```
\title{Stacks.h}
\begin{lstlisting}
#include <string>
using namespace std;


class Stack { //constructor
    int numChar;//number of elements in struct
private:

    struct Node {   //creates linked list
        char item;
        Node* tail;
    };
Node* stackPt; // assigns pointer value

public: // initializes functions
    Stack();
    void Push(const char item);
    char Pop();
    bool emptyStack() const;
};

\end{lstlisting}

\title{Stacks.cpp}
\begin{lstlisting}
using namespace std;

Stack::Stack()// constructor when adding new stack
{
    stackPt = NULL;// not pointing to anything at all
}


bool Stack::emptyStack() const {
     return  stackPt == NULL;
    //return numChar ==  0;
}

void Stack::Push(char item) {
    Node* n = new Node;
    n -> item = item;
    if(stackPt == NULL){
        stackPt = n;
        stackPt ->tail = NULL;
    }
    else{
        n ->tail = stackPt;
        stackPt = n;
    }

}
// Create our new node
// Set the item to our passed in character
// Set the tail of the new node to the current stack head
// Set the stack head pointer to point to the new node

char Stack::Pop() {
    if (stackPt == NULL) {
        //nothing on the stack
    } else {
        Node *p = stackPt;
        stackPt = stackPt->tail;
        p->tail = NULL;
        char item = p->item;
        delete p;
        return item;
    }

}
// Get the current stack head node in a temp node variable
// If the temp node is not null
// // copy out the item from the temp node to a char variable
// // set the head of the stack to point to the tail value of our temp node
// // delete the temp node
// // return the char we got from the temp node

\end{lstlisting}


\title{Queue.h}
\begin{lstlisting}
#pragma once
#include <iostream>
#include <cstdlib>


using namespace std;
```

```
class Queue {

private:

    struct QNode {
        char item;
        QNode* tail;
    };
    QNode* queueHead;



public:
    Queue();
    char dequeue();
    void enqueue(const char item);
    bool emptyQueue() const;
};
\end{lstlisting}

\title{Queue.cpp}
\begin{lstlisting}
#include <iostream>
#include<cstdlib>
using namespace std;
#include "../Queue.h"

Queue::Queue()// constructor when adding new node
{
    queueHead = NULL;
}

bool Queue::emptyQueue() const {
    return queueHead == NULL;
    //return queChar == 0;  //checks if empty
}

//pushes
void Queue::enqueue(char queChar) {
    //QNode* temp = new QNode;  //creates new node
    //temp->item;//points to value its enqueue
    //temp-> queuePt = NULL;//makes pointer NULL
    QNode* qu = new QNode;
    qu ->item = queChar;
    qu ->tail = NULL;
    if(queueHead == NULL){  //checks if only the head remains in the queue
        queueHead = qu;
    }else{ //else ii will move values in the queue via FIFO
        QNode* temp;
        temp = queueHead;
      while(temp ->tail != nullptr){
           temp = temp -> tail ;
      }
      temp -> tail = qu;
    }
}



//removes
char Queue::dequeue() {
    if (queueHead != nullptr) {
        QNode* tem = queueHead; //if head has value it pop what ever values enqueue has
        queueHead = tem -> tail;
        char item = tem -> item;
        delete tem;
        return item;
    }
}

\end{lstlisting}

\title{main.cpp}
\begin{lstlisting}
#include <cstdlib>
#include<fstream>
#include<iostream>
#include <string>
#include <vector>
#include "../Queue.h"
#include "../Stack.h"
#pragma

using namespace std;


bool compareList(Stack& s, Queue& q) {
    bool emptyS = s.emptyStack();  //checks empty function to determine there both empty or
    bool emptyQ = q.emptyQueue();
```

```
      while (emptyQ != true && emptyS != true) {

        if (s.Pop() == q.dequeue()){      //compares char values that are being Pop or dequeu
            cout << "is Palindrome \n";
            return true;
        }else{
            cout << "is not Palindrome \n";
            return false;
        }

       }
}
    int main(int argc, char** argv) {

        Stack stack;
        Queue queue;
        ifstream letterFile;
        string text;
        vector<string> words;

        letterFile.open("magicitems.txt", ios::in);// Opens files to be read

        if (!letterFile) {
            cout << "\n Error opening file"; //checks if there is an error opening files
            exit(0);
        }

        if (letterFile.is_open()) { // opens the
            while (std::getline(letterFile, text )) {
                words.push_back(text);
                cout << text << "\n";
            }

            letterFile.close();
        }

        for (int i = 0; i < words.size(); i++) {
            for (int j = 0; j < words[i].length(); j++) {
                stack.Push(words[i].at(j));
                queue.enqueue(words[i].at(j));
            }
            compareList(stack, queue); // compares the functions

            while(stack.emptyStack() == false) // conditions that check stack and queue can
                stack.Pop();
            while(queue.emptyQueue() == false)
```

```
                queue.dequeue();
        }

    }
\end{lstlisting}
```