Assignment 3
C/C++ Programming I

C1A3 General Information

Assignment 3 consists of FOUR (4) exercises:

C1A3E0 C1A3E1 C1A3E2 C1A3E3

All requirements are in this document.

Related examples are in a separate file.

Get a Consolidated Assignment 3 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 **C1A3_174273_U09845800** 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.

C1A3 General Information, continued

You Should Remember…

Decimal, hexadecimal, octal, and binary integer literals

When a positive integer literal is coded, such as in `x = 232`, it may be written in decimal, hexadecimal, octal, or in C++ only, in binary. The choice of which radix to use is strictly a matter of convention, but regardless of which is chosen the value is stored in binary internally. For example, the decimal, hexadecimal, octal, and binary literals `232`, `0xe8`, `0350`, and `0b11101000`, respectively, each represent exactly the same value and are stored internally as the binary pattern `11101000`.

Outputting/Inputting integer values in decimal, hexadecimal, or octal in C++

By default, an integer value written using `cout <<` is displayed in decimal. Similarly, an integer value read using `cin >>` is interpreted as decimal. However, the radix can be changed between decimal, hexadecimal, and octal as desired using the **dec**, **hex**, and **oct** *manipulators*. Their appropriate placement in `cout <<` and `cin >>` expressions changes the integer radix everywhere in the program, including in other functions and files. There is no manipulator for binary.

The effect of the **dec**, **hex**, and **oct** manipulators is "sticky", meaning that once set it remains in effect until explicitly changed. Because of this, specifying the same radix more than once for the same data stream or specifying it in a loop is an unnecessary waste of resources unless the radix must be changed in between.

A manipulator may be used alone in a `cout <<` or `cin >>` expression, or as part of such an expression that does other things. Some examples are:

Output

```
int x = 232;                              Stored value is binary 11101000

cout << oct;                              Change the output radix to octal.
cout << "Value is" << x;                  Output 232 as octal. Output is: 350
OR
cout << oct << "Value is" << x;           Do both of the above in one statement.
OR
cout << "Value is" << oct << x;           Same effect as the previous statement.

Output 232 as hexadecimal, octal, and decimal. Output is: e8 350 232
cout << hex << x << ' ' << oct << x << ' ' << dec << x;
```

Input

```
int x, a, b, c;

cin >> hex;                   Change the input radix to hexadecimal.
cin >> x;                     Input e8 as hexadecimal. Stored value is binary: 11101000
OR
cin >> hex >> x;              Do both of the above in one statement.

Input e8 350 232 as hexadecimal, octal, and decimal. All stored values are binary: 11101000
cin >> hex >> a >> oct >> b >> dec >> c;
```


C1A3E0 *(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. Basing an answer on actual testing is risky. Place your answers in a plain text "quiz file" named **C1A3E0_Quiz.txt** formatted as:

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

1. What is output: *printf("%d\n", !(6/3+2.2) + 3);*
(Note 3.2)
   A. *3*
   B. *7.2*
   C. It will not compile.
   D. *7*
   E. garbage because ***!(6/3+2.2) + 3*** 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

`Send an empty-body email to the assignment checker with the subject line **C1A3E0_174273_U09845800** 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.

C1A3E1 *(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 cubic sums. For the sake of this exercise, I have defined a "cubic sum" as the sum of the cubes of all numbers from *0* through some arbitrary value >= *0*. For example, the cubic sum for the number *5* can be calculated as $0^3 + 1^3 + 2^3 + 3^3 + 4^3 + 5^3$ and has a value of *225*. Below is a table of cubic sums for *0* through *5*.

```
nbr    cubic sum
---------------
 0            0
 1            1
 2            9
 3           36
 4          100
 5          225
```

IMPORTANT:
Your code must use a type **short** (not **unsigned short**) variable to represent the value of the **cubic sum** in the table above, but use type **int** variables for everything else. The results must be correct up to the maximum value type **short** 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 **32767**, it could conceivably be so great that hundreds of digits would be required to represent it.

In addition to the above requirements, your program must:

1. prompt the user to enter an integer value >= 0 and store it in a type **int** variable.
2. compute and display a table like the one illustrated above for all values from *0* through the value entered by the user. Values must be displayed as decimal integers with 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 3 for the first column and 10 for the second is fine for the 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 will not run together.
4. use a row of hyphens to separate the column titles from the values.
5. not use floating point literals, floating point variables, floating point functions, or floating point type casts.
6. not use an **if** statement or more than 1 looping statement.
7. 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 3 input values:

1 25 36

If you find that any of the cubic sum values are incorrect determine if the expected values exceed the maximum value supported by type **short** 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, without restricting user input or the number of values output, the program could be modified so that all values would be correct, but do not 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". Note that even using type `unsigned long long` or type `long double` will not eliminate eventual erroneous values.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A3E1_174273_U09845800** 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:

Use 1 type **int** variable to get the user input value, another to represent the value to be cubed (1, 2, 3, 4, 5, etc.), and a type **short** variable to represent the sum of all previous cubes (the cubic sum). Then implement the following algorithm, which is completely independent of the number of digits required to represent the maximum value type **short** can represent. Testing whether an expression is true or false can be done in several different ways and the words **If** and **Else** do not necessarily refer to an actual "if" or "if-else" statement.

1. Get the user input value.
2. Initialize both the value to be cubed and the cubic sum to 0.
3. If the value to be cubed is less than or equal to the user input value:
    a. Calculate the cube and add it to the cubic sum.
    b. Display the value that was cubed and the cubic sum.
    c. Increment the value to be cubed.
    d. Repeat from step 3.

    Else, you are done!

C1A3E2 *(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 decimal 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:

- prompt the user to enter any decimal integer value and use `cin >>` to read the entire value at once into a type **int** variable.
- use `cout <<` as often as necessary to 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-"`
- not use arrays or define custom functions.
- not declare variables or use casts that are not type **int** or type **bool**.
- not use anything involving floating point types (the `pow` function, `math.h`, type **double**, etc.).
- not use any separate code to handle a user input of zero.

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 an empty-body email to the assignment checker with the subject line **C1A3E2_174273_U09845800** 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 recommended, but not required, algorithm below uses a "do" loop to pick off and display the digits of the user input value one at a time moving right-to-left. A special case to handle zero is unnecessary. Testing whether an expression is true or false can be done in several different ways and the words **If** and **Else** do not necessarily refer to an actual "if" or "if-else" statement.

1. Prompt the user for input then read it into a variable named **inValue**.
2. Display the required output message up to where the reversed value should start.
3. Use a Boolean variable to remember if the input value was positive or negative.
4. If the input value was negative, make **inValue** positive.
5. Modulo-divide **inValue** by 10 to produce its least significant digit (LSD), then display that LSD.
6. Divide **inValue** by 10 to remove its LSD and assign the result back into **inValue**.
7. If **inValue** is not equal to 0, repeat from step 5.
8. Else, if the original user input value was negative, display a minus sign.
9. Finish the display.
10. You are done!

C1A3E3 *(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 octal 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. **If you are not familiar with the octal number system, see note C.1.** Here are some sample input values and the expected words:

Input	Words
00000	**zero**
573	**five seven three**
-573	**minus five seven three**
-500	**minus five zero zero**
-000500	**minus five zero zero**

Your program **must** interpret all integer input and display all integer output as octal (notes 1.14 and 1.12, respectively).

Your program must:

- Prompt the user to enter any octal integer value and use `cin >>` to read the entire value at once into a type `int` variable.
- Use `cout <<` as often as necessary to 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
  `-00000000000000000000000026450`
  the following would get displayed:
  `"-26450" in words is "minus two six four five zero"`
- Not declare variables or use casts that are not type `int` or type `bool`.
- Not use anything involving floating point types (the `pow` function, `math.h`, type `double`, etc.).
- 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.
- Not use nested loops – they are unnecessary.
- Not use any separate code to handle a user input of zero.

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

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A3E3_174273_U09845800** 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. Do not put a space before or after the number words in the string literals in your `switch` statement cases. Instead, separately add a space before each word just before displaying it unless it is the first word in the sequence.

A detailed algorithm hint is on the next page…

Hints, continued

2. First see notes 1.12 & 1.14 for information on doing octal I/O in C++. Then implement the optional algorithm below, which displays a user octal integer input value in words, one at a time moving left to right. 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. Testing whether an expression is true or false can be done in several different ways and the words **If** and **Else** do not necessarily refer to an actual "if" or "if-else" statement.

   Part A:
   - A1. Prompt the user, get his/her input, and display the initial double quote of the output message.
   - A2. If the user input value is negative change it to positive and display a minus sign.
   - A3. Display the positive user input value (made positive by step A2 if necessary).
   - A4. Display more of the output message up to the point where the first word of the value is needed.
   - A5. If the original input value was negative display the word "minus", followed by a space.

   Part B ("for" loop is used):

   Find a power of 8 divisor that will produce the most significant digit (MSD) of the positive input value as follows:
   - B1. Assign 1 to a divisor variable and the positive input value to a dividend variable.
   - B2. If the value of the dividend is greater than 7:
     a. Multiply the divisor by 8; the product becomes the new divisor.
     b. Divide the dividend by 8; the quotient becomes the new dividend.
     c. Repeat from step B2.

     Else Proceed to Part C below.

   Part C ("do" loop is used):

   The starting value for the divisor used in this part will be the value computed for it in Part B above. Part C will pick off the digits of the positive input value left to right and display them as words as follows:
   - C1. Assign the positive input value to a dividend variable.
   - C2. Divide the dividend by the divisor, which yields the MSD. Display it as a word using an 8-case switch statement (see below).
   - C3. Multiply the MSD by the divisor and reduce the dividend's value by that amount. (This removes the dividend's MSD.)
   - C4. Divide the divisor by 8; the result becomes the new divisor.
   - C5. If the new divisor is not equal to 0, repeat from step C2.
     Else You are finished displaying the number in words!

   <u>About the recommended "switch" statement…</u>

   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 7. 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.
}
```