```
// StackString.h
// Header file for our Stack of strings Class

#pragma once
#include<string>

using namespace std;

struct Node {
    string value;
    Node *link;
};

class StackString {
public:
    // constructor
    StackString();

    // destructor
    ~StackString();

    // append value at end
    void push(string value);

    // return value at index
    string pop();

private:
    // array of elements
    Node *data;
};
```

```cpp
// StackString.cpp
// Implementation of StackString class

#include "StackString.h"

// constructor
StackString::StackString(){
    data = nullptr; // no elements yet
}

// destructor
StackString::~StackString(){
    // walk through stack and delete all elements
    while (data != nullptr) {
        Node *temp = data->link;
        delete data;
        data = temp;
    }
}

// push onto stack
void StackString::push(string value){
    Node *newNode = new Node;
    newNode->value = value;
    newNode->link = data;
    data = newNode;
}

// return the element at index
string StackString::pop(){
    string toReturn = data->value;
    Node *nextNode = data->link;
    delete data;
    data = nextNode;

    return toReturn;
}
```

```h
// QueueString.h
// Header file for our Queue of strings Class

#pragma once
#include<string>

using namespace std;

struct Node {
    string value;
    Node *link;
};

class QueueString {
public:
    // constructor
    QueueString();

    // destructor
    ~QueueString();

    // append value at end
    void enqueue(string value);

    // return value at index
    string dequeue();

private:
    Node *front;
    Node *back;
};
```

```cpp
// QueueString.cpp
// Implementation of QueueString class

#include "QueueString.h"

// constructor
QueueString::QueueString(){
    front = nullptr; // no elements yet
    back = nullptr; // no elements yet
}

// destructor
QueueString::~QueueString(){
    // walk through queue and delete all elements
    while (front != nullptr) {
        Node *temp = front->link;
        delete front;
        front = temp;
    }
}

// enqueue onto back of queue
void QueueString::enqueue(string value){
    Node *newNode = new Node;
    newNode->value = value;
    newNode->link = nullptr;
    // special case -- first node
    if (back == nullptr) {
        back = newNode;
        front = newNode;
    } else {
        back->link = newNode; // update previous back
        back = newNode; // set the back to the new node we just created
    }
}

// dequeue from front of queue
string QueueString::dequeue(){
    string toReturn = front->value;
    Node *nextNode = front->link;
    delete front;
    front = nextNode;

    return toReturn;
}
```