Assignment 1
C/C++ Programming I

C1A1 General Information

Course Assignment/Exercise Notation Conventions: Each weekly "assignment" consists of several "exercises". Throughout this course I commonly refer to these using an abbreviated notation, where a form like **C1A2E3** would refer to exercise 3 in assignment 2 of the "C/C++ Programming I" course and **C1A2** would refer to the entirety of assignment 2 of that course.

Getting Started: Before starting your first assignment you must have the appropriate tools for developing your software and the best way to get them is to download and install one of the many free integrated development environment (IDE) packages. These integrate the compiler, editor, and other necessary tools into a convenient GUI application. Although you are free to use any tools you wish and any operating system that will support them, I recommend Microsoft's "Visual Studio Community" for Windows, "Xcode" for macOS, and "Code::Blocks" for Linux. Information on obtaining, installing, and using them is available in the appropriate version of the course document titled "Using the Compiler's IDE...", a link to which is located on the "Assignments" page of the course website. I am sorry but I do not have information on other IDE's or operating systems.

Source Code Files: Header Files and Implementation Files: "Source code" files contain the code necessary for a program to be built without errors and are divided into the two categories "header" files (`.h`, etc.) and "implementation" files (`.c`, `.cpp`, etc.). Not all programs require header files but at least one implementation file is always required. Header files are designed to be included in other files using the `#include` directive but implementation files are not. By placing items that might be needed by multiple other files in header files and including them in those files the bad practice of literally duplicating the needed items in each file can be avoided. Because of their multiple usages, however, header files must never contain anything that will result in an error if more than one file includes them. Files containing data that your program reads or writes are not considered source code files but are instead "data files".

Although some of the following terminology has not yet been discussed in this course it is placed here for completeness and for future reference: Header files typically contain things like macro definitions, inline function definitions, function prototypes, referencing declarations, typedefs, class/structure/union descriptions, and templates, although any of these that will only ever be needed by one specific implementation file may be placed in that file instead. Header files must not contain non-inline function definitions or defining variable declarations; these must be placed in implementation files instead. The header files that are supplied with a compiler provide the information it needs to properly compile code that uses the various functions, macros, and data types available in the compiler's libraries.

Exercise Submission Procedure: Get an exercise to work first on your computer, then submit it to the "assignment checker" and wait for the results to be returned. If there are any errors or warnings make the appropriate corrections and resubmit, repeating as necessary until all issues are corrected. Additional details are provided in each exercise and the course document titled "How to Prepare and Submit Assignments".

Get a Consolidated Assignment 1 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 **C1A1_162461_U09339356** 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.

C1A1E0 *(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 **C1A1E0_Quiz.txt** formatted as:

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

1. In a situation where each of
   xyz = xyz + 1
   xyz += 1
   xyz++
   ++xyz
   xyz = xyz++
   would produce the wanted value, which should be chosen?
   (Note 1.7)
   A. xyz = xyz + 1
   B. xyz += 1
   C. xyz++
   D. ++xyz
   E. xyz = ++xyz

2. Which of the following is not a string literal?
   (Note 1.5)
   A. "%\\"
   B. "%*n*\"
   C. 5 single quotes between 2 double quotes
   D. "$/\\"
   E. "\"\x72\" "

3. Assuming the ASCII character set, predict the output from:
   cout << "\x4a\146\155\153"
   "\x70\x0021";
   (Notes 1.5 and B.1)
   A. nothing – It will not compile.
   B. \x4a\146\155\153
   C. Ifmmp!
   D. Jfmm
   E. Jfmkp!

4. What data types are acceptable for x in the expression *printf("%lli", x)* (The *ll* is two lowercase ells, not two ones.)
   (Note 1.11)
   A. **char**, **short**, **int**, **long**, and **long long** only
   B. **char**, **short**, **int**, and **long** only
   C. **long double** only
   D. **long long double** only
   E. **long long int** only

5. Of the arithmetic data types **char**, **short**, **int**, **long**, **long long**, **float**, **double**, and **long double**, which are acceptable for x in the expression *cin* >> x
   (Note 1.14)
   A. all except **char** and **long double**
   B. all except **char**
   C. all
   D. **char**, **short**, **int**, and **double** only
   E. **char**, **short**, and **int** only

6. If a user presses the spacebar 10 times before typing ***357*** and if variables *a*, *b*, *c*, *d*, and *e* are of an appropriate type, which of the following expressions will not skip all those spaces before attempting to store something into the variable?
   (Notes 1.15 and 1.16)
   A. cin >> a
   B. scanf("\n%c", &b)
   C. scanf("%c", &c)
   D. scanf("%d", &d)
   E. scanf("%Lg", &e)

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A1E0_162461_U09339356** 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.

C1A1E1 *(7 points – C++ Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one, naming it **C1A1E1_main.cpp**. Write your program in that file.

Note 1.7 of the course book illustrates how some expressions can be written more compactly, where the most compact form is defined as the form containing the fewest non-whitespace characters. For example, `x = x + y` should always instead be written as `x += y`. The most compact form is always the most appropriate form and is the only form allowed in this course.

Note 2.7 illustrates the most appropriate way to negate (change the sign of) the value of a variable. For example, unconditionally negating the current value of `x` should always be written as `x = -x`, never in any other way. It is the only form allowed in this course.

Write a program that displays the most appropriate form of each of the following 14 expressions with respect to the updated value of integer variable `ax`. Do not attempt to actually evaluate the expressions. Of these expressions:

- Two are already in their most appropriate form.
- Two have two most appropriate forms.
- Four only involve the negation of the value of `ax`.

```
ax = ax + bx
ax = ax / -bx
ax = bx / ax
ax = -1 * ax
ax = -ax * ax
ax = -bx * ax
ax = bx - ax
ax = 2 + ax
ax = 1 + ax
ax = ax - 37
ax = ax - 1
ax *= -1
ax /= -1
ax = 0 - ax
```

Display each expression on a separate line along with its most appropriate form with double-quotes around each. Be sure to display the words **should be** after the original expression and the word **or** between the most appropriate forms if there is more than one. Here is an example of the required output format for three hypothetical expressions:

```
"abc = def - ghi" should be "abc = def - ghi"
"abc = abc * def" should be "abc *= def"
"abc += 1" should be "++abc" or "abc++"
```

- The code in your `main` function must start with `cout <<` or `std::cout <<` on a line by itself.
- Do not use `cout` or the **<<** operator more than once.
- Do not declare any variables.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A1E1_162461_U09339356** 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:
See notes 1.5, 1.7, and 2.7.

C1A1E2 (*7 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one, naming it **C1A1E2_main.c**. Write your program in that file.

Many different units are used to represent distances, with feet, meters, miles, nautical miles, cubits, fathoms, and furlongs being among them. Conversion between these units is often necessary and is usually straightforward. For example, to convert from feet to:

meters:	multiply by `0.3048`
miles:	multiply by `0.000189394`
nautical miles:	divide by `6076`
cubits:	multiply by `0.666667`
fathoms:	divide by `6`
furlongs:	divide by `660`

Using the exact conversions provided above write a program that prompts (asks) the user to enter a decimal value representing an arbitrary number of feet then displays the equivalent in meters, miles, nautical miles, cubits, fathoms, and furlongs. Use the exact display format illustrated below, where a user input value of `27.38` is being used as an example:

```
27.38 feet
= 8.34542 meters
= 0.00518561 miles
= 0.00450625 nautical miles
= 18.2533 cubits
= 4.56333 fathoms
= 0.0414848 furlongs
```

Your program must:
1. use type `double` for all variables.
2. use `printf`'s `%g` conversion specifier to display all numeric values.
3. not call any functions other than `printf` and `scanf`.
4. not call `printf` more than twice.
5. not call `scanf` more than once.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A1E2_162461_U09339356** 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:
The compiler automatically concatenates multiple string literals separated only by zero or more whitespaces into one string, including string literals on separate lines.