Assignment 2
C/C++ Programming I

C1A2 General Information

Limiting the "Scope" of Variables

The scope of an identifier (a name) is defined as the section of code over which it is accessible. The scope of a variable declared inside a function extends from that declaration to the end of the block in which it is declared. Good programming practice dictates that the scopes of non-const variables be as small as possible to prevent their values from being changed by code that should not change them. Constant variables that are being used in C++ instead of macros are exceptions. For readability these are often defined in the same place the equivalent macros would have been defined if it were C code. Otherwise, they should be defined first in the function or block that uses them. Consider the following examples:

```
1    type Function(...)
2    {
3       int x, y, z;
4       for (x = 0; x < VAL1; ++x)
5       {
6          for (y = 0; y < VAL2; ++y)
7          {
8             if (x + y > VAL3)
9             {
10               z = x - y;
11            }
12         }
13      }
14   }
15
16   type Function(...)
17   {
18      int x;
19      for (x = 0; x < VAL1; ++x)
20      {
21         int y;
22         for (y = 0; y < VAL2; ++y)
23         {
24            if (x + y > VAL3)
25            {
26               int z = x - y;
27            }
28         }
29      }
30   }
31
32   type Function(...)
33   {
34      for (int x = 0; x < VAL1; ++x)
35      {
36         for (int y = 0; y < VAL2; ++y)
37         {
38            if (x + y > VAL3)
39            {
40               int z = x - y;
41            }
42         }
43      }
44   }
```

Poor Declaration Placement

All variables are declared on line 3, which is inside the block that starts on line 2 and ends on line 14. Thus, their scopes extend from line 3 to line 14 and all are accessible anywhere within that region. However, since variable **y** is only needed from line 6 through line 10 and variable **z** is only needed on line 10, their scopes are both wider than necessary. Regardless of scope, good practice dictates that whenever appropriate a "for" statement's loop count variable be initialized in its "initial expression" rather than in the variable's original declaration.

Better Declaration Placement

Variable **x** is declared as in the previous example because it is needed from line 19 through line 26. Its scope extends from line 18 to line 30. However, since variable **y** is only needed from line 22 through line 26 it is declared on line 21, which is inside the block that begins on line 20 and ends on line 29. Thus, its scope only extends from line 21 to line 29. Finally, since variable **z** is only needed on line 26 it is declared there, which is inside the block that begins on line 25 and ends on line 27. Its scope only extends from line 26 to line 27.

Best Declaration Placement

Although variables that are not being used as "for" loop counters should be declared as in the previous example, those that are being used for that purpose should be declared and initialized as shown in this example if appropriate. This limits their scope to the "for" statement only. That is, the scope of variable **x** is now from line 34 to line 43 and the scope of variable **y** is now from line 36 to line 42.

Get a Consolidated Assignment 2 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 **C1A2_162461_U09339367** 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.

C1A2E0 *(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 **C1A2E0_Quiz.txt** formatted as:

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

1. Which of the following is wrong regarding the standard macro named *CHAR_BIT*? (Note 2.3)
   A. You must never define it yourself.
   B. It is available in header files *limits.h* and *climits*.
   C. It represents the number of bits in a byte (in data type **char**).
   D. Program code should use it instead of a literal 8 to represent the number of bits in type **char.**
   E. Its value is always 8

2. If your program needs both integer and floating variables and if any type from each category will work, which are usually preferred, respectively? (Notes 2.1 & 2.4)
   A. **char** and **float**
   B. **int** and **float**
   C. **unsigned int** and **double**
   D. **int** and **double**
   E. **long long int** and **long double**

3. The data types of the literals *2* and *2.0*, respectively, are: (Notes 2.1, 2.2, & 2.4)
   A. **integer** and **long double**
   B. **int** and **double**
   C. **int** and **floating**
   D. **int** and **float**
   E. implementation dependent

4. What is the data type of an expression containing only type **char** and type **short** operands? (Note 2.10)
   A. **char**
   B. **short**
   C. **char**, **unsigned char**, or **short**
   D. **int** or **unsigned int**
   E. **int** or **long**

5. What is the biggest problem with macro SUM below, which is intended to represent the sum of 5 and 3?
   #define SUM 5+3
   (Note 2.13)
   A. Its value might not be 8
   B. It contains magic numbers.
   C. There is no space around the + operator.
   D. Macros are not supported in C++.
   E. There is no major problem.

6. What is wrong with:
   *printf("%u", **sizeof**(**double**));*
   (Note 2.12)
   A. There is nothing wrong.
   B. **sizeof**(**double**)should be **sizeof**(**unsigned int**) to match the *%u*.
   C. The data type produced by **sizeof** might not be compatible with *%u*.
   D. printf cannot reliably display values produced by **sizeof**.
   E. The value produced by **sizeof** might be negative.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A2E0_162461_U09339367** 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.

C1A2E1 *(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 **C1A2E1_main.cpp**. Write a program in that file whose primary purpose is to convert an uppercase character to its lowercase equivalent.

Your program must:

1. Prompt (ask) the user to enter any character, then use `cin.get` to read it.
2. Add a constant value to that character and display the resulting character. The constant value must be such that if the input character is uppercase, the displayed character will be its lowercase equivalent if the ASCII character set is used.
3. Display the results in the following format with single quotes around the original and converted characters, respectively. For example, if the user inputs the character **A** the output must be exactly:

   `The lowercase equivalent of 'A' is 'a'`

   Or if the user inputs the character **$** the output must be exactly:

   `The lowercase equivalent of '$' is 'D'`

Your program must:

4. Not test anything. Simply apply the same conversion algorithm to any input character regardless of what it might be.
5. Not use any bitwise operators (from section 11 of the course book).
6. Not use `tolower` or any other function to do the conversion (although `tolower` is the best solution in "real life").
7. Not name any variable `uppercase` (to avoid standard library conflicts and a bogus assignment checker warning).

Manually re-run your program several times, testing with at least the following 6 input characters:

`B Z p 0 %` ***a literal space***

Explain what happens and why in the following two situations and place these explanations as comments in your "Title Block":

1. The user enters anything other than an uppercase character.
2. The user precedes the input character with a whitespace.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A2E1_162461_U09339367** 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 most general way to represent the numerical difference between the ASCII uppercase and lowercase character sets is the expression **`'a' - 'A'`**. Initialize a constant variable (Note 2.14) to that expression and use it in your code and comments as needed. Note, however, that the standard library function `tolower` provides the most portable solution, although you are not allowed to use it in this exercise. This function and its `toupper` counterpart will do the conversions in a completely portable way without regard for the specific characteristics of whatever character set is being used. These functions are described in note 3.19.

C1A2E2 *(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 **C1A2E2_main.c**. Write a program in that file to display a triangle of characters on the screen.

Your program must:

1. Prompt (ask) the user to enter any positive decimal integer value.
2. Use nested "for" loops to display that number of lines of characters on the console screen starting in column 1, with each successive line containing one less character than the previous. Each line must end with a "diagonal" character and any preceding characters must be "leader" characters. The last line will only contain the diagonal character. For example, if the user inputs a **4**, the leader character is **^**, and the diagonal character is **@**, the following will be displayed:

   ```
   ^^^@
   ^^@
   ^@
   @
   ```

3. Use the exact identifier names `LEADER_CHAR` and `DIAGONAL_CHAR` for macros that represent the desired leader and diagonal characters, respectively. Embedding the actual leader and/or diagonal characters themselves (or their actual names such as "*at*", "*wave*", "*pound*", "*hash*", "*dollar*", "*percent*", "*dot*", "*two*", "*five*", etc.) in the body of your code or in any of your comments is an inappropriate use of "magic numbers".
4. Not contain more than two loop statements.

Manually re-run your program several times, testing with several different line counts, leader characters, and diagonal characters. To change the leader and diagonal characters you must change the values associated with the `LEADER_CHAR` and `DIAGONAL_CHAR` macros and recompile.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A2E2_162461_U09339367** 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:

A "nested loop" is merely a loop of any type that is within the body of another loop of any type. The code in the following example uses two "for" loops to display sequential values from 00 through 99, where the outer loop controls the most significant digit and the inner loop controls the least significant digit. "Magic Numbers" are used only for illustration. Additional code may be added where desired.

```
for (int msd = 0; msd < 10; ++msd)          // the loop that controls the MSD
{
   for (int lsd = 0; lsd < 10; ++lsd)       // the loop that controls the LSD
   {
      cout << msd << lsd << '\n';           // display the MSD and LSD
   }
}
```

For this exercise the outer loop should be used to keep track of the number of lines and the inner loop should be used to keep track of the number of leader characters on a line. A "for" loop should normally be used whenever a variable must be initialized when the loop is first entered, then tested and updated for each iteration. It is inappropriate to use a "while" loop or a "do" loop under these conditions. Be

sure to choose meaningful names for your loop count variables noting that names like "i", "j", "k", "outer", "inner", "loop1", "loop2", "counter", etc., are non-informative and totally inappropriate. Only three variables are necessary to complete this exercise and no "if" statement is necessary. If you use more than three variables or an "if" statement you are unnecessarily complicating the code.

THERE IS ANOTHER EXERCISE ON THE NEXT PAGE

C1A2E3 *(4 points – C++ Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one, naming it **C1A2E3_main.cpp**. Write a program in that file to display a triangle of characters on the screen.

Your program must:

1. Prompt (ask) the user to enter any positive decimal integer value.
2. Use nested "for" loops to display that number of lines of characters on the console screen starting in column 1, with each successive line containing one fewer character than the previous. Each line must end with a "diagonal" character and any preceding characters must be "leader" characters. For example, if the user inputs a **4**, the leader character is **^**, and the diagonal character is **$**, the following will be displayed:

   ^^^$
   ^^$
   ^$
   $
3. Use two separate statements on two separate lines to declare and initialize **const char** variables named `LEADER_CHAR` and `DIAGONAL_CHAR` that represent the desired leader and diagonal characters, respectively.
4. DO NOT embed the actual leader or diagonal characters (or their names such as "*at*", "*wave*", "*pound*", "*hash*", "*dollar*", "*percent*", "*dot*", "*two*", "*five*", etc.) in the body of your code or in any of your comments. That would be an inappropriate use of "magic numbers".
5. DO NOT use more than two loop statements.

Manually re-run your program several times, testing with several different line counts, leader characters, and diagonal characters. To change the leader and diagonal characters you must change the values associated with the `LEADER_CHAR` and `DIAGONAL_CHAR` variables and recompile.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A2E3_162461_U09339367** 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:
A "nested loop" is merely a loop of any type that is within the body of another loop of any type. The code in the following example uses two "for" loops to display sequential values from 00 through 99, where the outer loop controls the most significant digit and the inner loop controls the least significant digit. "Magic Numbers" are used only for illustration. Additional code may be added where desired.

```
for (int msd = 0; msd < 10; ++msd)          // the loop that controls the MSD
{
   for (int lsd = 0; lsd < 10; ++lsd)       // the loop that controls the LSD
   {
      cout << msd << lsd << '\n';           // display the MSD and LSD
   }
}
```

For this exercise the outer loop should be used to keep track of the number of lines and the inner loop should be used to keep track of the number of leader characters on a line. A "for" loop should normally be used whenever a variable must be initialized when the loop is first entered, then tested and updated

for each iteration. It is inappropriate to use a "while" loop or a "do" loop under these conditions. Be sure to choose meaningful names for your loop count variables noting that names like "i", "j", "k", "outer", "inner", "loop1", "loop2", "counter", etc., are non-informative and totally inappropriate. Only three variables other than `LEADER_CHAR` and `DIAGONAL_CHAR` are necessary to complete this exercise and no "if" statement is necessary. If you use more than three variables or an "if" statement you are unnecessarily complicating the code.

