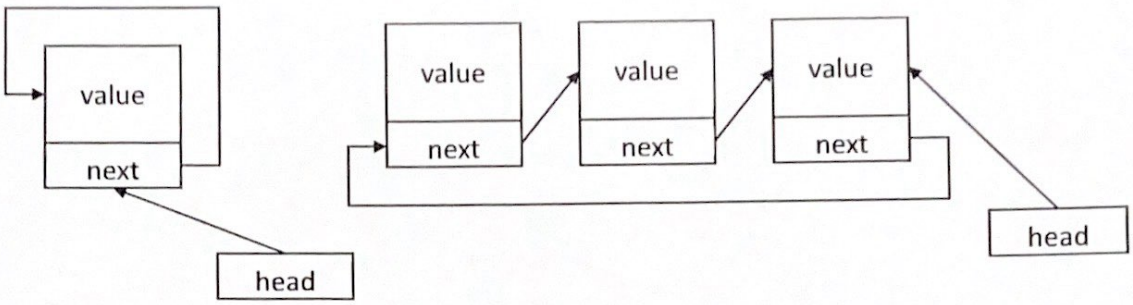
Name and ID#:_____

Problem 2: Consider the concept of a ***Circular Queue*** that implements ***Queue ADT*** as a circular list. It has a pointer ***node *head*** that points to the last node – the back, not the first one – the top. The address of the top, therefore, is kept in the pointer ***node *next*** of the last node – ***head->next***. If ***cqueue*** is empty, ***head = NULL***. If it has just one node, ***head = head->next***. Examples of one-node and three-node ***cqueue*** objects are shown below:



As usually, ***cqueue*** inserts a new value at the back, removes the top value and retrieves the top value. Derive a C++ ***class cqueue*** from ***class base*** that implements the concept of circular queue. Write the header and source files. The header file of ***class base*** and all its functions are implemented:

```
class base
{
public:
      base(); //the default constructor creates an empty base
      base(const base &that); //copy constructor
      ~base(); //destructor
      bool is_empty();
protected:
      struct node
      {
            int value;
            node *next;
            node(int new_value, node *new_next) : value(new_value), next(new_next){} ;
      };
      node *head;

      void insert(int new_value, int at, node* &from);
      bool remove(int at, node* &from);
      int retrieve(int at, node* from);
}
```

```
class cqueue
{ public:
  cqueue();
 ~cqueue();
cqueue(const cqueue &that)
}
  cqueue::cqueue(const cqueue &that)
{ head = that.head;
head -> next = that.head->next;
}
```

```
cqueue::cqueue()
{  if(!is_empty)
   { head = NULL;
   }

cqueue::~cqueue()
} if(!is_empty)
 { remove(0, head)
  head = head -> next;
  {
```

Use the backside, if needed

DS.MT1.181016.L758