**CS301 Bootcamp: Link-Lists**

**Why linked lists**

* Linked lists can help us do more stuff with data!
  + We can \_\_\_\_\_\_\_\_\_\_\_\_\_ stuff into the middle of a list
  + We can \_\_\_\_\_\_\_\_\_\_\_\_\_\_ stuff from the middle of a list
  + We can easily add stuff to the \_\_\_\_\_\_\_\_\_\_\_ or \_\_\_\_\_\_\_\_\_\_\_\_\_ of a list
  + Linked lists live in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_memory, so the size can be easily adjusted
* For Computer Science and Computer Engineering majors: you will take an entire course on Data Structures. You will also use data structures in an algorithms course. So make sure you remember this stuff about pointers and linked lists!

**Where linked lists are more efficient**

* Linked lists are very useful when…
  + you want to insert items at the beginning or middle of your list.
  + Your data needs to be able to easily grow and shrink as needed
  + You want to use dynamic memory
* Linked lists are not very useful when…
  + You need to access a particular element or “index” (linked lists are not efficient for this – we have to read through the whole list!)

**Linked list organization**

Chart, box and whisker chart

Description automatically generated

* A Linked list contains 0 or more nodes:
* The linked list’s head to point to first node
* Last node points to null(address 0)

**Example**

**Draw what the code produces and explain your answer:**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Example**

**Draw what the code produces and explain your answer:**

**Graphical user interface, text, application

Description automatically generated**

**Example**

**Draw what the code produces and explain your answer:**

Diagram

Description automatically generated

**Example**

**Draw what the code produces and explain your answer:**

Diagram

Description automatically generated

**Example**

Diagram

Description automatically generated

**Example**

**Draw what the code produces and explain your answer**

Diagram

Description automatically generated

**Linked List good practices**

* Make sure you check for an \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ list before manipulating it
* You should always keep one pointer variable pointing to the head of the linked list. This pointer can be used as the linked list argument in functions and to help you access the data.
* Be careful not to lose your \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_node. If you overwrite your pointer which points to the head node, you can lose your ability to access that node. You could lose your ability to access your memory, and also tie up memory that cannot be used for other parts of your program.

**Memory leaks and lost nodes**

* Lost nodes can create \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
* Memory leaks can cause the operating system to crash when it runs out of memory.
* To avoid memory leaks, always keep some pointer pointing to the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of the linked list.

Diagram

Description automatically generated

**Linked List Operations to know**

* Basic operations:
  + append a node to the end of the list
  + insert a node within the list
  + traverse the linked list
  + delete a node
  + delete/destroy the list

**C++ Example: Adding a node to the head of a list**

void IntLinkList::Add(int elem)

{

// Make and initialize a new node

Node\* node = new Node;

node->next = nullptr;// Good practice to immediatly initialize

node->data = elem;

// Add the new node to front

node->next = head;

head = node;// Node is now the head

// Book keeping... list size increased by one

length++;

}

**Overview: Inserting a node into a linked list**

**To insert, you need to**

**1) add a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ from the new node to the list, and**

**2) then add a\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ from the list to the new node.**

Diagram, schematic

Description automatically generated

CAUTION!!!! when changing the pointers, be careful not to “\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_” any nodes or lose the head! Be sure to maintain some connection to your \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and all nodes at all times. Otherwise, your program will create memory leaks and lost nodes.

**Example: inserting nodes**

Diagram

Description automatically generated

**Example: inserting nodes**

1. **Identify the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ where you want to insert the new nodes**

Diagram

Description automatically generated

**2. Connect the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ from the new node to the list to create a link**

Diagram

Description automatically generated with medium confidence

**3. Now we add a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_from the list to the new node:**

Diagram

Description automatically generated

**4. Now we need to update the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pointer, since the head pointer has changed from head to otherListHead**

Diagram

Description automatically generated**All finished!**

Diagram

Description automatically generated

**Codey sense:**

Chart

Description automatically generated

**Key takeaway:** **inserting nodes. To insert, you need to:**

1. **add a link from the new node to the list, and**
2. **then add a link from the list to the new node.**

**CAUTION!!!! when changing the pointers, be careful not to “\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_” any nodes or lose the head! Be sure to maintain some \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ to your head and all nodes at all times. Otherwise, your program will create memory leaks and lost nodes.**

**Stepping through a linked list**

Recall, the -> operator allows us to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ through a linked list.

If next is the name of our pointer, we can use \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ to advance to the next item.

Diagram

Description automatically generated with medium confidence

**Quick question**

A picture containing text, clock

Description automatically generated

**Things to be aware of when stepping**

* You can only travel one \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ in a (singly) linked list. You cannot use -> next to go \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
* There are special types of linked lists (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ linked lists) where you have two pointers per node, allowing travel in both directions.
* Our book likes to name the linked list pointers “link” but programmers also commonly name the pointers “next”.

**How to search in a linked list: pseudocode**

1. Declare a pointer (“curr”) to point to a node type struct.
2. Set curr to point to the head initially
3. Check whether the node curr is pointing to contains the desired value
4. If the curr pointer’s points at the desired value, return the pointer curr.
5. If not, move to the next node of the linked list (curr = curr->next)
6. Otherwise, if we search all the way until we reach null, the desired value must not be present in the Linked List.

**Hints: Searching and inserting**

If you need to search and then insert something into a linked list, you should declare two pointers to help you search.

One pointer to point at the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ location

One pointer to point at the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ location

Diagram

Description automatically generated

You can use a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ to step through the linked list until the current and previous pointers find the desired location.

Then, you can use current and previous pointers to help insert the node you want to insert:

**More hints: Searching and Inserting**

Diagram

Description automatically generated

**Codey sense: A vs B**

Graphical user interface, application

Description automatically generated

**Making constructors**

* Recall, you can use a class to make nodes for a linked list or other data structure
* You can declare constructors to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ objects for nodes.
  + It’s a good idea to initialize pointers to NULL initially and initialize data in your object to a known value

**Example: Constructor for IntLinkList class:**

IntLinkList::IntLinkList()

{

// Set up the list

head = nullptr;

length = 0;

}

**Deleting items and Destructors**

Destructors are special member functions used to delete objects from a class.

You put a ~ in front of the class name to create a destructor.

Destructors allow us to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ linked lists and nodes when we are finished.

If we do not delete our linked lists, we could run out of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_and cause our computer to crash.

It’s a good practice to always include destructors when using linked lists so that you can clean up your memory when you are finished using it.

**One way to define a destructor**

* You can also make a destructor using a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ to delete the items in the linked list.
* This code will read each item of the linked list, starting with the head and ending with null. This code changes head to be the next value in the list, and then deletes the current value.
* Once we reach null, all nodes will be deleted.

Diagram

Description automatically generated\

**Example: Destructor for IntLinkList class**

// Destructor

IntLinkList::~IntLinkList()

{

Node\* oldHead;

while (head != nullptr) // Don't need to delete things if the head is null

{

oldHead = head;// Remember the original head

head = head->next;// Advance the head

delete oldHead;// Delete the old head

}

}

**Important notes about destructors**

* Destructors must be declared and defined by the programmer in order to actually \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_nodes.
* Like the default constructor, the compiler will automatically create a destructor for you if you forget, but the destructor will be \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. It will not do anything unless we define it.

**FYI: Copy constructors**

* When you pass a linked list object into a function, it is passed by value. This can be a problem when we are passing memory addresses.
* A copy constructor is used to make a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ copy is a copy of a linked list at an entirely new memory address. This helps prevent the linked list and its copy from sharing data at the same memory address.

**Example: Write the Add function to add a node to the front of the list**

Graphical user interface, text, application, chat or text message

Description automatically generated

Example: Write the Contains function

Graphical user interface, text, application, chat or text message

Description automatically generated