Directions: Read the problems carefully. *Circle* **all** correct answers to multiple choice questions. Explain "why" if you *circle* FALSE. Fill in **all** blanks (unless you decide a T/F is FALSE). *Circle* the answer you feel most appropriately fills the blank if choices are given. Answer to the best of your ability.

This is a take-home test. You may ask **ME** questions for clarification — *not* each other. Feel free to attach pages to the end of the test if you need to continue a long answer. Please keep in mind that questions that ask you to write code are intended to be done without the aid of the compiler.[1] Take your time, but don't lose track of how much time is left.

Good luck! Have fun!

1) If a function needs to change a variable that exists in the caller, it must have a _____ argument (through which that variable can be passed). Such an argument can be distinguished from a value argument by the presence of a _____ symbol between the type and argument name. These arguments need no additional memory space and execute more _(quick / slow)_ -ly than value arguments. This change in execution speed is because there is no need to _(change / copy / calculate)_ the actual argument's value during the call — the formal argument is simply linked to the actual argument's memory space.

2) Two [or more!] functions can be overloaded if they have the same _____ but either a different _____ or _____ of argument(s). Most programmers will expect that two overloaded functions will behave in a _(similar / different)_ way, but this may not be the case. That is why good programmers writing a function always place _____ before the function's prototype and why good programmers calling a function always read these to clarify any possible misconceptions before calling the function.

3) Default arguments must be placed at the _(left / right)_ end of the argument list. The default value specification can only be placed on the _(first / second / last)_ function header the compiler sees. Therefore, they are generally placed in the _____ of a regular function or on the _____ of an inline function.

4) A library typically consists of two files. The first is known as a(n) _____ file (whose file name usually ends in _____). And the second is known as a(n) _____ file (whose file name usually ends in _____). The first file may contain any _____ , _____ , and/or the _____ of non-`inline` functions which the library wants to share with others. The second will contain the _____ of those 'public' non-`inline` functions as well as any other items or information the library uses but doesn't want to be publicly known.

5) When writing a library, we place preprocessor directives in the interface file. What problem(s) are we trying to alleviate with these three preprocessor directives? (Hint: There are actually *two <u>related</u>* problems.)

[1] I can't keep you from using it, but try to do it on your own first. . . please?! *big sad puppy dog eyes*

6) Given that the following two function calls exist together in a single program — which compiles and runs successfully, explain how the compiler can distinguish what function to call in each case. (Hint: There are *at least* two possible explanations. . .)

```
func(a, b, c);
func(a, b);
```

7) Write a function which will read in a phone number from the user and 'return' it to the caller. The caller is expected to prompt the user before calling your function. The core code is given, just fill in your function around it. (Hint: What values should be arguments and which ones should be local variables?) (Hint 2: What kind of argument passing mechanism should you use?)
(Thought-provoker: Why should your function **not** prompt [literally] for the phone number?) (Thought-provoker 2: Could your function prompt and avoid the problem(s) this causes?)

```
cin >> area >> t >> exchange >> t >> line;
```

8) For each of the following, indicate whether you could use **inline**-ing, **default** arguments, **both**, or **neither**. (It would be helpful if you explained *what* you felt would default if you choose to default arguments. . .)

_____ i) function to generate random values in a specified range

_____ ii) function to return true if an event of a given probability has 'happened'

_____ iii) function to input a date from the user (mm/dd/yy) — adjusting for Y2K issues

_____ iv) function to calculate the area of a triangle

_____ v) function to print a specified number of copies of the current document to the specified printer

9) TRUE / FALSE Actual arguments must always be variables (i.e. memory locations).

TRUE / FALSE Actual arguments may be literals, constants, or expressions if the formal argument is passed with the reference mechanism.

TRUE / FALSE The return mechanism of a function is often used to produce two or more values.

TRUE / FALSE Value arguments are literal actual arguments — as opposed to actual arguments which are variables or expressions.

10) TRUE / FALSE A function may change the values of its local variables (including any value arguments it accepts).

TRUE / FALSE Functions may not change any constants they have access to: global or local.

TRUE / FALSE Although a function normally cannot change the memory locations owned by another function (its caller, for instance), it can when using the reference mechanism.

TRUE / FALSE When a programmer calls a function with a reference argument, they are giving that function permission to alter the memory location they place in the corresponding actual argument.

TRUE / FALSE That's why good programmers never call functions with reference arguments.

11) Given functions with the purposes on the left, indicate which of the libraries on the right you would place each into by writing its letter in the blank next to the library 'title'. If none of the libraries seems right to you, you may 'create' your own in the extra spaces provided — but think carefully before you decide to do so. (And do **NOT** suggest a standard library!)

A) prints a point
B) makes random whole numbers
C) rounds decimal numbers
D) calculates distance between points
E) calculates midpoints
F) makes random decimal numbers
G) clears a line of input
H) clears the screen
I) determines which of two numbers is larger
J) makes random characters

_____ point library
_____ random library
_____ i/o library
_____ math library
_____ _____ library
_____ _____ library

12) Show the new layout for a program using functions (but not necessarily libraries). (i.e. Where's `main`? The `#includes`? Etc.) (Thought-provoker: Where would `inline` functions go?)

13) When you are writing a library, what three preprocessor directives must you place in your interface file?

What symbol/name must you provide?

Show a skeleton of the layout for an interface file with all these features in place.

14) Write a function to round a number (specified by the caller) to the nearest value at a certain decimal place. The decimal place should be the one's place by default. `return` the number resulting from the rounding to the caller. The formula to use is given — just fill in your function around it. (Thought provoker: Would any similar functions be potentially useful to your caller?)

```
floor(num/place+.5)*place
```

15) We should only place the _____ keyword in front of functions that are really simple and uncomplicated. (Otherwise the compiler will do the machine equivalent of laughing at us.) When we do this to a function, it must be _____ right away so that the compiler can replace any upcoming _____ to it with the binary code for the function.

16) The _____ arguments appear in the function's head. They may be passed to the function via either the _____ mechanism or the _____ mechanism. The latter will accept a copy of the _____ argument (the one listed in the function call). The former will instead form a direct link to the memory location of the caller's argument — thus allowing the function to change that memory even though the function doesn't own it (i.e. that memory location isn't for one of the function's _(global / local)_ variables/`const`ants).

Bonus Problems:

17) TRUE / FALSE We cannot have default arguments on an `inline` function.
TRUE / FALSE We cannot have both value and reference arguments on the same function.
TRUE / FALSE We cannot have a reference argument with a default value.
TRUE / FALSE We cannot (effectively) give prototypes for `inline` functions.

18) Explain the differences between a value argument and a reference argument in how they are treated during a function call. (Hint: there are **three (3)** major differences I am looking for here: one affects the caller, another affects the designer, and the third is in how the mechanisms work.)

19) Why must `inline` function definitions be placed in a library interface file with their fellow functions' prototypes?

20) Write a function that will `return true` a certain percent of the time (the caller specifies the percent; it should default to 50%) and `false` the rest of the time. The formula to use is given — just fill in your function around it.

```
rand_range(0., 1.) <= probability
```

21) Can you use branching constructs in any function besides `main`?

What about branching in a library function?

Can branches be used in `inline` functions?

22) TRUE / FALSE When a programmer calls a function with default arguments, they can leave out any arguments they don't have a particular value for.

TRUE / FALSE Default arguments must go at the beginning of the argument list so that the compiler will know they've been skipped when it sees something like `f(,,val)` in the call.

TRUE / FALSE Only reference arguments may be defaulted to a particular value.

TRUE / FALSE Value arguments must be defaulted to a global `const`ant.

TRUE / FALSE The default value itself must be placed on all function heads the compiler finds — especially when they are in separate files (like with a function in a library).

23) What would happen if the symbol used in the `#ifndef`/`#define` directives wasn't unique? (Hint: Proof by contradiction is a wonderful thing. . .)

24) TRUE / FALSE By passing a `string` argument to a function, we can often make that function more re-usable.

TRUE / FALSE This is especially true for functions that control input — prompts make ideal candidates for `string` arguments.

TRUE / FALSE As always, we like to pass `string`s by reference to avoid the nasty copying issues of value arguments.

TRUE / FALSE But, since we aren't going to change the prompts, we typically make the argument a `const`ant reference — fast from not copying and still protected from accidental change like a value argument!

TRUE / FALSE Being `const`ant reference (instead of plain reference), we can even provide default values for these arguments! (A common one for prompts is `""` — the empty `string`.)

25) Overload three functions for printing data. One will print a time-of-day (`22:40`). Another will display a 2D point (`(3.4,-4.2)`). And a third will print a date (`12/30/98`) to the screen. Each should display in its normal format (as indicated by the examples given).

Is this particular overloading a good idea? Why/Why not?

26) If a library consisted entirely of `inline` functions (and/or `const`ants and/or `typedef`initions), would it need an implementation file? Why/Why not?

27) When you place the keyword `const` in front of a formal argument's type, what does it do to that formal argument?

Why would a function designer place `const` on a formal argument that was being passed via the reference mechanism? (Hint: How does each alter the argument? Now combine them logically.) (Hint 2: Recall why we prefer to pass `string` arguments this way. . .)

28) Show the code for a function to generate a random integer in a range specified by the caller. Make sure the programmer can call your function with any of the following expressions from their code: `rand_range(min, max)`, `rand_range(low, 2*high+1)`, `rand_range(-6, 12)`. The caller will store or use your `return`ed value as they need/see fit. The formula to use is given — just fill in your function around it.

```
rand() % (top - bot + 1) + bot
```