ICS 46 (Shindler) Spring 2022
Project #0: Getting to Know the ICS 46 VM

Due date and time: **April 5, 2022 at 8:29 AM**.

If you took Professor Thornton's ICS 45C, the VM should look *very familiar* to you, as will these setup instructions. However, this is not the same thing as Project #0 from when you were in that class, nor is it the same as the Project #0 from his ICS 46, so you still have some work to do.

In fact, many of this assignment instructions are drawn from his similar assignment for ICS 46[1] and he was instrumental in helping me set up the ICS 46 VM. That having been said, you should still read these directions, as they may have changed from when you took ICS 45C, even if he was your teacher for that class.

Remember:

- You need to download the VM for this class, not the one for Prof. Thornton's ICS 46
    - https://www.ics.uci.edu/~mikes/ics46/vm/ics46-shindler-2022spring.ova
- You need to do this project, not the one for Prof. Thornton's ICS 46
- Prof. Thornton is also teaching ICS 46 this quarter; if you are enrolled in my (Shindler's) lecture, please remember to do the projects for my class, not for his.

After downloading the VM, follow the instructions here for setting up the VM. https://www.ics.uci.edu/~thornton/ics45c/ProjectGuide/Project0/. One key difference, though: stick with the "Installing VMware and the x64 version" instructions. At the moment, we are not set up to support M1 Macs. Stop when you get to the section describing the program that the class writes: you aren't writing a block pyramid for ICS 46. Also, note that the username and password for this VM are "ics46" and not the other course's set.

For students who wish to work in a quiet environment on campus, I recommend loading the VM onto a USB "thumb" drive and taking it to the ICS third floor computer lab. I also recommend you be very careful in backing up your work in progress, no matter where you are doing this, but especially if you are working via thumb disk in a computer lab.

The Program's Requirements

As a warm-up, this project asks you to write and submit a short C++ program. The program itself isn't actually the interesting part, though it's one that you might find takes you a little bit of time to write. The main goal here is to be sure you're able to use the ICS 46 VM to do your work, that you learn what you need to know about one of the available text editors to write your program, and that you use the provided tools to gather your files for submission. Even if you normally prefer a different working environment, you would be well-served to use the ICS 46 VM for this project, to be sure that you can use it for your work later in the quarter. Also, if your project does not compile and run in the VM, we will not be able to grade it, and that will cause you to get a zero. Every quarter this happens to some students who, to put it gently, are not pleased with their grade on any such project. Please do not let this happen to you!

---

[1] You know how I said that if you get significant but allowable help from someone else on an assignment, you should cite who and what clearly? That rule applies to me too.

Creating a new project on the ICS 46 VM

If you have not already done so, run the command **ics46 refresh** to be sure you have the current version of the environment needed. Typically, you will do this every time a new project gets released.

At a shell prompt in your VM, issue the command **ics46 start proj0 project0**. This will create a new project in the directory **~/projects/proj0**, using the **project0** template. Having done that successfully, change into that directory (**cd ~/projects/proj0**) and you're ready to get started!

Compiling and running your program

Whatever C++ code you write should be placed into the **app** directory inside your project directory. When you're reading to compile it, change into the directory **~/projects/proj0** (if you're not already there) and issue the command **./build app**. (Note the dot and the slash in front of the word "**build**", and the space between "**build**" and "**app**"; those are important.) If compiling is successful, then issue the command **./run** to run your program.

If you want to run the unit tests, which your professor advises as a great way to test programs, use the command **./build gtest** and **./run gtest**. This will run a series of unit tests. Some starting point tests are provided for you in directory gtest, file tests1.cpp. These are a **starting point, and are not intended as comprehensive tests**. While getting less than 100% of these to pass will guarantee you do not get 100% on the project, passing each of these does not make any similar guarantees for the project itself. You are encouraged to test your code comprehensively. **For the portion requiring templating, be sure to create cases that involve non-numeric types, such as std::string.**

A good idea is to, after reading the requirements, think about what test cases your code would need to pass for you to be satisfied that you have written a good project. Then, add these test cases to tests1.cpp -- yes, even before you have written code to make them pass! The starting point code should help you see the format that a Google Test case expects, and the professor would be happy to help you if you are unable to do this on your own (make a private post on Ed with your desired test case(s) described and we'll go from there). That way, when you are writing your program, you can check at any point if these cases have yet passed. In fact, some students write their test cases first, and then write their code to *just pass the next test case that is not yet passing, without "breaking" any previously passing ones*. Then they backup their work (such as to a private git repository) and then continue. This is known as *test driven development* and is a very good software engineering practice.

The Program's Requirements

Part one: puzzle verification. Consider combinatorial puzzles, like those presented in lecture during lecture two. For example, one such puzzle is POT + PAN = BIB. In these puzzles, you must assign each letter a distinct digit, such that if we substitute the digits in place of the letters, the resulting mathematical equation is true.

In our continuing example, if we set P=2, O=3, T=1, A=7, N=4, B=5, and I=0, we get 231 + 274 = 505.
In the file proj0.cpp, you will find an incomplete implementation for:

`bool verifySolution(std::string s1, std::string s2, std::string s3, const std::unordered_map<char, unsigned> & mapping)`

The first part of your assignment is to finish implementing this. To find out what the proposed mapping has for s1[0] (the first character in std::string s1), you can use the member function **at()** of the **unordered_map** class: that is, **`mapping.at(s1[0])`**.

You **are not** required to check that the digits assigned are distinct, merely that it satisfies the equation. Note that s1, s2, and s3 might not be equal in size. Furthermore, you may assume that all characters appearing in any of strings s1, s2, and/or s3 will appear in the mapping; you do not need to check for that.

There are many ways to solve this problem. If you are having trouble finding *any* way to write a solution to it, please see a member of course staff. You may not use the standard library for any portion that solves a significant portion of your assignment, other than making use of accessor methods in the `std::unordered_map` class. Standard library portions that do not solve a significant part of the assignment, such as the `std::atoi()` function, are allowed. You are not required to use that function and many correct solutions without it exist.

If you want to test, and you should, two test cases have been provided in the gtest folder. Build your project (either fully or just the testing portion) and run the testing portion to see. While I will have other tests when I grade it, I promise these two will be part of it. *In general, I will provide you with some test cases for every project. However, it is not guaranteed that passing these cases is sufficient for a 100% on the assignment. You still need to develop your own test cases. Testing your code is an important part of the assignment, not a mere afterthought.*

Part two: A linked-list based queue data structure. The concepts of a queue and linked list are available in your textbook's first assigned reading, and of course, we are available to answer any questions you may have. To complete this, you will need to fill in LLQueue.hpp by adding the required functions. We have a few requirements.

- Your implementation must be based on the idea of a linked list. This must be a linked list that you implement.
- Your implementation must fit the interface given
- Your implementation must be templated
    - An implementation of LLQueue that does not properly template may cause you to get a zero on this portion assignment. **Do not assume that working for ints is sufficient, despite those being the only cases in the provided test cases.** Test with non-numeric data types as well!
    - **No, really; pay attention to that warning. Students may end up with zeroes for this project if their code is not properly templated, regardless of other considerations.**
- When attempting to access the front of an empty queue, or to dequeue from an empty queue, you must throw a QueueEmptyException as appropriate.
    - Do not throw an exception upon addition of an element: there is no capacity (beyond what memory allows).
    - **Please note:** the project will not build successfully until you have at least stubbed code for the Queue functions.

- You also need to write a copy constructor and an assignment operator. These must be “deep copies” -- you should end up with a second queue that has the same contents as the first, not merely two objects each with a pointer to the front of the same linked list.

You are prohibited from using any standard library container classes Using a standard library container, including std::vector or std::queue, in implementing your LLQueue will be treated as a *serious academic integrity violation.*

When we grade this, any test case that takes more than two minutes to run on the instructor’s (reasonably modern) computer may be deemed an unsuccessful test case.

Gathering and retrieving files from the ICS 46 VM for submission

When you've completed your work on this project, you'll need to submit your C++ source and header files to us. Because we'll be using some automated tools to test your work, it's important that everyone submits their work in the same format — files arranged in the same way, in the same directories, etc. — so I've provided a script called **gather** in your project directory that will gather up the necessary files, arrange them in the way that we expect, and prepare a file called **project0s22.tar.gz**, which is the only file you'll need to submit.

The **gather** script will gather all of the files in your **app** and **gtest** directories and no others, so be sure that anything you want to submit is in the **app** or **gtest** directories before running it. Additionally, only files meeting certain characteristics will be gathered, even if they're in these directories:

- Only files whose names end in either **.hpp** or **.cpp** are gathered.
- Only files whose sizes are not more than 128KB are gathered.

Files not meeting these characteristics are extremely unlikely to be things that we're interested in — they'll generally be temporary files generated by your editor or development environment, for example, which has led to students attempting to submit hundred-megabyte submissions for what are otherwise small programs.

Make note of the list of files that the **gather** script picked up; they'll be shown to you. It's your responsibility to ensure that nothing is missing, and "I wanted to submit this file but **gather** didn't pick it up" does not constitute an excuse to submit late work.

What's a .tar.gz?

A **.tar.gz** file (sometimes called a "tarball") is a compressed archive format common on Linux and various other flavors of Unix. (The **.tar** means that it is an archive of files; the **.gz** means that the archive has been compressed using an algorithm called gzip.)

Submitting your work using the ICS 46 VM's GUI

When you're ready to submit your **project0s22.tar.gz** file, you'll want to submit it to our web-based system called Checkmate (described below). The easiest way to do this is to use the web browser (Firefox) built into the ICS 46 VM, which will allow you to log into Checkmate and submit the file as you would on your host operating system.

Alternatively, you might prefer to retrieve the file from your VM and submit it from your host operating system; if so, see the next section for instructions on how to retrieve the file. (It's not a bad idea to learn how to do this, anyway, as it provides you a way to back up your work, by copying it off of your VM and on to something else.)

Retrieving the tarball from your VM

If you'd like to retrieve the tarball from your VM — because you prefer to submit it from your host operating system, or simply because you want to keep a separate copy of it — this is one way to do it.

The simplest method for retrieving a file is to use a protocol called SCP. SCP is a protocol that allows you to copy files from one machine to another. You'd connect to your VM using SCP, then copy the tarball to your host. How you use SCP depends on your host operating system.

- Windows has no built-in support for SCP, but there are a few well-known programs you can download and install. The simplest is called WinSCP, which provides a graphical user interface for connecting to your VM using SCP.
    - Download WinSCP from www.winscp.net . The easiest option is to select the **Installation package**, which you would want to download and then run; it will install WinSCP and make it available like a typical Windows application (e.g., from the Start menu).
    - Launch the WinSCP application. Click New to create a new session and enter the following values:
        - Under **Host name**, specify the IP address of your VM
        - Under **Port number**, specify **22**
        - Under **User name**, specify **ics46**
        - Under **Password**, specify the password belonging to the ics46 user account on your VM — you did change it already, didn't you? If not, do that now (see above).
        - Leave **Private key file** blank
        - Next to File protocol, select SCP
        - Now that you've set these up, you can click the **Save...** button to save these settings so you don't have to enter them again, then click **Login**.
    - Once logged in, you should see the files on your VM. Navigate to the **/home/ics46/projects/proj0** directory. You should now see the **project0sc.tar.gz** file you created earlier, which you can drag and drop to your host machine wherever you'd like it to be stored.
    - After copying the file to your host machine, you can close WinSCP.
- If you use Mac OS X, Linux, or other flavors of Unix, you're in luck! SCP support is built into your operating system. Your best bet is to bring up a shell prompt (e.g., a Terminal window in Mac OS X), decide what directory on your host machine you want to copy the file to, then to issue the command scp **ics46@VM_IP_ADDRESS:~/projects/proj0/project0s22.tar.gz TARGET_DIRECTORY_ON_HOST** to copy the file.
    - So, for example, if you want to copy the tarball to the directory **/Users/boo/Documents** on your host machine, you would issue this command: **scp ics46@VM_IP_ADDRESS:~/projects/proj0/project0s22.tar.gz /Users/boo/Documents**
    - Note that whatever directory you choose on your host machine will already have to exist.

Inspecting your tarball before submitting it

It's not a bad idea to inspect the contents of your tarball before you submit it, just to be sure that (a) all of the files you want to submit are included, and (b) the right *versions* of the files are included. How you open the file depends on your host operating system:

- On Windows, you will need to obtain a tool that can open a **.tar.gz** file; Windows doesn't support this ability by default. You may have such a tool already (e.g., WinZip, WinRAR); if not, a good free choice is 7-Zip: http://www.7-zip.org/
- On Mac OS X, you'll find that you can simply navigate to the file in Finder and double-click on it, which will extract its contents. It should create a directory called **app** and another called **gtest** inside of the directory where the **.tar.gz** file resides, containing all of the files that are included in the tarball.
- On other flavors of Unix or Linux, you can extract the tarball most easily from a shell prompt, by changing into the directory containing the tarball and issuing the command **tar xvfz project0s22.tar.gz.**

Deliverables

After using the gather script in your project directory to gather up your C++ source and header files into a single **project0s22.tar.gz** file, then submit that file (and only that file!) to Checkmate.

This quarter, we're using an ICS-built system called Checkmate, which allows you to submit your projects online. In order to use it, you first have to activate your UCInetID if you haven't already. UCI provides all its students with basic computing services, including electronic mail and other Internet services, via a "UCInetID" computer account. Instructions for activating this account are available from several sources, but perhaps the easiest way is to activate your account on-line at activate.uci.edu. All students enrolled in this offering of ICS 46 need this account, both for using Checkmate, and also so that the course staff can email you when necessary. If you have not activated your UCInetID, do so immediately.

To use Checkmate once you've activated your UCInetID, all you need to do is go to www.checkmate.ics.uci.edu, log in using your UCInetID, and follow the instructions there. (If you get a message about security certificates, just click "Continue.")

Checkmate is not linked to the Registrar's database, so it is not aware that you are enrolled in the course; instead, you will need to register yourself for the course within Checkmate.

While it has proven to be resilient in the past, you may encounter occasional outages, unexpected behavior, or unclear error messages when using Checkmate. If you have any reason to believe that the system isn't working as you believe it should, read the instructions carefully, try it a second time, and if you're still having difficulty, send Shindler an email describing the problem as precisely as possible. If you keep us informed about problems in a timely way, we will make sure that any problems with Checkmate won't affect your grade.

If you want to submit a newer version before the deadline, Checkmate allows you to remove the files you originally submitted, then resubmit new versions.

We will grade only what was submitted to Checkmate before the deadline. If you replaced some of your files with newer versions before the deadline, we will grade only the most recent submission of each.

We will not grade files submitted after the deadline has passed, nor will we grade files submitted via email or in paper form.

You are responsible for submitting the version of your project that you want graded. We will grade only what you submitted before the deadline. Accidentally submitting the wrong version, or forgetting to submit files, will not be considered grounds for a regrade.

This project is not included in the late work policy for this course. **It needs to be completed on schedule and submitted before the due date above.** Submissions beyond that deadline will not be considered.

Understanding the risks of using something other than the ICS 46 VM
You are certainly within your rights to use something other than the ICS 46 VM to do your work this quarter, but you should be aware that you are bearing some risks by doing so:

- You'll be responsible for setting up and configuring the necessary software. As a rule, we won't be available to help configure environments other than the ICS 46 VM; you'll be on your own. For example, the provided project templates are specifically set up to work on the ICS 46 VM; making them work on something else will be up to you, and you might find this to be a tricky task, even if you believe you have all the same software we do.
- We'll be using the ICS 46 VM to grade your work, so if you write a program that compiles and runs on your own setup, but does not compile and run in the ICS 46 VM, you'll be the one to bear that risk. (This is not at all unrealistic, given differences in C++ compiler versions and the operating systems they run on, so don't discount this risk lightly.)
- When it comes time to submit your work, we expect everyone to use the provided gather script to gather their files for submission, so that everyone's submission is in the same format. We have test automation tools that depend on that. If you work outside of the ICS 46 VM, you'll have the additional problem of making sure that your submission is in the right format — and, if it's not, risk us not being able to grade your work, or that we refuse to grade your submission because we would have to perform special workarounds to be able to grade it.

At minimum, my suggestion is to get the ICS 46 VM set up, so that you can test your projects in it before submitting them and then run the gather script before submission, even if you prefer to do your day-to-day work elsewhere.

Your grade on this project

There are 2.5 points possible on this project; they are only available by test cases. We will run test cases with the code you submit; each test case is worth some fraction of the grade. Test cases that take longer than two minutes to run on the instructor's (reasonably modern) computer may be deemed incorrect runs, even if a longer amount of time available to them would cause a correct answer. The first point of the test cases is for the verifySolution() code, while the remaining points come from the LLQueue. For the second part, we will also check for memory leaks; a test case that is correct in result but has memory problems may have reduced (but not zero) credit. To check your code for memory leaks, use the --memcheck flag when running your built program.