CODE 1:

#include <iostream>

using namespace std;

class Node

{

public:

    string url;

    Node \*next;

    Node()

    {

        url = "N-A";

        next = NULL;

    }

    Node(string u)

    {

        url = u;

        next = NULL;

    }

};

class Stack

{

public:

    Node \*backTop, \*forwardTop;

    Stack()

    {

        backTop = NULL;

        forwardTop = NULL;

    }

    bool emptystack()

    {

        if (backTop == NULL && forwardTop == NULL)

        {

            return true;

        }

        else if (backTop == NULL)

        {

            return true;

        }

        else

        {

            return false;

        }

    }

    bool emptyforward()

    {

        if (forwardTop == NULL)

        {

            return true;

        }

        else

        {

            return false;

        }

    }

    void pushback(Node \*n)

    {

        Node \*temp;

        if (emptystack())

        {

            backTop = n;

            // cout << "Node Pushed as Head Node\n";

        }

        else

        {

            temp = backTop;

            backTop = n;

            n->next = temp;

            // cout << "Node Pushed\n";

        }

    }

    void pushforward(Node \*n)

    {

        Node \*tempfor;

        if (emptyforward())

        {

            forwardTop = n;

            // cout << "Forward pushed\n";

        }

        else

        {

            tempfor = forwardTop;

            forwardTop = n;

            n->next = tempfor;

            // cout << "Forward pushed\n";

        }

    }

    Node \*forwardPop()

    {

        Node \*temp;

        if (emptyforward())

        {

            cout << "Forward stack is empty\n";

        }

        else

        {

            temp = forwardTop;

            forwardTop = forwardTop->next;

            return temp;

        }

    }

    void backpeek()

    {

        if (emptystack())

        {

            cout << "Stack is empty\n";

        }

        else

        {

            cout << backTop->url << endl;

        }

    }

    void forwardpeek()

    {

        if (emptyforward())

        {

            cout << "Stack is empty\n";

        }

        else

        {

            cout << forwardTop->url << endl;

        }

    }

    Node \*backpop()

    {

        Node \*temp = NULL;

        if (emptystack())

        {

            cout << "Stack is underflow\n";

            return temp;

        }

        else

        {

            temp = backTop;

            backTop = backTop->next;

            return temp;

        }

    }

};

int main()

{

    int option;

    Stack s;

    int choice;

    do

    {

        cout << "What do you want to do\n";

        cout << "1: For visiting new website\n";

        cout << "2: For going Back\n";

        cout << "3: For going Forward\n";

        cout << "0: For quiting\n";

        cin >> choice;

        if (choice == 1)

        {

            string url;

            cout << "Enter the url\n";

            cin >> url;

            Node \*n = new Node(url);

            s.pushback(n);

        }

        else if (choice == 2)

        {

            if (s.emptystack())

            {

                cout << "Stack Underflow\n";

                continue;

            }

            else if (s.backTop->next == NULL)

            {

                cout << "Already at the beginning\n";

                continue;

            }

            else

            {

                Node \*temp;

                temp = s.backpop();

                s.pushforward(temp);

                s.backpeek();

            }

        }

        else if (choice == 3)

        {

            if (s.emptyforward())

            {

                cout << "Forward stack is empty\n";

                continue;

            }

            else if (s.forwardTop->next == NULL)

            {

                cout << "Already at the end\n";

                continue;

            }

            else

            {

                Node \*temp;

                temp = s.forwardPop();

                s.pushback(temp);

                s.backpeek();

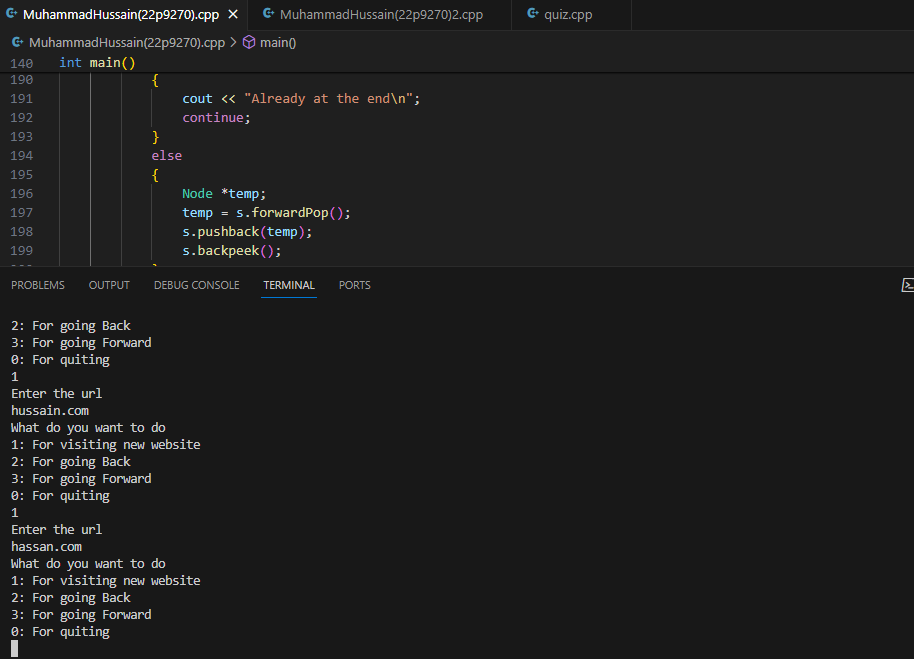
            }

        }

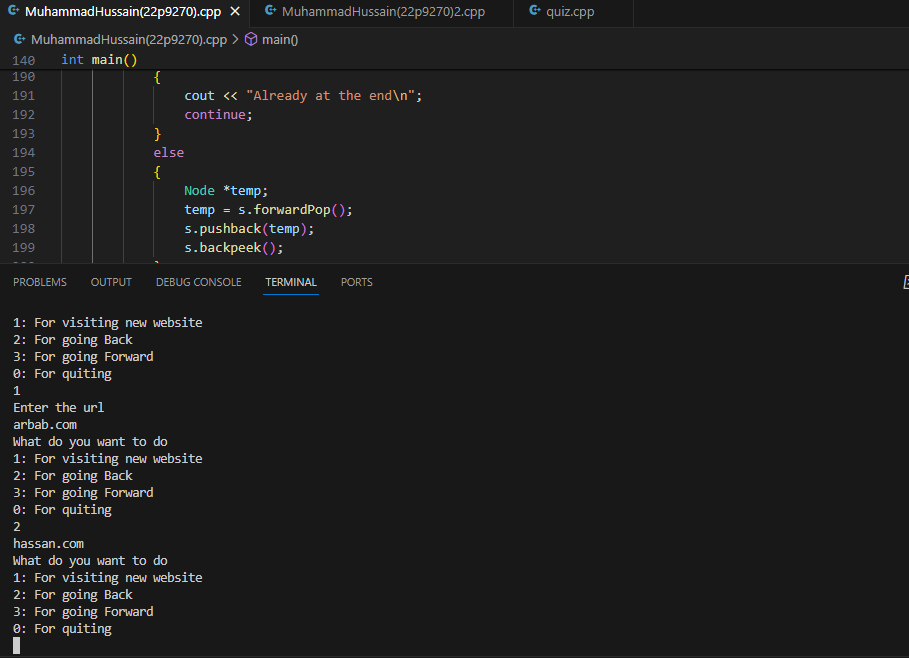
    } while (choice != 0);

    return 0;

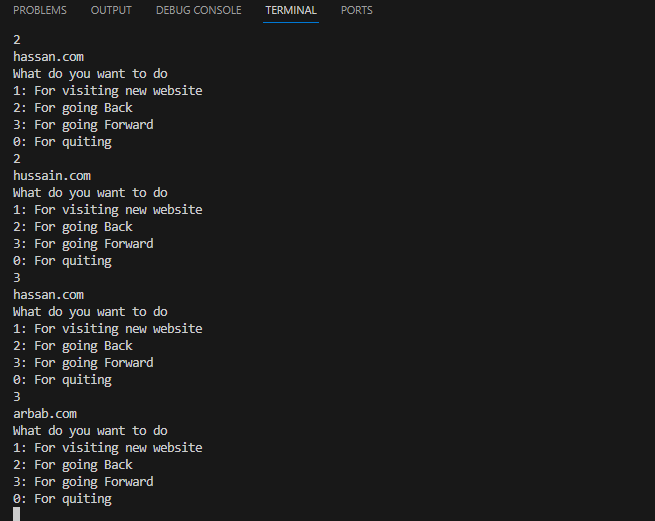
}



Enter two urls



After entering third url going back to second one



Then going back and forth for testing

CODE 2:

Test cases for checking valid and invalid :

#include <iostream>

using namespace std;

class Node

{

public:

    char data;

    Node \*next;

    Node(char d) : data(d), next(nullptr) {}

};

class Stack

{

private:

    Node \*top;

public:

    Stack() : top(nullptr) {}

    void push(char c)

    {

        Node \*newNode = new Node(c);

        newNode->next = top;

        top = newNode;

    }

    char pop()

    {

        if (isEmpty())

        {

            cerr << "Stack is empty" << endl;

            return '\0';

        }

        char popped = top->data;

        Node \*temp = top;

        top = top->next;

        delete temp;

        return popped;

    }

    char peek()

    {

        if (isEmpty())

        {

            cerr << "Stack is empty" << endl;

            return '\0';

        }

        return top->data;

    }

    bool isEmpty()

    {

        return top == nullptr;

    }

};

bool isValid(string s)

{

    Stack st;

    for (char c : s)

    {

        if (c == '(' || c == '[' || c == '{')

        {

            st.push(c);

        }

        else

        {

            if (st.isEmpty())

            {

                return false;

            }

            char topChar = st.pop();

            if ((c == ')' && topChar != '(') ||

                (c == ']' && topChar != '[') ||

                (c == '}' && topChar != '{'))

            {

                return false;

            }

        }

    }

    return st.isEmpty();

}

int main()

{

    cout << boolalpha;

    cout << isValid("()") << endl;

    cout << isValid("()[]{}") << endl;

    cout << isValid("(]") << endl;

    cout << isValid("({ }]") << endl;

    return 0;

}

