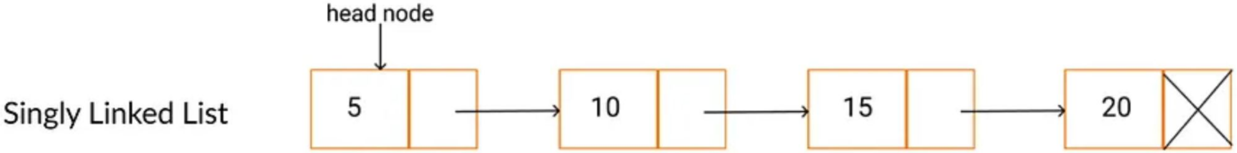
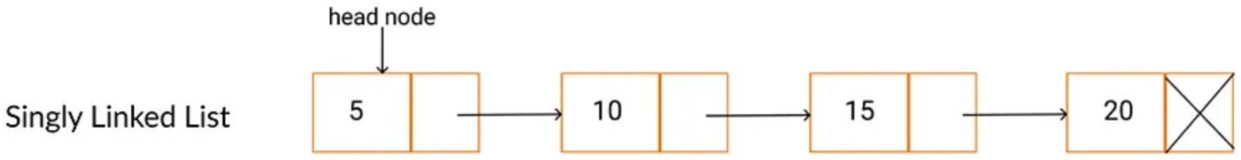
Linked List

Md. Tanvir Alam

Linked List?



Linked List?

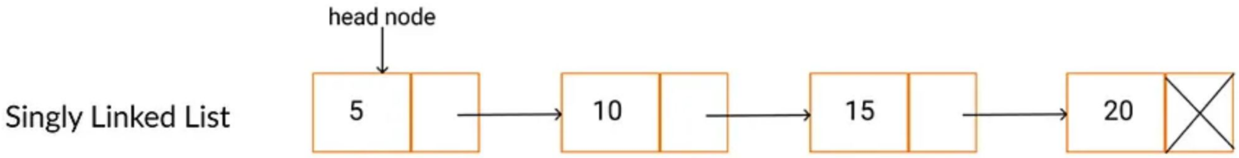


```
struct node{



};
```

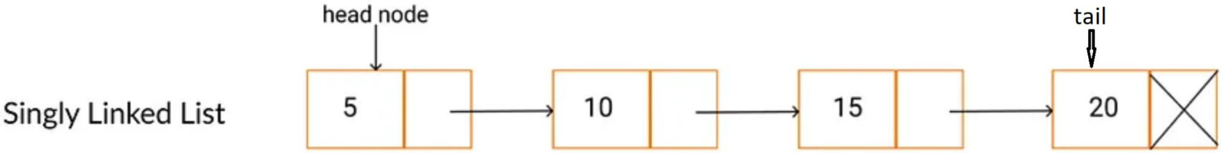
Linked List?



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

Linked List Structure

```
struct ll{
    struct node{
        node *next;
        int val;
    };
    node *head=NULL;
    node *tail=NULL;
};
```



Insert at First

```
void insert_first(int x){
        //insert x at first
        node *a = (node*)malloc(sizeof(node));
        a->next=NULL;
        a->val=x;
        if(head==NULL){
            head=a;
            tail=a;
        }
        else{
            a->next=head;
            head=a;
        }
    }
```

Insert at Last

```
void insert_last(int x){
        //insert x at last
        node*a=(node*)malloc(sizeof(node));
        a->next=NULL;
        a->val=x;
        if(tail){
            tail->next=a;
            tail=a;
        }
        else{
            head=a;
            tail=a;
        }
    }
```

Traverse

```
void print(){
        node *p=head;
        while(p)
        {
            printf("%d ",p->val);
            p=p->next;
        }
}
```

Delete First Element

```
int delete_f()
    {
        //delete first element
        if(head==NULL)
            return -1;
        if(head==tail)
        {
            int x=head->val;
            head=NULL;
            tail=NULL;
            return x;
        }
        else
        {
            int x=head->val;
            head=head->next;
            return x;
        }
    }
```

Delete Last Element

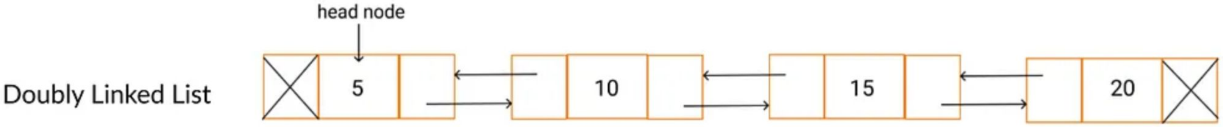
?

Delete

```
void delfh(int x){
        //delete the first x
        node *p=head;
        node *tmp;
        if(head->val==x){
            head = head->next;
        }
        while(p->next)
        {
            if(p->next->val==x)
            {
                p->next = p->next->next;
                break;
            }
            p=p->next;
        }
}
```

Doubly Linked List

```
struct dll
{
    struct node
    {
        node* prev;
        node* next;
        int val;
    };
    node* head=NULL;
    node* tail=NULL;
};
```



Insert at First

```
void insert_first(int x)
    {
        node* a=(node*)malloc(sizeof(node));
        a->prev=NULL;
        a->next=NULL;
        a->val=x;
        if(head==NULL)
        {
            head=a;
            tail=a;
        }
        else
        {
            a->next=head;
            head->prev=a;
            head=head->prev;
        }
    }
```

Insert at Last

```
void insert_last(int x){
        node* a=(node*)malloc(sizeof(node));
        a->prev=NULL;
        a->next=NULL;
        a->val=x;
        if(head==NULL)
        {
            head=a;
            tail=a;
        }
        else
        {
            a->prev=tail;
            tail->next=a;
            tail=tail->next;
        }
    }
```

Traverse

```
void print()
    {
        node* p=head;
        while(p)
        {
            printf("%d ",p->val);
            p=p->next;
        }
    }
```

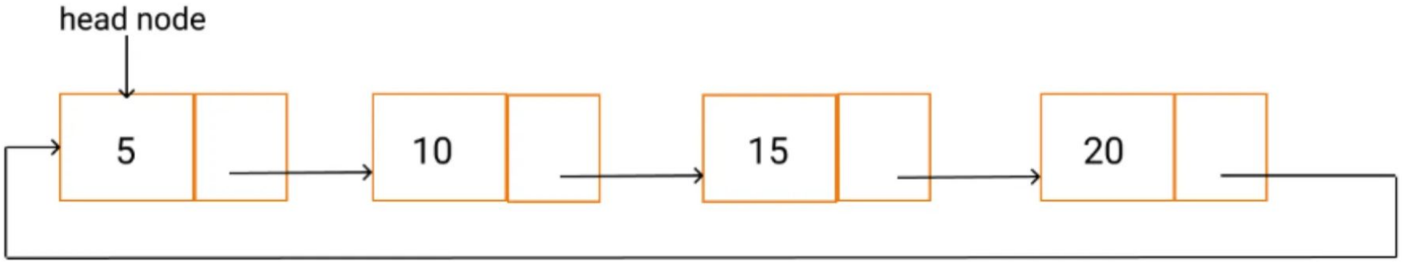
Delete First Element

```
int delete_first()
    {
        if(head==NULL)
            return -1;
        else if(head->next==NULL)
        {
            int x=head->val;
            head=NULL;
            tail=NULL;
            return x;
        }
        else
        {
            int x=head->val;
            head=head->next;
            head->prev=NULL;
            return x;
        }
    }
```

Delete Last Element

```
int delete_last()
    {
        if(head==NULL)
            return -1;
        else if(head->next==NULL)
        {
            int x=head->val;
            head=NULL;
            tail=NULL;
            return x;
        }
        else
        {
            int x=tail->val;
            tail=tail->prev;
            tail->next=NULL;
            return x;
        }
    }
```

Circular Linked List



Why Linked List?

- Can grow and shrink at runtime
- No memory wastage