Welcome to Lab Debug!
Course Website: https://courses.engr.illinois.edu/cs225/sp2022

Overview
In this week’s lab, you will get to practice an essential skill in computer science: debugging. This worksheet will get you familiar with some “best practices” and questions to ask yourself when debugging your code. For a more comprehensive list, see lab_debug’s webpage.

Understanding the Logic
The first step in debugging is to understand what the code is meant to do. This will make catching “logic errors” (errors in the logic of the code) easy.

One good way to debug such errors is to execute the code in your head, line by line, and explain to yourself (even a rubber duck!): What is this line trying to do? Is it doing what it is supposed to do?

Exercise 1: There are two bugs in this piece of code - find and correct them.

```cpp
void blackStripes(PNG* myimage)
{  
    for(unsigned h=0;h<myimage->height(); h++)
    {
       // WILL CAUSE INFINITE LOOP IF THERE IS NO h++
       for (unsigned w=0;w<myimage->width();w+=2)
       {
             HSLAPixel* current = myimage->getPixel(w,h);
             double* lum = &current.l;
             lum = 0;  // LUM IS A POINTER: CURRENT.L VALUE
             // WILL NOT CHANGE!!
             INSTEAD DO:
             *lum = 0; OR:
             current.l = 0;
       }
    }
}
```

Exercise 2.1: For each variable below, state whether it is stored on the stack or the heap. For pointers, also answer where it is pointing to.
- `width` is stored on: __STACK__
- `cube` is stored on: __STACK__ as a pointer, it points to an address that is on: __HEAP__
- `cube_double` is stored on: __STACK__ as a pointer, it points to an address that is on: __STACK__
- `v` and `s` are stored on: __STACK__

Exercise 2.2: One line in the code below may cause a segfault when the code is run. Which line is it? _line 10_. Fix the code so no segfault occurs. Note: please do not change function signatures!

Line 10 might segfault because `cube_double` is a pointer to the return address of `CreateDoubleCube()` which is on the stack (c). By the time `cube_double->getVolume()` is called in line 10, there is NO GUARANTEE that the stack memory where c was stored hasn’t been overwritten already and does not represent an Cube anymore.

```cpp
main.cpp

```
Copying Correctly
When copying variables, we need to think about two things - what we want to copy (value or address) and what is the type of the variable we want to copy (primitive or complex). Depending on the case, we can “copy an object” or “copy a pointer”. Copying an object copies values of each element in the object over. On the other hand, copying a pointer just copies the address without allocating new memory. Keep this in mind as you work through Exercise 3.

Exercise 3.1: What will be printed out in lines 10 and 12 of main.cpp? Both lines will print out 3\times3\times3=27. BOTH width variables have been changed because line 7 (c2 = c1) only copies the address of c1 to c2, meaning c2 will point to the SAME heap memory address that c1 points to, IT WILL NOT CREATE A NEW CUBE OBJECT! Thus if we change one width variable, the other automatically changes too.

Exercise 3.2: Fix the code so that the content of c1 is copied into c2.

---

<table>
<thead>
<tr>
<th>Cube.h</th>
<th>Cube.cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1   #pragma once</td>
<td>1   #include &quot;Cube.h&quot;</td>
</tr>
<tr>
<td>2   class Cube{</td>
<td>3   double</td>
</tr>
<tr>
<td>3     public:</td>
<td>4     Cube::getVolume(){</td>
</tr>
<tr>
<td>4         double w;</td>
<td>5         return w * w * w;</td>
</tr>
<tr>
<td>5     double getVolume();</td>
<td>6 }</td>
</tr>
<tr>
<td>7 }</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

main.cpp