CS162: Introduction to Computer Science II

Week 2b – Singly Linked List

01/2024

What is in CS162 today?

- Lecture: Dynamic Data Structures
  - Review of pointers and the new operator
  - Introduction to Linked Lists
  - Begin walking thru examples of linked lists

CS162 - Pointers

- What advantage do pointers give us?
- How can we use pointers and new to allocating memory dynamically
- Why allocating memory dynamically vs. statically?
- Why is it necessary to deallocate this memory when we are done with the memory?

CS162 - Pointers and Arrays

- Are there any disadvantages to a dynamically allocated array?
  - The benefit - of course - is that we get to wait until run time to determine how large our array is.
  - The drawback - however - is that the array is still fixed size.... it is just that we can wait until run time to fix that size.
  - And, at some point prior to using the array we must determine how large it should be.

CS162 - Linked Lists

- Our solution to this problem is to use **linear linked lists** instead of arrays to maintain a “list”
- With a linear linked list, we can grow and shrink the size of the list as new data is added or as data is removed
- The list is ALWAYS sized exactly appropriately for the size of the list

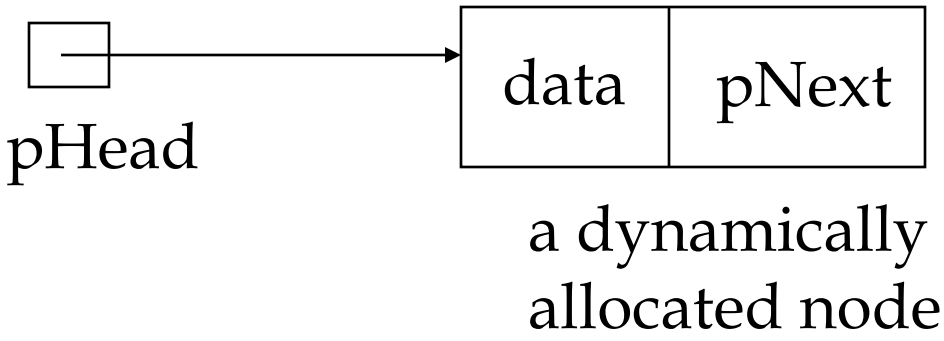
CS162 - Linked Lists

- A linear linked list starts out as empty
  - An empty list is represented by a null pointer
  - We commonly call this the **pHead** pointer

pHead

CS162 - Linked Lists

- As we add the first data item, the list gets one **Node** added to it
  - So, **pHead** points to a **Node** instead of being null
  - And, a **Node** contains the data to be stored in the list and a **pNext** pointer (to the next node...if there is one)

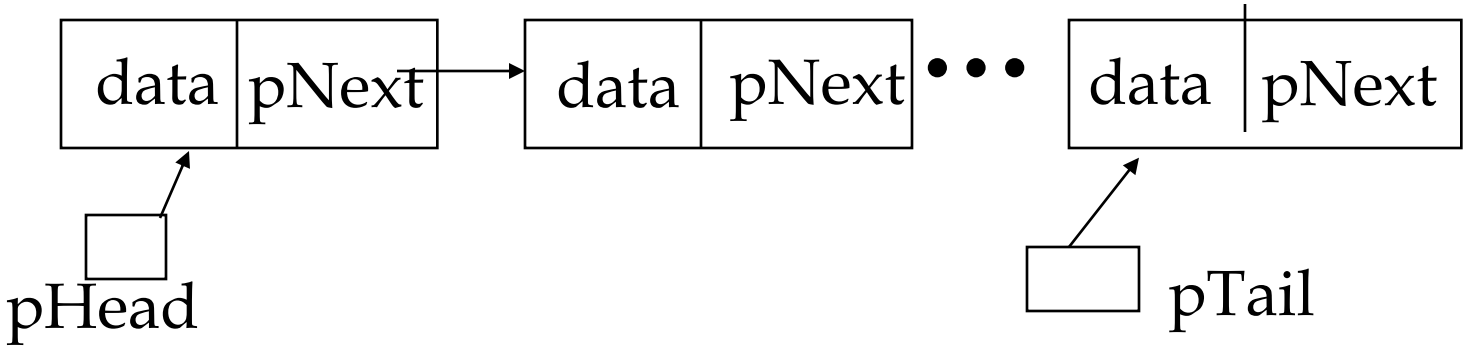


CS162 - Linked Lists

- To add another data item we must first decide in what order
  - does it get added at the beginning
  - does it get inserted in sorted order
  - does it get added at the end
- This term, we will learn how to add in each of these positions.

CS162 - Linked Lists

- Ultimately, our lists could look like:



Sometimes we also have a tail pointer. This is another pointer to a node -- but keeps track of the end of the list. This is useful if you are often adding data to the end

CS162 - Linked Lists

- So, how do linked lists differ than arrays?
  - An array is direct access; we supply an element number and can go directly to that element (through pointer arithmetic)
  - With a linked list, we must either start at the head or the tail pointer and **sequentially traverse** to the desired position in the list

CS162 - Linked Lists

- In addition, linear linked lists (singly) are connected with just one set of **pNext** pointers.
  - This means you can go from the first to the second to the third to the forth (etc) nodes
  - But, once you are at the forth you can't go back to the second without starting at the beginning again.....

CS162 - Linked Lists

- Besides linear linked lists (singly linked)
  - There are other types of lists
    - Circular linked lists
    - Doubly linked lists
    - Non-linear linked lists (CS163)

CS162 - Linked Lists

- For a linear linked lists (singly linked)
  - We need to define both the head pointer and the node
  - The node can be defined as a struct or a class; for these lectures we will use a struct but on the board we can go through a class definition in addition (if time permits)

CS162 - Linked Lists

- We'll start with the following:

```
struct Video {    //our data
   char* title;
   char category[5];
   int quantity;
};
```

- Then, we define a node structure:

```
struct Node {
   Video data;   //or, could be a pointer
   Node* pNext;  //a pointer to the next
};
```

CS162 - Linked Lists

- To manage the list

```
node* pHead;
```

- We want the list to be empty to begin with, so head should be set to **nullptr**

```
pHead = nullptr;
```

CS162 - Traversing

- To show how to traverse a linear linked list, let's spend have a look on the source code

```
Node* cur = pHead;
while (cur != nullptr) {
  cout << cur->data.title << '\t'
   << cur->data.category << endl;
  cur = cur->pNext;
}
```

CS162 - Traversing

- Let's examine this step-by-step:
  - Why do we need a "cur" pointer?
  - What is "cur"?
  - Why couldn't we have said:

```
while (pHead != nullptr) {
   cout <<pHhead->data.title << '\t'
      <<pHead->data.category << endl;
   pHead = pHead->pNext;    //NO!!!!!!
}
```

We would have lost our list!!!!!!

CS162 - Traversing

- Next, why do we use the nullptr stopping condition:

```
while (cur != nullptr) {
```

- This implies that the very last node's next pointer must have a **nullptr** value
  - so that we know when to stop when traversing
  - nullptr is 0 (zero)
  - So, we could have said:

```
while (cur) {
```

CS162 - Traversing

- Now let's examine how we access the data's values:

```
cout <<cur->data.title <<'\t'
<<cur->data.category <<endl;
```

- Since current is a pointer, we use the -> operator (indirect member access operator) to access the "data" and the "pNext" members of the Node structure
- But, since "data" is an object (and not a pointer), we use the . operator to access the title, category, etc.

CS162 - Linked Lists

- If our node structure had defined data to be a pointer:

```
struct Node {
   Video* pData;
   Node* pNext;
};
```

- Then, we would have accessed the members

```
cout << cur->pData->title << '\t'
          << cur->pData->category << endl;
```

(And, when we insert nodes we would have to remember to allocate memory for a Video object in addition to a Node object...)

CS162 - Traversing

- So, if current is initialized to the head of the list, and we display that first node
  - to display the second node we must **traverse**
  - this is done by:

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

  - why couldn't we say:

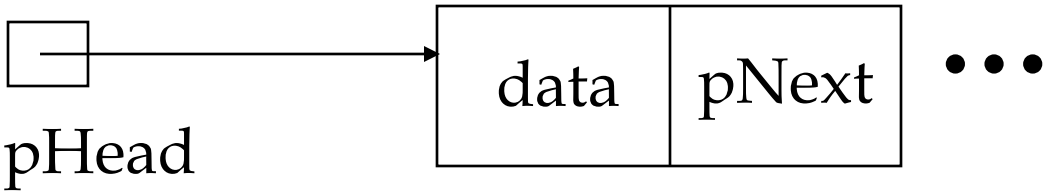
```
cur = pHead->pNext;   //NO!!!!!
```

CS162 - Building

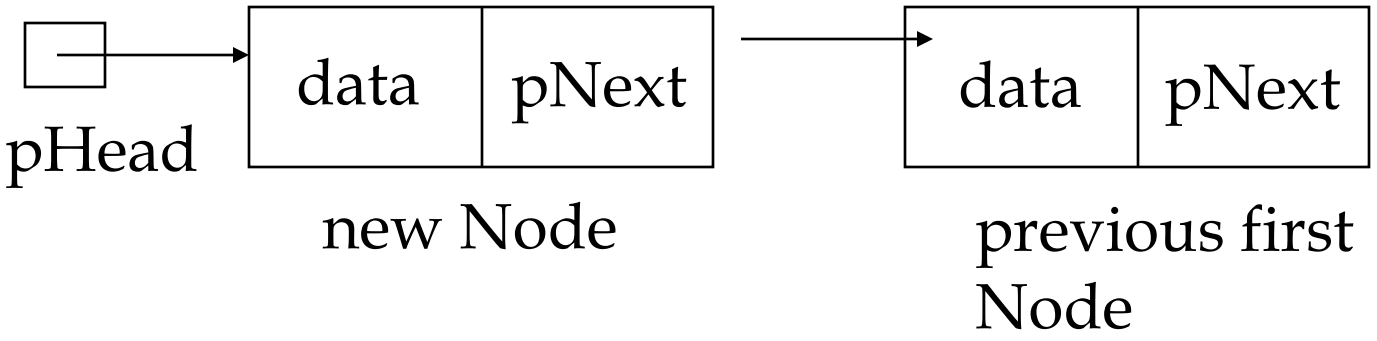
- Well, this is fine for traversal
- But, you should be wondering at this point, how do I create (build) a linked list?
- So, let's write the algorithm to add a Node to the **beginning** of a linked list

CS162 - Insert at Beginning

- We go from:



- To:



- So, can we say:

```
pHead = new Node; //why not???
```

CS162 - Insert at Beginning

- If we did, we would lose the rest of the list!
- So, we need a temporary pointer to hold onto the previous head of the list

```
Node * cur = pHead;
pHead = new Node;
pHead->data = new Video; //if data is a pointer
pHead->data->title = new char[strlen(newtitle)+1];
strcpy(pHead->data->title, newtitle);  //etc.
pHead->pNext = cur;  //reattach the list!!!
```

CS162 - For Next Time

- Practice Linked lists
- Do the following and bring to class:
  - add data to the end of a linear linked list
  - remove data at the beginning of a linear linked list