CS19001/CS19002 PROGRAMMING AND DATA STRUCTURES LABORATORY
LAB TEST – ODD PC
Last Date of Submission: 06–April–2015

Part 1

You are given *m* red balls and *n* green balls. Your task is to print all possible permutations of the *m* + *n* balls.

Write a program that first reads *m* and *n* from the user. The program then prints all of the (*m* + *n*)! / (*m*! *n*!) permutations of the *m* + *n* balls. Each permutation must be printed only once and in a single line.

Sample output

```
m = 2
n = 3
RRGGG
RGRGG
RGGRG
RGGGR
GRRGG
GRGRG
GRGGR
GGRRG
GGRGR
GGGRR
```

Part 2

Let *A* be an array of size *n*. Each cell *A*[*i*] in the array stores an index in the range 0 to *n* – 1. A *walk* in the array *A* is defined in the following way. You start from some index *i* in the range 0 to *n* – 1. You then visit the cells *A*[*i*], *A*[*A*[*i*]], *A*[*A*[*A*[*i*]]], … in that sequence. The cells visited are *i*, *A*[*i*], *A*[*A*[*i*]], *A*[*A*[*A*[*i*]]], … Since *A* is a finite array, you cannot visit new cells for an infinite number of iterations. In this part, you solve a problem on walks in the array *A*.

Ask the reader to enter the size *n* of the array *A*. This should be a dynamic array. Write a function to allocate memory to *A*, and populate the entries of *A* by random indices in the range [0 … *n* – 1]. Return the array (an int pointer actually).

Now, the user supplies two different indices *s* and *t*. Write a function to do the following. Consider two walks in *A* starting from *s* and *t*. Find out whether the walks share a common index. If not, print that and return. If there is a common index *r*, the function should print the walk from *s* to *r* and the walk from *t* to *r*. The index *r* should be so chosen that the two walks printed must not have any common index except the last one (*r*).

Hint: Mark visited indices.

Sample output

```
n = 60
The array A is:
     33     33     56     35     14      9     53     31     44     23
     13      9     35     58     50     22     31     27     40     52
     21     35     15     29      8     17     31     54     38      3
     32     12     36     20     47     43     30     32     14     14
     56     20     16     23     18      6     45     50     33     17
     34     46     45     41     15     53     58     38     48     37
s = 5
t = 8
+++ Path from Cell 5:
      5      9     23
+++ Path from Cell 8:
      8     44     18     40     56     58     48     33     20     21
     35     43     23
```

Submit *two separate files* for the two parts (with the names *permute.c* and *walk.c*). Write your name, roll number, and PC number in both the files. Your programs *must not* use any global variables.

Part 1

You are given an infinite supply of coins of two different integer denominations c and d Rupees. Your task is to find out all possible ways of making a change of n Rupees using the coins of the given denominations. The order in which the coins are chosen matters. For example, the changes 2+2+5, 2+5+2 and 5+2+2 are considered different.

Write a program that first reads c, d and n from the user. All possible changes are then printed for n Rupees using coins of denominations c and d Rupees.

Sample Output

```
c = 2
d = 3
n = 12
2+2+2+2+2+2
2+2+2+3+3
2+2+3+2+3
2+2+3+3+2
2+3+2+2+3
2+3+2+3+2
2+3+3+2+2
3+2+2+2+3
3+2+2+3+2
3+2+3+2+2
3+3+2+2+2
3+3+3+3
```

Part 2

Let A be an array of size n. Each cell $A[i]$ in the array stores an index in the range 0 to $n-1$. A *walk* in the array A is defined in the following way. You start from some index i in the range 0 to $n-1$. You then visit the cells $A[i]$, $A[A[i]]$, $A[A[A[i]]]$, … in that sequence. The cells visited are i, $A[i]$, $A[A[i]]$, $A[A[A[i]]]$, … Since A is a finite array, you cannot visit new cells for an infinite number of iterations. In this part, you solve a problem on walks in the array A.

Ask the reader to enter the size n of the array A. This should be a dynamic array. Write a function to allocate memory to A, and populate the entries of A by random indices in the range $[0 \ldots n-1]$. Return the array (an int pointer actually).

Now, the user supplies an index s. As stated above, a walk starting from s will be eventually periodic. Indeed, there is an initial part of the walk (may be empty). However, after the *first* duplication of a visited index, the walk will be cyclic (that is, the same sequence of indices will be visited again and again). Write a function that given the array A and the starting index s, prints the initial part of the walk (before the cycle begins), and the indices in the cycle in the sequence as they are visited. Print also the length of the cycle.

Hint: Mark visited indices.

Sample Output

```
n = 60
The array A is:
     44     47     19     15     23      6     49      7     11     40
     49     54     25     12     22     29      2     24     15     34
      9      4     46     12     12      2      9     41     30     18
      9      6     57     20     21     20     26     10     20     29
     43      9     23      0     21     38     21     15      2     36
     49     11     41     35     23     53     29     32     27     51
s = 31
Initial part:
     31      6     49     36     26      9     40     43      0     44
Cyclic part:
     21      4     23     12     25      2     19     34
Length of the cycle = 8
```

Submit *two separate files* for the two parts (with the names *coin.c* and *walk.c*). Write your name, roll number, and PC number in both the files. Your programs *must not* use any global variables.