```
public static <T> void removeLast(DoubleLinkedList<T>  l1)
{
     l1.findFirst();

     while(! l1.last())
          l1.findNext();

     l1.remove();
}

public static <T> void removeBeforeCurrent(DoubleLinkedList<T>  l1)
{
     l1.findPrevious();
     l1.remove();
}

public static <T> void removeBeforeCurrentCheck(DoubleLinkedList<T>  l1)
{
     int sizeFromCurrent = 0;

     while(! l1.last())
     {
          sizeFromCurrent++;
          l1.findNext();
     }

     if(sizeFromCurrent == 1)
          return;

     sizeFromCurrent++;

     int fullSize = 0;

     l1.findFirst();

     while(! l1.last())
     {
          fullSize++;
          l1.findNext();
     }

     fullSize++;

     if(sizeFromCurrent != fullSize && ! l1.empty())
     {
          l1.findFirst();

          for(int i = 0 ; i < fullSize - sizeFromCurrent ; i++)
               l1.findNext();

          l1.findPrevious();
          l1.remove();
     }
}
```

```
public static Integer findSmallest(Queue <Integer> q)
{
     Integer min = q.serve();
     q.enqueue(min);
     Integer x;

     for (int i = 0 ; i < q.length() - 1; i++)
     {
          x = q.serve();

          if(x < min)
               min = x;

          q.enqueue(x);
     }

     return min;
}

public static <T> void copy(LinkedQueue<T> q,int i,int j)
{
     T tmp = null,x;

     for(int k = 0 ; k < q.length() ; k++)
     {
          x = q.serve();

          if (k == i)
          {
               tmp = x;
               q.enqueue(x);
          }
          else if(k == j)
          {
               q.enqueue(tmp);
          }
          else
          {
               q.enqueue(x);
          }
     }
}
```

```
public static <T> void exchange(LinkedQueue<T> q,int i,int j)
{
     T tmp1 = null, tmp2 = null, x;

     for(int k = 0 ; k < q.length() ; k++)
     {
          x = q.serve();
          q.enqueue(x);

          if (k == i)
               tmp1 = x;
          else if(k == j)
               tmp2 = x;
     }

     for(int k = 0 ; k < q.length() ; k++)
     {
          x = q.serve();
          if (k == i)
               q.enqueue(tmp2);
          else if(k == j)
               q.enqueue(tmp1);
          else
               q.enqueue(x);
     }
}

public static <T> void split(LinkedQueue<T> q1,LinkedQueue<T> q2
                             ,LinkedQueue<T> q3)
{
     T x;

     for(int i = 0 ; i < q1.length() ; i++)
     {
          x = q1.serve();
          q1.enqueue(x);

          if (i % 2 == 0)
               q2.enqueue(x);
          else
               q3.enqueue(x);
     }
}

public static <T> void insertFirst(LinkedList<T> l1, T val)
{
     if (! l1.empty())
     {
          l1.findFirst();
          T x = l1.retrieve();
          l1.update(val);
          l1.insert(x);
          l1.findFirst();
     }
}
```

```
public static <T> void exchange(LinkedList<T> l,int i,int j)
{
     //Go to ith node
     l.findFirst();

     for (int k = 0 ; k < i ; k++)
          l.findNext();

     //Save ith node
     T temp1 = l.retrieve();

     //Go to jth node
     l.findFirst();

     for (int k = 0 ; k < j ; k++)
          l.findNext();

     //Save jth node
     T temp2 = l.retrieve();

     //Replace jth node by ith node
     l.update(temp1);

     //Go to ith node
     l.findFirst();

     for (int k = 0 ; k < i ; k++)
          l.findNext();

     //Replace ith node by jth node
     l.update(temp2);
}
```

```
    public static <T> void findPrev(LinkedList<T> l1)
    {
         //count how many nodes from current to the end of the list
         int sizeFromCurrent = 0;

         while(! l1.last())
         {
              sizeFromCurrent++;
              l1.findNext();
         }

         sizeFromCurrent++;

         //count how many nodes in the whole list
         int fullSize = 0;

         l1.findFirst();

         while(! l1.last())
         {
              fullSize++;
              l1.findNext();
         }

         fullSize++;

         //move current to previous node by going to first node
         //then moving current by subtracting the fullSize from
sizeFromCurrent

         l1.findFirst();

         for(int i = 0 ; i < fullSize - sizeFromCurrent - 1; i++)
              l1.findNext();
    }
```

```
    public static <T> void reverse(LinkedList<T> l1)
    {
        if (! l1.empty())
        {
            l1.findFirst();
            // first we will count the number of elements
            int count = 1;// for the last element

            while (! l1.last())
            {
                count++;
                l1.findNext();
            }

            if (count > 1)
            {
                T temp1 = null;
                T temp2 = null;

                l1.findFirst();

                for (int i = 0; i < (count / 2); i++)
                {
                    temp1 = l1.retrieve();

                    for (int j = i; j < count - i - 1; j++)
                        l1.findNext();

                    temp2 = l1.retrieve();
                    l1.update(temp1);

                    l1.findFirst();

                    for (int j = 0; j < i; j++)
                        l1.findNext();

                    l1.update(temp2);
                    l1.findNext();
                }
            }
        }
    }
}
```