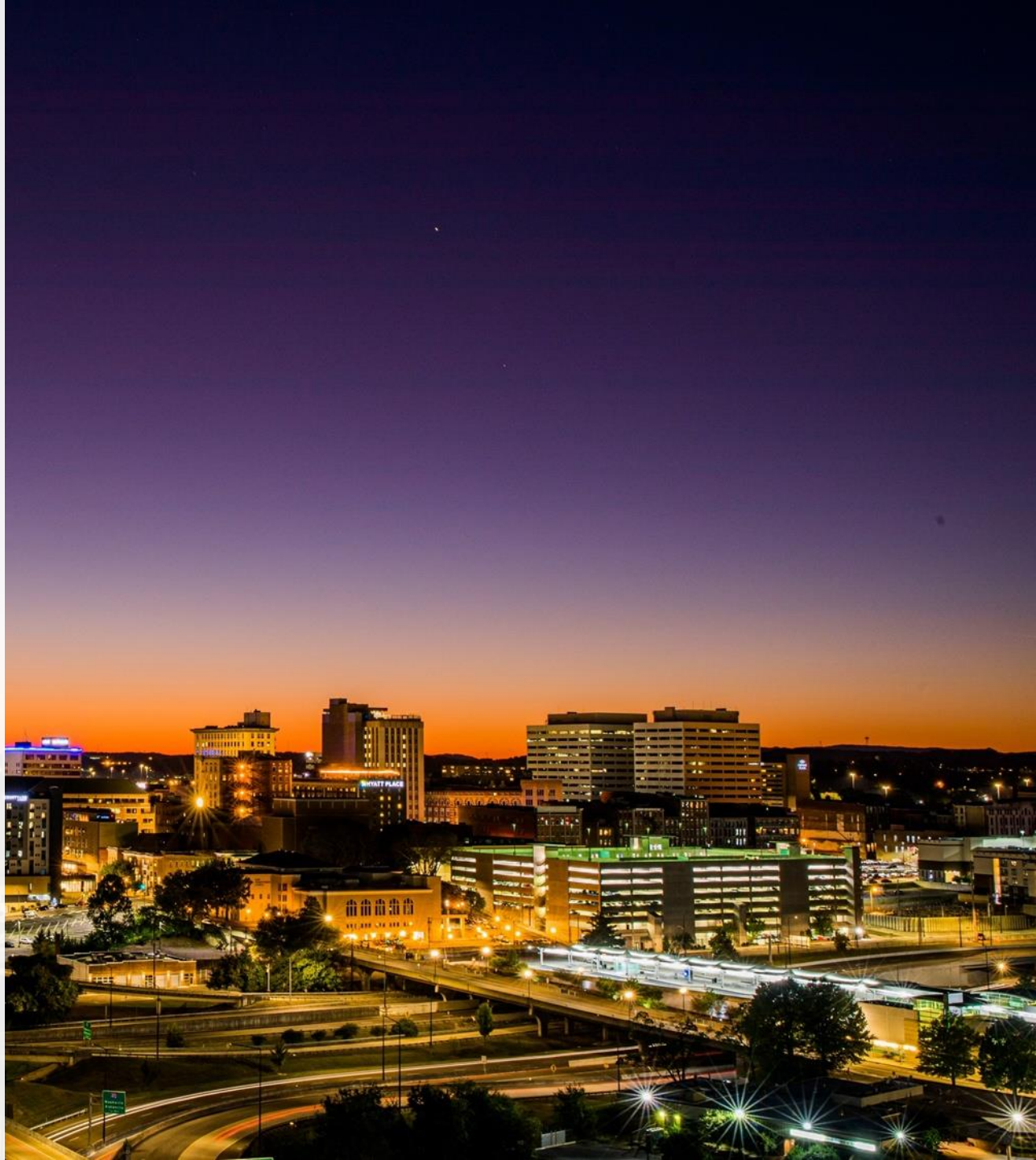
NE697: Introduction to Geant4

C++ Basics

September 2nd, 2021

Dr. Micah Folsom



Today's Agenda

- Server access for auditors – email sent
- Any other administrative items?
- Anything I should go over again before we dive back into C++?

Assignment 2

- Demonstrate the C++ development workflow we've learned using our new tools
  - Make a new folder in your Github repo called **assignment2**
  - Create a CMakeLists.txt file
  - Write a simple program (1 source file) that takes the first command-line argument *N* and prints 2^N to standard out
    - Headers for: power function and converting text to numbers
  - Commit your code and push to Github
    - Add your build directory to .gitignore if the folder is in the repo

Our C++ Tools So Far

- Basic types: char, int flavors, float/double, bool, nullptr, void
- Constants/literals
- Functions: int main(int argc, char* argv[])
- Namespaces
- Standard IO: std::string, std::cout, std::endl, std::stringstream

C++ Basics: Operators

- Assignment operator
  - =
- Math
  - +, -, *, /, %
- Compound assignment (operator, then assignment)
  - +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=
- Logical → bool
  - !, &&, ||
- Comparison/relational → bool
  - ==, !=, <, >, <=, >=

C++ Basics: Bitwise Operators

- Won’t need to use these unless you’re managing your own binary data
- Single & for bits, double && for logic
- AND: &
- OR: |
- XOR: ^
- NOT: ~
- Left/right shift by N bits: “<< N”, “>> N”

C++ Basics: Other Useful Operators

- sizeof()
  - sizeof(std::uint32_t) == 4
- Increment and decrement by 1
  - ++, --, can be prefix or suffix

```
a = 3;
b = ++a; // b = 4
b = a++; // b = 3
```

- C-style type-casting (kind of like an operator)
  - Be explicit, and communicate your intent

```
b = int(a);
b = (int)a;  // equivalent
b = a;       // implicit conversion
b = std::static_cast<int>(a);  // C++ style
c = std::dynamic_cast<int>(b); // c inherits from b
```

  - *b must have a virtual function

C++ Basics: Operator Precedence

Use parentheses to be explicit!

From greatest to smallest priority, C++ operators are evaluated in the following order:

Level	Precedence group	Operator	Description	Grouping
1	Scope	`::`	scope qualifier	Left-to-right
2	Postfix (unary)	`++ --`	postfix increment / decrement	Left-to-right
		`()`	functional forms	
		`[]`	subscript	
		`. ->`	member access	
3	Prefix (unary)	`++ --`	prefix increment / decrement	Right-to-left
		`~ !`	bitwise NOT / logical NOT	
		`+ -`	unary prefix	
		`& *`	reference / dereference	
		`new delete`	allocation / deallocation	
		`sizeof`	parameter pack	
		`(type)`	C-style type-casting	
4	Pointer-to-member	`.* ->*`	access pointer	Left-to-right
5	Arithmetic: scaling	`* / %`	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	`+ -`	addition, subtraction	Left-to-right
7	Bitwise shift	`<< >>`	shift left, shift right	Left-to-right
8	Relational	`< > <= >=`	comparison operators	Left-to-right
9	Equality	`== !=`	equality / inequality	Left-to-right
10	And	`&`	bitwise AND	Left-to-right
11	Exclusive or	`^`	bitwise XOR	Left-to-right
12	Inclusive or	`|`	bitwise OR	Left-to-right
13	Conjunction	`&&`	logical AND	Left-to-right
14	Disjunction	`||`	logical OR	Left-to-right
15	Assignment-level expressions	`= *= /= %= += -= >>= <<= &= ^= |=`	assignment / compound assignment	Right-to-left
		`?:`	conditional operator	
16	Sequencing	`,`	comma separator	Left-to-right

When an expression has two operators with the same precedence level, *grouping* determines which one is evaluated first: either left-to-right or right-to-left.

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

Flow Control

- if (*truthy expression*)
- else if (*truthy expression*) [note: two words!]
- else
- while (*truthy expression*)
- for (*initialization* ; *end condition* ; *increment*)
- for (*declaration* : *range*)
- Switch (*expression evaluating to a constant*)
- Dishonorable mentions: do while, goto
- Watch out for one-liners! I always use {}
- [DEMO]

Arrays

- We could spend entire lectures talking about arrays
- Arrays are for when you expect the number of elements to be fixed
  - You often know how many ahead of time (but not always)
- Two types
  - C-style: **int arr[10];**
    - arr is a pointer to the first element
    - **int arr[] = {…10 values…}** will technically work. Do not do this.
  - C++-style: std::array<int, 10> arr;
    - **arr is an object!** .empty(), .fill()
    - Can get the C-array with .data()

Vectors

- Dynamically-sized array (literally a C-array underneath)
- For when you don’t know what size it will be and/or it will change
- Resizing means copying, so avoid where possible
- std::vector<T>
  - T = whatever type you want
  - std::vector<float> vec;
  - std::vector<float> vec(10);
  - std::vector<float> vec(10, 5.0);
  - std::vector<float> vec = {5.0, 4.2 }

Arrays + Vectors

- [DEMO]
  - Instantiating each
  - Accessing elements
  - Iterating over the elements
    - Standard for-loop
    - Ranged-based for-loop
    - For-loop with iterators (pointer/C-style)
  - Multidimensional arrays