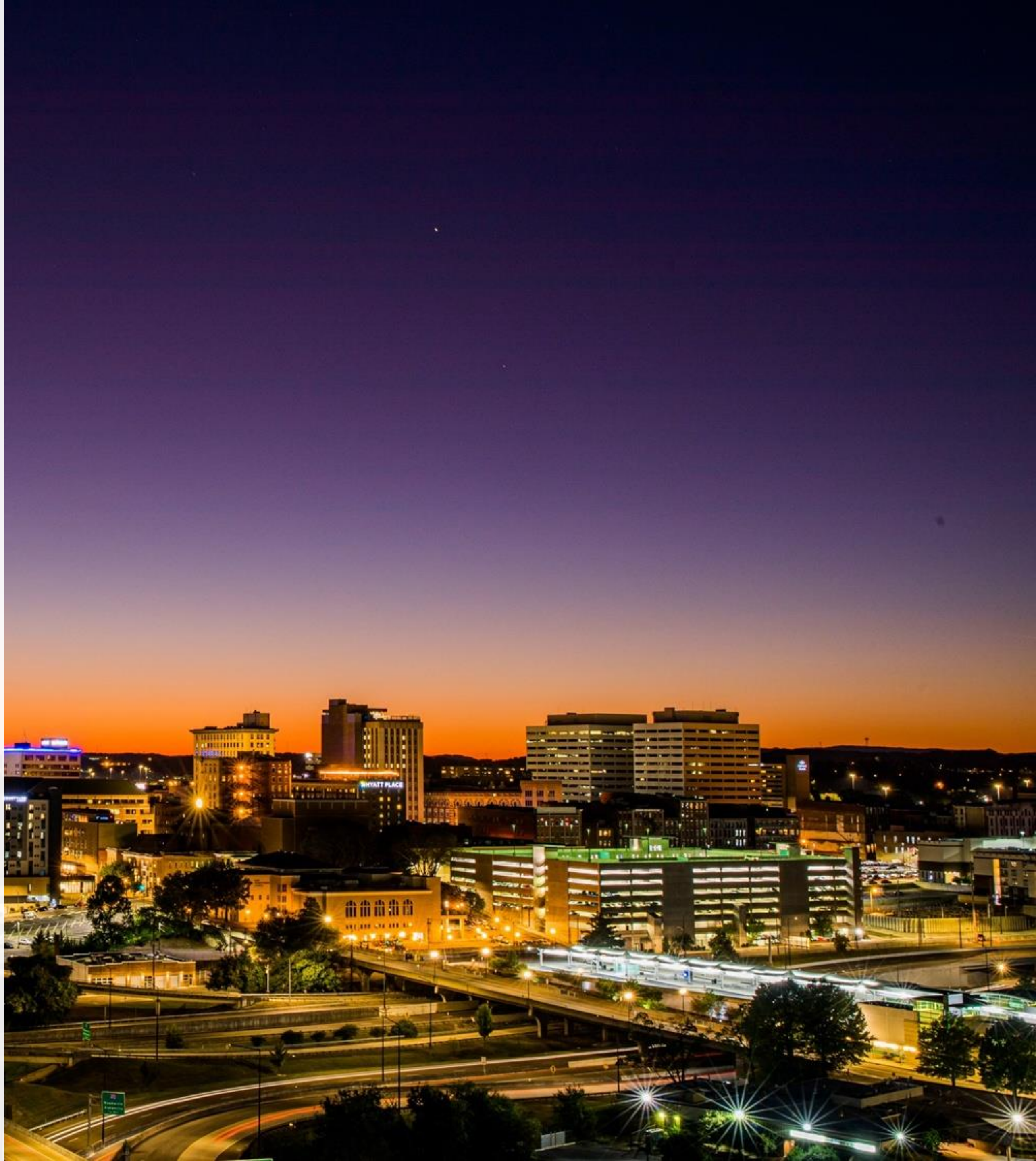
NE697: Introduction to Geant4

C++ Classes

September 16th, 2021
Dr. Micah Folsom

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE



Today’s Agenda

- Administrative items
  - Videos for auditors uploaded

- Assignments graded, need to post and send out feedback

- Some more C++, and then assignment 3

Last Time, On NE697…

- Small-ish correction on pass-by-const-reference
  - Suboptimal for primitive (built-in) types (int, bool, float, etc)
    - Compiler will do special optimizations because it’s smarter than us
    - They’re generally smaller than a pointer (=address)
      - Remember, my int was 4 bytes but my pointer to it was 8
  - Everything I said is still true for non-primitive types
- [DEMO]
  - Point class revisited, with the above in mind

C++: Static Class Members

- **static** members are shared by all instances of the class
  - In fact, we don’t even need an instance!
  - There’s just 1 copy of that variable or function
  - Use the class scope to call: Point::MAX_VALUE, Point::random()
- Useful for related constants or functions that don’t need to operate on an instance
- **static** is just a special keyword, we can apply it to any variable, type, etc
- **[DEMO]**

C++: Custom Operators

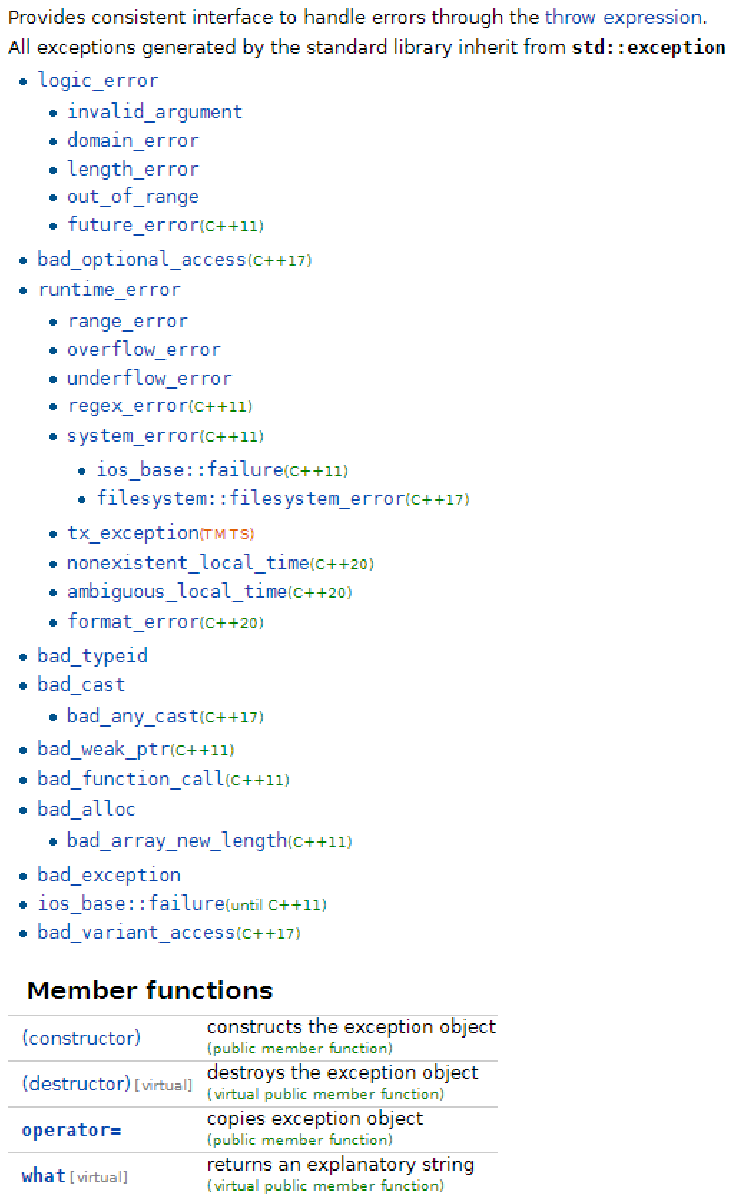
- What if we want to add 2 **Points**?
  - pt1.add(pt2);
    - Does this modify pt1, or return the result of pt1 + pt2?
  - pt3 = pt1 + pt2;
  - pt1 += p2;
- We can define our own +, -, <, ==, (), and more
- Still just member functions
- [DEMO]

C++: Error Handling

- C-style: return codes
  - int my_function(float& arg1, bool& arg2);
    - Note the pass-by-refs; we’re using the return value for the code, so this is the only way we can communicate back to the caller
  - Returns an error code
  - Somewhere else, there’s a bunch of “#define ERR_INVALID_ARG 1”
    - Then, in the code, you do “return ERR_INVALID_ARG;”
- C++-style: exceptions
  - #include <exception> or #include <stdexcept>
  - Exceptions are “throw”n and “catch”ed

C++: Exceptions

- throw [exception]
  - Same as raise in python
- throw std::invalid_argument(“Must supply 2 arguments!”);
- throw std::out_of_range(“Index out of range”);
- try { … } catch ([exception]) { … }
- catch (std::exception const& ex) {}
- catch (std::invalid_argument const& ex)



Provides consistent interface to handle errors through the throw expression.

All exceptions generated by the standard library inherit from **std::exception**

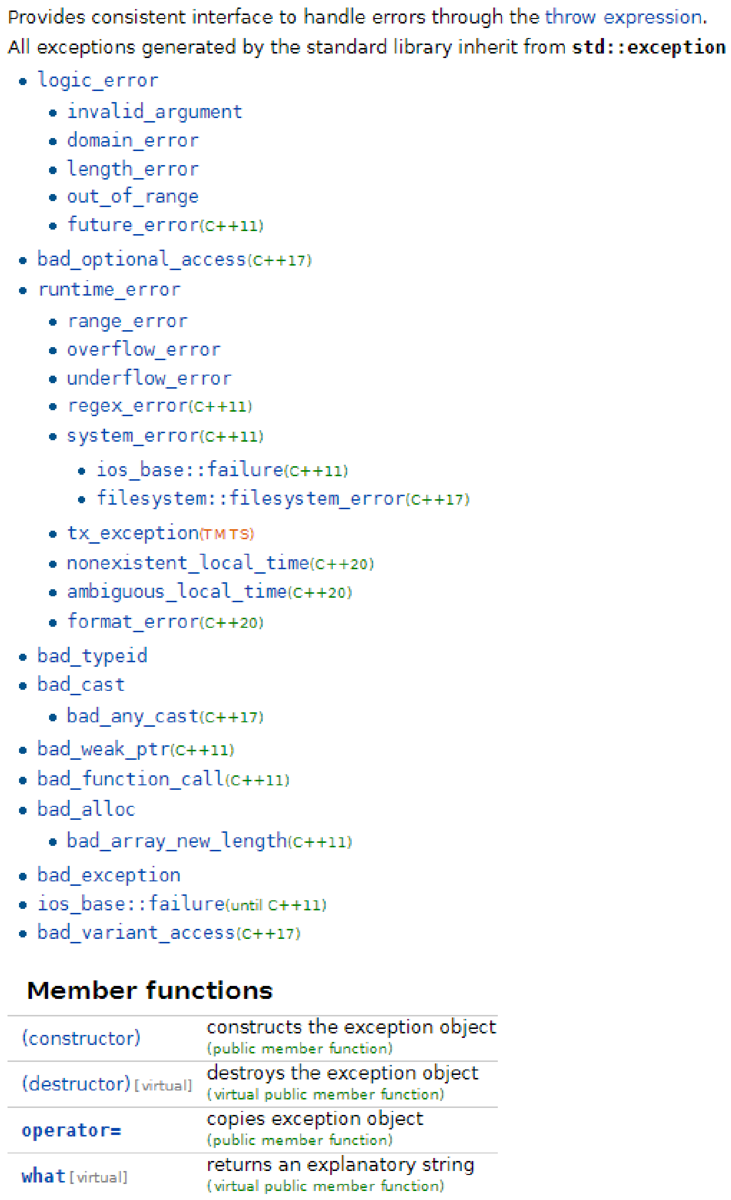
- logic_error
  - invalid_argument
  - domain_error
  - length_error
  - out_of_range
  - future_error(C++11)
- bad_optional_access(C++17)
- runtime_error
  - range_error
  - overflow_error
  - underflow_error
  - regex_error(C++11)
  - system_error(C++11)
    - ios_base::failure(C++11)
    - filesystem::filesystem_error(C++17)
  - tx_exception(TM TS)
  - nonexistent_local_time(C++20)
  - ambiguous_local_time(C++20)
  - format_error(C++20)
- bad_typeid
- bad_cast
  - bad_any_cast(C++17)
- bad_weak_ptr(C++11)
- bad_function_call(C++11)
- bad_alloc
  - bad_array_new_length(C++11)
- bad_exception
- ios_base::failure(until C++11)
- bad_variant_access(C++17)

Member functions

(constructor)	constructs the exception object (public member function)
(destructor) [virtual]	destroys the exception object (virtual public member function)
operator=	copies exception object (public member function)
what [virtual]	returns an explanatory string (virtual public member function)

C++: Exceptions

- Geant4 uses exceptions for error handling
- [DEMO]
  - Throwing exceptions
  - Catching exceptions



Provides consistent interface to handle errors through the throw expression.

All exceptions generated by the standard library inherit from **`std::exception`**

- `logic_error`
  - `invalid_argument`
  - `domain_error`
  - `length_error`
  - `out_of_range`
  - `future_error`(C++11)
- `bad_optional_access`(C++17)
- `runtime_error`
  - `range_error`
  - `overflow_error`
  - `underflow_error`
  - `regex_error`(C++11)
  - `system_error`(C++11)
    - `ios_base::failure`(C++11)
    - `filesystem::filesystem_error`(C++17)
  - `tx_exception`(TM TS)
  - `nonexistent_local_time`(C++20)
  - `ambiguous_local_time`(C++20)
  - `format_error`(C++20)
- `bad_typeid`
- `bad_cast`
  - `bad_any_cast`(C++17)
- `bad_weak_ptr`(C++11)
- `bad_function_call`(C++11)
- `bad_alloc`
  - `bad_array_new_length`(C++11)
- `bad_exception`
- `ios_base::failure`(until C++11)
- `bad_variant_access`(C++17)

Member functions

(constructor)	constructs the exception object (public member function)
(destructor) [virtual]	destroys the exception object (virtual public member function)
`operator=`	copies exception object (public member function)
`what` [virtual]	returns an explanatory string (virtual public member function)

C++: Useful Containers

- std::array<Type, Size>
  - Fixed size. Generally, for when you know the size at compile time
- std::vector<Type>
  - Resizable. For when you don’t know the size, or it’s changing (.push_back())
- std::map<KeyType, ValueType>
  - Like a python dictionary, but Types are fixed
  - my_map[key] = value;
- std::queue<Type>
  - First-In-First-Out: push(), front(), pop(), and empty()

C++: Useful Containers

- [OPTIONAL DEMO]
  - Map declaration syntax
    - Looping with iterators? Could be useful, I think we'll see this in Geant4 in a few places

- [NOTE]
  - Even though I use these somewhat frequently, I still end up looking up the exact member functions
  - When looking stuff up, I prefer **cppreference.com** over cplusplus.com

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

- 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 outline of the ArgParser class
  - Sketching out main()