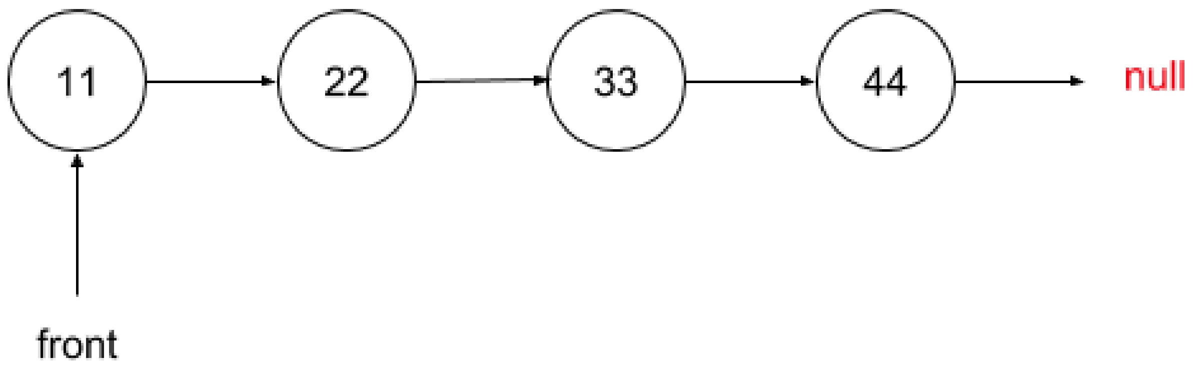
Additional information and examples – Homework 1

To understand the conceptual view of a linked list, below, is a picture (extracted from https://courses.cs.washington.edu/courses/cse373/17au/project1/diagrams/singly-linked-list.png)



Each record that holds data and a pointer is called a node. front is a pointer to the first element in the list. What is actually stored depends on how the list is represented.
We'll use an array to implement our list and the above list storage would actually look like what you see below,assuming that the elements were inserted in the order they are stored. (A little later we'll discuss how theprogram decides where to store new data, but when storage is initially empty, it is straightforward.)

Index	Data	Next
[0]	11	1
[1]	22	2
[2]	33	3
[3]	44	-1
[4]		
[5]		
[6]		

front: 0

A call to printList would display: 11, 22, 33, 44

If the elements were inserted in the order 33, 22, 11, 44, then the list would look like:

Index	Data	Next
[0]	33	3
[1]	22	0
[2]	11	1
[3]	44	-1
[4]		
[5]		
[6]		

front: 2
A call to printList would display: 11, 22, 33, 44

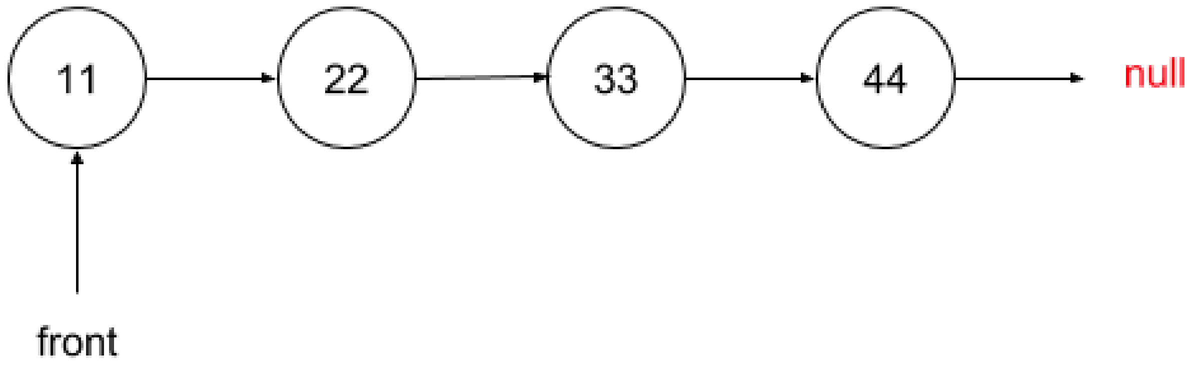
If the elements were inserted in the order 44, 33, 22, 11, then the list would look like:

Index	Data	Next
[0]	44	-1
[1]	33	0
[2]	22	1
[3]	11	2
[4]		
[5]		
[6]		

front: 3

A call to printList would display: 11, 22, 33, 44

Before moving on, please be sure you understand why all three pictures above represent the same list, shown again here.



Assuming the last picture is the actual representation currently in effect, a call to remove 22, would result in the following storage:

Index	Data	Next
[0]	44	-1
[1]	33	0
[2]	22	1
[3]	11	1
[4]		
[5]		
[6]		

front: 3
A call to printList would display: 11, 33, 44
Note: there is no need to actually remove the 22, the list just bypasses it.

Where would a call to insert 6, place 6?

In order to manage the free space, we need to keep track of it.
This is to be done using a linked list of free space. When the storage is initialized, and the data list is empty, storage should look like the following:
front: -1

Index	Data	Next
[0]		1
[1]		2
[2]		3
[3]		4
[4]		5
[5]		6
[6]		-1

empty: 0
Note: that when the data list is empty, front = -1. When the data list is full, empty = -1.
A call to dumpList would show what you see above, including the values of front and empty.

What is shown above is the list of data is empty, but the list of available spaces to store data starts with location 0, followed by location 2, etc. So the complete picture of the storage of the previous example (before the empty list) would be:
front: 3

Index	Data	Next
[0]	44	-1
[1]	33	0
[2]	22	4
[3]	11	1
[4]		5
[5]		6
[6]		-1

empty: 2

A call to printList would display: 11, 33, 44
A call to dumpList would show what you see above, including the values of front and empty.

Above, there is no need to actually remove the 22, the list just bypasses it. And when a new data item is inserted, it gets written over.

Consider the data shown next, as a result of insert 40 (which should come between 33 and 44 in the list since it is ordered).
front: 3

Index	Data	Next
[0]	44	-1
[1]	33	2
[2]	40	0
[3]	11	1
[4]		5
[5]		6
[6]		-1

empty: 4

A call to printList would display: 11, 33, 40, 44.
A call to dumpList would show what you see above, including the values of front and empty.

Here is what the lists look like after a series of remove operations:
remove (11)
front: 1
empty: 3

Index	Data	Next
[0]	44	-1
[1]	33	0
[2]	40	3
[3]	11	4
[4]		5
[5]		6
[6]		-1

A call to printList would display: 33, 40, 44.

remove (40)
front: 1
empty: 2

Index	Data	Next
[0]	44	2
[1]	33	0
[2]	40	3
[3]	11	4
[4]		5
[5]		6
[6]		-1