CSCI 2170 Variations on Linked List

- **Circular Linked List**

Time sharing environment, all jobs are put in a job queue, where each is allocated a slice of CPU time when its turn is up. When the slice of time is over, if the job is not completed, the job is put back onto the job queue, the next job in the queue will get the CPU time.
The job queue can be maintained in the form of a linked list. For convenience, it is often desirable to be able to go ahead and process the next job in the queue even if it is at the end of the queue. i.e., from the last job in the queue, go directly to the first job in the queue.
This can be done by adding a link from the next field of the last node of the list to the first node in the list → circular linked list.

What does the list look like? What about an empty list?

Characteristics:

- every node in a circular linked list has a successor node
- no node in a circular linked list has NULL next pointer
- can start at any node in the list and traverse the entire list
- still need external pointer(s) to point to >=1 node in the list.
- What is the best way to set up these pointers?
    a. Use 1 pointer points to the beginning of the list
    b. Use 1 pointer points to the end of the list
    c. Use 2 pointers pointing to the beginning and the end of the list

Traverse list:

```
if (last  != NULL)
{
        ptrType   first=last→next;
        ptrType    curr = first;
        do
        {
                display (curr→item);
                cur = cur→next;
        } while (curr != first);
}
```

- **Dummy head in a linked list**

Why do we want to add a dummy head in the linked list?
It makes insertion and deletion easier. We no longer have to differentiate the two cases of insertion and deletion. Both cases will be handled the same way.

What will a linked list having a dummy node look like?
What will an empty linked list with a dummy head look like?

What will a circular linked list having a dummy head look like?
What will an empty circular linked list having a dummy head look like?

Insertion/deletion operation with <u>circular linked list with dummy head</u>.

<u>Insert operation:</u> (assuming a sorted linked list)

```
NodePtr newPtr = new Node;
newPtr→item = newItem;
newPtr→next = NULL;

curr = head→next;
prev = head;
while ((curr != head)&&(curr→item <newItem))
{
        prev = curr;
        curr = curr→next;
}

// prev points to the node before the insertion position
newPtr→next = curr;
prev→next = newPtr;
```

<u>Delete operation</u>

```
curr = head→next;
prev = head;
while ((curr != head)&&(curr→item != delItem))
{
        prev = curr;
        curr = curr→next;
}

if (curr!=head)
{
        prev->next = curr->next;

        curr->next = NULL;
        delete curr;
        curr = NULL;
}
else  // delItem is not in the list
{ …. }
```

- **Doubly Linked List**

Why do we need doubly linked list?
insertion and deletion in doubly linked list requires a single pointer traversal
What does a doubly linked list look like?
What does an empty doubly linked list look like?
What does a doubly linked list with a single node look like?

Data structure

```
struct Node
{
        Node * precede;
        ListItemType   item;
        Node * next;
};
typedef Node * NodePtr;
```

- **Circular Doubly linked list with dummy head**

What does a circular doubly linked list with dummy head look like?
What about an empty circular doubly linked list with dummy head?

Insertion and deletion with circular doubly linked list with dummy head requires a single pointer traversal and handles all situations in one case.

Insertion:

```
NodePtr newPtr = new node;
newPtr→item = newItem;

curr=head→next;
while ((curr != head) && (curr→item < newItem))
        curr = curr→next;

// needs to change four pointers
newPtr→next = curr;
newPtr→precede = curr→precede;

curr→precede→next = newPtr;
curr→precede = newPtr;
…
```

Deletion:

```
curr=head→next;
while ((curr != head) && (curr→item != itemToDelete))
{
        curr = curr→next;
}

if (curr!=head)
{
        curr→precede →next = curr→next;
        curr→next→precede = curr→precede;

        curr→next=NULL;
        curr→precede=NULL:
        delete curr;
        curr=NULL;
}
else {  …} // itemToDelete not in the list
```