

NE697: Introduction to Geant4

C++ Basics

September 9th, 2021
Dr. Micah Folsom



THE UNIVERSITY OF
TENNESSEE
KNOXVILLE



Today's Agenda

- Administrative items?
- Will grade assignment 2 this weekend

Pointers and Memory

- There are many different ways to do things, + it's hard!
- Often times, it's a design choice for a specific app/library
- You're stuck with what you're given
 - If the library uses raw pointers, it's hard to use smart pointers, and vice versa
- In addition, Geant4 uses the “singleton” pattern frequently
 - One and only one instance of an object (think “manager” types)
 - Effectively a global variable, which is discouraged
- My personal strategy: avoid pointers where possible
- The `std::` collections (array, vector, etc) have pointers underneath, so passing those objects around is cheap

Pointers and Memory

- Remember: it's all about bytes
 - “Types” are just a filter for interpreting bytes
 - When you declare a variable
 - Request N bytes of contiguous memory (int: 4 or 8)
 - Receive the address of the bytes and map it to the variable
 - Using that variable just says “interpret these bytes as an int”
 - The same is true **whether it's on the stack or the heap**
 - **Heap:** instead of just mapping variable → memory (value), we have a variable that holds the memory address
 - **[DEMO CONT'D]**
 - **[Quick DEMO on function arguments]**

C++ Basics: Classes and Structs

- The heart of object-oriented programming
- Think in terms of objects that have interfaces
 - Objects are just collections of data (and/or functions)
 - Turn the knob, pull the lever, access the data
- Make things compact and self-contained, with default values
 - Always a trade-off in complexity/maintainability/performance
 - Watch out for over-engineering

C++ Basics: Classes and Structs

- Classes and structs are (almost) the same under the hood! These differences are just how people use them
- “members”: variables/functions belonging to an object
- **struct**: data structure
 - Typically contains *only* data members that are publicly accessible
 - An easy way to capture multiple values to pass around
- **class**: complex object
 - Typically contains data members *and* functions
 - Usually split into public and private/protected access levels

C++ Basics: Classes and Structs

- Fundamentally the same, except one thing:
 - structs default to **public** members
 - classes default to **private** members
- Access levels
 - **public**: anyone with access to an instance can see/modify
 - **protected**: can only access within the object (inherited)
 - **private**: can only access within the object (not inherited)
- Protect your data by making them private
 - Manipulate with “accessors,” a well-defined interface
 - Will lead to more boilerplate. That’s C++ for ya!

C++ Basics: Classes and Structs

- Anatomy of a class
 - **Constructor:** same as `__init__()` in python. Initializes your object
 - There is a “default constructor” that is special and takes no args
 - The compiler will make a trivial one for you, *if and only if* you don’t make one
 - Member variables: used to store data in the object
 - Exist for the lifetime of the instance of the object
 - Member functions: do things with the object
 - Access to all the member variables via “this->” pointer (self. in python)
 - Don’t need to pass “this” in, it’s always there in your class scope
 - **Destructor:** clean up so the object’s memory can be safely returned to the pool

C++ Basics: Classes and Structs

- [DEMO]
 - “Point” struct
 - Nested struct (Event)
 - Allocating/de-allocating for a little pointer practice
 - Upgrade Point to a class
 - Constructors and destructors
 - Basic member functions
 - Moving Point to its own source files
 - Organizing project code files: src/ and include/
 - Updating CMakeLists.txt for the new structure