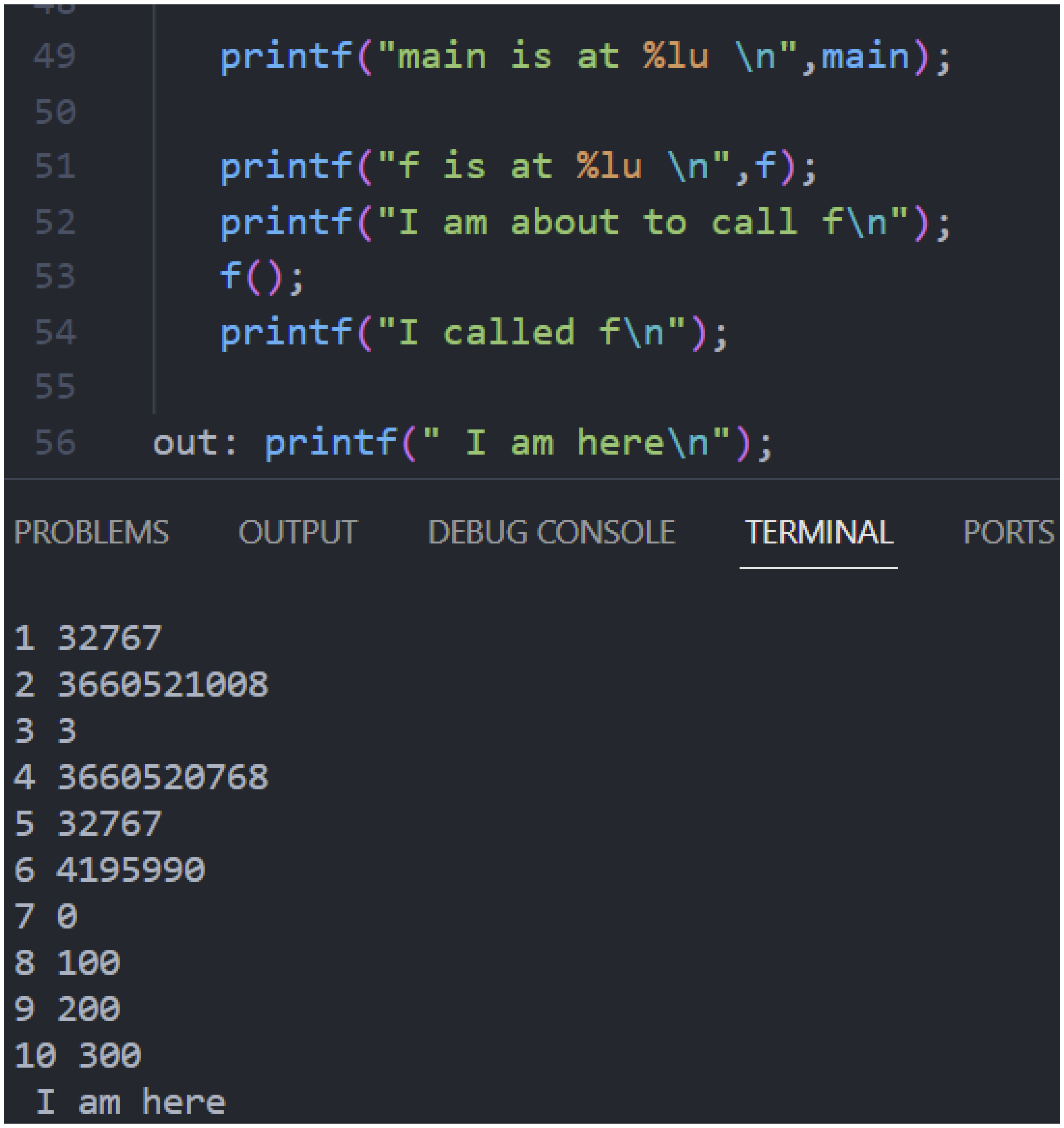
Joey Troyer

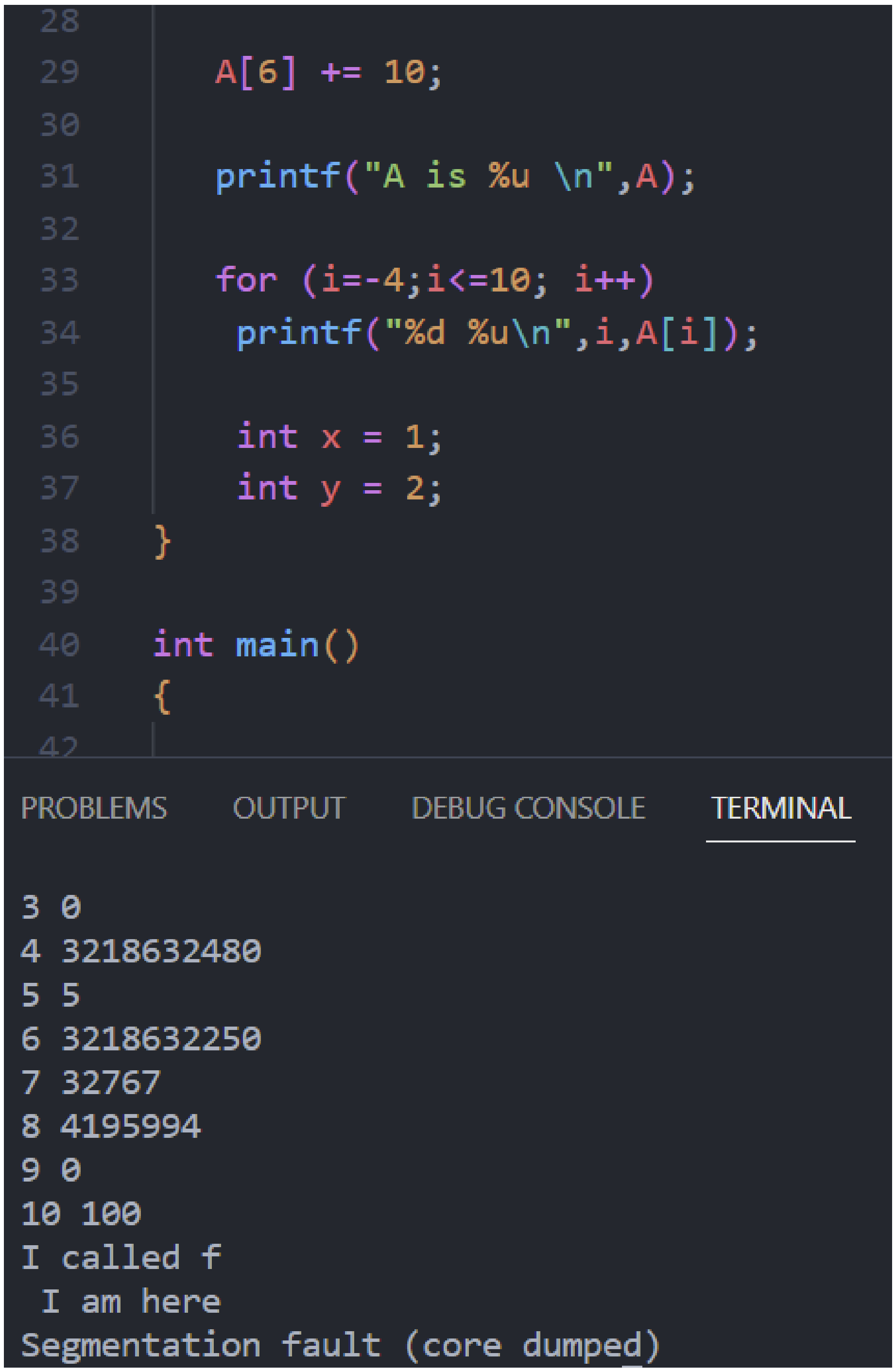
The first printf upon return is skipped over because the function f() set a variable A to the address of itself. This allows us to access the stack, and we are able to find the return address at a[6]. We can add 10 to A[6], which in this case is the number of bytes the printf is, so when we return, we skipped over it.



```
    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");
```

```
1 32767
2 3660521008
3 3
4 3660520768
5 32767
6 4195990
7 0
8 100
9 200
10 300
 I am here
```

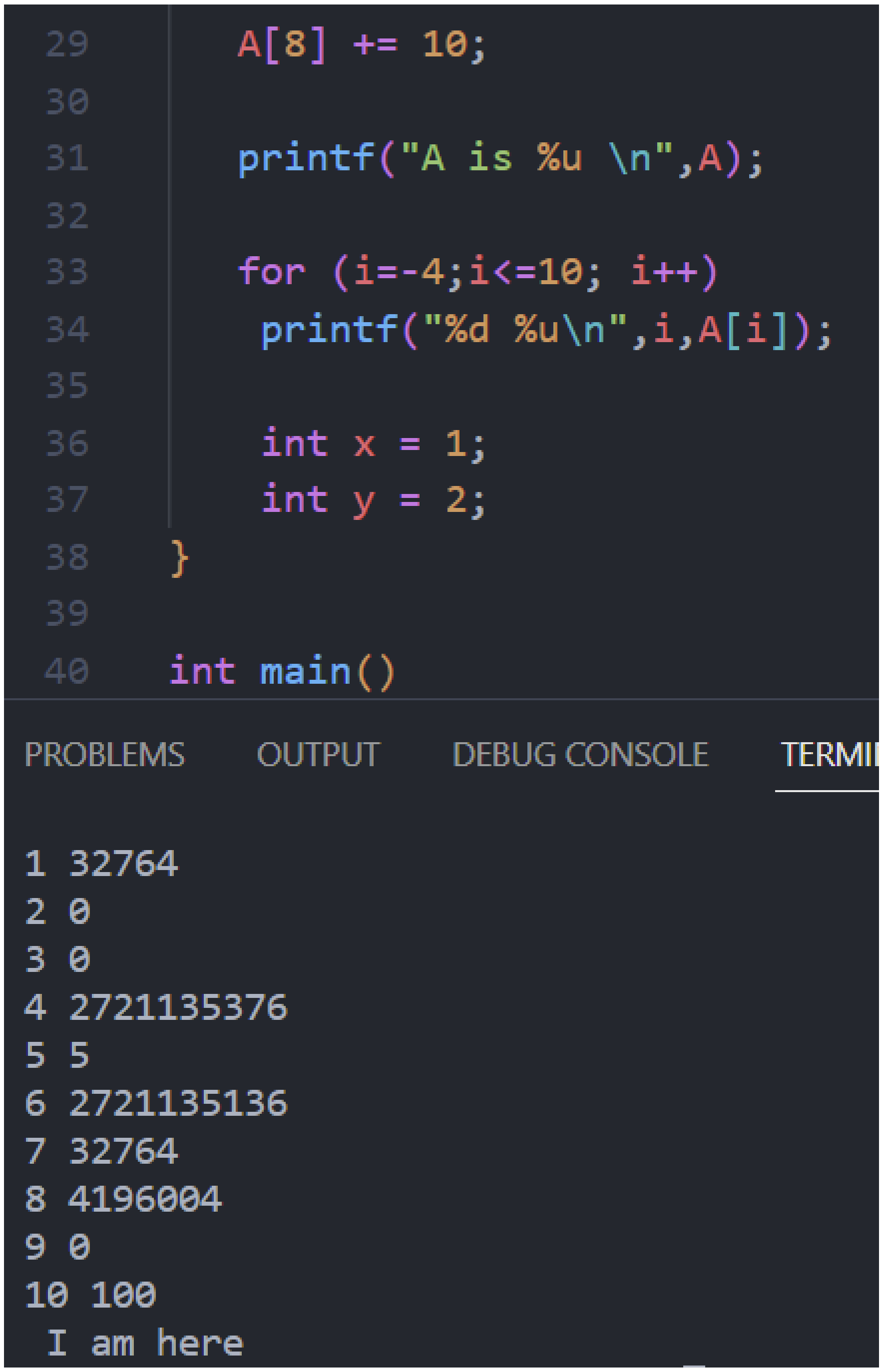
Adding two variables to the end of f() causes a segmentation fault. This is because A[6] is no longer the return address. Notice that now it prints “I called f” the printf that was previously skipped. The reason it seg faults is because we are changing memory on the stack above what memory the os has dedicated for this program. I am not sure exactly what we are changing, but it is an illegal part of memory that we should not access.



```c

    A[6] += 10;

    printf("A is %u \n",A);

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

     int x = 1;
     int y = 2;
}

int main()
{
```

```
3 0
4 3218632480
5 5
6 3218632250
7 32767
8 4195994
9 0
10 100
I called f
 I am here
Segmentation fault (core dumped)
```

We can skip over the first printf again while the two variables are still there by increasing the index of A by 2. So instead of adding ten to a[6] we add ten to a[8], and it skips the first printf again.

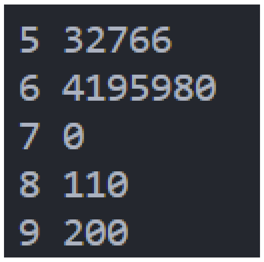


```
    A[8] += 10;

    printf("A is %u \n",A);

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

     int x = 1;
     int y = 2;
}

int main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMI

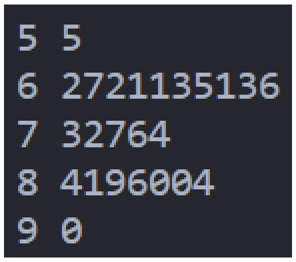
```
1 32764
2 0
3 0
4 2721135376
5 5
6 2721135136
7 32764
8 4196004
9 0
10 100
 I am here
```

We can even look at the values of A[6] and A[8] on the terminal and see the return address move. We already know that the return address is at index six, so here is the output without the two variables in the function. We can see that index 6 is around 4 million



```
5 32766
6 4195980
7 0
8 110
9 200
```

Now, if we add those two variables to f() and look at the output. The number that is really close to the value of index 6 in the picture above is now in index 8.



```
5 5
6 2721135136
7 32764
8 4196004
9 0
```