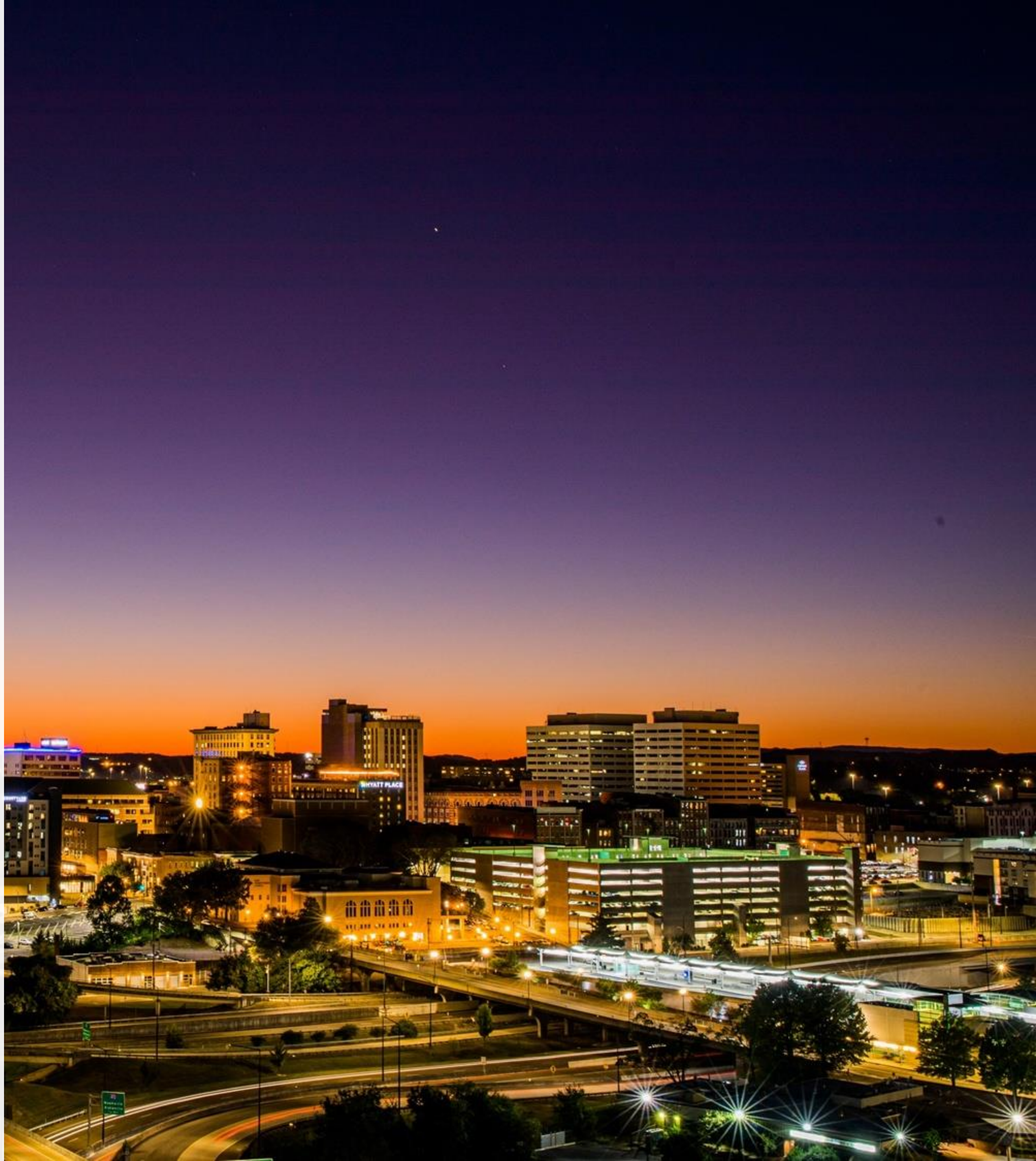
NE697: Introduction to Geant4

Introductions & C++ Basics

August 24th, 2021
Dr. Micah Folsom

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE



Today’s Agenda

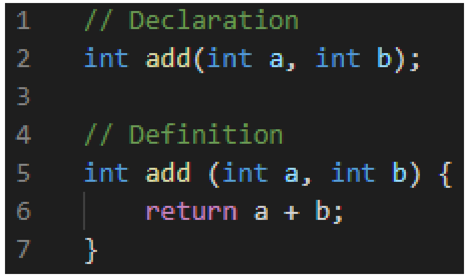
- Server access & Github
- Code editors
- C++ Basics
- VSCode Walkthrough

Server Access & Github

- Has everyone tried to log in?
  - Please change your passwords! (bonus: and set up an SSH key)
- I encourage you to develop and test on your own computers
- I'll be compiling and running on the server
  - If testing locally, make sure you're on the same version of Geant4
- Make a Github account (free)
  - Make a **ne697-geant4** repo and invite me (micahfolsom)

Last Time

- Reminder: language concepts are often very transferrable
- Declarations vs definitions



```
// Declaration
int add(int a, int b);

// Definition
int add (int a, int b) {
    return a + b;
}
```

- Header vs implementation

C++ Basics: Compiling

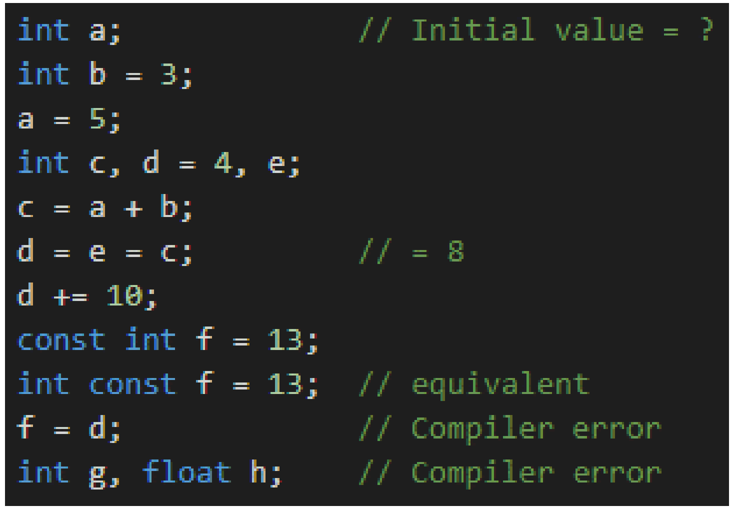
- For this class, we will use **g++** on Ubuntu 20.04
  - If you're on OS X, you'll encounter **clang**; it works similarly
  - C++ build system goal: build the g++/clang command & run it
- We will use **CMake** as our build system
  - This is the sane way to build C++ in 2021
  - Writing Makefiles is terrible, please don't do this to yourself
  - Geant4 uses it!
  - Most major IDEs will parse CMakeLists.txt
- Learning bash will help in general
  - https://learnxinyminutes.com/docs/bash/

C++ Basics: Compiling

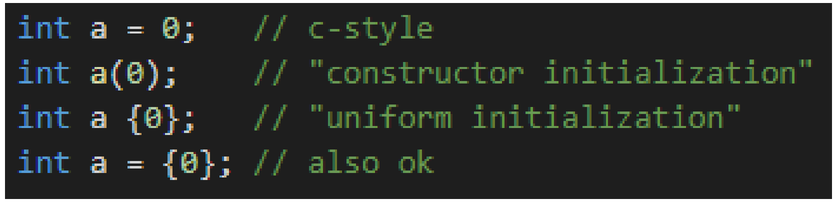
- Separate your **build** directory from your **code** directory
- **Compiler** options/args → **g++** (warnings, includes, lib linking, src lists)
  - Typically, these are somewhere inside CMakeLists.txt, or a .cmake file
- **CMake** options → **CMake** (build type, install paths, app features)
- Example:
  - g++ -std=c++17 ex.cpp -o ex
    - Extension doesn't matter, g++ will produce an executable binary
  - cmake –DCMAKE_INSTALL_PREFIX=/usr/local/ –DGEANT4_INSTALL_DATA=ON ../geant4-src
    - From build/, with CMakeLists.txt in geant-src/

C++ Basics: Declaring Variables

- <type> <variable name> = <value/expression>;
- Assignment goes from right to left
- +=, -=, *=, /=
- const applies to the left first, and if nothing is there, then right
  - This is why you'll see it in different places
- Always initialize your variables
  - Define when you declare if possible



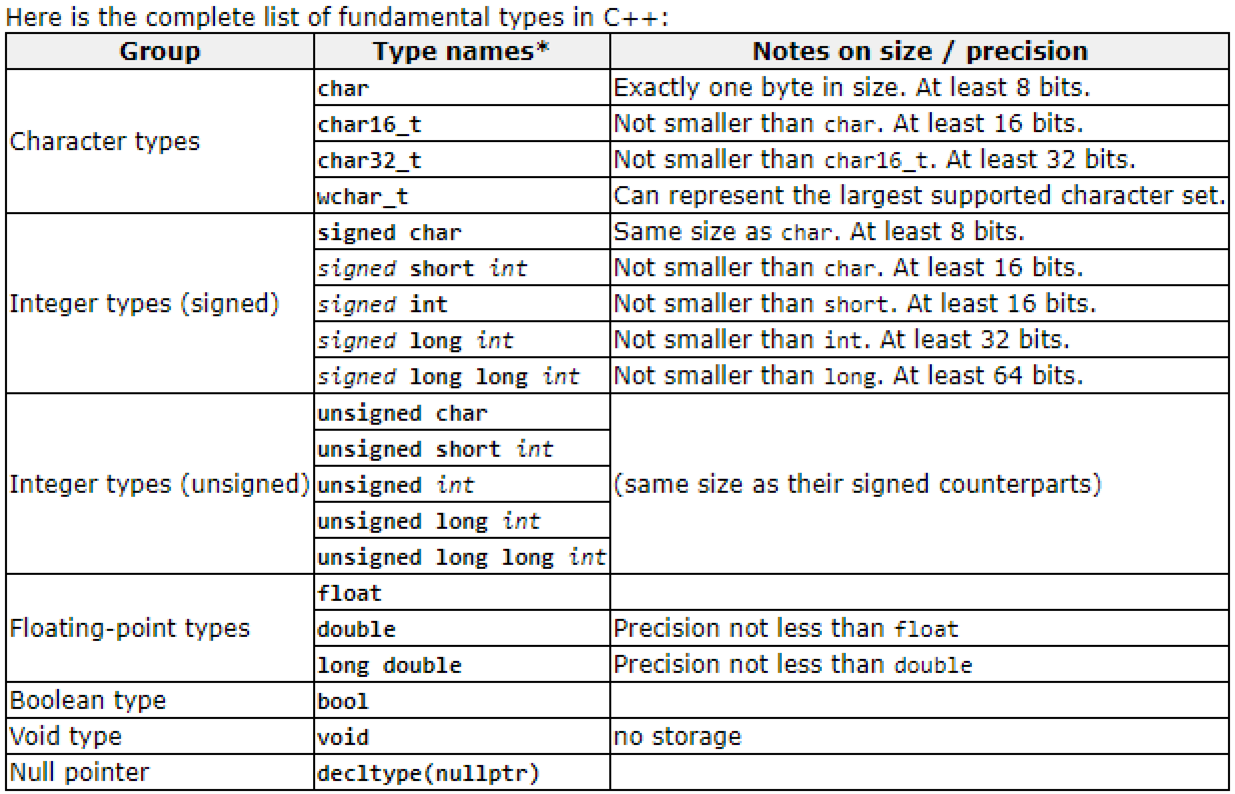
```
int a;            // Initial value = ?
int b = 3;
a = 5;
int c, d = 4, e;
c = a + b;
d = e = c;        // = 8
d += 10;
const int f = 13;
int const f = 13; // equivalent
f = d;            // Compiler error
int g, float h;   // Compiler error
```



```
int a = 0;   // c-style
int a(0);    // "constructor initialization"
int a {0};   // "uniform initialization"
int a = {0}; // also ok
```

C++ Basics: Types

- “signed” is optional
- Don’t worry too much about float vs double
  - I default to float until there’s a reason
- NULL == nullptr
- “at *least* N bits”
- char (somewhat rare), int, float, bool will be your workhorses
- The rest are all objects!



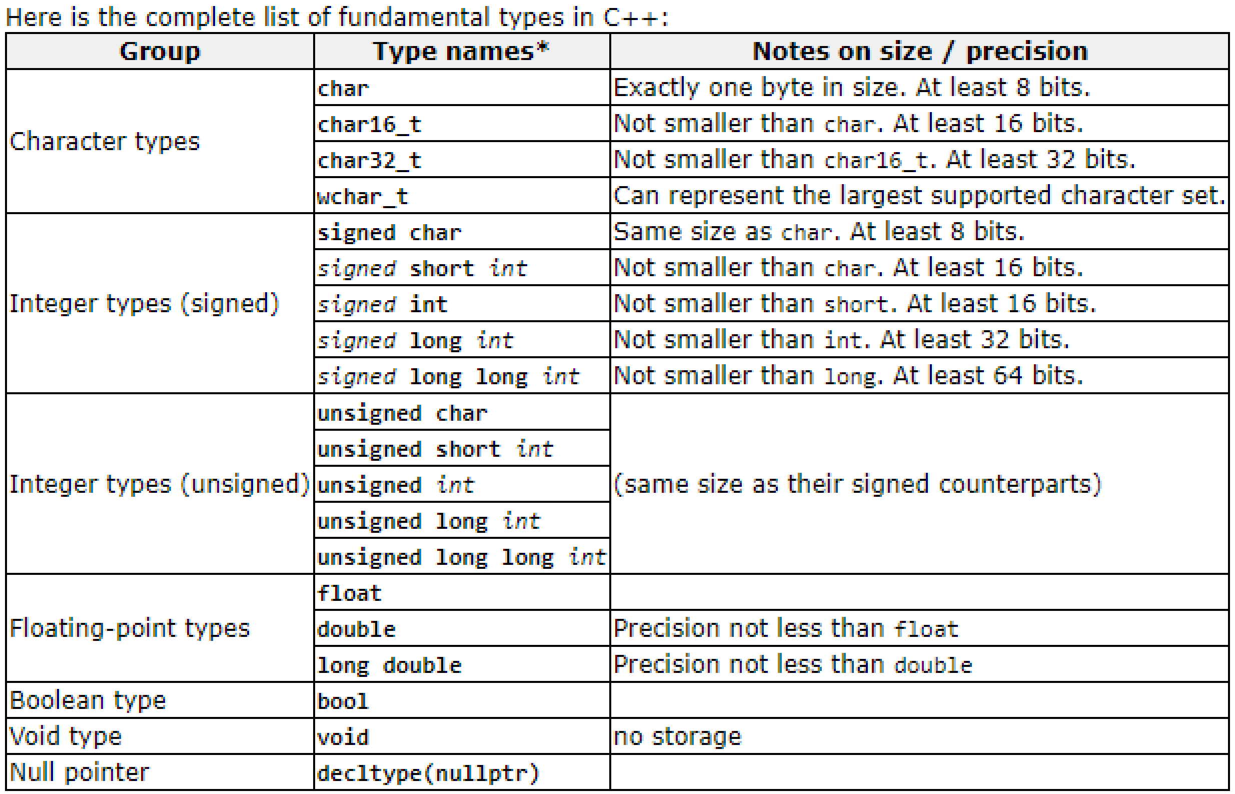
Here is the complete list of fundamental types in C++:

Group	Type names*	Notes on size / precision
Character types	`char`	Exactly one byte in size. At least 8 bits.
	`char16_t`	Not smaller than `char`. At least 16 bits.
	`char32_t`	Not smaller than `char16_t`. At least 32 bits.
	`wchar_t`	Can represent the largest supported character set.
Integer types (signed)	`signed char`	Same size as `char`. At least 8 bits.
	`signed` `short` *`int`*	Not smaller than `char`. At least 16 bits.
	`signed` `int`	Not smaller than `short`. At least 16 bits.
	`signed` `long` *`int`*	Not smaller than `int`. At least 32 bits.
	`signed` `long long` *`int`*	Not smaller than `long`. At least 64 bits.
Integer types (unsigned)	`unsigned char`	(same size as their signed counterparts)
	`unsigned short` *`int`*	
	`unsigned` *`int`*	
	`unsigned long` *`int`*	
	`unsigned long long` *`int`*	
Floating-point types	`float`	
	`double`	Precision not less than `float`
	`long double`	Precision not less than `double`
Boolean type	`bool`	
Void type	`void`	no storage
Null pointer	`decltype(nullptr)`	

https://www.cplusplus.com/doc/tutorial/variables/

C++ Basics: Types

- Honorable mention for **enums**
  - Typically, a collection of options or types of a thing
  - enum ParticleName { Gamma, Neutron, … };
  - ParticleName pt = Gamma;
  - if (pt == Gamma) {
  - …
- Treat basic **enums** like **unsigned ints**



Here is the complete list of fundamental types in C++:

Group	Type names*	Notes on size / precision
Character types	char	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	signed char	Same size as char. At least 8 bits.
	signed short *int*	Not smaller than char. At least 16 bits.
	signed int	Not smaller than short. At least 16 bits.
	signed long *int*	Not smaller than int. At least 32 bits.
	signed long long *int*	Not smaller than long. At least 64 bits.
Integer types (unsigned)	unsigned char	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	float	
	double	Precision not less than float
	long double	Precision not less than double
Boolean type	bool	
Void type	void	no storage
Null pointer	decltype(nullptr)	

https://www.cplusplus.com/doc/tutorial/variables/

C++ Basics: Constants

- Literals: literally the value in the code
  - 1, 5.0, “geant4”, true, nullptr
- Numeric literals
  - 1234, 0xA8, 6.022e23, 10.5, 3.14f
  - Suffixes: u/U = unsigned, l/L = long, ll/LL = long long, f/F = float, l/L = long double
- String literals
  - Single quotes for single characters (char val = ‘a’;)
  - Double quotes for strings (std::string val = “hello”;)

C++ Basics: Functions

- General form:
  - <return type> <function name>(arg1, arg2, …) [const];
  - int add(int a, int b);
  - void do_something();
- Can **just** define (no declaration) if only using in that one .cpp file
- Declaration in .hpp is needed for other .cpp files to know the signature
- **const** after is only for class member functions
- Don’t need to worry about lambda functions and function pointers

C++ Basics: main()

- Every C++ program starts with main()
  - int main(int argc, char* argv[])
  - int main(int argc, char** argv)
- Command-line args are passed in
  - **int argc** = argument count (minimum 1)
  - **char* argv[]:** array of char-strings, starting with the executable name (argv[0])
- Returns exit code (0=success)

C++ Basics: Namespaces

- Just a way to separate things, avoid naming conflicts, and make things explicit
- Good modern libraries all use them *cough*
- Scoped using {}: **namespace NS { <code> }**
- **Do not “using namespace <whatever>” in .hpp files**
  - Every file that includes it will now be polluted!
  - Generally ok in .cpp files, just be aware of potential conflicts
- Accessed with NS::<thing>
  - std::cout, Eigen::MatrixXd::MatrixXd

C++ Basics: Standard IO

- **#include <iostream>**
- **std::cout** pipes to the standard output (left to right)
  - std::cout << “Hello world” << std::endl;
    - **std::endl**: appends “\n” and flushes output stream
- std::cin pipes from the standard input
  - std::cin >> user_input;
  - **python**: user_input = raw_input()
  - std::cin >> a >> b >> c >> d;

C++ Basics: Standard IO

- C++ doesn’t have anything like python f-strings, sadly
  - print(f”My variable = {variable}”)
- std::string has limited formatting options
  - .find(), .replace(), .substr()
  - Can + together, along with std::to_string() (fancy sprintf())
- std::stringstream is your best bet
  - Syntax like std::cout and can handle basic types
  - Can use it to build your string, then access .str()
  - In Geant4, we can use G4String

C++ Basics: Hello, World

- main.cpp:
  - Includes
  - Namespace (optional)
  - main() definition
  - “Hello, world”
- Compile with: g++ main.cpp -o ex
- [DEMO]

VSCode Setup Demo