Practice Question Answers

Week 5

Answers

1. A do-while loop guarantees that the body of the loop is run at least once. A while loop offers no such guarantee. A do-while loop is often more useful for user input, because you want to grab input at least once. A while loop is cleaner in most other cases. For instance, if you wanted to implement a "forever" loop, you'd generally do:

   ```
   while(TRUE) { }
   ```

   instead of a do-while. This is simply for clarity while reading.

2. FOR LOOP

   ```
   int sum = 0;
   for(int i = 1; i <= 10; i++)
      sum += i;
   ```

   WHILE LOOP

   ```
   int sum = 0, i = 1;
   while(i <= 10) {
       sum += i;
       i++;
   }
   ```

3. $12_{10} : 1100_2$
   $33_{10} : 100001_2$
   $47_{10} : 101111_2$

   $1010110_2 : 86_{10}$

4. Hexadecimal is another word for base 16. It is related to binary because a group of 4 binary digits can be represented in 1 hexadecimal digit. This is because $2^4 = 16$. Thus, for example,
   $1101_2 \equiv 13_{10} \equiv \mathrm{D}_{16}$

5. 'a' $\rightarrow$ 97
   'A' $\rightarrow$ 65

6. Typecasting temporarily "transforms" a variable of one type into another. That is, the value of `x` after:

   ```
   double y = 90.00;
   char x = (char)(int)y;
   ```

   is `'Z'`, as expected.

7. I'll let you do this one. You may want to pay attention to the shapes of the various pieces because different types have different shapes. Just open Scratch and play around, though!

8. 8

9. A variable is a named storage location created through a declaration. (Eg., `int i;`) Once we've defined a variable, we can then store values in it and change its value as we want.

10. Two's complement is the way numbers are generally represented in memory. To change the sign of a number in two's complement form we flip the bits and add 1. This representation is a good one because because arithmetic operations are easier for the hardware to do from two's complement's representation. By "easier", we mean that you can add positive and negative numbers in the usual way, whereas sign-magnitude and one's complement require special attention.

11. `gcc example.c -lcs50`, `gcc -o example1 example.c -lcs50`

12. `#include <stdio.h>` makes the funtions in stdio's library available to us to use in our program. We generally bring stdio.h in so that we can use printf. There's lots of other interesting functions in stdio.h: http://en.wikipedia.org/wiki/Stdio.h , but the small details are beyond the scope of the quiz.

13. Header files enclosed in angle brackets describe system libraries. Header files enclosed in quotation marks are user-created and are generally located in the same directory as the C file that references them.

14.
    - `argv` should be `char`, not `int`
    - should be `GetInt()`, not `getInt()`
    - should be `%d` instead of `%s`
    - should be `x % y` instead of `x / y`
    - would need to `#include <cs50.h> and <stdio.h>`, but this isn't something we were looking for on this one

15. 1000

16. The lack of a `break` statement in any of the `case`s causes them to "drop-through", and thus everyone in the class who got above a 60 gets a D, and everyone else gets an F.

17. This prints A through Z. This has a counter, c, and it just iterates through the ascii chart to print A, then B, then C, etc. It stops after it prints Z.

18.
```
int i,j;

for(i = 1; i <= 10; i++) {
    for(j = 1; j <= 10; j++) {
        printf("%d\t", i * j);
    }
    printf("\n");
}
```

19. The values of a and b will not change, because swap will only change the values of its own local variables (swap does not have access to change a and b themselves!). One could perform a swap by making a and b global variables, or by using pointers.

20. `pow` will be on the top of the stack, `domath` will be in the middle of the stack, and `main` will be waiting at the bottom of the stack for the other functions to finish.

21. They help eliminate magic numbers. Not using magic numbers is useful for:

    - code readability
    - updating your code–if you want to change a constant, changing magic numbers throughout the code is much more difficult than changing the `#define` in one place

22. We use `#define` as a macro to replace some "placeholder" (e.g. `SENTINEL`) with a constant of our choice (e.g. `-1`). We use `#include` to "paste in" the contents of a file so we don't have to do the copy and pasting ourselves.

23. 13, 14

24.
```
string s = "hello";
int i, len;
for(i = 0, len = strlen(s); i < len; i++)
{
    printf("%c\n", s[i]);
}
```

25. `argv[1]` is `"pick"`, so `argv[1][1]` is `'i'`.

26.
```
int one_D[3] = {1,2,3};
int two_D[2][3] = {{1,2,3},{4,5,6}};
int three_D[2][2][2] = {{{1,2},{3,4}},{{5,6},{7,8}}};
```

    You can also use nested for loops to initialized them. For a 2-dimensional array, you'd nest the loops 2-deep. For a 3-dimensional array, you'd nest the loops 3-deep, etc.

27. (a)
```
int i,j;
for(i = 0; i < 5; i++) {
    for(j = 0; j < 3; j++) {
        printf("%d\t", numbers[i][j]);
    }
}
```

    (b)
```
int i,j;
for(j = 0; j < 3; j++) {
    for(i = 0; i < 5; i++) {
        printf("%d\t", numbers[i][j]);
    }
}
```

28.
```
bool ascending_order(int array[], int size) {
    int i;

    // Note that i starts at 1, instead of 0.  Think about why!
    for(int i = 1; i < size; i++) {
        if(array[i] < array[i-1])
            return false;
    }

    return true;
}
```

29.
```
bool StringEqual(string s1, string s2) {
    int len1 = strlen(s1);
    int len2 = strlen(s2);

    // strings of different lengths cannot be equal
    if(len1 != len2) {
        return false;
    }

    for(int i = 0; i < len1; i++){
        if(s1[i] != s2[i])
            return false;
    }

    return true;
}
```

30. (c) 26^n

31. This program will print out a series of numbers. The numbers depend on the number of arguments passed to the program. It's best to show through some examples:

```
> a.out arg1 arg2 arg3
4 3 2 1 0

> a.out
1 0

> a.out arg1 arg2 arg3 arg4 arg5
6 5 4 3 2 1 0
```

This is a slightly more advanced question, so it is probably not very important that you understand all the details about what's going on here, but if you have questions, don't hesitate to ask.

32. It will print out the value of `k`, which is 3.

33. `double *myDoubles = malloc(8 * sizeof(double));`

34. This is called a memory leak, which if persistent will eventually sap all of your system resources. Another major error that we can make, besides neglecting to `free` memory, is to `free` it twice.

35. Any $O(n^2)$ sorting algorithm is acceptable.

36. Don't let the quirky variable names and pointers throw you off. This simply prints out `<input string> <reversed input string>`.