**Question 1:**

The else is paired with the inner if statement. The following code shows how I tested it:

#include <stdio.h>

int main (){

int a, b;

scanf("%d %d", &a, &b);

if (a != b)

if (a < b)

printf("inner if");

else

printf("made it to else");

printf("skipped else");

}

I compiled the code and gave it various inputs and tested all branches (Branch Coverage testing)

**Question 2:**

First I tried to do it in the following method:

#include <stdio.h>

int main(){

int array[2];

static int a = 2;

array[3] = 4;

printf("%d", a);

}

But I was getting warnings from the compiler about accessing memory I do not own:

array index 3 is past the end of the array (which contains

2 elements) [-Warray-bounds]

array[3] = 4;

And When I try to run it I get “Abort Trap: 6” and it fails. I tired using global, or static variables but it did not work either. So I chose to do it using a pointer that points to the memory address right after the last array index. Here is the code:

#include <stdio.h>

int main (){

// Define the Array

int array[] = {0,1,2,3,4};

printf("%p\n\r", &array[4] + 0x1);

// Define a pointer to the memory address

int \*next\_address = (int\*)(&array[4] + 0x1);

printf("next\_address = %d at address %p\n\r", \*next\_address, &(\*next\_address));

// Modify the elements of the array

for (int i=0; i<6; i++) {

printf("Index %d: Value of %d at address %p\n\r", i, array[i], &array[i]);

array[i] = i\*100;

printf("Index %d: Value of %d at address %p\n\r", i, array[i], &array[i]);

}

// Print the value the pointer is pointing to, to verify it was changed implicitly

printf("next\_address = %d at address %p\n\r", \*next\_address, &(\*next\_address));

}

The compiler compiles the code and executes it with no indication that something was wrong. After a lot of tweaking I was able to demonstrate the following oddity: The value of next\_address was changed when I updated the values of the array, despite never actually updating the value of next\_address. Here is the output from terminal:

next\_address = 32767 at address 0x7fff5ad81b44

Index 0: Value of 0 at address 0x7fff5ad81b30

Index 0: Value of 0 at address 0x7fff5ad81b30

Index 1: Value of 1 at address 0x7fff5ad81b34

Index 1: Value of 100 at address 0x7fff5ad81b34

Index 2: Value of 2 at address 0x7fff5ad81b38

Index 2: Value of 200 at address 0x7fff5ad81b38

Index 3: Value of 3 at address 0x7fff5ad81b3c

Index 3: Value of 300 at address 0x7fff5ad81b3c

Index 4: Value of 4 at address 0x7fff5ad81b40

Index 4: Value of 400 at address 0x7fff5ad81b40

Index 5: Value of 32767 at address 0x7fff5ad81b44

Index 5: Value of 500 at address 0x7fff5ad81b44

next\_address = 500 at address 0x7fff5ad81b44

**Question 3:**

For the following problem, I am using the gcc compiler on a Mac OS.

**Part a)** I wrote a brief function called int multiplyTwoNums(int first, int second).

This function takes two arguments, both of them being of type int.

The first test was to input only one parameter. The compiler gave the following error:

hw2.c:9:19: error: too few arguments to function call, expected 2, have 1

multiplyTwoNums(a);

The second test was to give the function a wrong parameter. I passed in a float instead of an int, and got no error.

Here is the code:

#include <stdio.h>

int multiplyTwoNums(int first, int second){

return first\*second;

}

int main (){

int a = 2, b = 3;

char f = 'a';

multiplyTwoNums(a,f);

}

**Part b)**

--First Test Case: Omitting the ‘&’ sign

Code:

#include <stdio.h>

int main(){

int i;

scanf(“%d”, i);

return 0;

}

Compiler Output:

**test.c:4:13: warning: format specifies type 'int \*' but the argument has type**

**'int' [-Wformat]**

scanf("%d", i);

So the compiler gave out a warning but when I ran the code and entered 2, it spit out a segmentation fault. So since the value of I is an undefined address in memory. The compiler cannot access it and it throws out a Segmentation fault.

-- Second Test Case: discrepancies in number of arguments

Code:

#include <stdio.h>

int main(){

int a = 3, b = 5, c = 0;

printf("%d, %d", a, b, c);

return 0;

}

Compiler Output:

**hw2.c:35:25: warning: data argument not used by format string**

**[-Wformat-extra-args]**

printf("%d, %d", a, b, c);

So the compiler indicated that something was wrong. When I ran the file, it printed the values of a, and b. However it did nothing for c. This is because the format string did not specify that there was a third parameter so the compiler did not even worry about it.

--Third Test Case: Discrepancies in type of data arguments

Code:

#include <stdio.h>

int main(){

float z = 3.0;

int a = 3;

printf("%d, %d", a, z);

return 0;

}

Compiler Output:

**hw2.c:41:22: warning: format specifies type 'int' but the argument has type**

**'float' [-Wformat]**

printf("%d, %d", a, z);

So the compiler indicated that there was something wrong. But when I ran it, it printed: 3, 1451621264. So for the float value it printed a different number instead of 3.0. This is because C automatically converts float to double in function calls. So when we specify %d, we are telling the compiler to read 4 bytes (size of int) out of the stack rather than 8 bytes (size of float). So it reads an unexpected value for z and prints out a random number.

**Question 4:**

AX: 0x1514

BX: 0x1502

CX: 0x0100

DX: 0x2524