adında ikinci bir pointer daha vardır. Diğerleri aynı yapıdadır.

Bağlı Listeler İle İşlemler

Bağlı listeler üzerinde;

1- Liste oluşturma,
2- Listeye eleman eklemek,
3- Listeden eleman silmek,
4- Arama yapmak,
5- Listenin elemanlarını yazmak,
6- Listenin elemanlarını saymak. vb. gibi işlemler yapılabilir.

6.2 Tek Bağlı Doğrusal Listeler

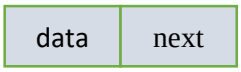
Tek bağlı doğrusal liste, öğelerinin arasındaki ilişki (*Logical Connection*)'ye göre bir sonraki öğenin bellekte yerleştiği yerin (*Memory Location*) bir gösterge ile gösterildiği yapıdır. Bilgisayar belleği doğrusaldır. Bilgiler sıra sıra hücrelere saklanır. Her bir bilgiye daha kolay ulaşmak için bunlara numara verilir ve her birine node adı verilir. Data alanı, numarası verilen node'da tutulacak bilgiyi ifade eder. Next (*link*) alanı ise bir node'dan sonra hangi node gelecekse o node'un bellekteki adresi tutulur.

Tek bağlı listelerin genel yapısı aşağıda verilmiştir. Konu anlatılırken daima bu temel yapı kullanılacağından unutmamalısınız.

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

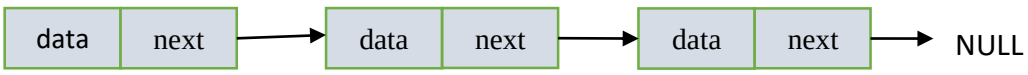
Bağlı listeler içerisindeki düğümlerin yukarıdaki tanımlamayla iki öğesinin olduğu görülüyor. Birinci öğe olan `data`, her türden veri içerebilir, örneğin telefon numaraları, TC kimlik numaraları vb. gibi. Biz `int` türden bir nesneyi yeterli bulduk. İkinci öğe olan `next`, bir bağlı listede mutlaka bulunması gereken bir öğedir. Dikkat edilirse `struct node` tipinde bir işaretçidir.

Tek Bağlı Listelerde bir node'un mantıksal yapısı:



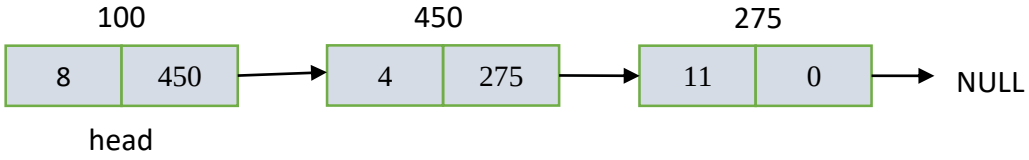
Tek bağlı listelerin yapısında bulunan `node` tipindeki `*next` işaretçisi, rekürsif fonksiyonlarla karıştırılmamalıdır. Burada `next`, bir sonraki `node` türden düğümün adresini gösterecektir.

Tek Bağlı Liste yapısı:



Alttaki şekilde her çiftli kutucuk liste yapısını temsil etmektedir. Bu kutucukların üstündeki numaralar ise bellekte bulundukları yerin adresidir. Burada önemli olan, `next` göstericisinin değerlerinin, yani düğüm adreslerinin sıralı olmayışıdır. İlk düğümün adresi 100, ikincisinin 450 ve üçüncüsünün ise 275'tir. Yani bellekte neresi boşsa o adresi almışlardır. Oysa dizilerde tüm elemanlar sıralı bir şekilde bellekte yer kaplıyordu.

Liste Yapısı ve Bellekteki Adreslerinin Mantıksal Gösterimi



Listenin ilk elemanı genellikle `head` olarak adlandırılır. `head`'den sonra diğer elemanlara erişmek kolaydır. Bazı kaynaklarda listenin sonundaki elemanın ismi `tail` olarak adlandırılmıştır.

6.2.1 Tek Bağlı Doğrusal Liste Oluşturmak ve Eleman Eklemek

Bu örnekte ilk olarak listeyi oluşturacağız, ardından eleman ekleme yapacağız.

```
main() {
     struct node *head; // henüz bellekte yer kaplamıyor
     head = (struct node *)malloc(sizeof(struct node));
     // artık bellekte yer tahsis edilmiştir.
     head -> data = 1;
     head -> next = NULL;
     /* listeye yeni eleman ekleme */
     /* C++'ta head -> next = new node() şeklinde kullanılabilir. */
     head -> next = (struct node *)malloc(sizeof(struct node));
     head -> next -> data = 3;
     head -> next -> next = NULL;
}
```

Peki, eleman eklemek istersek sürekli olarak `head->next->next...` diye uzayacak mı? Tabi ki hayır! Elbette ki bunu yapmanın daha kolay bir yolu var.

6.2.2 Tek Bağlı Doğrusal Listenin Başına Eleman Eklemek

Bir fonksiyonla örnek verelim. Siz isterseniz `typedef` anahtar sözcüğünü kullanarak sürekli `struct` yazmaktan kurtulabilirsiniz fakat akılda kalıcı olması açısından `struct`'ı kullanmaya devam edeceğiz.

```
// Tek bağlı doğrusal listenin başına eleman eklemek
struct node *addhead(struct node *head,int key) {
     struct node *temp = (struct node *)malloc(sizeof(struct node));
     temp -> data = key;
     temp -> next = head; // temp'in next'i şu anda head'i gösteriyor.
```

```
    /* Bazen önce listenin boş olup olmadığı kontrol edilir, ama gerekli
    değil aslında.
    Çünkü boş ise zaten head=NULL olacaktır ve temp olan ilk düğümün
    next'i NULL gösterecektir. */
    head = temp; /* head artık temp'in adresini tutuyor yani eklenen
    düğüm listenin başı oldu. */
    return head;
}
```

6.2.3 Tek Bağlı Doğrusal Listenin Sonuna Eleman Eklemek

Listenin sonuna ekleme yapabilmek için liste sonunu bilmemiz gerekiyor. Listede eleman olduğunu varsayıyoruz. Liste sonunu bulabilmek içinse bu liste elemanları üzerinde tek tek ilerlemek gerektiğinden `head`'in adresini kaybetmemek için bir değişken oluşturacağız ve `head`'in adresini bu değişkene atayacağız.

```
struct node *addlast(struct node *head,int key) {
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    /* C++'ta struct node *temp = new node(); * şeklinde
    kullanılabileceğini unutmayınız. */
    temp -> data = key;
    temp -> next = NULL;
    struct node *temp2 = head;
    /* Aşağıdaki while yapısı traversal(dolaşma) olarak adlandırılır */
    while(temp2 -> next != NULL)
        temp2 = temp2 -> next;
    temp2 -> next = temp;
    return head;
}
```

6.2.4 Tek Bağlı Doğrusal Liste Elemanlarının Tüm Bilgilerini Yazdırmak

Listedeki elemanların adreslerini, data kısımlarını ve sonraki düğüm adresini ekrana basan `listinfo` ismindeki fonksiyon aşağıdaki gibi yazılabilir.

```
void listinfo(struct node* head) {
    int i = 1;
    while(head != NULL) {
        printf("%d. Dugumunun Adresi = %p \n", i, head);
        printf("%d. Dugumunun verisi = %d \n", i, head -> data);
        printf("%d. Dugumunun Bagli Oldugu Dugumun Adresi= %p\n\n",i,
    head->next);
        head = head -> next;
        i++;
    }
```

```
}
```

6.2.5 Tek Bağlı Doğrusal Listenin Elemanlarını Yazdırmak

Fonksiyon sadece ekrana yazdırma işi yapacağından `void` olarak tanımlanmalıdır. Liste boşsa ekrana listede eleman olmadığına dair mesaj da verilmelidir.

```
void print(struct node *head) {
     if(head == NULL) {
          printf("Listede eleman yok");
          return;
     }
     struct node *temp2 = head;
     while(temp2!= NULL) { // temp2->next!=NULL koşulu olsaydı son düğüm
     yazılmazdı
          printf("%d\n", temp2 -> data);
          temp2 = temp2 -> next;
     }
}
```

Şüphesiz elemanları yazdırmak için özyinelemeli bir fonksiyon da kullanılabilirdi.

```
//Tek bağlı liste elemanlarını özyinelemeli yazdırmak
void print_recursive(struct node *head) {
     if(head == NULL)
          return;
     printf("%d\t", head -> data);
     print_recursive (head -> next);
}
```

SORU: Yukarıdaki fonksiyon aşağıdaki gibi yazılırsa çıktısı ne olur.

```
void print_recursive2(struct node *head) {
     if(head == NULL)
          return;
     print_recursive2 (head -> next);
     printf("%d\t", head -> data);
}
```

6.2.6 Tek Bağlı Doğrusal Listenin Elemanlarını Saymak

Listenin elemanlarını saymak için `int` tipinden bir fonksiyon oluşturacağız. Ayrıca listede eleman olup olmadığını da kontrol etmeye gerek yoktur. Çünkü eleman yok ise `while` döngüsü hiç çalışmayacak ve 0 değerini döndürecektir.

```
int count(struct node *head) {
     int counter = 0;
     while(head != NULL) { // head->next!=NULL koşulu olsaydı son düğüm
     sayılmazdı
          counter++;
          head = head -> next;
     }
     return counter;
}
```

Bu işlemi özyinelemeli yapmak istersek:

```
int count_recursive(struct node *head) {
     if (head == NULL)
          return 0;
     return count_recursive(head->next) + 1;
}
```

6.2.7 Tek Bağlı Doğrusal Listelerde Arama Yapmak

Bu fonksiyon ile liste içinde arama yapılmaktadır. Eğer aranan bilgi varsa, bulunduğu node'un adresiyle geri döner. Bu fonksiyon bulundu ise true bulunmadı ise false döndürecek şekilde de düzenlenebilir.

```
struct node* locate(struct node* head, int key) {
     struct node* locate = NULL;
     while(head != NULL)
          if(head -> data != key)
              head = head -> next;
          else {
              locate = head;
              break;
          }
     return(locate);
}
```

6.2.8 Tek Bağlı Doğrusal Listelerde İki Listeyi Birleştirmek

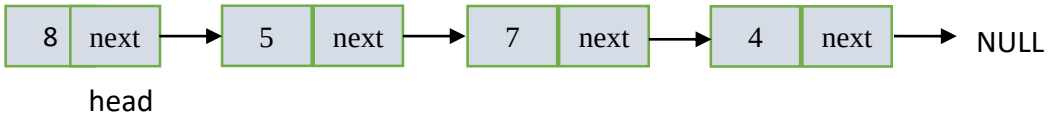
`list_1` ve `list_2` adındaki iki listeyi birleştirmek için `concatenate` fonksiyonunu kullanabiliriz.

```
void concatenate(struct node*& list_1, node* list_2) { // parametrelere
     dikkat
     if(list_1 == NULL)
          list_1 = list_2;
```

```
    else
        last(list_1) -> next = list_2; // last isimli fonksiyona çağrı
    yapılıyor
}
```

6.2.9 Tek Bağlı Doğrusal Listelerde Verilen Bir Değere Sahip Düğümü Silmek

Tek bağlı listenin mantıksal yapısı:



Tek bağlı listelerde verilen bir düğümü silme işlemi, o düğümün başta ya da ortada olmasına göre farklılık gösterir. İlk düğüm silinmek istenirse ikinci elemanın adresini yani `head`'in `next` işaretçisinin tuttuğu adresi `head`'e atayarak baştaki eleman silinebilir. Eğer ortadan bir düğüm silinmek istenirse bir önceki düğümü, silinmek istenen düğümden bir sonraki düğüme bağlamak gerekir. Şimdi `_remove` ismini vereceğimiz bu fonksiyonu yazalım.

```
struct node *remove(struct node *head, int key) {
    if(head == NULL) {
        printf("Listede eleman yok\n");
        return;
    }
    struct node *temp = head;
    if(head -> data == key) { // ilk düğüm silinecek mi diye kontrol
    ediliyor.
    head = head -> next; // head artık bir sonraki eleman.
    free(temp);
    }
    else if(temp -> next == NULL) { // Listede tek düğüm bulunabilir.
        printf("Silmek istediginiz veri bulunmamaktadir.\n\n");
        return head;
    }
    else {
        while(temp -> next -> data != key) {
            if(temp -> next -> next == NULL) {
                printf("Silmek istediginiz veri
    bulunmamaktadir.\n\n");
                return head;
            }
            temp = temp -> next;
        }
        struct node *temp2 = temp -> next;
```

```
            temp -> next = temp -> next -> next;
            free(temp2);
        }
        return head;
    }
```

6.2.10 Tek Bağlı Doğrusal Listelerde Verileri Tersten Yazdırmak

Tek bağlı listenin elemanlarını tersten yazdıran `print_reverse` adlı bir fonksiyon yazalım. Bu fonksiyonu yazarken her veriyi yeni bir listenin başına ekleyeceğiz ve böylece ilk listenin tersini elde etmiş olacağız. Bunun için `addhead` adlı fonksiyonu kullanacağız. `print_reverse fonksiyonunda struct node*` türden yeni bir değişken daha tanımlayacağız ve `head`'in elemanlarını bu yeni listenin sürekli başına ekleyerek verileri ters bir biçimde sıralayacağız ve yazdırma işlemini gerçekleştireceğiz. Aslında tersten yazdırma işini rekürsif olarak yapan bir fonksiyon daha önce SORU olarak sorulmuş fonksiyondur. Rekürsif fonksiyonları iyice kavramanız için bolca örnek yapmalısınız. Çünkü veri yapılarında rekürsif fonksiyonların çok büyük bir önemi vardır.

```
void print_reverse(struct node *head) {
    struct node *head2 = NULL; // yeni listenin başını tutacak adres
    değişkeni
    struct node *temp = head;
    while(temp != NULL) {
        head2 = addhead(head2, temp -> data);
        temp = temp -> next;
    }
    print(head2);
}
```

6.2.11 Tek Bağlı Doğrusal Listenin Kopyasını Oluşturmak

`head`'in kopyası oluşturulup `kopya` geri gönderilmektedir. `kopya` listesinde veriler aynı fakat adresler farklıdır.

```
struct node* copy(struct node* head) {
    struct node* kopya = NULL;
    if(head != NULL)
        do {
           concatenate(kopya, cons(head -> data));
           head = head -> next;
        } while(head != NULL);
    return kopya;
```

```
}
```

6.2.12 Listeyi Silmek

Listelerin kullanımı bittiğinde bunların bellekte yer işgal etmemesi için tüm düğümlerinin silinmesi gereklidir. `head` düğümünün silinmesi listenin kullanılmaz hale gelmesine neden olur ancak `head` den sonraki düğümler hala bellekte yer kaplayama devam eder.

```
struct node *destroy(struct node *head) {
     if(head == NULL) {
          printf("Liste zaten bos\n");
          return;
     }
     struct node *temp2;
     while(head!= NULL) { // while içindeki koşul temp2 -> next, NULL
     değilse
          temp2=head;
          head = head->next;
          free(temp2);
     }
     return head;
}
```

SORU: destroy fonksiyonunu özyinelemeli olarak yazınız.

6.2.13 Main Fonksiyonu İçinde Tanımladığımız Tüm Fonksiyonların Çağırılması

Yukarıda tanımladığımız fonksiyonların çalıştırılması için tüm fonksiyonlar, `struct` tanımlaması dâhil aşağıdaki `main()` fonksiyonu üzerinde yazılır.

```
main(){
     int secim,data;
     struct node *head = NULL;
     while(1){
          printf("1-Listenin Basina Eleman Ekle\n");
          printf("2-Listenin Sonuna Eleman Ekle\n");
          printf("3-Listeyi Gorme\n");
          printf("4-Listeden verilen bir degere sahip dugum silmek\n");
          printf("5-Listeyi sil\n");
          printf("6-Listedeki eleman sayisi\n");
          printf("7-Listenin tum eleman bilgileri\n");
          printf("8-Programdan Cikma\n");
          printf("Seciminiz....?");
          scanf("%d",&secim);
```

```
            switch(secim){
                case 1: printf("Eklemek istediginiz degerini giriniz..?");
                scanf("%d",&data);
                head=addhead(head,data);
                break;
            case 2:
                printf("Eklemek istediginiz degerini giriniz..?");
                scanf("%d",&data);
                head=addlast(head,data);
                break;
            case 3:
                print(head);
                break;
            case 4:
                printf("Silmek istediginiz degerini giriniz..?");
                scanf("%d",&data);
                head=delete(head,data);
                break;
            case 5:
                head=destroy(head);
                break;
            case 6:
                printf("Listede %d eleman vardir\n",count(head));
                break;
            case 7:
                listinfo(head);
                break;
            case 8:
                exit(1);
                break;
            default:
                printf("Yanlis secim\n");
            }
        }
}
```