ECE 2574: Data Structures and Algorithms - Memory Management

C. L. Wyatt

Today we will review memory management in C++ and extend your toolbox for dealing with it.

- Warmup
- Allocation using new
- Freeing memory using delete
- RAII
- Rule of 3 (now 5)
- Exercise

Warmup #1

Suppose I have a class Foo. What C++ code will dynamically allocate an object my_foo of type Foo using the default constructor?

- `Foo my_foo = new Foo;` 7%
- `Foo* my_foo = new Foo;` 34% (correct answer)
- `Foo* my_foo = new Foo();` 66% (correct answer)
- `Foo my_foo;` 7%
- `auto my_foo = new Foo;` 2% (correct answer)

Warmup #2

Given the same Foo class, what C++ code will dynamically allocate an array of 10 Foo objects my_foo_array?

- `Foo * my_foo_array = new Foo[10];` 80% (correct answer)
- `Foo my_foo_array[10];` 9%
- `my_foo_array = new Foo(10);` 13%
- `Foo my_foo_array[] = new Foo[10];` 5%

Warmup #3

What C++ code will free the memory allocated in question 1?

- `delete Foo;` 4%
- `delete my_foo;` 57% (correct answer)
- `delete [] my_foo;` 30%
- `my_foo.delete();` 11%

Warmup #4

What C++ code will free the memory allocated in question 2?

- `delete my_foo_array;` 11%
- `delete [] my_foo_array;` 84% (correct answer)
- `delete(my_foo_array,10);` 0%
- `delete [] Foo;` 9%

Forms of `delete`

- The form of delete depends on whether a single object or an array of objects is allocated.
- The behavior if you use the wrong one is undefined.

Review of allocation/deallocation

- This is one of the primary uses of pointers in C++, handling memory.
- To allocate space for an object and initialize it with its constructor use `new`, which returns a pointer to the memory allocated
- To deallocate, or free, the previously allocated space use `delete` or `delete []` on the allocated pointer.

Memory management is one of the tricky aspects of C++

Whose responsibility is it to make sure memory gets allocated (and usually more difficult) deallocated at the correct time.
Fortunately there are a number of patterns and tools that help here.

- Resource Acquisition is Initialization (RAII)
- Smart Pointers
- Using containers specialized for memory management

We will look at RAII today:

One of the advantages of C++ over C is the increased ease in controlling object state

- Variables have a lifetime.
- The lifetime of a (automatic) stack allocated variable is determined by it's scope. This is not true for dynamically allocated variables.
- The lifetime is bounded by birth (Construction) and death (Destruction).

RAII uses the methods all classes have (implicitly or explicitly) to control resources.
For the case of memory, you allocate in a constructor and deallocate in a destructor of a statically allocated object, which transfers management to scope.

However its not quite that easy since all types support more than just construction/destruction

C++ Rule of three: If you write a (non-trivial) destructor you should also write a

- copy constructor and
- copy assignment operator

C++11 defines *moves semantics* which extends this to the rule of five, adding

- move constructor
- move assignment

In may cases you don't want to wrap every allocation inside a class and implement the 5 methods.

Instead internally use a standard container specialized for resource management:

- std::array
- std::vector
- std::list
- std::unique_pointer

However, in many situations in EE/CPE the standard library may not be allowed. So you might have to write your own container. Thus, you need practice managing resources yourself.

Exercise: A dynamically allocated array implementation of Bag

See website

Next Actions and Reminders

- Read CH pp. 133-148
- Warmup due before noon on Monday
- P1 is due on 9/27/17