Signature Assignment

This assignment will exercise the topics learned throughout the course. The goal is to develop a set of analytical functions that determine various aspects of a data set. Each module has a recommended set of work to achieve to build toward a final solution. While it is *optional* to do the work in this order, it is highly *recommended* to follow this outline.

Each module will have a set of function headers or class methods that should be used to guide implementation. If you need additional or fewer functions or methods, feel free to follow your preferences. Again, these are recommended, not required—the results are what will be graded.

Module 1 Recommended Implementation Details

In this first part of the signature assignment, develop code to write and read a binary file. An array of random numbers will be created, and the number of elements and values of each element will be written into a binary file. Then, to ensure that all is working well, the array is deleted, a new one is created, and the binary file is read. The first value is the number of elements to read, which value should control a loop. Then, each value is read and stored in an array element.

Write a function named `createArray` that creates a dynamic array of 1000 integers. The `main` function must supply an argument for the length of the array. Use a constant defined with the value of 1000 in the calling code. It is best to define the constant outside the main function. In `createArray`, initialize the array with random numbers between 0 and 999 inclusive. Return a pointer to the dynamically created array to the calling code in `main`.

Write a second function named `writeBinary` that takes as a parameter a pointer to an integer array and a second parameter, which is the array's length. Open a binary output file named "binary.dat" as output and write to it an integer indicating the array's length (the value of the parameter). Then, in a loop, write each element in the array to the file and close the file. Be sure to use a `reinterpret_cast` to convert the location of each integer to a constant character pointer.

Write a third function named `readBinary` that opens the binary file "binary.dat" file for input. Its only function parameter should reference an integer in the main function that will store the length of the returned array—the length must be read from the file, not passed as a parameter. Read the first value from the binary file, which is an integer representing the length of the new array to create; use the address of the reference parameter as the target of a `reinterpret_cast`. Next, create a new dynamic integer array using the length value for the size. In a loop, read each number from the binary file and save the value in the array. Return the pointer to the array from the function.

In the `main` function, call `createArray,` passing the array's length (use the defined constant). This function will return a pointer to the array, which should be saved in a local

variable. Call the `writeBinary` function, passing the pointer to the array as the first argument and the length as a second argument. After `writeBinary` returns, delete the array created in the first function and set its pointer to `nullptr`.

Define a local integer variable for an array's length (do not use the defined constant) and an integer pointer for the array itself. Call the `readBinary` function, passing a reference to the variable for the array length. The function will return a pointer to a new array, which should be saved, and the length variable will be updated. Write a loop to iterate over each value in the array and print out its value separated by a space. Delete the dynamic array and terminate the program.

Use these function headers:

```
int* createArray(int length)
int* readBinary(int& length)
void writeBinary(int* values, int length)
```

Example output:

```
383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 …
…
308 593 278 197 555 672 774 445 0 325 997 283 412 127 382 421
```

Module 2 Recommended Implementation Details

This second part of the signature assignment will rework the code to create the binary file. Then, you will develop classes for reading and analyzing the binary data read from the file.

Write a function named `createBinaryFile,` which takes the name of a file as its first parameter and the size of the data as the second parameter and returns nothing. Copy the code from the `createArray` function for creating the array of random values into this function. Modify `writeBinary` to take an additional string parameter; it is the file name. Add a call to the existing `writeBinary` function and pass three parameters. When this code works correctly, eliminate the `createArray` function since it is no longer needed.

Write a class called `BinaryReader`. It should have instance variables of an integer pointer and an integer. The pointer will point to an array of integers; named `values`. The integer will hold the array's size, named `size`. Add a constructor that takes a single parameter, a constant string reference to a file name. Add a private method called `readValues` with one parameter, the name of the input file. Copy the code from the `readBinary` function to open the file, input the array size into the `size` instance variable, and create the dynamic array using the `values` instance variable. Now, read the values from the file into the array. Eliminate the existing `readBinary` function. The constructor should call the `readValues` method to populate the

array and size instance variables. Add a destructor to delete the integer array. Finally, add `getValues` and `getSize` as public methods that return the instance variables.

Write a second class called `Analyzer`. This class has two instance variables: an integer pointer representing an array of integers named `values` and an integer representing the array's size named `size`. Add a constructor with two parameters: an integer pointer and an integer. Add a private `cloneValues` method that creates an integer array, copies the values from the array parameter into its elements, copies the size, and returns nothing. The constructor should call the `cloneValues` method, passing it the two parameter values. Add a destructor to delete the dynamic array. Add an `analyze` method that determines the mean (mathematical average), minimum value, and maximum value from the values in the array. Be certain to return a double value for the mean value. Return a string that holds this statistical information.

Update the `main` function to call `create binary file,` passing the name as "binary.dat." Create a `BinaryReader` instance and write a loop that iterates over the values from the instance to ensure it works correctly. After it is working, the loop should be removed. Next, create an instance of the `Analyzer` class, passing its constructor the values and size using the `BinaryReader`'s get methods for its values and size. Call the `analyze` method and print out the results.

Use these function and class headers:

```
class Analyzer
    void cloneValues(int* values, int size)
    Analyzer(int* values, int size)
    ~Analyzer()
    std::string analyze()

class BinaryReader
    void readValues(const std::string& name)
    BinaryReader(const std::string& name)
    ~BinaryReader()
    int* getValues()
    int getSize()

void createBinaryFile(const std::string& name, int length)
void writeBinary(int* values, int length, const std::string&
name)
```

Example output:

```
The minimum value is 0
The maximum value is 999
The mean value is 499.495
```

Module 3 Recommended Implementation Details

In this third part of the signature assignment, you will develop code for additional classes to analyze the data. Inheritance will be used to relate the analyzer classes into a hierarchy.

Modify the `Analyzer` class to make it a base class by changing the `analyze` method to make it a pure virtual function; adding a virtual function means the class is intended to be a base class. Change the `analyze` method to virtual. Because `Analyzer` has at least one virtual function, the destructor should be marked virtual; its implementation remains the same since the class owns the integer array. Change the `values` and `size` variables from `private` to `protected`.

Create a class called `DuplicateAnalyser` that publicly inherits from `Analyzer`. This class will scan the data and determine which values appear more than once. Add a constructor that takes an integer array and its size. The constructor should call the base class constructor and pass these parameter values. Implement the `analyze` method to count the duplicated values; use the `override` keyword appropriately. Return a `std::string` with explanatory text and the count of duplicates. Only some values will be duplicated, and others will not exist or appear only once.

Create a second class called `MissingAnalyzer` that publicly inherits from `Analyzer`. This class will scan the data and determine which values do not appear. Add a constructor that takes an integer array and its size. The constructor should call the base class constructor and pass these parameter values. Implement the `analyze` method to count the missing values; use the `override` keyword appropriately. Return a `std::string` with explanatory text and the count of missing values. Only some values will be missing, and others will be duplicated or appear only once.

Create a third class called `StatisticsAnalyzer,` which performs the same statistical analysis of the data as the `Analyzer` class did in Part 2. This includes the minimum, maximum, and mean values. Add a constructor that takes an integer array and its size. The constructor should call the base class constructor and pass these parameter values. Implement the `analyze` method using the code from the `Analyzer` class and use the `override` keyword appropriately.

Modify the `Analyzer` class to remove the implementation of the `analyze` method and make it a pure virtual function.

Modify the `main` function to create an instance from each of the `StatisticsAnalyzer`, `DuplicateAnalyzer`, and `MissingAnalyzer` classes. Call the `analyze` method for each instance and print out the returned results.

Use these function and class headers:

```
class Analyzer
    void cloneValues(int* values, int size)
    Analyzer(int* values, int size)
    virtual ~Analyzer()
    virtual std::string analyze() = 0

class StatisticsAnalyzer : public Analyzer
    StatisticsAnalyzer(int* values, int size)
    std::string analyze() override

class DuplicateAnalyzer : public Analyzer
    DuplicateAnalyzer(int* values, int size)
    std::string analyze() override

class MissingAnalyzer : public Analyzer
    MissingAnalyzer(int* values, int size)
    std::string analyze() override
```

Example output:

```
The minimum value is 0
The maximum value is 999
The mean value is 499.495
There were 269 duplicated values
There were 361 missing values
```

Module 4 Recommended Implementation Details

In this fourth part of the signature assignment, code is developed for sorting and searching using a binary search algorithm.

Write a function named `selection_sort`, which takes two parameters: an integer pointer and an integer size. Search the Internet or consult a textbook for an example of selection sort. Add code to implement a selection sort algorithm. Add any needed functions to support the selection sort. The function should sort the array in place and return nothing.

Write a recursive function named `binary_search_recursive,` which takes four parameters from a helper function. The first is an integer pointer representing the array of values, the second is an integer representing the value to be searched for, the third is an integer

for the starting index, and the fourth is an integer for the ending index. The function returns a Boolean value representing if the searched-for value was found.

Write a helper function named `binary_search` that takes three parameters: the first is a pointer to an integer array, the second is the array size, and the third is a searched-for value. This function calls the recursive function and passes the array, the search value, the starting index, and the ending index; remember, the size and the ending index are related but not equivalent. The function returns the Boolean value returned by the recursive search function.

Add a new class named `SearchAnalyzer,` which subclasses Analyzer and implements the `analyze` method. Use the override keyword appropriately. The constructor should first initialize the base class by calling the Analyzer constructor, then in the constructor body, call the `selection_sort` function, passing the instance variable `values` and the `size` (do not pass the parameter values and size!). In the `analyze` method, generate 100 random integer values from 0 to 999 and use the `binary_search` function to see if that value exists in the data. Count the number of found values and return the count in a `std::string`.

Modify the `StatisticsAnalyzer` constructor to call the `selection_sort` function, passing it the correct argument values. Alter the minimum and maximum values computation, considering the array is now sorted. Add functionality to implement the median and mode averages using the sorted data. The median is the exact middle value if there is an odd number of elements, and the mean of the two middle values is if there is an even number of elements. The mode is the most frequently occurring value; if more than one value occurs with this frequency, pick the first one.

The final main function should look like this:

```
const int SIZE = 1000;
int main() {
    createBinaryFile("binary.dat", SIZE);
    BinaryReader br("binary.dat");

    StatisticsAnalyzer sa(br.getValues(), br.getSize());
    std::cout << sa.analyze() << '\n';
    DuplicateAnalyzer da(br.getValues(), br.getSize());
    std::cout << da.analyze() << '\n';
    MissingAnalyzer ma(br.getValues(), br.getSize());
    std::cout << ma.analyze() << '\n';
    SearchAnalyzer ra(br.getValues(), br.getSize());
    std::cout << ra.analyze() << '\n';

    return 0;
}
```

Use these function and class headers:

```
class SearchAnalyzer : public Analyzer
    SearchAnalyzer(int* values, int size)
    std::string analyze() override

void selection_sort(int* values, int size)
bool binary_search_recursive(int* values, int key, int start,
int end)
bool binary_search(int* values, int size, int key)
```

Example output:

```
The minimum value is 0
The maximum value is 999
The mean value is 499.495
The median value is 497
The mode value is 81 which occurred 5 times
There were 269 duplicated values
There were 361 missing values
There were 62 random values found
```