COEN 79

Object-Oriented Programming and Advanced Data Structures

Assignment #4 – linked list
Name: Yen-Jung Lu **Date:** 05/07/2023

- Points per question: 4

---

1. Write a function to remove duplicates from a *forward linked list. Do not use a temporary buffer. The ordering of items is not important.*

```
1.  void list_remove_dups(node* head_ptr)
2.  // Postcondition: All the duplicates are removed from the linked list
3.  // Example: If the list contains 1,1,1,2, after running this function the list
4.  // contains 1,2
```

```
if (head_ptr == nullptr || head_ptr -> next == nullptr)
{   return;
}
node * prev_ptr = head_ptr;
node * curr_ptr = head_ptr -> next;
while ( curr_ptr != nullptr )
  {  node * search_ptr = head_ptr;
    while ( search_ptr != curr_ptr)
      if ( search_ptr -> data == curr_ptr -> data)
        {
          prev_ptr -> next = curr_ptr -> next;
           delete  curr_ptr;
           curr_ptr = prev_ptr -> next;
        }
    }
```

2. The `node` class is defined as follows:

```
1.  class node {
2.      public: // TYPEDEF
3.          typedef double value_type;
4.
5.      // CONSTRUCTOR
6.      node(const value_type& init_data = value_type(), node* init_link = NULL) {
7.              data_field = init_data;
8.              link_field = init_link;  }
9.
10.     // Member functions to set the data and link fields:
11.     void set_data(const value_type& new_data) { data_field = new_data; }
12.     void set_link(node* new_link) { link_field = new_link;  }
13.
14.     // Constant member function to retrieve the current data:
15.     value_type data() const { return data_field; }
16.
17.     // Two slightly different member functions to retreive the current link:
18.     const node* link() const { return link_field; }
19.     node* link() { return link_field; }
20.
21.     private:
22.         value_type data_field;
23.         node* link_field;
24. };
```

Implement the following function. (Note: *No toolkit function is available*. Only the node class is available.)

```
1.  void list_copy (const node* source_ptr, node*& head_ptr, node*& tail_ptr)
2.  // Precondition: source_ptr is the head pointer of a linked list.
3.  // Postcondition: head_ptr and tail_ptr are the head and tail pointers for a new list that
4.  // contains the same items as the list pointed to by source_ptr
5.  {
        if (source_ptr == NULL)
        {   head_ptr = NULL;
            tail_ptr = NULL;
        }
        else {
            head_ptr = new node (source_ptr -> data());
            tail_ptr = head_ptr;


        }
```

3. Please justify why the linked list toolkit functions are not member functions of the `node` class?

It is not member functions of the node class because they operate on entire linked list.

4. In the following function, why `cursor` has been declared as a `const` variable? What happens if you change it to a non-const variable?

```
1.  size_t list_length (const node* head_ptr)
2.  // Precondition: head_ptr is the head pointer of a linked list.
3.  // Postcondition: The value returned is the number of nodes in the // linked list.
4.  {
5.          const node* cursor;
6.          size_t answer;
7.          answer = 0;
8.          for (cursor = head_ptr; cursor != NULL; cursor = cursor - > link())
9.              ++answer;
10.         return answer;
11. }
```

The cursor has been declared as const because it is used to traverse the linked list. If we change it to non-const, I think it will still run, but this could get bugs if linked_list is modified accidently.

5. What is the output of this code? Please explain your answer.

```
1.  #include < iostream >
2.  using namespace std;
3.
4.  class student {
5.  public:
6.      static int ctor;
7.      static int cc;
8.      static int dest;
9.      static int asop;
10.     student() {
11.         name = "scu";
12.         ++ctor;
13.     };
14.     student(const student & source) {
15.         this -> name = source.name;
16.         this -> id = source.id;
17.         ++cc;
18.     };
19.     ~student() {
20.         ++dest;
21.     };
22.     void operator = (const student & source) {
23.         this -> name = source.name;
24.         this -> id = source.id;
25.         ++asop;
26.     }
27. private:
28.     string name;
29.     int id;
30. };
31.
32. int student::ctor= 0;
33. int student::cc = 0;
34. int student::dest = 0;
35. int student::asop = 0;

36. student stuFunc(student input) {
37.     return input;
38. }
39.
40. int main(int argc, const char * argv[]) {
41.
42.     std::cout << "ctor = " << student::ctor << "  cc = " << student::cc << "  dest = " <<
           student::dest << "  asop = " << student::asop << endl;
43.
44.     student mySt1;
45.     std::cout << "ctor = " << student::ctor << "  cc = " << student::cc << "  dest = " <<
                                      student::dest << "  asop = " << student::asop << endl;
46.
47.     stuFunc(mySt1);
48.     std::cout << "ctor = " << student::ctor << "  cc = " << student::cc << "  dest = " <<
                                    student::dest << "  asop = " << student::asop << endl;
49.
50.     student mySt2 = stuFunc(mySt1);
51.     std::cout << "ctor = " << student::ctor << "  cc = " << student::cc << "  dest = " <<
                                    student::dest << "  asop = " << student::asop << endl;
```

```
52.
53.     return 0;
54. }
```

The out put;

ctor = 0 cc = 0 dest = 0 asop = 0

ctor = 1 cc = 0 dest = 0 asop = 0 → new object of "student" class is created in line 48

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

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

In line 50, the copy constructor make a copy of the object returned by stuFunc(myst1) ⇒ incrementing cc to 1

6. We are interested in implemented the back and forward arrow functionality in a mobile app. When back arrow is touched, the previous screen is shown, and when forward arrow is pressed (if applicable), then the forward screen is shown. Please justify what type of data structure you would use.

A stack data structure can be used to do that. Each screen visited can be pushed onto the stack,

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

   A. One is public, the other is private.
   B. One is to use with a const pointer, the other with a regular pointer.
   C. One returns the forward link, the other returns the backward link.
   D. One returns the data, the other returns a pointer to the next node.

8. The bag class is defined as follows:

```
1.  class bag {
2.      public:
3.      // TYPEDEFS
4.      typedef std::size_t size_type;
5.      typedef node::value_type value_type;
6.
7.      // CONSTRUCTORS and DESTRUCTOR
8.      bag();
9.      bag(const bag & source);
10.     ~bag();
11.
12.     // MODIFICATION MEMBER FUNCTIONS
13.     size_type erase(const value_type & target);
14.     bool erase_one(const value_type & target);
15.     void insert(const value_type & entry);
16.     void operator += (const bag & addend);
17.     void operator = (const bag & source);
18.
19.     // CONSTANT MEMBER FUNCTIONS
20.     size_type size() const {  return many_nodes;  }
21.     size_type count(const value_type & target) const;
22.     value_type grab() const;
23.
24.     private:
25.         node* head_ptr; // List head pointer
26.         size_type many_nodes; // Number of nodes on the list
27. };
```

Considering the `node` class definition given in Question 2, implement the assignment operator for the `bag` class. (Note: *No toolkit function is available*.)

```
1. void bag::operator = (const bag & source)
2. // Library facilities used: node1.h
3. {
```

```
if ( this == & source )
 return;

node * source_ptr = source.head_ptr;
node * tail_ptr = NULL;
node * new_list_ptr;

while (head_ptr != NULL)
{ new_list_ptr = head_ptr;
  head_ptr = head_ptr -> link();
  delete new_list_ptr;
 }
many_nodes = 0;

while (source_ptr != NULL)
{  insert (source_ptr -> data());
   source_ptr = source_ptr -> link();
  }
}
```