Programming #3 -- digging into the runtime stack

Submit Assignment

Due Friday by 11:59pm **Points** 25 **Submitting** a file upload **File Types** pdf

Learning about the runtime stack is helpful in understanding how a compiled language utilizes the RAM (memory). As part of being a Computer Scientist, we need to "experiment" with how languages are implemented to give us deeper insight on how compilers and hardware are inter-related. You

You are to take the below program and compile and run it on a CS computer. You are to explain why, on the return of f(), the first printf() is skipped.

You are to extend the program by incrementally adding single variables to the function f(). As you do, the system will have a runtime error. Explain why. You are then to make changes to f() so that the first printf() is jumped over once again.

```
/* Program to demonstrate how to over write the
* return address inside of function
* we will use a global variable to store
* the address we want to go to on return
* and we will use an array in the function to
* seek the location and replace with the new value


Shaun Cooper

2020 September
*/

#include <stdio.h>

// dummy function which makes one important change

void f() {


    unsigned int *A;
    int i;

   A =(unsigned int *) &A;

    for (i=0;i<=10; i++)
       printf("%d %u\n",i,A[i]);
   A[6]=A[6]+10;
   printf("A is %u \n",A);

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

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");
   f();
   printf("I called f\n");

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

}
```