DSA
INTERNSHALA
Doubly Linked List

Traversing a Doubly Linked List

```
[Algorithm to traverse doubly linked list]
   1. Set temp:=start
   2. Repeat steps 3,4 while temp != NULL
   3. Write: data[temp]
   4. Set temp:=link[temp]
   5. Return

[Algorithm to traverse doubly linked list in reverse]
   1. Set temp:= end
   2. Repat steps 3,4 while temp != NULL
   3. Write : data[temp]
   4. Set temp:=prev[temp]
   5. Return
```

Inserting to new node before a node with a given address

```
[Algorithm to insert a node with before a given address]
   1. If avail==NULL, then:
      Write: overflow error
      Return
      [end of if]
   2. Set new:=avail, avail:=link[avail]
   3. Set data[new]:=val
   4. Set link[new]:=addr
   5. link[prev[addr]]:=new
   6. Set prev[new]:=prev[addr]
   7. Set prev[addr]:=new
   8. Return
```

Inserting a new node after a node with a given address

[Algorithm to insert a new node after a given address]

1. If avail==NULL, then:
   Write: overflow error
   Return
   [end of if]
2. Set new:=avail, avail:=link[avail]
3. Set data[new]:=val
4. Set link[new]:=link[addr]
5. Set prev[new]:=addr
6. Set prev[link[addr]:=new
7. Return

Deletion of a node with a given address

[Algorithm to delete a node]

1. Set link[prev[addr]]:=link[addr]
2. Set prev[link[addr]]:=prev[addr]
3. Set link[addr]:=avail
4. Set avail:=addr
   [garbage collection in steps 3,4]
5. Return

Creating a doubly linked list

```
void create(node **start, node **end) {
      node *temp, *p;
      char ch='y';
      system("cls");
      if(*start!=NULL) {
            printf("\n\n\t****LIST ALREADY EXISTS****");
            getch();
            return;
      }
      fflush(stdin);
      printf("\n\n\t****INPUT BLOCK****\n");
      while(ch=='y'){
            temp=(node*)malloc(sizeof(node));
            printf("\n\tEnter the no:=>");
            scanf("%d",&temp->no);
            temp->next=NULL;
            if(*start==NULL){
                  p = *start->temp;
                  temp->prev=NULL;
            } else {
                  p->next=temp;
                  temp->prev=p;
```

```
                        p=temp;
                }
                *end=temp;
                fflush(stdin);
                printf("\n\tDo you want to continue (y/n)?:");
                ch=getchar();
        }
        return;
}
```

Main function calls create(&start,&end);

Traversal

```
void print(node *start,node *end) {
        node *temp;
        system("cls");
        printf("\n\n\t ****DISPLAY BLOCK****");
        printf("\n\n\tBase Address Number Next Link");
        printf("=====================================");
        for(temp=start;temp!=NULL;temp=temp->next){
                printf("\n %10u %10d %10u",temp,temp->no,temp->next);
        }
        getch();
        printf("\n\n\t End address Number Prev Link");
        printf("=====================================");
        for(temp=end;temp!=NULL;temp=temp->prev){
                printf("\n %10u %10d %10u",temp,temp->no,temp->prev);
        }
        printf(""\n\n\t Press any key to goto main block…);
        getch();
        return;
}
```

Inserting in Beginning

```
node* insf(node *start, int item){
        node *p;
        p=(node *) malloc(sizef(node));
        p->no = item;
        start->prev=p;
        p->next=start;
        p->prev=NULL:
        start=p;
        printf("\n\n\t Element is successfully inserted");
        getch();
        retrun start;
}
```

Inserting before a given node

```
node *insb(int item, int item1, node *start){
	node *temp, *p;
	for(temp=start;temp->no!=item1;temp=temp->next){
		if(temp == NULL) {
			printf("\n\n\t No. not found in the Linklist");
			getch();
			return start;
		}
	}
	if(temp==start){
	start=insf(start,item);
	return start;
	}
	p=(node *)malloc(sizeof(node));
	p->no=item;
	p->prev=temp->prev;
	p->next=temp;
	temp->prev->next=p;
	temp->prev=p;
	if(p->prev==NULL){
		start=p;
	}
	printf("\n\n\t Element is successfully inserted");
	getch();
	return start;
}
```

Inserting after a given node

```
node* insa(int item, int item1, node *start, node *end){
	node *temp, *p;
	p = (node *) malloc(sizeof(node));
	for(temp = start; temp->no!=item1;temp=temp->next){
		if(temp == NULL){
			printf("\n\n\t no. not found in the linked list");
			getch();
			return end;
		}
	}
	p->no=item;
	p->next=temp->next;
	p->prev=temp;
	if(temp->next!=NULL){
		temp->next->prev=p;
	}
	temp->next=p;
	if(p->next==NULL){
		end=p;
	}
```

```
        printf("\n\n\t Element is successfully inserted");
        getch();
        return end;
}
```

Deleting the first node

```
node *delf(node *start){
        int item;
        node *tp;
        tp = start;
        item=start->no;
        start=start->next;
        start->prev=NULL:
        printf("\n\n\t  Element  [%d]  is  successfully  deleted  from  the
linkedlist",item);
        free(tp);
        getch();
        return start;
}
```

Deleting the last node

```
node *dele(node *end){
        int item;
        item = end->no;
        end = end->prev;
        end->next = NULL;
        printf("\n\n\t Element [%d] is deleted from Linked list",item);
        getch();
        return end;
}
```

Deleting a particular node

```
void delp(int item, node** start, node **end){
        node *temp, *temp1;
        for(temp* start; temp->no!=item;temp=temp->next){
                temp1=temp;
        }
        if(temp == NULL){
                printf("\n\n\t No. not found in the Linked List");
                getch();
                return;
        }
        if(temp->prev == NULL){
                *start=delf(*start);
        } else if(temp->next == NULL) {
                *end=dele(*end);
```

```
        } else {
              temp1->next=temp->next;
              temp->next->prev=temp1;
              printf("\n\n\tElement    [%d]    is    deleted    from    the
        linkedlist",item);
        }
        return;
}
```