



1 Because hints sometimes tend to be misleading or stifle creative solutions, I recommend trying your best to do all
2 exercises without them. The following hints do not necessarily address all aspects of each exercise but instead
3 provide a starting point to hopefully put you on the right track. They also sometimes allude to common mistakes
4 that have been made by students in the past. If you consider these hints inadequate, please consider how many
5 courses provide hints at all.

Assignment 1 Exercise Hints

- 10 1. —No hint—
- 11
- 12 2. —No hint—
- 13
- 14 3. —No hint—
- 15
- 16 4. —No hint—
- 17
- 18 5. C/C++ has no exponentiation operator.
- 19
- 20 6. —No hint—
- 21
- 22 7. —No hint—
- 23
- 24 8. Be sure to restore any changed variables between tests.
- 25

Assignment 2 Exercise Hints

- 29 1. Do not use the constant 32 anywhere in your program.
- 30
- 31 2. A "Nested Loop" is a loop that is part of the body of another loop.
- 32
- 33 3. —No hint—
- 34

Assignment 3 Exercise Hints

- 38 1. —No hint—
- 39
- 40 2. Use `%10` to produce the least significant digit of any decimal integral value. Use `/10` to eliminate that
41 digit from the value.
- 42
- 43 3. Consider the following algorithm:
 - 44 1. Find a divisor that will put the number's most significant digit (msd) in the units place.
 - 45 2. Divide the number by the divisor. This yields the msd, which is then printed using the **switch**
46 construct.
 - 47 3. Multiply the msd found in step 2 by the divisor and subtract it from the number. The result
48 becomes the new number (which is the old number with its msd removed).
 - 49 4. Divide the divisor by 10.
 - 50 5. Repeat steps 2 through 4 until the divisor becomes zero.
- 51
- 52

Assignment 4 Exercise Hints



1. Good programming practice dictates that no executable code be placed in a header file. See your compiler documentation or the documentation on the course website concerning creating a program from multiple source files. IDE users may simply add the additional files to their project.
2. When called in the presence of a function prototype, all compatible arguments are converted to the type of the corresponding function parameter. Always return type **void** from a function unless a non-**void** return value would serve a meaningful purpose.
3. —No hint—
4. A macro substitution list containing more than one token must be placed in parentheses. Additionally, parentheses must be placed around every usage of macro arguments within a substitution list.

Assignment 5 Exercise Hints

1. —No hint—
2. Cast the return pointer to (*double **) if your compiler complains.
3. Cast the return reference to (*double &*) if your compiler complains.

Assignment 6 Exercise Hints

1. Most library functions that compute values, including *strlen*, do no printing. They merely return those values.
2. Most library functions that compute values, including *strcmp*, do no printing. They merely return those values.
3. Save the *result* pointer first so you can return it later. Don't mistakenly return a pointer to the end of your substring. Don't repeatedly perform unneeded math operations, such as *source + start* or *start + count*. The expressions **result++* and **source++* should appear somewhere in your program.

Assignment 7 Exercise Hints

1. No pointers other than the two parameters themselves are needed in the *DetermineElapsedTime* function. If both times are equal, the difference is 24 hours. If two large type **int** values are multiplied together the potential product could be greater than type **int** can represent. During assignments from a “more-precise” type to a “less-precise” type, typecast the “more-precise” type to eliminate compiler warnings. Never return a pointer to any automatic object.
2. Simply declaring a pointer does not make it point to a valid location. All pointers must be explicitly initialized before dereferencing. In this exercise the three uninitialized *name* pointers must be pointed to a usable area of memory before the food names are stored. Failing to test dynamic memory allocations for success is an error.

Assignment 8 Exercise Hints

1. Some C/C++ functions that read text input leave the terminating newline character in the input buffer. If it is still in the buffer when the C++ *getline* method is called, *getline* reads only up through that character. Beware of improper operation of *getline* in Microsoft compilers.



- 1 2. Failing to test files for a successful open is an error. EOF can not be correctly stored or detected using data
2 type **char** or **unsigned char** (Use type **int** instead.).
3
- 4 3. Failing to test files for a successful open is an error. The *tmpnam* function only provides a name for a file
5 and does not actually open that file. Beware of improper operation of *getline* in Microsoft compilers. EOF
6 can not be correctly stored or detected using data type **char** or **unsigned char** (Use type **int** instead.).