Lab Assignment-10

ROLL: 2005535 | NAME: SAHIL SINGH | DATE: 05/10/21

QUES 1: [1] Write a menu driven program to perform the following operations in a CIRCULAR QUEUE ADT (Using an Array) by using suitable user defined functions for each case.

1. Inserting an element into the queue [Define Isfull() function to check overflow]
2. Deleting an element from the queue [Define Isempty() function to check underflow]
3. Display the elements of queue.

SOLUTION:

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

#define DEFNULL -1
#define MAXSIZE 100

typedef struct Queue
{
    int front;
    int rear;
    int data[MAXSIZE];
} Queue;

void enqueue(Queue *, int);
int dequeue(Queue *);
int isFull(Queue *);
int isEmpty(Queue *);
void show_queue(Queue *);

int main()
{
    Queue q1 = {-1, -1};
    int choice;
    do
    {
        printf("1. Insertion\n2. Display\n3. Deletion\n4. Exit\n->: ");
        scanf("%d", &choice);
        int val;
        printf("\n");
        switch (choice)
        {
        case 1:
            printf("Enter value to insert: ");
            scanf("%d", &val);
            enqueue(&q1, val);
            show_queue(&q1);
            break;
```

```
        case 2:
            show_queue(&q1);
            break;
        case 3:
            printf("Deleted element: ");
            printf("%d\n", dequeue(&q1));
            show_queue(&q1);
            break;
        default:
            printf("Exiting...\n");
        }
        printf("-----------------------------------\n");
    } while (choice >= 1 && choice <= 3);
    return 0;
}

void enqueue(Queue *que, int num)
{
    if (isFull(que))
    {
        printf("Overflow!\n");
        return;
    }
    else if (isEmpty(que))
        que->front = que->rear = 0;
    else
        que->rear = (que->rear + 1) % MAXSIZE;
    que->data[que->rear] = num;
}

int dequeue(Queue *que)
{
    int retIndex = que->front;
    if (isEmpty(que))
    {
        printf("Underflow!\n");
        return DEFNULL;
    }
    else if (que->front == que->rear)
    {
        que->front = que->rear = -1;
        return que->data[retIndex];
    }
    que->front = (que->front + 1) % MAXSIZE;
    return que->data[retIndex];
}

int isFull(Queue *que)
{
    if ((que->rear + 1) % MAXSIZE == que->front)
```

```
        return 1;
    return 0;
}

int isEmpty(Queue *que)
{
    if (que->front == -1)
        return 1;
    return 0;
}

void show_queue(Queue *que)
{
    Queue tempQue = {-1, -1};
    while (!isEmpty(que))
    {
        enqueue(&tempQue, que->data[que->front]);
        printf("%d->", dequeue(que));
    }
    while (!isEmpty(&tempQue))
    {
        enqueue(que, dequeue(&tempQue));
    }
    printf("\b\b \n");
}
```

OUTPUT:

```
1. Insertion
2. Display
3. Deletion
4. Exit
->: 1

Enter value to insert: 10
10 >
---------------------------------
1. Insertion
2. Display
3. Deletion
4. Exit
->: 1

Enter value to insert: 20
10->20 >
---------------------------------
1. Insertion
2. Display
3. Deletion
4. Exit
```

```
->: 1

Enter value to insert: 30
10->20->30 >
---------------------------------
1. Insertion
2. Display
3. Deletion
4. Exit
->: 2

10->20->30 >
---------------------------------
1. Insertion
2. Display
3. Deletion
4. Exit
->: 3

Deleted element: 10
20->30 >
---------------------------------
1. Insertion
2. Display
3. Deletion
4. Exit
->: 2

20->30 >
---------------------------------
1. Insertion
2. Display
3. Deletion
4. Exit
->: 4

Exiting...
---------------------------------
```

QUES 2: [2] Write a program to implement QUEUE ADT (FIFO) using STACK ADT (LIFO).

SOLUTION:

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

typedef struct Stack
{
    int data;
    struct Stack *link;
```

```c
} Stack;

void push(Stack **, int);
int pop(Stack **);
int isEmpty(Stack *);
void display(Stack *);
void enqueue(Stack **, int);
int dequeue(Stack **);

int main()
{
    typedef Stack Queue;
    Queue *queue = NULL;
    int choice;
    do
    {
        int val;
        printf("1) Insert in Stack\n2) display\n3) Delete top\n4) Exit\n->: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter value: ");
            scanf("%d", &val);
            enqueue(&queue, val);
            printf("\ntop->");
            display(queue);
            break;
        case 2:
            printf("\ntop->");
            display(queue);
            break;
        case 3:
            printf("\nDeleted element: %d\n", dequeue(&queue));
            break;
        default:
            printf("\nExiting...\n");
        }
        printf("----------------------------\n");
    } while (choice >= 1 && choice <= 3);
    return 0;
}

void push(Stack **Stack_top, int num)
{
    Stack *temp = (Stack *)malloc(sizeof(Stack));
    temp->data = num;
    temp->link = *Stack_top;
    *Stack_top = temp;
}
```

```c
int pop(Stack **Stack_top)
{
    if (isEmpty(*Stack_top))
    {
        printf("\nUnderflow!");
        return -9999999;
    }
    Stack *temp = (*Stack_top);
    *Stack_top = (*Stack_top)->link;
    int val = temp->data;
    free(temp);
    return val;
}

int isEmpty(Stack *Stack_top)
{
    if (!Stack_top)
        return 1;
    return 0;
}

void display(Stack *Stack_top)
{
    if (isEmpty(Stack_top))
    {
        printf("\b\b \n");
        return;
    }
    int temp = pop(&Stack_top);
    printf("%d->", temp);
    display(Stack_top);
    push(&Stack_top, temp);
}

void enqueue(Stack **Stack_top, int num)
{
    if (isEmpty(*Stack_top))
    {
        push(Stack_top, num);
        return;
    }
    int temp = pop(Stack_top);
    enqueue(Stack_top, num);
    push(Stack_top, temp);
}
int dequeue(Stack **Stack_top)
{
    return pop(Stack_top);
}
```

OUTPUT:

```
int dequeue(Stack **Stack_top)
{
    return pop(Stack_top);
}

1) Insert in Stack
2) display
3) Delete top
4) Exit
->: 1
Enter value: 10

top->10 >
----------------------------
1) Insert in Stack
2) display
3) Delete top
4) Exit
->: 1
Enter value: 20

top->10->20 >
----------------------------
1) Insert in Stack
2) display
3) Delete top
4) Exit
->: 1
Enter value: 30

top->10->20->30 >
----------------------------
1) Insert in Stack
2) display
3) Delete top
4) Exit
->: 2

top->10->20->30 >
----------------------------
1) Insert in Stack
2) display
3) Delete top
4) Exit
->: 3

Deleted element: 10
----------------------------
```

```
1) Insert in Stack
2) display
3) Delete top
4) Exit
->: 2

top->20->30 >
----------------------------
1) Insert in Stack
2) display
3) Delete top
4) Exit
->: 4

Exiting...
----------------------------
```

QUES 3: [3] Write a program to implement STACK ADT (LIFO) using QUEUE ADT (FIFO).

SOLUTION:

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

#define DEFNULL -1

typedef struct Node
{
    int data;
    struct Node *link;
} Node;

typedef struct Queue
{
    Node *front;
    Node *rear;
} Queue;

void enqueue(Queue *, int);
int dequeue(Queue *);
int peek(Queue *);
int isEmpty(Queue *);
void display(Queue *);
void push(Queue *, int);
int pop(Queue *);

int main()
{
    typedef Queue Stack;
    Stack stack = {NULL, NULL};
```

```c
    int choice, val;
    do
    {
        printf("1) Insert\n2) Delete\n3) Display\n4) Exit\n->: ");
        scanf("%d", &choice);
        printf("\n");
        switch (choice)
        {
        case 1:
            printf("Enter value to insert: ");
            scanf("%d", &val);
            push(&stack, val);
            display(&stack);
            break;
        case 2:
            printf("Deleted element: ");
            printf("%d\n", pop(&stack));
            display(&stack);
            break;
        case 3:
            display(&stack);
            break;
        default:
            printf("Exiting...\n");
        }
        printf("----------------------------\n");
    } while (choice >= 1 && choice <= 3);
    return 0;
}

void enqueue(Queue *que, int num)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = num;
    temp->link = NULL;
    if (isEmpty(que))
    {
        que->front = que->rear = temp;
        return;
    }
    que->rear->link = temp;
    que->rear = que->rear->link;
}

int dequeue(Queue *que)
{
    if (isEmpty(que))
        return DEFNULL;
    Node *temp = que->front;
    que->front = que->front->link;
```

```
    if (que->front == NULL)
        que->rear = NULL;
    int n = temp->data;
    free(temp);
    return n;
}

int peek(Queue *que)
{
    if (isEmpty(que))
        return DEFNULL;
    return que->front->data;
}

int isEmpty(Queue *que)
{
    if (que->front == NULL)
        return 1;
    return 0;
}

void display(Queue *que)
{
    Queue temp = {NULL, NULL};
    while (!isEmpty(que))
    {
        printf("%d->", peek(que));
        enqueue(&temp, dequeue(que));
    }
    printf("\b\b \n");
    que->front = temp.front;
    que->rear = temp.rear;
}

void push(Queue *que, int num)
{
    Queue temp = {NULL, NULL};
    while (!isEmpty(que))
        enqueue(&temp, dequeue(que));
    enqueue(que, num);
    while (!isEmpty(&temp))
        enqueue(que, dequeue(&temp));
}

int pop(Queue *que)
{
    return dequeue(que);
}
```

OUTPUT:

```
1) Insert
2) Delete
3) Display
4) Exit
->: 1

Enter value to insert: 10
10 >
----------------------------
1) Insert
2) Delete
3) Display
4) Exit->: 1

Enter value to insert: 20
20->10 >
----------------------------
1) Insert
2) Delete
3) Display
4) Exit
->: 1

Enter value to insert: 30
30->20->10 >
----------------------------
1) Insert
2) Delete
3) Display
4) Exit
->: 3

30->20->10 >
----------------------------
1) Insert
2) Delete
3) Display
4) Exit
->: 2

Deleted element: 30
20->10 >
----------------------------
1) Insert
2) Delete
3) Display
4) Exit
->: 3

20->10 >
```

```
----------------------------
1) Insert
2) Delete
3) Display
4) Exit
->: 4

Exiting...
----------------------------
```

QUES 4: [4] Write a program to implement a priority queue using an array.SOLUTION:

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

#define DEFNULL -1
#define MAXSIZE 10

static int maxpriority = -1;

typedef struct Queue
{
    int front[MAXSIZE];
    int rear[MAXSIZE];
    int data[MAXSIZE][MAXSIZE];
} Queue;

Queue *create();
void enqueue(Queue **, int, int);
int dequeue(Queue **);
int peek(Queue *);
int isFull(Queue *, int);
int isEmpty(Queue *, int);
void show_queue(Queue *);

int main()
{
    Queue *q1 = create();
    int choice;
    do
    {
        printf("1) Insertion\n2) Display\n3) Deletion\n4) Exit\n->: ");
        scanf("%d", &choice);
        int val;
        printf("\n");
        switch (choice)
        {
        case 1:
            printf("Enter value to insert: ");
            scanf("%d", &val);
```

```
            printf("Enter priority: ");
            int pri;
            scanf("%d", &pri);
            enqueue(&q1, val, pri);
            show_queue(q1);
            break;
        case 2:
            show_queue(q1);
            break;
        case 3:
            printf("Deleted element: ");
            printf("%d\n", dequeue(&q1));
            show_queue(q1);
            break;
        default:
            printf("Exiting...\n");
        }
        printf("-----------------------------------\n");
    } while (choice >= 1 && choice <= 3);
    return 0;
}

Queue *create()
{
    Queue *queue = (Queue *)malloc(sizeof(Queue));
    for (int i = 0; i < MAXSIZE; i++)
    {
        queue->front[i] = queue->rear[i] = -1;
    }
    return queue;
}

void enqueue(Queue **que, int num, int priority)
{
    if (priority > MAXSIZE || priority < 0)
    {
        printf("Invalid Priority\n");
        return;
    }
    if (isFull(*que, priority))
    {
        printf("Overflow!\n");
        return;
    }
    else if (isEmpty(*que, priority))
        (*que)->front[priority] = (*que)->rear[priority] = 0;
    else
        (*que)->rear[priority] =
            ((*que)->rear[priority] + 1) % MAXSIZE;
    if (priority > maxpriority)
```

```
        maxpriority = priority;
    (*que)->data[priority][(*que)->rear[priority]] = num;
}

int dequeue(Queue **que)
{
    for (int i = maxpriority; i >= 0; i--)
    {
        int retIndex = (*que)->front[i];
        if (!isEmpty(*que, i) && (*que)->front[i] ==
                                            (*que)->rear[i])
        {
            (*que)->front[i] = (*que)->rear[i] = -1;
            return (*que)->data[i][retIndex];
        }
        else if (!isEmpty(*que, i))
        {
            (*que)->front[i] =
                ((*que)->front[i] + 1) % MAXSIZE;
            return (*que)->data[i][retIndex];
        }
    }
    return DEFNULL;
}

int peek(Queue *que)
{
    for (int i = maxpriority; i >= 0; i++)
    {
        if (!isEmpty(que, i))
            return que->data[i][que->front[i]];
    }
    return DEFNULL;
}

int isFull(Queue *que, int priority)
{
    if ((que->rear[priority] + 1) % MAXSIZE ==
        que->front[priority])
        return 1;
    return 0;
}

int isEmpty(Queue *que, int priority)
{
    if (que->front[priority] == -1)
        return 1;
    return 0;
}
```

```c
void show_queue(Queue *que)
{
    for (int i = maxpriority; i >= 0; i--)
    {
        int start = que->front[i];
        while (start != que->rear[i])
        {
            printf("%d->", que->data[i][start]);
            start = (start + 1) % MAXSIZE;
        }
        if (!isEmpty(que, i))
            printf("%d->", que->data[i][start]);
    }
    printf("\b\b \n");
}
```

OUTPUT:

```
1) Insertion
2) Display
3) Deletion
4) Exit
->: 1

Enter value to insert: 10
Enter priority: 2
10 >
---------------------------------
1) Insertion
2) Display
3) Deletion
4) Exit
->: 1

Enter value to insert: 20
Enter priority: 3
20->10 >
---------------------------------
1) Insertion
2) Display
3) Deletion
4) Exit
->: 1

Enter value to insert: 5
Enter priority: 1
20->10->5 >
---------------------------------
1) Insertion
2) Display
```

```
3) Deletion
4) Exit
->: 1

Enter value to insert: 89
Enter priority: 8
89->20->10->5 >
---------------------------------
1) Insertion
2) Display
3) Deletion
4) Exit
->: 2

89->20->10->5 >
---------------------------------
1) Insertion
2) Display
3) Deletion
4) Exit
->: 3

Deleted element: 89
20->10->5 >
---------------------------------
1) Insertion
2) Display
3) Deletion
4) Exit
->: 2

20->10->5 >
---------------------------------
1) Insertion
2) Display
3) Deletion
4) Exit
->: 4

Exiting...
---------------------------------
```

QUES 5: [5] Write a program to implement a priority queue using linked list.

SOLUTION:

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

#define DEFNULL -1
```

```c
typedef struct Node
{
    int data;
    int priority;
    struct Node *link;
} Node;

typedef struct Queue
{
    Node *front;
    Node *rear;
} Queue;

void enqueue(Queue *, int, int);
int dequeue(Queue *);
int peek(Queue *);
int isEmpty(Queue *);
void show_queue(Queue *);

int main()
{
    Queue que = {NULL, NULL};
    int choice;
    do
    {
        printf("1) Insertion\n2) Display\n3) Deletion\n4) Exit\n->: ");
        scanf("%d", &choice);
        int val;
        printf("\n");
        switch (choice)
        {
        case 1:
            printf("Enter value to insert: ");
            scanf("%d", &val);
            printf("Enter priority: ");
            int pri;
            scanf("%d", &pri);
            enqueue(&que, val, pri);
            show_queue(&que);
            break;
        case 2:
            show_queue(&que);
            break;
        case 3:
            printf("Deleted element: ");
            printf("%d\n", dequeue(&que));
            show_queue(&que);
            break;
        default:
```

```
            printf("Exiting...\n");
        }
        printf("----------------------------------\n");
    } while (choice >= 1 && choice <= 3);
    return 0;
}

void enqueue(Queue *que, int num, int priority)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = num;
    temp->priority = priority;
    temp->link = NULL;
    if (isEmpty(que))
    {
        que->front = que->rear = temp;
        return;
    }
    Node *tempPrev = NULL;
    Node *tempFront = que->front;
    while (tempFront && priority <= tempFront->priority)
    {
        tempPrev = tempFront;
        tempFront = tempFront->link;
    }
    if (!tempPrev)
    {
        temp->link = que->front;
        que->front = temp;
        return;
    }
    temp->link = tempFront;
    tempPrev->link = temp;
}

int dequeue(Queue *que)
{
    if (isEmpty(que))
        return DEFNULL;
    Node *temp = que->front;
    que->front = que->front->link;
    if (que->front == NULL)
        que->rear = NULL;
    int n = temp->data;
    free(temp);
    return n;
}

int peek(Queue *que)
{
```

```
    if (isEmpty(que))
        return DEFNULL;
    return que->front->data;
}
int isEmpty(Queue *que)
{
    if (que->front == NULL)
        return 1;
    return 0;
}
void show_queue(Queue *que)
{
    Queue temp = {NULL, NULL};
    while (!isEmpty(que))
    {
        printf("%d->", peek(que));
        int pri = que->front->priority;
        enqueue(&temp, dequeue(que), pri);
    }
    printf("\b\b \n");
    que->front = temp.front;
    que->rear = temp.rear;
}
```

OUTPUT:

```
1) Insertion
2) Display
3) Deletion
4) Exit
->: 1

Enter value to insert: 10
Enter priority: 4
10 >
---------------------------------
1) Insertion
2) Display
3) Deletion
4) Exit
->: 1

Enter value to insert: 20
Enter priority: 3
10->20 >
---------------------------------
1) Insertion
2) Display
3) Deletion
4) Exit
```

```
->: 1

Enter value to insert: 50
Enter priority: 1
10->20->50 >
---------------------------------
1) Insertion
2) Display
3) Deletion
4) Exit
->: 1

Enter value to insert: 100
Enter priority: 8
100->10->20->50 >
---------------------------------
1) Insertion
2) Display
3) Deletion
4) Exit
->: 2

100->10->20->50 >
---------------------------------
1) Insertion
2) Display
3) Deletion
4) Exit
->: 3

Deleted element: 100
10->20->50 >
---------------------------------
1) Insertion
2) Display
3) Deletion
4) Exit
->: 2

10->20->50 >
---------------------------------
1) Insertion
2) Display
3) Deletion
4) Exit
->: 4

Exiting...
---------------------------------
```