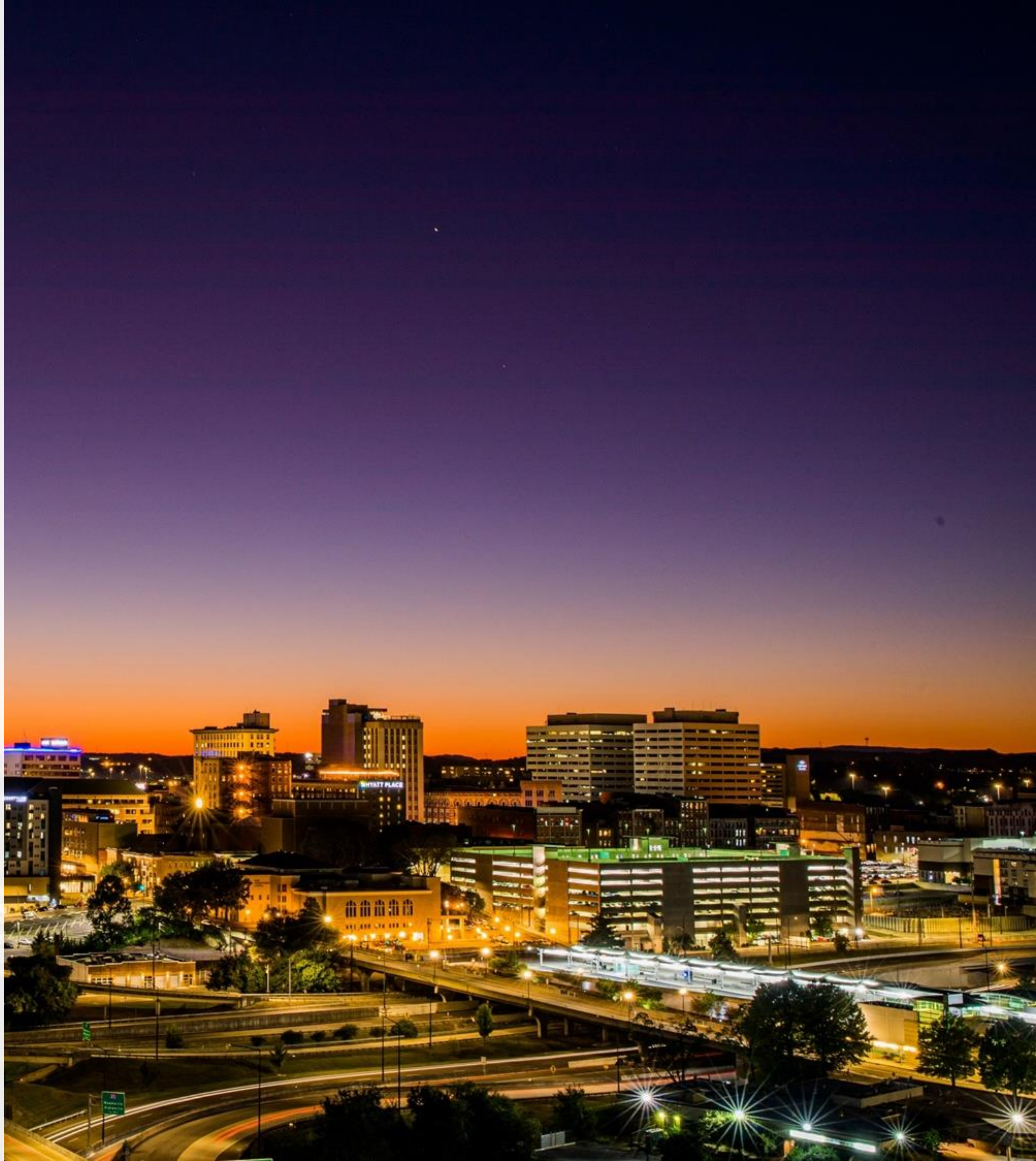
NE697: Introduction to Geant4

C++ Classes, Geant4 Intro

September 21st, 2021
Dr. Micah Folsom

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE



Today's Agenda

- Administrative items
  - Assignment feedback sent

- Finish outlining assignment 3

- Class inheritance

- Run Geant4 exampleB1

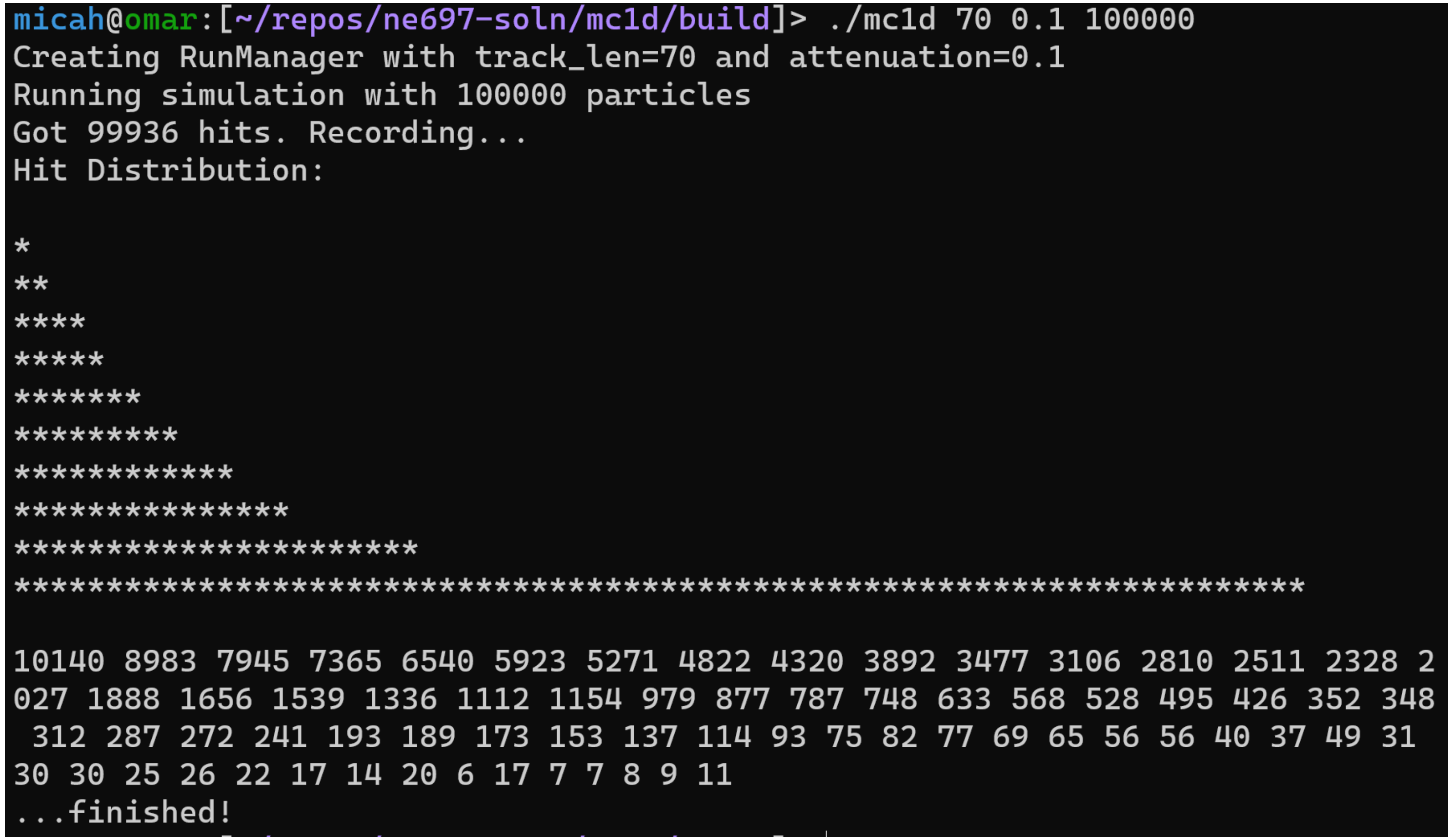
Last Time, On NE697…

- Custom operators
  - “Canonical implementations”: https://en.cppreference.com/w/cpp/language/operators
- Static class members
- Error handling & exceptions
- Useful C++ std:: containers

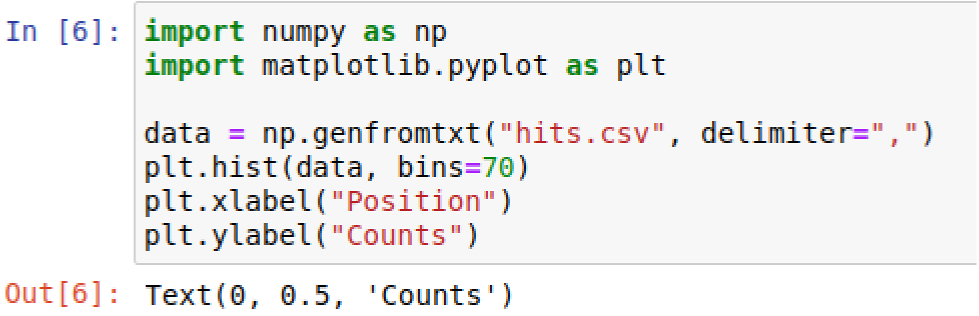
Assignment 3

- 1-D Monte Carlo code that transports a particle along a track
- Inputs
  - Track length, absorption probability, number of particles to run
- Physics
  - Just absorption with a per-unit-length probability
- Outputs
  - Summary of simulation
  - .csv file with hit information (each line is a hit index)
- Finally: make a histogram of the results (program of your choice)!

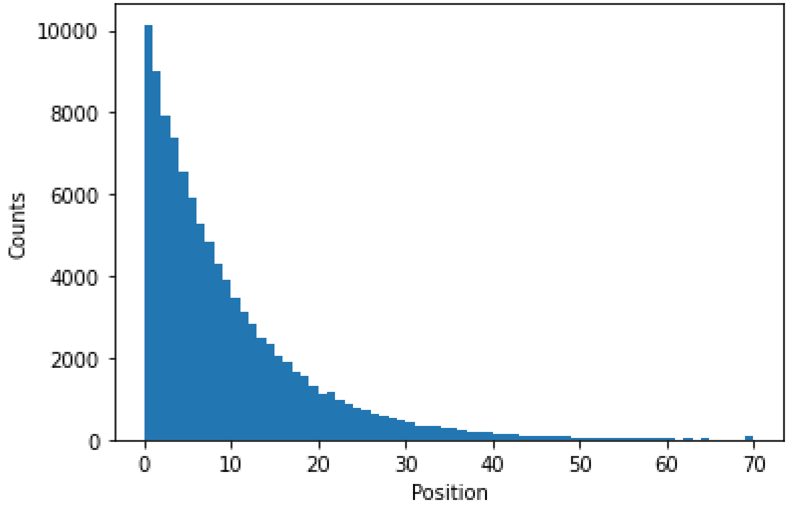
Assignment 3 Example Output



```
micah@omar:[~/repos/ne697-soln/mc1d/build]> ./mc1d 70 0.1 100000
Creating RunManager with track_len=70 and attenuation=0.1
Running simulation with 100000 particles
Got 99936 hits. Recording...
Hit Distribution:

*
**
****
*****
*******
*********
************
***************
**********************
**********************************************************************

10140 8983 7945 7365 6540 5923 5271 4822 4320 3892 3477 3106 2810 2511 2328 2
027 1888 1656 1539 1336 1112 1154 979 877 787 748 633 568 528 495 426 352 348
 312 287 272 241 193 189 173 153 137 114 93 75 82 77 69 65 56 56 40 37 49 31
30 30 25 26 22 17 14 20 6 17 7 7 8 9 11
...finished!
```



```
In [6]: import numpy as np
        import matplotlib.pyplot as plt

        data = np.genfromtxt("hits.csv", delimiter=",")
        plt.hist(data, bins=70)
        plt.xlabel("Position")
        plt.ylabel("Counts")

Out[6]: Text(0, 0.5, 'Counts')
```



Assignment 3

- Design approach – classes to define
  - **ArgParser**: consumes argc and argv[], becomes an object with getters for the 3 parameters (track length, absorption prob, and n particles)
    - Error-checks inputs
  - **RunManager**: manages our simulation. Consumes parameters from ArgParser
    - run(), write_results()
  - **Particle**: (class) object that we transport, keeps track of position, index, etc
  - **Hit**: (struct) object, just a record of an absorption
- We will use exceptions for error handling

Assignment 3

- It is a design choice to use exceptions and it allows us to design the classes differently
- Method 1: Return Error Codes
  - RunManager() constructed without args, then **run_manager.initialize(params…)**
  - If initialize() fails, it can return false or a non-zero error code, allowing us to recognize this in main() and exit the program
- Method 2: Using Exceptions
  - RunManager(params…) constructed with args; exception thrown if invalid
  - No need for initialize(); will try {} catch() {} and error-handle accordingly

Assignment 3

- [DEMO]
  - Setting up CMakeLists.txt and directories
  - Writing the ArgParser class
  - Sketching out main()
  - Outlining RunManager (but not implementing)
  - Particle and Hit .hpp and .cpp totally up to you to write
- Questions? Formal announcement will go out today
- Due next Tuesday, September 28th

C++: Class Inheritance

- Often termed as *polymorphism*
  - You can refer to a *derived* class using a pointer to the *base* class
- Great for defining common functionality and properties
  - All of these objects have a “name” member → derive from a common base
  - Avoid having to define a “std::string m_name” member in all objects
- In Geant4, it’s used to define the interface that the user leverages
  - G4VUserDetectorConstruction::Construct()
  - We can derive and then implement Construct() and Geant4 will call our version!

C++: Class Inheritance

- Must specify the access level of the inheritance
- Determines the access level in the derived class
  - **Public**: all public members stay public, protected stays protected (no change)
  - **Protected**: public becomes protected, protected stays protected
  - **Private**: public and protected become private
- You will almost always (*always* in Geant4) use public inheritance
- Methods must be *virtual* to use the polymorphic behavior
  - Be careful, if you forget, it will probably still compile

C++: Class Inheritance

- **Virtual** is a way to tell the compiler that the function in the *base class* will be superseded by the same signature in a *derived class*
- If you have a Base* to an instance of a Derived object, calling the inherited virtual function will call the Derived version
- “Pure virtual” functions
  - virtual void my_function() = 0;
  - This has no implementation! We cannot instantiate this object!
  - “Abstract base class”
  - We *force* the derived classes to implement this function

C++: Class Inheritance

- [DEMO]
  - Bases, abstract bases
  - Virtual functions, pure virtual functions
    - magnitude()
  - Point2D, Point3D
  - Using a Point3D instance with a Point2D pointer