CS 246 Fall 2019 — Tutorial 6

October 23, 2019

Summary

1 Uniform Initialization

There are many different ways objects can be initialized in C++. Uniform initialization (that is, with brace brackets) is preferred in C++11 and onwards.

Exercise: What is one instance where initialization with `=` or round brackets works, but uniform initialization does not?

Exercise (Hard!): What is one instance where uniform initialization works, but initialization with round brackets does not?

2 Destructor

The destructor is a method which is called when a object is destroyed. This is either when it is heap allocated and `delete` is called on it, or when it goes out of scope.

The destructor of a class `Foo` is the method `~Foo()`. Like with constructors, do not specify a return type when writing a destructor.

Exercise: For this tutorial, pick any object which manages dynamic memory. In these exercises, it will be called `Foo`. Some suggestions are linked lists, strings, or binary trees of integers. Write a destructor for `Foo`.

3 Copy Constructor

Recall that functions return-by-value. So to return a `Foo` from a function, we must describe how it can be copied. To do this, we define a copy constructor, which has type `Foo(const Foo&)`.

Exercise: Why is the parameter to this constructor a const reference?

Exercise: Write a copy constructor for `Foo`.

4 Copy Assignment Operator

Copy constructors only allow objects to be *initialized* using other objects of the same type. To modify an object after its initialization, must define `operator=`.

Exercise: Define a copy assignment operator for `Foo`, by defining `Foo &operator=(const Foo&)`.

The copy-and-swap idiom suggests that we reuse the copy constructor: use it to make a copy of the argument, and then simply swap each field.

Exercise: Why does copy-and-swap work? Write a copy-and-swap version of `Foo`'s assignment operator.