lab_debug : Disastrous Debugging

Week #2 – September 5-7, 2018

Welcome to Lab Debug!

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

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.

Exercise 1: How is the function blackStripes() intended to modify myimage? Does it actually accomplish what it intends to? If not, where are the errors?

```
 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        HSLAPixel& current = myimage-> getPixel(w,h);
 6        double* lum = &current.l;
 7        lum = 0;
 8
 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's a stack variable or a heap variable.

Exercise 2.2: One line in the code below may cause a segfault when the code is run. Which line is it? _________ How would you fix it?

main.cpp

```
 1  HSLAPixel* foo(PNG* input){
 2    int midW = input->width()/2;
 3    int midH = input->height()/2;
 4    double hue = input->getPixel(midW,midH).h;
 5
 6    HSLAPixel midPix(hue,0.5,0.5);
 7    return &midPix;
 8  }
 9
10  void transform(PNG* original) {
11    HSLAPixel* center = foo(original);
12    double* myhue = new double;
13    *myhue = center->h;
14    for(unsigned w=0;w<original->width();w++){
15      for(unsigned h=0;h<original->height();h++){
16        HSLAPixel& curr = original->getPixel(w,h);
17        curr.h = *myhue;
18      }
19    }
20  }
21  int main(){
22    PNG png;
23    png.readFromFile("alma.png");
24    transform(&png);
25    return 0;
26  }
```

- midPix is saved in: ________________
- center is saved in: ________________
- myhue is saved in: ________________
- w and h are saved in: ______________
- curr is saved in: ________________

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 use a "deep copy" or a "shallow copy." A deep copy allocates new memory and copies values over. On the other hand, a shallow copy just copies the pointer without allocating new memory. Keep this in mind as you work through Exercise 3.

<u>Exercise 3.1:</u> What will be printed out in lines **10** and **12** of main.cpp?

<u>Exercise 3.2:</u> Change the code so that **c1** is copied correctly into **c2**.

Cube.h

```
#pragma once

class Cube{
  public:
    double w;
    double getVolume();


};
```

Cube.cpp

```
#include "Cube.h"

double
Cube::getVolume(){
  return w*w*w;
}
```

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! ☺