CSCI 2270 – Data Structures

Homework 5 - Stacks and Queues

OBJECTIVE

Implement stack using linked list
Implement queue using array

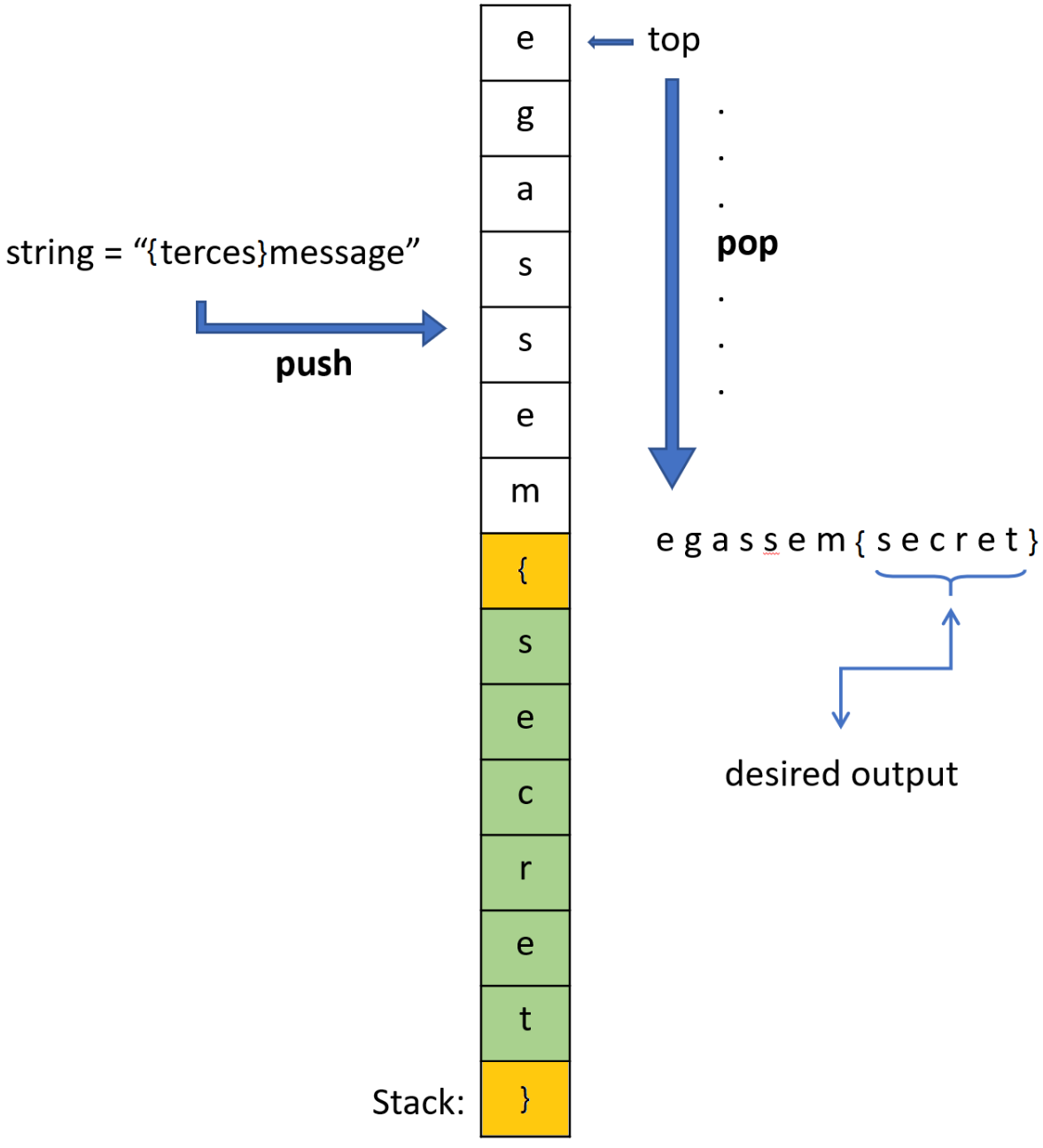
Overview

Stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. A queue is a container of objects (a linear collection) that are inserted and removed according to the first-in first-out (FIFO) principle. Both of them can be implemented using Arrays or LinkedLists. In this assignment, stacks are implemented using Linked Lists and Queues are modified to a circular queues implemented using arrays.

Part 1 - Tv Buff Decoder

You are being asked to guess a movie or tv show name. But they are unfortunately encoded in a secret message!
Using the provided Stack class implementation, write the *evaluate()* function to print secret messages directly to the terminal. Secret messages are surrounded by parenthesis inside of otherwise readable strings (e.g. "This sentence contains a {terces} message"). The secret messages are just words in reverse, which we can use a stack to flip back into a readable format. The *evaluate()* method should decode secret messages by pushing an entire message string onto a stack, and then identifying secret messages between '{' and '}' characters while popping each element off. You can assume there is only one potential secret message in any given string.

CSCI 2270 – Data Structures



MessageDecoder Class

You will build a parser that determines if there's a secret message embedded within a string. This class utilizes the functionality of a Linked List stack, which employs the following struct (included in *MessageDecoder.hpp*)

```
struct Parser
{
        char ch;
        Parser* next;
};
```

CSCI 2270 – Data Structures

Implement the MessageDecoder class according to the following specifications:

Parser* head

- ➔ Points to the top of the stack (first node in the Linked List). This is defined in the header file.

MessageDecoder()

- ➔ Constructor--set the **head** pointer to NULL

~MessageDecoder()

- ➔ Destructor--**pop** everything off the stack and set **head** to NULL

bool isEmpty()

- ➔ Returns true if the stack is empty (i.e. **head** is NULL), otherwise false

void push(char ch)

- ➔ Insert a new node with **ch** onto the top of the stack (beginning of linked list).

void pop()

- ➔ If the stack is empty, print *"Stack empty, cannot pop an item."* and return. Otherwise delete the top most item from the stack.

Parser* peek()

- ➔ If the stack is empty, print *"Stack empty, cannot peek."* and return NULL. Otherwise, return a pointer to the top of the stack.

Parser* getStackHead()

Already Implemented , no need to change.

void evaluate(std::string symbol)

- ➔ Push all the characters into the stack
- ➔ pop all the elements from the stack, while doing so, start storing the characters to a character array between “}” and “{”
- ➔ if the parenthesis matched, then print the secret message.
- ➔ else print ”No secret message”

CSCI 2270 – Data Structures

MessageDecoder main driver file

- ➔ Create a stack by instantiating an MessageDecoder object
- ➔ Take input string from terminal
- ➔ call evaluate function and pass the input string

Below are some sample outputs

```
“Watch {ylimaF nredoM ehT}”
   expected output: “The Modern Family”
“I am watching {secretmessage bye
   expected output:”No secret message”
“hellosecretmessagebye}”
   expected output: “No secret message”
“{}”
   expected output: “”
```

Instructions for the `main_1.cpp` file

- ➔ The driver code should start with printing `Enter the encoded sentence` with an endline.
- ➔ Print `#>` and then accept the encoded string from the user.
- ➔ If the string array is empty print `Nothing to evaluate`
- ➔ Otherwise call the evaluate function with the string array.
- ➔ Free any unused memory

CSCI 2270 – Data Structures

Order of function implementation

- ➔ 1. push
- ➔ 2. pop
- ➔ 3. peek
- ➔ 4. evaluate
- ➔ 5. Destructor
- ➔ 6. main

Part 2 – TV Shows Queue

For this part, you will create an application that mimics a TV show queue. Specifically, the application allows users to both add (enqueue) and retrieve (dequeue) TV shows. The implementation will require creating a data buffer in the form of a queue. You will simulate this interaction using a circular queue.

EnqueueDequeue Class

Beware of edge cases that arise from the array being circular

In this class you will build a queue using the circular array implementation. Implement the methods of **EnqueueDequeue** according to the following specifications.

std::string queue[SIZE]

- ➔ A circular queue of strings in the form of an array. **SIZE** is initialized to a default value of 20 in *EnqueueDequeue.hpp*

int queueFront

- ➔ Index in the array that keeps track of the front item

int queueEnd

- ➔ Index in the array that keeps track of the first available empty space (in case the queue is full, queueEnd points to queueFront).

int counter

- ➔ variable to keep track of the number of shows in the queue. Initialized to 0 in *EnqueueDequeue.hpp.*

EnqueueDequeue()

- ➔ Constructor--Set **queueFront** and **queueEnd** to -1

bool isEmpty()
- ➔ Return true if the queue is empty, false otherwise

bool isFull()
- ➔ Return true if the queue is full, false otherwise

void enqueue(std::string show)
- ➔ If the queue is not full, then add the **show** to the end of the queue and modify **queueFront** and/or **queueEnd** if appropriate, else print "*Queue full, cannot add a new show*"

void dequeue()
- ➔ Remove the first show from the queue if the queue is not empty and modify **queueFront** and/or **queueEnd** if appropriate. Otherwise print *"Queue empty, cannot dequeue show"*

int queueSize()
- ➔ Return the number of shows in the queue.

std::string peek()
- ➔ If the queue is empty then print "*Queue empty, cannot peek*" and return an empty string. Otherwise, return the first show in the queue.

main_2.cpp driver file

Your program will start by displaying a menu by calling the **menu()** function which is included in the provided skeleton code. The user will select an option from the menu to decide upon what the program will do, after which the menu will be displayed again. Below are the specifics of the menu *(Tip: use getline for all inputs)*

Option 1: Enqueue (Add shows to the queue) - This is an enqueue operation

- This option prompts the user to enter the number of shows being enqueued using the below print statement

```
cout << "Enter the number of shows to be
enqueued:" << endl;
```

- Then prompt the user to enter each show using the below print statement

```
cout << "Show" << <SHOW_NO> << ":" << endl;
```

- Add ***(enqueue)*** all the shows to the queue

Option 2: Dequeue (Remove shows from the queue) - This is a dequeue operation

- This option prompts the user to enter the number of shows being removed using the below print statement

```
cout << "Enter the number of shows to be dequeued:"
<< endl;
```

- Remove (***dequeue***) shows from the queue. If the number of shows to be removed is greater than the total number of shows in the queue, then remove (***dequeue***) all the available shows in the queue and notify the user with a below statement:

```
cout<< "No more shows to retrieve from queue" <<
endl;
```

This message should never be printed more than once.

- For each item retrieved, print the following

```
cout << "Retrieved:  " << show << endl;"
```

where **show** is the string you are dequeuing.

Option 3: Return the queue size and exit

- This option prompts the user to return the number of shows in the queue using the below print statement, before exiting the program.

```
cout << "Number of shows in the queue:" <<
<shows_count>;
```

If user enters any other options you need to print the below message

```
cout << “Enter a valid option (1 or 2 or 3)” << endl;
```

Below is a sample output

```
*---------------------------------------*
Choose an option:
1. Enqueue new shows (Add shows to the queue)
2. Dequeue (Retrieve shows from the queue)
3. Return the queue size and exit
*---------------------------------------*
1
Enter the number of shows to be enqueued:
3
Show1:
Breaking bad
Show2:
House of dragons
Show3:
Rings of power
*---------------------------------------*
```

```
Choose an option:
  1. Enqueue new shows (Add shows to the queue)
  2. Dequeue (Retrieve shows from the queue)
  3. Return the queue size and exit
*----------------------------------------*
2
Enter the number of shows to be retrieved: 2
Retrieved: Breaking bad
Retrieved: House of dragons
*----------------------------------------*
Choose an option:
  1.Enqueue new shows (Add shows to the queue)
  2.Dequeue (Retrieve shows from the queue)
  3.Return the queue size and exit
*----------------------------------------*
3
Number of shows in the queue:1
```

Enqueue Dequeue Starter Code

Do not modify the provided header file. You are provided with a skeleton of your driver program (*EnqueueDequeue.cpp*) which you need to complete.

Order of function implementation

- ➔ 1. queueSize
- ➔ 2. isEmpty
- ➔ 3. isFull
- ➔ 4. enqueue
- ➔ 5. dequeue
- ➔ 6. peek
- ➔ 7. main