Programming in C++ - Final exam spring 2022 - first exam period

In programming problems you can always use any C++ types and STL data structures.
All problems must work with the given main(), but students can always add more tests.

Part A: Knowledge (20%)
Multiple choice questions in Canvas Quiz.

Part B: Basic core skills (30%)

1. Make the operation **count_value - (10%)**
    - It ***returns*** the number of times a specific *integer* value is in an *integer array*
    - Takes as parameters the ***array***, its ***length*** and the ***value***
    - While running also prints a single line on the terminal:
        - Shows each index where the correct value is found
        - Example for array **[4, 0, 0, 8, 8, 0, 5, 0, 6, 6]** *(length 10)* and value **0**:
            - "Indexes with correct value: 1 2 5 7"
2. Make the class **Result - (10%)**
    - Result has three data variables
        - **analyst**, which is a string
        - **count**, which is an integer
        - **amount**, which is a real number
        - **confirmed**, which is a boolean value
    - Result has three constructors
        - A **default constructor** which sets the string to an empty string ("") and the numbers to zero and boolean to false.
        - A **parameter constructor** which takes all four parameters
        - A **copy constructor**
    - Result has **get_** and **set_** operations for each variable
        - **get_analyst**,**set_analyst**, **get_count**,**set_count**, **etc...**
    - Result can be sent directly onto a print stream with the **operator <<**
        - *The string and numbers and either "confirmed" or "not confirmed"*
3. Make the operation **read_results_from_file(filename) - (10%)**
    - The operation opens a text file with the name **filename**
    - Each line in the text file has a **string** (there will be no whitespace in the strings), an **integer**, a **real number** and either 0 (false) or 1 (true) in text format (*example in code folder*).
    - The operation returns a collection of instances of the class Result
        - *This can be a simplified version of Result or you can use your **result.cpp** and **result.h** files from B2. It only needs the data and << operator*.
        - The collection can be any **STL data structure**, **dynamic array** or **homemade**. You can even change the syntax in **main()** if you want but it will work unchanged with most **STL data structures**.

Part C: Core skills (30%)

- Make the function/operation **operate_on_any_type - (10%)**
  - *It can accept any types* as **parameter1** and **parameter2** and does the following:
    - Prints the values of the parameters
    - Shows the result of applying the operator + to the parameters
      - *Both ways, with each param on either side of the operator*
    - Shows which one is less than the other, based on comparison operators, or states that they are equal.
  - The output should be clear on what it is showing in each line
- Make the class **LabelledList - (10%)**
  - It has the following operations:
    - add
      - Takes a ***string*** (the label) and an instance of ***NumericData*** as parameters (two parameters)
    - print_list
      - Prints to the terminal all data that has been added to this instance of **LabelledList**
      - Label and NumericData that were added together should be printed in same line
  - You can use any built-in data structures for storing or use pointers and arrays, whichever is best.
- Use ***inheritance*** to make the class **WritableList - (10%)**
  Has all the exact same functionality as **LabelledList**, *plus two operations*:
  - **write_to_file(filename)**
    - Writes to a binary file with the name filename
    - Writes enough data to recreate the entire list
  - **read_from_file(filename)**
    - Reads from a binary file with the name filename
    - After reading the container should have the exact same labels and data as the container that wrote into the file

Part D: Advanced skills (20%)

Following are four problems.
Solve one of them correctly for 15%
Solve another one of them correctly for 5%
Solve all of them, in addition to correct solutions to parts B and C, to be inducted into the **Programming in C++ hall of fame**.

In the folder cpp_finalexam there is a file ***README.txt***
Edit this file to state which advanced problems you have solutions to and which two of them should be graded for the main 20%

Each problem follows on a separate page:

D1: Networking

Write a server that runs on port 4061. It is OK if you use another port, but same in both client and server. Then you must make it very simple to change the port in the code.
Write a TCP client that connects to that server on localhost (IP: 127.0.0.1).
Make it very simple to change the client so that it connects to skel (IP: 130.208.243.61) but if you test the server on skel use another port between 4001 and 4099 (as many students may be doing so at the same time).
The client and server should do the following:

- Server sends the message “Hello”.
- Client receives that message and **checks that it is in fact** “Hello”.
- Client sends the message “Good day”.
- Server receives that message and **checks that it is in fact** “Good day”.
- Server sends any text message (shorter than 100 bytes).
- Client receives that message and prints it to the terminal.
- Client sends any text message (shorter than 100 bytes).
- Server receives that message and prints it to the terminal.
- Both server and client quit (without crashing) after this communication.

NOTE:
If “Hello” or “Good day” is not the correct message the recipient should print a message to the terminal saying that it is the wrong message and then both should quit.
Your solution client and server should not need to print this error message, but if the client is tested against another server or the server against another client this may come up.

D2: Multithreading and concurrency

You have a main() program that calls the operation **process_data** with a **reference** to an instance of **DataStore**, an instance of **DataProcessing** and a **number** (id number of this instance of process_data).

1. Change this program so that it runs five (5) instances of **process_stuff** simultaneously which finish all the calculations, each data part only calculated once. Each instance should have a unique id number (1-5) but a reference to the same instance of **DataStore**.

No code should be added in the **DataStore** class or the files **data_processing.h** or **data_processing.cpp**.

2. Change the code in the operation process_stuff so that instances don't get mixed up when calling operations on DataStore and also so that each thread gets to finish printing whole lines to the terminal uninterrupted. You may have to change some of the logic, in addition to adding functionality between lines. Also make sure that the time consuming calculations are performed as much in parallel as possible. You should see a very clear difference in run time depending on whether each of the 100 data calculations run in series or in parallel.

D3: Operations as parameters

You have a main program that takes two lists of integers (in vector<int>) and prints the sum of the first ten elements of the lists.

When your solution is ready the final result is a program which reads numbers from two files into two lists, then allows the user to select a method to apply, then calls **do_operation_on_whole list()** with the lists and the method itself. **do_operation_on_whole list()** runs the operation on each pair of list items, builds a new list which **main()** then prints.

It is OK to change declarations of operations, parameters etc. wherever needed, but this must be solved by sending an object or value representing the calculating operation to **do_operation_on_whole list()** and calling it there for the entire list.

Implementation suggestion:

1. Finish implementing read_list so that it fills a list with the numbers in the file.
2. Instead of looping through the first 10 items, call **do_operation_on_entire_list()** and have that loop through the entire lists and calculate the sum of the numbers in each index.
3. Write operations for multiplying, subtracting and selecting the lower or higher number of two numbers.
4. Change the main() operation so that the user can choose which operation to run on the whole list and have the main() function send the correct operation to **do_operation_on_whole list()**. Change the parameters of **do_operation_on_whole list()** so that it does not make this choice with if statements but simply calls the operation that is passed to it through the parameters, for each value in the lists.
5. Have **do_operation_on_whole list()** fill a result vector with the results and then print the values from that result vector in **main()**. It does not matter whether you use the return value of **do_operation_on_whole list()** or add a parameter for the result.

D4: Python module

Make the python module **digit_module**.

It has a single operation: **get_most_significant_digit_for_integers(lis)**

The parameter is a list of integer values.
The return value is also a list of integer values, of the same length as the input.

Each integer value in the result is the most significant digit (for regular decimal format) of the corresponding integer value in the parameter list.

There is a test python script (tester.py) that should print the following output:
[1, 1, 2, 2]
[1, 9, 9, 6, 4, 2, 1]
[1, 2, 0]
[1]
[]

I have given the structure for a python module the way I have done them in class. Students can make the module with other methods or other libraries if they wish. **That means you can delete everything (except the test script) and start over**.

To continue with the code given try searching for operations starting with **PyList_**, e.g. PyList_New, PyListCheck, PyList_Size, PyList_SetItem, PyList_GetItem, etc.
Also operations with **PyLong_** for working with integers, e.g. PyLong_Check, PyLong_AsLong, etc.