**Recitation 13**

**Topics**

* recursion

**Recitation Instructions**

* Solve the tasks below. They should all be fairly straight forward.
* Once you have tested your code and are confident that it is working correctly, *then run each of your functions through the debugger* for your system and examine the call stack as you go.  
  Usually we use the debugger to [duh] debug our programs. Here you are using it to understand the way your code works. The key to understanding recursion is to understand recursion... No, no. I meant to say the key is understanding how the call stack represents each step of the recursion.

Ok, here are the things to do today. Hopefully by the time you are done, you will be an ace at recursion.

1. Write a function to display an integer in binary. Trace its execution in the debugger. Show the lab worker your code and let them watch your trace.
2. Write a function to create and return a new list that is the sum of the values in the the two lists passed in. You may assume that the two lists are the same length. Show the lab worker your code and let them watch your trace.
3. Write a function to return the maximum of the values in a binary tree. Use the attached code. Note, there is a constant INT\_MIN defined in the include file <climits> that is the value of the smallest possible int. This could make your life easier.
4. Write a function to return the int "sum" of the characters in a c-string. [see below for c-string issues] Show the lab worker your code and let them watch your trace.
5. Implement the binary search algorithm for an array of characters. Trace through the execution of the function on an array of 14 or more characters. [Remember that the array *must*be sorted for this search algorithm to work.] Experiment with using different targets that will cause the function to search through different portions of the array, including not being found - at the front, at the back, in the middle and so on. Show the lab worker your code. They will ask you to predict the output of your program (without running it!). They will then watch you trace your code in the debugger and will check your prediction.
6. Trace the execution of the following function with various inputs. Show the lab worker your code. They will ask you to predict the output of your program (without running it!). They will then watch you trace your code in the debugger and will check your prediction.

void f(int n) {

if (n > 1) {

cout << 'a';

f(n/2);

cout << 'b';

f(n/2);

}

cout << 'c';

}

c-string issues

* Recall that a *c-string* is an array of characters (for example: char cstring[15] = "I love C++"; ) with the null character ('\0') used to indicate the end of the *cstring* (the array cstring can hold a cstring that is up to 14 characters long. As defined here it contains the 10 characters 'I', ' ', 'l', 'o', 'v', 'e', ' ', 'C', '+' and '+' and the null character follows the second '+'character.)
* Recall also that every character (printable and non-printable), in C++, is stored as a one byte value: its ASCII value. This is a number between 0 and 255. You can add these integer values just as you can add int values:

char a('A'), b;

// a contains 'A' whose ASCII value is 65

cout << "a contains '" << a

<< "' whose ASCII value is " << int(a) << endl; // note the cast to int on this line

b = a + 1;

// b contains 'B' whose ASCII value is 66

cout << "b contains '" << b

<< "' whose ASCII value is " << int(b) << endl; // note the cast to int on this line

char c( 'a' - 17 + '\n' ); // not sure what number will be stored or what character it represents but let's see

cout << "b contains '" << b

<< "' whose ASCII value is " << int(b) << endl; // note the cast to int on this line