Programming in C++ - Final exam spring 2021

In programming problems you can always use any C++ types and STL data structures.
Exception: In B1 you can not use std::string
All problems must work with the given main(), but students can always add more tests.
Exception: In B3 you can change the main() function to suit your return value

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

Part B: Basic core skills (30%)

1. Make the operation **compare_strings - (10%)**
    - It returns true if the strings are the same, otherwise false
    - Use character arrays/pointer, ***not std::string***
        - You can still use string operations from <string.h>
    - If not the same the operation also prints the following information to the terminal
        - each index that differs up to the length of the longer string, so every index differers after the last index of the shorter string.
        - The difference in the length of strings (as a positive integer).
2. Make the class **Player - (10%)**
    - Player has three data variables
        - **name**, which is a string
        - **number**, which is an integer
        - **result**, which is a real number
    - Player has three constructors
        - A **default constructor** which sets the string to an empty string ("") and the numbers to zero.
        - A **parameter constructor** which takes all three parameters
        - A **copy constructor**
    - Player has **get_** and **set_** operations for each variable
        - **get_name**,**set_name**, **get_number**,**set_number**, **get_result**,**set_result**
    - Player can be sent directly onto a print stream with the **operator <<**
3. Make the operation **read_players_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** and a **real number** in text format (*example in code folder*).
    - The operation returns a collection of instances of the class Player
        - *This can be a simplified version of Player or you can use your* ***player.cpp*** *and* ***player.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()** although that will work with most **STL data structures** unchanged.

Part C: Core skills (30%)

- Make the class **KeyedContainer - (10%)**
    - *It can accept any types* as **key** and **data** and has the following operations:
        - **PutIntoContainer(key, data)**
            - Associates this data value with this key value
        - **GetFromContainer(key)**
            - returns the corresponding data for this key value
- Make the class **SpecialContainer - (10%)**
    - Use inheritance to inherit the functionality of **KeyedContainer**
    - **SpecialContainer** only accepts ***int*** as **key** and ***NumericData*** as **data**
    - Otherwise SpecialContainer works exactly like KeyedContainer, but you don't need to specify the types when making an instance (***see main()***)
- Add two operations to **SpecialContainer - (10%)**
    - **write_to_file(filename)**
        - Writes to a binary file with the name filename
        - Writes enough data to recreate the container
    - **read_from_file(filename)**
        - Reads from a binary file with the name filename
        - After reading the container should have the exact same keys 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

There is a server running on skel.ru.is (ip: 130.208.243.61) using port 4893.
Write a TCP client that connects to that server, then does the following:

- Receives a message from the server (message will be no longer than 1020 bytes)
- Sends a message to the server of equal size and containing the exact same values
- Repeat two more times (*so in all, three messages received and echoed back*).
- Receive a final message from the server and print it to the terminal
    - *This message will state whether the previous messages were correct*

D2: Multithreading and concurrency

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

Change this program so that it runs seven (7) instances of **process_data** simultaneously which finish all the calculations, each data piece only calculated once. Each instance should have a unique id number (1-7) but a reference to the same instance of **DataStorage**.

Change the code in the operation process_data so that instances don't get mixed up when calling operations on DataStorage 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: File format handling

Make a program that is run like this:
./d3 <name_of_input_file> <name_of_output_file>

The input and output files are both 24 bit bitmap files, made in Windows (so they have the headers described below). *The program only needs to read 24 bit Windows bitmaps but it should print a message if the file is not a valid BMP or not a 24 bit format*.
The program should **invert the colors** of the input file and write the resulting image to the output file. This means every separate color value in the result is as far down from a "full value" as the original value is up from zero. Zero becomes full value, full value becomes zero, half value stays the same. ⅓ of full value becomes ⅔ of full value, etc. etc.

Here is a description of a bitmap header:

Offset hex	Offset dec	Size	Purpose
0	0	2 bytes	always the string "BM"
2	2	4 bytes	The size of the BMP file in bytes
6	6	2 bytes	Doesn't matter. can be 0
8	8	2 bytes	Doesn't matter. can be 0
0A	10	4 bytes	The offset, i.e. starting address, of the byte where the bitmap image data (pixel array) can be found.

Here is the continued header in the exact format used in the test files in the folder D3:

Offset hex	Offset dec	Size	Purpose
0E	14	4	the size of this header, in bytes (40)
12	18	4	the bitmap width in pixels (signed integer)
16	22	4	the bitmap height in pixels (signed integer)
1A	26	2	the number of color planes (must be 1)
1C	28	2	the number of bits per pixel. 24 in our case.
1E	30	4	the compression method being used. No compression in this case
22	34	4	the image size. This is the size of the raw bitmap data.
26	38	4	the horizontal resolution of the image. (pixel per metre, signed integer)
2A	42	4	the vertical resolution of the image. (pixel per metre, signed integer)
2E	46	4	the number of colors in the color palette, or 0 to default to 2n
32	50	4	the number of important colors used; generally ignored

D4: Python module

Make the python module **digit_count_module**.

It has a single operation: **get_digit_count_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 digit count (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:

```
[5, 9, 1, 3]
[2, 3, 7, 1, 1, 4, 4]
[1, 6, 0]
[3]
[]
```

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. **<u>That means you can delete everything (except the test script) and start over</u>**.

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.