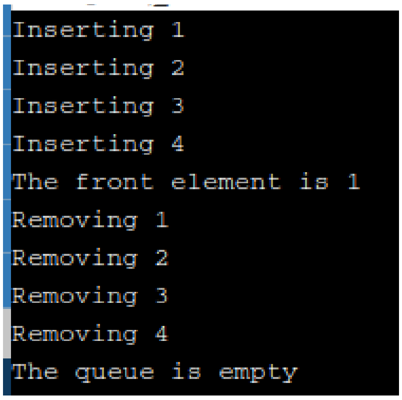
Lab 7

Q1

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

// A Linked List Node
struct Node
{
  int data;                        // integer data
  struct Node *next;               // pointer to the next node
} *rear = NULL, *front = NULL;

// Utility function to allocate the new queue node
struct Node * newNode (int item)
{
  // allocate a new node in a heap
  struct Node *node = (struct Node *) malloc (sizeof (struct Node));

  // check if the queue (heap) is full. Then inserting an element would
  // lead to heap overflow
  if (node != NULL)
    {
      // set data in the allocated node and return it
      node->data = item;
      node->next = NULL;
      return node;
    }
  else
    {
      printf ("\nHeap Overflow");
      exit (EXIT_FAILURE);
    }
}

// Utility function to dequeue the front element
int dequeue ()                     // delete at the beginning
{
  if (front == NULL)
    {
      printf ("\nQueue Underflow");
      exit (EXIT_FAILURE);
    }

  struct Node *temp = front;
  printf ("Removing %d\n", temp->data);

  // advance front to the next node
  front = front->next;

  // if the list becomes empty
  if (front == NULL)
    {
      rear = NULL;
    }

  // deallocate the memory of the removed node and
  // optionally return the removed item
  int item = temp->data;
  free (temp);
  return item;
}

// Utility function to add an item to the queue
void enqueue (int item)                  // insertion at the end
{
  // allocate a new node in a heap
  struct Node *node = newNode (item);
  printf ("Inserting %d\n", item);

  // special case: queue was empty
  if (front == NULL)
    {
      // initialize both front and rear
      front = node;
      rear = node;
    }
  else
    {
      // update rear
      rear->next = node;
      rear = node;
    }
}

// Utility function to return the top element in a queue
int peek ()
{
  // check for an empty queue
  if (front != NULL)
    {
      return front->data;
    }
  else
    {
      exit (EXIT_FAILURE);
    }
}

// Utility function to check if the queue is empty or not
int isEmpty ()
{
  return rear == NULL && front == NULL;
```

```
}

int main ()
{
  enqueue (1);
  enqueue (2);
  enqueue (3);
  enqueue (4);

  printf ("The front element is %d\n", peek ());

  dequeue ();
  dequeue ();
  dequeue ();
  dequeue ();

  if (isEmpty ())
    {
      printf ("The queue is empty");
    }
  else
    {
      printf ("The queue is not empty");
    }

  return 0;
}
```



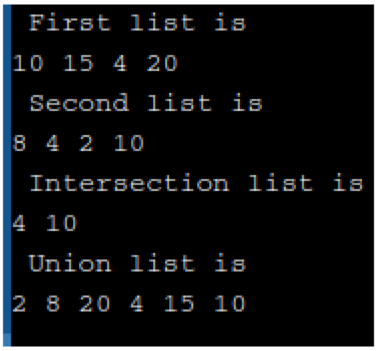
```
Inserting 1
Inserting 2
Inserting 3
Inserting 4
The front element is 1
Removing 1
Removing 2
Removing 3
Removing 4
The queue is empty
```

Q2

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
struct Node
{
  int data;
  struct Node *next;
};
```

```
void push (struct Node **head_ref, int new_data);
bool isPresent (struct Node *head, int data);
struct Node * getUnion (struct Node *head1, struct Node *head2)
{
  struct Node *result = NULL;
  struct Node *t1 = head1, *t2 = head2;
  while (t1 != NULL)
    {
      push (&result, t1->data);
      t1 = t1->next;
    }
  while (t2 != NULL)
    {
      if (!isPresent (result, t2->data))
            push (&result, t2->data);
      t2 = t2->next;
    }
  return result;
}
struct Node * getIntersection (struct Node *head1, struct Node *head2)
{
  struct Node *result = NULL;
  struct Node *t1 = head1;
 while (t1 != NULL)
    {
      if (isPresent (head2, t1->data))
            push (&result, t1->data);
      t1 = t1->next;
    }
  return result;
}
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;
    }
}
bool isPresent (struct Node *head, int data)
{
  struct Node *t = head;
  while (t != NULL)
    {
      if (t->data == data)
```

```
            return 1;
      t = t->next;
    }
  return 0;
}
int main ()
{
  struct Node *head1 = NULL;
  struct Node *head2 = NULL;
  struct Node *intersecn = NULL;
  struct Node *unin = NULL;

  push (&head1, 20);
  push (&head1, 4);
  push (&head1, 15);
  push (&head1, 10);

  push (&head2, 10);
  push (&head2, 2);
  push (&head2, 4);
  push (&head2, 8);

  intersecn = getIntersection (head1, head2);
  unin = getUnion (head1, head2);

  printf ("\n First list is \n");
  printList (head1);

  printf ("\n Second list is \n");
  printList (head2);

  printf ("\n Intersection list is \n");
  printList (intersecn);

  printf ("\n Union list is \n");
  printList (unin);
  return 0;
}
```



```
 First list is
10 15 4 20
 Second list is
8 4 2 10
 Intersection list is
4 10
 Union list is
2 8 20 4 15 10
```