***ASSIGNMENT-5***

1. ***WAP to implement Linked List.  
   a. Create a linked list.  
   b. Insert an element at the start of the linked list.  
   c. Insert an element at the end of the linked list.  
   d. Insert an element before an existing element whose information is x in a linked list.  
   e. Insert an element after an existing element whose information is x in a linked list.  
   f. Delete the first element of the linked list.  
   g. Delete the last element of the linked list.  
   h. Delete the element whose information is x from a linked list.  
   i. Display the contents of the linked list.***

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*next;

};

typedef struct node NODE;

NODE \*create(NODE \*head);

NODE \*insertstart(NODE \*head,int data);

NODE \*insertlast(NODE \*head,int data);

NODE \*insertafter(NODE \*head,int item,int data);

NODE \*insertbefore(NODE \*head,int item,int data);

NODE \*deletefront(NODE \*head);

NODE \*deletelast(NODE \*head);

NODE \*delete(NODE \*head,int x);

void display(NODE \*head);

int main()

{

NODE \*head = NULL;

int ch,data,x;

printf("Enter your choice what you want to do\n");

printf("1.Create a Linked list\n");

printf("2.add an element at front\n");

printf("3.add an element at last\n");

printf("4.add an element after a specific element\n");

printf("5.add an element before a specific element\n");

printf("6.Delete an element from front\n");

printf("7.Delete an element from last\n");

printf("8.delete an specific element\n");

printf("9.display list\n");

printf("10.Exit\n");

while(1)

{

scanf("%d",&ch);

switch(ch)

{

case 1:

head = create(head);

break;

case 2:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertstart(head,data);

break;

case 3:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertlast(head,data);

break;

case 4:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertafter(head,x,data);

break;

case 5:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertbefore(head,x,data);

break;

case 6:

head = deletefront(head);

break;

case 7:

head = deletelast(head);

break;

case 8:

printf("Enter element which is to deleted\n");

scanf("%d",&x);

head = delete(head,x);

break;

case 9:

display(head);

break;

case 10:

exit(1);

}

}

return 0;

}

NODE \*create(NODE \*head)

{

if(head!=NULL)

{

NODE \*p=head,\*q;

while(p!=NULL){

q=p;

p = p->next;

free(q);

}

}

head=NULL;

int n;

printf("Enter number of nodes: ");

scanf("%d",&n);

int i,x;

printf("Enter %d values:",n);

for(i=1;i<=n;i++)

{

scanf("%d",&x);

head = insertlast(head,x);

}

return head;

}

NODE \*insertstart(NODE \*head,int data)

{

NODE \*tmp;

tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

tmp->next = head;

return tmp;

}

NODE \*insertlast(NODE \*head,int data)

{

NODE \*tmp,\*p = head;

tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

tmp->next = NULL;

if(head==NULL)

return tmp;

while(p->next!=NULL)

p = p->next;

p->next = tmp;

return head;

}

NODE \*insertafter(NODE \*head,int item,int data)

{

NODE \*tmp,\*p = head;

tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head==NULL || item==-1)

{

tmp->next = head;

head = tmp;

}

else{

while(p!=NULL && p->info!=item)

p = p->next;

if(p==NULL)

{

printf("Element is not found in linked list,So inserted at front\n");

tmp->next = head;

head = tmp;

}

else

{

tmp->next = p->next;

p->next = tmp;

}

}

return head;

}

NODE \*insertbefore(NODE \*head,int item,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head==NULL)

return tmp;

if(head->info==item)

{

tmp->next=head;

return tmp;

}

NODE \*p=head->next,\*q=head;

while(p!=NULL && p->info!=item){

q=p;

p=p->next;

}

if(p==NULL)

{

printf("Element is not found,So inserted at front\n");

tmp->next=head;

return tmp;

}

q->next=tmp;

tmp->next=p;

return head;

}

NODE \*deletefront(NODE \*head)

{

if(head==NULL)

printf("List is empty\n");

else

{

NODE \*tmp = head->next;

free(head);

return tmp;

}

return head;

}

NODE \*deletelast(NODE \*head)

{

if(head==NULL){

printf("list is empty\n");

return NULL;

}

NODE \*p=head,\*q;

if(head->next==NULL)

{

free(head);

return NULL;

}

while(p->next!=NULL)

{

q=p;

p = p->next;

}

q->next = NULL;

free(p);

return head;

}

NODE \*delete(NODE \*head,int x)

{

NODE \*p=head,\*q;

if(head==NULL)

{

printf("List is empty\n");

return NULL;

}

while(p!=NULL && p->info!=x)

{

q=p;

p = p->next;

}

if(p==NULL)

{

printf("Element not found\n");

return head;

}

if(p==head)

{

free(p);

return head->next;

}

q->next = p->next;

free(p);

return head;

}

void display(NODE \*head)

{

NODE \*p = head;

if(p==NULL)

printf("List is empty\n");

while(p!=NULL)

{

printf("%d\t",p->info);

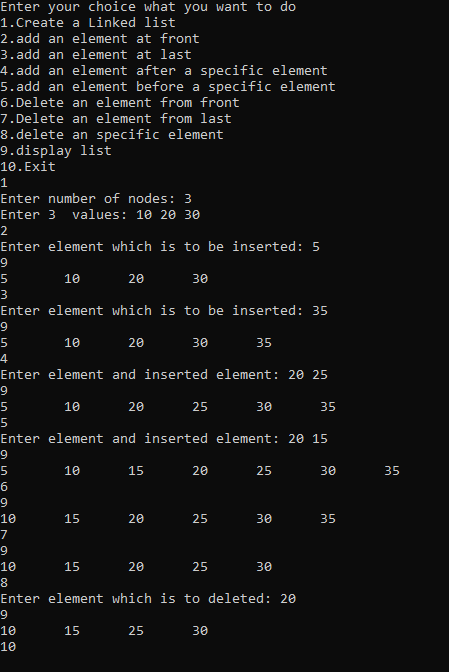
p = p->next;

}

printf("\n");

}

***OUTPUT:***



1. ***WAP to implement Circular Linked List.***

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*next;

};

typedef struct node NODE;

NODE \*last;

NODE \*create(NODE \*head);

NODE \*insertstart(NODE \*head,int data);

NODE \*insertlast(NODE \*head,int data);

NODE \*insertafter(NODE \*head,int item,int data);

NODE \*insertbefore(NODE \*head,int item,int data);

NODE \*deletefront(NODE \*head);

NODE \*deletelast(NODE \*head);

NODE \*delete(NODE \*head,int x);

void display(NODE \*head);

int main()

{

NODE \*head = NULL;

int ch,data,x;

printf("Enter your choice what you want to do\n");

printf("1.Create a linked list\n");

printf("2.add an element at front\n");

printf("3.add an element at last\n");

printf("4.add an element after a specific element\n");

printf("5.add an element before a specific element\n");

printf("6.Delete an element from front\n");

printf("7.Delete an element from last\n");

printf("8.delete an specific element\n");

printf("9.display list\n");

printf("10.Exit\n");

while(1)

{

scanf("%d",&ch);

switch(ch)

{

case 1:

head = create(head);

break;

case 2:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertstart(head,data);

break;

case 3:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertlast(head,data);

break;

case 4:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertafter(head,x,data);

break;

case 5:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertbefore(head,x,data);

break;

case 6:

head = deletefront(head);

break;

case 7:

head = deletelast(head);

break;

case 8:

printf("Enter element which is to deleted\n");

scanf("%d",&x);

head = delete(head,x);

break;

case 9:

display(head);

break;

case 10:

exit(1);

}

}

return 0;

}

NODE \*create(NODE \*head)

{

if(head!=NULL)

{

NODE \*p=head,\*q;

while(p!=NULL){

q=p;

p = p->next;

free(q);

}

}

head=NULL;

int n;

printf("Enter number of nodes: ");

scanf("%d",&n);

int i,x;

printf("Enter %d values:",n);

for(i=1;i<=n;i++)

{

scanf("%d",&x);

head = insertlast(head,x);

}

return head;

}

NODE \*insertstart(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head==NULL){

tmp->next = tmp;

last = tmp;

return tmp;

}

tmp->next = head;

last->next = tmp;

return tmp;

}

NODE \*insertlast(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head==NULL)

{

tmp->next = tmp;

last = tmp;

return tmp;

}

tmp->next = last->next;

last->next = tmp;

last = tmp;

return head;

}

NODE \*insertafter(NODE \*head,int item,int data)

{

NODE \*p=head;

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head==NULL || item == -1)

{

if(head==NULL)

{

tmp->next = tmp;

last = tmp;

}

else

{

tmp->next = head;

last->next = tmp;

}

return tmp;

}

else

{

do{

if(p->info==item)

break;

p = p->next;

}while(p!=head);

if(p==head && p->info!=item)

{

printf("Element is not found in the list,So inserted at first\n");

tmp->next = head;

last->next = tmp;

return tmp;

}

else

{

tmp->next = p->next;

p->next = tmp;

if(p==last)

last = tmp;

return head;

}

}

}

NODE \*insertbefore(NODE \*head,int item,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info=data;

if(head==NULL){

tmp->next=tmp;

return (last=tmp);

}

NODE \*p=head,\*q;

while(p->next!=head && p->info!=item)

{

q=p;

p=p->next;

}

if(p->next==head && p->info!=item){

printf("Element is not found,So inserted at first\n");

tmp->next=head;

last->next=tmp;

return tmp;

}

else

{

if(p==head)

{

tmp->next=head;

last->next=tmp;

return tmp;

}

q->next=tmp;

tmp->next=p;

return head;

}

}

void display(NODE \*head)

{

if(head==NULL)

printf("List is empty\n");

else

{

NODE \*p=head;

do{

printf("%d ",p->info);

p = p->next;

}while(p!=head);

printf("\n");

}

}

NODE \*deletefront(NODE \*head)

{

if(head==NULL)

{

printf("List is empty\n");

return head;

}

if(head->next==head){

free(head);

return (last=NULL);

}

last->next = head->next;

free(head);

return last->next;

}

NODE \*deletelast(NODE \*head)

{

if(head==NULL)

{

printf("List is empty\n");

return NULL;

}

if(head->next==head)

{

free(head);

return (last=NULL);

}

NODE \*q,\*p = head;

while(p->next!=last)

p = p->next;

p->next = last->next;

q = last;

last = p;

free(q);

return head;

}

NODE \*delete(NODE \*head,int x)

{

if(head==NULL)

{

printf("List is empty\n");

return head;

}

NODE \*p=head,\*q=last;

do{

if(p->info==x)

break;

q=p;

p = p->next;

}while(p!=head);

if(p==head && p->info!=x)

{

printf("Element is not found\n");

return head;

}

else

{

if(head->next == head)

return (last=NULL);

q->next = p->next;

free(p);

if(p==last)

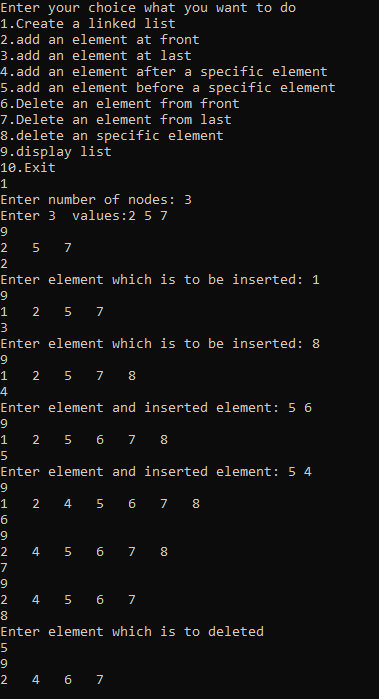
last = q;

return last->next;

}

}

***OUTPUT:***



1. ***WAP to implement Header Linked List.***

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*next;

};

typedef struct node NODE;

NODE \*create(NODE \*head);

NODE \*insertstart(NODE \*head,int data);

NODE \*insertlast(NODE \*head,int data);

NODE \*insertafter(NODE \*head,int item,int data);

NODE \*insertbefore(NODE \*head,int item,int data);

NODE \*deletefront(NODE \*head);

NODE \*deletelast(NODE \*head);

NODE \*delete(NODE \*head,int x);

void display(NODE \*head);

int main()

{

NODE \*head = (NODE \*)malloc(sizeof(NODE));

head->next = NULL;

int ch,data,x;

printf("Enter your choice what you want to do\n");

printf("1.Create a Linked list\n");

printf("2.add an element at front\n");

printf("3.add an element at last\n");

printf("4.add an element after a specific element\n");

printf("5.add an element before a specific element\n");

printf("6.Delete an element from front\n");

printf("7.Delete an element from last\n");

printf("8.delete an specific element\n");

printf("9.display list\n");

printf("10.Exit\n");

while(1)

{

scanf("%d",&ch);

switch(ch)

{

case 1:

head = create(head);

break;

case 2:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertstart(head,data);

break;

case 3:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertlast(head,data);

break;

case 4:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertafter(head,x,data);

break;

case 5:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertbefore(head,x,data);

break;

case 6:

head = deletefront(head);

break;

case 7:

head = deletelast(head);

break;

case 8:

printf("Enter element which is to deleted: ");

scanf("%d",&x);

head = delete(head,x);

break;

case 9:

display(head);

break;

case 10:

exit(1);

}

}

return 0;

}

NODE \*create(NODE \*head)

{

if(head->next!=NULL)

{

NODE \*p=head->next,\*q;

while(p!=NULL){

q=p;

p = p->next;

free(q);

}

}

head->next=NULL;

int n;

printf("Enter number of nodes: ");

scanf("%d",&n);

int i,x;

printf("Enter %d values:",n);

for(i=1;i<=n;i++)

{

scanf("%d",&x);

head = insertlast(head,x);

}

return head;

}

NODE \*insertstart(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

tmp->next = head->next;

head->next = tmp;

return head;

}

NODE \*insertlast(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

NODE \*p=head;

while(p->next!=NULL)

p = p->next;

tmp->next = NULL;

p->next = tmp;

return head;

}

NODE \*insertafter(NODE \*head,int item,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head->next==NULL || item==-1)

{

tmp->next = head->next;

head->next = tmp;

return head;

}

NODE \*p=head->next;

while(p->next!=NULL && p->info!=item)

p = p->next;

if(p->next==NULL && p->info!=item)

{

printf("Element is not found,So inserted at front\n");

tmp->next = head->next;

head->next = tmp;

}

else

{

tmp->next = p->next;

p->next = tmp;

}

return head;

}

NODE\*insertbefore(NODE \*head,int item,int data)

{

NODE \*tmp=(NODE \*)malloc(sizeof(NODE));

tmp->info=data;

NODE \*p=head->next,\*q=head;

while(p!=NULL && p->info!=item)

{

q=p;

p=p->next;

}

if(p==NULL)

{

printf("Element is not found,So inserted at front\n");

tmp->next=head->next;

head->next=tmp;

return head;

}

q->next = tmp;

tmp->next=p;

return head;

}

NODE \*deletefront(NODE \*head)

{

if(head->next==NULL)

{

printf("List is empty\n");

return head;

}

NODE \*p = head->next;

head->next = p->next;

free(p);

return head;

}

NODE \*deletelast(NODE \*head)

{

if(head->next==NULL)

{

printf("List is empty\n");

return head;

}

NODE \*p=head,\*q;

while(p->next!=NULL){

q = p;

p = p->next;

}

q->next = NULL;

free(p);

return head;

}

NODE \*delete(NODE \*head,int x)

{

if(head->next==NULL)

{

printf("List is empty\n");

return head;

}

NODE \*p=head->next,\*q=head;

while(p->next!=NULL && p->info!=x)

{

q = p;

p = p->next;

}

if(p->next==NULL && p->info!=x)

printf("Element is not found\n");

else

{

q->next = p->next;

free(p);

}

return head;

}

void display(NODE \*head)

{

if(head->next==NULL)

printf("List is empty\n");

else

{

NODE \*p = head->next;

do{

printf("%d ",p->info);

p=p->next;

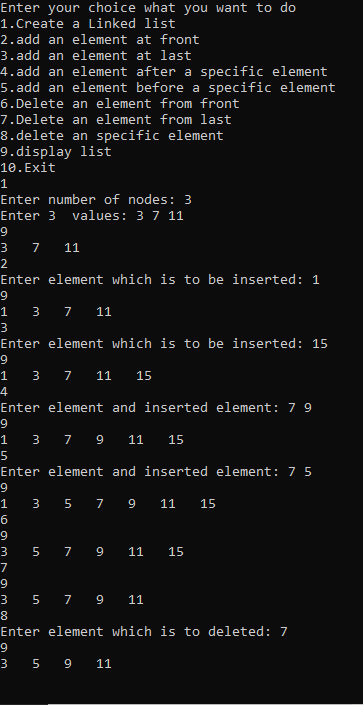
}while(p!=NULL);

}

printf("\n");

}

***OUTPUT :***



1. ***WAP to implement Header based Circular Linked List.***

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*next;

};

typedef struct node NODE;

NODE \*last=NULL;

NODE \*create(NODE \*head);

NODE \*insertstart(NODE \*head,int data);

NODE \*insertlast(NODE \*head,int data);

NODE \*insertafter(NODE \*head,int item,int data);

NODE \*insertbefore(NODE \*head,int item,int data);

NODE \*deletefront(NODE \*head);

NODE \*deletelast(NODE \*head);

NODE \*delete(NODE \*head,int x);

void display(NODE \*head);

int main()

{

NODE \*head = (NODE \*)malloc(sizeof(NODE));

head->next = head;

last = head;

int ch,data,x;

printf("Enter your choice what you want to do\n");

printf("1.Create a Linked list\n");

printf("2.add an element at front\n");

printf("3.add an element at last\n");

printf("4.add an element after a specific element\n");

printf("5.add an element before a specific element\n");

printf("6.Delete an element from front\n");

printf("7.Delete an element from last\n");

printf("8.delete an specific element\n");

printf("9.display list\n");

printf("10.Exit\n");

while(1)

{

scanf("%d",&ch);

switch(ch)

{

case 1:

head = create(head);

break;

case 2:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertstart(head,data);

break;

case 3:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertlast(head,data);

break;

case 4:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertafter(head,x,data);

break;

case 5:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertbefore(head,x,data);

break;

case 6:

head = deletefront(head);

break;

case 7:

head = deletelast(head);

break;

case 8:

printf("Enter element which is to deleted: ");

scanf("%d",&x);

head = delete(head,x);

break;

case 9:

display(head);

break;

case 10:

exit(1);

}

}

return 0;

}

NODE \*create(NODE \*head)

{

if(head->next!=head)

{

NODE \*p=head->next,\*q;

while(p!=head){

q=p;

p = p->next;

free(q);

}

}

head->next=head;

int n;

printf("Enter number of nodes: ");

scanf("%d",&n);

int i,x;

printf("Enter %d values:",n);

for(i=1;i<=n;i++)

{

scanf("%d",&x);

head = insertlast(head,x);

}

return head;

}

NODE \*insertstart(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

tmp->next = head->next;

if(head->next==head)

last = tmp;

head->next = tmp;

return head;

}

NODE \*insertlast(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

tmp->next = last->next;

last->next = tmp;

last = tmp;

return head;

}

NODE \*insertafter(NODE \*head,int item,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head->next==head || item==-1)

{

tmp->next = head->next;

if(head->next==head)

last = tmp;

head->next = tmp;

return head;

}

NODE \*p=head->next;

while(p!=head && p->info!=item)

p = p->next;

if(p==head)

{

printf("Element is not found in the list,So inserted at start\n");

tmp->next = head->next;

head->next = tmp;

}

else

{

if(p==last)

last = tmp;

tmp->next = p->next;

p->next = tmp;

}

return head;

}

NODE \*insertbefore(NODE \*head,int item,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head->next==head)

{

tmp->next = head;

head->next=tmp;

last = tmp;

return head;

}

NODE \*p=head->next,\*q=head;

while(p!=head && p->info!=item)

{

q=p;

p=p->next;

}

if(p==head && p->info!=item)

{

printf("Element is not found,So inserted at front\n");

tmp->next = head->next;

head->next = tmp;

return head;

}

q->next=tmp;

tmp->next = p;

return head;

}

NODE \*deletefront(NODE \*head)

{

if(head->next == head)

printf("List is empty\n");

else

{

NODE \*p=head->next;

if(p->next==head)

last=head;

head->next = p->next;

free(p);

}

return head;

}

NODE \*deletelast(NODE \*head)

{

if(head->next==head)

printf("List is empty\n");

else

{

NODE \*p = head,\*q;

while(p->next!=last)

p = p->next;

p->next = last->next;

q = last;

last = p;

free(q);

}

return head;

}

NODE \*delete(NODE \*head,int x)

{

if(head->next==head)

printf("List is empty\n");

else

{

NODE \*p=head->next,\*q=head;

while(p!=head && p->info!=x){

q = p;

p =p->next;

}

if(p==head)

printf("Element is not found\n");

else

{

q->next = p->next;

if(p==last)

last = q;

free(p);

}

}

return head;

}

void display(NODE \*head)

{

if(head->next==head)

printf("List is empty\n");

else

{

NODE \*p =head->next;

while(p!=head)

{

printf("%d ",p->info);

p = p->next;

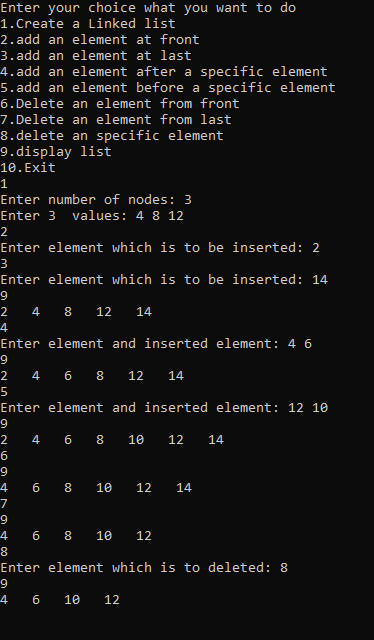
}

printf("\n");

}

}

***OUTPUT :***



1. ***WAP to implement Doubly Linked List.***

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*next;

struct node \*pre;

};

typedef struct node NODE;

NODE \*create(NODE \*head);

NODE \*insertstart(NODE \*head,int data);

NODE \*insertlast(NODE \*head,int data);

NODE \*insertafter(NODE \*head,int item,int data);

NODE \*insertbefore(NODE \*head,int item,int data);

NODE \*deletefront(NODE \*head);

NODE \*deletelast(NODE \*head);

NODE \*delete(NODE \*head,int x);

void display(NODE \*head);

int main()

{

NODE \*head = NULL;

int ch,data,x;

printf("Enter your choice what you want to do\n");

printf("1.Create a Linked list\n");

printf("2.add an element at front\n");

printf("3.add an element at last\n");

printf("4.add an element after a specific element\n");

printf("5.add an element before a specific element\n");

printf("6.Delete an element from front\n");

printf("7.Delete an element from last\n");

printf("8.delete an specific element\n");

printf("9.display list\n");

printf("10.Exit\n");

while(1)

{

scanf("%d",&ch);

switch(ch)

{

case 1:

head = create(head);

break;

case 2:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertstart(head,data);

break;

case 3:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertlast(head,data);

break;

case 4:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertafter(head,x,data);

break;

case 5:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertbefore(head,x,data);

break;

case 6:

head = deletefront(head);

break;

case 7:

head = deletelast(head);

break;

case 8:

printf("Enter element which is to deleted: ");

scanf("%d",&x);

head = delete(head,x);

break;

case 9:

display(head);

break;

case 10:

exit(1);

}

}

return 0;

}

NODE \*create(NODE \*head)

{

if(head!=NULL)

{

NODE \*p=head,\*q;

while(p!=NULL){

q=p;

p = p->next;

free(q);

}

}

head=NULL;

int n;

printf("Enter number of nodes: ");

scanf("%d",&n);

int i,x;

printf("Enter %d values:",n);

for(i=1;i<=n;i++)

{

scanf("%d",&x);

head = insertlast(head,x);

}

return head;

}

NODE \*insertstart(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

tmp->pre = NULL;

tmp->next = head;

if(head!=NULL)

head->pre = tmp;

return tmp;

}

NODE \*insertlast(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

tmp->next = NULL;

if(head==NULL)

{

tmp->pre = NULL;

return tmp;

}

NODE \*p = head;

while(p->next!=NULL)

p = p->next;

p->next = tmp;

tmp->pre = p;

return head;

}

NODE \*insertafter(NODE \*head,int item,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head==NULL || item==-1){

tmp->next = head;

tmp->pre = NULL;

if(head!=NULL)

head->pre = tmp;

return tmp;

}

NODE \*p=head;

while(p->next!=NULL && p->info!=item)

p = p->next;

if(p->next==NULL && p->info!=item)

{

printf("Element is not found,So inserted at front\n");

tmp->next = head;

tmp->pre = NULL;

head->pre = tmp;

return tmp;

}

else

{

tmp->pre = p;

tmp->next = p->next;

if(p->next!=NULL)

p->next->pre = tmp;

p->next = tmp;

return head;

}

}

NODE \*insertbefore(NODE \*head,int item,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head==NULL)

{

tmp->next = tmp->pre=NULL;

return tmp;

}

NODE \*p=head;

while(p->next!=NULL && p->info!=item)

p = p->next;

if((p==head)||(p->next==NULL && p->info!=item))

{

if(p->info!=item)

printf("Element is not found,So inserted at front\n");

tmp->next = head;

tmp->pre = NULL;

head->pre = tmp;

return tmp;

}

p->pre->next = tmp;

tmp->pre = p->pre;

tmp->next = p;

p->pre = tmp;

return head;

}

NODE \*deletefront(NODE \*head)

{

if(head==NULL){

printf("List is empty\n");

return NULL;

}

if(head->next!=NULL)

head->next->pre = NULL;

NODE \*p = head->next;

free(head);

return p;

}

NODE \*deletelast(NODE \*head)

{

if(head==NULL)

{

printf("List is empty\n");

return NULL;

}

if(head->next==NULL)

{

free(head);

return NULL;

}

NODE \*p = head;

while(p->next!=NULL)

p=p->next;

p->pre->next = NULL;

free(p);

return head;

}

NODE \*delete(NODE \*head,int x)

{

if(head==NULL)

{

printf("List is empty\n");

return NULL;

}

NODE \*p=head;

while(p->next!=NULL && p->info!=x)

p = p->next;

if(p->next==NULL && p->info!=x){

printf("Element is not found\n");

return head;

}

else

{

NODE \*q = head;

if(head->next==NULL)

{

free(head);

return NULL;

}

if(p!=head)

p->pre->next = p->next;

if(p->next!=NULL)

p->next->pre = p->pre;

if(p==head)

q = head->next;

free(p);

return q;

}

}

void display(NODE \*head)

{

if(head==NULL)

printf("List is empty\n");

else

{

NODE \*p = head;

while(p!=NULL){

printf("%d ",p->info);

p = p->next;

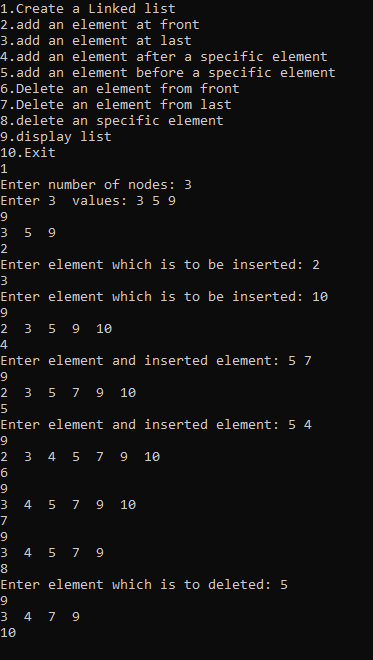
}

printf("\n");

}

}

***OUTPUT :***



1. ***WAP to implement Circular Doubly Linked List***

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*next;

struct node \*pre;

};

typedef struct node NODE;

NODE \*last=NULL;

NODE \*create(NODE \*head);

NODE \*insertstart(NODE \*head,int data);

NODE \*insertlast(NODE \*head,int data);

NODE \*insertafter(NODE \*head,int item,int data);

NODE \*insertbefore(NODE \*head,int item,int data);

NODE \*deletefront(NODE \*head);

NODE \*deletelast(NODE \*head);

NODE \*delete(NODE \*head,int x);

void display(NODE \*head);

int main()

{

NODE \*head = NULL;

int ch,data,x;

printf("Enter your choice what you want to do\n");

printf("1.Create a Linked list\n");

printf("2.add an element at front\n");

printf("3.add an element at last\n");

printf("4.add an element after a specific element\n");

printf("5.add an element before a specific element\n");

printf("6.Delete an element from front\n");

printf("7.Delete an element from last\n");

printf("8.delete an specific element\n");

printf("9.display list\n");

printf("10.Exit\n");

while(1)

{

scanf("%d",&ch);

switch(ch)

{

case 1:

head = create(head);

break;

case 2:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertstart(head,data);

break;

case 3:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertlast(head,data);

break;

case 4:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertafter(head,x,data);

break;

case 5:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertbefore(head,x,data);

break;

case 6:

head = deletefront(head);

break;

case 7:

head = deletelast(head);

break;

case 8:

printf("Enter element which is to deleted: ");

scanf("%d",&x);

head = delete(head,x);

break;

case 9:

display(head);

break;

case 10:

exit(1);

}

}

return 0;

}

NODE \*create(NODE \*head)

{

if(head!=NULL)

{

NODE \*p=head,\*q;

do{

q=p;

p=p->next;

free(q);

}while(p!=head);

}

head=NULL;

int n;

printf("Enter number of nodes: ");

scanf("%d",&n);

int i,x;

printf("Enter %d values:",n);

for(i=1;i<=n;i++)

{

scanf("%d",&x);

head = insertlast(head,x);

}

return head;

}

NODE \*insertstart(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head==NULL)

{

tmp->next = tmp->pre = tmp;

return (last = tmp);

}

tmp->pre = last;

tmp->next = head;

head->pre = tmp;

last->next = tmp;

return tmp;

}

NODE \*insertlast(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head==NULL)

{

tmp->pre =tmp->next = tmp;

return (last=tmp);

}

NODE \*p = head;

while(p->next!=head)

p = p->next;

p->next = tmp;

tmp->pre = p;

tmp->next = head;

head->pre = tmp;

last = tmp;

return head;

}

NODE \*insertafter(NODE \*head,int item,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head==NULL){

tmp->next = tmp->pre = tmp;

return (last=tmp);

}

NODE \*p=head;

while(p->next!=NULL && p->info!=item)

p = p->next;

if(p->next==NULL && p->info!=item)

{

printf("Element is not found,So inserted at front\n");

tmp->next = head;

tmp->pre = last;

head->pre = tmp;

last->next = tmp;

return tmp;

}

else

{

tmp->pre = p;

tmp->next = p->next;

p->next->pre = tmp;

p->next = tmp;

if(p==last)

last = p;

return head;

}

}

NODE \*insertbefore(NODE \*head,int item,int data)

{

NODE \*tmp=(NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head==NULL)

{

tmp->next = tmp->pre = tmp;

return tmp;

}

NODE \*p=head;

while(p->next!=head && p->info!=item)

p=p->next;

if(p->next==head && p->info!=item)

{

printf("element is not found,So inserted at front\n");

tmp->next = head;

head->pre = last->next = tmp;

tmp->pre = last;

return tmp;

}

p->pre->next = tmp;

tmp->pre = p->pre;

tmp->next = p;

p->pre = tmp;

if(p==head)

return tmp;

return head;

}

NODE \*deletefront(NODE \*head)

{

if(head==NULL){

printf("List is empty\n");

return NULL;

}

if(head->next==head)

{

free(head);

return (last=NULL);

}

NODE \*p=head->next;

last->next = head->next;

head->next->pre = last;

free(head);

return p;

}

NODE \*deletelast(NODE \*head)

{

if(head==NULL)

{

printf("List is empty\n");

return NULL;

}

if(head->next==head)

{

free(head);

return NULL;

}

NODE \*p = last;

last->pre->next = head;

head->pre = last->pre;

last = last->pre;

free(p);

return head;

}

NODE \*delete(NODE \*head,int x)

{

if(head==NULL)

{

printf("List is empty\n");

return NULL;

}

NODE \*p=head;

while(p->next!=head && p->info!=x)

p = p->next;

if(p->next==head && p->info!=x){

printf("Element is not found\n");

return head;

}

else

{

if(head->next==head)

{

free(head);

return NULL;

}

p->next->pre = p->pre;

p->pre->next = p->next;

if(p==head){

last = p->pre;

head = head->next;

}

free(p);

return head;

}

}

void display(NODE \*head)

{

if(head==NULL)

printf("List is empty\n");

else

{

NODE \*p = head;

do{

printf("%d ",p->info);

p = p->next;

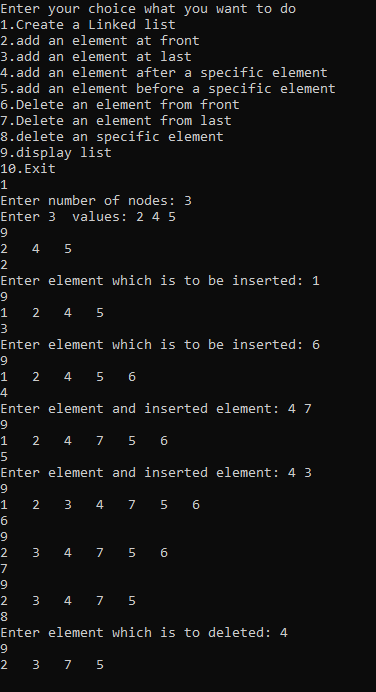
}while(p!=head);

printf("\n");

}

}

***OUTPUT:***



1. ***WAP to implement Header Doubly Linked List***

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*next;

struct node \*pre;

};

typedef struct node NODE;

NODE \*create(NODE \*head);

NODE \*insertstart(NODE \*head,int data);

NODE \*insertlast(NODE \*head,int data);

NODE \*insertafter(NODE \*head,int item,int data);

NODE \*insertbefore(NODE \*head,int item,int data);

NODE \*deletefront(NODE \*head);

NODE \*deletelast(NODE \*head);

NODE \*delete(NODE \*head,int x);

void display(NODE \*head);

int main()

{

NODE \*head = (NODE \*)malloc(sizeof(NODE));

head->next = head->pre = head;

int ch,data,x;

printf("Enter your choice what you want to do\n");

printf("1.Create a Linked list\n");

printf("2.add an element at front\n");

printf("3.add an element at last\n");

printf("4.add an element after a specific element\n");

printf("5.add an element before a specific element\n");

printf("6.Delete an element from front\n");

printf("7.Delete an element from last\n");

printf("8.delete an specific element\n");

printf("9.display list\n");

printf("10.Exit\n");

while(1)

{

scanf("%d",&ch);

switch(ch)

{

case 1:

head = create(head);

break;

case 2:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertstart(head,data);

break;

case 3:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertlast(head,data);

break;

case 4:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertafter(head,x,data);

break;

case 5:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertbefore(head,x,data);

break;

case 6:

head = deletefront(head);

break;

case 7:

head = deletelast(head);

break;

case 8:

printf("Enter element which is to deleted: ");

scanf("%d",&x);

head = delete(head,x);

break;

case 9:

display(head);

break;

case 10:

exit(1);

}

}

return 0;

}

NODE \*create(NODE \*head)

{

if(head->next!=head)

{

NODE \*p=head->next,\*q;

do{

q=p;

p=p->next;

free(q);

}while(p!=head);

}

int n;

printf("Enter number of nodes: ");

scanf("%d",&n);

int i,x;

printf("Enter %d values:",n);

for(i=1;i<=n;i++)

{

scanf("%d",&x);

head = insertlast(head,x);

}

return head;

}

NODE \*insertstart(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

tmp->next = head->next;

tmp->pre = head;

if(head->next!=head)

head->next->pre = tmp;

head->next = tmp;

return head;

}

NODE \*insertlast(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

NODE \*p=head;

while(p->next!=head)

p = p->next;

tmp->next = head;

tmp->pre = p;

p->next = tmp;

return head;

}

NODE \*insertafter(NODE \*head,int item,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head->next==head || item==-1)

{

tmp->next = head->next;

tmp->pre = head;

if(head->next!=head)

head->next->pre = tmp;

head->next = tmp;

return head;

}

NODE \*p=head->next;

while(p->next!=head && p->info!=item)

p = p->next;

if(p->next==head && p->info!=item)

{

printf("Element is not found,So inserted at front\n");

tmp->next = head->next;

tmp->pre = head;

head->next->pre = tmp;

head->next = tmp;

}

else

{

tmp->next = p->next;

tmp->pre = p;

if(p->next!=head)

p->next->pre = tmp;

p->next = tmp;

}

return head;

}

NODE \*insertbefore(NODE \*head,int item,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

NODE \*p = head->next;

while(p!=head && p->info!=item)

p = p->next;

if(p==head)

{

printf("Element is not found,So inserted at front\n");

tmp->next = head->next;

tmp->pre = head;

head->next->pre = tmp;

head->next = tmp;

return head;

}

p->pre->next = tmp;

tmp->pre = p->pre;

tmp->next = p;

p->pre = tmp;

return head;

}

NODE \*deletefront(NODE \*head)

{

if(head->next==head)

{

printf("List is empty\n");

return head;

}

NODE \*p = head->next;

head->next = p->next;

if(p->next!=head)

p->next->pre = head;

free(p);

return head;

}

NODE \*deletelast(NODE \*head)

{

if(head->next==head)

{

printf("List is empty\n");

return head;

}

NODE \*p=head,\*q;

while(p->next!=head){

q = p;

p = p->next;

}

q->next = head;

free(p);

return head;

}

NODE \*delete(NODE \*head,int x)

{

if(head->next==head)

{

printf("List is empty\n");

return head;

}

NODE \*p=head->next,\*q=head;

while(p->next!=head && p->info!=x)

{

q = p;

p = p->next;

}

if(p->next==head && p->info!=x)

printf("Element is not found\n");

else

{

q->next = p->next;

if(p->next!=head)

p->next->pre = q;

free(p);

}

return head;

}

void display(NODE \*head)

{

if(head->next==head)

printf("List is empty\n");

else

{

NODE \*p = head->next;

do{

printf("%d ",p->info);

p=p->next;

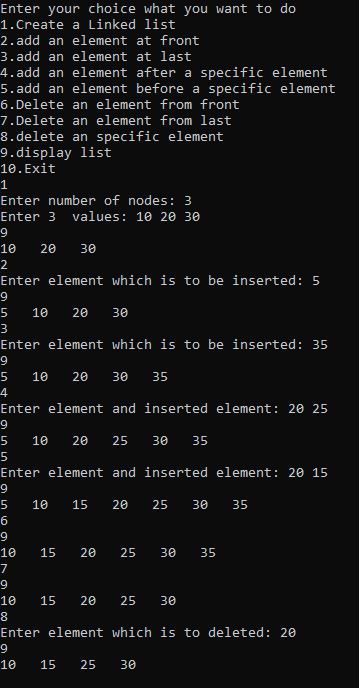
}while(p!=head);

printf("\n");

}

}

***OUTPUT:***



1. ***WAP to implement Header based Circular Doubly Linked List.***

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*next;

struct node \*pre;

};

typedef struct node NODE;

NODE \*last=NULL;

NODE \*create(NODE \*head);

NODE \*insertstart(NODE \*head,int data);

NODE \*insertlast(NODE \*head,int data);

NODE \*insertafter(NODE \*head,int item,int data);

NODE \*insertbefore(NODE \*head,int item,int data);

NODE \*deletefront(NODE \*head);

NODE \*deletelast(NODE \*head);

NODE \*delete(NODE \*head,int x);

void display(NODE \*head);

int main()

{

NODE \*head = (NODE \*)malloc(sizeof(NODE));

head->next = head->pre = head;

last = head;

int ch,data,x;

printf("Enter your choice what you want to do\n");

printf("1.Create a Linked List\n");

printf("2.add an element at front\n");

printf("3.add an element at last\n");

printf("4.add an element after a specific element\n");

printf("5.add an element before a specific element\n");

printf("6.Delete an element from front\n");

printf("7.Delete an element from last\n");

printf("8.delete an specific element\n");

printf("9.display list\n");

printf("10.Exit\n");

while(1)

{

scanf("%d",&ch);

switch(ch)

{

case 1:

head = create(head);

break;

case 2:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertstart(head,data);

break;

case 3:

printf("Enter element which is to be inserted: ");

scanf("%d",&data);

head = insertlast(head,data);

break;

case 4:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertafter(head,x,data);

break;

case 5:

printf("Enter element and inserted element: ");

scanf("%d%d",&x,&data);

head = insertbefore(head,x,data);

break;

case 6:

head = deletefront(head);

break;

case 7:

head = deletelast(head);

break;

case 8:

printf("Enter element which is to deleted: ");

scanf("%d",&x);

head = delete(head,x);

break;

case 9:

display(head);

break;

case 10:

exit(1);

}

}

return 0;

}

NODE \*create(NODE \*head)

{

if(head->next!=head)

{

NODE \*p=head->next,\*q;

do{

q=p;

p=p->next;

free(q);

}while(p!=head);

}

int n;

printf("Enter number of nodes: ");

scanf("%d",&n);

int x,i;

printf("Enter %d values : ",n);

for(i=1;i<=n;i++)

{

scanf("%d",&x);

head = insertlast(head,x);

}

return head;

}

NODE \*insertstart(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

tmp->next = head->next;

tmp->pre = head;

head->next->pre = tmp;

if(head->next==head)

{

last =tmp;

head->pre = last;

}

head->next = tmp;

return head;

}

NODE \*insertlast(NODE \*head,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

tmp->next = last->next;

tmp->pre = last;

last->next = tmp;

head->pre = tmp;

last = tmp;

return head;

}

NODE \*insertafter(NODE \*head,int item,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

if(head->next==head){

tmp->next = head->next;

tmp->pre = head;

head->next = head->pre = last = tmp;

return head;

}

NODE \*p=head->next;

while(p->next!=head && p->info!=item)

p = p->next;

if(p->next==head && p->info!=item)

{

printf("Element is not found,So inserted at front\n");

tmp->next = head->next;

tmp->pre = head;

head->next->pre = tmp;

head->next = tmp;

return head;

}

else

{

tmp->next = p->next;

tmp->pre = p;

p->next->pre = tmp;

p->next = tmp;

if(p==last)

last = tmp;

return head;

}

}

NODE \*insertbefore(NODE \*head,int item,int data)

{

NODE \*tmp = (NODE \*)malloc(sizeof(NODE));

tmp->info = data;

NODE \*p = head->next;

while(p!=head && p->info!=item)

p = p->next;

if(p==head)

{

printf("Element is not found,So inserted at front\n");

tmp->next = head->next;

tmp->pre = head;

head->next->pre = tmp;

if(head->next ==head)

last = tmp;

head->next = tmp;

return head;

}

p->pre->next = tmp;

tmp->pre = p->pre;

tmp->next = p;

p->pre = tmp;

return head;

}

NODE \*deletefront(NODE \*head)

{

if(head->next==head){

printf("List is empty\n");

return head;

}

NODE \*p = head->next;

head->next = p->next;

p->next->pre = head;

free(p);

return head;

}

NODE \*deletelast(NODE \*head)

{

if(head->next==head)

{

printf("List is empty\n");

return head;

}

NODE \*p = last;

last->pre->next = last->next;

last->next->pre = last->pre;

last = last->pre;

free(p);

return head;

}

NODE \*delete(NODE \*head,int x)

{

if(head->next==head)

{

printf("List is empty\n");

return head;

}

NODE \*p=head->next;

while(p->next!=head && p->info!=x)

p = p->next;

if(p->next==head && p->info!=x){

printf("Element is not found\n");

return head;

}

else

{

p->pre->next = p->next;

p->next->pre = p->pre;

if(p==last)

last = p->pre;

free(p);

return head;

}

}

void display(NODE \*head)

{

if(head->next==head)

printf("List is empty\n");

else

{

NODE \*p = head->next;

do{

printf("%d ",p->info);

p = p->next;

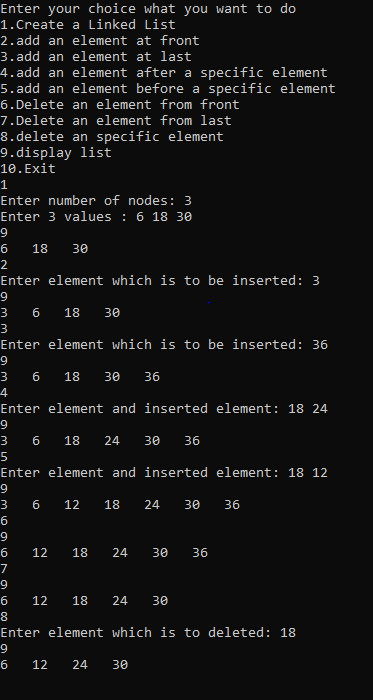
}while(p!=head);

printf("\n");

}

}

***OUTPUT:***



***/\*The End\*/***