Assignment 5
C/C++ Programming I

C1A5 General Information

--- No General Information for This Assignment---

Get a Consolidated Assignment 5 Report (optional)

If you would like to receive a consolidated report containing the results of the most recent version of each exercise submitted for this assignment, send an empty email to the assignment checker with the subject line **C1A5_*ID***, where ***ID*** is your 9-character UCSD student ID. Inspect the report carefully since it is what I will be grading. You may resubmit exercises and report requests as many times as you wish before the assignment deadline.

C1A5E0 *(6 points total - 1 point per question – No program required)*

Assume language standards compliance and any necessary support code unless stated otherwise. Testing erroneous or implementation dependent code by running it can be misleading. These are not trick questions and each has only one correct answer. Major applicable course book notes are listed.

1. The declaration ***int** (*(*x))());* in English is: (Note 6.1)
   A. "x is a function returning a pointer to a function returning a pointer to an int"
   B. "x is a pointer to a function returning a pointer to an int"
   C. "x is a pointer to a function returning a pointer to a function returning an int"
   D. "x is a function returning a pointer to a function returning an int"
   E. This is not a valid declaration!

2. Which is the most likely output from:
```
const int save[] = {1, 2, 3, 4};
for (int index = 0; index < 6; ++index)
        cout << save[index] << ' ';
```
   (Note 6.2)
   A. *1 2 3 4 5 6*
   B. *1 2 3 4* then two garbage values and/or a system error occurs
   C. *1 2 3 4 5* then a garbage value and/or a system error occurs
   D. *1 2 3 4 0 0*
   E. *1 2 3 4*

3. What is the most important problem with these two successive lines of code?
```
double *p;
*p = 47E3;
```
   (Notes 6.6 & 6.7)
   A. *p* is initialized to a null pointer by default.
   B. *p* is uninitialized.
   C. Pointers to **double**s are not reliable in standard C/C++.
   D. 47E3 is not a legal address.
   E. There is no significant problem!

4. For:
```
int x = 5, y = 10;
fcnA(&x, &y);
```
   (Note 6.9)
   A. **int** types are being passed.
   B. C++ reference types are being passed.
   C. The types of *x* and *y* are implementation dependent.
   D. *fcnA* can't change the value of *x* or *y*.
   E. Pointer types are being passed

5. If *fcnA* only does *return(*iP1 + *iP2)* what, if anything, is seriously wrong with:
```
int fcnA(int *iP1, const int *iP2);
const int x = 5, y = 10;
fcnA(&x, &y);
```
   (Note 6.10)
   A. ***const int** * is legal in C++ but not C.
   B. ***int const** *iP2* is illegal in C and C++
   C. The right argument causes an error.
   D. The left argument causes an error.
   E. There are no reference types in C.

6. What is wrong with:
```
int *fcnA(int y)
{
    int x = y;
    return(&x);
}
```
   (Note 6.12)
   A. An automatic variable is returned.
   B. A reference to an automatic variable is returned.
   C. A pointer to an automatic variable is returned.
   D. **return**(&x) should be **return**(*x)
   E. Nothing is wrong.

Submitting your solution

Using the format below place your answers in a plain text file named **C1A5E0_Quiz.txt** and send it to the Assignment Checker with the subject line **C1A5E0_*ID***, where ***ID*** is your 9-character UCSD student ID.

```
-- Place an appropriate "Title Block" here --
1. A
2. C
etc.
```

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

C1A5E1 *(6 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one, naming it **C1A5E1_main.c**. Write a program in that file to implement a survey that prompts respondents to enter a decimal integer value within a specified range to indicate how much they like a product. The survey will end and the results will be displayed when either a specific number of legal responses have been entered or a specific number of consecutive out-of-range responses have been entered.

Since the program must be adaptable to any arbitrary number of respondents, response values, and out-of-range counts, define and use the following 4 required macros wherever this information is needed:

1. `MAX_RESPONDENTS` – the maximum number of respondents;
2. `MIN_RESPONSE_VALUE` – the lowest permissible response value;
3. `MAX_RESPONSE_VALUE` – the highest permissible response value;
4. `OUT_OF_RANGE_LIMIT` – the number of consecutive out-of-range responses required to end the survey.

You may define additional macros if you wish, but it must not be necessary to explicitly change any of their values if the values associated with any of the 4 required macros above are changed.

The code in function `main` must conduct the survey and display the results as follows:

1. Use a 1-dimensional array having exactly the number of elements as there are legal response values. Each element represents a specific legal response value and contains a count of how many of them have occurred (similar to note 6.3). No other arrays are permitted.
2. Use a variable named `consecutiveRangeErrors` to count the number of consecutive out-of-range responses:
   a. If an illegal response occurs increment and test that variable. If the maximum count has not yet been reached display an error message and re-prompt the same user; if it has been reached end the survey.
   b. If a legal response occurs set the variable back to 0 and increment the appropriate array element. End the survey if the maximum number of respondents have entered legal responses.
3. Once the survey has ended display a table in descending rating order like the one below. It indicates the total number of responses for each possible rating. DO NOT put any blank lines or other dividers between table entries. The least significant digits of all values must be aligned for readability:

```
Rating       Responses
-------      ---------
   10           25
    9           50
    8          100
  ...          ...
-1239            0
```

Your program must work correctly for all cases where `MIN_RESPONSE_VALUE` is less than or equal to `MAX_RESPONSE_VALUE`, regardless of their actual values. This includes cases where one or both are negative.

NOTE:
If you think you need an array having `MAX_RESPONDENTS` elements you have not understood the requirements above and are totally on the wrong track.

Manually re-run your program several times with at least the following sets of macro values and a sufficient number of user input values to verify functionality. To test with different macro values you will, of course, need to recompile after each change:

(See next page.)

Test Value Sets				
Test Set	MAX_ RESPONDENTS	MIN_RESPONSE_ VALUE	MAX_RESPONSE_ VALUE	OUT_OF_RANGE_ LIMIT
A	3	0	0	4
B	20	3	15	2
C	5	-100	-86	3
D	17	-27	9	1

Submitting your solution

Send your source code file to the Assignment Checker with the subject line **C1A5E1_*ID***, where ***ID*** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

Hint:

See note 6.3 for a similar exercise. Do not initialize your automatic array to all 0s using a loop but instead initialize it when declared using the standard syntax:

```
int ratingCounters[RESPONSE_VALUES] = {0};
```

Be sure the array contains the correct number of elements. For example, if MIN_RESPONSE_VALUE were -3 and MAX_RESPONSE_VALUE were 7, the number of elements required would be 11. Also, be sure your algorithm can support negative response values and response ranges that do not start or end with 0. This can easily be accomplished by always subtracting MIN_RESPONSE_VALUE from the user input response value when using it as an index into the array. Finally, be sure you reset your "bad response" count to its initial value each time a correct response is received.

C1A5E2 *(4 points – C++ Program)*

Exclude any existing source code files that may already be in your IDE project and add three new ones, naming them **C1A5E2_ComputeMinimum.cpp**, **C1A5E2_ComputeMaximum.cpp**, and **C1A5E2_main.cpp**. Do not use `#include` to include any of these three files in each other or in any other file. However, you may use it to include any appropriate header file(s) you need.

File **C1A5E2_ComputeMinimum.cpp** must contain a function named `ComputeMinimum` and **C1A5E2_ComputeMaximum.cpp** must contain a function named `ComputeMaximum`. Each function must:

1. have exactly two formal parameters, each of type "reference to **const double**";
2. return type "reference to **double**" (not "reference to **const double**");
3. contain only one statement;
4. not use variables other than its formal parameters;
5. not use anything that requires `#define` or `#include`;
6. not use literal values;
7. not do assignment, addition, subtraction, multiplication, or division;
8. not use `if`, `switch`, or looping statements;
9. not call functions or macros;
10. not display anything.

`ComputeMinimum` must compare the values referenced by its parameters and return a reference to the smallest of those values whereas `ComputeMaximum` must compare the values referenced by its parameters and return a reference to the greatest of those values.

File **C1A5E2_main.cpp** must contain function `main`, which must:

1. use no more than two variables;
2. prompt the user to enter any space-separated pair of decimal numeric values on the same line;
3. pass references to the user-entered values to both `ComputeMinimum` and `ComputeMaximum` as arguments;
4. display the results of both function calls using the following 2-line format, where the question marks represent the values whose references are passed to and returned from the functions:

```
ComputeMinimum(?, ?) returned ?
ComputeMaximum(?, ?) returned ?
```

Do not attempt to detect cases where the user input values are equal. Instead, simply treat them exactly like any other values.

Manually re-run your program several times, testing with at least the following 5 sets of user input values, where each set represents the argument values in left-to-right order:

6.9 6.4 6.4 6.9 -5.8 5.8 -0.0 0.0 8.4e3 6.2e-1

Submitting your solution

Send your three source code files to the Assignment Checker with the subject line **C1A5E2_*ID***, where ***ID*** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

Hints:

Use the conditional operator to both compare the values referenced to by each parameter and produce the reference that references the greatest value. Return the entire conditional expression (Note 3.16), type casting it as ***(double &)*** to override the "**const**-ness". (Same principle as in note 6.12).

C1A5E3 *(4 points – C++ Program)*

Exclude any source code existing files that may already be in your IDE project and add three new ones, naming them **C1A5E3_ComputeMinimum.cpp**, **C1A5E3_ComputeMaximum.cpp**, and **C1A5E3_main.cpp**. Do not use `#include` to include any of these three files in each other or in any other file. However, you may use it to include any appropriate header file(s) you need.

File **C1A5E3_ComputeMinimum.cpp** must contain a function named `ComputeMinimum` and **C1A5E3_ComputeMaximum.cpp** must contain a function named `ComputeMaximum`. Each function must:

1. have exactly two formal parameters, each of type "pointer to **const double**";
2. return type "pointer to **double**" (not "pointer to **const double**");
3. contain only one statement;
4. not use variables other than its formal parameters;
5. not use anything that requires `#define` or `#include`;
6. not use literal values;
7. not do assignment, addition, subtraction, multiplication, or division;
8. not use `if`, `switch`, or looping statements;
9. not call functions or macros;
10. not display anything.

`ComputeMinimum` must compare the values pointed to by its parameters and return a pointer to the smallest of those values whereas `ComputeMaximum` must compare the values pointed to by its parameters and return a pointer to the greatest of those values.

File **C1A5E3_main.cpp** must contain function `main`, which must:

1. use no more than two variables;
2. prompt the user to enter any space-separated pair of decimal numeric values on the same line;
3. pass pointers to the user-entered values to both `ComputeMinimum` and `ComputeMaximum` as arguments;
4. display the results of both function calls using the following 2-line format, where the question marks represent the values whose pointers are passed to and returned from the functions and the ampersands are displayed literally to remind the user that pointers are being passed and returned, not the values themselves:

```
ComputeMinimum(&?, &?) returned &?
ComputeMaximum(&?, &?) returned &?
```

Do not attempt to detect cases where the user input values are equal. Instead, simply treat them exactly like any other values.

Manually re-run your program several times, testing with at least the following 5 sets of user input values, where each set represents the argument values in left-to-right order:

6.9 6.4 6.4 6.9 -5.8 5.8 -0.0 0.0 8.4e3 6.2e-1

Submitting your solution

Send your three source code files to the Assignment Checker with the subject line **C1A5E3_*ID***, where ***ID*** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

Hints:

Use the conditional operator to both compare the values pointed to by each parameter and produce the pointer that points to the greatest value. Return the entire conditional expression (Note 3.16), type casting it as ***(double *)*** to override the "**const**-ness". (Note 6.12).

