CSCI 262 Lecture 4 – Linked Lists, part 1

Outline

• Linked list structure & nodes
• Creating a linked list and printing its elements
  ○ Dynamic allocation of objects (more detail to come in a later lecture)
  ○ How to iterate on a linked list
• Making a class to better encapsulate our data structure

Readings
For today and next lecture: Chapter 14.1 – 14.2 in your textbook.

Self Check

1. A __________ is the basic building block of a linked list, and has storage for one value (an element of the list) and a pointer to the next object of the same type.

2. The start of the linked list is called the __________, and the end is called the __________.

3. The __________ operator applied to a pointer to an object lets us access a member variable of that object. (We will use this operator a lot in working with linked lists.)

4. When iterating through a linked list, we know we have reached the end when we encounter a __________ pointer value.

5. ______________ is the word we use to describe packaging data together with the methods that operate on that data; this technique helps us protect users from messing up the internal structure of our data structures.

For Further Practice

• The linked_list class code from the lecture notes works as written – copy and paste, or type it in to a CLion project, then write a main() function to test it.

• In anticipation of next lecture, see if you can write a working add_to_head() method for our class.

• A linked list node is a recursive object – it uses a pointer to its own type within itself. Linked structures tend to work well with recursive functions. If you work with just the node class examples (not the encapsulated linked_list class), can you write print_list() function that works recursively? Hint: you can think of a linked list as just a node object containing a pointer to a (smaller) linked list.
  ○ Now, can you print the list in reverse? (This is easy with recursion, very difficult with iteration!)