LAB- 11

WAP to implement stack using linked List.

```
#include <iostream>

using namespace std;

class Node {

private:
    int data;
    Node *next;

public:
   ~Node() {
     Node* head = this;
     while (head != nullptr) {
       Node* next = head->next;
       delete head;
       head = next;
   }
}
```

```
static Node* createLinkedList() {
  Node *head = nullptr, *p = nullptr,
       *q = nullptr;
  int size;

  cout << "Enter the number of nodes
    you want to create: ";
  cin >> size;

  head = new Node();
  p = head;
  q = head;
  p->next = nullptr;
  p-> previous = nullptr;

  cout << "Enter value: " << endl;
  cin >> head->data;
  p->data = head->data;

  for (int i=1; i<size; i++) {
    p->next = new Node();
    q = p->next;
    q->previous = p;
    p = p->next;

    cout << "Enter values: " << endl;
  } cin >> p->data;

  p->next = nullptr;
} return head;
```

```
int display() {
  Node *q = this;

if (q == nullptr) {
 cout << "LINKED LIST IS EMPTY!!"
      << endl;
} else {
 cout << "LINKED LIST IS EMPTY!!"
      << endl;
}
 cout << "Linked List values: ".
      << endl;
 while (q != nullptr) {
  cout << "values is: " << q->data
       << endl;
    q = q->next;
 }
}

  return 0;
}

Node * PUSH() {
 Node * head = this;
 node * p = new Node();

 Cout << "Enter value: " << endl;
 cin >> p->data;
 head->p->next = head;
   head = p;
} return head;
```

```
Node * pop() {
  Node* head = this;
  Node * p = nullptr;

  if (head == nullptr) {
    cout << "STACK IS EMPTY!!" << endl;
  } else {
    p = head;
    head = p -> next;
    cout << p -> data << ": is deleted"
         << endl;

    free(p);
  }
  return head;
};

int main() {
  Node * head = nullptr;
  int choice;

  do {
    cout << endl;
    cout << "1. PUSH" << endl;
    cout << "2. POP" << endl;
    cout << "3. Display" << endl;
    cout << "4. Exit" << endl;
```

```
cout << endl << "Enter choice: ";
cin >> choice;

switch (choice) {
  case 1: head = head -> PUSH();
          break;
  case 2: head = head -> POP();
          break;
  case 3: head -> display(); break;
  case 4: exit(0);
          break;

  default: cout << endl << "
      ENTER VALID CHOICE" << endl;
      break;
}
}while (choice != 4);

delete head;

return 0;
}
```

LAB - 12

WAP to implement queue using Linked List

```
#include <iostream>
using namespace std;

class Node {
private:
  int data;
  Node* next;

public:
  Node() {
    Node* head = this;
    while (head != nullptr) {
      Node* next = head->next;
      delete head;
      head = next;
    }
  }

  int display() {
    Node* q = this;
    if (q == nullptr) {
      cout << "QUEUE IS EMPTY!!" << endl;
    }
```

```
else {
    cout << "Queue IS EMPT
    values: " << endl;
while (q != nullptr) {
    cout << "value is: " << q->data
         << endl;
}
}
return 0;
}

Node* queueInsertion() {
Node* head = this;
Node *p = new Node();
Node* q = nullptr;
int value;

cout << "Enter Value: " << endl;
cin >> value;
p->data = value;
p->next = nullptr;

if (head == nullptr) {
    head = p;
} else {
for (q = head; q->next != nullptr;
    q = q->next) {}
} q->next = p;
} return head;
```

```
cin >> choice;

switch (choice) {
  case 1: head = head->queueInsertion();
          break;
  case 2: head = head->queueDeletion();
          break;
  case 3: head->display(); break;
  case 4: exit(0); break;

  default:
    cout << endl << "ENTER VALID
      CHOICE" << endl;
      break;
  }
} while (choice != 4);

delete head;

return 0;
```

```
Node * queueDeletion() {
Node* head = this;
Node * p = nullptr;

if(head == nullptr) {
  cout << "QUEUE IS EMPTY!!" <<endl;
}else{
  p = head;
  head = p->next;
  cout << p->data << ": is deleted" <<endl;

  free(p);
}
  return head;
};

int main() {
  Node* head = nullptr;
  int choice;

  do {
    cout << endl;
    cout << "1. Insert" << endl;
    cout << "2. Delete" << endl;
    cout << "3. Display" <<endl;
    cout << "4. Exist" << endl;

    cout << "Enter choice: ";
```

LAB-13

AP to implement Binary tree traversals

```
include <iostream>
include <stack>

ing namespace std;

ct TreeNode {
int val;
TreeNode * left;
TreeNode * right;
TreeNode(int x) : val(x), left(NULL),
                  right(NULL) {}


 inOrderTraversal (TreeNode * root) {
Stack <TreeNode*> stk;
TreeNode * curr = root;

while (curr != NULL || !stk.empty()){
  while(curr != NULL) {
     stk.push (curr);
     curr = curr -> left;
}
  curr = stk.top();
  stk.pop();
```

```
cout << curr->val << " ";
curr = curr->right;
}

void preOrderTraversal(TreeNode* root){
if(root == NULL) { return; }

stack<TreeNode*> stk;
stk.push(root);

while(!stk.empty()){
TreeNode* curr = stk.top();
stk.pop();

cout << curr->val << " ";

if(curr->right != NULL){
    stk.push(curr->right);
}
if(curr->left != NULL){
    stk.push(curr->left);
}
}
```

```
void postorderTraversal(TreeNode* root){
if(root == NULL) { return; }

stack<TreeNode*> stk1, stk2;
stk1.push(root);

while(!stk1.empty()) {
  TreeNode* curr = stk1.top();
  stk1.pop();
  stk2.push(curr);

if(curr->left != NULL) {
   stk1.push(curr->left);
}
if(curr->right != NULL) {
   stk1.push(curr->right);
}

while(!stk2.empty()) {
   cout << stk2.top()->val << " ";
   stk2.pop();
}
```

```
TreeNode * buildTree() {
int val;
cout << "Enter root value() {
~~if(val);~~
cin >> val;
if(val == -1) { return NULL; }

TreeNode * root = new TreeNode(val);

cout << "Enter left child of " << val
     << ": ";
root->left = buildTree();

cout << "Enter ~~left~~ right child of "
     << val << ": ";
root->right = buildTree();

return root;
}
```

```
int main() {
    cout << "Enter the elements of the binary tree: \n";
    TreeNode* root = buildTree();

    cout << "Inorder Traversal: " << endl;
    inorderTraversal(root);
    cout << endl;

    cout << "Preorder Traversal: " << endl;
    preorderTravel(root);
    cout << endl;

    cout << "postorder Traversal: " << endl;
    postorderTraversal(root);
    cout << endl;

    return 0;
}
```

Amit 19.4.24