1. (20 Points) The Lucas function $L(n)$ is defined for all $n \geq 1$ by the recurrence

$$L(n) = \begin{cases} 
1 & \text{if } n = 0 \\
3 & \text{if } n = 1 \\
L(n-1) + L(n-2) & \text{if } n \geq 2
\end{cases}$$

a. (10 Points) Write a recursive C function with the following heading that calculates and returns $L(n)$.

```c
int L(int n) {
    // your code goes here
    if (n == 0) {
        return 1;
    } else if (n == 1) {
        return 3;
    } else {
        return L(n-1) + L(n-2);
    }
}
```

b. (10 Points) Perform a box trace of the function call $L(5)$. Each box represents one invocation of function $L(\cdot)$, and should be labeled with the value of $n$ for that invocation. Each connection joining a box to a descendant box should be labeled with the value being returned by the descendant box. What integer is returned at the top level?

**Solution:** $L(5) = 18$
2. (20 Points) Write a recursive C function that takes two non-negative integers \( n \) and \( m \) as input, then returns the sum of the integers from \( n \) to \( m \) (inclusive) if \( n \leq m \), and returns 0 if \( n > m \). Do this in two ways as described below.

   a. (10 Points) Determine the sum of integers from \( n \) to \( m - 1 \) recursively, then add \( m \) to the result. Call this function \texttt{sum1()} and fill in the code details below.

   ```c
   int sum1(int n, int m)
   {
     // your code starts here

     if ( n>m ){
       return 0;
     }else{
       return sum1(n, m-1) + m;
     }
   }  // your code ends here
   ```

   b. (10 Points) Split the sequence of integers from \( n \) to \( m \) (roughly) in half, recur on the two half-sequences, then add the results. Call this function \texttt{sum2()} and fill in the code details below. Hint: model this function on \texttt{mergeSort()}.  

   ```c
   int sum2(int n, int m)
   {
     // your code starts here

     if ( n>m ){
       return 0;
     }else if( n==m ){
       return n;
     }else{
       int k = (n+m)/2;
       return sum2(n, k) + sum2(k+1, m);
     }
   }  // your code ends here
   ```
3. (20 Points) Consider the following C program.

```c
#include<stdio.h>
#include<stdlib.h>

int main(void){
    int i;
    double x = 2.3;
    double y = 0.0;
    double* p = &x;
    double* q = malloc(sizeof(double));
    double** pp = &q;
    double B[] = {1.2, 3.1, 5.3};

    for(i=0; i<3; i++) y += *(B+i);
    *q = *p + y;
    **pp *= 2;
    pp = &p;

    printf("%.1f, %.1f, %.1f\n", *p, *q, **pp);
    p = &y;
    printf("%.1f, %.1f, %.1f\n", *p, *q, **pp);
    q = B+2;
    printf("%.1f, %.1f, %.1f\n", *p, *q, **pp);

    return(EXIT_SUCCESS);
}
```

a. (6 Points) Write the output of this program exactly as it would appear on the screen:

**Program output:**

2.3, 23.8, 2.3
9.6, 23.8, 9.6
9.6, 5.3, 9.6

b. (8 Points) List the four pointer variables in this program, and for each one, state whether it points to stack memory or heap memory. If a pointer changes from stack to heap or heap to stack, make note of the point in the program where the change occurs.

**Solution:**
The variables \( p \), \( q \) and \( B \) are pointer-to-double, and \( pp \) is pointer-to-pointer-to-double.
The variables \( p \), \( B \) and \( pp \) point to stack memory.
The variable \( q \) points to heap memory until the assignment \( q = B + 2 \), which points it to stack.

c. (6 Points) Does this program contain any memory leaks? If so, what alteration(s) would be needed to eliminate the leak(s)?

**Solution:** The program contains one memory leak. Do `free(q)` immediately before the assignment \( q = B + 2 \) to eliminate the leak.
4. (20 Points) Write a C function called `mix()` with the heading given below, that takes as input two strings \( A \) and \( B \) (NUL '\0' terminated char arrays), copies the contents of \( A \) and \( B \) into the array \( C \) by alternating one character from \( A \) and one from \( B \) (starting with \( A \)), then adds the NUL character '\0' to \( C \), making it a valid string. Note that \( A \) and \( B \) may have different lengths. If one string is exhausted before the other, copy the remaining characters into \( C \) in order. Thus if \( A \) is the string "abcdef" and \( B \) is the string "xyz", then \( C \) will contain the string "axbyczdef". You may assume that \( C \) has sufficient space to hold all characters in \( A \), and in \( B \), and the NUL '\0' terminator.

**Solution:**
Note that there are a number of valid solutions to this problem, one of which is presented here.

```c
void mix(char A[], char B[], char C[]){
    // your code begins here
    int i = 0, j = 0, k = 0;
    int n = strlen(A), m = strlen(B);

    while( i<n && j<m ){
        if( k%2==0 ){
            C[k] = A[i];
            i++;
        }else{
            C[k] = B[j];
            j++;
        }
        k++;
    }
    while( i<n ){
        C[k] = A[i];
        i++;
        k++;
    }
    while( j<m ){
        C[k] = B[j];
        j++;
        k++;
    }
    C[k] = '\0';

    // your code ends here
}
```
5. (20 Points) Write a recursive C function called `printNumber()`, with the heading below, that prints out the base $b$ digits of the integer $n$ from most to least significant (left to right). Assume that the base $b$ is in the range $2 \leq b \leq 10$. You may also assume that $n$ is a positive integer, and in particular, your function need not print anything when $n$ is zero.

```c
#include<stdio.h>

void printNumber(int n, int b) {
    // your code starts here

    if( n>0 ){
        printNumber(n/b, b);
        printf("%d", n%b);
    }

    // your code ends here
}
```