CSCI 262 Lecture 17 – The Big 3

Outline

- When working with data structures using dynamically allocated memory, care must be taken to properly manage the allocated memory when copying, assigning, or destroying objects.
- The "Big 3":
    - Copy constructor – invoked whenever an object is copied, either during construction/initialization of another object, or passing into/out of a function or method call.
    - Assignment operator – invoked when one value is assigned from another.
    - Destructor – invoked when a stack allocated object goes out of scope, or when a dynamically allocated object is deleted.
- Deep copy versus shallow copy:
    - Deep copy copies regular variables, but not pointer values; instead, the memory *pointed to* is copied into newly allocated memory.

Readings

Read chapter 13.3 for Wednesday.

Programmer control of memory management in systems languages (such as C++) is generally considered necessary to get the most efficient use of the computer's resources. This contrasts with languages such as Java or Python, which are *garbage collected* – the language runtime manages memory on the programmer's behalf. (Of course, somewhere in the language runtime, dynamic memory management is happening!) Unfortunately, it is easy to get memory management wrong. One way to help programmers "get things right" is to apply *patterns* – well-tested, best practice approaches to solving general classes of problems. In the case of dynamic memory management, the pattern that is most closely associated with C++ (and is generally implemented in the Big 3) is called *Resource Acquisition is Initialization* (RAII). You can read more about RAII on Wikipedia: https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization

RAII in C++ may be made somewhat easier/safer with the application of *smart pointers*, a topic which is outside the scope of this course – see the <memory> header in the C++ standard library.

Self Check

1. Write three bits of code in which a copy constructor is being invoked (implicitly or explicitly).
2. Why must the parameter to a copy constructor be passed by reference?
3. What could happen if we don't put the "self assignment check" in the assignment operator overload?
4. Why don't we need to do anything special to "deep copy" non-member variables that hold on to complex types using dynamic allocation (e.g., vectors)?

For Further Practice

If you would like to get deeper into C++ memory management, you might want to learn about the "Big 5" (or "Rule of 5"), an extension of the Big 3 that provides efficiency/performance improvements for large objects in certain situations, such as returning by value.