Lab 8 - Implementing Linked Lists in C

Deadline: 24th Nov 2024

The objective of this lab is to provide students with a comprehensive understanding of circular linked lists and their practical applications in C programming. By completing this lab, students will learn the structure and functionality of circular linked lists, gaining hands-on experience in writing C code to create, insert, delete, and display nodes in a circular linked list. They will understand the practical applications of circular linked lists in real-life scenarios, such as simulating a traffic control system where each traffic signal has a specified duration. This lab will enhance students' problem-solving and debugging skills, ensuring they can write efficient and error-free code while managing memory effectively. By achieving these objectives, students will build a solid foundation in data structures and improve their ability to apply these concepts to various programming challenges.

Step 1: Define the Node Structure and Function Prototypes

File: circular_node.h

This header file will contain the definition of the node structure and the function prototypes.

```c
#ifndef CIRCULAR_NODE_H
#define CIRCULAR_NODE_H

struct Node {
    char signal[10];
    int duration;
    struct Node* next;
};

struct Node* createNode(const char* signal, int duration);
void insertAtEnd(struct Node** head, const char* signal, int duration);
void deleteNode(struct Node** head, const char* signal);
void displayList(struct Node* head);

#endif
```

Step 2: Implement the Functions

File: circular_node.c

This source file will contain the implementation of the functions declared in circular_node.h.

createNode Function creates a new node with the given signal color and duration. It allocates memory for the new node, sets the signal color and duration, and initializes the next pointer to point to itself.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "circular_node.h"

struct Node* createNode(const char* signal, int duration) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    strcpy(newNode->signal, signal);
    newNode->duration = duration;
    newNode->next = newNode;
    return newNode;

}
```

insertAtEnd function inserts a new node at the end of the circular linked list. It creates a new node with the given signal color and duration, and then updates the pointers to maintain the circular structure.

```c
// Function to insert a node at the end of the list
void insertAtEnd(struct Node** head, const char* signal, int duration) {
    struct Node* newNode = createNode(signal, duration); // Create a new node
    if (*head == NULL) {  // If the list is empty
        *head = newNode;  // Set the new node as the head
        return;
    }
    struct Node* temp = *head;  // Temporary pointer to traverse the list
    while (temp->next != *head) { // Traverse to the last node
        temp = temp->next;
    }
    temp->next = newNode;  // Set the last node's next pointer to the new node
    newNode->next = *head; // Set the new node's next pointer to the head
}
```

deleteNode deletes a node with the given signal color from the circular linked list. It handles different cases, such as deleting the head node, deleting a node in the middle, and deleting the last node.

```c
// Function to delete a node with a given signal color
void deleteNode(struct Node** head, const char* signal) {
    if (*head == NULL) return;
    struct Node *temp = *head, *prev = NULL;
    // If the head node itself holds the signal to be deleted
    if (strcmp(temp->signal, signal) == 0 && temp->next == *head) {
        *head = NULL;
        free(temp);
        return;
    }
    // If the head node itself holds the signal to be deleted and there are more nodes
    if (strcmp(temp->signal, signal) == 0) {
        while (temp->next != *head) {
            temp = temp->next;
        }
        temp->next = (*head)->next;
        free(*head);
        *head = temp->next;
        return;
    }
    // Search for the signal to be deleted, keeping track of the previous node
    while (temp->next != *head && strcmp(temp->signal, signal) != 0) {
        prev = temp;
        temp = temp->next;
    }
    // If the signal was not present in the list
    if (strcmp(temp->signal, signal) != 0) return;

    // Unlink the node from the list
    prev->next = temp->next;
    free(temp);  // Free the memory of the node to be deleted
}
```

displayList displays the circular linked list. It prints the signal color and duration of each node, indicating the circular nature of the list by showing the head node at the end.

```c
// Function to display the list
void displayList(struct Node* head) {
    if (head == NULL) return;
    struct Node* temp = head;
    do {
        printf("%s (%d seconds) -> ", temp->signal, temp->duration);  // Print the signal color and duration of each node
        temp = temp->next;
    } while (temp != head);
    printf("%s (head)\n", head->signal);  // Indicate the circular nature of the list
}
```

Step 3: Demonstrate the Circular Linked List Operations

File: main.c

This source file contains the main function, which demonstrates the operations of the circular linked list. The main function initializes the head of the list, inserts nodes representing traffic signals with their respective durations, displays the list, deletes a node, and then displays the updated list.

Complete the main.c file by following these instructions:

1. Initialize the Head:
    - Set the head of the list to NULL.
2. Insert Nodes:
    - Insert a node with the “Red” signal and a duration of 30 seconds.
    - Insert a node with the “Green” signal and a duration of 45 seconds.
    - Insert a node with the “Yellow” signal and a duration of 10 seconds.
3. Display the List:
    - Display the circular linked list to show the sequence of traffic signals and their durations.
4. Delete a Node:
    - Remove the node with the “Green” signal from the list.
5. Display the List After Deletion:
    - Display the circular linked list again to show the updated sequence of traffic signals.

Step 4: Create a MakefileFile: Makefile

Complete the Makefile by following the template we have been using since week 4. Ensure it includes the necessary rules to compile and link the source files into an executable.

Compilation and Execution

To compile and run the program, you can use the following commands:

make

./circular_linkedlist

Expected Output

```
Circular linked list (traffic signals):
Red (30 seconds) -> Green (45 seconds) -> Yellow (10 seconds) -> Red (head)
Circular linked list after deletion:
Red (30 seconds) -> Yellow (10 seconds) -> Red (head)
```

Bonus Activity

As a bonus activity, you can either create a new feature or make small changes to the existing code. Here are a few suggestions:

1. **Add a New Signal:**
    - Insert a new traffic signal with a different color and duration into the circular linked list.
2. **Modify Signal Durations:**
    - Update the duration of an existing traffic signal in the list.
3. **Cycle Through Signals:**
    - Implement a function that simulates cycling through the traffic signals, printing each signal and its duration in sequence.

Feel free to choose any of these activities or come up with your own creative enhancements to the circular linked list implementation.

What to Submit

1. **Source Code Files:**
    - circular_node.h
    - circular_node.c
    - main.c
    - Makefile
2. **Bonus Activity (Optional):**

- If you completed the bonus activity, include the modified source code and a description of the changes or new features added.

Marking Criteria

1. **Completeness of Given Code (8 marks):**
   - The code correctly implements the circular linked list operations as specified.
   - The program compiles and runs without errors.
   - The expected output matches the provided example.
2. **Completion of main.c and Makefile (2 marks):**
   - The main.c file is completed as per the instructions.
   - The Makefile is correctly set up to compile and link the source files.
3. **Bonus Activity (2 marks):**
   - Successful implementation of the bonus activity.
   - Clear explanation of the new feature or changes made.