q.1 Program to create linked list, delete elements from it and display data elements.

#include <iostream>

using namespace std;

//Declare Node

struct Node{

int num;

Node \*next;

};

//Declare starting (Head) node

struct Node \*head=NULL;

//Insert node at start

void insertNode(int n){

struct Node \*newNode=new Node;

newNode->num=n;

newNode->next=head;

head=newNode;

}

//Traverse/ display all nodes (print items)

void display(){

if(head==NULL){

cout<<"List is empty!"<<endl;

return;

}

struct Node \*temp=head;

while(temp!=NULL){

cout<<temp->num<<" ";

temp=temp->next;

}

cout<<endl;

}

//delete node from start

void deleteItem(){

if(head==NULL){

cout<<"List is empty!"<<endl;

return;

}

cout<<head->num<<" is removed."<<endl;

head=head->next;

}

int main(){

display();

insertNode(10);

insertNode(20);

insertNode(30);

insertNode(40);

insertNode(50);

display();

deleteItem(); deleteItem(); deleteItem(); deleteItem(); deleteItem();

deleteItem();

display();

return 0;

}

Q.2 menu driven program for linked list using class.

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

class Node

{

public:

    int info;

    Node\* next;

};

class List:public Node

{

    Node \*first,\*last;

public:

    List()

    {

        first=NULL;

        last=NULL;

    }

    void create();

    void insert();

    void delet();

    void display();

    void search();

};

void List::create()

{

    Node \*temp;

    temp=new Node;

    int n;

    cout<<"\nEnter an Element:";

    cin>>n;

    temp->info=n;

    temp->next=NULL;

    if(first==NULL)

    {

        first=temp;

        last=first;

    }

    else

    {

        last->next=temp;

        last=temp;

    }

}

void List::insert()

{

    Node \*prev,\*cur;

    prev=NULL;

    cur=first;

    int count=1,pos,ch,n;

    Node \*temp=new Node;

    cout<<"\nEnter an Element:";

    cin>>n;

    temp->info=n;

    temp->next=NULL;

    cout<<"\nINSERT AS\n1:FIRSTNODE\n2:LASTNODE\n3:IN BETWEEN FIRST&LAST NODES";

    cout<<"\nEnter Your Choice:";

    cin>>ch;

    switch(ch)

    {

    case 1:

        temp->next=first;

        first=temp;

        break;

    case 2:

        last->next=temp;

        last=temp;

        break;

    case 3:

        cout<<"\nEnter the Position to Insert:";

        cin>>pos;

        while(count!=pos)

        {

            prev=cur;

            cur=cur->next;

            count++;

        }

        if(count==pos)

        {

            prev->next=temp;

            temp->next=cur;

        }

        else

            cout<<"\nNot Able to Insert";

        break;

    }

}

void List::delet()

{

    Node \*prev=NULL,\*cur=first;

    int count=1,pos,ch;

    cout<<"\nDELETE\n1:FIRSTNODE\n2:LASTNODE\n3:IN BETWEEN FIRST&LAST NODES";

    cout<<"\nEnter Your Choice:";

    cin>>ch;

    switch(ch)

    {

    case 1:

        if(first!=NULL)

        {

            cout<<"\nDeleted Element is "<<first->info;

            first=first->next;

        }

        else

            cout<<"\nNot Able to Delete";

        break;

    case 2:

        while(cur!=last)

        {

            prev=cur;

            cur=cur->next;

        }

        if(cur==last)

        {

            cout<<"\nDeleted Element is: "<<cur->info;

            prev->next=NULL;

            last=prev;

        }

        else

            cout<<"\nNot Able to Delete";

        break;

    case 3:

        cout<<"\nEnter the Position of Deletion:";

        cin>>pos;

        while(count!=pos)

        {

            prev=cur;

            cur=cur->next;

            count++;

        }

        if(count==pos)

        {

            cout<<"\nDeleted Element is: "<<cur->info;

            prev->next=cur->next;

        }

        else

            cout<<"\nNot Able to Delete";

        break;

    }

}

void List::display()

{

    Node \*temp=first;

    if(temp==NULL)

    {

        cout<<"\nList is Empty";

    }

    while(temp!=NULL)

    {

        cout<<temp->info;

        cout<<"-->";

        temp=temp->next;

    }

    cout<<"NULL";

}

void List::search()

{

    int value,pos=0;

    bool flag=false;

    if(first==NULL)

    {

        cout<<"List is Empty";

        return;

    }

    cout<<"Enter the Value to be Searched:";

    cin>>value;

    Node \*temp;

    temp=first;

    while(temp!=NULL)

    {

        pos++;

        if(temp->info==value)

        {

            flag=true;

            cout<<"Element"<<value<<"is Found at "<<pos<<" Position";

            return;

        }

        temp=temp->next;

    }

    if(!flag)

    {

        cout<<"Element "<<value<<" not Found in the List";

    }

}

int main()

{

    List l;

    int ch;

    while(1)

    {

        cout<<"\n\*\*\*\* MENU \*\*\*\*";

        cout<<"\n1:CREATE\n2:INSERT\n3:DELETE\n4:SEARCH\n5:DISPLAY\n6:EXIT\n";

        cout<<"\nEnter Your Choice:";

        cin>>ch;

        switch(ch)

        {

        case 1:

            l.create();

            break;

        case 2:

            l.insert();

            break;

        case 3:

            l.delet();

            break;

        case 4:

            l.search();

            break;

        case 5:

            l.display();

            break;

        case 6:

            return 0;

        }

    }

    return 0;

}

Q.3 Menu driven Program for linked list without class.

1. #include<stdio.h>
2. #include<stdlib.h>
3. **void** create(**int**);
4. **void** search();
5. struct node
6. {
7. **int** data;
8. struct node \*next;
9. };
10. struct node \*head;
11. **void** main ()
12. {
13. **int** choice,item,loc;
14. **do**
15. {
16. printf("\n1.Create\n2.Search\n3.Exit\n4.Enter your choice?");
17. scanf("%d",&choice);
18. **switch**(choice)
19. {
20. **case** 1:
21. printf("\nEnter the item\n");
22. scanf("%d",&item);
23. create(item);
24. **break**;
25. **case** 2:
26. search();
27. **case** 3:
28. exit(0);
29. **break**;
30. **default**:
31. printf("\nPlease enter valid choice\n");
32. }
34. }**while**(choice != 3);
35. }
36. **void** create(**int** item)
37. {
38. struct node \*ptr = (struct node \*)malloc(sizeof(struct node \*));
39. **if**(ptr == NULL)
40. {
41. printf("\nOVERFLOW\n");
42. }
43. **else**
44. {
45. ptr->data = item;
46. ptr->next = head;
47. head = ptr;
48. printf("\nNode inserted\n");
49. }
51. }
52. **void** search()
53. {
54. struct node \*ptr;
55. **int** item,i=0,flag;
56. ptr = head;
57. **if**(ptr == NULL)
58. {
59. printf("\nEmpty List\n");
60. }
61. **else**
62. {
63. printf("\nEnter item which you want to search?\n");
64. scanf("%d",&item);
65. **while** (ptr!=NULL)
66. {
67. **if**(ptr->data == item)
68. {
69. printf("item found at location %d ",i+1);
70. flag=0;
71. }
72. **else**
73. {
74. flag=1;
75. }
76. i++;
77. ptr = ptr -> next;
78. }
79. **if**(flag==1)
80. {
81. printf("Item not found\n");
82. }
83. }
85. }

Q.4 Program for Tree traverals.

1. #include <stdio.h>
2. #include <stdlib.h>
3. struct node {
4. int data;
5. struct node\* left;
6. struct node\* right;
7. };
8. void inorder(struct node\* root){
9. if(root == NULL) return;
10. inorder(root->left);
11. printf("%d ->", root->data);
12. inorder(root->right);
13. }
14. void preorder(struct node\* root){
15. if(root == NULL) return;
16. printf("%d ->", root->data);
17. preorder(root->left);
18. preorder(root->right);
19. }
20. void postorder(struct node\* root) {
21. if(root == NULL) return;
22. postorder(root->left);
23. postorder(root->right);
24. printf("%d ->", root->data);
25. }
26. struct node\* createNode(value){
27. struct node\* newNode = malloc(sizeof(struct node));
28. newNode->data = value;
29. newNode->left = NULL;
30. newNode->right = NULL;
31. return newNode;
32. }
33. struct node\* insertLeft(struct node \*root, int value) {
34. root->left = createNode(value);
35. return root->left;
36. }
37. struct node\* insertRight(struct node \*root, int value){
38. root->right = createNode(value);
39. return root->right;
40. }
41. int main(){
42. struct node\* root = createNode(1);
43. insertLeft(root, 12);
44. insertRight(root, 9);
46. insertLeft(root->left, 5);
47. insertRight(root->left, 6);
49. printf("Inorder traversal \n");
50. inorder(root);
51. printf("\nPreorder traversal \n");
52. preorder(root);
53. printf("\nPostorder traversal \n");
54. postorder(root);
55. }
56. The output of the code will be
57. Inorder traversal
58. 5 ->12 ->6 ->1 ->9 ->
59. Preorder traversal
60. 1 ->12 ->5 ->6 ->9 ->
61. Postorder traversal
62. 5 ->6 ->12 ->9 ->1 ->

|  |
| --- |
| Q.5 Cpp program for Tree Traversal |
|  | #include<iostream> |
|  | using namespace std; |
|  |  |
|  | struct Node { |
|  | char data; |
|  | struct Node \*left; |
|  | struct Node \*right; |
|  | }; |
|  |  |
|  | //Function to visit nodes in Preorder |
|  | void Preorder(struct Node \*root) { |
|  | // base condition for recursion |
|  | // if tree/sub-tree is empty, return and exit |
|  | if(root == NULL) return; |
|  |  |
|  | printf("%c ",root->data); // Print data |
|  | Preorder(root->left); // Visit left subtree |
|  | Preorder(root->right); // Visit right subtree |
|  | } |
|  |  |
|  | //Function to visit nodes in Inorder |
|  | void Inorder(Node \*root) { |
|  | if(root == NULL) return; |
|  |  |
|  | Inorder(root->left); //Visit left subtree |
|  | printf("%c ",root->data); //Print data |
|  | Inorder(root->right); // Visit right subtree |
|  | } |
|  |  |
|  | //Function to visit nodes in Postorder |
|  | void Postorder(Node \*root) { |
|  | if(root == NULL) return; |
|  |  |
|  | Postorder(root->left); // Visit left subtree |
|  | Postorder(root->right); // Visit right subtree |
|  | printf("%c ",root->data); // Print data |
|  | } |
|  |  |
|  | // Function to Insert Node in a Binary Search Tree |
|  | Node\* Insert(Node \*root,char data) { |
|  | if(root == NULL) { |
|  | root = new Node(); |
|  | root->data = data; |
|  | root->left = root->right = NULL; |
|  | } |
|  | else if(data <= root->data) |
|  | root->left = Insert(root->left,data); |
|  | else |
|  | root->right = Insert(root->right,data); |
|  | return root; |
|  | } |
|  |  |
|  | int main() { |
|  | /\*Code To Test the logic |
|  | Creating an example tree |
|  | M |
|  | / \ |
|  | B Q |
|  | / \ \ |
|  | A C Z |
|  | \*/ |
|  | Node\* root = NULL; |
|  | root = Insert(root,'M'); root = Insert(root,'B'); |
|  | root = Insert(root,'Q'); root = Insert(root,'Z'); |
|  | root = Insert(root,'A'); root = Insert(root,'C'); |
|  | //Print Nodes in Preorder. |
|  | cout<<"Preorder: "; |
|  | Preorder(root); |
|  | cout<<"\n"; |
|  | //Print Nodes in Inorder |
|  | cout<<"Inorder: "; |
|  | Inorder(root); |
|  | cout<<"\n"; |
|  | //Print Nodes in Postorder |
|  | cout<<"Postorder: "; |
|  | Postorder(root); |
|  | cout<<"\n"; |
|  | } |