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-body email to the assignment checker with the subject line **C1A5_162461_U09339367** and no attachments.

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 standard library support unless stated otherwise. These are not trick questions and there is only one correct answer, but basing an answer on runtime results is risky. Place your answers in a plain text "quiz file" named **C1A5E0_Quiz.txt** formatted as:

a "Non-Code" Title Block, an empty line, then the answers:
1. A
2. C
etc.

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

`Send an empty-body email to the assignment checker with the subject line **C1A5E0_162461_U09339367** and with your quiz file attached.

See the course document titled "How to Prepare and Submit 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 code in that file's `main` function that implements a survey of how shoppers like a product by prompting them to enter a decimal integer rating value within an allowed range of `BEST` to `WORST`. If either all shoppers have entered an in-range value or any one shopper enters a specified number of out-of-range values, the function must display a table of survey results and then return.

Define the following five macros to have any integer values within the stated constraints and use them appropriately in your code, but your code must also work with any other integer values within those constraints. Although defining additional macros is unnecessary, you may do so as long as it is not necessary to explicitly change their values if the values of any of the required macros are changed:

`SHOPPERS` – number of participating shoppers (any value > 0)

`BEST` – highest allowed rating value (any negative, zero, or positive value)

`WORST` – lowest allowed rating value (any value <= `BEST`)

`CHOICES` – the number of integer values from `BEST` down to `WORST`
For example, if `BEST` is `3` and `WORST` is `-2`, the value of `CHOICES` will be `6`. The value of `CHOICES` <u>must be</u> derived from `BEST` and `WORST` and <u>not</u> specified as a literal number.

`MAX_ERRORS` – number of out-of-range ratings from any one shopper that ends prompting (> 0)

Your code must:

1. Use a one-dimensional automatic type `int` array having exactly `CHOICES` elements. No other arrays are permitted. Each element represents an allowed rating and will contain a count of how many of that rating have been entered so far (similar to note 6.3).
2. Initially output a message that indicates the allowed rating range and the `MAX_ERRORS` value. Then loop to individually prompt shoppers to enter their ratings. Use a variable named `errors` to keep count of the current shopper's out-of-range ratings:
   - If an in-range rating is entered, set `errors` back to 0, increment the array element that represents that rating, and prompt the next shopper.
   - If an out-of-range rating is entered, notify the shopper, increment `errors`, then compare it to `MAX_ERRORS`. If they are not equal, reprompt the same shopper. Else, if they are equal, go to step 3 below.
3. Return from `main` after first displaying a table like that below. The table must list <u>all</u> allowed rating values in the "Rating" column and the number of each that shoppers entered in the "Quantity" column. Entries must be in <u>best to worst rating order</u> and the least significant digits of all values must be aligned:

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

- If you believe you need more than one array or an array having other than `CHOICES` elements, you have not understood the requirements and should reread them as well as note 6.3.
- <u>Do not</u> test if the values of the required macros are actually valid.
- <u>Do not</u> use a loop or call a function to initialize your array to 0s.
- <u>Do not</u> produce the absolute value of anything.

Manually re-run your program several times with at least the sets of macro values shown on the next page, using a sufficient assortment of shopper rating values to verify functionality. To test with different macro values you will need to recompile after each change.

Test Value Sets				
Set	SHOPPERS	WORST	BEST	MAX_ERRORS
A	3	-3	-3	3
B	3	0	0	4
C	20	3	15	2
D	5	-100	-86	3
E	17	-27	9	1

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A5E1_162461_U09339367** and with your source code file attached.

See the course document titled "How to Prepare and Submit Assignments" for additional exercise formatting, submission, and assignment checker requirements.

Hints:

1. See note 6.3 for a similar exercise.

2. Whenever appropriate the best way to initialize an automatic array to all zeros is by using the standard syntax:
   ```
   int ratings[CHOICES] = {0};
   ```

3. During runtime testing the assignment checker will change the values your code uses for the `SHOPPERS`, `WORST`, `BEST`, and `MAX_ERRORS` macros as well as the shopper ratings responses. Make sure all other code remains valid regardless of the values (within the stated constraints) the assignment checker uses.

4. Be sure your array contains the correct number of elements. For example, if `WORST` is `-7` and `BEST` is `20`, the number of elements required (and the value of `CHOICES`) will be `28`.

5. Be sure your algorithm can support negative rating values and rating ranges that do not start or end with zero.

6. Be sure to reset your consecutive range error count back to zero each time an allowed rating is accepted.

7. Look up the `scanf` function online or in any C textbook to learn about its return value.

C1A5E2 (4 points – C++ Program)

Exclude any 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. be declared to 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. use no casts.
3. prompt the user to enter two space-separated decimal values on the same line.
4. pass references to the user-entered values to both `ComputeMinimum` and `ComputeMaximum` as arguments.
5. 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 ?
   ```
   For example, if the user enters **-5.8 5.8** the result should be:
   ```
   ComputeMinimum(-5.8, 5.8) returned -5.8
   ComputeMaximum(-5.8, 5.8) returned 5.8
   ```

- Do not attempt to detect cases where the user input values are equal. Instead, simply treat them exactly like any other values.
- Scientific and standard notation are both okay and may be mixed.
- Zeros that don't affect a fractional part's value may be omitted.
- If a fractional part is empty the decimal point may be omitted.

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 an empty-body email to the assignment checker with the subject line **C1A5E2_162461_U09339367** and with your three source code files attached.

See the course document titled "How to Prepare and Submit 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 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. be declared to 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. use no casts.
3. prompt the user to enter two space-separated decimal values on the same line.
4. pass pointers to the user-entered values to both `ComputeMinimum` and `ComputeMaximum` as arguments.
5. 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 &?
   ```
   For example, if the user enters **-5.8 5.8** the result should be:
   ```
   ComputeMinimum(&-5.8, &5.8) returned &-5.8
   ComputeMaximum(&-5.8, &5.8) returned &5.8
   ```

- Do not attempt to detect cases where the user input values are equal. Instead, simply treat them exactly like any other values.
- Scientific and standard notation are both okay and may be mixed.
- Zeros that don't affect a fractional part's value may be omitted.
- If a fractional part is empty the decimal point may be omitted.

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 an empty-body email to the assignment checker with the subject line **C1A5E3_162461_U09339367** and with your three source code files attached.

See the course document titled "How to Prepare and Submit 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).