

Welcome to Lab Memory!

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

Overview

In this week's lab, you will learn about memory management: how to allocate and de-allocate memory correctly in your program. You will discover ways memory management can go, and get to practice spotting memory bugs in the code.

Destructors

Destructors (dtors) are special member functions of classes. They are the opposite on constructors: their job is to release (de-allocate) memory when an object of the class is destroyed. Destructors are called automatically called; we never explicitly call a destructor, instead it is implicitly called when we use the keyword "delete" or when the lifetime of the object ends.

Exercise 1.1: Write the destructor for the **Orchard** class.

Exercise 1.2: When will Orchard and Tree destructors be called in **main.cpp**?

orchard.h

```
1 #pragma once
2
3 class Tree{
4 public:
5     string fruitName;
6     double yield;
7 };
8 class Orchard{
9 public:
10     static const int MAX_TREES = 500;
11     Orchard();
12     bool addTree(Tree& t);
13     Orchard(const Orchard& other);
14     Orchard& operator = (Orchard const & other);
15                                     // YOUR CODE HERE
16 private:
17     int size_;
18     Tree* trees_;
19 };
```

orchard.cpp

```
1 #include "orchard.h"
2
3 Orchard::Orchard(){
4     size_ = 0;
5     trees_ = new Tree[MAX_TREES];
6 }
7
8 bool Orchard::addTree(Tree& t){
9     if(size_ < MAX_TREES){
10         trees_[size_] = t;
11         size_++;
12         return true;
13     }else{ return false;}
14 }
15 Orchard::Orchard (const Orchard& other){
16     trees_ = new Tree[MAX_TREES];
17     for (int i=0; i<other.size_; i++){
18         trees_[i] = other.trees_[i];
19     }
20     size_ = other.size_;
21 }
22
23 Orchard& Orchard::operator = (Orchard const &
24 other){
25     if(this != &other){
26         for (int i=0; i<other.size_; i++){
27             (*this).trees_[i] = other.trees_[i];
28         }
29         (*this).size_ = other.size_;
30     }
31     return *this;
32 }
33
34 // YOUR CODE HERE
35
36
37
38
39
40
41
42
43
44
```

```

main.cpp
1 int main() {
2     Tree *t1 = new Tree;
3     (*t1).fruitName = "peach";
4     (*t1).yield = 25;
5     Tree *t2 = new Tree;
6     (*t2).fruitName = "apple";
7     (*t2).yield = 40;
8
9     Orchard * myorchard = new Orchard();
10    myorchard->addTree(*t1);
11    myorchard->addTree(*t2);
12    delete myorchard;
13    delete t1; delete t2;
14 }

```

Memory Errors

Memory errors occur when memory access is mismanaged: some ways it can occur are through 1) invalid memory access in heap or stack, 2) mismatched allocation/deallocation, or 3) missing allocation or uninitialized variable access. Memory errors often result in “segfaults” when the program is run.

Exercise 2.1: What should line 7 in **main.cpp** print out?

Exercise 2.2: A memory error will occur somewhere between **lines 10 and 16**. Find and correct this error.

```

main.cpp
1 void func(int idx){
2     HSLAPixel array[10];
3     array[idx] = new HSLAPixel(0,0,0);
4 }
5 int main() {
6     for (int i=0; i<20; i++){
7         std::cout<< i<< std::endl;
8         func(i);
9     }
10    HSLAPixel * pix1 = new HSLAPixel();
11    HSLAPixel * pix2 = new HSLAPixel();
12    pix2 = pix1;
13    delete pix1;
14    delete pix2;
15    return 0;
16 }

```

Memory Leaks

Memory leaks are a type of Memory Error. Memory leaks most commonly occur when memory that is no longer needed is not correctly released (still reachable block), or when an object/variable is stored in memory but cannot be accessed by the running code (lost block). Memory leaks are often harder to detect than memory errors as they won't always cause an error at runtime. Debugging tools such as Valgrind can help detect memory leaks.

Exercise 3: For each memory block allocated in the code below, decide if it has been released correctly. If not, add code to correctly release it.

```

main.cpp
1 int main(){
2     int* arr = new int[10];
3     int m = 300;
4     arr[0] = m;
5     PNG* image = new PNG(m,m);
6     HSLAPixel& mypix = (*image).getPixel(150,150);
7     // Clean up memory
8
9
10
11 }
12
13
14

```

In the programming part of this lab, you will:

- Learn about two memory debugging tools: Valgrind and ASAN
- Complete the given code for lab_memory
- Debug the given code by correcting memory errors and memory leaks

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