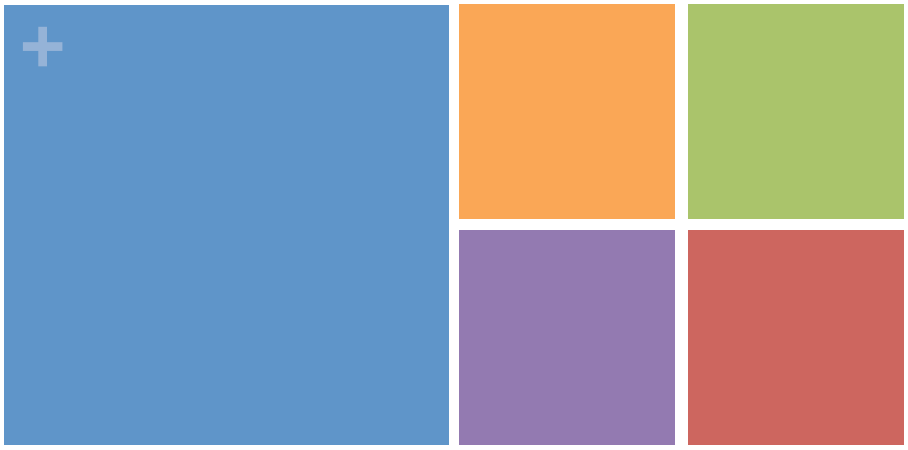

Lecture 30: More C++

Today

- Reading
  - Weiss Ch. 1, Ch. 2
  - (Weiss Ch. 4 is on classes)
- Objectives
  - Introduction to C++
  - Classes in C++
- Announcements
  - Quiz on Friday (one question on hash maps, one on C++)

This week's assignment

- Implement a priority queue in C++
- The header file is provided as starter code
- Use Aquamacs and Terminal
- Today's lecture and lab will help prepare you with the assignment
- "Java Structures" provides you already with an implementation for a priority queue (see the book)
- C++ Library Reference (link on course webpage)

Similarities between C++ and Java

- The `main` method
- Primitive types – int, double, float, etc
- Syntax – curly brackets, function syntax, return statements, etc
- Control constructs – loops and conditionals
- Generics
- Other things you'll discover along the way

Using the `std` namespace

- Namespaces are a generalization of packages
  - Named region of code contained in curly brackets
  - Helps disambiguate between variables and functions with same name
- The `std` namespace
  - Always have to specify
  - In C++, the `vector` type is in the `std` namespace
- To use `std`
  - Write `using std namespace;` at top of file
  - Use `::` operator, e.g. `std::vector` or `std::cout`

the "using" keyword is similar to import statement in Java

Operator Overloading

- Define a meaning for existing operations (e.g. + or []) for new class types

```
nums[i] // nums is a vector (i.e. an ArrayList in Java)
```

- Overloaded the [] operator so that it acts like the ``at'' method
  - The "at" method is bounds checked and throws an exception
  - There's no check in C++
- C++ documentation for vector shows the "operator[]" and "operator=" functions
- Makes classes look like primitives

Object management

- In Java, most types are objects
  - Everything except for primitives are allocated using `new`
  - Memory is taken from the heap
- In C++, everything is a primitive
  - Allocated on the stack not the heap
  - Allocate from heap by explicitly using the `new`

Object management

C++

```
vector<int> nums;
nums.push_back(5);
```

- Allocated on the stack
- Creates an empty vector of ints
- No need to use `new` keyword
- Looks similar to declaring a primitive

Java

```
ArrayList<Integer> nums;
nums = new ArrayList();
nums.add(5);
```

- Allocated on the heap
- Without the second line, we get a NullPointerException!

Object management

C++

```
vector<int> nums2 = nums;
```

Java

```
ArrayList<Integer> nums2;
nums2 = nums;
```

- *This difference between C++ and Java changes the semantics of assignment and parameter passing*
  - Assignment means copying
  - Parameters are copied (called pass-by-value)
  - Variables are no longer names with arrows pointing to the same memory location

Classes in C++

- See `IntCell` class in `intcell_onefile.cpp`
- Classes in C++
  - Class declaration ends with semicolon!
  - Visibility (public, private) declared for sections not individual members
  - Default is private
- Can only use `IntCell` class in same file!

Classes in C++

- Move class definition to header file (.h) so it can be imported
  - See `intcell_better.cpp` and `intcell_better.h`
- An even better idea: separate the declaration of the class from the implementation of the class
  - See `intcell_best.cpp`, `intcell_best.h`, and `intcell_tester.h`
- Only need to include the header file (.h) to compile a user's code
- Can change implementation without recompiling user's code

Classes in C++

- Include files can include more include files…
- Code won't compile if include a file more than once!

```
#ifndef INTCELL_BEST_H
#define INTCELL_BEST_H
// class declaration
#endif
```

- If variable `INTCELL_BEST_H` is already defined then won't include the class declaration
- Always do this when defining class declaration in header file!

The const keyword

- Accessor method vs. mutator method
- Using `const` tells the compiler that the function is an accessor
  - Promise function will not change the state of the object
  - Allows compiler to optimize your code

Destructors

- There is no garbage collection in C++
- Any memory you allocate from the heap, you must free!
  - Otherwise, you have a "memory leak"
- The destructor is called when the variable goes out of scope
- Name of the destructor is `~ClassName`

Assertions

- To write an assertion,

```
#include<cassert>  // need to include this
...
assert (boolean_expression);
```

- Can turn off assertions if you write

```
#define NDEBUG
```

before the `#include<cassert>` statement