COMP 271 Final Project
Due: Saturday, May 7 by 4:15PM via Sakai.

Final projects can either be individual projects or completed by pairs. Groups of 3 or more are not allowed. You must select from one of the provided project ideas. In most cases (unless specifically indicated), a project can be completed by an individual or a pair of students. Unless explicitly specified, all code must be developed in C++. It is your responsibility to equally distribute work between teammates. If your partner is not holding up their end of the bargain, I need to know this early.

Your final project consists of two components:

(**3PTS**) Documentation: This documentation will include the following:

1. One paragraph description of the project.
2. One paragraph description of the data structures used and ***why*** you selected this data structure(s) for the implementation.
3. A step-by-step instruction of how to use the code. This should include a test case for running your code.
4. List the IDE(s) and OS(s) used during development and testing.
5. If you worked with a partner, your documentation must also include a listing of who did what.

(**7PTS**) Code: Your code MUST be commented; no comments, no points. If your code relies on user input (either via the console or via a file), you should provide sample data for testing. You should assume that your user is tired and likely to make errors and thus should safe guard against errors. For example, if you request a file from the user, you should check it exists. If you ask the user to enter in a value, you should check it is valid. In either case, you should communicate with your user the problem and help her make the right selection. She will then be a happy chappy.

Your code will be graded as follows:

- 3pts: executes as expected. In other words, does your code implement the problem?
- 3pts: efficiency. Is your code neat and efficient? This means both in terms of memory usage as well as run-time. Do you have nine loops when you only need one, etc.?
- 1pt: design. Does your code take advantage of the benefits of object oriented programming? Does it promote code reuse? Are problems properly partitioned into sub-problems? For some projects this will be much more important than for others. Here the key challenge is design. Is the program designed well?

Opportunity for feedback: You may submit a (maximum) 1 page design summary via Sakai, no later than Friday, April 8 to get feedback.

Projects

1. Puddle Jumper. Much of the world connects via small commercial airlines (puddle jumpers). Your summer job is as a travel agent to assist people to book flights using the intricate network of puddle jumpers. These aircraft often fly between relatively close cities. As such, getting across the country or a few states will likely require changing planes and a few layovers. Given a list of flights (which includes both times of departure and arrival, distance, and prices), identify the (1) shortest – time – flights as well as (2) cheapest route for a customer. *Team Challenge*: Also identify the shortest in terms of distance.

2. Proliferate. Cells (as in the microscopic things that sustain life) can only divide between 40-60 times in humans before they die (the Hayflick limit). You start as a single cell. Replicate this process. Allow the user via the console to take a snapshot of the process and get statistics with regards to how many cells there are and how many have already died. Given the range of the number of times a cell can divide, this should be a pseudorandom decision made at the time of the "birth" of a cell. *Team Challenge*: Allow the user to introduce via the console events which will mutate the cells, maybe killing some off. (We'll avoid the chance of cancer… that's when cells just never die.)

3. Exploding Kittens. Have you played the game? It involves chance and strategy. Implement the standard deck game (http://www.explodingkittens.com/how) to work via the console for a single player and single computer player. *Team Challenge*: Design the game for 4 computer players.

4. Twitter Me This. (Team only) I just invented the coolest thing-a-ma-bob. It's going to make millions. To cut costs, I want to minimize my advertising budget and take advantage of social media. Luckily your friend is an expert hacker and hacks into Twitter to retrieve a list of all users and their followers. His/her only caveat is that you don't get caught. You want to reach the greatest number of users while still staying under the radar of Twitter's lawyers. Your friend is quite adamant about the number of Tweets you can send although sometimes its 20 while others its 100; it's always changing. For a given cap of tweets, how many individuals can you reach via Twitter? Who are the key people to tweet to?

5. Banana! (Solo only) You have been taken over by an evil minion overlord. He has promised to release you if and only if you help him with his evil plan. He'd like to make the Internet minionized. Develop code to help him in his quest and regain your freedom. For a given file you must replace English phrases with the minion version. For instance "one" is "hana". But this isn't just a simple find and replace. The word "tone" is not "thana". That's just nonsense. The internet is big, so here time is of the essence.

6. Contagion Part 2. Returning back to Homework #4, words of size 10 are a bit smaller than one would expect. In fact, we're more likely to see reads of size 250. Revamp the assignment to handle words of size 250. You'll likely crash your laptop though with real data and a simple solution. So this is where creativity comes in!

7. Contagion Part 2 (option 2). So a different tangent on Homework #4; DNA sequencers make mistakes (they say something that is really an A is a T). It happens. Adapt your code from Homework #4 to tolerate an error rate of 10% and still be able to identify matches between a genome and read. We'll push ourselves also a little closer to reality (rather than words of size 10). Let's consider words of size 30.

8. Divvy. (Team only) Divvy has put out this call before. They have data regarding the rentals of bikes from each individual station across the city. They have information as to which station/time bikes are rented and returned. Your summer job is to identify times and stations for which Divvy should ride out to move bikes from an underused station to a station in need.

9. In a World Without Surge Protectors. (Team only) A large storm fries all of the world's ATMs. Your part-time job at the bank is now a nightmare. So many phone calls from people requesting their money moved from one account to another. In fact there's a new call every second. It takes 5 seconds though to process a request. Your boss is getting upset because you're falling behind and some of the aristocrats with big bucks are tired of waiting and making a big fuss. You decide to throw your headset to the side and develop a piece of software such that your fellow tellers can add in the information: customer ID, amount to be transferred, account # withdrawing from, account # depositing to, customer's wealth, and time the deposit must be posted by. Your solution will find the best means of maximizing the transactions processed with minimal complaints from the aristocracy.

10. Neighborhood Watch. Your neighborhood is being terrorized by a bicycle bandit. The only lead is that the lady three doors down swears she saw the bandit load a bicycle into a car. The local PD is quite certain the person is using some sort of vehicle to nab the bikes. Your neighborhood is all metered via an app such that whenever a car parks it registers the time in and out of the car and the car's make, model, and license plate number. You have downloaded all of the parking information for the neighborhood from the parking company's website as well as a list of the bike thefts. These police reports include just the thefts in the neighborhood and include a time in which the bike was taken. Can you figure out which car(s) the police should be investigating? *Team Challenge*: In real life, we never know exactly when a bike was taken… "I parked my bike at the corner of Elm and 3rd at 2PM. When I returned after the movie at 5PM it was gone." Can you figure out which car(s) the police should be investigating?

11. Freshman scheduling. Remember when you were a freshman in UNIV 101 and you had to do your 4 year schedule? Remember how hard you thought that was. Now a seasoned veteran, you want to make a few bucks off of the incoming class of 2017. Design a program that will plan a student's schedule for a given major (pick one). Remember, some courses have prerequisites, also not every class is offered every semester. It's a 4 year schedule so make sure to pare it down to 8 semesters. When it comes to core, just specify Tier 2 History (or the like). Assume there is a class to satisfy every core requirement available every semester. *Team Challenge*: Design all possible schedules.

12. Threads. Develop software to read in a tree from file. This file includes two columns. The first is the name of the parent node data and the second is the child node data. Remember each parent is itself a child (with the exception of the root which is indicated by the node data "Root"). Assume a binary tree. Read in this file and thread the tree using inorder threading. Make your code capable to read in any provided file containing the tree data. Write out the list of threads to the console in the following format: Leaf Node data, Left Child Thread's Node's Data, Right Child Thread's Node's Data. *Team Challenge*: Allow the user to specify inorder, preorder, or postorder threading. Write out the threads to console for the threading strategy selected by the user.

13. Going off the grid. You've had it with the User Agreements required of Apple. You're going to make your own iTunes. Develop the data structures required to store all of your music. If you're going to take this commercial, you'll need to have the bells and whistles that Apple does. Include metadata for each song. And remember that songs are part of albums. I'm in the mood for some Belle and Sebastian. I'd like to access all songs by this artist. Note, in your data structures, you won't actually be "holding" the mp4 file but rather the address of the song within your hard drive.

14. Finding prophages. (Solo only) The biology is not important. I actually need to do this. I have files of sequences they have 1 line that has information about the sequence (like the organism it came from) and then it's just millions or billions of As, Ts, Cs, and Gs. I have a second file and it has a start and stop position within this string of As, Ts, Cs, and Gs followed by a description of the gene located within this sequence (the start and stop position are the start and stop position of a gene within the genome sequence if you're interested). I want to retrieve all prophage sequences and write them to a file. Their description will have the word "phage" in int. I'll need to know where these sequences came from so I'll want to write it out in the following format:

```
>This is the information that was on the first line of the genome file
| start | stop
ATGATTATCTTCGTATAGATCTGATTCGGATACCAGATTAGGCATTAGGCCCATAGAGGATCGGATACGA
TATTCCTCTGGTGCGGGC…
```

I've concatenated to the first line from the genome sequence a pipe followed by the start position of the gene and the stop. Note, there will be more than one (probably) per genome file. There also may be 0.

15. DIY Compiler. In homework #2 I made up a silly language and you implemented a compiler. Let's take it a step further. Implement the code necessary to firstly discount any white space. We can have 1000 spaces after an equal sign and it won't cause our code to break. Secondly, implement your code such that it can update variables for +, -, /, *, ^, and ( ). Remember order of operations. Include a call stack which is visualized to the user via the console as it executes the code. *Team Challenge*: In our original assignment, we only had three kinds of statements situations; we could have a function call or variable declaration or a variable reassignment (c=c+2). In addition to the functionality detailed for the solo option, this challenge can also consider conditional (if… else) statements.

16. Check Please! I'm opening up a new restaurant and the crowds are pouring in. I've got a certain number of tables and each table type (2-top, 4-top, etc.) can seat only a set number of people. How can I best seat the stream of people wandering in while minimizing the time they have to wait. (Waiting customers tend to walk off…) *Team Challenge*: Another caveat. I don't have a table for 20 but a gang of Loyola students come wandering in and want to sit together at one table… I'll have to push some tables together. Tables are now movable.

17. Long Enough Common Subsequences. (Solo only) You are given a list of sequences and want to identify all subsequences which are present amongst the sequences that meet a predefined threshold. This is a slight modification of the classic CS problem of finding the Longest Common Subsequence. I use this all the time. Say I have a lot of viruses and I want to find a subsequence (part of their genome) that is common amongst them all. This region has to be at least x long so that I can target it with a drug.

18. Huffman Coding. (Solo only) Ever heard of Huffman Coding? It's a great application of binary trees and likely saved you when you were in a pinch on more than one occasion. Huffman coding is often used for compression methods. This is how you get those zip files and MP3s. For MP3s it acts at the end of the compression to code information, and this is not therefore itself a compression algorithm but rather a coding method. This coding creates variable length codes on a whole number of bits. Higher probability symbols have shorter codes. Huffman codes have the property to have a unique prefix, they can therefore be decoded correctly in spite of their variable length. For this project you must implement the algorithm and do some compressing.

19. Word Cloud. Given a particular text, word clouds give greater prominence to words that appear more frequently in the text (i.e. they are bigger). If you Google "word cloud" you can see lots of examples. Write code that can generate the prominence of words within a text. Let the user specify the number of words they want in the word cloud and write this to file. *Team Challenge*: Piggyback off of the code required to determine the prominence of the words and visualize it. This will require resources which have not been covered in this class. It will require independence to identify a solution. The computational component must be conducted in C++ but the visualization does not have to be.

20. Ants! (Solo only) My kitchen counter is being overtaken by ants. The "blue" colony on the left side of my counter wanders around while another colony of ants ("red") have taken particular affinity to the right side of my counter. Each colony has a certain number of "steps" they can take in a day. When they meet, they will battle to the death. (Not very friendly ants.) I'm lazy so I'm just hoping that they'll eventually kill each other off. A few logistics, ants are silly; they wander in no particular direction (although they avoid standing on top of each other). Ants won't fight other ants belonging to their same colony. *Team Challenge*: I cleaned my counter; no ants. Now I have a hornet problem. Implement the ant problem but here for hornets. FYI, hornets can fly.