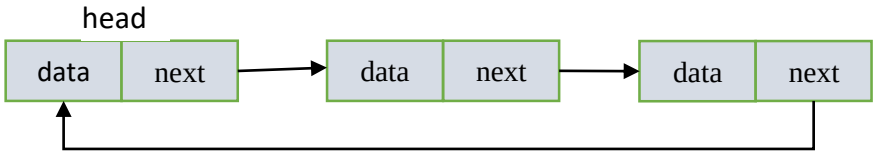
6.2 Tek Bağlı Dairesel (Circle Linked) Listeler

Tek bağlı dairesel listelerde, doğrusal listelerdeki birçok işlem aynı mantık ile benzer şekilde uygulanır; fakat burada dikkat edilmesi gereken nokta, dairesel bağlı listede son elemanının `next` işaretçisi `head`'i göstermektedir.

Dairesel Bağlı Liste Yapısı:



6.2.1 Tek Bağlı Dairesel Listelerde Başa Eleman Eklemek

Tek bağlı listelerde yaptığımızdan farklı olarak `head`'in global olarak tanımlandığı varsayılmıştır. Liste yoksa oluşturuluyor, eğer var ise, `struct node*` türden `temp` ve `last` düğümleri oluşturularak `last`'a `head`'in adresi atanıyor (`head`'in adresini kaybetmememiz gerekiyor). `temp`'in `data`'sına parametre değişkeninden gelen veri aktarıldıktan sonra `last` döngü içerisinde ilerletilerek listenin son elemanı göstermesi sağlanıyor. `temp` `head`'i gösterecek şekilde atama yapıldıktan sonra listenin son elemanını gösteren `last`'ın `next` işaretçisine de `temp`'in adresi atanıyor. Şu anda `last` eklenen verinin bulunduğu düğümü gösteriyor. `temp`'in `next` işaretçisi ise `head`'i gösteriyor. `head`'e `temp` atanarak işlem tamamlanmış oluyor. DİKKAT! Artık `temp`'in `next` göstericisi, `head`'in bir önceki adres bilgisini tutuyor.

```
void insertAtFront(int key) {
     if(head == NULL) {
          head = (struct node *)malloc(sizeof(struct node));
          head -> data = key;
          head -> next = head;
     }
     else {
          struct node *temp = (struct node *)malloc(sizeof(struct node));
          struct node *last = head;
          temp -> data = key;
          while(last -> next != head) // listenin son elemanı bulunuyor.
              last = last -> next;
          temp -> next = head;
```

```
            last -> next = temp;
            head = temp;
        }
    }
```

6.2.2 Tek Bağlı Dairesel Listelerde Sona Eleman Eklemek

Fonksiyon, başa ekleme fonksiyonuna çok benzemektedir. Listenin NULL olup olmadığı kontrol ediliyor.

```
void insertAtLast(int key) {
    if(head == NULL) {
        head = (struct node *)malloc(sizeof(struct node));
        head -> data = key;
        head -> next = head;
    }
    else {
        struct node *temp = (struct node *)malloc(sizeof(struct node));
        struct node *last = head;
        temp -> data = key;
        // listenin son elemanı bulunuyor.
        while(last -> next != head)
            last = last -> next;
        temp -> next = head;
        last -> next = temp;
    }
}
```

Görüldüğü gibi başa ekleme fonksiyonunun sonundaki `head = temp;` satırını kaldırmak yeterlidir. Aynı fonksiyonu aşağıdaki şekilde de yazabilirdik. Burada `temp` isminde bir değişkene ihtiyacımızın olmadığını gözlemleyin.

```
void insertAtLast(int key) {
        if(head == NULL) {
            head = (struct node *)malloc(sizeof(struct node));
            head -> data = key;
            head -> next = head;
        }
        else {
            struct node *last = head;
            while(last -> next != head) // listenin son elemanı
    bulunuyor.
                last = last -> next;
            last -> next = (struct node *)malloc(sizeof(struct node));
            last -> next -> next = head;
```

```
            last -> next -> data = key;
        }
    }
```

Yazılan fonksiyonları siz de bilgisayarınızda kodlayarak mantığı kavramaya çalışınız. Ancak çok miktarda alıştırma yaptıktan sonra iyi bir pratik kazanabilirsiniz.

6.2.3 Tek Bağlı Dairesel Listelerde İki Listeyi Birleştirmek

`concatenate` fonksiyonu tek bağlı dairesel listelerde iki listeyi verilen ilk listenin sonuna ekleyerek birleştiren fonksiyon olarak tanımlanacaktır. Bu fonksiyonun yazımında önemli olan doğru işlem sırasını takip etmektir. Eğer öncelikli yapılması gereken bağlantılar sonraya bırakılırsa son `node`'un bulunmasında sorunlar çıkacaktır. `list_1` boş ise `list_2`'ye eşitleniyor. Eğer boş değilse her iki listenin de `last()` fonksiyonuyla son elemanları bulunuyor ve next işaretçilerine bir diğerinin gösterdiği adres değeri atanıyor.

```
// list_1 listesinin sonuna list_2 listesini eklemek
void concatenate(struct node*& list_1, struct node* list_2){
    //parametrelere dikkat
    if(list_1 == NULL)
        list_1 = list_2;
    else {
        // Birinci listenin son düğümünü last olarak bulmak için
        struct node *last=list_1;
        while(last -> next != list_1)
            last = last -> next;
        last->next=list_2; //Birinci listenin sonu ikinci listenin
    başına bağlandı
        // İkinci listenin son düğümünü last olarak bulmak için
        last=list_2;
        while(last -> next != list_2)
            last = last -> next;
        last->next=list_2; //İkinci listenin sonu birinci listenin
    başına bağlandı
    }
}
```

6.2.4 Tek Bağlı Dairesel Listede Arama Yapmak

Bu fonksiyon ile liste içinde arama yapılmaktadır. node'lar taranarak data'lara bakılır. Eğer aranan bilgi varsa, bulunduğu node'un adresiyle geri döner.

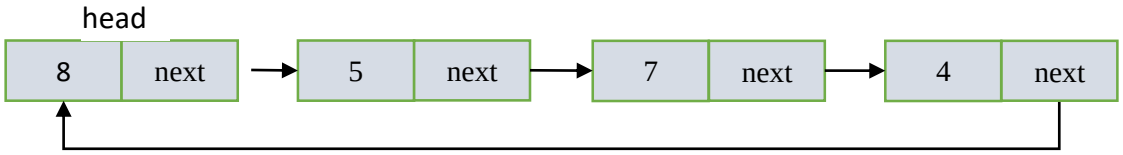
```
//head listesinde data'sı veri olan node varsa adresini alma
```

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

6.2.5 Tek Bağlı Dairesel Listelerde Verilen Bir Değere Sahip Düğümü Silmek

Silme işlemine dair fonksiyonumuzu yazmadan önce tek bağlı dairesel listelerin mantıksal yapısını tekrar gözden geçirmekte fayda vardır. Bu liste yapısında son düğümün `next` işaretçisi `head`'i gösteriyor ve dairesel yapı sağlanıyor.

Tek bağlı dairesel liste yapısı:



Tek bağlı dairesel listelerde verilen bir değere sahip düğümü silme işlemi için `deletenode` isimli bir fonksiyon yazacağız. Fonksiyonda liste boş ise hemen fonksiyondan çıkılır ve geriye `false` değeri döndürülür. Eğer silinecek düğüm ilk düğüm ise daha önce yaptığımız gibi ilk düğüm listeden çıkarılır. Ancak burada listenin son düğümünü yeni `head` düğümüne bağlamak gereklidir. Bu yüzden önce son düğüm bulunur.

```
struct node *deletenode(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.
          struct node *last=head;
          while(last -> next != head)
```

```
            last = last -> next;
        head = head -> next; // head artık bir sonraki eleman.
        last->next=head;
        free(temp);
    }
    else if(temp -> next == NULL) { // Listede tek düğüm bulunabilir.
        printf("Silmek istediginiz veri bulunmamaktadir.\n\n");
    }
    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;
}
```

6.3 Çift Bağlı Doğrusal (Double Linked) Listeler

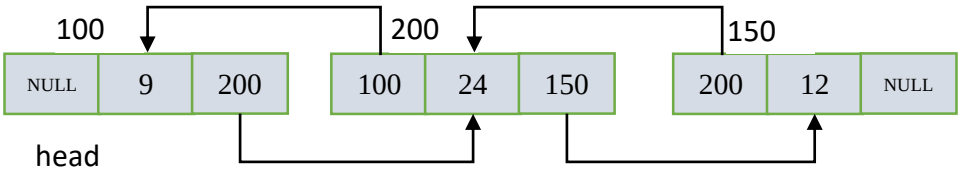
Her düğümün `data` adında verileri tutacağı bir değişkeni ile kendinden önceki ve sonraki düğümlerin adreslerini tutacak olan `prev` ve `next` isminde iki adet işaretçisi vardır. Listenin başını gösteren işaretçi `head` yapı değişkenidir. Şekilde `head`'in adresi `100`'dür ve `head`'in `prev` işaretçisi herhangi bir yeri göstermediğinden `NULL` değer içermektedir. `next` işaretçisi ise bir sonraki düğümün adresi olan `200` değerini içermektedir. İkinci düğümün `prev` işaretçisi `head`'in adresi olan `100` değerini tutmakta, `next` işaretçisi ise son düğümün adresi olan `150` değerini tutmaktadır. Nihayet son düğümün `prev` işaretçisi kendinden önceki düğümün adresini yani `200` değerini tutmakta ve `next` işaretçisi ise `NULL` değer içermektedir.

Çift bağlı listelerin `struct` yapısı:


```
struct node {
    int data;
```

```
    struct node* next;
    struct node* prev;
}
```

Çift bağlı listelerin mantıksal gösterimi:



6.3.1 Çift Bağlı Doğrusal Listenin Başına Eleman Eklemek

`head` değişkeninin global olarak tanımlandığını varsayarak `insertAtFirst` fonksiyonunu yazalım.

```
void insertAtFirst(int key) {
    if(head == NULL) { // liste yoksa oluşturuluyor
        head = (struct node *)malloc(sizeof(struct node));
        head -> data = key;
        head -> next = NULL;
        head -> prev = NULL;
    }
    else {
        struct node *temp = (struct node *)malloc(sizeof(struct node));
        temp -> data = key;
        temp -> next = head;
        temp -> prev = NULL;
        head -> prev = temp;
        head = temp;
    }
}
```

6.3.2 Çift Bağlı Doğrusal Listenin Sonuna Eleman Eklemek

```
void insertAtEnd(int key) {
    if(head == NULL) {
        head = (struct node *)malloc(sizeof(struct node));
        head -> data = key;
        head -> next = NULL;
        head -> prev = NULL;
    }
    else {
```

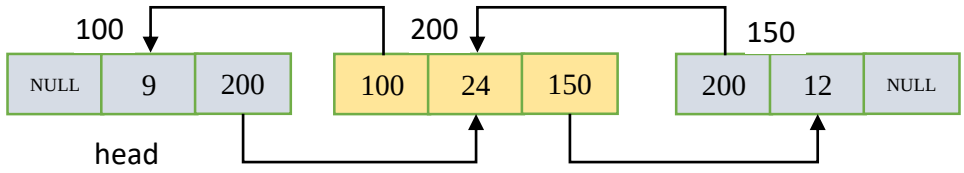
```
        struct node *temp = head;
        struct node *temp2 = (struct node *)malloc(sizeof(struct
    node));
        while(temp -> next != NULL) // listenin sonunu bulmamız
    gerekiyor.
            temp = temp -> next;
        temp2 -> data = key;
        temp2 -> next = NULL;
        temp2 -> prev = temp;
        temp -> next = temp2;
    }
}
```

6.3.3 Çift Bağlı Doğrusal Listelerde Araya Eleman Eklemek

```
// head listesinin n. düğümünün hemen ardına other_node düğümünü ekleme
void addthen(node* other_node, node*& list, int n) {
node* temp = head;
int i = 1;
while(i < n) {
head = head -> next;
i++;
}
other_node -> prev = head;
other_node -> next = head -> next;
head -> next = other_node;
head = temp;
}
```

6.3.4 Çift Bağlı Doğrusal Listelerde Verilen Bir Değere Sahip Düğümü Silmek

Aşağıdaki listeden ortadaki düğüm silinecektir:

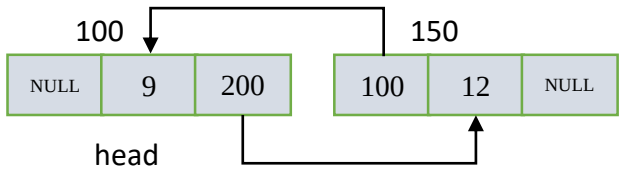


Silme işlemi diğer liste türlerine göre biraz farklılık göstermektedir. İlk düğüm silinmek istenirse `head`'in `next` işaretçisinin tuttuğu adresi `head`'e atadıktan sonra `prev` işaretçisinin değerini `NULL` yapmamız gerekir. Sonra `free()` fonksiyonuyla baştaki eleman silinebilir. Eğer ortadan bir düğüm silinmek istenirse, silinecek düğümün üzerinde durup bir önceki düğümü, silinmek istenen düğümden

bir sonraki düğüme bağlamak gerekir. Silinecek düğüm yukarıda görüldüğü gibi ortadaki düğüm olsun. `double_linked_remove` isimli fonksiyonumuzu yazarak konuyu kavrayalım.

```
    void double_linked_remove(int key) {
    struct node *temp = head;
    if(head -> data == key) { // silinecek değerin ilk düğümde olması durumu.
    head = head -> next;
    head -> prev = NULL;
    free(temp);
    }
    else {
    while(temp -> data != key)
    temp = temp -> next;
    temp -> prev -> next = temp -> next;
    /* silinecek düğümden bir önceki düğümün next işaretçisi, şimdi silinecek
        düğümden bir sonraki düğümü gösteriyor. */
    if(temp -> next != NULL) // silinecek düğüm son düğüm değilse
    temp -> next -> prev = temp -> prev;
    /* silinecek düğümden bir sonraki düğümün prev işaretçisi, şimdi
        silinecek düğümden bir önceki düğümü gösteriyor. */
    free(temp);
    }
}
```

Listenin, ortada bulunan düğümü silindikten sonraki liste görünümü:



6.3.5 Çift Bağlı Doğrusal Listelerde Arama Yapmak

```
    // head listesinde data'sı veri olan node varsa adresini alma
    struct node* locate(int veri, struct node* head) {
        struct node* locate = NULL;
        while(head != NULL) {
            if(head -> data != veri) {
                head = head -> next; // aranan veri yoksa liste taranıyor
            }
            else {
                locate = head;
                break;  // veri bulunursa döngüden çıkılarak geri
    döndürülüyor
            }
```

```
    }
    return locate;
}
```

6.3.6 Çift Bağlı Doğrusal Listede Karşılaştırma Yapmak

Verilen `node`'un bu listede var olup olmadığını kontrol eden fonksiyondur. Fonksiyon eğer listede node varsa `1`, yoksa `0` ile geri döner.

```
bool is_member(struct node* other_node, struct node* head) {
    while(head != NULL && head != other_node)
        head = head -> next;
    return(head == other_node); // ifade doğruysa 1, değilse 0 geri
    döndürülür.
}
```

Verilen örneklerdeki fonksiyonlar, varsayılan bir listenin yahut global tanımlanmış `head` değişkeninin varlığı, yine global olarak tanımlanmış yapıların olduğu kabul edilerek yazılmıştır. Eğer bu kabulümüz olmasaydı şüphesiz kontrol ifadelerini de içeren uzun kod satırları meydana gelirdi.

6.3.7 Çift Bağlı Doğrusal Listelerin Avantajları ve Dezavantajları

Avantajları

- Her iki yönde gezilebilir,
- Ekleme,Silme gibi bazı işlemler daha kolaydır.

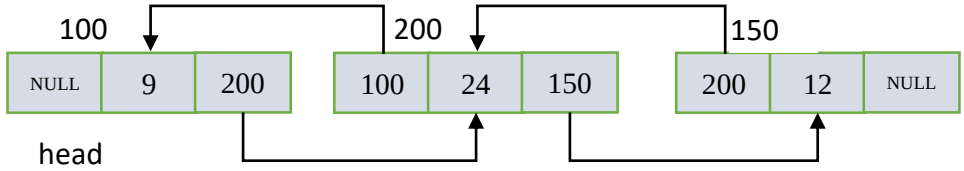
Dezavantajları

- Bellekte daha fazla yer kaplar,
- Her düğümün `prev` ve `next` adında iki işaretçisi olduğu için liste işlemleri daha yavaştır,
- Hata olasılığı yüksektir. Örneğin düğümlerin `prev` işaretçisinin bir önceki düğüme bağlanması ya da `next` işaretçisinin bir sonraki düğüme bağlanması unutulabilir.

6.4 Çift Bağlı Dairesel (Double Linked) Listeler

İki bağlı doğrusal listelerde listenin birinci düğümünün `prev` değeri ve sonuncu düğümünün `next` değeri `NULL` olmaktaydı. Ancak iki bağlı dairesel listelerde listenin birinci düğümünün `prev`

değeri sonuncu düğümünün `data` kısmını, sonuncu düğümünün `next` değeri de listenin başını (yani listenin ismini) göstermektedir. Fonksiyonlarda bununla beraber değişmektedir.



6.4.1 Çift Bağlı Dairesel Listelerde Başa Düğüm Eklemek

Listenin başına bir düğüm ekleyen `insertAtFirst` fonksiyonunu yazalım.

```
void addhead(struct node*&head, int key) {
     if(head == NULL) {
          head = (struct node *)malloc(sizeof(struct node));
          head -> data = key;
          head -> next = head;
          head -> prev = head;
     }
     else {
          struct node *temp = (struct node *)malloc(sizeof(struct node));
          temp -> data = key;
          struct node *last = head;
          // liste çift bağlı ve dairesel olduğu için son eleman head-
     >prev dir.
          head->prev->next=temp;
          temp->next=head;
          temp->prev=head->prev;
          head->prev=temp;
          head = temp;
     }
}
```

6.4.2 Çift Bağlı Dairesel Listenin Sonuna Eleman Eklemek

`addhead` fonksiyonundaki son satır `head = temp`; yazılmaz ise listenin sonuna ekleme yapılmış olur.

```
void addlast(struct node* temp, struct node*&head) {
     if(!head)
          head = temp;
     else {
          temp -> next = last(head) -> next;
```

```
            temp -> prev = last(head);
            last(head) -> next = temp;
            // last fonksiyonu ile son düğüm bulunuyor
            head -> prev = temp;
        }
}
```

6.4.3 Çift Bağlı Dairesel Listelerde İki Listeyi Birleştirmek

```
// list_1 listesinin sonuna list_2 listesini eklemek
void concatenate(struct node*& list_1, struct node* list_2){
     //parametrelere dikkat
     if(list_1 == NULL)
          list_1 = list_2;
     else {
          // Birinci listenin son düğümünü last olarak bulmak için
          struct node *last=list_1;
          while(last -> next != list_1)
              last = last -> next;
          last->next=list_2; //Birinci listenin sonu ikinci listenin
     başına bağlandı
          list2->prev=last; //İkinci listenin başı birinci listenin
     sonuna bağlandı
          // İkinci listenin son düğümünü last olarak bulmak için
          last=list_2;
          while(last -> next != list_2)
              last = last -> next;
          last->next=list_1; //İkinci listenin sonu birinci listenin
     başına bağlandı
          list1->prev=last; //Birinci listenin başı ikinci listenin
     sonuna bağlandı
     }
}
```

6.4.4 Çift Bağlı Dairesel Listelerde Araya Eleman Eklemek

```
// head listesinin n. düğümünün hemen ardına other_node düğümünü ekleme
void addthen(struct node* other_node, struct node*&head, int n) {
     node* temp = head;
     int i = 1;
     while(i < n) {
          head = head -> next;
          i++;
     }
     head -> next -> prev = other_node;
     other_node -> prev = head;
```

```
    other_node -> next = head -> next;
    head -> next = other_node;
    head = temp;
}
```