

C/C++ Programming 1 - Assignment 2

See the course website for detailed information on assignment requirements. All quiz and program submissions must be **MACHINE PRINTED**; handwritten materials are acceptable only for diagrams. Place the quiz first, followed by the program materials, as indicated below. For in-class turn-in please staple all materials together.

Quiz Requirements - 30% of Assignment Score

Quiz answers must be on one sheet in the following format. The choices I have shown are examples only:

Your name & programmer ID

Your email address

Assignment number

Date

Course Title, Section ID, Instructor's Name

1. A
 2. B
 3. C
 4. D
 5. E
 6. A
 7. B
 8. C
-

Exercise Requirements - 70% of Assignment Score

Separate exercises must be on separate sheets. Non-programming exercises (if any) simply require the submission of the answer(s) to the question(s). Programming exercises require that you submit the following for each, in the order shown (do not turn in compiler output):

1. the error-free & warning-free email report from the “Automated Assignment Checker”
2. the working, properly formatted program source code,
3. a screen dump (or equivalent) of the program run(s) and/or any files produced by the program, as appropriate. This **must** include the results of **all** required test cases plus any other appropriate tests.

All C program source code files (including any header files) must begin with a block comment in the form shown below (for C++ programs you may start each line with // instead, if desired). In addition, each individual function in the program (including *main*) must be preceded by a block comment describing its operation, parameters, and return. Appropriate comments related to pertinent lines or sections of code are also required.

```
/*  
 * Your name & programmer ID  
 * Your email address  
 * Assignment and Exercise number  
 * Date of development  
 * Name of source file (myProgram.c, myProgram.cpp, etc.)  
 * Your operating system (WinXP, Mac OS X, UNIX, LINUX, etc.)  
 * Your compiler & version (Visual C++ 6.0, Borland 7.0, etc.)  
 * Name of this course, section ID, and instructor  
 *  
 * Detailed description of the purpose of the code and any information pertinent to its  
 * maintenance, limitations, peculiarities, user interface, and I/O requirements.  
*/
```

Quiz (8 points)

This is a theoretical "paper only" quiz in which you must assume a perfectly implemented ANSI C/C++ compiler. How any particular program runs on your computer only indicates how your computer runs that program and not necessarily how it should run or how portable it is. There is only one correct answer to each question.

1. The data type of the literal 252767 is:
 - A. implementation dependent
 - B. **long**
 - C. **int**
 - D. not defined in pre-ANSI C
 - E. none of the above

2. The data types of:
 - a) unsuffixed floating literals and,
 - b) unsuffixed integer literals, are:
 - A. implementation dependent in both cases.
 - B. determined by the value of the literals in both cases.
 - C. a) **double**
b) determined by the number of digits
 - D. a) **double**
b) determined by the value of the literal
 - E. none of the above

3. Floating point data types should be used to represent numerical values:
 - A. only when integer types will not reasonably suffice
 - B. There are no guidelines covering this.
 - C. whenever you are in doubt about what type to use
 - D. only in pre-ANSI C programs
 - E. never

4. A mathematical operation where all operands are type **char** is
 - A. illegal - type of **char** is only used for character operations.
 - B. evaluated using type **signed char** arithmetic.
 - C. evaluated using type **char** arithmetic.
 - D. evaluated using type **int** or **unsigned int** arithmetic.
 - E. not as accurate as an operation with all type **long** operands.

5. Predict the output from:


```
printf("%d, %f\n", 5/-2, 5/-2.);
```

 - A. two possibilities:
-2, -2.500000 or -3, -2.500000
 - B. A general prediction cannot be made for all implementations
 - C. a compiler or run time error
 - D. -2, -2.500000
 - E. two possibilities:
-2, -2.500000 or -1, -2.500000

6. If **char** is 8 bits and **int** is 16, predict the output:


```
cout << -7 % 3 << ", ";  
cout << sizeof(-5 % 3) << endl;
```

 - A. two possibilities: -1, 2 or 2, 2
 - B. A general prediction cannot be made for all implementations.
 - C. -1, 2 or -1, 4 depending upon the data type of **sizeof**
 - D. two possibilities: -1, 2 or 1, 2
 - E. three possibilities: -1, 2 or 2, 2 or -2.3, 2

7. Which of the following statements guarantees the correct answer on any machine?
 - A. **long** value = 300 * 400 * 10L;
 - B. **long** value = 300 * 400L * 10;
 - C. **float** value = 300 * 400 * 10;
 - D. **double** value = **(double)**(300 * 400 * 10L);
 - E. none of the above, because the value of each is implementation dependent

8. An expression containing only one operator and no numbers, variables, or function calls, and whose value is the number of bytes in a **double** on any and every machine is:
 - A. **sizeof**(3.14159)
 - B. **sizeof(double)**
 - C. **sizeof(double) * 8**
 - D. **(double) * CHAR_BIT**
 - E. **GetByteCount(double)**

Exercises (10 points)

The point value of each exercise is shown in parentheses.

A "**magic number**" is defined as a numeric literal (and in some cases a character or string literal) embedded in a program's code or comments. They make programs cryptic and difficult to maintain because their meaning is not obvious and, if they must be changed, the likelihood of missing one or changing the wrong one is high. Instead, the `#define` directive (in C), **const**-qualified variables (in C++), or enumerated data (C and C++) should be used to associate meaningful names with literals. There are a few cases, however, where magic numbers are acceptable, including:

0 & 1 as array indices or loop start/end values, 1 as an increment/decrement value, 2 as way to double/halve a value or as a divisor to check for odd/even (but not in this course), coefficients in some mathematical formulae, some print strings, cases dictated by common sense

1. (4) **C++ Program** - Using no "magic numbers", write a program that:

- asks the user to input an upper case character and uses the `cin.get` method to read it,
- converts the character to lower case,
- uses the `cout.put` method to output the converted character.

Explain what happens when you input anything other than an upper case character or precede a character with whitespace.

2. (4) **C Program** - Using nested **for** loops, write a program that asks the user to input an integral number and then displays a diagonal line with that many dots on the console screen starting in column 1. You may assume that the user will not enter a number larger or smaller than the number of screen columns. For example, an input of 4 would display

```
.  
 .  
  .  
   .
```

Write the program in such a way that the character printed diagonally as well as the leader character(s) preceding it may be changed easily without digging through the actual code to find them, that is, use macros to define these characters.

3. (2) **C++ Program** - Rewrite the previous program in C++.