Module 5

Topic 1

```
#include <iostream>
using namespace std;
int n;
int stack[0];
void init_stack()
{
    cout << "Enter the stack size: ";
    cin >> n;
    stack[n];
}
void menu()
{
    cout << "***Stack Menu***" << endl;
    cout << "1. Push\n2. Pop\n3. Display\n4. Exit" << endl;
    cout << "Enter Your Option -> ";
}
class Stack
{
private:
    int top = -1;

public:
    void push()
    {
        int val;
        cout << "Enter your stack value : ";
        cin >> val;
        if (top >= n - 1)
            cout << "Stack Overflow" << endl;
        else
        {
            top++;
            stack[top] = val;
        }
    }
    void pop()
    {
        if (top <= -1)
            cout << "Stack Underflow" << endl;
        else
        {
            cout << "The poped element is : " << stack[top] << endl;
            top--;
```

```
        }
    }
    void display()
    {
        if (top >= 0)
        {
            for (int i = top; i >= 0; i--)
            {
                cout << "| " << stack[i] << " |" << endl;
                cout << "_____" << endl;
            }
        }
        else
            cout << "Stack is Empty.." << endl;
    }
};
int main()
{
    system("clear");
    Stack a;
    int m;
    while (m != 4)
    {
        menu();
        cin >> m;
        switch (m)
        {
        case 1:
            if (n == 0)
                init_stack();
            a.push();
            break;
        case 2:
            a.pop();
            break;
        case 3:
            a.display();
            break;
        case 4:
            break;
        }
    }
    return 0;
}
```

Topic 2

```
#include <iostream>
using namespace std;
int n = 5;
int stack[0];
class node
{
public:
    int data;
    node *next;
    node *previ;
};

void menu()
{
    cout << "***Stack Menu***" << endl;
    cout << "1. Push\n2. Pop\n3. Display\n4. Exit" << endl;
    cout << "Enter Your Option -> ";
}
class Stack
{
private:
    int top = -1;
    node *head, *tail;

public:
    Stack()
    {
        head = NULL;
        tail = NULL;
    }
    void push()
    {
        int val;
        cout << "Enter your stack value : ";
        cin >> val;
        if (top >= n - 1)
            cout << "Stack Overflow" << endl;
        else
        {
            node *tmp = new node;
            tmp->data = val;
            tmp->previ = NULL;
            tmp->next = NULL;
            if (head == NULL)
            {
```

```
                head = tmp;
            }
            if (tail == NULL)
            {
                tail = tmp;
                top++;
            }
            else
            {
                tail->next = tmp;
                tmp->previ = tail;
                tail = tmp;
                top++;
            }
        }
    }
    void pop()
    {
        if (top <= -1)
            cout << "Stack Underflow" << endl;
        else
        {
            cout << "The poped element is : " << tail->data << endl;
            tail = tail->previ;
            top--;
        }
    }
    void display()
    {
        node *temp = tail;
        if (top >= 0)
        {
            while (temp->previ != NULL)
            {
                cout << "| " << temp->data << " |" << endl;
                cout << "_____" << endl;
                temp = temp->previ;
            }
            cout << "| " << head->data << " |" << endl;
            cout << "_____" << endl;
        }
        else
            cout << "Stack is Empty.." << endl;
    }
};
```

```
int main()
{
    system("clear");
    Stack a;
    int m;
    while (m != 4)
    {
        menu();
        cin >> m;
        switch (m)
        {
        case 1:
            a.push();
            break;
        case 2:
            a.pop();
            break;
        case 3:
            a.display();
            break;
        case 4:
            break;
        }
    }
    return 0;
}
```

Topic 3

```
#include <iostream>
using namespace std;
class INFtoPOST
{
private:
    int top = -1;
    int stack[200];

public:
    void push(char x)
    {
        top++;
        stack[top] = x;
    }
    void pop()
    {
        stack[top--];
    }
```

```
    int precedence(char c)
    {
        if (c == '^')
            return 3;
        else if (c == '/' || c == '*')
            return 2;
        else if (c == '+' || c == '-')
            return 1;
        else
            return -1;
    }
    bool isOperand(char ch)
    {
        return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '0'
&& ch <= '9');
    }
    string reverseStr(string str)
    {
        int u = str.length();
        for (int i = 0; i < u / 2; i++)
            swap(str[i], str[u - i - 1]);
        return str;
    }
    string infixToPostfix(string infix)
    {
        int n = infix.size();
        string postfix;
        for (int i = 0; i < n; i++)
        {
            if (isOperand(infix[i]))
                postfix.push_back(infix[i]);
            else if (infix[i] == '(')
                push('(');
            else if (infix[i] == ')')
            {
                while (stack[top] != '(')
                {
                    postfix.push_back(stack[top]);
                    pop();
                }
                pop();
            }
            else
            {
```

```
                while (top != -1 && stack[top] != '(' && precedence(stack[top])
>= precedence(infix[i]))
                {
                    postfix.push_back(stack[top]);
                    pop();
                }
                push(infix[i]);
            }
        }
        while (top != -1)
        {
            postfix.push_back(stack[top]);
            pop();
        }
        return postfix;
    }
    string infixToPrefix(string infix)
    {
        infix = reverseStr(infix);
        int l = infix.size();
        for (int i = 0; i < l; i++)
        {

            if (infix[i] == '(')
            {
                infix[i] = ')';
            }
            else if (infix[i] == ')')
            {
                infix[i] = '(';
            }
        }
        infix = infixToPostfix(infix);
        return reverseStr(infix);
    }
};
int main()
{
    string infix;
    cout << "Enter your Infix Expression : ";
    getline(cin, infix);
    INFtoPOST a;
    string postfix = a.infixToPostfix(infix);
    string prefix = a.infixToPrefix(infix);
    cout << "Infix expression : " << infix << endl;
```

```
    cout << "Postfix expression : " << postfix << endl;
    cout << "Prefix expression : " << prefix << endl;
    return 0;
}

#include <iostream>
#include <cmath>
#include <stack>
using namespace std;
class Polish
{
private:
    int top = -1, top2 = -1;
    int stack[200];
    string sstack[200];

public:
    void push(int x)
    {
        top++;
        stack[top] = x;
    }
    int pop()
    {
        return stack[top--];
    }
    void spush(string x)
    {
        top2++;
        sstack[top2] = x;
    }
    string spop()
    {
        return sstack[top2--];
    }
    bool isOperand(char ch)
    {
        return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '0'
&& ch <= '9');
    }
    void postcalc(string s)
    {
        int size = s.length();
        int num, i = 0, a, b, c;
        while (i < size)
```

Topic 4

```
    {
        if (isdigit(s[i]))
        {
            num = s[i] - 48;
            push(num);
        }
        else if (isalpha(s[i]))
        {
            cout << "value determination is not possible" << endl;
            break;
        }
        else
        {
            a = pop();
            b = pop();
            switch (s[i])
            {
            case '+':
                c = a + b;
                break;
            case '-':
                c = b - a;
                break;
            case '*':
                c = a * b;
                break;
            case '/':
                c = b / a;
                break;
            case '^':
                c = pow(a, b);
                break;
            }
            push(c);
        }
        i++;
    }
    cout << "The Value of that Postfix expression : " << pop() << endl;
}
void precalc(string s)
{
    int size = s.length();
    int num, i = size - 1, a, b, c;
    while (i >= 0)
    {
```

```
        // cout << s[i] << endl;
        if (isdigit(s[i]))
        {
            num = s[i] - 48;
            push(num);
        }
        else if (isalpha(s[i]))
        {
            cout << "value determination is not possible" << endl;
            break;
        }
        else
        {
            a = pop();
            b = pop();
            switch (s[i])
            {
            case '+':
                c = a + b;
                break;
            case '-':
                c = a - b;
                break;
            case '*':
                c = a * b;
                break;
            case '/':
                c = a / b;
                break;
            case '^':
                c = pow(a, b);
                break;
            }
            push(c);
        }
        i--;
    }
    cout << "The Value of that Prefix expression : " << pop() << endl;
}
string postoinf(string postfix)
{
    int n = postfix.size();
    string infix;
    for (int i = 0; i < n; i++)
    {
```

```
            if (isOperand(postfix[i]))
            {
                string op(1, postfix[i]);
                spush(op);
            }
            else
            {
                string s1 = spop(), s2 = spop();
                spush("(" + s2 + postfix[i] + s1 + ")");
            }
        }
        return sstack[top2];
    }
    string pretoinf(string prefix)
    {
        int n = prefix.size();
        string infix;
        for (int i = n - 1; i >= 0; i--)
        {
            if (isOperand(prefix[i]))
            {
                string op(1, prefix[i]);
                spush(op);
            }
            else
            {
                string s1 = spop(), s2 = spop();
                spush("(" + s1 + prefix[i] + s2 + ")");
            }
        }
        return sstack[top2];
    }
};

int main()
{
    string s;
    int choice;
    Polish test;
    cout << "What do you gonna Input?\n1. Postfix\n2. Prefix" << endl;
    cout << "Choice : ";
    cin >> choice;
    switch (choice)
    {
    case 1:
```

```
        cout << "Enter your Postfix Expression : ";
        cin >> s;
        cout << "The infix expression is : " << test.postoinf(s) << endl;
        test.postcalc(s);
        break;
    case 2:
        cout << "Enter your Prefix Expression : ";
        cin >> s;
        cout << "The infix expression is : " << test.pretoinf(s) << endl;
        test.precalc(s);
        break;
    }
}
```