**CS 211 Introduction to Programming**
**Fall 2019**

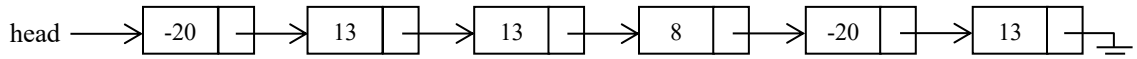
**Programming Assignment 9**
*(Assigned: November 7, 2019; Due: November 19, 2019)*

*The goal of this assignment is to get some experience with pointers and dynamic memory by implementing a simple linked list.*

## Background:

A *set* is a collection, in which any object may appear at most once. For example, {-20, 8, 13} is a set of three integer values. A *multiset* is like a set, but it allows objects to appear more than once. As an example, {-20, 13, 13, 8, -20, 13} is a multiset, in which -20 appears twice, 8 appears once, and 13 appears three times. Multisets are also called *bags*.

A *linked list* is a natural way of storing multisets in programs. The above example multiset can be stored as the following linked list:




Each member of the multiset is stored as a node in the list, defined in the program as:

```
struct Node
{
     int value;
     Node *next;
};
```

Observe that each node contains an integer value, and a pointer to the next node in the list. The head of the list can now be declared as:

```
Node *head;
```

## Program:

Write a C++ program that manages a multiset of integers. This multiset should be initially empty, i.e. it should have no elements to start with. The program should be capable of adding new integer values to the multiset, and counting the number of occurrences in the multiset of a given integer value. The user must be given a choice of these options:

1. Add a new value
2. Count occurrences of a value
3. Quit

The result of the user's selection must be displayed, and the menu repeated, until 3 for Quit is chosen.

**Notes:**

- A typical sample run of your program should look like (output `blue`, input `red`) :

```
1. Add a new value
2. Count occurrences of a value
3. Quit
Your choice: 1
New value: 13

1. Add a new value
2. Count occurrences of a value
3. Quit
Your choice: 1
New value: -20

1. Add a new value
2. Count occurrences of a value
3. Quit
Your choice: 2
Value to count: 13
13 appears 1 time(s)

1. Add a new value
2. Count occurrences of a value
3. Quit
Your choice: 2
Value to count: 8
8 appears 0 time(s)

1. Add a new value
2. Count occurrences of a value
3. Quit
Your choice: 1
New value: 13

1. Add a new value
2. Count occurrences of a value
3. Quit
Your choice: 2
Value to count: 13
13 appears 2 time(s)

1. Add a new value
2. Count occurrences of a value
3. Quit
Your choice: 3
```

- To add a new value in the multiset, you will need to dynamically create a new node in the list. This can be done by the **`new`** operator, described on textbook Page 512. The **`value`** field in this newly created node will need to be set to the new integer value, and the **`next`** field will need to be set to place this new node appropriately in the list. It is simplest to place this new node at the front of the list, which will involve updating the **`head`** variable.

- The name of your source file ***must*** be: `pa9.cpp`

## Grading Rubric:

Your program submission will be graded according to the following rubric:

| | |
|---|---|
| The name of the submitted program source file is `pa9.cpp` (*not* `pa9.c` or `Pa9.cpp` or `PA9.cpp` etc.) | 1 |
| The first 2 lines (student's name, student ID) of the required comments at the top of the program are included | 1 |
| A good indentation scheme is used throughout the program | 1 |
| The program compiles and links with no errors or warnings | 2 |
| The program works correctly for all inputs | 10 |
| **TOTAL** | **15** |

## Extra Credit:

Extra credit of up to **5 points** will be awarded if you implement a new menu option for removing one occurrence of a given value from the multiset, if that value exists. Do not remove more than one occurrence at a time. If the given value does not exist in the multiset, give an appropriate message.

Removal of a node from a linked list should be done carefully, by appropriately modifying the **`next`** pointer of the removed node's predecessor (or **`head`**, if the removed node was the first in the list).

After removing a node from the list, it is good practice to free the dynamic memory occupied by that node by the **`delete`** operator, described on textbook Page 517.