CS 102
Introduction to Programming Using C++

Chapter 7

Pointers

Homework

- Written homework
- R7.1, 2, 4, 7, 8, 15, 16, 19, 21 to 27

- Programs
- p. 344, choose one of P7.1, 2, or 4.
- Also, implement a linked list.

A Simple Example

- You just got hired by a small museum
- They would like to keep track of how many people visit the museum each day
- You decide to write a program that uses a linked list to store this information
- Your first step will be to design a node (struct) for the list
- The first two days' data are 122 visitors, 135 visitors

Creating Nodes:
First Things First

- A node should have two parts
  - It should have a data part
    - This is where you store the actual data
  - It should have a link part
    - This is the pointer to the next node

The Node + The List

- The node has to
  - Contain an integer
  - Contain a pointer that points to the next node in the list

```
struct a_node
{
   int number_of_visitors;
   a_node* next_node;  //  this will be a pointer to the
                       //  next node
};
a_node* linked_list;
```

Those Two Parts

- Let’s check on the two parts
    - number_of_visitors is where we will store the actual data
    - next_node is the link that will point to the next node
        - Notice how the type of next_node is a_node*
        - This is a pointer to a_node
            - That’s how we build a list
        - When we write the program, we will have to make sure that next_node points to the next node in the list

Creating the List

- Initially, we need the list to be empty
- We do this by

  ```
  linked_list = NULL;
  ```

- NULL is a special keyword that denotes a pointer that points nowhere
- Every time you use a pointer inside a program, you need to check if it's NULL or not
- If you try to use a pointer that is NULL, your program will crash

Building the List

- Let’s add a single node to the list
- Just like before, we need to request some memory
- We do this with new

```
linked_list = new a_node;
```

- linked_list is now a pointer to a_node
- Then we store something into the node

```
linked_list ->number_of_visitors = 135;
```

- Notice that we add the second day’s data first

Making Sure the List Ends

- We only have one node
  - Later, we will check on the list
  - We have to make sure that it has an end
- We set next_node to NULL:

```
 linked_list->next_node = NULL;
```

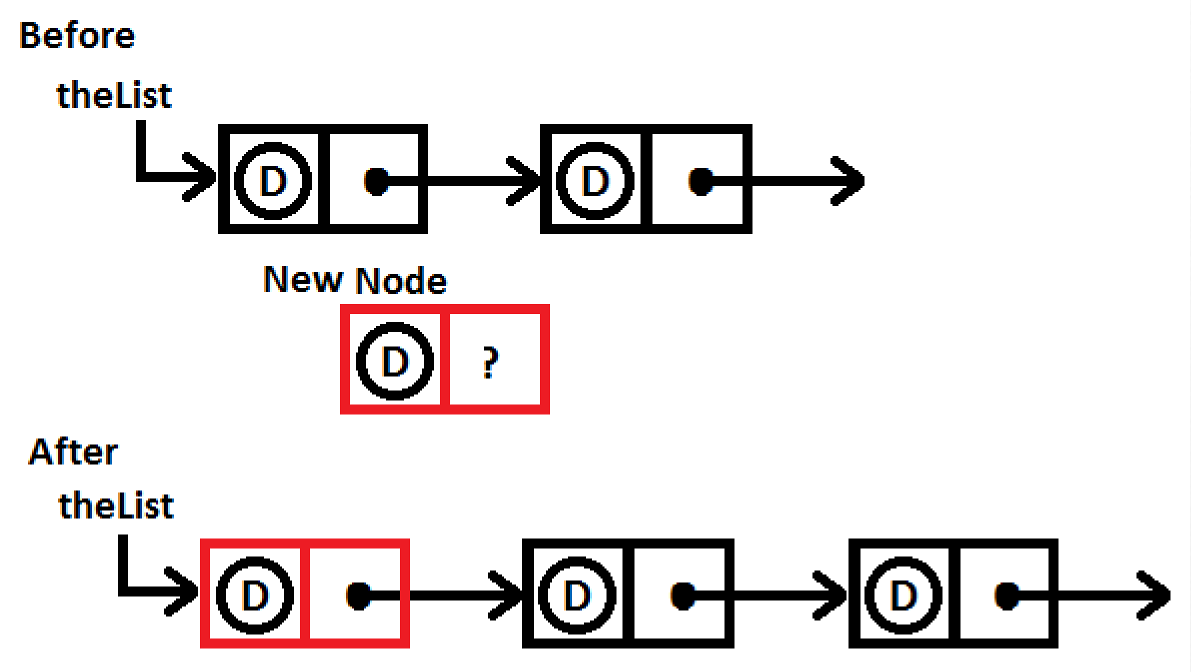
- Now, if we want, we can print that node

```
cout << “Linked list so far: "
     << linked_list->number_of_visitors’;
```

Adding More Nodes to the List

- Now we want to add more nodes
- We will add a new node in front of the node we now have
  - This is the easiest place to add, in general
  - We could however, add a node anywhere

A Reminder-Adding a Node at the Beginning of the List



Adding More Nodes at the Beginning of the List

- Now we want to add more nodes
- We will add a new node in front of the node we now have
  - This is the easiest place to add, in general
  - We could however, add a node anywhere

Adding a Node at the Beginning of the List: The Code

- Create a new node

  ```
  a_node* temp_node = new a_node;
  ```

- Store the data into the node

  ```
  temp_node->number_of_visitors = 122;
  ```

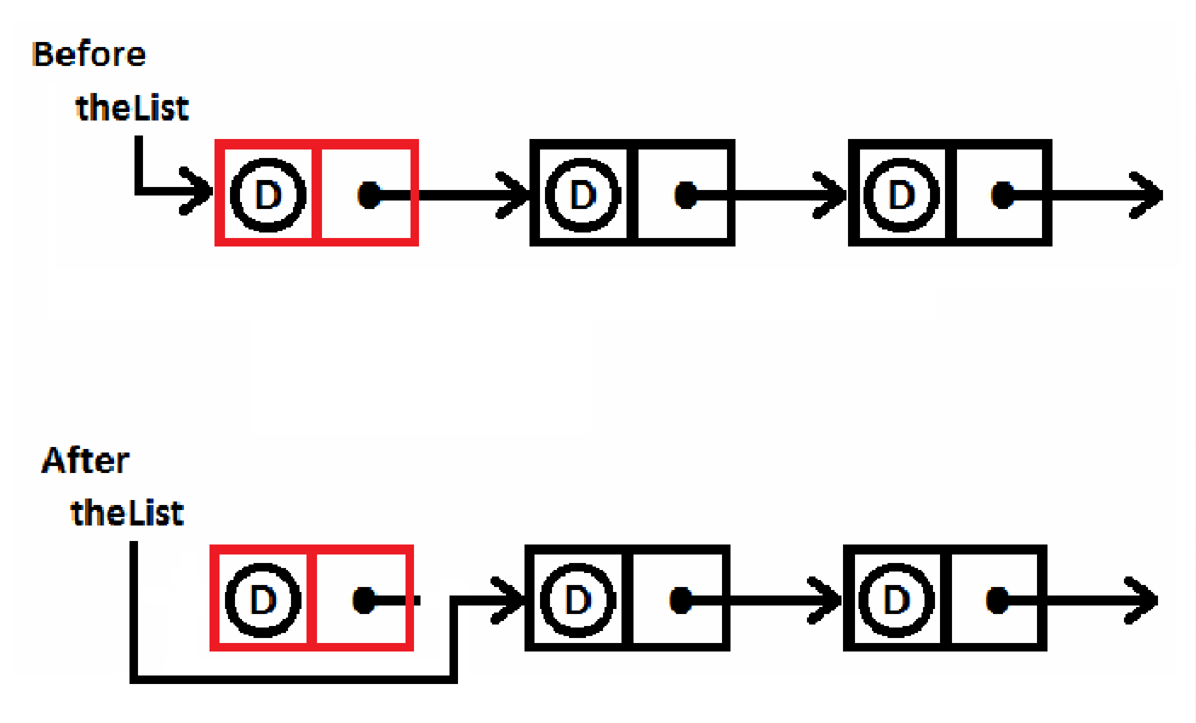
- Set temp_node’s next neighbor to linked_list

  ```
  temp_node->next_node = linked_list;
  ```

- Set linked_list to point to temp_node

  ```
  linked_list = temp_node;
  ```

A Reminder-Deleting a Node from the Beginning of the List



Deleting a Node from the Beginning of the List

- This is super easy!
- You just make linked_list point to the next node
- But, there is a slight problem
    - linked_list originally pointed to a node
    - We just ignored that node
- The OS doesn't know that we don't need the node any more
- We have no way of accessing the memory that theList used to point to
- We need to give this memory back to the OS

A Memory Leak

- We call the node previously pointed to by linked_ list a memory leak
- Where is that memory?
- We are in a weird situation
  - We cannot access the memory formerly pointed to by linked_list
  - The OS can’t give that memory to another program
  - Can some program use that memory or not?
  - It’s as if that memory isn’t even there!
- The solution is that we need to release any pointers that we are not using
- We need to tell the OS that we don’t need the memory any more
- That allows the OS to give it to another program

Releasing the Memory Used by a Pointer

- The way we release no longer needed memory is to use delete
- Here is the actual code

```
    a_node* save_ptr = linked_list;
    //  Code to actually remove the node from
linked_list
    delete save_ptr;
```

Dangling Pointers

- Be careful when using delete with a pointer
- What if you delete a pointer and forget that you deleted it?
- If you then try to use this pointer, we call it a dangling pointer
- It will usually lead to a program crash

Traversing a Linked List

- Oftentimes, we will need to process every node in the list
  - For example, we may want to print the list
- We use this code

```
a_node* temp_node = linked_list;
while (temp_node != NULL)
{
    // Code to process the current node (temp_node)
    temp_node = temp_node->next_node;
}
```

Using new

- What is the difference between these?
  - a_node* temp_node;
  - a_node* temp_node = new a_node;
  - a_node* temp_node = linked_list;
  - a_node* temp_node = NULL;

- When would you use each of them?

Pictures Aren’t Reality

- A word of caution:
  - There is no such (physical) thing as a linked list in a program
- It is an idea
  - The list is in memory, not in the program
- There is one node that is actually in the program
  - That is the first node in the list
  - It’s the only thing actually in the program
- The next node is
  - Somewhere in memory
  - It is linked to the first node
- That idea continues for the rest of the nodes
- All these extra nodes are only in memory

Errors with Pointers

- Many bugs in programs come from misuse of pointers
- Let's examine the errors discussed on p. 327

Writing a Program to Implement a Linked List

- You need to write a program to manage a linked list
  - In my examples, I will assume the data in the list is int data
  - However, you can put whatever you want in the list
  - You can (should!) build a struct that is meaningful to you

Writing a Program to Implement a Linked List

- You need to write a program to manage a linked list
- Your program should have four functions
  - First function: main ()
    - You knew that was coming!
    - The main() function should test the code in the other functions
  - Second function: void insert_node (int data)
    - Create a new node
    - Store data into it
    - Link the node into the start of the chain
  - Third function: int delete_node ()
    - Delete the node at the start of the chain
    - Release the node's memory
    - Return what was stored in the node

That Last Function

- The last function should be a function that prints the list
- You can use the code that traverses a list
  - You were going to memorize it after all ☺

The main() Function

- Your main() function should
  - Add a node to the (empty) list
  - Add a second node to the list
  - Add a third node to the list
  - Delete a node from the list
  - Print the data that was stored in the node that you just deleted
- You should print the list at each stage to verify that it's correct

Questions?

- Are there any questions?