CMPS 12B
Introduction to Data Structures
Summer 2019

Midterm Exam 1 Solutions

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 \( \text{sum1()} \) 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 \( \text{sum2()} \) and fill in the code details below. Hint: model this function on \( \text{mergeSort()} \).

```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, j;
    double x = 7.2, y;
    double * A = calloc(4, sizeof(double));
    double B[] = {9.8, 7.6, 5.4, 3.2};
    double *p, *q;

    p = malloc(sizeof(double));
    y = x+2;
    q = &y;
    *p = *q + 2.6;

    for(i=0; i<4; i++){
        j = 3-i;
        *(A+i) = B[j] + j;
    }
    printf("%.1f, %.1f, %.1f, %.1f\n", *A, *B, *p, *q);
    p = &x;
    printf("%.1f, %.1f, %.1f, %.1f\n", *A, *(A+1), *(A+2), *(A+3) );
    A = B;
    printf("%.1f, %.1f, %.1f, %.1f\n", *A, *(A+1), *(A+2), *(A+3) );
    return(EXIT_SUCCESS);
}
```

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

Program output:

```
6.2, 9.8, 11.8, 9.2
6.2, 7.4, 8.6, 9.8
9.8, 7.6, 5.4, 3.2
```

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 A, B, p, and q are all of type pointer-to-double.  
B and q point to stack memory.  
A points to heap memory until the assignment A = B, which points it to stack memory.  
p points to heap memory until the assignment p = &x, which points it to stack memory.

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 two memory leaks. Do free(A) immediately before the assignment A = B, and do free(p) before the assignment p = &x to eliminate the leaks.
4. (20 Points) Write a C function called `mix()` with the heading given below, that takes as input two C-strings `A` and `B`, (i.e. null '\0' terminated char arrays), allocates a new char array from heap memory of appropriate length, copies the contents of `A` and `B` into that array by alternating one character from `A` and one from `B` (starting with `A`), adds the null character '\0' to make the array a valid C-string, then returns the address of the newly allocated array. Note that `A` and `B` may have different lengths. If one string is exhausted before the other, copy the remaining characters in the other string to the new array in order. Thus if `A` is the string "abcdef" and `B` is the string "xyz", then the returned string will be "axbyczdef". You may assume the availability of the C library function `strlen()` from `string.h`, which returns the length of a C-string.

**Solution:**

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

```c
char* mix(char* A, char* B){
    // your code begins here

    int i = 0, j = 0, k = 0;
    int n = strlen(A), m = strlen(B);
    char* C = calloc(n+m+1, sizeof(char));

    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';
    return C;

    // your code ends here
}
```
5. (20 Points) Write a recursive C function called printTernary(), with the heading below, that prints out the ternary (base 3) digits of the number \( n \) from most to least significant (left to right). You may assume that \( n \) is always a positive integer, and in particular, it need not print out anything when \( n \) is zero.

```c
#include<stdio.h>

void printTernary(int x) {
    // your code starts here

    if( x>0 ){
        printTernary(x/3);
        printf("%d", x%3);
    }

    // your code ends here
}
```