NICK Alvarez

CS 326 Homework 6

1. a) It is possible the program never clears the stack after each call. So, i, while only existing in foo(), never "ceases to exist" from a memory standpoint. Thus, each call heads to the same location in the stack, incrementing i.

b) Now, let us add another function.

```
void foo() {...}
void foo2() {
    int i;
    i=1;
} int main() {
    int j;
    for (j=1; j<=10; j++) {
        foo();
        foo2();
    }
}
```

The output is 0 1 1 1 ... and from this we can see that foo() runs, gives us zero, and then increments i. foo2() runs, sets i to one, then foo() prints 1. Because of the post increment operator, foo2() will keep setting i to one.

2. A "workaround" is to have an array of factorial values, test what number is used in the macro, and print the corresponding value. i.e. FACTORIAL(6) prints 720. But there is no calculation there. In a traditional way of calculating factorials, there is no way to write a macro to do that. No recursion is available as a base case could not be defined.

3.a) Passing by value only copies the values of X and Y. The swap will be successful but the new values are in the swap() scope and will die when you exit the subroutine.

b) Passing by name solves the first issue of values dying in the scope. However, let's break down the code.

```
void swap(X, Y) {
    int temp = 0;
    temp = X;        temp = X (name, not value)
    X = Y;           X = Y (now X is Y itself)
    Y = temp;        Y = temp (temp was set to X,
                     which is set to Y,
                     so Y = Y)
}
```

4. a) Value: 1 10 11

No modifications are made to original variables in f().

b) Reference: 3 2 11

Original variables are modified as their address is passed in, including during function execution.

```
x = x+1     x=1+1 → x=2=z      z=x
y = z       y=z → y=2          i=3
z = z+1     z=x+1 (or z) → 2+1=3   a[1]=2, a[2]=11 (untouched)
```

c) Value-result: 2 1 11

First, i is returned from x+1, so 2.

a[1] = z → 1, so a[1]=1. a[2] isn't touched.

d. Name: 2 10 3

```
x = x+1     x=2 → 2
y = z       y=z=i=x=2 (a[i])
z = z+1     z=3
```

x is set to 2. a[2] = z, and z = 3, so a[2]=3 and a[1] isn't touched.

5. It will not run faster. The parameter is either assigned a) a default value or b) passed value. Assignment happens either way.