Risk Example and srand() function

Of the functions I listed in the previous lecture notes, all of them have a return value, EXCEPT for one function. This function, srand, has a return type of void, which means it doesn't return anything.

As we have seen from last lecture, any function that returns a value needs to be called as part of a statement where a value of that type would be valid.

So, what are the rules for calling a void function?

Nearly always, call a void function on a line by itself.

Since a void function doesn't return any value (instead it completes some specified task and has no need to actually return any value to its caller,) nothing that it returns needs to be stored or used. Thus, by calling a void function on a line by itself, that tells the function to execute its task. When it finishes, the calling function can simply continue to the next line of code.

The srand function takes in an integer. In order to get some variety with the random numbers generated, we must pass to this function a different value whenever we call it. The easiest way to create a different value each time we run a program is to call the time function which returns to us the number of seconds after January 1, 1970, the approximate birth of the unix operating system. We need to pass to the time function the value 0 for this to occur. Putting this all together, here is a valid call to seed the random number generator:

```
srand(time(0));
```

In order to use the srand function, we must include stdlib.h. In order to use the time function, we must include time.h.

In the example that follows, we will simulate a single battle in the game of risk between 2 attacking armies and 2 defending armies. The rules are as follows:

Each team rolls two dice. Compare the highest rolls of the two teams. If the attacker's value is higher, then they defeat the corresponding defending army. Now compare the lowest rolls of the two teams. Once again, if the attacker's value is higher here, they they defeat the corresponding defending army. In such an encounter, there are three outcomes:

1) Defender loses two armies.
2) Each team loses one army.
3) Attacker loses two armies.

Here is a short program to simulate a single battle and print the outcome to the screen:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {

     int a_roll1, a_roll2, d_roll1, d_roll2, temp, sum=0;

     srand(time(0));
     a_roll1 = 1+rand()%6;
     a_roll2 = 1+rand()%6;
     d_roll1 = 1+rand()%6;
     d_roll2 = 1+rand()%6;

     if (a_roll1 < a_roll2) {
          temp = a_roll1;
          a_roll1 = a_roll2;
          a_roll2 = temp;
     }

     if (d_roll1 < d_roll2) {
          temp = d_roll1;
          d_roll1 = d_roll2;
          d_roll2 = temp;
     }

    if (a_roll1 > d_roll1) sum++;
    if (a_roll2 > d_roll2) sum++;

    printf("The attackers killed %d armies.\n", sum);

    return 0;
}
```