***ASSIGNMENT – 6***

1. ***WAP to find the length of the Linked List using recursion.***

#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);

int count(NODE \*head);

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.Count number of nodes in Linked List\n");

printf("10.display list\n");

printf("11.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:

printf("Number of nodes in Linked List : %d",count(head));

break;

case 10:

display(head);

break;

case 11:

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;

}

int count(NODE \*head)

{

if(head==NULL)

return 0;

return (1 + count(head->next));

}

void display(NODE \*head)

{

NODE \*p = head;

if(p==NULL)

printf("List is empty\n");

while(p!=NULL)

{

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

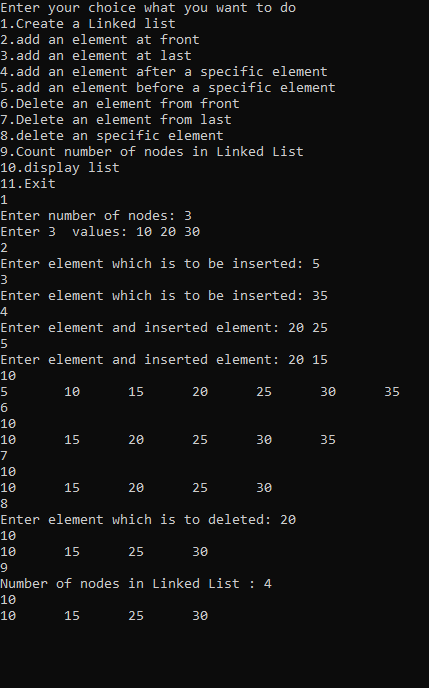
p = p->next;

}

printf("\n");

}

***OUTPUT:***



1. ***WAP to concatenate two Linked Lists.***

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*next;

};

typedef struct node NODE;

NODE \*create(NODE \*head);

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

NODE \*concat(NODE \*head1,NODE \*head2);

NODE \*deletefront(NODE \*head);

void display(NODE \*head);

int main()

{

NODE \*head2,\*head1;

head2 = head1 = NULL;

printf("Enter number of elements in first Linked List\n");

head1 = create(head1);

printf("Enter number of Elements in second Linked List\n");

head2 = create(head2);

head1 = concat(head1,head2);

display(head1);

return 0;

}

NODE \*create(NODE \*head)

{

int i,x,n;

scanf("%d",&n);

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

{

scanf("%d",&x);

head = insertlast(head,x);

}

return head;

}

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

{

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

tmp->info = data;

tmp->next = NULL;

if(head==NULL)

return tmp;

NODE \*p=head;

while(p->next!=NULL)

p = p->next;

p->next = tmp;

return head;

}

NODE \*deletefront(NODE \*head)

{

NODE \*p=head;

head = head->next;

free(p);

return head;

}

NODE \*concat(NODE \*head1,NODE \*head2)

{

if(head2==NULL)

return head1;

head1 = insertlast(head1,head2->info);

return concat(head1,head2->next);

}

void display(NODE \*head)

{

NODE \*p = head;

if(p==NULL)

printf("List is empty\n");

while(p!=NULL)

{

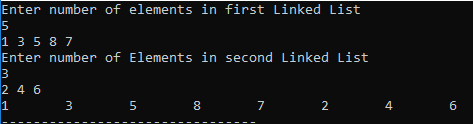
printf("%d\t",p->info);

p = p->next;

}

}

***OUTPUT:***



1. ***WAP to delete duplicate elements from a sorted Linked List.***

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*next;

};

typedef struct node NODE;

NODE \*create(NODE \*head);

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

NODE \*deldupli(NODE \*head);

void display(NODE \*head);

int main()

{

NODE \*head = NULL;

head = create(head);

int ch;

while(1)

{

printf("1.Delete Duplicate Element from Linked List\n");

printf("2.Display the Linked List\n");

printf("3.Exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

head = deldupli(head);

break;

case 2:

display(head);

break;

case 3:

exit(1);

}

}

return 0;

}

NODE \*create(NODE \*head)

{

int i,x,n;

printf("Enter number of elements in Linked list\n");

scanf("%d",&n);

printf("Enter %d elements of Linked List(In shorted order) : ",n);

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

{

scanf("%d",&x);

head = insertlast(head,x);

}

return head;

}

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

{

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

tmp->info = data;

tmp->next = NULL;

if(head==NULL)

return tmp;

NODE \*p=head;

while(p->next!=NULL)

p = p->next;

p->next = tmp;

return head;

}

NODE \*deldupli(NODE \*head)

{

if(head==NULL)

{

printf("Linked List is empty\n");

return head;

}

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

while(q!=NULL)

{

if(p->info==q->info)

{

p->next = q->next;

free(q);

q=p->next;

}

else

{

p = q;

q = p->next;

}

}

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);

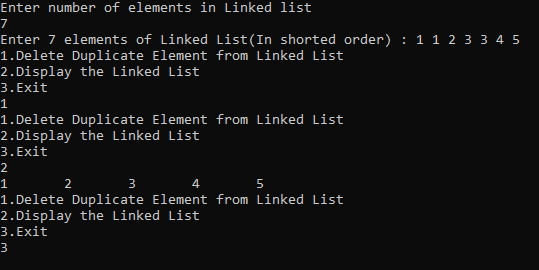
p = p->next;

}

printf("\n");

}

***OUTPUT:***



***4. WAP to delete every alternate element 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 \*insertlast(NODE \*head,int data);

NODE \*delalt(NODE \*head);

void display(NODE \*head);

int main()

{

NODE \*head = NULL;

head = create(head);

int ch;

while(1)

{

printf("1.Delete alternate Elements of Linked List\n");

printf("2.Display the Linked List\n");

printf("3.Exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

head = delalt(head);

break;

case 2:

display(head);

break;

case 3:

exit(1);

}

}

return 0;

}

NODE \*create(NODE \*head)

{

int i,x,n;

printf("Enter number of elements in Linked list\n");

scanf("%d",&n);

printf("Enter %d elements of Linked List(In shorted order) : ",n);

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

{

scanf("%d",&x);

head = insertlast(head,x);

}

return head;

}

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

{

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

tmp->info = data;

tmp->next = NULL;

if(head==NULL)

return tmp;

NODE \*p=head;

while(p->next!=NULL)

p = p->next;

p->next = tmp;

return head;

}

NODE \*delalt(NODE \*head)

{

if(head==NULL)

{

printf("List is empty\n");

return head;

}

NODE \*p=head,\*q;

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

{

q = p->next;

p->next = q->next;

p = p->next;

free(q);

}

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);

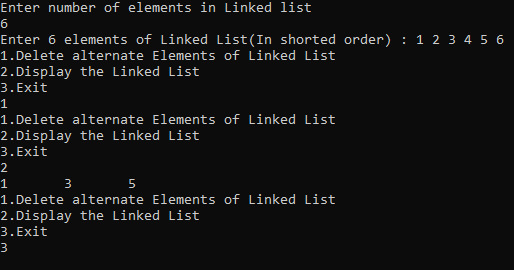
p = p->next;

}

printf("\n");

}

***OUTPUT:***



1. ***WAP to check whether the Linked List is a palindrome or not.***

#include<stdio.h>

#include<stdlib.h>

#define SIZE 10

struct stack{

int arr[SIZE];

int top;

}s;

struct node{

int info;

struct node \*next;

};

typedef struct node NODE;

NODE \*create(NODE \*head);

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

void push(int data);

int pop();

int main()

{

s.top = -1;

NODE \*head = NULL;

head = create(head);

int flag=0;

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

if(q!=NULL){

while(q->next!=NULL && q->next->next!=NULL)

{

p=p->next;

q=q->next->next;

}

if(q->next!=NULL)

flag=1;

q=head;

while(q!=p->next)

{

push(q->info);

q=q->next;

}

if(flag)

q=q->next;

while(q!=NULL){

if(q->info!=pop())

break;

q=q->next;

}

}

if(q==NULL)

printf("Linked List is palindrome\n");

else

printf("Linked List is not palindrome\n");

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;

}

void push(int data)

{

s.arr[++(s.top)] = data;

}

int pop()

{

return s.arr[(s.top)--];

}

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)

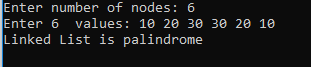
p = p->next;

p->next = tmp;

return head;

}

***OUTPUT:***



***6. WAP to reverse a Linked List.***

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*next;

};

typedef struct node NODE;

NODE \*create(NODE \*head);

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

NODE \*reverse(NODE \*head);

void display(NODE \*head);

int main()

{

NODE \*head = NULL;

head = create(head);

int ch;

while(1)

{

printf("1.Reverse the Linked List\n");

printf("2.Display the Linked List\n");

printf("3.Exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

head = reverse(head);

break;

case 2:

display(head);

break;

case 3:

exit(1);

}

}

return 0;

}

NODE \*create(NODE \*head)

{

int i,x,n;

printf("Enter number of elements in Linked list\n");

scanf("%d",&n);

printf("Enter %d elements of Linked List(In shorted order) : ",n);

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

{

scanf("%d",&x);

head = insertlast(head,x);

}

return head;

}

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

{

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

tmp->info = data;

tmp->next = NULL;

if(head==NULL)

return tmp;

NODE \*p=head;

while(p->next!=NULL)

p = p->next;

p->next = tmp;

return head;

}

NODE \*reverse(NODE \*head)

{

if(head==NULL)

{

printf("List is empty\n");

return head;

}

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

while(q!=NULL)

{

r = q->next;

q->next = p;

p = q;

q = r;

}

return p;

}

void display(NODE \*head)

{

NODE \*p = head;

if(p==NULL)

printf("List is empty\n");

while(p!=NULL)

{

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

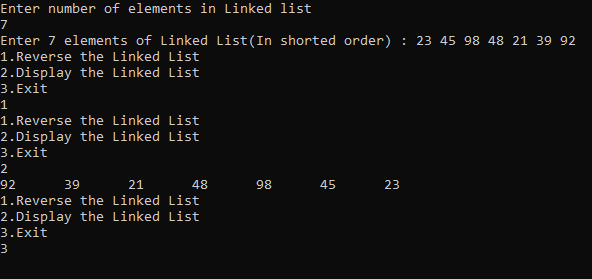
p = p->next;

}

printf("\n");

}

***OUTPUT:***



***7. WAP to merge two sorted lists.***

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*next;

};

typedef struct node NODE;

NODE \*create(NODE \*head);

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

NODE \*merge(NODE \*head1,NODE \*head2);

NODE \*deletefront(NODE \*head);

void display(NODE \*head);

int main()

{

NODE \*head2,\*head1;

head2 = head1 = NULL;

head1 = create(head1);

head2 = create(head2);

NODE \*head3 = merge(head1,head2);

printf("After merging The Linked Lists : \n");

display(head3);

return 0;

}

NODE \*create(NODE \*head)

{

int i,x,n;

printf("Enter number of elements in Linked List : ");

scanf("%d",&n);

printf("Enter %d elements(In shorted order) : ",n);

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

{

scanf("%d",&x);

head = insertlast(head,x);

}

return head;

}

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

{

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

tmp->info = data;

tmp->next = NULL;

if(head==NULL)

return tmp;

NODE \*p=head;

while(p->next!=NULL)

p = p->next;

p->next = tmp;

return head;

}

NODE \*deletefront(NODE \*head)

{

NODE \*p=head;

head = head->next;

free(p);

return head;

}

NODE \*merge(NODE \*head1,NODE \*head2)

{

NODE \*head3 = NULL;

NODE \*p = head1,\*q = head2;

while(p!=NULL && q!=NULL)

{

if(p->info > q->info)

{

head3 = insertlast(head3,q->info);

q = q->next;

head2 = deletefront(head2);

}

else if(p->info < q->info)

{

head3 = insertlast(head3,p->info);

p = p->next;

head1 = deletefront(head1);

}

else

{

head3 = insertlast(head3,p->info);

p = p->next;

q = q->next;

head2 = deletefront(head2);

head1 = deletefront(head1);

}

}

if(p!=NULL)

{

while(p!=NULL)

{

head3 = insertlast(head3,p->info);

p = p->next;

}

}

if(q!=NULL)

{

while(q!=NULL)

{

head3 = insertlast(head3,q->info);

q = q->next;

}

}

return head3;

}

void display(NODE \*head)

{

NODE \*p = head;

if(p==NULL)

printf("List is empty\n");

while(p!=NULL)

{

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

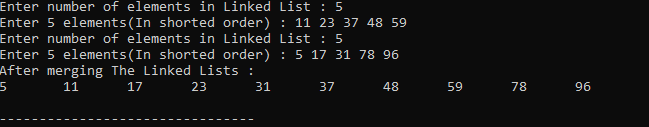
p = p->next;

}

printf("\n");

}

***OUTPUT:***



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