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. What is output: *printf("%d\n", !(6/3+2.2) + 3);*
   (Note 3.2)
   A. *3*
   B. *7.2*
   C. The output is implementation dependent.
   D. *7*
   E. garbage because the expression is type **double** but *%d* specifies type **int**

2. Predict the output from:

```
if (5 < 4)
    if (6 > 5)
        cout.put('1');
else if (4 > 3)
    cout.put('2');
else
    cout.put('3');
cout.put('4');
```

   (Note 3.15)
   A. *4*
   B. *2*
   C. *24*
   D. *4* or *24* depending upon implementation
   E. Nothing is printed.

3. Predict the output from:

```
switch (2 * 2)
{
    case 1+1:  cout << "value = 2 ";
    case 29:  cout << "value = 29 ";
    case 4:  cout << "value = 4 ";
    default:  cout << "illegal switch value ";
    case 'A':  cout << "Got an 'A' ";
}
```

   (Notes 3.17 & 3.18)
   A. *value = 4 illegal switch value Got an 'A'*
   B. *value = 4*
   C. *illegal switch value*
   D. *Got an 'A'*
   E. The output is implementation dependent.

4. What gets printed by:
   putchar('A') + putchar('B');
   (Note 3.3)
   A. *AB* only
   B. *BA* only
   C. either *AB* or *BA*
   D. either *A* or *B* but not both
   E. *A* only

5. What gets printed by:
   putchar('A') && putchar('B');
   (Note 3.3)
   A. *AB* only
   B. *BA* only
   C. either *AB* or *BA*
   D. either *A* or *B* but not both
   E. *A* only

6. For **int** x = 5; what is the value and data type of the entire expression on the next line:
   sqrt(9.0), ++x, printf("123"), 25
   (Note 3.11)
   A. 3.0 (type **double**)
   B. 3 (type **int**)
   C. 25 (type **int**)
   D. 6 (type **double**)
   E. implementation dependent

Submitting your solution

Using the format below place your answers in a plain text file named **C1A3E0_Quiz.txt** and send it to the Assignment Checker with the subject line **C1A3E0_*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.

Exercise 1 *(3 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one, naming it **C1A3E1_main.c**. Write a program in that file to compute and display a table of factorials. The factorial of an integral value *n*, written *n!*, is the product of the consecutive integers *n* down through *0*. Since *0!* is defined as having a value of *1* it can be omitted from the calculations since it doesn't affect the outcome. For example, *5* factorial (*5!*) can be calculated as *5 x 4 x 3 x 2 x 1* and has a value of *120*. Here is a table of *1!* Through *7!*:

```
nbr          nbr!
---------------
 1             1
 2             2
 3             6
 4            24
 5           120
 6           720
 7          5040
```

IMPORTANT:
Your code must use a type **long** (not **unsigned long**) variable to represent the value of factorial **nbr!** in the table above. The results must be correct up to the maximum value type **long** can represent on any and every machine on which your unaltered code is compiled and run. Since compiler manufacturers are allowed to make that maximum as great as they see fit as long as it is at least **2,147,483,647**, it could conceivably be so great that hundreds of digits would be required to represent it.

In addition to the above requirement, your program must:

1. prompt the user to enter an integer value greater than 0 and store it in a type **int** variable;
2. compute and display a table like the one illustrated above for all values *1!* through the factorial of the value entered by the user. Values must be displayed as decimal integers – no exponents or decimal points.
3. align the least significant digits in both columns for all entries in the table. Do not attempt to write code to compute the field widths needed for these columns. Instead, a fixed width of 2 for the 1st column and 11 for the 2nd is fine for the factorial values tested in this exercise unless you start getting misalignments or simply want to make them wider. Separate the fields with at least one space so the numbers won't run together.
4. not use floating point literals, floating point variables, floating point functions, or floating point type casts;
5. not use arrays or recursion; recursion occurs when a function is called before a previous call to that function has returned, for example, when a function calls itself;
6. not use more than 1 looping statement.

Manually re-run your program several times, testing with at least the following 3 input values:
1 25 36
If you find that any of the factorial values are incorrect determine if the expected values exceed the maximum value supported by type **long** on your machine, in which case they should be incorrect. Even if they are all correct they will eventually exceed the maximum if the user input value is increased sufficiently. Suggest a possible way the program could be improved to extend its range, but don't incorporate your suggestion into the code you will be submitting for grading. Instead, merely place your suggestion as a comment in the file's "Title Block".

Submitting your solution

Send your source code file to the Assignment Checker with the subject line **C1A3E1_*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 1 type **int** variable to get the user input value, another to represent the current factorial multiplier (1, 2, 3, 4, 5, etc.), and a type **long** variable to represent the result of all previous multiplications (the factorial result). Then implement the following algorithm, which is completely independent of the number of digits required to represent the maximum value type **long** can represent:

1. Get the user input value.
2. Initialize both the factorial multiplier and the factorial result to 1.
3. **IF** the factorial multiplier is less than or equal to the user input value:
    a. Multiply factorial result by the factorial multiplier; the product becomes the new factorial result.
    b. Display the factorial multiplier and the factorial result.
    c. Increment the factorial multiplier.
    d. Repeat from step 3.

Exercise 2 *(5 points – C++ Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one, naming it **C1A3E2_main.cpp**. Write a program in that file to reverse the digits of an arbitrary user-entered integer value based solely upon its numeric value, not the individual characters entered. If the value is negative the minus sign must be displayed last. Here are some sample input values and the expected reversals:

Input	Reversal
3987	**7893**
-2645	**5462-**
100	**001**
000120	**021**
-0023	**32-**
000	**0**

Your program must:

1. prompt the user to enter any integer value;
2. use `cin >>` to read the entire value at once into a type **int** variable;
3. display the variable's value and the reversed value in the format below, placing double-quotes around both for readability. For example if the user input is `-00000000000000000000000026450` the following would get displayed:
   `"-26450" in reverse is "05462-"`
4. not use any non-const variables that are not type **int** or type **bool**;
5. not use anything involving floating point types (the `pow` function, `math.h`, type **double**, etc.);
6. not use arrays or recursion; recursion occurs when a function is called before a previous call to that function has returned, for example, when a function calls itself.

Manually re-run your program several times, testing with at least the following 6 input values:
3 -123 0 1010 -1010 -0007000

Submitting your solution

Send your source code file to the Assignment Checker with the subject line **C1A3E2_*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 algorithm below picks off and displays the digits of the input value one-at-a-time moving right-to-left. Since it uses integer division (both standard and modulo), which is not portable if either operand is negative (Note 2.8), you must first test the user input. If it is negative change it to positive and use a variable to remember that it was originally negative. Then implement the following algorithm, in which the positive form is referred to as the "number":

1. Modulo-divide the number by 10, thereby producing the least significant digit (LSD), which is then displayed.
2. Divide the number by 10; the quotient becomes the new number (which is the previous number with its LSD removed).
3. **IF** the new number is not equal to 0 repeat from step 1.
4. Display a minus sign if the original user input value was negative.

Exercise 3 *(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 **C1A3E3_main.cpp**. Write a program in that file to convert an arbitrary user-entered integer value into words based solely upon its numeric value, not the individual characters entered. If the value is negative the word `minus` must be first. Here are some sample input values and the expected words:

Input	Words
00000	**zero**
593	**five nine three**
-593	**minus five nine three**
-500	**minus five zero zero**
-000500	**minus five zero zero**

Your program must:

1. prompt the user to enter any integer value;
2. use `cin >>` to read the entire value at once into a type **int** variable;
3. display the variable's value and the equivalent words in the format below, placing double-quotes around both for readability. For example if the user input is `-00000000000000000000026450` the following would get displayed:
   `"-26450" in words is "minus two six four five zero"`
4. not use any non-const variables that are not type **int**;
5. not use anything involving floating point types (the `pow` function, `math.h`, type **double**, etc.);
6. not use arrays or recursion; recursion occurs when a function is called before a previous call to that same function has returned, for example, when a function calls itself.
7. not use any nested loops – they are totally unnecessary.

Manually re-run your program several times, testing with at least the following 6 input values:
3 -123 0 1010 -1010 -0007000

Submitting your solution

Send your source code file to the Assignment Checker with the subject line **C1A3E3_*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:
Detailed hints are on the next page…

Hints (for Exercise 3):

The optional algorithm below displays a user decimal integer input value in words, one-at-a-time moving left-to-right. Since it uses integer division (both standard and modulo), which is not portable if either operand is negative (Note 2.8), the input value is tested and made positive if necessary. There are no nested loops, part A is completed before part B begins, and part B is completed before part C begins. Only one instance of the code for each part is necessary:

Part A:

A1. Prompt the user, get his/her input, and output the display message up to the point where the first word of the value is needed.

A2. If user input number is negative change it to positive and display the word "minus", followed by a space.

Part B ("for" loop is used):

Find a power of 10 divisor that will produce the most significant digit (MSD) of the positive number as follows:

B1. Set a divisor variable to 1 and a dividend variable to a copy of the number.

B2. If the value of the dividend is greater than 9:

a. Multiply the divisor by 10; the product becomes the new divisor.

b. Divide the dividend by 10; the quotient becomes the new dividend.

c. Repeat from step B2.

Part C ("do" loop is used):

Pick off the digits of the positive number left-to-right and display them as words as follows:

C1. Divide the number by the divisor, which yields the MSD. Display it as a word using a 10-case switch statement (see below).

C2. Multiply the MSD found in the previous step by the divisor and subtract the result from the number; the difference becomes the new number, which is merely the previous number with its MSD removed.

C3. Divide the divisor by 10; the result becomes the new divisor.

C4. If the new divisor is not equal to 0, repeat from step C1.

About the recommended "switch" statement…

While the use of "magic numbers" is usually a bad idea, in some situations they are appropriate such as for the "cases" used in the "switch statement" recommended for this exercise. Specifically, each case represents a unique numeric value ranging from 0 through 9. There is no underlying meaning to these values other than the values themselves, their purpose is obvious and unmistakable, there is no possibility that they might ever need to be changed, and there is no identifier (name) that would make their meaning any clearer. Thus, the literal values should be specified directly, as follows:

```
switch (…)
{
   case 0:   …
   case 1:   …
   case 2:   …
   etc.
}
```

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 **C1A3_*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.