# Assignment 4
C/C++ Programming I

## Exercise 0 (6 points – 1 point per question – No program required)

Language standards compliance and appropriate header file inclusion is assumed. Testing code by running it is sometimes misleading due to implementation dependence. These are not trick questions and there is only one correct answer to each. Applicable notes from the course book are listed.

1. C only – Look up the *sqrt* function. Assuming there is no prototype for it in scope, what is the value *sqrt* takes the square root of and what is the output from: *printf("%f", sqrt(25));* (Notes 5.4 & 5.5)
   A. 25.000000 and 5.000000
   B. 25 and 5
   C. 0 and 0
   D. 0.000000 and 0.000000
   E. garbage and garbage

2. In C, which statement is true concerning a major problem with the following if there is no prototype for function *Sum* prior to its call?

```
long double TestIt()
{
    double answer = Sum(1.1, 2.2, 3.3);

    printf("answer = %f", answer);
    return(answer);
}
double Sum(double a, double b, double c)
{
    return(a + b + c);
}
```

   (Note 5.4)
   A. The name *Sum* conflicts with a standard library math function.
   B. The **return** statement in *TestIt* returns type **double**.
   C. Return statements may not contain an algebraic expression (a + b + c).
   D. There is nothing wrong with the program.
   E. The call to *Sum* assumes that *Sum* returns type **int**.

3. If all of the following functions exist in a C++ program and are prototyped first, which does *TestIt('A', **int**('A'))* call? (Note 5.8)
   A. **int** TestIt(**char** ch, **char** ix);
   B. **int** TestIt(**char** ch, **short** ix);
   C. **int** TestIt(**char** ch = 'A');
   D. implementation dependent
   E. none

4. When used in a multi-file program, external variables/functions should be declared **static** if they: (Notes 5.15 & 5.16)
   A. are/return **void**.
   B. are in C++ programs.
   C. are referenced/called from another file.
   D. are not referenced/called from another file.
   E. are referenced/called from other functions.

5. The correct implementation of a function-like macro that produces the absolute value of its argument is: (Note 5.18)
   A. #define abs(x) x < 0 ? -x : x
   B. #define abs(x) (x < 0 ? -x : x)
   C. #define abs(x) (x) < 0 ? (-x) : (x)
   D. #define abs(x) ((x) < 0 ? (-x) : (x))
   E. #define abs(x) ((x) < 0 ? -(x) : (x))

6. Assuming *#define Sum(a, b) a + b* predict the value of: *5 * Sum(3 + 1, 2)*: (Note 5.18)
   A. 30
   B. 18
   C. 22
   D. none of the above
   E. implementation dependent

### Submitting your solution

Using the format below place your answers in a plain text file named **C1A4E0_Quiz.txt** and send it to the Assignment Checker with the subject line **C1A4E0_*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.*

## About This Assignment

### 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 an "include guard". This easy to use concept is discussed and illustrated in note D.2 in appendix D of the course book.

## Exercise 1 *(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. have exactly two formal parameters, each of type **double**;
2. return type **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 of its parameters and return the smallest of those values whereas `ComputeMaximum` must compare the values of its parameters and return the greatest of those values.

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

1. prompt the user to enter any space-separated pair of decimal numeric 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 ?
```

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 **C1A4E1_*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:**
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.

## Exercise 2 (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 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 display:

```
CCCCC
CCCCC
```

**C1A4E2_PrintLines-2.cpp** must contain a function named `PrintLines` that 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 display:

```
CCCCC
```

**C1A4E2_PrintLines-1.cpp** must contain a function named `PrintLines` that 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 display:

```
C
```

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

```
Z
```

**C1A4E2_main.cpp** must contain function `main`, which must execute a 5-iteration "for" loop that does the following during each iteration:

1. Prompts the user to enter all 3 of the following in order and space-separated on the same line:
   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. Makes the following 4 function calls, in order, passing the appropriate user entries as arguments:
   a. `PrintLines(`*characterToDisplay*`, `*numberOfCharacters*`, `*numberOfLines*`);`
   b. `PrintLines(`*characterToDisplay*`, `*numberOfCharacters*`);`
   c. `PrintLines(`*characterToDisplay*`);`
   d. `PrintLines();`

- Do not include header file `<string>` or use anything from the C++ `string` class in any of your files.
- 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 conditions students often ask about along with the required results from your program are as follows:

| | |
|---|---|
| `PrintLines('C', 0, 2)` | displays 0 characters on each of 2 lines (two blank lines) |
| `PrintLines('C', 5, 0)` | displays 5 characters on 0 lines (nothing is printed; no lines are used) |
| `PrintLines('C', 0, 0)` | displays 0 characters on 0 lines (nothing is printed; no lines are used) |
| `PrintLines('C', 0)` | displays 0 characters on 1 line (one blank line) |

## Submitting your solution

Send your five source code files to the Assignment Checker with the subject line **C1A4E2_*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:**
Function overloading is illustrated in note 5.8. However, one unrelated question students often have regarding this exercise pertains to the fact that they use a type **char** variable to obtain the desired character from the user, but the `PrintLines` functions use a type **int** parameter to represent this character. Note that when called in the presence of a function prototype all compatible arguments are converted to the type of the corresponding function parameter (Note 5.5). This means that even if you use a type **char** variable to obtain the desired character from the user it will automatically get converted to type **int** if passed to a function having a type **int** parameter. However, to avoid a possible compiler warning use a type cast when passing such an argument. Also, always declare functions to return type **void** unless returning a value would serve a meaningful purpose.

## Exercise 3 *(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 Procedure – Exercise 2 Must Be Correct First**

Copy the entire contents of files **C1A4E2_PrintLines-3.cpp** and **C1A4E2_main.cpp** from Exercise 2 into files **C1A4E3_PrintLines.cpp** and **C1A4E3_main.cpp**, respectively, of this exercise and then:

1. In file **C1A4E3_PrintLines.cpp** do **only** the following:
   a. Change any comment(s) affected by the exercise number and file name changes.
2. In file **C1A4E3_main.cpp** do **only** the following:
   a. Change any comment(s) affected by the exercise number and file name changes;
   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**.
3. If you make any changes other than those described above you are doing this exercise incorrectly.

Test your program the same way as in Exercise 2.

### Submitting your solution

Send your two source code files to the Assignment Checker with the subject line **C1A4E3_*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:**

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.

## Exercise 4 (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 of their parameter values;
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 the user to enter any three space-separated decimal numeric 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 ?
```

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

Do not define any functions or macros other than `main`, `mMaxOf2`, `mMaxOf3`, `fMaxOf2`, and `fMaxOf3`.

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 your two source code files to the Assignment Checker with the subject line **C1A4E4_*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:**
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**.

## Get a Consolidated Assignment 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 **C1A4_*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.