CS162: Introduction to Computer Science II

Week 3: Singly Linked List (cont.)

02/2024

CS162 – What is next?

- Lecture: Dynamic Data Structures
  - Walk through the examples assigned earlier
  - Do other operations
    - inserting at the end of the linked list
    - inserting after another node in a linked list
    - inserting before another node in a linked list
    - removing at the beginning of a linked list
    - removing a node from the linked list

CS162 - Inserting at End

- Add a node at the end of a linked list.
  - What is wrong with the following. Correct it in class:

```
Node* cur = pHead;
while (cur != nullptr)
        cur = cur->pNext;
cur = new Node;
cur->data = new Video;
cur->data = data_to_be_stored;
```

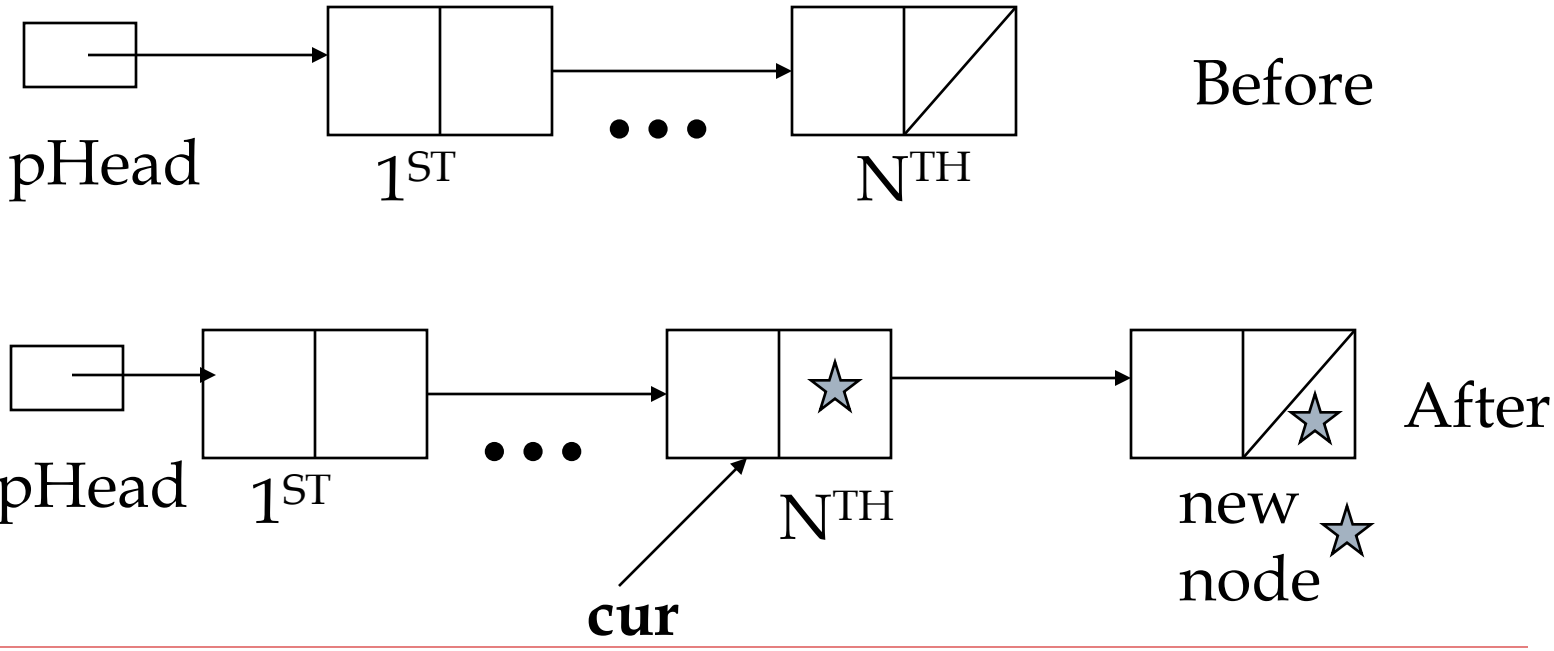
LOOK AT THE BOLD/ITALICS FOR HINTS OF WHAT IS WRONG!

CS162 - Inserting at End

- We need a temporary pointer because if we use the head pointer
  - we will lose the original head of the list and therefore all of our data
- If our loop's stopping condition is if current is not **nullptr --** then what we are saying is loop until current IS nullptr
  - well, if current is nullptr, then dereferencing current will give us a segmentation fault
  - and, we will NOT be pointing to the last node!

CS162 - Inserting at End

- Instead, think about the “before” and “after” **pointer diagrams:**



CS162 - Inserting at End

- So, we want to loop until current->pNext is not nullptr!
- But, to do that, we must make sure current isn't nullptr
  - This is because if the list is empty, cur will be nullptr and we'll get a fault (or should) by dereferencing the pointer

```
if (cur)
   while (cur->pNext != nullptr)
         cur = cur->pNext;
```

CS162 - Inserting at End

- Next, we need to connect up the nodes
  - having the last node point to this new node

```
cur->pNext = new Node;
```

  - then, traverse to this new node:

```
cur = cur->pNext;
```

```
cur->data = new Video;
```

  - and, set the next pointer of this new last node to null:

```
cur->pNext = nullptr;
```

CS162 - Inserting at End

- Lastly, in our first example for today, it was inappropriate to just copy over the pointers to our data
  - we allocated memory for a video and then immediately lost that memory with the following:

```
cur->data = new Video;
cur->data = data_to_be_stored;
```

  - the correct approach is to allocate the memory for the data members of the video and physically copy each and every one

Implement the following functions for the linked list

- inserting at the end of the linked list
- inserting after another node in a linked list
- inserting before another node in a linked list
- removing at the beginning of a linked list
- removing a node X from the linked list

CS162 - Removing at Beginning

- Now let's look at the code to remove at node at the beginning of a linear linked list.
- Remember when doing this, we need to deallocate **all** dynamically allocated memory associated with the node.
- Will we need a temporary pointer?
  - Why or why not...

CS162 - Removing at Beginning

- What is wrong with the following?

```
Node* cur = pHead->pNext;
delete pHead;
pHead = cur;
```

  - everything? (just about!)

CS162 - Removing at Beginning

- First, don't dereference the pHead pointer before making sure head is not nullptr

```
if (pHead) {
    Node* cur = pHead->pNext;
```

  - If pHead is nullptr, then there is nothing to remove!
- Next, we must deallocate all dynamic memory:

```
delete [] pHead->data->title;
delete pHead->data;
delete pHead;
pHead = cur; //this was correct....
```

CS162 - Removing at End

- Now take what you've learned and write the code to remove a node from the end of a linear linked list
- What is wrong with: (lots!)

```
Node* cur = pHead;
while (cur != nullptr)
      cur = cur->pNext;
delete [] cur->data->title;
delete cur->data;
delete cur;
```

CS162 - Removing at End

- Look at the stopping condition
  - if cur is nullptr when the loop ends, how can we dereference the cur? It isn't pointing to anything
  - therefore, we've gone too far again

```
Node* cur = pHead;
if (!pHead) return 0; //failure mode
while (cur->pNext != nullptr)
        cur = cur->pNext;
```

  - is there anything else wrong? (yes)

CS162 - Removing at End

- So, the deleting is fine....

```
delete [] cur->data->title;
delete cur->data;
delete cur;
```

  - but, doesn't the previous node to this still point to this deallocated node?
  - when we retraverse the list -- we will still come to this node and access the memory (as if it was still attached).

CS162 - Removing at End

- When removing the last node, we need to reset the new last node's next pointer to NULL
  - but, to do that, we must keep a pointer to the previous node
  - because we do not want to "retraverse" the list to find the previous node
  - therefore, we will use an additional pointer
    - (we will call it "previous")

CS162 - Removing at End

- Taking this into account:

```
        Node* cur= pHead;
        Node* prev = nullptr;
        if (!pHead) return 0;
        while (cur->next) {
              prev = cur;
              cur = cur->pNext;
         }
       delete [] cur->data->title;
       delete cur->data;
       delete cur;
       prev->pNext = nullptr;  //oops...
    }
```

Can anyone see the remaining problem?

CS162 - Removing at End

- Always think about what special cases need to be taken into account.
- What if...
  - there is only ONE item in the list?
  - prev->pNext won't be accessing the deallocated node (previous will be nullptr)
  - we would need to reset head to nullptr, after deallocating the one and only node

CS162 - Removing at End

- Taking this into account:

```
    •••
  if (!prev) //only 1 node
    pHead = nullptr;
  else
    prev->pNext = nullptr;
}
```

Now, put this all together as an exercise