

## C/C++ Programming 1 - Assignment 3

---

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 evaluation of expressions involving **&&** and **||**, (i.e., `i == 2 && j == 1`), is always:
  - A. from right to left and continues until all operands have been evaluated
  - B. done in an implementation defined order
  - C. from left to right and continues until all operands have been evaluated
  - D. from left to right and stops as soon as the outcome is known
  - E. terminated when either one of the operands is found to be true
  
2. What is output: `printf( "%d\n", !(6/3+2.2) + 3);`
  - 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**
  
3. 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');
      
```

  - A. 4
  - B. 2
  - C. 24
  - D. 4 or 24, depending upon implementation
  - E. Nothing is printed.
  
4. 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' ";
}
      
```

  - 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.
  
5. What gets printed by the following statement:
 

```

putchar('A') + putchar('B');
      
```

  - A. *AB* only
  - B. *BA* only
  - C. either *AB* or *BA*
  - D. either *A* or *B* but not both
  - E. *A* only
  
6. What gets printed by the following statement:
 

```

putchar('A') && putchar('B');
      
```

  - A. *AB* only
  - B. *BA* only
  - C. either *AB* or *BA*
  - D. either *A* or *B* but not both
  - E. *A* only
  
7. What gets printed by the following statement:
 

```

putchar('A') || putchar('B');
      
```

  - A. *AB* only
  - B. *BA* only
  - C. either *AB* or *BA*
  - D. either *A* or *B* but not both
  - E. *A* only
  
8. For **int** `x = 5`; what is the value and data type of the following:
 

```

sqrt(9.0), ++x, printf("123"), 25
      
```

  - A. 3.0 (type **double**)
  - B. 3 (type **int**)
  - C. 25 (type **int**)
  - D. 6 (type **double**)
  - E. implementation dependent

**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. (2) **C Program** - The factorial of an integral value  $n$ , written  $n!$ , is the product of the consecutive integers 1 through  $n$ . For example, 5 factorial is calculated as:  $5! == 5 \times 4 \times 3 \times 2 \times 1 == 120$ . Write a program to compute and display the first 20 factorials (1! through 20!). It must be written such that only one `#define` need be changed to compute any desired number of factorials. Use no floating literals, floating point variables, or floating point functions. Which values are incorrect, why, and what if anything can be done to fix the problem? (You don't have to fix it).

2. (3) **C++ Program** - Write a program to reverse the digits of an arbitrary user entered integral value. For example, if the user inputs 3987 the program will output 7893 while an input of -2645 will result in an output of 5462-.

- Use one *cin* to input an entire number at once into a type **int** variable.
- Make absolutely no assumptions about the largest value an **int** can hold.
- Do not use anything involving floating point types, including floating point functions.
- Do not use the *pow* function or include `<cmath>` or `<math.h>`.
- Do not use any variables that are not type **int**.
- Do not use arrays.
- Do not use recursion.
- Use at least the following numbers as test cases: 3, -123, 0, 1010, -1010
- Hint: consider the remainder operator, %, but beware of using it on negative numbers.

3. (5) **C++ Program** - Write a program that displays, in English, each digit of an arbitrary user entered integer. If the user enters 593, the program must display *five nine three*. The program must indicate negative numbers by printing *minus* and must operate properly on the number 0 and other numbers ending with one or more 0s.

- Use one *cin* to input an entire number at once into a type **int** variable.
- Make absolutely no assumptions about the largest value an **int** can hold.
- Do not use anything involving floating point types, including floating point functions.
- Do not use the *pow* function or include `<cmath>` or `<math.h>`.
- Do not use any variables that are not type **int**.
- Do not use arrays.
- Do not use recursion.
- Use at least the following numbers as test cases: 3, -123, 0, 1010, -1010
- Hint:
  - a. Find a divisor that will put the number's most significant digit (msd) in the units place.
  - b. Divide the number by the divisor. This yields the msd, which is then output in English.
  - c. Multiply the msd found in step *b* by the divisor and subtract the result from the number. The difference becomes the new number (which is the old number with its msd removed).
  - d. Divide the divisor by 10.
  - e. Repeat steps *b* through *d* until the divisor becomes zero.