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,0,0,0));
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;
}
1.1 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.

1.2 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
2 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();
std::cout << std::endl;
grid.transpose();
grid.print();
```

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.1 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().

```
Grid<int>
  width: 3    height: 2
  :upper_left upper_right:
  :lower_left lower_left:
```

2.2 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.
2.3 Implementation

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

\textit{sample solution: 5 line(s) of code}

Now implement \texttt{transpose}, as it would appear outside of the \texttt{Grid} class declaration.

\textit{sample solution: 16 line(s) of code}
3 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.

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>diamond</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>gem</td>
<td>malachite</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>jade</td>
<td>opal</td>
<td>rock</td>
<td>ruby</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>geode</td>
<td>pearl</td>
<td>t alc</td>
<td>stone</td>
<td>topaz</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>garnet</td>
<td>quartz</td>
<td>gypsum</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>amethyst</td>
<td>azurite</td>
<td>emerald</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fluorite</td>
<td>sapphire</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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.

3.1 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.
3.2 Helper Function Implementation

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.

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

```c
void organize_words() {
    while ( ) {
        sample solution: 2 line(s) of code
    }
    while ( ) {
        sample solution: 2 line(s) of code
    }
    while ( ) {
        sample solution: 8 line(s) of code
    }
}
```
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 → [1.5, 10.4]

```cpp
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.
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.

\[
\text{Lamp } \text{floorlamp} = \text{Bulb}(60,300,\text{"Phillips"}),3; \\
\text{bool } \text{success} = \text{floorlamp} \text{.install()}; \text{ assert(success); } \\
\text{floorlamp} \text{.On}(50); \\
\text{assert } (\text{floorlamp} \text{.getTotalWattage()} == 60);
\]

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.

\[
\text{success} = \text{floorlamp} \text{.install()}; \text{ assert(success); } \\
\text{success} = \text{floorlamp} \text{.install(Bulb}(40,120,\text{"GE"})\text{); assert(success); } \\
\text{success} = \text{floorlamp} \text{.install(Bulb}(120,500,\text{"Sylvania"})\text{); assert(!success); }
\]

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.

\[
\text{floorlamp} \text{.On}(100); \\
\text{assert } (\text{floorlamp} \text{.getTotalWattage()} == 160); \\
\text{floorlamp} \text{.On}(50); \\
\text{assert } (\text{floorlamp} \text{.getTotalWattage()} == 120); \\
\text{success} = \text{floorlamp} \text{.install(Bulb}(120,500,\text{"Sylvania"})\text{); assert(success); } \\
\text{assert } (\text{floorlamp} \text{.getTotalWattage()} == 240);
\]

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).

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

Which results in this output:

\[
\text{compare } 240 240 \\
\text{compare } 240 240 \\
\text{compare } 180 240 \\
\text{compare } 120 240 \\
\text{compare } 120 240 \\
\text{compare } 120 240 \\
\text{compare } 120 120 \\
\text{compare } 120 120 \\
\text{compare } 120 120 \\
\]

5.1 Bulb Class Declaration

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) {}

    // 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;
};

5.2 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.
5.3 Lamp Constructor [ / 9 ]

sample solution: 7 line(s) of code

5.4 Lamp Destructor [ / 5 ]

sample solution: 8 line(s) of code
sample solution: 10 line(s) of code
Write a recursive function named \texttt{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 $4 + 2 + 3 = 9$.

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

\textit{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?
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.
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:

```cpp
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
9 Dynamically Allocated & Templated Stairs

In this problem you will write a simple class to build a staircase-shaped storage shelf. Here’s an example usage of the class, which constructs the diagram on the right.

```cpp
Stairs<char> s(4,'_');
s.set(0,0,'A');
s.set(1,1,'B');
s.set(2,2,'C');
s.set(3,3,'D');
s.set(2,1,'U');
s.set(3,1,'S');
```

9.1 Stairs Class Declaration

First, fill in the blanks in the class declaration:

```cpp
class Stairs {

public:

  // constructor

  sample solution: 1 line(s) of code

  // destructor

  sample solution: 1 line(s) of code

  // prototypes of 2 other important functions related to the constructor & destructor

  sample solution: 2 line(s) of code

  // modifier

  void set(int i, int j, val) { data[i][j] = val; }

  /* NOTE: other Stair functions omitted */

private:

  // representation

  sample solution: 2 line(s) of code

};
```
9.2 Stairs Constructor [ / 9 ]
Now write the constructor, as it would appear outside of the class declaration (because the implementation is > 1 line of code).

sample solution: 10 line(s) of code

9.3 Stairs Destructor [ / 5 ]
Now write the destructor, as it would appear outside of the class declaration (because the implementation is > 1 line of code).

sample solution: 6 line(s) of code
Ben Bitdiddle is working on a software project for essay writing using a doubly-linked chain of nodes. His initial Node class is on the right.

One of the features of his software allows a user to compare the location of two words within the document and say which word appears first. Ben plans to implement this using two helper functions: search and compare.

### 10.1 Searching for a Word

First, let's write the search function, which takes in two arguments: a pointer to the first Node in the document (word chain) and the specific word we're looking to find. The function returns a pointer to the first Node containing that word. Use recursion to implement this function.

```cpp
class Node {
public:
    std::string word;
    Node* next;
    Node* prev;
};
```

*sample solution: 7 line(s) of code*

If the Node chain contains $n$ elements, what is the running time of the search function?

### 10.2 Comparing Positions within the Node Chain

Next, let's implement the compare function. This function takes in two Node pointers and returns true if the first argument appears closer to the front of the list than the second argument. For example, let's say a chain of word Nodes named sentence contains:

the quick brown fox jumps over the lazy dog

Here's an example using the search and compare functions:

```cpp
Node* over = search(sentence, "over");
Node* quick = search(sentence, "quick");
Node* lazy = search(sentence, "lazy");

assert (compare(quick, over) == true);
assert (compare(over, quick) == false);
assert (compare(quick, lazy) == true);
assert (compare(lazy, over) == false);
```
Again using recursion, implement the `compare` function.

If the `Node` chain contains $n$ elements, what is the running time of the `compare` function?

**Improving Word Position Comparison Performance**

Alyssa P. Hacker stops by to help, and suggests that Ben switch to a different data structure if he is frequently comparing word positions within a long essay.

But Ben’s a stubborn guy. Instead of switching to a different data structure, he has a plan to augment his list structure to improve the running time of `compare`. Ben explains that the new `distance` member variable in each node will indicate how far away the node is from the front of the list.

Here’s Ben’s new `compare` function:

```cpp
class Node {
  public:
    std::string word;
    Node* next;
    Node* prev;
    float distance;
};

bool compare_fast(Node *a, Node *b) {
  return a->distance < b->distance;
}
```

Ben reassures Alyssa that he’ll add some error checking to this code.  
*SIDE NOTE: Hopefully your implementation of the original compare function has some error checking!*

But Alyssa is more concerned about how this addition to the data structure will impact performance when the essay or sentence is edited. She says he can’t afford to change the `distance` in all or many `Nodes` in the data structure any time a small edit is made to the document.

Ben explains that the `push_back` function will assign the distance of the new `Node` to be the distance of the last `Node` in the chain plus 10.0. And similarly, `push_front` will assign the new `Node` to be the distance from the first `Node` minus 10.0. *BTW, negative distance values are ok.* Finally, Ben says the `insert_between` function (on the next page) can similarly be implemented without editing the distance value in any existing `Node`!
Continuing with the previous example, here’s a quick demonstration of how this function works:

```c
bool success = insert_between(sentence, "the", "lazy", "VERY");
assert (success);
Node* VERY = search(sentence, "VERY");
assert (compare(VERY, lazy) == true);
assert (compare(quick, VERY) == true);
assert (compare_fast(VERY, lazy) == true);
assert (compare_fast(quick, VERY) == true);
success = insert_between(sentence, "quick", "fox", "RED");
assert (!success);
```

And here’s the contents of the `sentence` variable after the above fragment of code:

```plaintext
the quick brown fox jumps over the VERY lazy dog
```

Implement `insert_between`. And yes, use recursion.
Write a function named `erase_middles` that takes in 2 arguments: an STL list named `data` and a `value`. The function should remove all instances of `value` from `data`, except the first and the last instances. The function returns the number of removed elements. For example, if `data` initially contains:

```
5 2 5 2 3 4 3 2 5 2 3 2 3 4 2 5
```

A call to `erase_middles(data, 5)` will return 2 and now `data` contains:

```
5 2 2 3 4 3 2 2 3 2 3 4 2 5
```

And then a call to `erase_middles(data, 2)` will return 4 and `data` contains:

```
5 2 3 4 3 3 3 4 2 5
```

*sample solution: 22 line(s) of code*
A complex recursive function seems to be entering an infinite loop, despite what I think are perfect base cases.

The program always gets the right answer, but when I test it with a complex input dataset that takes a long time to process, my whole computer slows down.

I'm unsure where the program is crashing.

I've got some tricky math formulas and I suspect I've got an order-of-operations error or a divide-by-zero error.

I'm implementing software for a bank, and the value of a customer's bank account is changing in the middle of the month. Interest is only supposed to be added at the end of the month.