Assignment 3
C/C++ Programming I

1
2
3
4
5
6 ## To Refresh Your Memory
7 ### Decimal, hexadecimal, octal, and binary integer literals
8 When a positive integer literal is coded, such as in `x = 232`, it may be written in decimal,
9 hexadecimal, octal, or in C++ only, in binary. The choice of which radix to use is strictly a matter of
10 convention, but regardless of which is chosen the value is stored in binary internally. For example,
11 the decimal, hexadecimal, octal, and binary literals `232`, `0xe8`, `0350`, and `0b11101000`, respectively,
12 each represent exactly the same value and are stored internally as the binary pattern `11101000`.
13
14 ### Outputting/Inputting integer values in decimal, hexadecimal, or octal in C++
15 By default, an integer value written using `cout <<` is displayed in decimal. Similarly, an integer value
16 read using `cin >>` is interpreted as decimal. However, the radix can be changed between decimal,
17 hexadecimal, and octal as desired using the **dec**, **hex**, and **oct** *manipulators*. Their appropriate
18 placement in `cout <<` and `cin >>` expressions changes the integer radix everywhere in the program,
19 including in other functions and files. There is no manipulator for binary.
20
21 The effect of the **dec**, **hex**, and **oct** manipulators is "sticky", meaning that once set it remains in
22 effect until explicitly changed. Because of this, specifying the same radix more than once for the
23 same data stream or specifying it in a loop is an unnecessary waste of resources unless the radix
24 must be changed in between.
25
26 A manipulator may be used alone in a `cout <<` or `cin >>` expression, or as part of such an
27 expression that does other things. Some examples are:
28
29   Output

```
30   int x = 232;                          Stored value is binary  11101000
31
32   cout << oct;                          Change the output radix to octal.
33   cout << "Value is" << x;              Output 232 as octal.  Output is:  350
34 OR
35   cout << oct << "Value is" << x;       Do both of the above in one statement.
36 OR
37   cout << "Value is" << oct << x;       Same effect as the previous statement.
38
39   Output 232 as hexadecimal, octal, and decimal.  Output is:  e8 350 232
40   cout << hex << x << ' ' << oct << x << ' ' << dec << x;
```

41
42   Input

```
43   int x, a, b, c;
44
45   cin >> hex;                           Change the input radix to hexadecimal.
46   cin >> x;                             Input e8 as hexadecimal.  Stored value is binary:  11101000
47 OR
48   cin >> hex >> x;                      Do both of the above in one statement.
49
50   Input e8 350 232 as hexadecimal, octal, and decimal.  All stored values are binary:  11101000
51   cin >> hex >> a >> oct >> b >> dec >> c;
```

Page 1 of 2 of C1A3 General Information

1 **Get a Consolidated Assignment 3 Report (optional)**

2 If you would like to receive a consolidated report containing the results of the most recent version of
3 each exercise submitted for this assignment:

4   `Send an empty-body email to the assignment checker with the subject line **C1A3_162461_U09339356**
5   and no attachments.

6 Inspect the report carefully since it is what I will be grading. You may resubmit exercises and report
7 requests as many times as you wish before the assignment deadline.

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, but basing an answer on runtime results 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("%i\n", 6/3 + !2.2 + 3);*
    (Note 3.2)
    A. *7*
    B. *7.2*
    C. It will not compile.
    D. *5*
    E. garbage because **6/3 + !2.2 + 3** is type **double** but *%i* specifies type **int**

2. Predict the output from:
    ```
    if (5 < 4)
        if (6 > 5)
            cout.put('4');
    else
        cout.put('3');
    else
        cout.put('2');
    cout.put('1');
    ```
    (Note 3.15)
    A. *31*
    B. *21*
    C. *321*
    D. *41*
    E. Implementation dependent

3. Predict the output from:
    ```
    switch (2 * 2)
    {
        case 8/2:  cout << "value = 4";
        case 29:  cout << "29 ";
        case 'E':  cout << 'E' << " ";
        default:  cout << "break ";
        break; case 2/2:  cout << "Got an 'A' ";
    }
    ```

    (Notes 3.17 & 3.18)
    A. *value = 4*
    B. *value = 429 E break*
    C. *value = 429 E*
    D. *Got an 'A'*
    E. The output is implementation dependent.

4. If the ASCII character set is being used, what gets printed by:
    putchar(putchar('z') - putchar('A'));
    (Notes 3.3 & B.1)
    A. *zA9* only
    B. *Az9* only
    C. either *zA9* or *Az9*
    D. either *zA9* or *Az9* or *9zA* or *9Az*
    E. Possibly something not listed above

5. What gets printed by:
    putchar('A') || putchar('\0x0');
    (Notes 1.5 & 3.3)
    A. *A* and garbage
    B. *A* and *\0x0*
    C. either *A* or *\0x0* but not both
    D. either *A* or *B* but not both
    E. *A* only

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

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A3E0_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.

---

1 **C1A3E1** *(3 points – C Program)*

2 Exclude any existing source code files that may already be in your IDE project and add a new one,
3 naming it **C1A3E1_main.c**. Write a program in that file to compute and display a table of factorials. The
4 factorial of an integral value *n*, written *n!*, is the product of the consecutive integers *n* down through *0*.
5 Since *0!* is defined as having a value of *1* it can be omitted from the calculations since it does not affect
6 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
7 *120*. Below is a table of *1!* through *7!*

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

18 **IMPORTANT:**
19 Your code must use a type **long** (not **unsigned long**) variable to represent the value of factorial **nbr!** in
20 the table above. The results must be correct up to the maximum value type **long** can represent on any
21 and every machine on which your unaltered code is compiled and run. Since compiler manufacturers
22 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
23 conceivably be so great that hundreds of digits would be required to represent it.

25 In addition to the above requirements, your program must:
26   1. Prompt the user to enter an integer value greater than 0 and store it in a type **int** variable.
27   2. Compute and display a table like the one illustrated above for all values *1!* through the factorial
28   of the value entered by the user. Values must be displayed as decimal integers – no exponents
29   or decimal points.
30   3. Align the least significant digits in both columns for all entries in the table. Do not attempt to
31   write code to compute the field widths needed for these columns. Instead, a fixed width of 2 for
32   the 1st column and 11 for the 2nd is fine for the factorial values tested in this exercise unless you
33   start getting misalignments or simply want to make them wider. Separate the fields with at least
34   one space so the numbers will not run together.
35   4. Use a row of hyphens to separate the column titles from the values.
36   5. Not use floating point literals, floating point variables, floating point functions, or floating point
37   type casts.
38   6. Not use an **if** statement or more than 1 looping statement.
39   7. Not use arrays or recursion; recursion occurs when a function is called before a previous call to
40   that function has returned, for example, when a function calls itself.

43 Manually re-run your program several times, testing with at least the following 3 input values:
44   **1   25   36**
45 If you find that any of the factorial values are incorrect determine if the expected values exceed the
46 maximum value supported by type **long** on your machine, in which case they should be incorrect. Even
47 if they are all correct, they will eventually exceed the maximum if the user input value is increased
48 sufficiently.

50 Suggest a possible way, without restricting user input or the number of values output, the program could
51 be modified so that all values would be correct, but do not incorporate your suggestion into the code
52 you will be submitting for grading. Instead, merely place your suggestion as a comment in the file's
53 "Title Block". Note that even using type `unsigned long long` or type `long double` will not eliminate
54 eventual erroneous values.

1

Submitting your solution

3 `Send an empty-body email to the assignment checker with the subject line **C1A3E1_162461_U09339356**
4 and with your source code file <u>attached</u>.

5 *See the course document titled "How to Prepare and Submit Assignments" for additional exercise*
6 *formatting, submission, and assignment checker requirements.*

7

8

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. 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 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.
   Else, you are done!

1   **C1A3E2** *(5 points – C++ Program)*

2   Exclude any existing source code files that may already be in your IDE project and add a new one,
3   naming it **C1A3E2_main.cpp**.   Write a program in that file to reverse the digits of an arbitrary
4   user-entered hexadecimal integer value based solely upon its numeric value, not the individual
5   characters entered. If the value is negative the minus sign must be displayed last. **If you are not familiar**
6   **with the hexadecimal number system, see note C.1.**   Here are some sample input values and the
7   expected reversals:

8
9

Input	Reversal
aBcD	**dcba**
-26C5	**5c62-**
100	**001**
000120	**021**
-002F	**f2-**
000	**0**

17
18   Your program **<u>must</u>** interpret all integer input and display all integer output as hexadecimal (notes 1.14
19   and 1.12, respectively).

20
21   Your program must:
22   • prompt the user to enter any hexadecimal integer value and use `cin >>` to read the entire value
23   at once into a type **int** variable.
24   • use `cout <<` to display the variable's value and the reversed value in the format below, placing
25   double quotes around both for readability. For example, if the user input is
26   `-00000000000000000000000Ab450`
27   the following would get displayed:
28   `"-ab450" in reverse is "054ba-"`
29   • <u>not</u> declare variables or use casts that are not type **int** or type **bool**.
30   • <u>not</u> use anything involving floating point types (the `pow` function, `math.h`, type **double**, etc.).
31   • <u>not</u> use arrays or recursion; recursion occurs when a function is called before a previous call to
32   that function has returned, for example, when a function calls itself.
33   • <u>not</u> use any separate code to handle a user input of zero.

34
35   <u>Uppercase/lowercase does not matter for the 6 hex letters.</u>

36
37   Manually re-run your program several times, testing with at least the following 6 input values:
38   **3   -A2b   0   10c0   -1010   -000F000**
39
40
41   **Submitting your solution**

42   `Send an empty-body email to the assignment checker with the subject line **C1A3E2_162461_U09339356**
43   and with your source code file <u>attached</u>.

44   *See the course document titled "How to Prepare and Submit Assignments" for additional exercise*
45   *formatting, submission, and assignment checker requirements.*

46
47   **Hints:**

48
49   Detailed hints are on the next page…

1   See notes 1.12 & 1.14 for information on doing hexadecimal I/O in C++. The recommended, but not
2   required, algorithm below uses a "do" loop to pick off and display the digits of the user input value one
3   at a time moving right-to-left. A special case to handle zero is unnecessary. Testing whether an
4   expression is true or false can be done in several different ways and the words **If** and **Else** do not
5   necessarily refer to an actual "if" or "if-else" statement.

6
7   1.   Prompt the user for input then read it into a variable named **inValue**.
8   2.   Display a double quote, which is the first character of the required output message.
9   3.   Use a Boolean variable to remember if the input value was positive or negative.
10   4.   If the input value was negative, make **inValue** positive and display a minus sign.
11   5.   Display more of the required output message up to where the reversed value should start.
12   6.   Modulo-divide **inValue** by 16 to produce its least significant digit (LSD), then display that LSD.
13   7.   Divide **inValue** by 16 to remove its LSD and assign the result back into **inValue**.
14   8.   If **inValue** is not equal to 0, repeat from step 6.
15   9.   Else, if the original user input value was negative, display a minus sign.
16   10. Finish the display.
17   11. You are done!

Page 2 of 2 of C1A3E2

1 **C1A3E3** *(6 points – C++ Program)*

2 Exclude any existing source code files that may already be in your IDE project and add a new one,
3 naming it **C1A3E3_main.cpp**.   Write a program in that file to convert an arbitrary user-entered
4 hexadecimal integer value into words based solely upon its numeric value, not the individual characters
5 entered.   If the value is negative the word `minus` must be first. **If you are not familiar with the**
6 **hexadecimal number system see note C.1.** Here are some sample input values and the expected
7 words:

Input	Words
00000	**zero**
5a3	**five A three**
-AbD	**minus A B D**
-500	**minus five zero zero**
-000500	**minus five zero zero**

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

20 Your program must:
21 • prompt the user to enter any hexadecimal integer value and use `cin >>` to read the entire value
22   at once into a type **int** variable.
23 • use `cout <<` to display the variable's value and the equivalent words in the format below, placing
24   double quotes around both for readability and using single letters as the words for the 6 hex
25   letters. For example, if the user input is
26   `-000000000000000000000002aBc50`
27   the following would get displayed:
28   **"-2abc50" in words is "minus two A B C five zero"**
29 • <u>not</u> declare variables or use casts that are not type **int** or type **bool**.
30 • <u>not</u> use anything involving floating point types (the `pow` function, `math.h`, type **double**, etc.).
31 • <u>not</u> use arrays or recursion; recursion occurs when a function is called before a previous call to
32   that same function has returned, for example, when a function calls itself.
33 • <u>not</u> use nested loops – they are unnecessary.
34 • <u>not</u> use any separate code to handle a user input of zero.

36 <u>Uppercase/lowercase does not matter for the 6 hex letters.</u>

38 Manually re-run your program several times, testing with at least the following 6 input values:
39   **3   -1aB   0   1010   -1010   -000f000**

42 **Submitting your solution**

43 `Send an empty-body email to the assignment checker with the subject line **C1A3E3_162461_U09339356**
44 and with your source code file <u>attached</u>.

45 *See the course document titled "How to Prepare and Submit Assignments" for additional exercise*
46 *formatting, submission, and assignment checker requirements.*

49 **Hints:**
50 Detailed hints are on the next page…

1	**Hints for Exercise 3:**
2	First see notes 1.12 & 1.14 for information on doing hexadecimal I/O in C++. Then implement the
3	optional algorithm below, which displays a user hexadecimal integer input value in words, one at a time
4	moving left to right. There are no nested loops, part A is completed before part B begins, and part B is
5	completed before part C begins. Only one instance of the code for each part is necessary. Testing
6	whether an expression is true or false can be done in several different ways and the words **If** and **Else** do
7	not necessarily refer to an actual "if" or "if-else" statement.
8	
9	Part A:
10	A1. Prompt the user, get his/her input, and display the initial double quote of the output message.
11	A2. If the user input value is negative change it to positive and display a minus sign.
12	A3. Display the positive user input value (made positive by step A2 if necessary).
13	A4. Display more of the output message up to the point where the first word of the value is
14	needed.
15	A5. If the original input value was negative display the word "minus", followed by a space.
16	
17	
18	Part B ("for" loop is used):
19	Find a power of 16 divisor that will produce the most significant digit (MSD) of the positive input
20	value as follows:
21	B1. Assign 1 to a divisor variable and the positive input value to a dividend variable.
22	B2. If the value of the dividend is greater than 15:
23	a. Multiply the divisor by 16; the product becomes the new divisor.
24	b. Divide the dividend by 16; the quotient becomes the new dividend.
25	c. Repeat from step B2.
26	Else Proceed to Part C below.
27	
28	Part C ("do" loop is used):
29	The starting value for the divisor used in this part will be the value computed for it in Part B
30	above. Part C will pick off the digits of the positive input value left to right and display them as
31	words as follows:
32	C1. Assign the positive input value to a dividend variable.
33	C2. Divide the dividend by the divisor, which yields the MSD. Display it as a word using a
34	16-case switch statement (see below).
35	C3. Multiply the MSD by the divisor and reduce the dividend's value by that amount. (This
36	removes the dividend's MSD.)
37	C4. Divide the divisor by 16; the result becomes the new divisor.
38	C5. If the new divisor is not equal to 0, repeat from step C2.
39	Else You are finished displaying the number in words!
40	
41	<u>About the recommended "switch" statement...</u>
42	While the use of "magic numbers" is usually a bad idea, in some situations they are appropriate
43	such as for the "cases" used in the "switch statement" recommended for this exercise. Specifically,
44	each case represents a unique numeric value ranging from 0 through 15 (hex F). There is no
45	underlying meaning to these values other than the values themselves, their purpose is obvious and
46	unmistakable, there is no possibility that they might ever need to be changed, and there is no
47	identifier (name) that would make their meaning any clearer. Thus, the literal values should be
48	specified directly, as follows:
49	`switch (…)`
50	`{`
51	`    case 0:    …`
52	`    case 1:    …`
53	`    case 2:    …`
54	`    etc.`
55	`}`