

CSCI-1200 Data Structures

Test 2 — Practice Problems

Note: This packet contains selected practice problems from Test 2 from three previous years. Your test will contain approximately one third to one half as many problems (totalling ~ 100 pts).

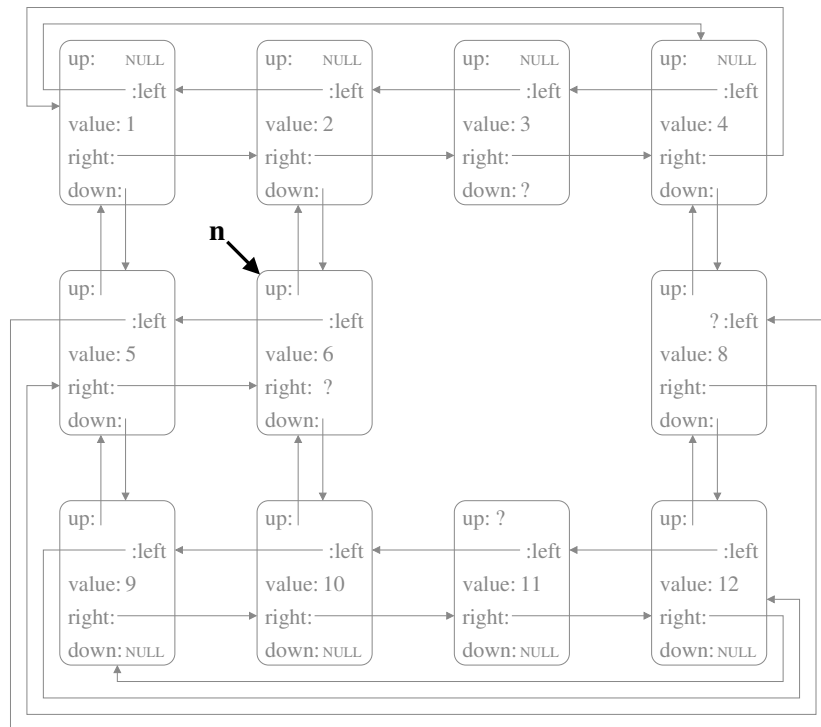
1 **Linked Tube Repair** [/ 33]

Alyssa P. Hacker is working on a modified linked list that is both two-dimensional and circular. A small sample with *height*=3 and *circumference*=4 is shown below. Each templated `Node` has pointers to its 4 neighbors. The top and bottom edges of the tube structure have `NULL` pointers. But the left and right edges wrap around, like a circularly linked list. This cylindrical *tube* structure may have any number of nodes for its height and its circumference.

```
template <class T>
class Node {
public:
    // REPRESENTATION
    T value;
    Node<T> *up;
    Node<T> *down;
    Node<T> *left;
    Node<T> *right;
};
```

1.1 Tube repair Diagram [/ 4]

First Alyssa wants to tackle the challenge of repairing a hole in the structure. Assume a single **Node** is missing from the structure, and we have a pointer **n** to the **Node** immediately to the left of the hole. Modify the diagram below to show all of the necessary edits for a call to `repair(n,7)`;



1.2 Thinking about Tube repair Complexity [3 / 3]

The **repair** function should have constant running time in most cases. Describe an example structure with a single missing **Node** that can be repaired, but *not* in constant time. Write 2-3 concise and well-written sentences. *You may want to complete the implementation on the next page before answering.*

--

1.3 Tube repair Implementation [/ 13]

Now, implement `repair`, which takes 2 arguments: a pointer to the `Node` immediately to the left of the hole and the value to be stored in the hole. You may assume a single `Node` is missing from the structure.

sample solution: 26 line(s) of code

1.4 Non-Iterative `destroy_tube` Implementation [/ 13]

Now write `destroy_tube` (and any necessary helper functions) to clean up the heap memory associated with this structure. The function should take a single argument, a pointer to any `Node` in the structure. You may assume the structure has no holes or other errors. You cannot use a `for` or `while` loop.

sample solution: 17 line(s) of code

2 Rehashing the Vec Assignment Operator [/ 15]

Complete the Vec assignment operator implementation below, while minimizing wasted heap memory. Assume the allocator is most efficient when all heap allocations are powers of two (1, 2, 4, 8, 16, etc.)

```
1  template <class T>
2  Vec<T>& Vec<T>::operator=(const Vec<T>& v) {
3      if (this != &v) {
4          delete [redacted] ;
5          m_size = [redacted] ;
6          m_alloc = [redacted] ;
7          m_data = [redacted] ;
8          for (int i = 0; i < [redacted] ; ++i) {
9              m_data[i] = [redacted] ;
10         }
11     }
12     return *this;
13 }
```

Add code below to perform a simple test of the assignment operator:

```
Vec<double> v; v.push_back(3.14159); v.push_back(6.02); v.push_back(2.71828);
```

Is line 12 necessary? Continue your testing code above with a test that would break if line 12 was omitted.

What is the purpose of line 3? Write code for a test that would break if lines 3 and 10 were omitted.

3 Essay Revision: Embellishment [/ 14]

Write a function `embellish` that modifies its single argument, `sentence` (an STL list of STL strings), adding the word “very” in front of “pretty” and adding “with a wet nose” after “grey puppy”. For example:

the pretty kitty sat next to a grey puppy in a pretty garden

Should become:

the very pretty kitty sat next to a grey puppy with a wet nose in a very pretty garden

sample solution: 20 line(s) of code

If there are w words in the input sentence, what is the worst case Big O Notation for this function? If we switched each STL `list` to STL `vector` in the above function, what is the Big O Notation?

STL `list`:

STL `vector`:

4 Essay Revision: Redundant Phrases [/ 15]

Complete `redundant`, which takes a sentence and 2 phrases and replaces all occurrences of the first phrase with the second, shorter phrase. For example “pouring down rain” is replaced with “pouring rain”:

it is pouring down rain so take an umbrella → it is pouring rain so take an umbrella

Or we can just eliminate the word “that” (the replacement phrase is empty):

I knew that there would be late nights when I decided that CS was the career for me

→ I knew there would be late nights when I decided CS was the career for me

```
typedef std::list<std::string> words;
```

```
void redundant(  sentence,  phrase,  replace) {
```

sample solution: 19 line(s) of code

```
}
```

5 Don't Ignore Compilation Warnings! [/ 15]

Write a useful but buggy segment of code (or function) that will compile with no errors but will produce the indicated compilation warning. Put a star ★ next to the line of code that will trigger the warning. Write a concise and well-written sentence describing the intended vs. actual (buggy) behavior of the code.

warning: comparison of integers of different signs: 'int' and 'unsigned int'

warning: control reaches / may reach end of non-void function

warning: variable is uninitialized when used here / in this function

warning: returning reference to local temporary object / reference to stack memory associated with a local variable returned

warning: expression result unused / expression has no effect

warning: unused variable / unused parameter

6 Cyber Insecurity [/ 5]

Ben Bitdiddle wrote the following code fragment to manage his personal information.

```
1  std::ifstream istr("my_information.txt");
2  std::string s;
3  std::vector<std::string> data;
4  while (istr >> s) { data.push_back(s); }
5  std::vector<std::string>::iterator password = data.begin()+4;
6  data.push_back("credit_card:");
7  data.push_back("1234-5678-8765-4321");
8  data[4] = "qwerty";
9  std::cout << "my password is: " << *password << std::endl;
```

my_information.txt

name: Ben Bitdiddle
password: pa\$\$word
SSN: 123-45-6789

Write “True” in the box next to each *true* statement. Leave the boxes next to the *false* statements empty.

☐

Lines 2 & 3 will produce an “uninitialized read” error when run under `gdb` or `lldb`.

☐

Line 5 is not a valid way to initialize an iterator.

☐

Ben’s credit card information is not saved back to the file.

☐

This program might behave differently if re-run on this computer or another computer.

☐

A memory debugger might detect an “unaddressable access of freed memory” error on Line 9.

☐

If we move lines 6 & 7 after line 9, this code fragment will run without memory errors.

☐

This code contains memory leaks that can be detected by Dr. Memory or Valgrind.

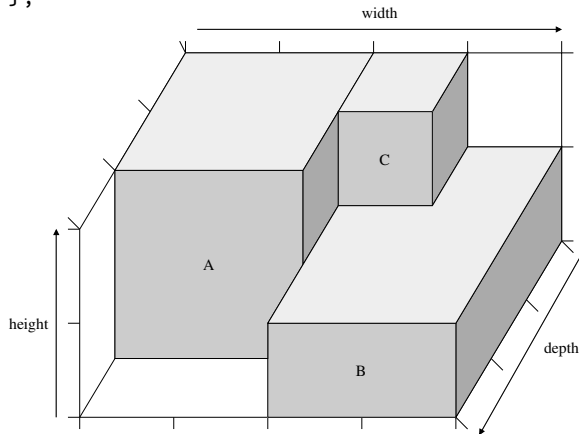
☐

These password choices disqualify Ben from any job in computer security.

7 Boxy Storage Solutions [/ 25]

Eva Lu Ator is working on her capstone project to manage physical storage facilities. She's mapped out the overall design and started implementation of the two classes.

```
class Box {
public:
    Box(int w, int d, int h) :
        width(w), depth(d), height(h) {}
    int width;
    int depth;
    int height;
};
```



```
Storage storage(4,3,2);
assert (storage.available_space() == 24);

Box *a = new Box(2,2,2);
assert (storage.add(a,0,0,0));
Box *b = new Box(3,2,1);
assert (!storage.add(b,2,0,0));
delete b;
Box *b_rotated = new Box(2,3,1);
assert (storage.add(b_rotated,2,0,0));
Box *c = new Box(1,1,1);
assert (storage.add(c,2,0,1));

assert (storage.available_space() == 9);
```

```
class Storage {
public:
    Storage(int w, int d, int h);

    // FILL IN FOR PART 1

    bool add(Box *b, int w, int d, int h);
    int available_space();

private:
    void remove(Box *b, int w, int d, int h);
    Box ****data;
    int width;
    int depth;
    int height;
};

bool Storage::add (Box *b, int w, int d, int h) {
    for (int i = w; i < w+b->width; i++) {
        if (i >= width) return false;
        for (int j = d; j < d+b->depth; j++) {
            if (j >= depth) return false;
            for (int k = h; k < h+b->height; k++) {
                if (k >= height) return false;
                if (data[i][j][k] != NULL) return false;
            }
        }
    }
    for (int i = w; i < w+b->width; i++) {
        for (int j = d; j < d+b->depth; j++) {
            for (int k = h; k < h+b->height; k++) {
                data[i][j][k] = b;
            }
        }
    }
    return true;
}
```

7.1 Missing functions from Storage Class Declaration [/ 5]

Her friend Ben Bitdiddle doesn't remember much from Data Structures, but he reminds her that classes with dynamically-allocated memory need a few key functions. Fill in the missing prototypes for PART 1.

7.2 Storage Destructor [/ 20]

Eva explains to Ben that the private `remove` member function will be useful in implementing the destructor. First write the `remove` member function:

sample solution: 10 line(s) of code

Now write the `Storage` class destructor:

sample solution: 14 line(s) of code

8 Transpose Linked Grid [/ 27]

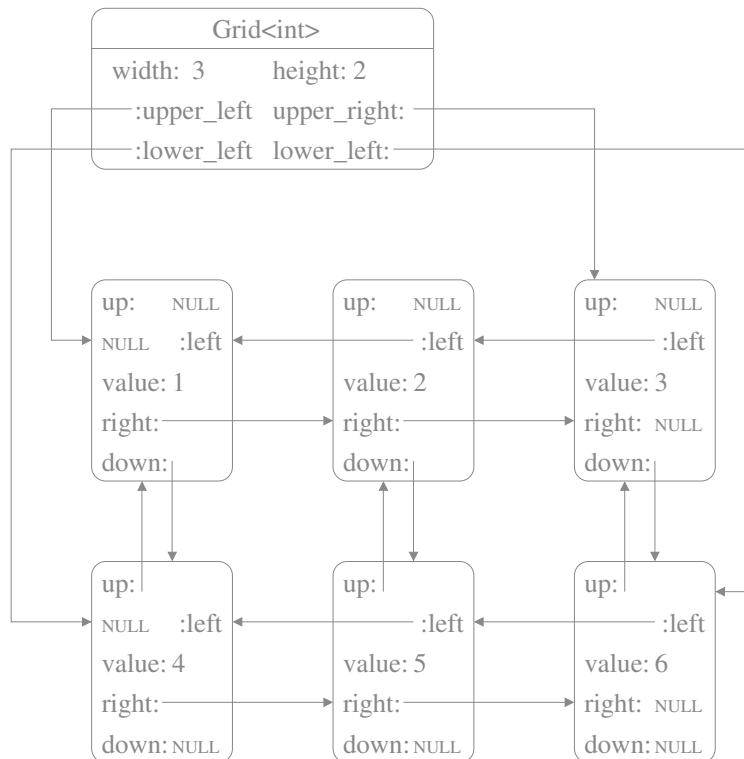
Louis B. Reasoner is working on a new member function for our Homework 5 Linked Grid named `transpose`. This function should mirror or flip the elements along the diagonal. Here's a sample grid with integer data and how it prints before and after a call to `transpose`:

```
grid.print();           1  2  3  4
std::cout << std::endl; 8  7  6  5
grid.transpose();       9 10 11 12
grid.print();           1  8  9
                        2  7 10
                        3  6 11
                        4  5 12
```

```
template <class T>
class Node {
public:
    // REPRESENTATION
    T value;
    Node<T> *up;
    Node<T> *down;
    Node<T> *left;
    Node<T> *right;
};
```

8.1 Diagram [/ 7]

First neatly modify the diagram of this smaller grid below to show all of the necessary edits that must be performed by a call to `transpose()`.



8.2 Complexity Analysis [/ 5]

What is the Big 'O' Notation for the running time of the `transpose()` member function? Assume the grid width is w and the height is h . Write 1-2 concise and well-written sentences justifying your answer. *You probably want to complete the implementation on the next page before answering.*

8.3 Implementation [/ 15]

Louis has suggested that we first implement a helper non-member function named `swap`, which will make the implementation of `transpose` more concise.

sample solution: 5 line(s) of code

Now implement `transpose`, as it would appear outside of the `Grid` class declaration.

sample solution: 16 line(s) of code

9 Organizing Words [/ 30]

Alyssa P. Hacker is working on a program to clean up a dataset of words. The task is to write a function named `organize_words` that takes in an STL `vector` of STL `lists` of words (STL `strings`). The function should organize the words into groups by word length, and ensure that the words are sorted within each group. Many or most of the words will already be in the right place. That is, they will already be in the slot of the vector that matches the length of the word. And the neighboring words in each slot/list will already be mostly alphabetized.

For example, given the data shown on the left, your implementation should move the four misplaced words to produce the data shown on the right.

0		0	
1	diamond	1	
2		2	
3	gem malachite	3	gem
4	jade opal rock ruby	4	jade opal rock ruby talc
5	geode pearl talc stone topaz	5	geode pearl stone topaz
6	garnet quartz gypsum	6	garnet gypsum quartz
7	amethyst azurite emerald	7	azurite diamond emerald
8	fluorite sapphire	8	amethyst fluorite sapphire
9		9	malachite

To make the problem a little more “fun”, you are *NOT ALLOWED* to use:

- the STL `vector` subscript/indexing operator, `[]`, or `.at()`,
- the STL `sort` function, or
- any of the `push` or `pop` functions on `vector` or `list`.

You may assume that the initial vector has at least as many slots as the longest word in the structure.

9.1 Complexity Analysis - Big 'O' Notation [/ 6]

Once you’ve finished your implementation on the next pages, analyze the running time of your solution. Assume there are w total words in the whole structure, v slots in the `vector`, a maximum of m words per `list`, and x words are misplaced and need to be moved. Write 2-3 concise and well-written sentences justifying your answer.

9.2 Helper Function Implementation [/ 12]

Alyssa suggests writing a helper function named `place` that will place a word in the correct location in the structure. Work within the provided framework below. Do not add any additional `for` or `while` loops.

```
void place(  ) {  
  
  
sample solution: 2 line(s) of code  
  
while (  ) {  
  
  
sample solution: 3 line(s) of code  
  
while (  ) {  
  
  
sample solution: 5 line(s) of code  
  
}  
  
  
sample solution: 5 line(s) of code  
  
}  
}
```

9.3 Organize Implementation [/ 12]

And now write the **organize** function, which calls the **place** function. Again, work within the provided framework below and do not add any additional **for** or **while** loops.

```
void organize_words(  ) {  
  


sample solution: 2 line(s) of code

  
while (  ) {  
  


sample solution: 2 line(s) of code

  
while (  ) {  
  


sample solution: 8 line(s) of code

  
}  


sample solution: 2 line(s) of code

  
}  
}
```

10 Merge-Spiration: Recursive Interval Computation [/ 15]

Ben Bitdiddle was inspired by the recursive merge sort example from Data Structures lecture and proposes it as a guide to compute the smallest interval that contains a collection of floating point numbers (e.g., the minimum and maximum). Implement Ben's idea, a recursive function named `compute_interval` that takes in an STL vector of floats and returns an `Interval` object.

For example: 6.2 4.3 10.4 2.5 8.4 1.5 3.7 \rightarrow [1.5, 10.4]

```
class Interval {
public:
    Interval(float i, float j)
        : min(i), max(j) {}
    float min;
    float max;
};
```

sample solution: 12 line(s) of code

Without resorting to personal insults, explain in two or three concise and well-written sentences why Ben's idea isn't going to result in significant performance improvements. Be technical.

11 How many DS students to change a lightbulb? [/ 38]

In this problem you will complete the implementation of two new classes named `Bulb` and `Lamp`. We begin with an example of how these classes are used.

First, we create a new lamp that will hold 3 bulbs and make a note of the manufacturer's recommended bulb: a 60 watt bulb with an estimated lifetime of 300 hours from Phillips. Note that initially this lamp has no bulbs installed. We install one of manufacturer's recommended bulbs and use the lamp (turn it "on") for a total of 50 hours.

```
Lamp floorlamp(Bulb(60,300,"Phillips"),3);
bool success;
success = floorlamp.install(); assert(success);
floorlamp.On(50);
assert (floorlamp.getTotalWattage() == 60);
```

Next, we attempt to install 3 bulbs, another of the manufacturer's recommended bulbs, and then two other brands of bulbs. The installation of the 3rd bulb made by Sylvania fails because there are no available sockets slots in the lamp and no bulbs are burnt out and need replacement.

```
success = floorlamp.install(); assert(success);
success = floorlamp.install(Bulb(40,120,"GE")); assert(success);
success = floorlamp.install(Bulb(120,500,"Sylvania")); assert(!success);
```

We then use the lamp for another 100 hours. Once the wattage drops (due to a burnt out bulb), we again try to install the Sylvania bulb and it is successful.

```
floorlamp.On(100);
assert (floorlamp.getTotalWattage() == 160);
floorlamp.On(50);
assert (floorlamp.getTotalWattage() == 120);
success = floorlamp.install(Bulb(120,500,"Sylvania")); assert(success);
assert (floorlamp.getTotalWattage() == 240);
```

Finally, we create a duplicate lamp. Note that when we do this, we match the bulbs currently installed in the original lamp, but the bulbs installed in the new lamp are brand new (and unused).

```
Lamp another(floorlamp);
assert (floorlamp.getTotalWattage() == another.getTotalWattage());
for (int i = 0; i < 10; i++) {
    floorlamp.On(50);
    another.On(50);
    std::cout << "compare " << floorlamp.getTotalWattage() << " "
               << another.getTotalWattage() << std::endl;
}
```

Which results in this output:

```
compare 240 240
compare 240 240
compare 180 240
compare 120 240
compare 120 240
compare 120 240
compare 120 120
compare 120 120
compare 120 120
compare 120 120
```

11.1 Bulb Class Declaration [/ 14]

The `Bulb` class is missing only one function. *You will need to read the rest of the problem to determine what's missing.* Fill in the missing function – implement the function right here, within the class declaration.

```

class Bulb {
public:
    // constructors
    Bulb(int w, int l, const std::string &b) :
        wattage(w),lifetime(l),hours_used(0),brand(b) {}

```

sample solution: 2 line(s) of code

```

    // accessors
    int getWattage() const { return wattage; }
    bool burntOut() const { return hours_used > lifetime; }
    const std::string& getBrand() const { return brand; }
    // modifier
    void On(int h) { hours_used += h; }
private:
    // representation
    int wattage;
    int lifetime;
    int hours_used;
    std::string brand;
};

```

11.2 Lamp Class Declaration [/ 14]

The **Lamp** class has a few more missing pieces. *Read through the rest of the problem before attempting to fill this in.* Write the *prototypes* (not the implementation!) for the four missing functions. You will implement some of these missing functions later. Also, fill in the member variables for the **Lamp** representation. Important: You may not use STL **vector** on this problem.

```

class Lamp {
public:
    // constructors, assignment operator, destructor

```

sample solution: 4 line(s) of code

```

// accessor
int getTotalWattage() const;
// modifiers
bool install(const Bulb &b = Bulb(0,0,""));
void On(int h);
private:
    // representation

```

sample solution: 3 line(s) of code

```
};
```

Lamp Class Implementation

Here's the implementation of one of the key member functions of the `Lamp` class.

```

bool Lamp::install(const Bulb &b) {
    // first, let's figure out where to install the bulb
    int which = -1;
    for (int i = 0; i < max_bulbs; i++) {
        // check for an empty socket
        if (installed[i] == NULL) {
            which = i;
            break;
        }
        // or a socket that contains a burnt out bulb
        if (installed[i]->burntOut()) {
            which = i;
            delete installed[i];
            break;
        }
    }
    // return false if we cannot install this bulb
    if (which == -1) return false;
    if (b.getWattage() == 0) {
        // install the manufacturer's recommended bulb type
        installed[which] = new Bulb(recommended);
    } else {
        // install the specified bulb
        installed[which] = new Bulb(b);
    }
    return true;
}

```

On the last two pages of this problem you will implement three important functions for the `Lamp` class, as they would appear outside of the class declaration (in the `lamp.cpp` file) because their implementations are > 1 line of code.

11.3 Lamp Constructor [/ 9]

sample solution: 7 line(s) of code

11.4 Lamp Destructor [/ 5]

sample solution: 8 line(s) of code

11.5 Lamp Assignment Operator [/ 9]

sample solution: 10 line(s) of code

12 Singly Linked List Subsequence Sum [/ 18]

Write a recursive function named `FindSumStart` that takes the head Node of a singly-linked list storing positive numbers. The function should return a pointer to the Node that begins a subsequence of numbers that ends in the sum of that subsequence. For example, given this sequence: 5 1 4 2 3 9 6 7 the function should return a pointer to the Node storing 4, because $4 + 2 + 3 = 9$.

```
template <class T>
class Node {
public:
    Node(const T& v)
        : value(v),
          next(NULL) {}
    T value;
    Node* next;
};
```

sample solution: 15 line(s) of code

Assuming the sequence has n numbers, what is the order notation for the running time of your function?

13 Reverse Splice [/ 20]

Write a function named `reverse_splice` that takes in 3 arguments: an STL list named `data` and two iterators `i` and `j`. The function should reverse the order of the data between those iterators. For example, if `data` initially stores this sequence: 1 2 3 4 5 6 7 8 9 and `i` refers to 3 and `j` refers to 7, then after the call `reverse_splice(data,i,j)`, `data` will contain: 1 2 7 6 5 4 3 8 9 , `i` will refer to element 7, and `j` will refer to element 3. Your function should return true if the operation was successful, and false if the request is invalid. Note: Your function may only use a constant amount of additional memory.

sample solution: 21 line(s) of code

14 Doubly Linked Factorization [/ 17]

```
class Node {
public:
    Node(int v) :
        value(v),
        next(NULL),
        prev(NULL) {}
    int value;
    Node* next;
    Node* prev;
};
```

Write a recursive function named **Factor** that takes in two arguments, pointers to the **head** and **tail** Nodes of a doubly linked list. This function should look for a non-prime number in the linked list structure, break the Node into two Nodes storing two of its factors, and then return true. If all elements are prime the function returns false. For example, if we start with a 3 element list containing 35 30 28 and repeatedly call Factor:

```
PrintNodes(head);
while (Factor(head,tail)) { PrintNodes(head); }
```

This is the output:

```
35 30 28
5 7 30 28
5 7 2 15 28
5 7 2 3 5 28
5 7 2 3 5 2 14
5 7 2 3 5 2 2 7
5 7 2 3 5 2 2 7
```

You may write a helper function. You do not need to write the **PrintNodes** function.

sample solution: 21 line(s) of code