CS 309: Autonomous Intelligent Robotics
FRI I

Lecture 6: C++ Recap

Instructor: Justin Hart

http://justinhart.net/teaching/2019_spring_cs309/

Today

- C++ Primer – Part 3

g++

- GNU c++ compiler
- G++ <input> -o

Ex01 - Hello World!

- Hello World is kind of a traditional programming exercise to demonstrate the basics of a programming language.
  - C & C++ versions of this look similar, but different

- Exercise Objectives:
  - #include
  - main()
  - printf/std::cout
  - return
  - Basic syntax
  - Invoking the compiler

Ex02 - Hello World!

- Same thing, but the C++ stream syntax

Ex03 – Variables

- Variable declarations
- Uninitialized variables have unknown values
- Assignment
- Addition
- Pre/post-increment

Ex04 – Namespaces

- C++ uses namespaces
- You can “use” a namespace to remove std:: (or similar)
- Namespaces prevent name collisions. There could be std::cout & special::cout, or others.
- In ROS we will use the ros namespace

Ex05 – Namespaces

- for
  - Generally iterates through lists, or until some variable hits a value based on increment
- while
  - While a condition is true
- do
  - Like while, but comparison is evaluated after the loop
- for(initializer, comparison, increment);
- while(comparison)
- {} forms a “block”
  - You can use 1 line after your loop (or if) syntax if not enclosed in a block

Ex06 – Functions

- A function with/without parameters
  - Formal parameters go onto the function
  - Actual parameters are used when the function is called
- Functions have return values
  - void means it returns nothing

Ex07 – Scoping

- In general, changing a variables value in a function does not affect its value outside of the function
  - Even if the variable shares a name
  - Even for parameters
  - References and pointers are special cases
    - A reference parameter **will** change
    - A pointer points to memory, so changing the memory contents in the function changes the contents everywhere
- Globals are possible.. but we'll avoid them

Ex08 – Header Files

- A function must have a *prototype* before it is used
  - But the full body isn't needed before its usage.
- Header files are files where the actual text is copied into the .cpp file where you *#include* it
- Header files have *#ifndef* / *#define* macros to prevent you from defining the same stuff twice (causing problems)
- Header files normally have function/class prototypes in them

Ex09 – Implementation Files

- The implementation of a function in a header, or really any function can go into its own .cpp file
- Prototypes in the header tell other files that that function exists
- You link all of your implementations by putting the .cpp files together behind g++
  - g++ a.cpp b.cpp -o program

Ex10 – if / else if / else

- if(boolean) {
  thing
  }
- if(boolean) {
  do this
  } else {
  do something else
  }
- if(boolean) {
  do this
  } else if(boolean) {
  do something else
  } else {
  do something else
  }

Ex11 – Types

- “int” types are *whole* numbers
- “float” types are *decimal* numbers
- C++ *truncates* floating-point numbers
  - Rounding functions are available
- Casting works like this (int) number

Ex12 – Types

- Pointers start with * and *point* to memory
- You can point to the memory storing a variable
  - Prefix the variable name with & to get a pointer to that variable
  - Prefix a pointer with * to *dereference* the pointer, returning the value *stored* rather than the pointer
- The *new* keyword will *allocate* memory
  - It returns a pointer to data of that type
- The *delete* keyword *frees* memory
  - You definitely should not delete pointers that you still need!
  - Or *statically allocated* memory (when you say int a = ..)
    - Memory allocated with *new* is *dynamically allocated*

Ex13 – Arrays

- Arrays are declared *type name[number];*
- Arrays are *indexed* name[index]
  - Indexing starts at 0
- Arrays have *n* items of *type*
  - The array index operator is really a special type of pointer syntax
  - You can allocate an array with *malloc* or *new[]*
    - And should delete/free accordingly

Ex14 – Vectors

- Vectors are an STL (Standard Template Library) template
  - push_back/pop_back
- Templates use *iterators* for access
- *erase* is is generally used for templates
- Vectors have a dynamic size, and do not need to be allocated all at once like arrays

Ex15 – Classes

- Classes hold related data together
- Initializers describe what that data should be initalized to (the part after the colon in the *constructor*)
- *Constructors* set up a class when an *object* is created
- Classes can have *methods* which are like functions
- Classes have *access control*
  - *Public*
  - *Private*
  - *Protected*

Ex16 – Pulling it Together

- Using a class with a header and an implementation

Ex17 – Inheritance and Abstract Classes

- Abstract classes tell us what needs to be in *child* classes
- Child classes *inherit* things from *parent* classes
- Imagine this:
  - You have a class for a sensor that returns pictures
    - Possibly different cameras require different things in order to get that picture out
    - You need a different child class for each camera, but each camera returns a picture, and so the parent describes that it returns a picture

Ex18 – Runtime errors & signed variables

- The syntax here is correct, but this program has a very predictable, understandable flaw.