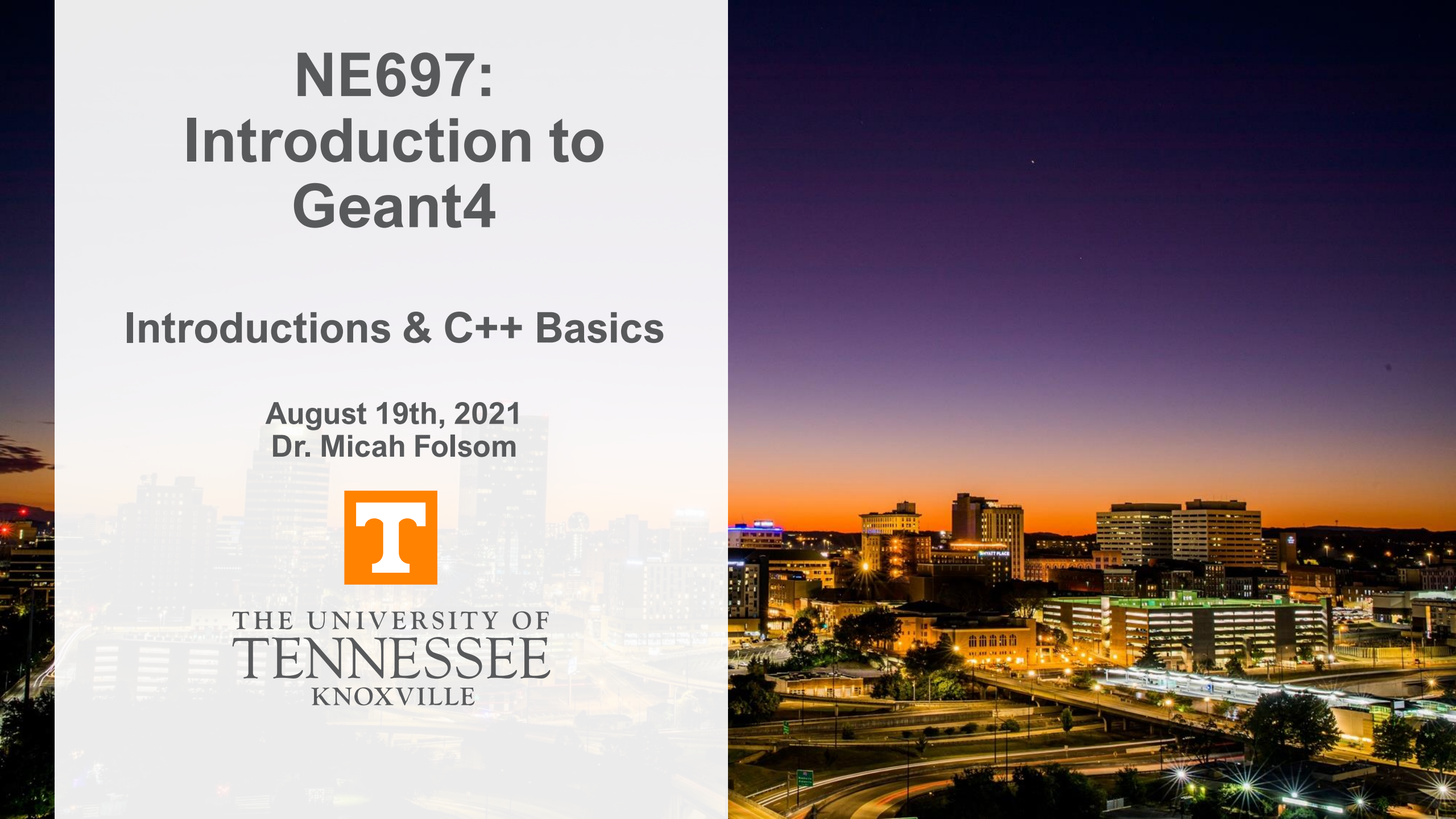
NE697: Introduction to Geant4

Introductions & C++ Basics

August 19th, 2021

Dr. Micah Folsom



THE UNIVERSITY OF TENNESSEE KNOXVILLE

Today's Agenda

- Introduce myself
- Student introductions
- Syllabus/course overview
- Linux server accounts
- Diving into C++ basics

Introductions

- UC Berkeley BSc, UT Knoxville MSc/PhD
- Engineer at LBNL in Applied Nuclear Physics
- Design and build portable detectors with contextual augmentation (LiDAR, IMU, cameras)
- Extensive C/C++ background, but I'm not a computer scientist
- Used Geant4 for PhD work and various side projects
- Learning Geant4 can be a **major bummer**
- I've wanted to teach this for years!

Student Introductions

- Name, major/department, year
- Programming experience? Geant4 experience?
- What would you like to use Geant4 for?

Course Overview

- Syllabus: on canvas (?)
  - Questions, comments?
- The goal is for you to learn to use Geant4
- I think the best way for you to learn is to write a bunch of code
- First time teaching this course, so open to changing as we go
- There's a **lot** to learn depending on your background
- I would like to make lectures as interactive as possible!
- Ways to connect and debug? Slack?

Linux Server

- nec-geant.ne.utk.edu
  - Just for our use. You shouldn't need sudo access
- Individual user accounts
  - Username is your NetID
  - I'll e-mail your passwords (/use vault.utk.edu)
- Geant4 installed in /usr/local/geant4
  - Might need to add "source /usr/local/geant4/bin/geant4.sh" to ~/.bashrc
  - Might be able to put this into the system template
- You're free to use another computer; this one will be used for grading

Editing Your Code

- I use **vim** because that's what the cool kids did in 1997 when I learned C
- If you've got Geant4 set up locally (recommended), use whatever editor or IDE you want – comfort is key!
  - **VSCode**, Eclipse, SublimeText, **emacs**, Code::Blocks, QtCreator
  - https://code.visualstudio.com/docs/remote/ssh
- We're going to use **git** to manage our code
  - Learn to commit frequently
  - It helps to think of coding jobs as a bunch of small sub-tasks
  - You're probably going to use it if you work with others

C++ Basics: Quick History

https://en.wikipedia.org/wiki/C%2B%2B

- Developed 1979 by Bjarne Stroustrup, “C With Classes”
- Standardized in 1998, new standards every 3 years
  - Big gap from 2003-2011
- The goal is to combine the performance of C with the development benefits of OOP and other abstractions

C++ Basics: Modern Times

- C++11 (~2011) was a game changer
  - **auto** keyword for lazy people like me
    - ThisReallyTerribleAwfulTypeName* var = new ThisReallyTerribleAwfulTypeName;
    - auto var = new ThisReallyTerribleAwfulTypeName;
  - Range-based for loops
    - std::vector<int> list = {1, 2, 3, 4};
    - **Old:** for (std::size_t i=0;i < list.size();++i) { <list[i] will be 1, 2, 3, 4> }
    - **New:** for (auto item : list) { <item will be 1, 2, 3, 4> }
  - Reference-counted pointers (not covered)
- Fixed-width types (std::uint8_t, std::int32_t, etc)
  - What size is **int**?

C++ Basics: Modern Times

- A bunch of **boost** was incorporated into the Standard Template Library (STL)
  - std:: namespace – even the old C stuff
  - #include <algorithm>
  - #include <numeric>
  - #include <random>
  - #include <thread>
  - #include <chrono>
  - #include <cmath>
- All of the basics are included

C++ Words of Caution

- Many design choices driven by support for backwards compatibility
- Decisions made by committee
- It's a fire hose – powerful if wielded correctly, a big mess if not
  - You turn on the nozzle and get 40 years of programming baggage
- Compiler warnings are your new best friend
  - A few can be deceiving, so really, Google/SO is your new best friend
- Be wary of over/pre-optimization. It's way more important that it 1) works and 2) is readable (comments!!)
  - Remember, whitespace doesn't matter, so use it

C++ Basics: Code Anatomy

- Code is split into the header (.h[pp]) and the implementation (.c[pp]) files
- Generally, *declarations* go in the header, and *definitions* go in the imp.
- **Declaration:** just the function signature
  - `int add(int a, int b);`
- **Definition:** signature + code
  - `int add(int a, int b) {`
  - `  return a + b;`
  - `}`

C++ Basics: Code Anatomy

- .hpp is **#include**'d from the .cpp
- As in most languages, things are case sensitive
- Unlike python, C++ doesn't care about white space
  - It's all about ;, {}, and ()
- "Pre-processor directives" (like include) start with #
  - Be aware of basics but we won't be using these much
- Generally, each class gets a hpp/cpp combo
  - Also, small namespaces (e.g. containing constants, functions)
- main() should be your only free-standing global function