CSE 15  
Introduction to Data Structures  
Midterm 1 Review  
Solutions to Selected Problems  

1. Recall the recursive function $C(n, k)$ in the program BinomialCoefficients.c discussed in lecture and posted on the webpage. Write a box trace of the function call $C(5, 3)$. Use this trace to find the value of $C(5, 3)$. Notice that in the full recursion tree for $C(5, 3)$, the value $C(3, 2)$ is evaluated 2 times, and $C(2, 1)$ is evaluated 3 times. Suggest a modification to the function that would allow it to avoid computing the same values multiple times. Carry out your suggestion by writing the C code.  

Solution to second question:  
Suggested modification: when $C(n, k)$ is computed for the first time (for a particular $n$ and $k$), save the value in a static 2-dimensional array for later re-use. If the value $C(n, k)$ is needed at some later time, look it up in the array instead of computing it again.  

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

#define SIZE 10  

int C(int n, int k, int BinCoef[SIZE+1][SIZE+1]){  
   if( BinCoef[n][k]==0 ){  
      return BinCoef[n][k];  
   }else if( k==0 || k==n ){  
      return 1;  
   }else{  
      BinCoef[n][k] = C(n-1,k-1, BinCoef)+C(n-1,k, BinCoef);  
      return BinCoef[n][k];  
   }  
}  

int main(){  
   int n = SIZE, k = 6, i, j;  
   int BC[SIZE+1][SIZE+1];  
   for(i=0; i<=n; i++)  
      for(j=0; j<=k; j++)  
         BC[i][j] = 0;  
   printf( "%d\n", C(n, k, BC) );  
   return EXIT_SUCCESS;  
}  
```
2. Write a recursive function called \texttt{sum(n)} that computes the sum of the integers from 1 to \textit{n}. Hint: emulate the factorial function discussed in lecture (replace multiplication by addition). Modify your answer to recursively compute the sum of the integers from \textit{n} to \textit{m}, where \textit{n} \leq \textit{m} (return 0 if \textit{n} > \textit{m}).

\textbf{Solution to second question:}

\begin{verbatim}
int sum(int n, int m){
  if (n<=m){
    return sum(n, m-1) + m;
  }else{
    return 0;
  }
}
\end{verbatim}

3. Write recursive functions that determine the sum of the elements in an int array. Do this in 3 ways.
   a. Write a recursive function \texttt{sumArray1(int A[], int n)} that returns the sum of the leftmost \textit{n} elements of \texttt{A[]}. Get the \textit{n}th element from the left, compute the sum of the leftmost \textit{(n-1)} elements recursively, then return the sum.
   b. Write a recursive function \texttt{sumArray2(int A[], int n)} that returns the sum of the rightmost \textit{n} elements of \texttt{A[]}. Get the \textit{n}th element from the right, compute the sum of the rightmost \textit{(n-1)} elements recursively, then return the sum.
   c. Write a recursive function \texttt{sumArray3(int A[], int p, int r)} that returns the sum of the subarray \texttt{A[p...r]}. \textbf{Hint: use MergeSort()} as a model.
Solution:
#include<stdlib.h>
#include<stdio.h>

// return the sum of the leftmost k elements in array A of length n
int sumArray1(int A[], int n, int k){
    int a, b;
    if( k<=0 ){                       // if array is empty
        return 0;                    // return zero
    }else{                          // else if array is non-empty
        a = A[k-1];                 // get kth element from the left
        b = sumArray1(A, n, k-1);   // compute sum of leftmost (k-1) elements
        return a+b;                 // return the sum
    }
} // notice n was not used

// return the sum of the rightmost k elements in array A of length n
int sumArray2(int A[], int n, int k){
    int a, b;
    if( k<=0 ){                       // if array is empty
        return 0;                    // return zero
    }else{                          // else if array is non-empty
        a = A[n-k];                 // get kth element from the right
        b = sumArray2(A, n, k-1);   // compute sum of rightmost (k-1) elements
        return a+b;                 // return the sum
    }
}

// return sum of the subarray A[p...r]
static int sumArray3(int A[], int p, int r){
    int a, b, q;
    if( p<r ){                       // if array is empty
        q = (p+r)/2;
        a = sumArray3(A, p, q);
        b = sumArray3(A, q+1, r);
        return a+b;
    }else{
        return A[p];
    }
}

int main(){
    int n, k;
    int X[] = {1, 2, 3, 4, 5, 6, 7};
    n = k = 7;

    printf("%d\n", sumArray1(X, n, k));
    printf("%d\n", sumArray2(X, n, k));
    printf("%d\n", sumArray3(X, 0, n-1));

    return EXIT_SUCCESS;
}
4. Write a modification of the recursive function BinarySearch() that prints out the sequence of array elements that are compared to the target.

Solution:
#include<stdlib.h>
#include<stdio.h>

int BinarySearch(int X[], int p, int r, int target){
    if( p<=r ){
        int q = (p+r)/2;
        printf("%d ", X[q]);
        if( target==X[q] ){
            return q;
        }else if( target<X[q] ){
            return BinarySearch(X, p, q-1, target);
        }else{  // target>X[q]
            return BinarySearch(X, q+1, r, target);
        }
    }else{
        return -1;
    }
}

int main(){
    int A[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27};
    int n = 14; // length of A[]
    //int t = 12; // target
    int t = 23; // target
    int k; // index of target, if found, -1 otherwise
    k = BinarySearch(A, 0, n-1, t);
    printf("\n");
    if( k>=0 ){
        printf("target found at index %d\n", k);
    }else{
        printf("target not found\n");
    }
    return EXIT_SUCCESS;
}
6. Perform a box trace of the following recursive function for the input $n = 100$, and determine the output. What does the function do?

```c
#include<stdio.h>

void doSomething(int n){
  if(n>=8){
    doSomething(n/8);
  }
  printf("%d", n%8);
}
```

Solution:
This function prints the octal digits of the input $n$. The box trace follows.

```
$ n = 100 $  Print 4
     |
$ n = 12 $  Print 4
     |
$ n = 1 $  Print 1
```

Output: 144

7. (Modified version) Use what you learned in problem 6 above to create a recursive function called `integerToString()` that returns a String representation of the integer $n$ expressed in base $b$. For instance, the function call `integerToString(100, 8)` would return the String “144”, which was printed in problem 6. Your function need not allow bases greater than 10 (although that would be a good exercise), but it should deal correctly with the input $n = 0$ (i.e. return the digit "0" in that case).

Solution:
The following full Java program defines `integerToString()` and tests it on various bases. It uses a helper function called `digit()` that produces digits in bases greater than 10.

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

// digit()
// pre: 0 <= d <= 35
// Returns a printable ascii char from the set {0, 1,..., 9, A,..., Z}
// representing d in the range 0 <= d <= 35.
char digit(int d){
    if( d<0 || d>35 ){ /*
        fprintf(stderr, "no digit available to represent %d\n", d);
        exit(EXIT_FAILURE);
    */
        return -1; /*
    }
    if( d<10 ){
        return (char)(d+48);  // 48 <= (d+48) <= 57
    }else{
        return (char)(d+55);  // 65 <= (d+55) <= 90
    }

}

// printInteger()
// Returns a string representation of n in base b, where b is in the range
// 2 <= b <= 36.
void printInteger(int n, int b){
    if( n>0 ){
        if( n>=b ){ /*
            printInteger(n/b, b);
        */
            printf("%c", digit(n%b)); // append another digit, doesn't work if n==0
        }else{
            printf("0");              // in case n==0
        }
    }
}

int main(){
    int b;
    int n = 43981;
    int n = 0;
    int n = 100;

    for(b=2; b<=36; b++){ /*
        printf("base = %d\t", b);
        printInteger(n, b);
        printf("\n");
    */
        return EXIT_SUCCESS;
    }
}
9. Write a C function called `search()` with the prototype below that takes as input a null ('\0') terminated char array S (i.e. a string) and a single char c, and returns the leftmost index in S at which the target c appears, or returns -1 if no such index exists.

**Solution:**

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

int search(char* S, char c){
    // your code goes here
    int i=0;

    while( S[i]!="\0" ){
        if( S[i]==c ) break;
        i++;
    }
    if( S[i]!="\0" ) return -1;
    else return i;
}

int main(){
    char str[] = "onewothreefourfivesixseveneightnineteen";

    printf("%d\n", search(str, 'x'));
    printf("%d\n", search(str, 'g'));
    printf("%d\n", search(str, 'z'));

    return EXIT_SUCCESS;
}
```