COEN 79 Homework 4

1. Write a function to remove duplicates from a forward linked list.

```
void list_remove_dups(node *head_ptr) {
     for (node *it1 = head_ptr; it1 != nullptr; it1 = it1->link()){
          node *it2 = it1;
          value_type data = it1->data();
          while (next != nullptr) {
               node *next = it2->link();
               if (next->data == data) {
                    list_remove(it2);
               }
               it2 = it2->link();
          }
     }
}
```

2. Implement the list_copy function using only the node class.

```
void list_copy(const node *src, node *&head, node *&tail) {
     assert(src != nullptr);
     node *it = src->link();          // source list iterator
     node *cur = head;                // new list iterator
     head->set_data(src->data());
     tail = head;
     while (it != nullptr) {
          node *n = new node(it->data());
          cur->set_link(n);
          tail = n;
          it = it->link();
          cur = cur->link();
     }
}
```

3. Why are the linked list toolkit functions not node member functions?
   - They are not member functions because they do not have direct access to node class private member

4. In `size_t list_length`, why is `cursor` defined as a `const` variable?

- The function makes no changes to the list's nodes thus it takes a pointer to a constant node (head). In the for loop, cursor is set to head thus it must also be type const.

5. What is the output of the code?
   - Static variables share their values between all instances of the class.
   - When default constructor is called, ctor += 1
   - When copy constructor is called, cc += 1
   - When destructor is called, dest += 1
   - When the assignment operator is called, asop += 1
   - stuFunc calls the copy constructor and destructor
   - Line 50 uses copy constructor not assignment operator because mySt2 is not previously initialized

Output:

```
ctor = 0  cc = 0  dest = 0  asop = 0
ctor = 1  cc = 0  dest = 0  asop = 0
ctor = 1  cc = 1  dest = 1  asop = 0
ctor = 1  cc = 3  dest = 2  asop = 0
```

6. Write the implementation of a function that detects loops within the list.

```
node* list_loop_detect(const node* head) {
    node* s = head;
    node* f = head;
    while (s != f) {
        if (f == nullptr) return nullptr; // no loop
        s = s->link();
        f = f->link()->link();
    }
    s = head;
    while (s != f) {
        s = s->link();
        f = f->link();
    }
    return s;
}
```

7. What data structure would you use:
   a. to implement back functionality in a web browser - if there is no forward functionality then only a singly linked list would be necessary; "previous" pointer that stores information about the most recently visited page.

b. To implement printer spooler so that jobs can be printed in the order of their arrival - a queue which uses FIFO (first in first out) principle, implemented using a dynamic array; CPU potentially brings "chunks" of jobs since array data is continuous allowing for better performance

8. Why does our node class have two versions of the link member function?

- (B) One is to use with a const pointer, the other with a regular pointer.

9. Discuss the two major effects of storing elements of a data structure in contiguous memory locations.

- As mentioned in 7b, the CPU brings chunks of data from RAM into the cache (cache access is much faster than RAM access speeds), since arrays store data continuously, there is a high probability that sequential elements in an array will be brought to the cache at a single time leading to better performance
- Additionally, arrays have the random access operator (`[]`) or subscript operator that allows for any element within the array to be accessed via its index, this is done in O(1) time. With non contiguous memory locations, this is not possible and accessing a particular element has a worst-case time complexity of O(n)

10. Write the implementation of `bag::operator =`.

```
void bag::operator =(const bag &source) {
     // check self-assignment
     if (this == &source) return;

     // free all current nodes
     node *cur = head;
     node *it = head->link();
     while (it != nullptr) {
          delete cur;
          cur = it;
          it = it->link();
     }
     many_nodes = 0;

     // copy source list (using imp. from Q2)
     node *tail;
     list_copy(source.head, head, tail);
     many_nodes = source.many_nodes;
{
```