

CSCI-1200 Data Structures

Test 1 — Practice Problems

Note: This packet contains selected practice problems from three previous exams. Your exam will contain approximately one third to one half as many problems.

1 Loading the Dishwasher [/33]

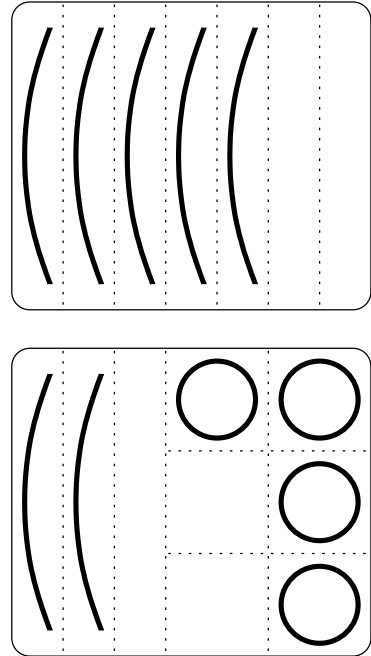
In this problem you will implement a simple class named `Dishwasher` to keep track of the contents of a dishwasher as it's being loaded with plates, cups, forks, knives, and spoons. The total volume of a `Dishwasher` is measured in number of plates. The diagram on the right shows a dishwasher that has room to hold 7 plates. If we want to wash a mix of plates and cups, we can load 3 cups in the space normally occupied by 2 plates. Note: There is no limit on space for forks, knives, or spoons. *We never run out of space for utensils!*

Here's how we create a new dishwasher object, specifying the volume in number of plates:

```
Dishwasher dw(7);
```

And here's how we begin to load the dishwasher. Note that we perform error checking each time we add a cup or plate. None of the error checks in this portion of the example are triggered.

```
if (!dw.addPlate("blue")) { std::cerr << "ERROR 1: cannot add another plate" << std::endl; }
if (!dw.addPlate("red"))  { std::cerr << "ERROR 2: cannot add another plate" << std::endl; }
if (!dw.addPlate("green")) { std::cerr << "ERROR 3: cannot add another plate" << std::endl; }
if (!dw.addCup())         { std::cerr << "ERROR 4: cannot add another cup"   << std::endl; }
if (!dw.addPlate("green")) { std::cerr << "ERROR 5: cannot add another plate" << std::endl; }
if (!dw.addPlate("red"))  { std::cerr << "ERROR 6: cannot add another plate" << std::endl; }
if (!dw.addCup())         { std::cerr << "ERROR 7: cannot add another cup"   << std::endl; }
dw.addSpoon(); dw.addFork(); dw.addKnife(); dw.addFork(); dw.addSpoon();
dw.addFork();  dw.addFork(); dw.addKnife(); dw.addSpoon(); dw.addFork();
```



Next, we print the contents of the dishwasher and the number of complete sets of utensils (1 fork + 1 knife + 1 spoon). *It seems like we always have extra forks!*

```
dw.printContents();
std::cout << dw.completeUtensilSets() << " complete utensil set(s)" << std::endl;
```

Here is the output (to `std::cout`) after the above statements. Note that we print the colors of the plates in the order they were inserted.

```
5 plate(s): blue red green green red
2 cup(s)
10 utensil(s)
2 complete utensil set(s)
```

Finally, let's explore what happens if we try to load more cups & plates into the dishwasher...

```
if (!dw.addPlate("red")) { std::cerr << "ERROR 8: cannot add another plate" << std::endl; }
if (!dw.addCup())        { std::cerr << "ERROR 9: cannot add another cup"   << std::endl; }
if (!dw.addCup())        { std::cerr << "ERROR 10: cannot add another cup"  << std::endl; }
```

Two of these additions fail, as we can see by this output to `std::cerr`:

```
ERROR 8: cannot add another plate
ERROR 10: cannot add another cup
```

1.1 Dishwasher Class Declaration [/15]

Using the sample code on the previous page as your guide, write the class declaration for the `Dishwasher` object. That is, write the *header file* (`dishwasher.h`) for this class. You don't need to worry about the `#include` lines or other pre-processor directives. Focus on getting the member variable types and member function prototypes correct. Use `const` and call by reference where appropriate. Make sure you label what parts of the class are `public` and `private`. Save the implementation of all functions > 1 line for the `dishwasher.cpp` file, which is the next part.

sample solution: 19 line(s) of code

1.2 Dishwasher Class Implementation [/18]

Now implement the member functions, as they would appear in the corresponding `dishwasher.cpp` file.

sample solution: 31 line(s) of code

2 Resizable Histogram [/20]

Let's build a histogram of students organized by their grade on Homework 1 into buckets of size 5. An example of the data is to the right and the expected output is on the far right.

First, write a fragment of code (as it would appear in `main.cpp`) to open the file `hw1_scores.txt` and read the data into an STL vector of STL vectors of STL strings. The vector should be resized as necessary to adapt to the maximum score present in the file.

`hw1_scores.txt`

```
alice 30
bob 32
chris 23
dan 29
erin 39
fred 10
georgia 27
harry 30
```

`std::cout`

```
[ 0- 4]
[ 5- 9]
[10-14] fred
[15-19]
[20-24] chris
[25-29] dan georgia
[30-34] alice bob harry
[35-39] erin
```

sample solution: 12 line(s) of code

Next, write code to output the data stored in the vector to `std::cout` as shown above.

sample solution: 7 line(s) of code

3 **Sorting L33T5P34K (a.k.a. LEETSPEAK) [/15]**

We saw in Homework 2 that the default sorting of STL `string` objects places numerical digits (0-9) before capital letters (A-Z). Therefore, following the STL defaults, this sequence of strings is considered sorted:

3ND 4ND 5LØP 5L1P 5L3PT 5L4P DØG D1G

Let's instead view these strings as English words encoded by 1980's hackers or texting teenagers in so-called *LEETSPEAK*, using these substitutions: Ø↔O, 1↔I, 3↔E, 4↔A, and 5↔S. Therefore, this is a more appropriate alphabetization of these strings:

4ND D1G DØG 3ND 5L4P 5L3PT 5L1P 5LØP

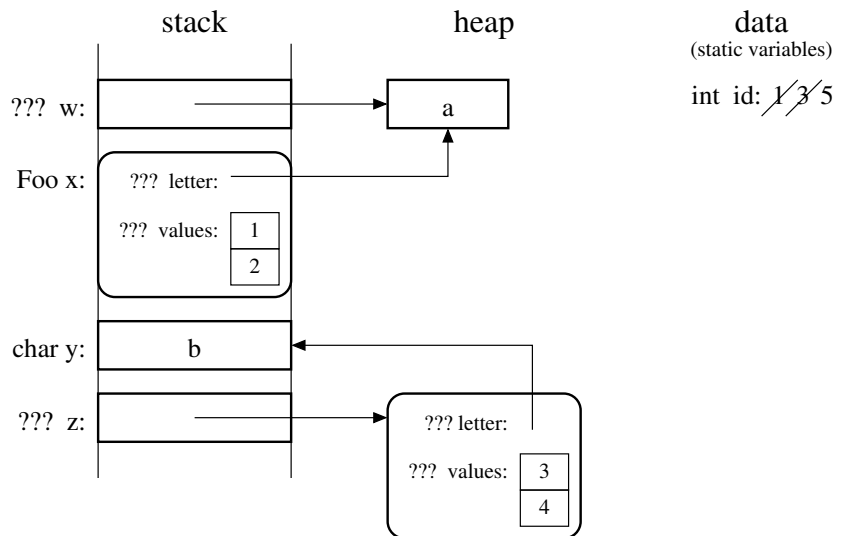
Implement a function named `leetspeak_sorter`, that may be passed to the STL `sort` routine for STL `vectors` to alphabetize a collection of LEETSPEAK words. You may assume all characters of these words are either capital letters or numbers and use only the letter substitutions listed above.

sample solution: 14 line(s) of code

4 Memory Diagramming [/20]

Write code to produce the memory structure shown in the diagram to the right.

Some types have been omitted (marked with “???”).



sample solution: 21 line(s) of code

5 Opening a New Hair Salon [/32]

In this problem you will implement a simple class named `Customer` to keep track of customers at a hair salon. First, we create 6 `Customer` objects:

```
Customer betty("Betty");      Customer chris("Chris");      Customer danielle("Danielle");
Customer erin("Erin");        Customer fran("Fran");        Customer grace("Grace");
```

Then, we can track customers as they come to the salon for appointments on specific dates with one of the salon's stylists. We use the `Date` class we discussed in Lecture 2. You may assume the appointments are entered chronologically, with increasing dates.

```
betty.hairCut (Date(1,15,2015), "Stephanie");
chris.hairCut (Date(1,17,2015), "Audrey" );
grace.hairCut (Date(1,20,2015), "Stephanie");
danielle.hairCut(Date(1,28,2015), "Stephanie");
chris.hairCut (Date(2, 5,2015), "Audrey" );
betty.hairCut (Date(2, 9,2015), "Stephanie");
fran.hairCut (Date(2,12,2015), "Audrey" );
danielle.hairCut(Date(2,18,2015), "Lynsey" );
betty.hairCut (Date(2,20,2015), "Stephanie");
```

In the system, each customer record will store the customer's preferred stylist. The preferred stylist is defined as a customer's most recent stylist. A message is printed to `std::cout` on each customer's first visit to the salon, or if a customer switches to a new stylist. Here is the output from the above commands:

```
Setting Stephanie as Betty's preferred stylist.
Setting Audrey as Chris's preferred stylist.
Setting Stephanie as Grace's preferred stylist.
Setting Stephanie as Danielle's preferred stylist.
Setting Audrey as Fran's preferred stylist.
Setting Lynsey as Danielle's preferred stylist.
```

Next, we insert the customers into an STL vector:

```
std::vector<Customer> customers;
customers.push_back(betty); customers.push_back(chris); customers.push_back(danielle);
customers.push_back(erin); customers.push_back(fran); customers.push_back(grace);
```

And then sort & print them first alphabetically by stylist, and secondarily by most recent visit to the salon:

```
std::sort(customers.begin(),customers.end(),stylist_then_last_appointment);
for (int i = 0; i < customers.size(); i++) {
    std::cout << customers[i].getName() << " has had "
              << customers[i].numAppointments() << " appointment(s) at the salon";
    if (customers[i].numAppointments() > 0) {
        std::cout << ", most recently with " << customers[i].getStylist()
                  << " on " << customers[i].lastAppointment(); }
    std::cout << "." << std::endl;
}
```

Which results in this output to the screen:

```
Erin      has had 0 appointment(s) at the salon.
Chris     has had 2 appointment(s) at the salon, most recently with Audrey on 2/ 5/2015.
Fran     has had 1 appointment(s) at the salon, most recently with Audrey on 2/12/2015.
Danielle  has had 2 appointment(s) at the salon, most recently with Lynsey on 2/18/2015.
Grace     has had 1 appointment(s) at the salon, most recently with Stephanie on 1/20/2015.
Betty     has had 3 appointment(s) at the salon, most recently with Stephanie on 2/20/2015.
```

Note: Don't worry about output formatting/spacing. You may assume that the `Date` class has an `operator<` to compare/sort dates chronologically and an `operator<<` to print/output `Date` objects.

5.1 Customer Class Declaration [/15]

Using the sample code on the previous page as your guide, write the class declaration for the `Customer` object. That is, write the *header file* (`customer.h`) for this class. You don't need to worry about the `#include` lines or other pre-processor directives. Focus on getting the member variable types and member function prototypes correct. Use `const` and call by reference where appropriate. Make sure you label what parts of the class are `public` and `private`. Include prototypes for any related non-member functions. Save the implementation of all functions for the `customer.cpp` file, which is the next part.

sample solution: 14 line(s) of code

5.2 Customer Class Implementation [/17]

Now implement the member functions and related non-member functions of the class, as they would appear in the corresponding `customer.cpp` file.

sample solution: 26 line(s) of code

6 Color Analysis of HW1 Images [/21]

Write a function named `color_analysis`, that takes three arguments: `image`, an STL vector of STL strings representing a rectangular ASCII image (similar to HW1); an integer `num_colors`; and a character `most_frequent_color`. The function scans through the image and returns (through the 2nd & 3rd arguments) the number of different colors (characters) in the image & the most frequently appearing color.

sample solution: 27 line(s) of code

What is the order notation of your solution in terms of w & h , the width & height of the image, and c , the number of different colors in the image?

7 Power Matrix Construction [/16]

Write a function named `make_power_matrix` that takes in two arguments, `num_rows` and `num_columns`, and creates and returns a 2D matrix using STL vectors. Each element of the matrix $m_{r,c}$ stores the value r^c , that is, the row index raised to the power of the column index. For example, `make_power_matrix(5,7)` should produce this matrix:

1	0	0	0	0	0	0
1	1	1	1	1	1	1
1	2	4	8	16	32	64
1	3	9	27	81	243	729
1	4	16	64	256	1024	4096

Try to write this function *without using* the `pow` function.

sample solution: 13 line(s) of code

What is the order notation of your solution in terms of r , the number of rows, and c , the number of columns?

8 Diagramming Pointers & Memory [/15]

In this problem you will work with pointers and dynamically allocated memory. The fragment of code below allocates and writes to memory on both *the stack* and *the heap*. Following the conventions from lecture, draw a picture of the memory after the execution of the statements below.

```
char* cat;  
char** dog;  
char fish[2];  
char horse;  
dog = new char*[3];  
dog[0] = new char;  
fish[0] = 'b';  
fish[1] = 'i';  
dog[1] = &fish[1];  
dog[2] = &horse;  
cat = dog[0];  
*cat = 'r';  
horse = 'd';
```

Now, write a fragment a C++ code that cleans up all dynamically allocated memory within the above example so that the program will not have a memory leak.

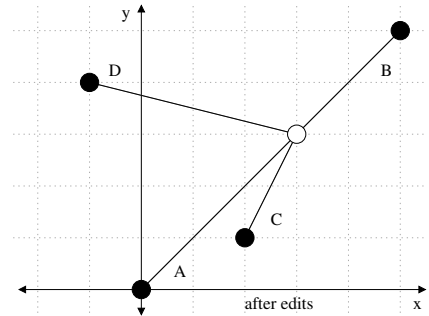
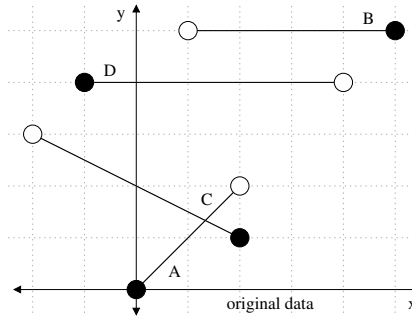
9 Classy Line Slopes [/28]

In this problem you will implement a simple class named `Line` to keep track of two dimensional lines. Lines are defined by two endpoints with integer coordinates. We will calculate the slope and y-axis intercept of each of the lines. Remember from algebra/geometry class that the equation for a line is $y = mx + b$, where m is the slope and b is the y-intercept. You may assume that the two endpoints are not the same point, and that the line is not exactly vertical (which would correspond to $\text{slope} = \infty$).

In the example below we make four `Line` objects and put them in an STL `vector`. The diagram on the left shows the original position of each of the lines – the black dot is the “first” endpoint and the white dot is the “second” endpoint. The `Line` class allows us to edit the second endpoint of each line, as shown in the diagram on the right, and in the code below.

```
Line a ("A", 0, 0, 2, 2);
Line b ("B", 5, 5, 1, 5);
Line c ("C", 2, 1, -2, 3);
Line d ("D", -1, 4, 4, 4);
```

```
std::vector<Line> lines;
lines.push_back(a);
lines.push_back(b);
lines.push_back(c);
lines.push_back(d);
```



Here's a helper function that outputs information about each line stored in a `vector`:

```
void printLines(const std::vector<Line> &lines) {
    for (int i = 0; i < lines.size(); i++) {
        std::cout << "Line " << lines[i].getName()
                    << std::fixed << std::setprecision(2)
                    << " with slope=" << std::setw(5) << lines[i].getSlope()
                    << " and y intercept=" << std::setw(5) << lines[i].getYIntercept()
                    << std::endl;
    }
}
```

Here's a code fragment that will first sort the line collection by slope and then print the lines:

```
std::cout << "original data, sorted by slope" << std::endl;
sort(lines.begin(), lines.end(), by_slope);
printLines(lines);
```

Now we edit the second endpoint of each line to be the point (3,3) and then sort and print the data again:

```
for (int i = 0; i < lines.size(); i++) {
    lines[i].setNewSecondPoint(3,3);
}
std::cout << "after changing second point to (3,3)" << std::endl;
sort(lines.begin(), lines.end(), by_slope);
printLines(lines);
```

The code above results in this output to the screen:

```
original data, sorted by slope
Line C with slope=-0.50 and y intercept= 2.00
Line D with slope= 0.00 and y intercept= 4.00
Line B with slope= 0.00 and y intercept= 5.00
Line A with slope= 1.00 and y intercept= 0.00
after changing second point to (3,3)
Line D with slope=-0.25 and y intercept= 3.75
Line B with slope= 1.00 and y intercept= 0.00
Line A with slope= 1.00 and y intercept= 0.00
Line C with slope= 2.00 and y intercept=-3.00
```

9.1 Line Class Declaration [/13]

Using the sample code on the previous page as your guide, write the class declaration for the `Line` object. That is, write the *header file* (`line.h`) for this class. You don't need to worry about the `#include` lines or other pre-processor directives. Focus on getting the member variable types and member function prototypes correct. Use `const` and call by reference where appropriate. Make sure you label what parts of the class are `public` and `private`. Include prototypes for any related non-member functions. Save the implementation of all functions for the `line.cpp` file, which is the next part.

sample solution: 12 line(s) of code

9.2 Line Class Implementation [/15]

Now implement the member functions and related non-member functions of the class, as they would appear in the corresponding `line.cpp` file.

sample solution: 28 line(s) of code

10 Common C++ Programming Errors [/12]

For each code fragment below, choose the letter that best describes the program error. *Hint: Each letter will be used exactly once.*

- A) Uninitialized memory
- B) Compile error: type mismatch
- C) Accessing data beyond the array bounds
- D) Infinite loop

- E) Math error (incorrect answer)
- F) Memory leak
- G) Syntax error
- H) Does not contain an error

```
float* floating_pt_ptr = new float;
*floating_pt_ptr = 5.3;
floating_pt_ptr = NULL;
```

```
unsigned int x;
for (x = 10; x >= 0; x--) {
    std::cout << x << std::endl;
}
```

```
int* apple;
int banana[5] = {1, 2, 3, 4, 5};
apple = &banana[2];
*apple = 6;
```

```
std::vector<std::string> temperature;
temperature.push_back(43.5);
```

```
double x;
for (int i = 0; i < 10; i++) {
    x += sqrt(double(i));
}
```

```
int balance = 100;
int withdrawal;
std::cin >> withdrawal;
if (withdrawal <= balance)
    balance -= withdrawal;
    std::cout << "success\n";
else
    std::cout << "failure\n";
```

```
float a = 2.0;
float b = -11.0;
float c = 12.0;
float pos_root =
    -b + sqrt(b*b - 4*a*c) / 2*a;
float neg_root =
    -b - sqrt(b*b - 4*a*c) / 2*a;
```

```
int grades[4] = { 1, 2, 3, 4 };
std::cout << "grades" << grades[1]
    << " " << grades[2]
    << " " << grades[3]
    << " " << grades[4]
    << std::endl;
```


11 Detecting Compound Words [/18]

Write a C++ function that takes in a collection of English words stored as an STL **vector** of STL **strings**. The function should return a **vector** containing all *compound words* from the input collection. We define a compound word as two or three words joined together to make a different word. For example, given the input collection:

```
a back backlog backwoods backwoodsman cat catalog
less log man none nonetheless ship the woods woodsman
```

Your function should return (in any order):

```
backlog backwoods backwoodsman catalog nonetheless woodsman
```

sample solution: 21 line(s) of code

If there are n words in the input, what is the order notation of your solution?