```
//Devin Hardy
//CS372
//Stack and Queue

#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdlib>

using namespace std;

typedef char v_t;

class List
{
private:

    static const int CAP = 20;

    v_t Array[CAP];
    int pos;
    int used;

    void toShift(int form, int to);
```

```
public:

    //Constructor
    List();

    //Work Methods
    bool empty();
    void first();
    void last();
    void prev();
    void next();
    int getPos();
    void setPos(int v);
    void insertBefore(v_t item);
    void insertAfter(v_t item);
    v_t getElement();
    int size();
    void replace(v_t val);
    void erase();
    void clear();

    //Overload
    bool operator==(List L1);
    bool operator!=(List L1);
    List operator+(List L1);
```

```
    void operator+=(List L1);
    void operator=(List L1);
    friend ostream& operator<<(ostream &out, List &L1);

};


class Stack
{
public:
    Stack(); // Constructor
    void push(v_t val); // Add to stack
    void pop(); // Remove from stack
    bool empty(); // Is empty?
    int size(); // Size of stack?
    v_t top(); // Return top element
    void clear(); // Clear Stack

private:
    List Lstack;

};


class Queue
```

```
{
public:
    Queue(); // Constructor
    void inqueue(v_t val); // Add to queue
    v_t dequeue(); // Remove from queue
    int size(); // Size of queue
    bool empty(); // Is empty?
    void clear(); // Clear Queue

private:
    List LQueue;

};

    ////////////////////
    // List Methods
    List::List()
    {
        v_t zero = 0;
        pos = 0;
        used = 0;
        for(int i = 0; i < CAP; i++)
        {
            Array[i] = zero;
        }
```

```
}

bool List::empty()
{
    return !used;
}

void List::first()
{
    pos = 0;
}

void List::last()
{
    pos = used - 1;
    if(used == 0)
        pos = 0;
}

void List::prev()
{
    if(used == 0)
        pos = 0;
    else if(pos < 0)
        pos = 0;
```

```
    else pos = pos - 1;
}

void List::next()
{
    if(used == 0)
        pos = 0;
    else if(pos > used)
        pos = used - 1;
    else
        pos = pos + 1;
}

int List::getPos()
{
    return pos;
}

void List::setPos(int v)
{
    if(used == 0)
        pos = 0;
    else if(v > used)
        pos = used - 1;
    else
```

```
        pos = v;
    }

    void List::insertBefore(v_t item)
    {
        if(used == 0)
        {
            used++;
            pos = 0;
            Array[pos] = item;
        }
        else
        {
            if(used == CAP)
                return;
            else
            {
                used++;

                for(int i = used-1; i > pos; i--)
                {
                    Array[i] = Array[i-1];
                }

                Array[pos] = item;
```

```
        }
    }
}

void List::insertAfter(v_t item)
{
    if(used == 0)
    {
        used++;
        pos = 0;
        Array[pos] = item;
    }
    else
    {
        if(used == CAP)
            return;
        else
        {
           used++;

            pos++;

            Array[pos] = item;
        }
    }
```

```
}

v_t List::getElement()
{
    return(Array[pos]);
}

int List::size()
{
    return (used);
}

void List::replace(v_t val)
{
    Array[pos] = val;
}

void List::erase()
{
    // Erase / Shift / Done
    if(used == 0)
        return;
    else
    {
        for(int i = pos; i < used; i++)
```

```
        {
            Array[i] = Array[i+1];
        }
        used--;
    }
    if(pos >= used)
        pos = used - 1;
}

void List::clear()
{
    used = 0;
}

////////////
//Overload

bool List::operator==(List L1)
{
    int temp;
    temp = L1.getPos();
    L1.first();
    for(int i = 0; i < used; i++)
    {
        if(Array[i] != L1.getElement())
```

```
            return 0;
        L1.next();
    }
    L1.setPos(temp);
    return 1;
}

bool List::operator!=(List L1)
{
    int temp;
    temp = L1.getPos();
    L1.first();
    for(int i = 0; i < used; i++)
    {
        if(Array[i] == L1.getElement())
            return 0;
        L1.next();
    }
    L1.setPos(temp);
    return 1;
}

List List::operator+(List L1)
{
    int temp1, temp2;
```

```
    int length;
    List TempL;
    temp1 = pos;
    temp2 = L1.getPos();
    length = L1.size();
    L1.first();
    pos = used - 1;
    for(int i = 0; i < used ; i++)
    {
        TempL.insertAfter(Array[i]);
    }
    for(int i = 0; i < length ; i++)
    {
        TempL.insertAfter(L1.getElement());
        L1.next();
    }
    pos = temp1;
    L1.setPos(temp2);
    return TempL;
}

void List::operator+=(List L1)
{
    int temp;
    int length;
```

```
    temp = L1.getPos();
    length = L1.size();
    L1.first();
    pos = used - 1;
    for(int i = 0; i < length ; i++)
    {
        this -> insertAfter(L1.getElement());
        L1.next();
    }
    L1.setPos(temp);
    return;
}

void List::operator=(List L1)
{
    int length;
    L1.first();
    length = L1.size();
    for(int i = 0; i < length ; i++)
    {
        used++;
        Array[i] = L1.getElement();
        L1.next();
    }
}
```

```
ostream& operator<<(ostream &out, List &L1)
{
    int length;
    length = L1.size();
    L1.first();
    for(int i = 0; i < length ; i++)
    {
        out << L1.getElement() << " ";
        L1.next();
    }
    return out;
}

//////////////////
// Stack Methods

Stack::Stack()
{
    Lstack.clear();
}

void Stack::push(v_t val) // Add to stack
{
    Lstack.last();
```

```
    Lstack.insertAfter(val);
}

void Stack::pop() // Remove from stack
{
    Lstack.last();
    Lstack.erase();
}

bool Stack::empty() // Is empty?
{
    return Lstack.empty();
}

int Stack::size() // Size of stack?
{
    return Lstack.size();
}

v_t Stack::top() // Return top element
{
    return Lstack.getElement();
}

void Stack::clear() // Clear Stack
```

```
{
    Lstack.clear();
}

//////////////////
// Queue Methods

Queue::Queue()
{
    LQueue.clear();
}

void Queue::inqueue(v_t val)
{
    LQueue.first();
    LQueue.insertBefore(val);
}

v_t Queue::dequeue()
{
    v_t val;
    LQueue.last();
    val = LQueue.getElement();
    LQueue.erase();
    return val;
```

```
    }

    int Queue::size()
    {
        return LQueue.size();
    }

    bool Queue::empty()
    {
        return LQueue.empty();
    }

    void Queue::clear()
    {
        LQueue.clear();
    }


int main()
  {
    ofstream outfile;
    ifstream stackFile;
    ifstream queueFile;
    outfile.open("S and Q File.out");
    stackFile.open("StackFile.txt");
```

```
queueFile.open("QueueFile.txt");

char val;
int Line = 1;
int space = 0;

Stack symbol;

Stack palin;
Queue drome;


//Stack
outfile << " Stack Check " << endl << endl;
while(stackFile.peek() != EOF)
{
    outfile << "Line " << Line++ << " : ";
    while(stackFile.peek() != '\n')
    {
        stackFile >> val;
        outfile << val << " ";
        if(val == ')' && symbol.top() == '(')
            symbol.pop();
        else if(val == ']' && symbol.top() == '[')
            symbol.pop();
```

```
            else if(val == '}' && symbol.top() == '{')
                symbol.pop();
            else
                symbol.push(val);

        }
        stackFile.get(val);
        if(!symbol.empty())
            outfile << " : This Line had matching ends" << endl;
        else
            outfile << " : This Line did not match " << endl;
        symbol.clear();
    }
    outfile << endl << endl;

    Line = 1;
    // Queue
    outfile << " Queue Check " << endl << endl;
    while(queueFile.peek() != EOF)
    {
        outfile << "Line " << Line++ << " : ";
        while(queueFile.peek() != '\n')
        {
            queueFile >> val;
            outfile << val;
```

```
            palin.push(val);
            drome.inqueue(val);

        }
        outfile << " ";
        space = drome.size();
        for(int i = 0; i < space; i++)
        {
            if(palin.top() == drome.dequeue())
                palin.pop();
        }
        if(!palin.empty())
            outfile << " : This is a Palindrome" << endl;
        else
            outfile << " : This is not a Palindrome" << endl;
        palin.clear();
        drome.clear();
        queueFile.get(val);
    }


    queueFile.close();
    stackFile.close();
    outfile.close();
return 0;
```

```
}
```