Report for coding project #1, LRU Algorithm implementation

20131218 Park Kee Hun

1. newNode function

```
Node* newNode(int item_) {
    //TODO
        Node *N = (Node*)malloc(sizeof(Node));
        N->item = item_;
        N->next = NULL;
        N->prev = NULL;
        return N;
}
```

This function makes new node with item_. When I make a new node, I used the pointer dynamic memory allocation. Because if I don't use the pointer, the allocated memory of new node I made wil disappear when go out from newNode function. Moreover, I can't declared the pointer without any address, so I use dynamic memory allocation. After make the new node, I stored item_ into N->item and NULL into N->next, N->prev. At last, return the node made in this function.

2. createQueue function

```
Queue* createQueue(int size) {
    //TODO
        Queue *qq = (Queue*)malloc(sizeof(Queue));
        qq->queue_size = size;
        qq->count_ = 0;
        qq->head = NULL;
        qq->tail = NULL;
        return qq;
}
```

This function makes new queue with size. I also used the pointer dynamic memory allocation due to same reason in newNode function. I set the entire queue size to the value given from main.c, count_ to 0, head and tail to NULL. At last, return the queue made in this function.

3. isQueueEmpty function

```
int isQueueEmpty(Queue* queue_) {
    //TODO
        if(queue_->count_==0) return 1;
        else return 0;
}
```

This function is for check whether queue is empty. In structure queue, count_ variable means the number of elements in queue, so I just check whether queue count is zero. If it is empty, return 1. Otherwise, return 0.

4. DeQueue function

```
void DeQueue(Queue* queue_) {
    //TODO
        Queue *qd = queue_;
        qd->tail = qd->tail->prev;
        qd->tail->next = NULL;
        qd->count_--;
}
```

This function removes the last element(tail) in queue_. I just stored NULL into prev and next of tail of queue. Then I should do count_--; because tail of queue is terminated.

5. EnQueue function

```
void EnQueue(Queue* queue_, int item_) {
    //TODO
        Queue *qe = queue_;
        Node *Ae = newNode(item_);
        Ae->next = qe->head;
        qe->head->prev = Ae;
        qe->head = Ae;
}
```

This function adds the new node into the head of queue. This node take the queue_ and item_, then make the new node with item_ value. At last, I just connect the new node to head of queue, and then designate the new node to head of queue.

6. Reference function

```
void Reference(Queue* queue_, int item_) {
    //TODO
        Queue *qr = queue_;
        Node *Ar = newNode(item_);
        int check = 1;

        if(isQueueEmpty(qr)){
                qr->head = Ar;
                qr->tail = Ar;
                qr->count_++;}
        else{
                Node *temp = qr->head;
                while(1){
                    if(item_==temp->item){
                        if(check==1){
                            qr->head = temp->next;
                            temp->next->prev = NULL;
                            temp->next = NULL;}
                        else if(check==qr->count_){
                            qr->tail = temp->prev;
                            temp->prev->next = NULL;
                            temp->prev = NULL;}
                        else{
                            temp->prev->next = temp->next;
                            temp->next->prev = temp->prev;}
                        qr->count_--;
                        break;}
                    if(check==qr->count_) break;
                    temp = temp->next;
                    check++;}
                 EnQueue(qr,item_);
                 qr->count_++;
                if(qr->count_>qr->queue_size) DeQueue(qr);}
}
```

int check = 1; used to identify whether the node of queue is head or tail, when item of new node equals to the existing item of queue. I should devide the case of that node is head or tail, when items are same each other, to avoid segmentation fault.

By using function of isQueueEmpty, if queue is empty, the new node become both head and tail of the queue.

Otherwise, node is added to queue by EnQueue function. And before adding the node, I should check whether the same item of new node exist in the queue by using “check” variable. If there is a node whose item is same as item of new node, delete that existing node in queue, and do count_--;, and then add the new node with count_++;.

At last, if count of queue is larger than size of queue, delete the tail of queue by using DeQueue function.

7. ClearQueue function

```
void ClearQueue(Queue* queue_) {
    //TODO
        Queue *qc = queue_;
        Node *temp = qc->head;
        int check = 1;

        while(check <= qc->count_){
                if(check<qc->count_) temp = temp->next;
                free(qc->head);
                qc->head = temp;
                check++;}
        free(qc);
}
```

I used the pointer dynamic memory allocation in making new node and queue, I should clear the memory of them, when every work is done in main.c. So by using free function, release the memory allocated for each node in the queue.