CS 115 Final Review Quiz

Rules

- Reference material on the STL classes and algorithms, as well as functions that operate on C-strings, is included at the end of this document.
- You must briefly explain your answers to receive partial credit.
- When a snippet of code is given to you, you can assume that the code is enclosed within some function, even if no function definition is shown. You can also assume that the `main` function is properly defined and that the `iostream`, `fstream`, `iomanip`, `vector`, `algorithm`, `string`, `cstring`, and `cmath` libraries have been included at the beginning of the program.
- When you are asked to write *a snippet* of code, you may also assume that it is enclosed within some function that any necessary libraries have been included.
- When you are asked to write *a complete program*, you must write the `#include` statements, the `int main()`, etc. in your solution for full credit.
- A line consisting solely of "..." represents one or more unspecified C++ statements, some of which may change the values of program variables.

Problem 1: 15 points.

Match the following descriptions with the term they describe by writing that term in the space provided. The choices of terms are:

- this
- constructor
- destructor
- class
- object
- pointer
- vector
- overloading
- operator

Not all terms will be used.

(a) An example of an STL container

(b) A function that is automatically called when an object is created

(c) A variable whose data type is a class

(d) A variable containing the address of another variable

(e) A pointer to the object whose member function is being called

Problem 2: 10 points.

Write a **complete program** that meets the following requirements:

- The program should be run with one command-line argument. For example, if the executable file is called program.out, you might type:
  ```
  ./program.out hello
  ```
  at the Linux command line.
- If too many or too few command-line arguments are supplied, your program should print an error message and exit.
- If the correct number of command-line arguments is supplied, your program should print "You said" followed by the command-line argument. For the example above, your program should print out:
  ```
  You said hello
  ```

Problem 3: 10 points.

Write a *snippet* of code that does the following:

- Declares a vector of integers
- Adds the integers 2, 3, and 4 to the vector (in that order).
- Reverses the vector
- Prints the first element of the vector
- Prints the size of the vector

Remember to use the reference material at the end of this test.

Problem 4: 10 points.

(a) What's wrong with the following snippet of code? (You don't have to fix it; just clearly state what's wrong.)

```
/* Declare a "Person" class for use by the rest of the
program. */
class Person {
     string firstName;
     string lastName;
};
```

(b) Assume that the `Word` class has been declared with the following private data members (in addition to a bunch of public functions):

```
private:
     char * data; int length;
```

You are in charge of defining what the == operator means for this class; you decide that `(w1 == w2)`, where `w1` and `w2` are words, will return `true` if and only if the data fields of `w1` and `w2` are equal. Otherwise, it will return `false`. Someone has already written the skeleton for you; you just need to fill it in. For your reference, the definition of the + operator is also shown. **Remember that you can't use == to directly compare C-strings!**

```
// Addition operator
Word Word::operator + (const Word& other) const {
   Word temp;
   temp.length = this->length + other.length;
   temp.data = new char [temp.length + 1];
   strcpy (temp.data, this->data);
   strcat (temp.data, other.data);
   return temp;
}
// Equality operator: fill this in
bool Word::operator == (const Word& other) const {




}
```

Problem 5: 15 points.

For this problem, you must write a **class definition** for a class named **`Grades`** that contains the following:

Note that data members should be private and member functions should be public.

- A student's name (as a `string`)
- A student's test average and homework average (as two variables of type `double`)
- Prototype for a default constructor
- Prototype for a function called `SetGrades`. This function will take two variables of type `double` as inputs and will return a `bool`.
- Prototype for a function called `SetName`. This function will take a `string` as input and will not return anything.
- Prototype for a function called `GetAvg`. This function will return the student's overall average based on his/her test average and homework average.
- Prototype for a function called `IsPassing`. This function will return `true` if the student's overall average is passing and `false` otherwise.

Problem 6: 20 points.

In this problem, you will write definitions for the functions in the class `Grades`. Here is a little bit more information about the functions. You may assume that `<string>` is included. *Note that none of your code for this problem should include* `cin` *or* `cout` *statements!*

- The default constructor will initialize the name to `""` (the empty string), the test average to 0, and the homework average to 0.
- The `SetGrades` function will work as follows:
    - If one or both of the inputs is less than zero or greater than 100, it will return `false`.
    - Otherwise, it will set the test average equal to the first input and the homework average equal to the second input. Then it will return `true`.
- You do not have to write the `SetName` function, but you can assume in Problem 7 that someone else has provided it.
- The `GetAvg` function will return the student's overall average. Tests are 60% of the overall grade, and homework is 40% of the overall grade. In other words:
  *Overall Grade = 0.6 * (test average) + 0.4 * (homework average)*
- The `IsPassing` function will return `true` if the student's overall average is greater than or equal to 60, and `false` otherwise. This function should call `GetAvg` in order to compute the average, rather than duplicating code.

Problem 7: 20 points.

Assume that the class definitions you wrote in Problems 5 and 6 are located in a file called `grades.h` in the same directory as the program you're about to write.

Write **a complete program** that uses the `Grades` class from `grades.h` to do the following:

- Create one `Grades` object with the student name "Alice", a test average of 75, and a homework average of 100.
- Create another `Grades` object with the student name "Bob." Ask the user to supply Bob's test and homework averages and update the object with that information. You can assume that the user's input is valid.
- Using the member function `isPassing`, print out the number of students who are passing the class (0, 1, or 2).

REFERENCE

C-string functions:

`strlen`	Input is a C-string. Returns the length of the string (not including the null terminator). Usage example: `length = strlen(name);`
`strcat`	Input is two C-strings. Appends the second string to the end of the first string (the first string is changed, but the second is not). Usage example: `strcat(str1, str2);`
`strcpy`	Input is two C-strings. Copies the second string to the first string, overwriting the original contents. Usage example: `strcpy(str1, str2);`
`strcmp`	Input is two C-strings. Returns 0 if they are the same, a negative number if `str2` is alphabetically greater than `str1`, and a positive number if `str1` is greater than `str2`. Usage example: `if(strcmp(str1, str2) > 0)`

Vector member functions:

`begin()`	Returns an iterator to the vector's first element
`clear()`	Removes all elements from the vector
`empty()`	Returns a bool which is true if the vector is empty (has zero elements) and false otherwise
`end()`	Returns an iterator pointing just past the vector's last element
`pop_back()`	Removes the last element from the vector
`push_back(value)`	Inserts value as a new element at the end of the vector
`size()`	Returns the number of elements in the vector

STL algorithms (NOT member functions)
Here *iter1* and *iter2* are iterators pointing to elements of an STL class such as `vector`.

`binary_search(iter1, iter2, value)`	Returns `true` if `value` is found in the range between `iter1` and `iter2`, `false` otherwise
`count(iter1, iter2, value)`	Returns the number of times `value` appears in the range between `iter1` and `iter2`
`reverse(iter1, iter2)`	Reverses the order of the elements between `iter1` and `iter2`
`random_shuffle(iter1, iter2)`	Randomly changes the order of the elements between `iter1` and `iter2`
`sort(iter1, iter2)`	Sorts the elements in the range between `iter1` and `iter2` in ascending order