C++ Collections and Raylib Assignment

Assignment A02

Welcome to your second C++ assignment. In this assignment you will be working with std::strings, std::map, namespaces, and a touch of raylib.

This is a “no ai” assignment, meaning that your source code should be a genuine reflection of your personal effort and understanding. That said, feel free to “discuss” the C++ topics we covered in class this week with a generative ai tool, just don’t ask it to code your assignment for you. :)

Like the first assignment, please keep track of how long each program took to write for your reflection. Also, please submit a Visual Studio Project or CMake project for each program, not just the source code files.

# Part 1 – The Program Within You

## 1A – Bioinformatics and Base Counts (std::string, std::map)

DNA or Deoxyribonucleic Acid is the molecule inside of your cells carrying your genetic code. DNA has a double [helix](https://simple.wikipedia.org/wiki/Helix) shape, which is like a ladder twisted into a [spiral](https://simple.wikipedia.org/wiki/Spiral). Each step of the ladder is a pair of [nucleotides](https://simple.wikipedia.org/wiki/Nucleotide). There are four types of nucleotides (called bases), [Adenine](https://simple.wikipedia.org/wiki/Adenine) (A), [Thymine](https://simple.wikipedia.org/wiki/Thymine) (T), [Cytosine](https://simple.wikipedia.org/wiki/Cytosine) (C), [Guanine](https://simple.wikipedia.org/wiki/Guanine) (G). In the world of bioinformatics (the computational processing of DNA information) DNA strands are often represented by strings comprised of only the characters A, T, C, and G.

Write a function that takes as input a DNA string and returns the count for each type of nucleotide found in the string. Assume perfect data (i.e., no input validation is necessary).

Example input: AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC

Example base count:

A:20 C:12 G:17 T:21

Marks will be assigned for the following:

* The DNA string base counting is implemented as a function.
* You have researched the std::map collection type and used a std::map (example std::map<char, int>) as the return type of your base counting function.
* Your base counting function is implemented in a separate cpp file with an associated header file.
* Your base counting function is placed within a custom namespace that is given a domain appropriate name.

## 1B – Hamming Distance (std::string)

A mutation is a mistake that occurs during the creation or copying of a nucleic acid like DNA. The simplest and most common type of nucleic acid mutation is a point mutation, which replaces a single nucleotide with another. The Hamming Distance between two DNA strings is the number of corresponding symbols that differ between the strings.

The following two DNA strings have a Hamming distance of 7, meaning that there exist 7 point mutations between the two. The point mutations are shown in red:

Text

Description automatically generated with medium confidence

Add a function to the namespace you created in 1A that takes two DNA strings as input and returns the integer Hamming distance between the strings. Assume perfect data (i.e., no input validation is necessary).

Sample input:  
GAGCCTACTAACGGGAT  
CATCGTAATGACGGCCT

Sample Output:  
Hamming Distance Between DNA Strings: 7

## 1C – Bioinformatics Processing Program

Finally, write a program that uses your namespaced bioinformatics functions in the following way:

* The program should prompt the user for two DNA strings.
* If the two DNA strings are the same length:
  + Output the base counts for each string.
  + Output the Hamming distance between the strings.
* If the strings are not the same length:
  + Inform the user of the length requirement and re-prompt for new DNA strings.
* Assume perfect data (i.e., no input validation is necessary beyond the length check).

# Part 2 – Raylib Control Panel Playground

Create a Raylib application that draws a whimsical control panel using basic shapes and colors. Think of this as concept art for the UI of a game that includes a stylized control panel. At a minimum, your program must use at least ten different Raylib functions. You can research the functions using [this Raylib Cheatsheet](https://www.raylib.com/cheatsheet/cheatsheet.html). Remember, the easiest way to add Raylib to a VS project is by using NuGet. [See Cate’s demonstration from last week](https://youtu.be/kGXGu9ndqmM?si=NZoFhCkPa2Co64UY&t=950).

Your control panel should include a minimum of two custom widgets of your own design. Widgets should be implemented as functions. Your panel should display multiple instances of each of your widgets. You should use at least four colors (either custom colors or built-in color constants). Your panel should include both graphics and text.

Extra marks available for keyboard or mouse interactivity, or non-interactive animation.

[Here is an example C++ Raylib program that includes two interactive widgets, a concentric gauge and a two-lamp toggle switch](https://gist.github.com/stungeye/5570d473e7f62f843482472b9c7e20fc). *(Apologies for the magic numbers in the code. I started to refactor them away, but some remain!)*

A screenshot of a video game

AI-generated content may be incorrect.

# Part 3 – Reflection

Write a short reflection on this assignment. One or two paragraphs will be sufficient. Please include the following:

* What problems you encountered and/or discoveries you made while working on this assignment.
* Personal difficulty rating for each problem: Easy, Medium, Hard
* Approximate length of time you worked on each problem.

# Marking Rubric (18 Marks)

## 1A – Bioinformatics and Base Counts (4 marks)

* **Valid base counting**: Function correctly counts the number of each nucleotide (A, C, G, T).
* **Function return type:** Efficiently uses a std::map to return base counts.
* **Separate compilation unit:** Your base counting function is implemented in a separate cpp file with an associated header file.
* **Namespacing:** Your base counting function is implemented within a custom namespace with a domain appropriate name.

## 1B – Hamming Distance (2 marks)

* **Hamming distance calculation**: Function correctly calculates the number of point mutations between given DNA strings.
* **Implementation details:** Your hamming distance function is implemented within the same namespace/files as your base count function from 1A.

## 1C – Bioinformatics Program (2 marks)

* **Identical Length DNA output:** If the user-input DNA strings are the same length:
  + Output the base counts for each string.
  + Output the Hamming distance between the strings.
* **Invalid input:** If the DNA strings are of different lengths, inform the user of the length requirement and re-prompt for new DNA strings.

## 2 – Raylib Control Panel Playground (8 marks)

* At least ten Raylib functions have been used.
* At least four colors have been used.
* One marks for each custom widget function of your own design. (Max 2 marks)
* Your widgets are well modularized as functions, and your panel includes multiple instances of each widget.
* Your panel includes both graphics and text.
* One mark each for instances of keyboard interaction, mouse interaction, or a non-interactive animation. (Max 2 marks)

## 3 - Reflection (2 marks)

|  |  |  |  |
| --- | --- | --- | --- |
| **Overall** | **Excellent (2)** | **Acceptable (1)** | **Poor (0)** |
| **Reflection** | Reflection is insightful, clearly written, and addresses all required points. | Reflection is complete but may lack depth or miss some of the required points. | Reflection is missing or fails to address many of the required points. |

## Code Formatting, Naming, & Distribution (Marks deducted for infractions)

* **Consistent Formatting**: The code follows consistent indentation and spacing rules, making it easy to read.
* **Descriptive Naming**: Variables, functions, and classes are named descriptively, reflecting their purpose or use.
* **Source Code Packaged Project**: Source code should be easy to open in Visual Studio. It should be packaged as a Visual Studio project with a \*.sln file, or if you are not using Visual Studio yourself, as a CMake project with a CMakeLists.txt file.

## Code Commenting (Marks deducted for infractions)

* **Function Documentation**: Each function includes a comment describing its purpose, input parameters, return values, and any side effects.
* **Purposeful Comments (When Code Isn’t Obvious)**: Comments explain the why, not just the what. They describe the intent or reasoning behind complex or non-obvious code.
* **Avoids Over-Commenting**: The code isn’t cluttered with obvious comments that state the obvious (e.g., int x = 10; // Assign 10 to x).

## Additional Requirements (Marks deducted for infractions)

* **No Recursive Calls**: None of your programs include functions that call themselves, or
* **No Function Side Effects**: Functions should only alter the state of their own local variables. No function should access or modify a variable outside of its local scope.