lab_debug : Disastrous Debugging
Week #2 – January 29-31, 2020

Welcome to Lab Debug!

Course Website: https://courses.engr.illinois.edu/cs225/sp2020

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.

```
 1
 2  void blackStripes(PNG* myimage){
 3    for(unsigned h=0; h<myimage->height() ){
 4      for (unsigned w=0; w<myimage->width(); w+=2){
 5
 6        HSLAPixel& current = myimage-> getPixel(w,h);
 7        double* lum = &current.l;
 8        lum = 0;
 9
10
11      }
12    }
13  }
14
```

Stack or Heap?

Remember that stack and heap memory have different *lifetimes*. The lifetime of a variable on the **stack** is based on its "scope." Once its scope is over, it is de-allocated automatically. The lifetime of a variable on the **heap** is controlled by you. Heap memory is de-allocated only when the application exits or when you explicitly free it. You can request memory on the heap using the keyword **new**.
A segmentation fault (**segfault**), occurs when a program tries to access memory that doesn't belong to it. **Segfaults often occur when using uninitialized, null or invalid pointers.**
When declaring and initializing variable, think about where it should be saved: on the stack or on the heap.

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: _____
- **cube** is stored on: _____
  as a pointer, it points to an address that is on: _____
- **cube_double** is stored on: _____
  as a pointer, it points to an address that is oo: _____
- **v** and **s** are stored on: _____

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

main.cpp

```
 1
 2  Cube *CreateDoubleCube(Cube *original) {
 3    double width = original->w;
 4    Cube c(2*width);
 5    return &c;
 6  }
 7
 8  int main() {
 9    Cube *cube = new Cube(10);
10    Cube *cube_double = CreateDoubleCube(cube);
11    double v = cube_double->getVolume();
12    double s = cube_double->getSurfaceArea();
13    cout << v << " " << s << endl;
14    return 0;
15  }
```

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?

line 10: _____ **line 12:** _____

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

Cube.h

```
1  #pragma once
2
3  class Cube{
4    public:
5      double w;
6      double getVolume();
7
8
9  };
```

Cube.cpp

```
1  #include "Cube.h"
2
3  double Cube::getVolume(){
4    return w * w * w;
5  }
6
7
8
9
```

main.cpp

```
1
2  int main(){
3    Cube* c1 = new Cube();
4    c1->w = 4;
5    Cube* c2;
6
7    c2 = c1;
8    c2->w = 3;
9
10   std::cout<<c1->getVolume()<<std::endl;
11
12   std::cout<<c2->getVolume()<<std::endl;
13
14   // Clean up memory
15   delete c1;
16   delete c2;  //ERORR !! Why?
17
18 }
```

In the programming part of this lab, you will:

- Learn about debugging techniques and best practices
- Explore the given code and discover how it modifies images
- Find and correct bugs in the code

As your TA and CAs, we're here to help with your programming for the rest of this lab section! ☺