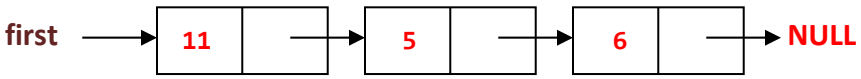
Review Activity 5 Solutions

List ADT, Recursion Exercises

1) Consider the following code fragment that is executed on a LinkedList* list.

```
LinkedList* list = new LinkedList(); // creates a new list
InsertFront(list, new Node(6)); // inserts node at the front
InsertFront(list, new Node(5));
InsertFront(list, new Node(3));
InsertBack(list, new Node(12)); // inserts node at the end
DeleteFirst(list); // deletes node from the front
InsertFront(list, new Node(11));
DeleteLast(list); // deletes node from the end
```

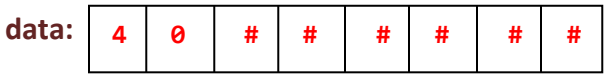
In the diagram below, illustrate what happens (the final outcome) after this code is run by drawing additional nodes, filling in the node values, and connecting nodes as appropriate. Each node includes data of `int` type and a pointer to the next node, and both values have to be set correctly.



2) Consider the following pseudocode that is executed on a sequential implementation of List ADT.

```
insert('m', 1);
insert('t', 0);
insert('e', 2);
insert('1', 2);
insert('4', 0);
insert('0', 1);
delete(2);
```

In the diagram below, illustrate what happens (the final outcome) after this code is run by filling in the node values below. Each node includes data of char type, and to indicate empty space, use '#' symbol. You can assume that the list is empty at first.



3) Design a recursive function that converts a string of digits into the matching integer. For example, ascii2int("2456") returns 2456.

```
int pow(int base, int n) { // compute base^n
	if (n <= 0) // handle only positive powers
		return 1;
	else if (n == 1)
		return base;
	else
		return base * pow(base, n - 1);
}
```

```
int ascii2int(std::string str) {
	if (!str.length()) // base case
		return 0;
	else if (str.at(0) == '-') // handle negative values
		return -1 * ascii2int(str.substr(1));
	else if (str.at(0) > '9' || str.at(0) < '0') // insert 0 for non-digits
		return ascii2int(str.substr(1));
	else // recursive case
		return ascii2int(str.substr(1)) +
			(str.at(0) - '0') * pow(10,str.length()-1);
}
```

4) Design a recursive function that converts an integer into the matching string of digits. For example, int2ascii(3452) returns "3452".

```
std::string int2ascii(int val) {
   if (val < 0) // handle negative values
         return '-' + int2ascii(-val);
   else if (val <= 9) // check for single digits
         return std::string(1, '0' + val);
   else // recursive case
         return int2ascii(val / 10) + char('0' + val % 10);
}
```

5) Design a recursive function that converts a given binary string into the matching decimal number. For example, bin2dec("101") returns 5.

```
int bin2dec(std::string str) {
   if (!str.length()) // base case
         return 0;
   if (str.at(0) == '-') // handle negative values
         return -1 * bin2dec(str.substr(1));
   else if (str.at(0) > '1' || str.at(0) < '0') // insert 0 for non-digits
         return bin2dec(str.substr(1));
   else
         return bin2dec(str.substr(1)) + (str.at(0) - '0') * pow(2,str.length()-1);
}
```

6) Design a recursive function that converts a given decimal number into the matching binary string. For example, dec2bin(12) returns "1100".

```
std::string dec2bin(int n) {
    if (n < 0) // handle negative ints using signed magnitude
         return "-" + dec2bin(-n);
   else if (n == 0) // base case1
         return "0";
   else if (n == 1) // base case2
         return "1";
   else
         return dec2bin(n / 2) + (n % 2 ? "1" : "0");
}
```