CS101 Lab 6: Recursion; Passing arrays as parameters

1 What to do?

1. Read Sections 2, 3, 4 and 5. Refer to lecture slides/programs.
2. Solve problems in Section 6.

2 Recursion

A function can call itself. That is recursion. Iterations can be converted to recursion. We will try this out in today's lab. for example,
fact(n) = n * fact(n-1)
is a recursive function. Fact is defined on non-negative integers. However, we need a terminating condition to end recursion, which in this case is fact(0)=1. We can easily express this recursive formulation in terms of a function calling itself. Now carry on with section 3.

3 A recursive function with a bug

```
// function fact() is meant to compute factorial of n recursively
/ fix errors in it-- this is your problem statement 1.

unsigned int fact (unsigned int);
unsigned int fact (unsigned int n) {

 return n * fact (n-1);
 if (n=1) return 1;

}

int main () {

 unsigned int n;
 cin >> n;
 cout << fact (n);
}
```

4 How to refer to a tail of an existing array

It is possible to create a different variable to refer to a tail of an existing array. Execute the following program. B starts from location no. 1 of array A. B is

not another copy of A, just that it is positioned inside A. A is a pointer, and so is B. This technique will be used in writing recursive programs with arrays as parameters as shown in Section 5.

```
#include<iostream>
using namespace std;

int main () {

int *A, *B;

 A = new int [3];
 A[0]=1;
 A[1]=2;
 A[2]=3;

 B=A+1;

 cout << A[0] << A[1] << A[2] <<  endl;
 cout << B[0] << B[1] << endl;
}
```

5 Passing an array into parameter

```
#include<iostream>
using namespace std;

int func (int A[], int n) {

  if (n==1) return A[0];
        else
          return A[0]+func(A+1,n-1);
}

int main () {

 int n;
 int *A;

        cin >> n;

        A = new int [n];
        for (int i=0; i<n; i++) cin >> A[i];

        cout << func (A,n) << endl;
}
```

6 Problem statement

Do not use iteration.

1. Try the program given in Section 3. Fix it.
2. Write a program that computes Fibonacci(n) recursively. Compute the no. of times function Fibonacci() gets called for any given number n.
3. Try out program given in section 4. What does it do? Now try out program in Section 5. What does it compute? Copy the function into another function called 'average'. Modify this newly created function to compute the average of values in the input array. Stay in recursion.
4. Write a recursive function that computes the maximum value from a given array.
5. (Optional, not that easy) Invert an array of size n without using iteration.