SENG1050 – DATA STRUCTURES

ASSIGNMENT 2 – STACKS AND QUEUES USING LINKED LISTS

OVERVIEW

- The objective of this assignment is to implement simplified web browser history manager using “stacks” and “queues”.

OBJECTIVES

- Create and use stacks and queues.
- Use common algorithms to enhance software development.
- Use best practices to effectively produce quality software.

ACADEMIC INTEGRITY AND LATE PENALTIES

- Link to Academic Integrity Information
- Link to Late Policy

EVALUATION

- The evaluation of this assignment will be done as detailed in the Marking lecture in Week 2 of the C course.

PREPARATION

- Understand how stack and queue work.
- Review the linked list examples.

REQUIREMENTS

Stack Implementation

Define a structure `StackNode` to represent each node in the linked list. This structure should contain:

- A pointer to a string representing the URL visited (e.g., `char* url`).
- A pointer field to point to the next node (e.g., `struct StackNode* next`).

- Define a structure `Stack` to represent the stack itself. This structure should contain:
- A pointer to the top node of the stack (e.g., `struct StackNode* top`).

➢ **Implement the following functions**:

- `void push(Stack* stack, char* newUrl)`: Adds a new URL to the top of the stack.
- `char* pop(Stack* stack)`: Removes and returns the URL from the top of the stack.
- `char* peek(Stack* stack)`: Returns the URL from the top of the stack without removing it.
- `bool isEmpty(Stack* stack)`: Checks if the stack is empty.

Ensure proper memory management by freeing memory for nodes that are popped from the stack.

Queue Implementation

Define a structure `QueueNode` to represent each node in the linked list. This structure should contain:

- A pointer to a string representing the URL visited (e.g., `char* url`).
- A pointer field to point to the next node (e.g., `struct QueueNode* next`).
- Define a structure Queue to represent the queue itself. This structure should contain:
- Pointers to the front and rear nodes of the queue (e.g., `struct QueueNode* front`, `struct QueueNode* rear`)

➢ **Implement the following functions:**

- `void enqueue(Queue* queue, char* newUrl)`: Adds a new URL to the rear of the queue.
- `char* dequeue(Queue* queue)`: Removes and returns the URL from the front of the queue.
- `char* peek(Queue* queue)`: Returns the URL from the front of the queue without removing it.
- `bool isEmpty(Queue* queue)`: Checks if the queue is empty.

Ensure proper memory management by freeing memory for nodes that are dequeued from the queue.

Web Browser History Manager

- Use the stack and queue implementations.

- When a user visits a new web page, push the URL onto the stack and enqueue it into the queue.

➢ **Implement a menu-based user interface allowing users to**:

1. Visit a new web page and add it to the browsing history.
2. Navigate back to the previous web page.
3. Navigate forward to the next web page.
4. Display the current web page and the browsing history.
5. Exit the browser.

Ensure that navigating back and forth updates both the stack and queue appropriately to maintain consistency in browsing history.

Navigating Backward and Forward

- When navigating backward (using the back button), the program should pop the top URL from the stack to retrieve the previous page URL. This URL should then be displayed to the user, and it should be enqueued into the forward queue to support forward navigation.
- When navigating forward (using the forward button), the program should dequeue the next URL from the queue to retrieve the next page URL. This URL should then be pushed onto the stack to support backward navigation.
- Ensure that the stack and queue are updated appropriately to maintain consistency in browsing history during navigation.
- Handle edge cases such as reaching the beginning or end of the browsing history, ensuring that the user cannot navigate beyond the available pages.

Browser History Manager In Action

```
Menu:
1. Visit New Web Page
2. Navigate Backward
3. Navigate Forward
4. Display Current Page and History
5. Exit
```

```
Enter your choice: 1
Enter the URL of the web page: https://example.com
Visited: https://example.com

Menu:
1. Visit New Web Page
2. Navigate Backward
3. Navigate Forward
4. Display Current Page and History
5. Exit

Enter your choice: 2
Previous Page: https://example.com

Menu:
1. Visit New Web Page
2. Navigate Backward
3. Navigate Forward
4. Display Current Page and History
5. Exit

Enter your choice: 3
Next Page: https://example.com

Menu:
1. Visit New Web Page
2. Navigate Backward
3. Navigate Forward
4. Display Current Page and History
5. Exit

Enter your choice: 4
Current Page: https://example.com
```

```
Backward History:
1. https://example.com
Forward History:
1. https://example.com

Menu:
1. Visit New Web Page
2. Navigate Backward
3. Navigate Forward
4. Display Current Page and History
5. Exit

Enter your choice: 5
Thank you for using the web browser. Goodbye!
```

GIT REQUIREMENTS

- Use GitHub classroom link to join the repository. It is expected that you have a reasonable number of commits with meaningfully descriptive commit comments.

CHECKLIST REQUIREMENTS

- Create a requirements checklist. This should contain the specific requirements from this assignment as well as any relevant requirements that have been covered in lecture or that are found in the SET Coding Standards or SET Submission Standards. Do it in whatever form you wish. Hand in your completed checklist in PDF form as checklist.pdf. Not having this checklist will result in a cap of 80 on your mark.

FILE NAMING REQUIREMENTS

- You must call your source file m2.cpp.
- You must call your checklist checklist.pdf.

SUBMISSION REQUIREMENTS

- Do not hand in any other files.
- Submit your files to the *Assignment 2* Submission Folder.