**DSA Lab Assignment 1**

**Linked List**

Dibyajyoti Mondal

Roll No: 001910701029

1. **Implement Singly Linear Linked List.**

**Code :**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#include<stdbool.h>

/\* Author : Dibyajyoti Mondal

B.E. ETCE UG2 DSA LAB\*/

/// singly connected linear linked list

/// create node of a linked list

struct Node{

int value;

struct Node \*next;

};

/// declaring head of a linked list

struct Node\* head=NULL;

/// insertion function

void insert\_node(int num)

{

/// dynamically allocate memory for a new node

struct Node \*NewNode=(struct Node\*)malloc(sizeof(struct Node));

NewNode->value=num;

NewNode->next=NULL;

/// if head is NULL, i.e Linked list is empty, set the new node as head

if(head==NULL)

head=NewNode;

else

{

/// traverse till the next node is NULL

struct Node \*temp=head;

while(temp->next!=NULL)

temp=temp->next;

temp->next=NewNode;

}

print\_list();

return;

}

/// deletion function

void delete\_node(int index)

{

/// index=1 means head node

if(head==NULL)

{

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

return;

}

if(index==1)

{

struct Node \*current=head;

head=head->next;

free(current);

}

else

{

struct Node \*current=head,\*prev=NULL;

int current\_index=1;

/// prev is used to keep track of previous node

while(current\_index<index)

{

if(current==NULL)

{

printf("Node out of range\n");

return;

}

prev=current;

current=current->next;

current\_index++;

}

prev->next=current->next;

free(current);

}

print\_list();

return;

}

void insert\_at\_index(int val,int index)

{

struct Node \*curr=head,\*prev=NULL;

int curr\_index=1;

struct Node \*new\_node=(struct Node \*)malloc(sizeof(struct Node));

new\_node->value=val;

new\_node->next=NULL;

if(index==1)

{

new\_node->next=head;

head=new\_node;

}

else

{

while(curr\_index<index)

{

if(curr==NULL)

{

printf("Invalid position\n");

return;

}

prev=curr;

curr=curr->next;

curr\_index++;

}

prev->next=new\_node;

new\_node->next=curr;

}

print\_list();

return;

}

/// printing the linked list

void print\_list()

{

struct Node \*current=head;

while(current!=NULL)

{

printf("%d-->",current->value);

current=current->next;

}

printf("\n");

return;

}

int main()

{

printf("====================Welcome to Singly Linear Linked List==================\n\n\n");

printf("Insert At End : 1 number\nInsert At Index: 2 number Index\nDelete At Index: 3 Index\n");

while(true)

{

int w,x,y,z;

printf("Enter a number to perform operation\n");

scanf("%d",&x);

if(x==1)

{

scanf("%d",&y);

insert\_node(y);

}

else if(x==2)

{

scanf("%d%d",&y,&z);

insert\_at\_index(y,z);

}

else if(x==3)

{

scanf("%d",&y);

delete\_node(y);

}

else

printf("Invalid Input !!!\n");

}

return 0;

}

1. **Implement Singly Circular Linked List.**

**Code :**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#include<stdbool.h>

/\* Author : Dibyajyoti Mondal

B.E. ETCE UG2 DSA LAB\*/

/// implementing singly connected circular linked list

/// defining a node

struct Node{

int value;

struct Node \*next;

};

typedef struct Node Node;

/// defining a head pointer

Node \*head=NULL;

/// inserting value

void insert\_node(int val)

{

Node \*new\_node=(Node\*)malloc(sizeof(Node));

new\_node->value=val;

new\_node->next=NULL;

if(head==NULL)

{

head=new\_node;

head->next=head;

}

else

{

Node \*curr=head;

while(curr->next!=head)

curr=curr->next;

curr->next=new\_node;

new\_node->next=head;

}

print\_list();

return;

}

/// printing linked list

void print\_list()

{

Node \*curr=head;

if(head==NULL)

{

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

return;

}

while(curr->next!=head)

{

printf("%d-->",curr->value);

curr=curr->next;

}

printf("%d\n",curr->value);

return;

}

/// deleting node in a circular linked list

void delete\_node(int index)

{

Node \*prev=NULL,\*curr=head;

int curr\_index=1;

/// list is empty

if(head==NULL)

{

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

return;

}

/// head is the only node

else if(head->next==head && index==1)

{

head=NULL;

free(curr);

printf("all elements deleted, list is now empty!!!\n");

return;

}

else if(head->next!=head && index==1)

{

while(curr->next!=head)

{

curr=curr->next;

}

curr->next=head->next;

Node \*temp=head;

head=head->next;

free(temp);

print\_list();

return;

}

else

{

while(curr\_index<index)

{

/// end of cycle

if(curr->next==head)

{

printf("index out of range!\n");

return;

}

else

{

prev=curr;

curr=curr->next;

curr\_index++;

}

}

prev->next=curr->next;

free(curr);

print\_list();

return;

}

}

/// insert at index

void insert\_at\_index(int val,int index)

{

/// dynamically allocating memory

Node \*new\_node=(Node\*)malloc(sizeof(Node));

new\_node->value=val;

new\_node->next=NULL;

/// insertion in the beginning

if(index==1)

{

new\_node->next=head;

Node \*curr=head;

while(curr->next!=head)

curr=curr->next;

curr->next=new\_node;

head=new\_node;

}

else

{

int curr\_index=1;

Node \*curr=head,\*prev=NULL;

while(curr\_index<index)

{

if(curr->next==head)

{

printf("index out of range!\n");

return;

}

else

{

prev=curr;

curr=curr->next;

curr\_index++;

}

}

prev->next=new\_node;

new\_node->next=curr;

}

print\_list();

return;

}

int main()

{

printf("====================Welcome to Singly Circular Linked List==================\n\n\n");

printf("Insert At End : 1 number\nInsert At Index: 2 number Index\nDelete At Index: 3 Index\n");

while(true)

{

int w,x,y,z;

printf("Enter a number to perform operation\n");

scanf("%d",&x);

if(x==1)

{

scanf("%d",&y);

insert\_node(y);

}

else if(x==2)

{

scanf("%d%d",&y,&z);

insert\_at\_index(y,z);

}

else if(x==3)

{

scanf("%d",&y);

delete\_node(y);

}

else

printf("Invalid Input !!!\n");

}

return 0;

}

1. **Implement Doubly Linear Linked List.**

**Code :**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#include<stdbool.h>

/\* Author : Dibyajyoti Mondal

B.E. ETCE UG2 DSA LAB\*/

/// creating node

struct Node{

struct Node \*prev;

int val;

struct Node \*next;

};

typedef struct Node Node;

/// defining head Node

Node \*head=NULL;

/// insertion at end

void InsertAtEnd(int num)

{

/// dynamic memory allocation

Node \*New\_node=(Node\*)malloc(sizeof(Node));

New\_node->val=num;

New\_node->prev=NULL;

New\_node->next=NULL;

if(head==NULL)

{

head=New\_node;

}

else

{

Node \*curr=head;

while(curr->next!=NULL)

{

curr=curr->next;

}

curr->next=New\_node;

New\_node->prev=curr;

}

print\_list();

return;

}

/// insert at any index

void Insert(int num,int index)

{

/// dynamic memory allocation

Node \*New\_node=(Node\*)malloc(sizeof(Node));

New\_node->val=num;

New\_node->prev=NULL;

New\_node->next=NULL;

if(index==1)

{

if(head==NULL)

head=New\_node;

else

{

head->prev=New\_node;

New\_node->next=head;

head=New\_node;

}

}

else

{

int curr\_index=1;

Node \*curr=head;

while(curr\_index<index && curr!=NULL)

{

curr=curr->next;

curr\_index++;

}

if(curr==NULL)

{

printf("Invalid Index!!!\n");

return;

}

Node \*prev\_node=curr->prev;

prev\_node->next=New\_node;

New\_node->prev=prev\_node;

New\_node->next=curr;

curr->prev=New\_node;

}

print\_list();

return;

}

/// deleting node at given index

void DeleteNode(int index)

{

if(head==NULL)

{

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

return;

}

if(index==1)

{

Node \*temp=head;

head=head->next;

if(head!=NULL)

head->prev=NULL;

free(temp);

}

else

{

int curr\_index=1;

Node \*curr=head;

while(curr\_index<index && curr!=NULL)

{

curr=curr->next;

curr\_index++;

}

if(curr==NULL)

{

printf("Invalid Index!!!\n");

return;

}

Node \*prev\_node=curr->prev, \*next\_node=curr->next;

prev\_node->next=next\_node;

if(next\_node!=NULL)

next\_node->prev=prev\_node;

free(curr);

}

print\_list();

return;

}

/// printing linked list

void print\_list()

{

Node \*curr=head;

while(curr!=NULL)

{

printf("%d-->",curr->val);

curr=curr->next;

}

printf("\n");

return;

}

/// rev printing linked list

void reverse\_print\_list()

{

Node \*curr=head;

while(curr->next!=NULL)

curr=curr->next;

while(curr!=NULL)

{

printf("%d-->",curr->val);

curr=curr->prev;

}

return;

}

int main()

{

printf("====================Welcome to Doubly Linear Linked List==================\n\n\n");

printf("Insert At End : 1 number\nInsert At Index: 2 number Index\nDelete At Index: 3 Index\nPrint In Reverse Order: 4\n");

while(true)

{

int w,x,y,z;

printf("Enter a number to perform operation\n");

scanf("%d",&x);

if(x==1)

{

scanf("%d",&y);

InsertAtEnd(y);

}

else if(x==2)

{

scanf("%d%d",&y,&z);

Insert(y,z);

}

else if(x==3)

{

scanf("%d",&y);

DeleteNode(y);

}

else if(x==4)

{

reverse\_print\_list();

}

else

printf("Invalid Input !!!\n");

}

return 0;

}

1. **Implement Doubly Circular Linked List.**

**Code :**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#include<stdbool.h>

/\* Author : Dibyajyoti Mondal

B.E. ETCE UG2 DSA LAB\*/

struct Node{

struct Node\* prev;

int val;

struct Node\* next;

};

typedef struct Node Node;

Node \*head=NULL;

void print\_list()

{

Node \*temp=head;

while(temp->next!=head)

{

printf("%d-->",temp->val);

temp=temp->next;

}

printf("%d-->\n",temp->val);

return;

}

/// insertion at end

void insertAtEnd(int num)

{

Node \*new\_node=(Node\*)malloc(sizeof(Node));

new\_node->val=num;

new\_node->prev=NULL;

new\_node->next=NULL;

if(head==NULL)

{

head=new\_node;

head->prev=head;

head->next=head;

}

else

{

Node \*temp=head;

while(temp->next!=head)

temp=temp->next;

temp->next=new\_node;

new\_node->prev=temp;

new\_node->next=head;

head->prev=new\_node;

}

print\_list();

return;

}

/// insert at index

void insertAtIndex(int index,int num)

{

/// creating a new node

Node \*new\_node=(Node\*)malloc(sizeof(Node));

new\_node->val=num;

new\_node->prev=NULL;

new\_node->next=NULL;

int curr\_index=1;

if(index==1)

{

if(head==NULL)

{

head=new\_node;

head->prev=head;

head->next=head;

print\_list();

return;

}

head->prev->next=new\_node;

new\_node->prev=head->prev;

head->prev=new\_node;

new\_node->next=head;

head=new\_node;

}

else

{

Node \*temp=head;

while(temp->next!=head && curr\_index<index)

{

temp=temp->next;

curr\_index++;

}

if(curr\_index<index)

{

printf("Index Out of Range!!!\n");

return;

}

else

{

Node \*prev\_node=temp->prev;

prev\_node->next=new\_node;

new\_node->prev=prev\_node;

new\_node->next=temp;

temp->prev=new\_node;

}

}

print\_list();

return;

}

/// delete at end

void deleteAtEnd()

{

if(head==NULL)

{

printf("List is Empty!!!\n");

return;

}

else

{

/// 1 node only

Node \*temp;

if(head->next==head && head->prev==head)

{

temp=head;

head=NULL;

free(temp);

printf("List is now Empty!!!\n");

return;

}

else

{

/// more than 1 node

temp=head;

while(temp->next!=head)

temp=temp->next;

Node \*prev\_node=temp->prev;

prev\_node->next=head;

head->prev=prev\_node;

free(temp);

}

}

print\_list();

return;

}

/// delete at given index

void deleteAtIndex(int index)

{

if(head==NULL)

{

printf("List is Empty!!!\n");

return;

}

else

{

Node \*temp;

if(index==1)

{

/// only 1 node

if(head->next==head && head->prev==head)

{

temp=head;

head=NULL;

free(temp);

printf("List is now Empty!!!\n");

return;

}

else

{

/// more than 1 node

temp=head;

head->prev->next=head->next;

head->next->prev=head->prev;

head=head->next;

free(temp);

}

}

else

{

int curr\_index=1;

temp=head;

while(curr\_index<index && temp->next!=head)

{

temp=temp->next;

curr\_index++;

}

if(curr\_index<index)

{

printf("Index out of range!!!\n");

return;

}

else

{

Node \*prev\_node=temp->prev,\*next\_node=temp->next;

prev\_node->next=next\_node;

next\_node->prev=prev\_node;

free(temp);

}

}

}

print\_list();

return;

}

int main()

{

printf("====================Welcome to Doubly Circular Linked List==================\n\n\n");

printf("Insert At End : 1 number\nInsert At Index: 2 Index number\nDelete At Index: 3 Index\nDelete At End: 4\n");

while(true)

{

int w,x,y,z;

printf("Enter a number to perform operation\n");

scanf("%d",&x);

if(x==1)

{

scanf("%d",&y);

insertAtEnd(y);

}

else if(x==2)

{

scanf("%d%d",&y,&z);

insertAtIndex(y,z);

}

else if(x==3)

{

scanf("%d",&y);

deleteAtIndex(y);

}

else if(x==4)

{

deleteAtEnd();

}

else

printf("Invalid Input !!!\n");

}

return 0;

}