In-Lab – 1(page No: 162)

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *front = NULL;
struct node *rear = NULL;

void enQueue()
{
   struct node *newnode = (struct node*)malloc(sizeof(struct node));
   if (newnode == NULL)
   {
      return;
   }
   scanf("%d", &newnode->data);
   newnode->next = NULL;

   if (rear == NULL)
   {
      front = rear = newnode;
   }
   else
   {
      rear->next = newnode;
      rear = newnode;
   }
}
void deQueue()
{
   if (front == NULL)
   {
      printf("Queue is empty\n");
      return;
   }
   struct node *temp = front;
   front = front->next;
   if (front == NULL)
   {
      rear = NULL;
```

```
    }
    free(temp);
}
void display()
{
    if (front == NULL)
    {
        printf("NULL\n");
        return;
    }
    struct node *temp = front;
    while (temp != NULL)
    {
        printf("->%d", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main()
{
    int choice;
    while (1)
    {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                enQueue();
                break;
            case 2:
                deQueue();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
        }
    }
    return 0;
}
```

In-Lab – 2(page No: 164)

```
#include <stdio.h>
#define size 5
int queue[size];
int front = -1, rear = -1;

void enQueue()
{
   int value;
   scanf("%d", &value);
   if ((front == 0 && rear == size-1) || (rear==front-1))
   {
      printf("Queue Overflow\n");
      return;
   }
   else if( front == -1 && rear == -1)
   {
      front=rear=0;
   }
   else if(rear == size-1 && front != 0)
   {
      rear=0;
   }
   else
   {
      rear++;
   }
   queue[rear]=value;
}
void deQueue()
{
   if (front == -1)
   {
      printf("Queue Underflow\n");
      return;
   }
   else if(front == rear)
   {
      front=-1;
      rear=-1;
   }
   else if(front == size-1 )
   {
        front=0;
    }
    else
    {
        front++;
    }
}
void display() {
    if (front == -1)
    {
        printf("NULL\n");
        return;
    }
    int i;
    if(front <= rear)
    {
        for(i=front;i<=rear;i++)
        {
            printf("%d ",queue[i]);
        }
    }
    else
    {
        for(i=front;i<=size-1;i++)
        {
            printf("%d ",queue[i]);
        }
        for(i=0;i<=rear;i++)
        {
            printf("%d ",queue[i]);
        }
    }
    printf("\n");
}
int main()
{
    int choice;
    while (1)
    {
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                enQueue();
                break;
            case 2:
                deQueue();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
        }
    }
    return 0;
}
```

In-Lab – 3(page No: 166)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100000

int stack1[MAX], stack2[MAX];
int top1 = -1, top2 = -1;

void push(int value)
{
    stack1[++top1] = value;
}

int pop()
{
    if (top2 == -1)
    {
        while (top1 >= 0)
        {
            stack2[++top2] = stack1[top1--];
        }
    }
    if (top2 >= 0)
    {
        return stack2[top2--];
    }
    return -1;
}

int peek()
```

```
{
   if (top2 == -1)
   {
      while (top1 >= 0)
      {
         stack2[++top2] = stack1[top1--];
      }
   }
   if (top2 >= 0)
   {
      return stack2[top2];
   }
   return -1;
}

int main()
{
   int q, type, value;
   scanf("%d", &q);
   while (q--)
   {
      scanf("%d", &type);

      if (type == 1)
      {
         scanf("%d", &value);
         push(value);
      }
      else if (type == 2)
      {
         pop();
      }
      else if (type == 3)
      {
         printf("%d\n", peek());
      }
   }
   return 0;
}
```

Post-Lab – 1(page No: 168) (execution only DEVC++)

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

#define MAX 100

typedef struct
{
   char name[50];
   int ticketNum;
} Customer;

typedef struct
{
   Customer queue[MAX];
   int front, rear;
} Queue;

void initQueue(Queue *q)
{
   q->front = q->rear = -1;
}

int isEmpty(Queue *q)
{
   return (q->front == -1);
}

int isFull(Queue *q)
{
   return (q->rear == MAX - 1);
}

void enqueue(Queue *q, char name[], int ticketNum)
{
   if (isFull(q)) return;
   if (isEmpty(q)) q->front = 0;
   q->rear++;
   strcpy(q->queue[q->rear].name, name);
   q->queue[q->rear].ticketNum = ticketNum;
}
```

```
int searchQueue(Queue *q, char name[], int ticketNum)
{
            int i;
    if (isEmpty(q))
                        return 0;
    for (i = q->front; i <= q->rear; i++)
            {
        if (strcmp(q->queue[i].name, name) == 0 && q->queue[i].ticketNum == ticketNum) {
            return 1;
        }
    }
    return 0;
}

int main()
{
    Queue q;
    initQueue(&q);

    int n;
    printf("Enter number of customers registering: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
            {
        char name[50];
        int ticketNum;
        printf("Enter Name: ");
        scanf("%s", name);
        printf("Enter Ticket Number: ");
        scanf("%d", &ticketNum);
        enqueue(&q, name, ticketNum);
    }

    printf("\nData has been uploaded.\n\n");

    int m;
    printf("Enter number of customers checking entry: ");
    scanf("%d", &m);
    for (int i = 0; i < m; i++)
            {
        char checkName[50];
        int checkTicket;
        printf("\nEnter Name: ");
        scanf("%s", checkName);
        printf("Enter Ticket Number: ");
        scanf("%d", &checkTicket);

        if (searchQueue(&q, checkName, checkTicket)) {
            printf("Data found, Enjoy the movie!\n");
        }
            else {
            printf("Ticket ID not found, re-register.\n");
        }
    }
    return 0;
}
```

Post-Lab – 2(page No:170)

```
typedef struct
{
    int data[100];
    int top;
} Stack;
void initStack(Stack *s)
{
    s->top = -1;
}
bool isEmpty(Stack *s)
{
    return s->top == -1;
}
void pushStack(Stack *s, int x)
{
    s->data[++(s->top)] = x;
}
int popStack(Stack *s)
{
    return s->data[(s->top)--];
}
int peekStack(Stack *s)
{
    return s->data[s->top];
}
typedef struct
{
    Stack stack1;
    Stack stack2;
} MyQueue;

MyQueue* myQueueCreate()
{
    MyQueue *queue = (MyQueue*)malloc(sizeof(MyQueue));
    initStack(&(queue->stack1));
    initStack(&(queue->stack2));
    return queue;
}
void myQueuePush(MyQueue* q, int x)
{
    pushStack(&(q->stack1), x);
}
int myQueuePop(MyQueue* q)
{
    if (isEmpty(&(q->stack2)))
    {
        while (!isEmpty(&(q->stack1)))
        {
            pushStack(&(q->stack2), popStack(&(q->stack1)));
        }
    }
    return popStack(&(q->stack2));
}
int myQueuePeek(MyQueue* q)
{
    if (isEmpty(&(q->stack2)))
    {
        while (!isEmpty(&(q->stack1)))
        {
            pushStack(&(q->stack2), popStack(&(q->stack1)));
        }
    }
    return peekStack(&(q->stack2));
}
bool myQueueEmpty(MyQueue* q)
{
    return isEmpty(&(q->stack1)) && isEmpty(&(q->stack2));
}
void myQueueFree(MyQueue* q)
{
    free(q);
}
```

Skill Lab-1(page No:172)

```
#include <stdio.h>

void insertMinHeap(long long heap[], int *size, long long val) {
    if (*size < 3) {
        heap[(*size)++] = val;
    }
    else if (val > heap[0]) {
        heap[0] = val;
    }
    for (int i = 0; i < *size - 1; i++)
    {
        for (int j = 0; j < *size - i - 1; j++)
        {
            if (heap[j] > heap[j + 1])
            {
                long long temp = heap[j];
                heap[j] = heap[j + 1];
                heap[j + 1] = temp;
            }
        }
    }
}

void findTopThreeProduct(long long arr[], long long n)
{
    long long heap[3];
    int size = 0;

    for (long long i = 0; i < n; i++)
    {
        insertMinHeap(heap, &size, arr[i]);

        if (size < 3) {
            printf("-1\n");
        }
        else {
            printf("%lld\n", heap[0] * heap[1] * heap[2]);
        }
    }
}
```

```
int main()
{
   long long n;
   scanf("%lld", &n);

   long long arr[n];
   for (long long i = 0; i < n; i++)
   {
      scanf("%lld", &arr[i]);
   }

   findTopThreeProduct(arr, n);
   return 0;
}
```

Skill Lab-2(page No:174)

```
typedef struct
{
   int *heap;    // Min-heap to store k largest elements
   int size;     // Current number of elements in heap
   int capacity; // Maximum size (k)
} KthLargest;
int kthLargestAdd(KthLargest* kth, int val);
KthLargest* kthLargestCreate(int k, int* nums, int numsSize)
{
   KthLargest *kth = (KthLargest*)malloc(sizeof(KthLargest));
   kth->heap = (int*)malloc(sizeof(int) * k);
   kth->size = 0;
   kth->capacity = k;
   for (int i = 0; i < numsSize; i++)
   {
      kthLargestAdd(kth, nums[i]);
   }
   return kth;
}
int kthLargestAdd(KthLargest* kth, int val)
{
   if (kth->size < kth->capacity)
   {
      int i = kth->size;
      kth->heap[kth->size++] = val;
      while (i > 0)
```

```
    {
        int parent = (i - 1) / 2;
        if (kth->heap[i] < kth->heap[parent])
        {
            int temp = kth->heap[i];
            kth->heap[i] = kth->heap[parent];
            kth->heap[parent] = temp;
            i = parent;
        }
        else {
            break;
        }
    }
}
else if (val > kth->heap[0])
{
    kth->heap[0] = val;
    int i = 0;
    while (1)
    {
        int left = 2 * i + 1;
        int right = 2 * i + 2;
        int smallest = i;

        if (left < kth->size && kth->heap[left] < kth->heap[smallest])
        {
            smallest = left;
        }
        if (right < kth->size && kth->heap[right] < kth->heap[smallest])
        {
            smallest = right;
        }
        if (smallest != i)
        {
            int temp = kth->heap[i];
            kth->heap[i] = kth->heap[smallest];
            kth->heap[smallest] = temp;
            i = smallest;
        }
        else {
            break;
        }
    }
}
```

```
   return kth->heap[0];
}
void kthLargestFree(KthLargest* kth)
{
   free(kth->heap);
   free(kth);
}
```

Skill Lab-3(page No:176)

```
#include <stdio.h>
#include <stdlib.h>
int* next_higher_peak(int* heights, int n)
{
   int* result = (int*)malloc(n * sizeof(int));
   int* stack = (int*)malloc(n * sizeof(int));
   int top = -1,i,index;
   for (i = 0; i < n; i++)
   {
      result[i] = -1;
   }
   for (i = 0; i < n; i++)
   {
      while (top != -1 && heights[i] > heights[stack[top]])
      {
         index = stack[top];
         result[index] = heights[i];
         top--;
      }
      top++;
      stack[top] = i;
   }
   free(stack);
   return result;
}
int main()
{
   int n;
   scanf("%d", &n);
   int* heights = (int*)malloc(n * sizeof(int));
   for (int i = 0; i < n; ++i)
   {
      scanf("%d", &heights[i]);
```

```
    }
    int* result = next_higher_peak(heights, n);
    for (int i = 0; i < n; ++i)
    {
        printf("%d ", result[i]);
    }
    free(heights);
    free(result);
    return 0;
}
```

Skill Lab-4(page No:178)

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int t;
    scanf("%d", &t);
    while (t--)
    {
        int n, k;
        scanf("%d %d", &n, &k);

        int* q = (int*)malloc(n * sizeof(int));

        for (int i = 0; i < n; i++)
        {
            scanf("%d", &q[i]);
        }
        if (k > n) k = n;
        for (int i = k; i < n; i++)
        {
            printf("%d ", q[i]);
        }
        for (int i = 0; i < k; i++)
        {
            printf("%d ", q[i]);
        }
        printf("\n");
        free(q);
    }
    return 0;
}
```

Skill Lab-5(page No:180)

```
int countStudents(int* students, int studentsSize, int* sandwiches, int sandwichesSize)
{
     int count0 = 0, count1 = 0, i;
    for (i = 0; i < studentsSize; i++)
    {
        if (students[i] == 0)
            count0++;
        else
            count1++;
    }
    for (i = 0; i < sandwichesSize; i++)
    {
        if (sandwiches[i] == 0) {
            if (count0 == 0)
                return count1;
            count0--;
        }
        else {
            if (count1 == 0)
                return count0;
            count1--;
        }
    }
    return 0;
}
```