Solutions to Quiz Section Problems

1. Given the following descriptions of problems (i.e., functions) that need to be solved, provide the appropriate function prototype (i.e., return type, name and formal parameter list):

```
/*
 * PrintTime
 * ---------
 * Given the number of seconds since midnight, this
 * function prints the time in the more standard hour,
 * minute, second format.
 */
void PrintTime(int secondsFromMidnight)

/*
 * RollOneNormalDie
 * ----------------
 * Produces the result of rolling a single six-sided, fair
 * die.
 */
int RollOneNormalDie(void)

/*
 * RollNNormalDice
 * ---------------
 * Produces the result of rolling N six-sided, fair dice.
 */
int RollNNormalDice(int n)

/*
 * ConfirmInput
 * ------------
 * Prompts the user to confirm the input of some value.
 * The return value should indicate whether or not the
 * user confirmed the input.
*/
int ConfirmInput(double input)

/*
 * DrawHangman
 * -----------
 * Draws a stick-figure, like the one in Hangman.  (The
 * details of how this might be done are irrelevant.)
 */
void DrawHangman(void)
```

2. Recall that one function can call another function. This approach is very common (in other words, get used to doing so). Assume that someone else has written the function `RollXSidedDie` (the prototype is below). Provide the implementation of `RollOneNormalDie` and `RollTwoNormalDice`. (You should also be able to write the implementation for `PrintTime` and `ConfirmInput`, although the solution will not be provided.)

```
int RollXSidedDie(int n) { ... }
int RollOneNormalDie(void) { return RollXSidedDie(6); }
int RollTwoNormalDice(void) {
  return RollOneNormalDie() + RollOneNormalDie();
}
```

Note that functions can be nested an arbitrary number of levels. In this case, `RollTwoNormalDice` calls `RollOneNormalDie`, which calls `RollXSidedDie` (which calls other functions). Do not be intimidated by multiple layers of nesting.

Aside: These short functions are called wrapper functions. Even though the code is very simple, they provide a way to describe more accurately what is being done. A good guideline for when to use a wrapper function is when the action (function) can easily be named. An example of a pointless wrapper function might be:

```
void PrintMyName(void) { printf(MY_NAME); }
```

In this case, the code says just as much as the function name.

3. What do each of the `printf`'s display?

```
int bar(int x, int y) {
  x = x + 1;
  return (2 * x - y);
}
int bar_square(int w) {
  return bar(w, w);
}
int main(void) {
  int w, x, y, z;
  w = 6;
  x = 7;
  y = 8;
  z = 9;
  printf("%d\n", bar(x, y));            /* 8   */
  printf("%d\n", bar(y, x));            /* 11  */
  printf("%d\n", bar_square(w));        /* 8   */
  z = bar(z, w);                        /* z=14 */
  printf("%d\n", bar_square(z));        /* 16  */
  printf("%d\n", bar_square(z, w));     /* !16! */
}
```

Note that the last line will compile (albeit with a warning). The code will even run (and display 16). Even so, your code should never contain any compilation warnings.

4. Discussion question: What does `void` mean? When do you use `void`? (What are the neo-political, para-psychological and quasi-religious ramifications of `void`?)

5. Are the two code fragments identical? Which is easier to read? Provide an example where else-if is easier to read. Provide an example where nested-ifs are easier to read.

```
if (a && b) {
  /* Code A */
} else if (!a && b) {
  /* Code B */
} else if (a && !b) {
  /* Code C */
} else {
  /* Code D */
}
```

```
if (a) {
  if (b) {
    /* Code A */
  } else {
    /* Code C */
  }
} else {
  if (b) {
    /* Code B */
  } else {
    /* Code D */
  }
}
```

The fragments are identical; which is easier to read depends on the person reading the code. In general, else-ifs are useful when testing a variety of conditions sequentially. Nested-ifs are most useful when the first decision (in this case a) influences how the second decision is made.

```
if (a) {
  /* Code A */
} else if (b) {
  /* Code B */
} else if (c) {
  /* Code C */
} else {
  /* Code D */
}
```

```
if (a) {
  if (b) {
    /* Code A */
  } else { /* !b */
    /* Code B */
  }
else { /* !a */
  if (c) {
    /* Code C */
  } else { /* !c */
    /* Code D */
  }
}
```

6. Draw a flowchart diagramming this code:

```
int booleanTest1(double x) { ... }
int booleanTest2(double x, double y) { ... }
int booleanTest3(int a, int b) { ... }
int main(void) {
  double a, b, c;
  int x, y, z;
  scanf("%lf%lf%lf", &a, &b, &c);
  x = booleanTest1(a);
  y = booleanTest2(b, c);
  z = booleanTest3 (x, y);
  if (x || z) {
    a = b - c;
    y = booleanTest2(b, a);
  }
  if (x && y) {
    printf("A");
  } else if (z) {
    printf("B");
  } else {
    printf("C");
    if (!x) {
      printf("D");
    else if (!y) {
      printf("E");
    }
  }
}
```