Xiana Lara
September 18, 2020
Programming #3
CS 471

Programming #3 -- digging into the runtime stack

The first printf() is skipped because its address in memory is changed. Essentially, when it is called it is looking in the wrong spot. This is because of the A[6] = A[6] + 10, it moves the pointer away from the print statement. However, this must be done in a positive multiple of 10 so as not receive an error. If you have a negative 10 an infinite loop is called, however if you have a positive 20 it then skips the second print statement also.

When adding variables to function f() runtime errors were caused because of spacing issues. So, when the first variable 'a' is added there is no errors because there is room still in the stack. However, when the second variable is added I received a segmentation fault error because an address takes up 64 bits, so with one instruction it just has 32 bits and there is room for 32 more. So when filling it I need to change A[6] to A[8] so as not receive an error.

```c
/*
Xiana Lara
September 18, 2020
Programming 3: Digging into the Runtime Stack

Original Purpose
Program to demonstrate how to overwrite a return address
inside of function using a global variable to store
the address we want, and we will use an array in the
function to seek the location and replace with the new value

Current Purpose
Program to visualize how addresses are places in the stack
and how memory is allocated in the stack

Input: none
Output: Addresses and values currently in the stack

Originally written by:
Shaun Cooper

2020 September
*/

#include <stdio.h>

// dummy function which makes one important change

void f() {
```

Xiana Lara
September 18, 2020
Programming #3
CS 471

```
    unsigned int *A;
    int i;

    // Variables added to find errors in code
    int a;
    int b;
    int c;
    int d;

    // Values for new variables
    a = 50;
    b = 51;
    c = 52;
    d = 53;

    A = (unsigned int *) &A;

    for (i=0;i<=10; i++)
       printf("%d %u\n",i,A[i]);

    // orginally: A[6] = A[6] + 10
    // was changed to conducts tests
    A[10] = A[10] + 10;
    printf("A is %u \n",A);

   for (i=-4;i<=10; i++)
    printf("%d %u\n",i,A[i]);

} // of f()

int main(){

   int A[100];
   unsigned int L[4];
   L[0]=100;
   L[1]=200;
   L[2]=300;
   L[3]=400;
    for (int i=0; i < 100; i++) A[i]=i;

   printf("main is at %lu \n",main);

   printf("f is at %lu \n",f);
   printf("I am about to call f\n");

   // calling f()
   f();

   printf("I called f\n");

out: printf(" I am here\n");

} // of main
```