R VEDHA VISALAKSHI

Module 2 Assessment

1. Write a C program to remove duplicate elements from the 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;
      }
      else
      {
         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, 4);
    push(&head, 3);
    push(&head, 3);
    push(&head, 2);
    printf("ORIGINAL LINKED LIST \n");
    printList(head);
    printf("\n");
    printf("LINKED LIST AFTER REMOVING DUPLICATES \n");
    removeDuplicates(head);
    printList(head);
    return 0;
}
```

<u>OUTPUT</u>

```
Output                                                    Clear
/tmp/ZrBGoCD1Dj.o
ORIGINAL LINKED LIST
2-> 3-> 3-> 4->
LINKED LIST AFTER REMOVING DUPLICATES
2-> 3-> 4->
```

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

```
#include<stdio.h>
#include<stdlib.h>

struct Node {
    char data;
    struct Node* prev;
    struct Node* next;
};

void rotate(struct Node** head_ref, int N) {
    if (N == 0)
        return;
```

```
    struct Node* current = *head_ref;
    int count = 1;
    while (count < N && current != NULL) {
        current = current->next;
        count++;
    }
    if (current == NULL)
        return;
    struct Node* NthNode = current;
    while (current->next != NULL)
        current = current->next;
    current->next = *head_ref;
    (*head_ref)->prev = current;
    *head_ref = NthNode->next;
    (*head_ref)->prev = NULL;
    NthNode->next = NULL;
}

void push(struct Node** head_ref, char new_c) {
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
    new_node->data = new_c;
    new_node->prev = NULL;
    new_node->next = (*head_ref);
    if ((*head_ref) != NULL)
        (*head_ref)->prev = new_node;
    *head_ref = new_node;
}

void printList(struct Node* node) {
    while (node->next != NULL) {
        printf("%c ", node->data);
        node = node->next;
    }
    printf("%c\n", node->data);
}

int main(void) {
    struct Node* head = NULL;
    push(&head, 'e');
    push(&head, 'd');
    push(&head, 'c');
    push(&head, 'b');
    push(&head, 'a');

    int N = 2;

    printf("Input: (When N=%d)\n", N);
    printList(head);
```

```
    rotate(&head, N);
    printf("Output:\n");
    printList(head);
    N = 4;
    printf("Input: (When N=%d)\n", N);
    printList(head);
    rotate(&head, N);
    printf("Output:\n");
    printList(head);
    return 0;
}
```

OUTPUT

```
Output                                                    Clear
/tmp/Z6xbx4cx87.o
Input: (When N=2)
a b c d e
Output:
c d e a b
Input: (When N=4)
c d e a b
Output:
b c d e a
```

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

```
#include <stdio.h>
#define MAX_SIZE 100
int queue[MAX_SIZE];
int front = -1, 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;
}

int dequeue() {
    if (front == -1 || front > back) {
```

```
        printf("Error: Queue is empty\n");
        return -1;
    }
    int item = queue[front];
    front++;
    return item;
}

void display() {
    if (front == -1) {
        printf("Error: Queue is empty\n");
        return;
    }
    for (int i = front; i <= back; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}

void sort_queue_asc() {
    int i, j, temp;
    int n = back - front + 1;

    for (i = 0; i < n - 1; i++) {
        for (j = i + 1; j < n; j++) {
            if (queue[i] > queue[j]) {
                temp = queue[i];
                queue[i] = queue[j];
                queue[j] = temp;
            }
        }
    }
}

int main() {
    enqueue(4);
    enqueue(2);
    enqueue(7);
    enqueue(5);
    enqueue(1);
    printf("INPUT\n");
    display();
    printf("OUTPUT\n");
    sort_queue_asc();

    display();

    return 0;
```

```
}
```

OUTPUT

```
Output                                                    Clear
/tmp/aehMhQbE1F.o
INPUT
4 2 7 5 1
OUTPUT
1 2 4 5 7
```

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

The following operations that are available in the queue are

- Enqueue: Add an element to the end of the queue
- Dequeue: Remove an element from the front of the queue
- IsEmpty: Check if the queue is empty
- IsFull: Check if the queue is full
- Peek: Get the value of the front of the queue without removing it

5. Reverse the given string using stack

```
#include <stdio.h>
#include <string.h>

#define max 100
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--]);
}
```

```
void main()
{
  printf("Input: (string)\n");
  printf("LetsLearn\n");
  char str[]="LetsLearn";
  printf("Output: (string)\n");
  int len = strlen(str);
  int i;

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

  for(i=0;i<len;i++)
     pop();
}
```

OUTPUT

Output Clear

```
/tmp/ZrBGoCD1Dj.o
Input: (string)
LetsLearn
Output: (string)
nraeLsteL
```

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 printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d  ", temp->data);
        temp = temp->next;
    }
    printf("\n");
```

```
}

void insertAfter(struct Node* prevNode, int data) {
    if (prevNode == NULL) {
        printf("Previous node cannot be NULL.\n");
        return;
    }

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = prevNode;
    newNode->next = prevNode->next;
    if (prevNode->next != NULL) {
        prevNode->next->prev = newNode;
    }
    prevNode->next = newNode;
}

int main() {
    struct Node* head = NULL;
    struct Node* node3 = (struct Node*)malloc(sizeof(struct Node));
    struct Node* node5 = (struct Node*)malloc(sizeof(struct Node));
    struct Node* node8 = (struct Node*)malloc(sizeof(struct Node));
    struct Node* node10 = (struct Node*)malloc(sizeof(struct Node));
    struct Node* node12 = (struct Node*)malloc(sizeof(struct Node));
    node3->data = 3;
    node3->prev = NULL;
    node3->next = node5;
    node5->data = 5;
    node5->prev = node3;
    node5->next = node8;
    node8->data = 8;
    node8->prev = node5;
    node8->next = node10;
    node10->data = 10;
    node10->prev = node8;
    node10->next = node12;
    node12->data = 12;
    node12->prev = node10;
    node12->next = NULL;
    head = node3;
    printf("Initial Doubly Linked List\n");
    printList(head);
    insertAfter(node8, 9);
    printf("Doubly Linked List after insertion of 9\n");
    printList(head);
    free(node3);
    free(node5);
```

```
    free(node8);
    free(node10);
    free(node12);
    return 0;
}
```

```
Output                                   Clear
/tmp/ZrBGoCD1Dj.o
Initial Doubly Linked List
3  5  8  10  12
Doubly Linked List after insertion of 9
3  5  8  9  10  12
```

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;
}
```

OUTPUT

Output Clear

```
/tmp/Ovmy466fZm.o
Initialize a queue!
Check the queue is empty or not? Yes
Number of elements in queue: 0

Insert some elements into the queue:
Queue elements are: 1 2 3 Number of elements in queue: 3

Delete two elements from the said queue:
Queue elements are: 3 Number of elements in queue: 1

Insert another element into the queue:
Queue elements are: 3 4 Number of elements in the queue: 2
```

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, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};
    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;
}
```

OUTPUT

```
Output                                                Clear
/tmp/kZtbfVRJDN.o
arr2[] is subset of arr1[]
```