Module 2 Assessment

By Shailesh Babu.M

1. Write a C program to remove duplicate element from sorted Linked List.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
        int data;
        struct Node* next;
};
void removeDuplicates(struct Node* head)
{
        struct Node* current = head;
        struct Node* next_next;
        if (current == NULL)
                return;
        while (current->next != NULL) {
                if (current->data == current->next->data) {
                        next_next = current->next->next;
                        free(current->next);
                        current->next = next_next;
                }
                {
                        current = current->next;
                }
        }
}

void push(struct Node** head_ref, int new_data)
{
        struct Node* new_node
                = (struct Node*)malloc(sizeof(struct Node));
        new_node->data = new_data;
        new_node->next = (*head_ref);
        (*head_ref) = new_node;
}
void printList(struct Node* node)
{
        while (node != NULL) {
                printf("%d ", node->data);
                node = node->next;
        }
}
int main()
{
        struct Node* head = NULL;
        push(&head, 20);
        push(&head, 13);
        push(&head, 13);
        push(&head, 11);
        push(&head, 11);
        push(&head, 11);

        printf("\n Linked list before duplicate removal \n");
        printList(head);
        removeDuplicates(head);
        printf("\n Linked list after duplicate removal \n");
        printList(head);
        return 0;
}
```

2. Write a C program to rotate a doubly linked list by N nodes.

```
#include <stdio.h>
struct node{
    int data;
    struct node *previous;
    struct node *next;
};
int size = 0;
struct node *head, *tail = NULL;
void addNode(int data) {
    struct node *newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = data;
    if(head == NULL) {
        head = tail = newNode;
        head->previous = NULL;
        tail->next = NULL;
    }
    else {
        tail->next = newNode;
        newNode->previous = tail;
        tail = newNode;
        tail->next = NULL;
    }
    size++;
}
```

```
void rotateList(int n) {
    struct node *current = head;
    if(n == 0 || n >= size)
        return;
    else {
        for(int i = 1; i < n; i++)
            current = current->next;
        tail->next = head;
        head = current->next;
        head->previous = NULL;
        tail = current;
        tail->next = NULL;
    }
}
void display() {
    struct node *current = head;
    if(head == NULL) {
        printf("List is empty\n");
        return;
    }
    while(current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main()
{
    addNode(1);
    addNode(2);
    addNode(3);
    addNode(4);
    addNode(5);
    printf("Original List: \n");
    display();
    rotateList(3);
    printf("Updated List: \n");
    display();
    return 0;
}
```

3. Write a C program to sort the elements of a queue in ascending order.

```
#include <stdio.h>
int main()
{
   //Initialize array
   int arr[] = {5, 2, 8, 7, 1};
   int temp = 0;

   //Calculate length of array arr
   int length = sizeof(arr)/sizeof(arr[0]);

   //Displaying elements of original array
   printf("Elements of original array: \n");
   for (int i = 0; i < length; i++) {
      printf("%d ", arr[i]);
   }

   //Sort the array in ascending order
   for (int i = 0; i < length; i++) {
      for (int j = i+1; j < length; j++) {
        if(arr[i] > arr[j]) {
           temp = arr[i];
           arr[i] = arr[j];
           arr[j] = temp;
        }
      }
   }

   printf("\n");

   //Displaying elements of array after sorting
   printf("Elements of array sorted in ascending order: \n");
   for (int i = 0; i < length; i++) {
      printf("%d ", arr[i]);
   }
   return 0;
}
```

4. List all queue function operations available for manipulation of data elements in c

1. enqueue

2. dequeue.

3. peek

4. isEmpty

5. isFull

6. size

7. clear

8. toArray

9. front

10. rear

5. Reverse the given string using stack

```
#include <stdio.h>
#include <string.h>
int top,stack[max];
void push(char x){
      if(top == max-1){
          printf("stack overflow");
      }  else {
          stack[++top]=x;
      }
}

void pop(){
      printf("%c",stack[top--]);
}
main()
{
   char str[]="sri lanka";
   int len = strlen(str);
   int i;
   for(i=0;i<len;i++)
```

```
        push(str[i]);
    for(i=0;i<len;i++)
      pop();
}
```

6. Insert value in sorted way in a sorted doubly linked list. Given a sorted doubly linked list and a value to insert, write a function to insert the value in sorted way.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};
void sortedInsert(struct Node** headRef, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;

    if (*headRef == NULL || (*headRef)->data >= data) {
        newNode->next = *headRef;
        if (*headRef != NULL)
            (*headRef)->prev = newNode;
        *headRef = newNode;
        return;
    }

    struct Node* current = *headRef;
    while (current->next != NULL && current->next->data < data) {
        current = current->next;
    }

    newNode->next = current->next;
    if (current->next != NULL)
        current->next->prev = newNode;
    current->next = newNode;
    newNode->prev = current;
}
void printList(struct Node* head) {
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}
int main() {
    struct Node* head = NULL;
    // Create a sorted doubly linked list
    sortedInsert(&head, 5);
    sortedInsert(&head, 10);
    sortedInsert(&head, 15);
    sortedInsert(&head, 20);
    printf("Original sorted list: ");
    printList(head);
    int valueToInsert = 12;
    printf("Inserting %d in a sorted way...\n", valueToInsert);
    sortedInsert(&head, valueToInsert);
    printf("Updated sorted list: ");
    printList(head);
    return 0;
}
```

7. Write a C program to insert/delete and count the number of elements in a queue.

```
#include <stdio.h>
#define MAX_SIZE 100
int queue[MAX_SIZE];
int front = -1;
int back = -1;
void enqueue(int item) {
    if (back == MAX_SIZE - 1) {
        printf("Error: Queue is full\n");
        return;
    }
    if (front == -1) {
        front = 0;
    }
    back++;
    queue[back] = item;
}
void display() {
    if (front == -1 || front > back) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements are: ");
    for (int i = front; i <= back; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}
void dequeue() {
    if (front == -1 || front > back) {
        printf("Error: Queue is empty\n");
        return;
    }
    front++;
}
int is_empty() {
    if (front == -1 || front > back) {
        return 1;
    }
    return 0;
}
int count() {
    int count = 0;
    if (front != -1 && back != -1) {
        for (int i = front; i <= back; i++) {
            count++;
        }
    }
    return count;
}

int main() {
    printf("Initialize a queue!");
    printf("\nCheck the queue is empty or not? %s\n", is_empty() ? "Yes" : "No");
    printf("Number of elements in queue: %d\n", count());
    printf("\nInsert some elements into the queue:\n");
    enqueue(1);
    enqueue(2);
    enqueue(3);
    display();
    printf("Number of elements in queue: %d\n", count());
    printf("\nDelete two elements from the said queue:\n");
    dequeue();
    dequeue();
    display();
    printf("Number of elements in queue: %d\n", count());
    printf("\nInsert another element into the queue:\n");
    enqueue(4);
    display();
    printf("Number of elements in the queue: %d\n", count());
    return 0;
}
```

8. Write a C program to Find whether an array is a subset of another array.

```
#include <stdio.h>
int isSubset(int arr1[], int arr2[], int m, int n)
{
    int i = 0;
    int j = 0;
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            if (arr2[i] == arr1[j])
                break;
        }

        if (j == m)
           return 0;
    }

    return 1;
}

int main()
{
    int arr1[] = { 11, 10, 13, 21, 30, 70 };
    int arr2[] = { 11, 30, 70, 10 };

    int m = sizeof(arr1) / sizeof(arr1[0]);
    int n = sizeof(arr2) / sizeof(arr2[0]);

    if (isSubset(arr1, arr2, m, n))
        printf("arr2[] is subset of arr1[] ");
    else
        printf("arr2[] is not a subset of arr1[]");

    return 0;
}
```