CS261: Exam 1

1 Problem 1: Remove All Deque – 70 points

Complete the C function for removing the element at the front or back of Deque implemented as a dynamic array. Also, if this element has multiple repetitions at the front (or back), remove all of its multiple repetitions until you encounter a different value which will then become the new front of Deque (or back). Input arguments of the function include the pointer to Deque, and `flag` that indicates removal from the front of Deque if `flag` = 1, or removal from the back of Deque if `flag` = -1. Your implementation should allow that indices of the front and back of Deque may "wrap around" the block of memory occupied by Deque.

2 Problem 2: Initialize Deque – 30 points

Complete the C function for initializing Deque in the program. An input argument of the function is the pointer to Deque.

```
/* input arguments:
  dq -- pointer to deque
  flag -- flag = 1 remove from front; flag = -1 remove from back
*/

void removeAllDeque(struct Deque *dq, int flag)
{
   TYPE val; /*auxiliary variable*/

/*5 points; check input arguments*/
   assert(dq && (flag == 1 || flag == -1) && dq->size > 0);

/*5 points; check whether to remove from front or back*/
   if(flag == 1){  /* remove from front */

/*2 points; memorize the front element*/
      val = dq->data[dq->front];
/*8 points; loop to remove repetitions, must check if dq->size>0*/
      while (EQ(val, dq->data[dq->front]) && dq->size > 0){

/*10 points; compute the new front and wrap around*/
         dq->front++;
         if(dq->front == dq->capacity) dq->front = 0;

 /*5 points; maintain the size*/
         dq->size--;
      }
   }
   else{ /* remove from back */

/*10 points; compute the back index modulo capacity*/
      int backIndex = (dq->front + dq->size - 1) % dq->capacity;

/*2 points; memorize the back element*/
      val = dq->data[backIndex];
/*8 points; loop to remove repetitions, must check if dq->size>0*/
      while (EQ(val, dq->data[backIndex]) && dq->size > 0){

/*10 points; compute the back index and wrap around*/
         backIndex--;
         if(backIndex < 0) backIndex = dq->capacity - 1;

 /*5 points; maintain the size*/
         dq->size--;
      }}}
```

```
/* Initialize Deque */
void initDeque(struct Deque * dq, int cap) {

/*5 points; check input arguments*/
   assert (dq && cap > 0);

/*10 points; initialize capacity, size, front index*/
   dq->capacity = cap;
   dq->size = dq->front = 0;

/*10 points; allocate a block of memory*/
   dq->data = (TYPE *) malloc(dq->capacity * sizeof(TYPE));

/*5 points; check if malloc was successful*/
   assert (dq->data != 0);
}
```