Solutions to Practice Problems

Practice Problem 7.1
What feature of the C language makes a buffer overflow attack possible?

Solution: C programs do not automatically check to make sure they are writing to memory that actually belongs to an array. C programs blindly go to the start of the array, multiply the requested index by the array data type of the array, and access that memory, whether or not it is beyond the bounds of the array.

Practice Problem 7.2
Describe the mechanism by which a segmentation fault occurs.

Solution: A variable on the stack is written to so far beyond its allocated boundaries that it is able to overwrite the saved (prior) value of ebp and/or the return address, thus causing unsuccessful attempts to access main memory addresses that are assigned to other programs.

The OS will not allow a program to access the memory of another program, and will crash the errant program with a segmentation fault.

Practice Problem 7.3
For the pawn function below, is it possible to overwrite the value you will get for your item with an amount of your choosing by overwriting the value variable on the stack during the scanf( ) call below? Explain.

```c
void pawn()
{
    char item[12];
    int value = 100;
    printf("What have you come to sell? ");
    scanf("%s", item);
}

int main()
{
    pawn();
}
```

Solution: Not possible. The array named item is placed on the stack first, and then the variable named value is placed on the stack next (above the array). Since value is higher up on the stack than item, it cannot be overwritten by going beyond the boundary of the array named item.
### Practice Problem 7.4

When the `echo_string` function is called in `main` from the following code sample, the stack pictured below is created.

```c
#include<stdio.h>
void echo_string()
{
    int count;
    char entered_string[10];
    printf("Enter a string: ");
    scanf("%s", entered_string);
    for(count=0; count < 10; count=count+1)
    {
        printf("%s\n",entered_string);
    }
}
int main()
{
    echo_string();
}
```

Assuming there is no padding (extra spaces) when the frame is created. How many characters can be entered before the return address is overwritten?

Solution:
- 10 characters consumes 10 bytes
- 1 integer uses 4 bytes
- 4 bytes are needed to store ebp

Thus, 18 bytes appear before the return address. Thus **17 characters** can be entered (remember—the 18th character, the NULL is automatically appended!) before we start overwriting the space that the return address is saved in.

### Practice Problem 7.5

For the following program invocation:  

```
midshipman@EC310 ~$ ./a.out wait 8 mate
```

A) What is the value of `argc`?

B) What is the value of `argv[1]`?

C) What is the data type of `argv[2]`?

Solution:  
(a) 4  
(b) "wait"  
(c) string

### Practice Problem 7.6

Pertaining to taking in command line arguments for a program, choose the best description for `argc`.

- (A) holds the number of command line arguments excluding the program name.
- (B) holds the total number of command line arguments available to the program.
- (C) holds the number of integer variables entered at the command line before the program begins.
- (D) None of the above.

Solution:  
B
Practice Problem 7.7

In the following sentence, circle the correct choices.

 argv is a(n) array / index / variable used to store each command line parameter / index / argument in a binary / string / numeric format.

Solution:

 argv is a(n) array / index / stack used to store each command line parameter / index / argument in a binary / string / numeric format.