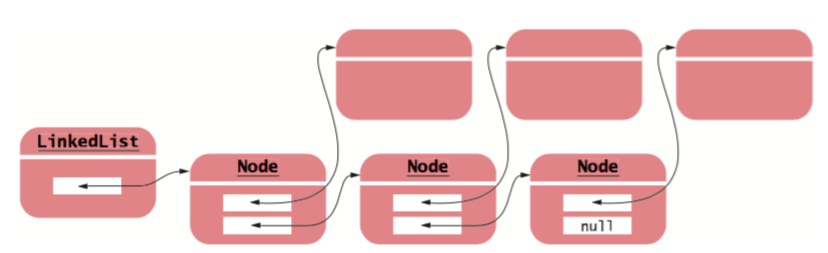
# Self Check Questions

1. Do LinkedLists take more storage than arrays of the same size?
   1. Yes, because each entry of a LinkedList is a Node object that stores both a value and a reference to the next Node, while each entry of an array only stores a value.
2. Why don’t we need iterators with arrays?
   1. Because an array allows for random access of its elements, while a LinkedList only allows for sequential access.
3. Trace through the addFirst method when adding an element to an empty list.
   1. A new Node named newNode is created. Its variables, data and next, are assigned the value of element and the address for the private variable first, respectively. The variable first is then assigned the address for newNode.
4. Conceptually, an iterator points between elements (see Figure 3). Does the position reference point to the element to the left or to the element to the right?
   1. The position reference points to the element to the right.
5. Why does the add method have two separate cases?
   1. Because the method normally adds a new Node after the previously visited one, but if there is no previously visited Node (i.e. if the list is empty), then a new one has to be created.
6. What is the advantage of viewing a type abstractly?
   1. It’s possible to look at the fundamental properties of the data type without being bogged down by the concrete implementation of it.
7. How would you sketch an abstract view of a doubly linked list? A concrete view?
   1. 
   2. 
8. How much slower is the binary search algorithm for an abstract list compared to the linear search algorithm?
9. Draw a sketch of the abstract queue type, similar to Figures 9 and 11.
   1. 
10. Why wouldn’t you want to use a stack to manage print jobs?
    1. Because you wouldn’t want to interrupt a printing operation with another one that was just added.

# Programming Exercise

P15.22: Use a stack to enumerate all permutations of a string. Suppose you want to find all permutations of the string meat. Push the string +meat on the stack. Now repeat the following operations until the stack is empty:

* Pop off the top of the stack.
* If that string ends in a + (such as tame+), remove the + and print the string.
* Otherwise, remove each letter in turn from the right of the +, insert it just before the +, and push the resulting string on the stack. For example, after popping e+mta, you push em+ta, et+ma, and ea+mt.

<https://repl.it/Lhy8>

# Projects

Project 15.1: Implement a class Polynomial that describes a polynomial such as p(x) = 5x10 + 9x7 − x − 10.

* Store a polynomial as a linked list of terms. A term contains the coefficient and the power of x. For example, you would store p(x) as (5, 10), (9, 7), (−1, 1), (−10, 0)
* Supply methods to add, multiply, and print polynomials, and to compute the derivative of a polynomial.

Project 15.3: Implement the following algorithm for the evaluation of arithmetic expressions.

* Each operator has a precedence. The + and - operators have the lowest precedence, \* and / have a higher (and equal) precedence, and ^ (which denotes “raising to a power” in this exercise) has the highest. For example, 3 \* 4^2 + 5 should mean the same as (3 \* (4 ^ 2)) + 5 with a value of 53.
* In your algorithm, use two stacks. One stack holds numbers, the other holds operators. When you encounter a number, push it on the number stack. When you encounter an operator, push it on the operator stack if it has higher precedence than the operator on the top of the stack. Otherwise, pop an operator off the operator stack, pop two numbers off the number stack, and push the result of the computation on the number stack. Repeat until the top of the operator stack has lower precedence. At the end of the expression, clear the stack in the same way.