C++ course (2020) Homework Assignments

Please submit your solutions on github (or any other git) as cmake projects. Please do not commit build directories or IDE-specific files! Commit source files and CMakeLists.txt only.

Part 1: C++ basics

1.1 (cin, cout, if, for) Enter an integer number n from the console. Print all prime numbers to n inclusive. For example, if n=19 or 22, then print 2, 3, 5, 7, 11, 13, 17, 19.

1.2 (Built-in types, bit shift, vector) Enter a 64-bit (8 byte) unsigned number (unsigned long long) from the console. Add each of its 8 bytes to vector<uint8_t>. Then print the vector in hexadecimal. Hint: bit shift operators might help.

1.3 (For the *bright* people only !) Write a quine in C++: The program which prints (exactly) its own text into stdout. Check the results with diff or fc.

1.4 (Built-in types) Write functions
char toUpper(char c)
char toLower(char c)
Which converts c to upper or lower case respectively if c is a latin character, otherwise c is not changed (important). Do not use any standard C/C++ functions for upper/lower case, work with ASCII codes directly!

1.5 (Modules, class, namespaces) Write a program with at least one module (2 files fun.cpp, fun.h) and at least one class (2 files MyClass.cpp, MyCLass.h). Put everything into namespace Fun. Write a function main() in main.cpp which uses MyClass and fun. Build all this with cmake.

Part 2: Classes. Inheritance.

2.1 (Class basics) Write your own class in 2 files, e.g. MyClass.h, MyClass.cpp. The class should include >=2 private fields with getters+setters, >=2 constructors, >=2 other methods, >=1 static fields, >=1 static methods. Implement some methods in the .h file, the others in .cpp file. Show the usage of the class in main().

2.2 (Inheritance) Create a simple architecture with inheritance: Interface or abstract base class, at least 2 children of the base class, and at least 2 grandchildren. Base class shall be either

Monster, Weapon, Deity or Hero for the cool people, or Person, Student, Employee for the uncool ones. Define in the base class and implement in children only: >=2 virtual methods. Override some in grandchildren. Use protected fields. Children and grandchildren shall have constructors which initialize all meaningful protected fields. Don't forget virtual destructor.

2.3 (Optional, for self-learning) Create a diamond problem example with virtual inheritance.

Part 3: Streams. Smart Pointers. Containers.

3.1. (IO streams, strings) Create a program to parse a config file in property=value format (some particular simple file, not the generic one) and write output as JSON. Require a few fields of different types. Handle spaces and newlines correctly. Allow for quoted strings in properties. Use only ifstream, ofstream, not any external libraries! Use string methods and/or string streams when needed.

For example:
Input file:

```
name =   'Brianna'

weapon=Lightsaber
age   =   19
```

Should give the output:

```
{
  "name" : "Brianna",
  "weapon" : "Lightsaber",
  "age" : 19
}
```

3.2 (IO streams, strings) The opposite: Read JSON, write properties.

3.3. (unique_ptr, move) Create a correct unique_prt lifecycle example (e.g. unique_ptr<string>): Source (function that creates a unique_prt), filter (function that processes it) and sink (function that prints and destroys it).

3.4 (shared_ptr, weak_ptr) Create a double-linked list container class DLList by using shared_ptr, weak_ptr and node structure Node. Implement methods find, insert, erase, size. Position in the list can be specified by either number or shared_ptr<Node>. Optional, for the

bright people: 1. Implement your own iterator type for this container and iterator-based find, insert, erase. 2. Make DLList a template over content type, e.g. DLList<std::string>.

3.5. (std::map) Create your own example (e.g. phonebook or something like this) which uses std::map<Record>, where Record is some data structure. The example should include insert, delete and search operations, plus printing the whole map.

3.6. (std::deque) Create your own moving average filter class, which keeps the last n-values in a std::deque container.

Part 4: Classes 2 + Other stuff

4.1. (threads, lambdas) Create some numerical algorithm of your choice, like sum of many numbers or something more interesting. Parallelize this algorithm with std::thread. Check that the results are identical to the serial version. Use mutex and/or atomic variables. Do the same with std::async, with result returned in e.g. std::future<double>. Use lambda expressions to define your threads.

4.2. (Operator overloading) Create class Vector3D with double fields x, y, z. Define all usual arithmetic operators for it, including vector*number, and vector*vector (dot product). Don't forget the operators +=, -=, *=, /=.

4.3. (Templates) Create the ring buffer RingBuffer<T> template class of size n. Keep data in a std::vector field.

4.4 (Optional, for bright people) Implement RingBuffer2<T> using raw pointers, new[] and delete[]. Implement correct copy, move and swap operations for this class.

Course project

Implement a computer vision project of your choice with C++ and OpenCV. Try to use some cool C++ features in your code, if possible (classes, containers, shared_ptr, threads, lambdas etc.). Ugly stuff is not allowed (global variables, raw pointers). People who did OpenCV + C++ projects before don't have to do this.