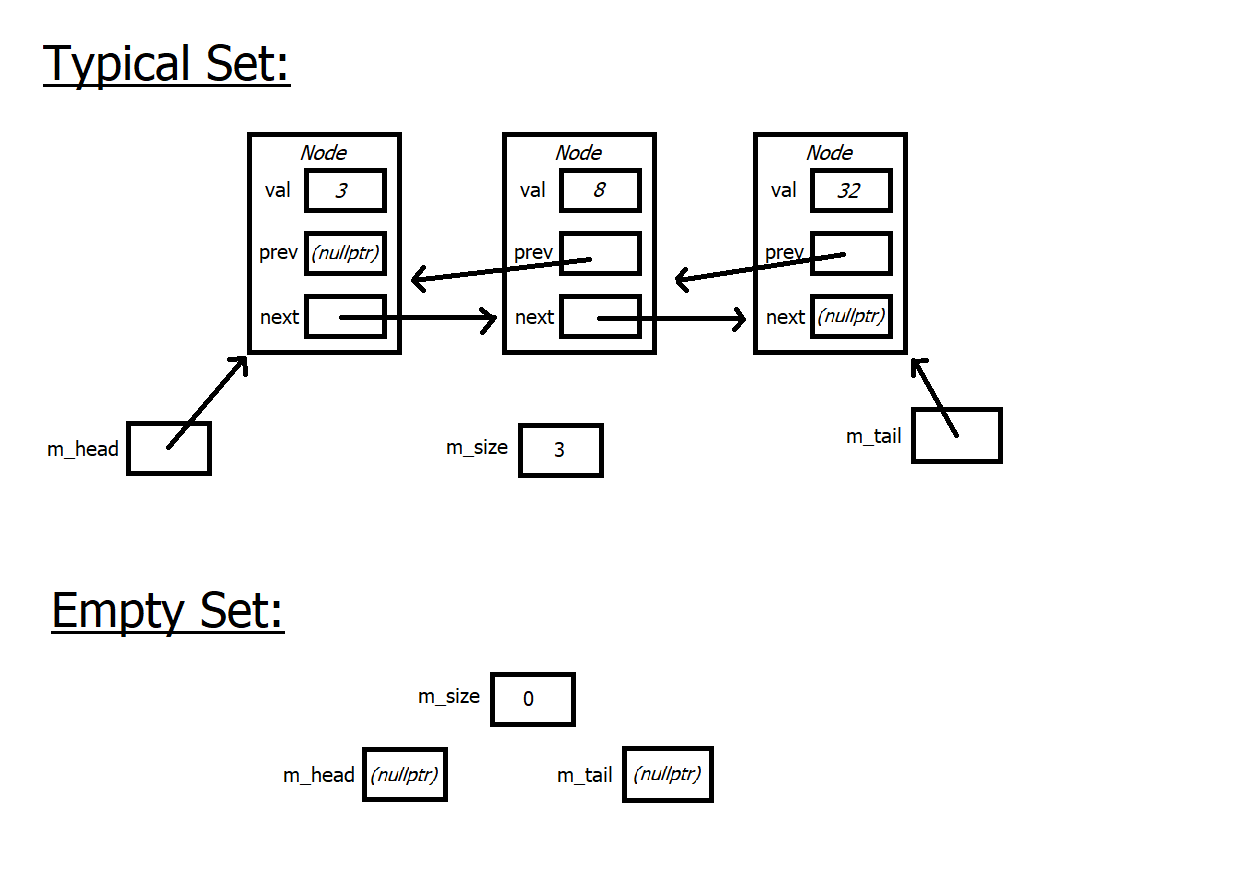
Harrison Cassar

ID: 505114980

Discussion 1C

TA: Hackett, T.R.

Report for CS32 Project 2: Double Trouble

1. Brief description of the design of my doubly-linked list implementation:
   * My doubly-linked list consists of a series of connected "Node" objects that each consist of a "previous" and "next" Node pointer that essentially link themselves to the Node both in front of and behind them. The first Node, as it has no Node before it to connect to, has its "previous" pointer set to the null pointer, and, similarly, the last Node, since it has no next Node to connect to, has its "next" pointer set to the null pointer. Since this is a regular doubly-linked list (not a circular one), the Set class contains both a "head" and "tail" pointer that indicates where in memory the beginning and end of the linked list exists. The Node struct itself (which is located in the private section of the Set class) additionally contains a "value" variable that holds the actual value of that object in the set. The implementation of the linked list ensures that all the elements in the set is sorted. This ensures that the “get()” function of Set has a much easier time determining the correct target position while not sacrificing much difference in performance.
   * Here is a helpful diagram of a typical and empty set:  
     
2. Psuedocode for non-trivial algorithms:
   * subtract():  
     Result Set gets the elements in Set1  
     repeating for Set2’s size:  
      receive value of element of Set2  
      call erase on result with that value
   * unite():  
     Result Set gets the elements in Set1  
     repeating for Set2’s size:  
      receive value of element of Set2  
      call insert on result with that value
   * erase():  
     check for empty list  
     if erasing first item in list:  
      delete Node and update all pointers and size  
      if list was only one-node long, tail should be nullptr  
     traverse linked list:  
      if found Node with the desired value:  
      break early  
     if we did find a matching value:  
      delete Node and update next/prev pointers and size  
      if Node was last in the list, tail is updated
   * insert():  
     check for list already containing the value (no duplicates allowed)  
     traverse linked list:  
      if found the sorted position for the new value :  
      break early  
     if sorted position is at beginning:  
      check if adding to an empty list  
      create new Node and update all pointers and size properly  
     if sorted position is at end:  
      create new Node and update all pointers and size properly  
     otherwise  
      create new Node and update all pointers and size properly