Lab 05 - Count Sort & Hashing

Instructions:

- ***Hashing*** **is the process of applying a hash function to data to generate an integer value—called a** ***hash*****—that enables fast storage, lookup, and retrieval in hash-based structures such as a** ***hash table*****. Because a hash function maps elements from a potentially large data domain into a fixed range of integer values [0,..., n], it can be used to generalize the `CountSort()` algorithm to work with arbitrary data types by supplying the hash function as a parameter. In this sense, the algorithm also functions as a partitioning algorithm, as the hash function effectively assigns each element to a distinct bucket within the fixed range.**

  In C++, a function may be passed as an argument to another function either through a function pointer parameter or through a generic type parameter. The latter approach is typically preferred because it supports all possible callable types (including function pointers, functors, and lambda expressions) and provides greater flexibility and type safety.

- **The objective is to define a `CountSort()` function that can operate on arbitrary data types by incorporating a user-provided hash function. The function should accept a generic container (such as a vector), a hash function, and a hash size (the number of possible hash values). After implementing the algorithm, the goal is to simulate its execution and validate its correctness using several test examples.**

- **Your source code must compile successfully and may only include the following libraries: `'iostream'`, `'sstream'`,`'iomanip'`, `'vector'`, `'cctype'`, and `'string'`. No credit will be given if additional libraries are used.**

- **A cumulative task will not receive credit if the required previous tasks are not completed.**

- **Your submissions must be submitted to the GitHub repository in the Lab05 directory.**

- **Cheating of any kind is prohibited and will not be tolerated.**

- **Violating or failing to follow any of the rules above will result in an automatic zero (0) for the lab.**

Grading

Task	Maximum Points	Points Earned
1	1	
2	1	
3	1	
4	1	
5	1	
Total	5	

Note: solutions will be provided for tasks colored blue only.

Task 1

- Create a header file named '`Algorithm.h`' that defines a C++ function named `toString()` that takes a constant generic vector reference parameter and returns a string representation of its elements enclosed in square brackets.

Task 2

- Within the header file '`Algorithm.h`' created in Question 1, define a C++ function `HashCountSort()` that accepts a generic vector reference, a hash function, and a hash size as parameters. The function should implement the `CountSort()` algorithm using the provided hash function with the hash size as k.

Task 3

- Create a text file named '`simulateA.txt`' that simulates the execution of your algorithm from Question 2 for the function call `HashCountSort(A, hash, 27)` where `A = ['R','e','I','N','V','O','K','E','S']` and the hash function is defined as:

$$\texttt{hash(key)} = \begin{cases} \texttt{key - 'A'} & \text{if key is uppercase} \\ \texttt{key - 'a'} & \text{if key is lowercase} \\ 26 & \text{otherwise} \end{cases}$$

  In the file, simulate and record the changes to the array during each iteration of the final loop of the `HashCourtSort()` algorithm.

  Example:

  For the invocation `HashCountSort(A,hash,27)` where `A = ['P','a','c','E']` and `hash()` has the above definition, the simulations would be

  1. `['P','a','c','E']`
  2. `['P','a','E','E']`
  3. `['P','c','E','E']`
  4. `['a','c','E','E']`
  5. `['a','c','E','P']`

Task 4

- Create a text file named '`simulateB.txt`' that simulates the execution of your algorithm from Question 2 for the function call `HashCountSort(A, hash, 10)` where `A = [28,52,99,53,41,71,84,83,32]` and the hash function is defined as:

```
hash(key) = key % 4
```

  in the same manner as Question 3.

Task 5

- Create a C++ file named '`test.cpp`' that includes '`Algorithm.h`' and verifies the outputs produced in Questions 3 and 4. For each test array, display the array both before and after invoking `HashCountSort()`.