Name: [redacted] (as it would appear on official course roster)	
Umail address: [redacted] @umail.ucsb.edu	section
Optional: name you wish to be called if different from name above.	
Optional: name of "homework buddy" (leaving this blank signifies "I worked alone")	

1

h04

CS24 W19

h04: Chapter 5, section 5.1 -5.4

ready?	assigned	due	points
true	Wed 01/30 02:00PM	Mon 02/04 09:00AM	50

You may collaborate on this homework with AT MOST one person, an optional "homework buddy".

MAY ONLY BE TURNED IN IN THE LECTURE/LAB LISTED ABOVE AS THE DUE DATE, OR IF APPLICABLE, SUBMITTED ON GRADESCOPE. There is NO MAKEUP for missed assignments; in place of that, we drop the lowest scores (if you have zeros, those are the lowest scores.)

Please:

- **No Staples.**
- **No Paperclips.**
- **No folded down corners.**

Complete your reading of Chapter 5, section 5.1 -5.4 (If you don't have a copy of the textbook yet, there is one on reserve at the library under "COMP000-STAFF - Permanent Reserve").

1. (2 pts) Describe a problem that can occur if you dereference a null pointer. A segmentation fault will occur due to attempt to access an inaccessible address

2. (8 pts) Suppose you want to use a linked list where the items stored in the list are strings from the standard library string class, how would you change the node1.h header file on page 257 ?

Change "typedef double value_type;" to "typedef string value_type;"

3. (10 pts) Write the implementation of a non-member function `Node* deleteSecond(Node* head_ptr)`, where Node is a class defined on page 257. The function takes as input a pointer to the head of a linked list consisting of numbers. The function should remove the second item in the list. If the list had only one item, the function should delete that item. If the list was empty, then let the list remain empty. In all cases return the new head of the list

```
Node* deleteSecond (Node* head_ptr){
    if (head_ptr == NULL){
        return head_ptr;
    }
    else if (head_ptr->link() == NULL){
        list_head_remove (head_ptr);
        head_ptr = NULL;
    }
    else {
        Node* p = (head_ptr->link())->link();
        list_remove (head_ptr);
        head_ptr->set_link (p);
    }
    return head_ptr; }
```

4. (30 pts) Read the definition of the bag class on page 262 that uses a linked-list to store the items in the bag. Then explain each of the following:

2

h04

CS24 W19

i. Why does the class overload the copy constructor?

The class overloads the copy constructor in order to create a deep copy of the source object by creating new nodes with values replicated from the source

ii. Why does the class overload the assignment operator?

The assignment operator needs to be overloaded to manually assign each private variable, delete existing memory on heap and assign new replicated memory

iii. Why is it a good idea to typedef node::value_type as value_type in the bag class definition?

It is a good idea because setting bag's value_type to node's value_type simplifies the usage of the bag class so that the user is not required to understand the concept of nodes within the underlying data structure of bag