Assignment 4
C/C++ Programming I

C1A4 General Information

Assignment 4 consists of FIVE (5) exercises:

C1A4E0 C1A4E1 C1A4E2
C1A4E3 C1A4E4

All requirements are in this document.

Related examples are in a separate file.

Get a Consolidated Assignment 4 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 **C1A4_167109_U09609277** 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.

C1A4 General Information, continued

To Refresh Your Memory

Source Code file – A file containing only code appropriate for the programming language being used.

Header file – A source code file designed to be included in another file using `#include` rather than being compiled directly. The most common extension for header files is `.h` except for C++ standard library header files, which typically have no extension at all.

Implementation file – A source code file designed to be compiled directly rather than being included in another file. The most common extension for C implementation files is `.c` whereas for C++ it is `.cpp`.

Writing Programs Using Multiple Source Code Files

All exercises in this assignment require that you write multiple functions and place them in separate source code files. This is typical of the way all but the simplest of professional programs are organized and is much more versatile than putting everything into just one file. Any number of files may be added to an IDE project by merely repeating the same steps used to add one file, and the procedure for doing this is explained in detail in the appropriate "Using the Compiler's IDE…" course document. The IDE will then automatically compile and link these files together and produce a single program. What you must not do is use `#include` to include an implementation file in any other file, although header files are designed to be included this way.

"Include Guards"

Good programming practice dictates that the contents of **every header file**, but **never an implementation file**, be protected by a 3-line "include guard". This easy to use concept is discussed and illustrated in note D.2 in appendix D of the course book and a quick example is provided below for a header file named **Hi$tory2File.h**. The name of an include guard should be the name of the header file in all uppercase with any characters that are not allowed in identifiers (note 1.4) replaced by underbars. In addition, if the file name begins with a numeric character the include guard name must be preceded by an underbar.

```
#ifndef HI_TORY2FILE_H          ← 1st line of include guard
#define HI_TORY2FILE_H          ← 2nd line of include guard

…everything else in the header file…

#endif                          ← 3rd line of include guard
```

C1A4E0 *(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 **C1A4E0_Quiz.txt** formatted as:

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

1. Look up the *floor* function. If there is no prototype for it in scope but the code below compiles, what value does *floor* take the "floor" of and what value is assigned into x:
   int *x = floor(-60.28);*
   (Notes 5.2, 5.4, and 5.5)
   A. -60.28 and 0
   B. -60.28 and garbage
   C. -61 and garbage
   D. -60.28 and -61
   E. none of the above

2. If the following code compiles what value does *Divider* return?
```
long double Divider()
{
    return(Quotient(8.6F, 2.0F));
}
double Quotient(float a, float b)
{
    return(a / b);
}
```
   (Note 5.3)
   A. 0
   B. 4.3
   C. 4 (because of integer division)
   D. garbage
   E. none of the above

3. If the following C++ functions all exist and are prototyped as shown, which is called by:
   double z = Average(200, 28.**F**, -3);
   (Note 5.8)
   A. **int** Average(**float** f, **char** c, **int** i, **short** s);
   B. **long** Average(**int** i1, **float** f, **int** i2);
   C. **double** Average(**float** f);
   D. **double** Average();
   E. none

4. When used in a multi-file program, external variables and functions should be **static** if: (Notes 5.15 & 5.16)
   A. they will not be used in another file.
   B. they will not be used more than once.
   C. initialization is not required.
   D. the program is not multi-threaded.
   E. they are used by other **static** functions.

5. Which of the following function-like macros produces the negated value of its argument most appropriately?
   (Note 5.18)
   A. #define neg(x) (-x)
   B. #define neg(x) (-(x))
   C. #define neg(x) ((x) * -1)
   D. #define neg(x) (0 - x)
   E. **A** and **B** are both most appropriate

6. Assuming: *#define Quotient(n, d) n / d*
   Predict the value of: *5 + Quotient(3.5 + 1, 2)*
   (Note 5.18)
   A. 5.5
   B. 7.25
   C. 8.5
   D. 9.0
   E. implementation dependent

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A4E0_167109_U09609277** 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.

C1A4E1 *(3 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add three new ones, naming them **C1A4E1_ComputeMinimum.c**, **C1A4E1_ComputeMaximum.c**, and **C1A4E1_main.c**. 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 **C1A4E1_ComputeMinimum.c** must contain a function named `ComputeMinimum` and **C1A4E1_ComputeMaximum.c** must contain a function named `ComputeMaximum`. Each function must:

1. Return type **double** and have exactly two formal parameters, each of type **double**.
2. Contain only one statement.
3. Not use variables other than its formal parameters.
4. Not use anything that requires `#define` or `#include`.
5. Not use literal values.
6. Not do assignment, addition, subtraction, multiplication, or division.
7. Not use `if`, `switch`, or looping statements.
8. Not call functions or macros.
9. Not display anything.

`ComputeMinimum` must compare its parameter values and return the smallest value passed to it. `ComputeMaximum` must do the same for the largest value.

File **C1A4E1_main.c** must contain function `main`, which must:

1. Prompt (ask) the user to enter two space-separated decimal values on the same line.
2. Pass the user-entered values to both `ComputeMinimum` and `ComputeMaximum` as arguments.
3. Display the results of both function calls using the following 2-line format, where the question marks represent the values 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 treat equal values as a special case.
- Scientific and standard notation are both okay and may be mixed.
- Zeros that do not 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 **C1A4E1_167109_U09609277** and with all 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:

The appropriate solution for `ComputeMinimum` and `ComputeMaximum` involves the use of the "conditional" operator described in note 3.16. Simply use it to compare the values of each function's parameters and directly return the resulting value.

C1A4E2 *(4 points – C++ Program)*

The purpose of this exercise is to familiarize you with function overloading. Exclude any existing source code files that may already be in your IDE project and add five new ones, naming them **C1A4E2_PrintLines-3.cpp**, **C1A4E2_PrintLines-2.cpp**, **C1A4E2_PrintLines-1.cpp**, **C1A4E2_PrintLines-0.cpp**, and **C1A4E2_main.cpp**. Do not use `#include` to include any of these five files in each other or in any other file. However, you may use it to include any appropriate header file(s) you need.

C1A4E2_PrintLines-3.cpp must contain a function named `PrintLines` that returns **void** and has exactly three formal parameters, all of type **int**. From left-to-right those parameters represent the value of a character to be displayed, the number of times the character is to be displayed on a line, and the number of lines to be displayed. For example, `PrintLines('C', 5, 2)` would output:

CCCCC *(followed by a newline character)*
CCCCC *(followed by a newline character)*

C1A4E2_PrintLines-2.cpp must contain a function named `PrintLines` that returns **void** and has exactly two formal parameters, both of type **int**. Those parameters have the same meaning as the first two parameters in the 3-parameter version above, but only one line is displayed. For example, `PrintLines('C', 5)` would output:

CCCCC *(followed by a newline character)*

C1A4E2_PrintLines-1.cpp must contain a function named `PrintLines` that returns **void** and has exactly one formal parameter, which must be of type **int**. That parameter has the same meaning as the first parameter in the 2-parameter version above, but only one character is displayed on one line. For example, `PrintLines('C')` would output:

C *(followed by a newline character)*

C1A4E2_PrintLines-0.cpp must contain a function named `PrintLines` that returns **void** and has exactly 0 formal parameters. It displays only one 'Z' character on one line. For example, `PrintLines()` would output:

Z *(followed by a newline character)*

C1A4E2_main.cpp must contain function `main`. `main` must contain a two-iteration `for` statement that executes the loop body code indicated below once during each of its two iterations:

```
for (…code to cause two iterations…)
{
```

1. Prompt (ask) the user to enter the 3 items below in order and space-separated on the same line. Store them in variables named `charValue`, `charCount`, and `lineCount`, respectively:
   a. the character to display (do not put quotes around the character).
   b. the number of times to display the character on each line.
   c. the number of lines to display.
2. Make the 4 function calls below in order:
   a. `PrintLines(charValue, charCount, lineCount);`
   b. `PrintLines(charValue, charCount);`
   c. `PrintLines(charValue);`
   d. `PrintLines();`

loop body

```
}
```

- Do not call any version of `PrintLines` recursively or from another version of `PrintLines`.
- Do not include header file `<string>` or use anything from the C++ `string` class in any of your files.
- Do not add a blank line to separate the output from each function.
- Test your program with at least the following 5 sets of input values:
  U 20 10 **V 0 10** **W 25 0** **X 25 1** **Y 150 3**

Some boundary value results are on the next page. No special code is necessary to handle them:

`PrintLines('C', 0, 2)`	outputs two newline characters only
`PrintLines('C', 5, 0)`	outputs nothing, not even a newline character
`PrintLines('C', 0, 0)`	outputs nothing, not even a newline character
`PrintLines('C', 0)`	outputs one newline character only

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A4E2_167109_U09609277** 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. Function overloading is illustrated in note 5.8.
2. Students are sometimes confused about using a type **char** variable to obtain the user character and then passing it as an argument to the `PrintLines` functions whose first parameter is type **int**. However, this is not a problem. When a function is called in the presence of its prototype, all arguments that are not already the function's corresponding parameter types are automatically converted to those types (Note 5.5). Thus, a type **char** argument passed to `PrintLines` will be automatically converted to type **int**. No cast is necessary to avoid a warning because type **int** can represent any value (and more) than type **char** can represent.
3. Always declare functions to return type **void** unless returning a value would serve a meaningful purpose.

Example 1: Expected output for an input of `% 25 1`

```
%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%
%
Z
```

Example 2: Expected output for an input of `J 20 10`

```
JJJJJJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJJJJJ
J
Z
```

Example 3: Expected output for an input of `@ 25 0`

```
@@@@@@@@@@@@@@@@@@@@@@@@@
@
Z
```

C1A4E3 *(3 points – C++ Program)*

The purpose of this exercise is to familiarize you with default function arguments. Exclude any existing source code files that may already be in your IDE project and add two new ones, naming them **C1A4E3_PrintLines.cpp** and **C1A4E3_main.cpp**. Do not use `#include` to include either of these files in each other or in any other file. However, you may use it to include any appropriate header file(s) you need.

Required Steps – Exercise 2 Must Be Complete and Cleanly Passing the Assignment Checker Before Proceeding

1. You should have just added files **C1A4E3_PrintLines.cpp** and **C1A4E3_main.cpp** to your exercise 3 IDE project. Replace anything they might contain with the entire contents of exercise 2 files **C1A4E2_PrintLines-3.cpp** and **C1A4E2_main.cpp**, respectively.
2. In file **C1A4E3_PrintLines.cpp** do **only** the following - **Credit will be deducted** for other changes:
   a. Change the exercise number, file name, and date in the title block. If anything else needs to be changed it's because the exercise 2 version of this file was written improperly.
3. In file **C1A4E3_main.cpp** do **only** the following - **Credit will be deducted** for other changes:
   a. Change any comment(s) affected by the different requirements of exercise 3.
   b. Delete the prototypes for the 0, 1, and 2-parameter versions of `PrintLines`.
   c. Modify the prototype for the 3-parameter version of `PrintLines` so the entire program will produce **identically the same results as Exercise 2**.
   d. If anything else needs to be changed it is because the exercise 2 version of this file was written improperly.

Test your program the same way you tested Exercise 2. The results should be identical in every way.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A4E3_167109_U09609277** 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. Default function arguments are illustrated in note 5.7. See the bottom of note 5.8 for an illustration of converting two overloaded functions into a single default argument function. Standard good practice dictates that you not put default argument values in the declaration part of a function definition except in very rare cases, which this exercise is not. Default argument values are used only if actual arguments are not provided.

2. If you're having trouble determining the default argument values consider required behavior of the `PrintLines` functions having less than three parameters in the previous exercise. What values are they using in place of those omitted parameters?

C1A4E4 *(4 points – C++ Program)*

Exclude any existing source code files that may already be in your IDE project and add two new ones, naming them **C1A4E4_MaxOf.h** and **C1A4E4_main.cpp**. You may not include (`#include`) **C1A4E4_main.cpp** in any other file but you must include **C1A4E4_MaxOf.h** in any file that needs its contents.

File **C1A4E4_MaxOf.h** must contain a 2-parameter macro named `mMaxOf2`, a 3-parameter macro named `mMaxOf3`, a 2-parameter "inline" function named `fMaxOf2`, and 3-parameter "inline" function named `fMaxOf3`, as follows:

`mMaxOf2`, `mMaxOf3`, `fMaxOf2`, and `fMaxOf3` must:

1. return the maximum value passed to them.
2. support any arithmetic values within the range and precision of type **long double**.
3. not use variables other than their formal parameters.
4. not need `#define` or `#include`, except for the `#define`s used to define `mMaxOf2` and `mMaxOf3`.
5. not use literal values.
6. not use assignment, addition, subtraction, multiplication, or division.
7. not use `if`, `switch`, or looping statements.
8. not display anything.

`mMaxOf3` and `fMaxOf3` must:

1. not use the conditional operator (**?:**) or any relational/equality operators (**<**, **>**, ==, etc.)

`mMaxOf3` must:

1. do any needed comparisons using only `mMaxOf2`, calling it no more than twice.

`fMaxOf3` must:

1. do any needed comparisons using only `fMaxOf2`, calling it no more than twice.

File **C1A4E4_main.cpp** must contain function `main`, which must:

1. prompt (ask) the user to enter three space-separated decimal values on the same line.
2. pass the user-entered values to both `mMaxOf3` and `fMaxOf3` as arguments.
3. display the results of both calls using the following 2-line format, where the question marks represent the values passed to and returned from the macro and function:

   `mMaxOf3(?, ?, ?) returned ?`
   `fMaxOf3(?, ?, ?) returned ?`

   For example, if the user enters **-3.8 -3.5 -3.2** the result should be:

   `mMaxOf3(-3.8, -3.5, -3.2) returned -3.2`
   `fMaxOf3(-3.8, -3.5, -3.2) returned -3.2`

- Do not define any functions or macros other than `main`, `mMaxOf2`, `mMaxOf3`, `fMaxOf2`, and `fMaxOf3`.
- 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 4 sets of user input values, where each set represents the argument values in left-to-right order:

-3.8 -3.5 -3.2 -3.2 -3.5 -3.8 -3.5 -3.8 -3.2 8.4e3 6.2e-1 .02e2

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A4E4_167109_U09609277** and with both 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:

See note 5.18 for an example of code that is similar to what is expected for a typical `mMaxOf2` macro and note 5.19 for the inline function version (but both of these examples compute the minimum rather than the maximum). A macro replacement list containing more than one token must be placed in parentheses. In addition, parentheses must be placed around every argument usage in the replacement list, even if that argument is passed to another macro whose arguments are already properly parenthesized. However, all of this parenthesizing is neither necessary nor desirable in equivalent inline functions. Never create prototypes for macros. Be sure to use "include guards" (note D.2) in header file **C1A4E4_MaxOf.h**. Use `#include` to include this file in file **C1A4E4_main.cpp**.