IE531: Algorithms for Data Analytics
Spring, 2018
Homework 1: Review of Linear Algebra, Probability & Statistics and Computing
Due Date: February 16, 2018

©Prof. R.S. Sreenivas

Instructions

1. You can modify any of the C++ code on Compass to solve these problems, if you want. It might help you with honing your programming skills. If these attempts (at using C++ code) is turning out to to be intense, you can use MATLAB just this once.
2. You will submit a PDF-version of your answers on Compass on-or-before midnight of the due date.

Instructions

1. **Linear Algebra**: (*30 points*) Consider the following set of linear equations involving six equations in five unknowns $x_1, x_2, \ldots, x_5$.

$$\underbrace{\begin{pmatrix} 2 & 0 & 0 & 6 & 2 \\ 1 & -1 & -1 & 4 & 0 \\ 2 & -2 & 2 & 4 & 0 \\ 2 & -2 & 0 & 6 & 0 \\ -4 & 4 & -8 & -4 & 0 \\ 4 & 0 & -2 & 14 & 4 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}}_{\mathbf{x}} = \underbrace{\begin{pmatrix} 2 \\ 7 \\ 18 \\ 16 \\ -40 \\ 2 \end{pmatrix}}_{\mathbf{y}}$$

You can use NEWMAT or MATLAB for this part. I used the MATLAB code shown in figure 1.

(a) (*2 points*) Show that there a solution to these equations.
Since $Rank(\mathbf{A}) = Rank(\mathbf{A} \mid \mathbf{y}) = 3$, it follows that there is a solution to this equation. Since there are five-columns in $\mathbf{A}$, it follows that (1) there are infinitely-many solutions to this equation, and (2) the general-solution will have two "degrees-of-freedom" (or, it will be the affine-combination of three, 5-dimensional, basic-solutions.

(b) (*10 points*) Present a general formula for the set of all solutions that satisfy these equations.
Look at the MATLAB code of figure 1 – for the solutions I chose, I get the general solution of

$$\begin{pmatrix} 8 - 4b - 7a \\ -7a - 7b \\ 1 - b \\ -b \\ 7a + 7b - 7 \end{pmatrix}$$

```
% IE531 Spring 2018 Homework 1 Problem 1
% can be done using NEWMAT (preferred)
A = [2     0     0     6     2;
     1    -1    -1     4     0;
     2    -2     2     4     0;
     2    -2     0     6     0;
    -4     4    -8    -4     0;
     4     0    -2    14     4];
y = [2 7 18 16 -40 2]';
rank(A)
rank([A y])
% rank of these two matrices will be 3... solution exists!
% there are 10 waysB7 = [A(:,2) A(:,3) A(:,5)] of picking 3 items from a set of 5
B1 = [A(:,1) A(:,2) A(:,3)];
B2 = [A(:,1) A(:,2) A(:,4)];
B3 = [A(:,1) A(:,3) A(:,4)];
B4 = [A(:,2) A(:,3) A(:,4)];
B5 = [A(:,1) A(:,2) A(:,5)];
B6 = [A(:,1) A(:,3) A(:,5)];
B7 = [A(:,2) A(:,3) A(:,5)];
B8 = [A(:,1) A(:,4) A(:,5)];
B9 = [A(:,2) A(:,4) A(:,5)];
B10 = [A(:,3) A(:,4) A(:,5)];
% since rank(B3) and rank(B5) is not 3, we can drop them
rank(B3)
rank(B5)
% solutions
s1 = inv(B1'*B1)*B1'*y; s1 = [s1(1); s1(2); s1(3); 0; 0];
s2 = inv(B2'*B2)*B2'*y; s2 = [s2(1); s2(2); 0; s2(3); 0];
s4 = inv(B4'*B4)*B4'*y; s4 = [0; s4(1); s4(2); s4(3); 0];
s6 = inv(B6'*B6)*B6'*y; s6 = [s6(1); 0; s6(2); 0; s6(3)];
s7 = inv(B7'*B7)*B7'*y; s7 = [0; s7(1); s7(2); 0; s7(3)];
s8 = inv(B8'*B8)*B8'*y; s8 = [s8(1); 0;0; s8(2); s8(3)];
s9 = inv(B9'*B9)*B9'*y; s9 = [0; s9(1); 0; s9(2); s9(3)];
s10 = inv(B10'*B10)*B10'*y; s10 = [0; 0; s10];
S = [s1 s2 s4 s6 s7 s8 s9 s10]; rank(S)
% pick the affine combination of any rank-3 submatrix of S
S1 = [s1 s2 s6]; rank(S1)
% this works!
syms a b;
genl_soln = a*s1 + b*s2 + (1-a-b)*s6
% Verification
A*genl_soln
```

Figure 1: MATLAB code for problem 1

where $a, b \in \mathcal{R}$.

(c) (*3 points*) Verify the general formula using MATLAB.

Done within MATLAB (see figure 1) – what this shows is that the general solution derived earlier where $a, b \in \mathcal{R}$ is a solution to the equation. To complete the picture, we have to show that $\overline{\underline{any}}$ solution to the equation shown above is captured by assigning a specific value to a and b in the general solution. While this is certainly true, we are not showing/proving this part in this part of the homework.

(d) (*5 points*) Show that

$$\begin{pmatrix} -7\lambda_1 - 4\lambda_2 + 8 \\ -7\lambda_1 - 7\lambda_2 \\ 1 - \lambda_2 \\ -\lambda_2 \\ -7 + 7\lambda_1 + 7\lambda_2 \end{pmatrix} \tag{1}$$

and

$$\begin{pmatrix} a \\ -7a - 7b \\ -\frac{8}{3}a - \frac{7}{3}b + \frac{11}{3} \\ -\frac{8}{3}a - \frac{7}{3}b + \frac{8}{3} \\ -7 + 7a + 7b \end{pmatrix}. \tag{2}$$

are solutions to the equation $\mathbf{Ax} = \mathbf{y}$ identified earlier.

To quote the shoemaker *Nike*© – *Just do it!*

(e) (*5 points*) Present a cogent argument in support of the claim that even though the solution identified by equations 1 and 2 look different, they are in effect the same.

Equations 1 and 2 are identical, provided you made the substitution

$$\begin{aligned} a &= -7\lambda_1 - 4\lambda_2 + 8 \\ b &= 8\lambda_1 + 5\lambda_2 - 8. \end{aligned}$$

(f) (*5 points*) Show that if add the constraints

$$\begin{aligned} 7x_4 + x_5 &= 7 \\ x_2 + 2x_5 &= 7, \end{aligned}$$

in addition to those already in the requirement that $\mathbf{Ax} = \mathbf{y}$ as described in problem 1, there can be only one solution to the combined set of equation-s/constraints.

From the general solution we can infer that the above conditions would require $\lambda_1 = 2$ and $\lambda_2 = 1$, which means there is just a single solution to the resulting problem, and it is

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} -10 \\ -21 \\ 0 \\ -1 \\ 14 \end{pmatrix}$$

Alternately, you could incorporate the two extra conditions as additional rows to the formulation in problem 1 and solve for

$$\underbrace{\begin{pmatrix} 2 & 0 & 0 & 6 & 2 \\ 1 & -1 & -1 & 4 & 0 \\ 2 & -2 & 2 & 4 & 0 \\ 2 & -2 & 0 & 6 & 0 \\ -4 & 4 & -8 & -4 & 0 \\ 4 & 0 & -2 & 14 & 4 \\ 0 & 0 & 0 & 7 & 1 \\ 0 & 1 & 0 & 0 & 2 \end{pmatrix}}_{\mathbf{B}} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}}_{\mathbf{x}} = \underbrace{\begin{pmatrix} 2 \\ 7 \\ 18 \\ 16 \\ -40 \\ 2 \\ 7 \\ 7 \end{pmatrix}}_{\mathbf{z}}$$

It is not hard to show that $rank(\mathbf{B}) = 5$, which means there can be only one solution, and it is given by the expression $(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{z}$, this will also result in the solution

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} -10 \\ -21 \\ 0 \\ -1 \\ 14 \end{pmatrix},$$

which should reaffirm our faith in the general solution.

2. (*30 points*) **Generating RVs using the Inverse Transform Technique**: You may want to use the various C++ code samples on Compass (along with some graphing/plotting software) to verify/experiment-with things before you put your answers down for this problem.

   (a) (*10 points*) The *triangular distribution* has a *probability density function*

$$f(x) = \begin{cases} x & 0 \le x \le 1 \\ 2 - x & 1 < x \le 2 \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

   In three/four sentences describe how you can generate i.i.d. r.v's that are distributed according to equation 3.

   The cumulative distribution function is

$$F(x) = \begin{cases} 0 & x \le 0 \\ \frac{x^2}{2} & 0 < x \le 1 \\ 1 - \frac{(2-x)^2}{2} & 1 < x \le 2 \\ 1 & x > 2 \end{cases}$$

   So, if x = get_uniform(),

$$y = \begin{cases} \sqrt{2x} & \text{if } x \le 0.5 \\ 2 - \sqrt{2 - 2x} & \text{otherwise} \end{cases}$$

will generate i.i.d. r.v's that are distributed according to equation 3.

Extra Material (Start)

To make it easier to follow what is going on with the next two problems, this above scheme can be implemented as

```
double blah( )
{
 (double) x = get_uniform();
 if (x <= 0.5)
             return ( $\sqrt{2x}$);
 else
        return (2 − $\sqrt{2 - 2x}$);
}
```

Extra Material (End)

(b) (*10 points*) What is the i.i.d. distribution generated by repeated calls to the code shown below?

```
double blah1( )
{
 (double) x = get_uniform();
 if (x <= 0.5)
             return (1 + $\sqrt{2x}$);
 else
        return (1 − $\sqrt{2 - 2x}$);
}
```

Compare the code listed above with that of the code for problem 2a (which gives us the "*A-shaped*" Triangular Distribution).

In blah() when $x <= 0.5$ it returned $\sqrt{2x}$; while blah1() returned $1 + \sqrt{2x}$ when $x <= 0.5$. That is, blah1() = 1 + blah() when $x <= 0.5$. For the region $x <= 0.5$, we know blah() gives us the portion of the PDF that corresponds to figure 2(a); this would mean, for the same region, blah1() should give us the portion of the PDF that corresponds to what is shown in figure 2(b).

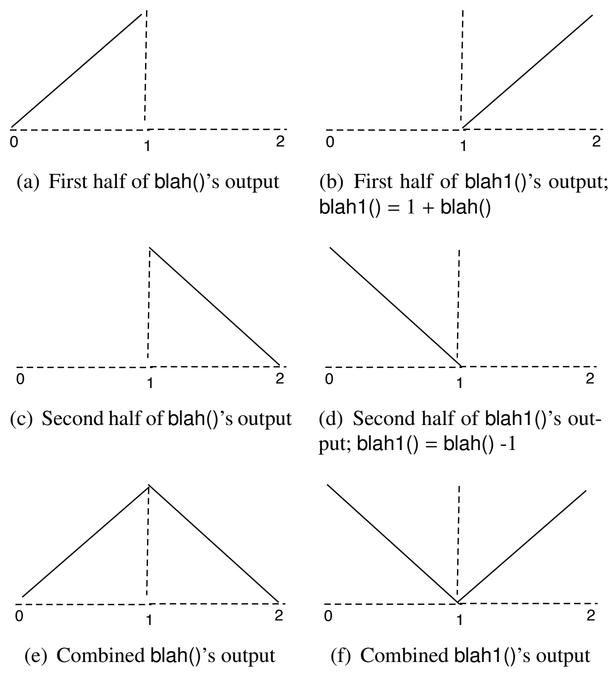
Similarly, for $x > 0.5$, we have blah1() = blah() -1 (why?). For the region $x > 0.5$, we know blah() gives us the portion of the PDF that corresponds to figure 2(c); this would mean, for the same region, blah1() should give us the portion of the PDF that corresponds to what is shown in figure 2(d).

Combining the two halfs we get the PDF genereated by blah() shown in figure 2(e); while that of blah1() is shown in figure 2(f).

You an verify it by running the C++ code s18_hw1_pr2b.cpp on compass. A little thought should tell you why we get this distribution out of blah1().

(c) (*10 points*) What is the i.i.d. distribution generated by repeated calls to the code shown below?

```
double blah2( )
```



(a) First half of blah()'s output (b) First half of blah1()'s output; blah1() = 1 + blah()

(c) Second half of blah()'s output (d) Second half of blah1()'s output; blah1() = blah() -1

(e) Combined blah()'s output (f) Combined blah1()'s output

Figure 2: A graphical illustration of the PDF generated by blah1(), knowing that blah() generates the "A-shaped" Triangular distribution.

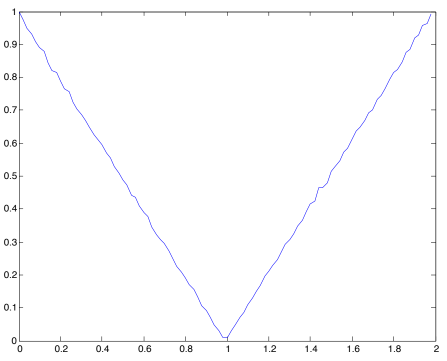


Figure 3: Experimentally observed PDF of the RVs generated by blah1().

```
{
  (double) x = get_uniform();
  if (x <= 0.5)
            return (2 − $\sqrt{1-2x}$);
  else
      return ( $\sqrt{2x-1}$);
}
```

Here again, compare the code shown above, with that of problem 2a. When $x <= 0.5$ blah() returned $\sqrt{2x}$ (call this **A** in short-hand); for $x > 0.5$, it returned $2 - \sqrt{2-2x}$ (call this **B** in short-hand).

That is, 50% of the time (i.e. $x <= 0.5$) blah() did **A**; and the other 50% of the time (i.e. $x > 0.5$) it did **B**. It does not matter which 50% – that is if $x > 0.5$ you can do **A**; $x <= 0.5$ you can do **B** – and you will still get the "A-shape" Triangular PDF. When you switch the 50%'s like this you have to adjust **A** and **B** accordingly.

That is, if $x > 0.5$ you want to do **A** (keep in mind **A** requires the $0 \leq x \leq 0.5$), you have to subtract 0.5 from x for things to work. That is for $x > 0.5$, if you want to do **A**, you should output $\sqrt{2(x-0.5)} = \sqrt{2x-1}$. Similarly, if for $x <= 0.5$, you wish to do **B**, you should output $2 - \sqrt{2-2(x+0.5)} = 2 - \sqrt{1-2x}$. This is exactly what blah2() does – therefore, we expect blah2() to produce the "A-shaped" Triangular PDF as well.

Figure 4 shows the experimentally generated density function, which confirms the RVs generated by blah2() follow the triangular distribution.

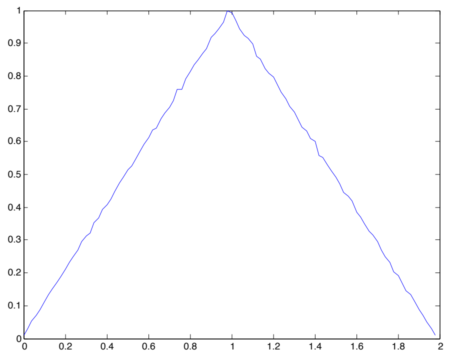


Figure 4: Theoretical and Experimentally observed PDF of the RVs generated by blah2().

3. (*40 points*) **General Programming Concepts**:

   (a) (*10 points*) Consider the two versions of the code shown in figure 5. Suppose I type the string "kablooey" as input, give me the reasoning behind the output each of the programs generate[1].

```
#include <iostream>
#include <cmath>
using namespace std ;
void f()
{
    char c;
    cin.get(c);
    // this line reads character "c"
    if ('\n' == c)
        // '\n' is "return/newline"
        return;
    cout << c;
    f();
}
int main()
{
    f();
    cout << endl;
}
```

(a) Problem 3a(I)

```
#include <iostream>
#include <cmath>
using namespace std ;
void f()
{
    char c;
    cin.get(c);
    // this line reads character "c"
    if ('\n' == c)
        // '\n' is "return/newline"
        return;
    f();
    cout << c;
}
int main()
{
    f();
    cout << endl;
}
```

(b) Problem 3a(II)

Figure 5: (Two versions of the) Code for problem 3a.

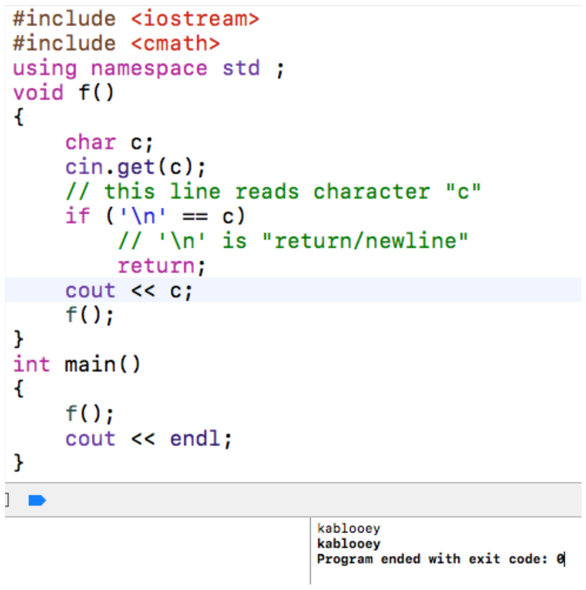
If you typed the string "kablooey" and pressed the return-key, you will get the string "kablooey" as output. The reasoning is quite straightforward – the character that is read is immediately put in a "queue" to be printed, and when the return-key (i.e. \n) is read, the original string is printed out. This is what we would get with the code sample shown in figure 3a(I) above (cf. figure 6(I))

In the code shown in figure 3a(II), we put the cout << c statement *after* the recursive call to f(), the string will be printed in reverse order (cf. figure 6(II)).

(b) (*10 points*) The *greatest common divisor* (GCD) of two positive numbers m and n is the largest number k that divides m and n. That is, $\frac{m}{k}$ and $\frac{n}{k}$ leaves no remainder. Consider the C++ code shown in figure 7. Using the fact that the GCD of two numbers does not change if the smaller number is subtracted from the larger number, show that the recursive funtion gcd(m, n) in the code shown in figure 7 implements the GCD computation.

(**Note**: Please do not give me an illustrative example as a "proof" of that the

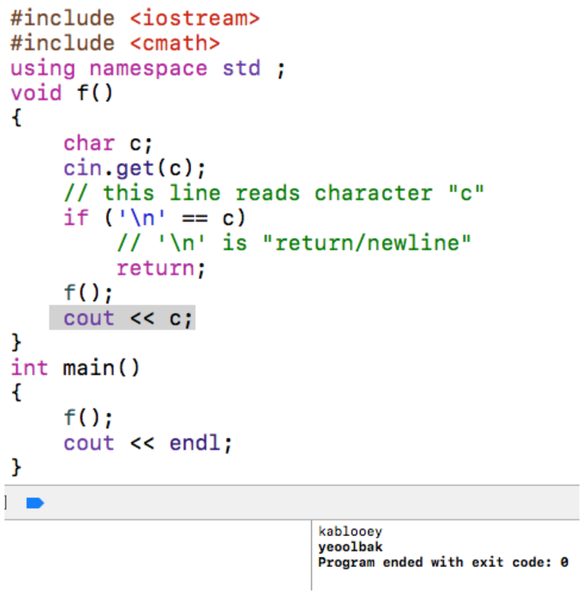
[1] I am looking for an answer that justifies why we get to see the output generated by each of these programs.



```
#include <iostream>
#include <cmath>
using namespace std ;
void f()
{
    char c;
    cin.get(c);
    // this line reads character "c"
    if ('\n' == c)
        // '\n' is "return/newline"
        return;
    cout << c;
    f();
}
int main()
{
    f();
    cout << endl;
}
```

```
kablooey
kablooey
Program ended with exit code: 0
```

(a) Problem 6(I)



```
#include <iostream>
#include <cmath>
using namespace std ;
void f()
{
    char c;
    cin.get(c);
    // this line reads character "c"
    if ('\n' == c)
        // '\n' is "return/newline"
        return;
    f();
    cout << c;
}
int main()
{
    f();
    cout << endl;
}
```

```
kablooey
yeoolbak
Program ended with exit code: 0
```

(b) Problem 6(II)

Figure 6: Output of the (Two versions of the) Code for problem 3a.

code does what it is supposed to do. *I am looking for a rigorous attempt at showing the correctness of the code for any pair of numbers.*)

```
#include <iostream>
using namespace std;
long gcd(long m, long n)
{
        if (m == n)
                return n;
        else if (m < n)
                return gcd(m,n-m);
        else
                return gcd(m-n, n);
}
int main()
{
        cout << gcd(259, 111) << endl;
}
```

Figure 7: Problem 3b.

If two numbers m and n are divisible by p then it follows that $m + n$ and $m - n$ (assuming $m > n$, without loss of generality) is also divisible by p, It is also easy[2] to see that $gcd(n, n) = n$. So, it follows that,

$$gcd(m, n) = \begin{cases} gcd(m - n, n) & \text{if } m > n \\ gcd(m, n - m) & \text{if } n > m \\ n & \text{if } n = m. \end{cases}$$

So, at the i-th stage of any trace, we are replacing $gcd(m.n)$ with either $gcd(m - n, n)$, $gcd(m, n - m)$ or $gcd(n, n)(= n)$ depending on whether $m > n, n > m$ or $n = m$. Since each stage is correct, and one of the two arguments of m or n decreases in size in each step, it follows that the recursion will stop eventually (with the correct answer).

(c) (*10 points*) What is the output of the C++ code shown in figure 8. Make sure you give me your reasoning for your answer.

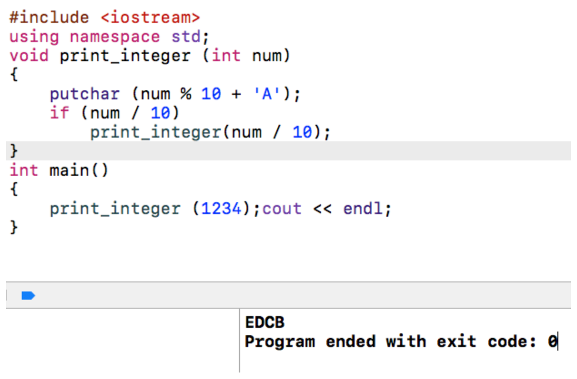
[2] If $m > n$, and $p = gcd(m, n) \Rightarrow gcd(m - n, n) \geq p$. Also, $q = gcd(m - n, n) \Rightarrow gcd(m, n) \geq q$ (why? because $(m - n) = q \times \alpha, n = q \times \beta \Rightarrow m = q \times (\alpha + \beta)$). Therefore $gcd(m, n) = gcd(m - n, n)$.

```
#include <iostream>
using namespace std;
void print_integer (int num)
{
    putchar (num % 10 + 'A');
        if (num / 10)
                print_integer(num / 10);
}
int main()
{
        print_integer (1234);cout << endl;
}
```

Figure 8: Problem 3c.

What is the output of the above program? What is the reasoning behind the output that this program generates?

When I compile and run the C++ code shown in figure 8, I get the output shown in figure 9 – *EDCB*.



```
#include <iostream>
using namespace std;
void print_integer (int num)
{
    putchar (num % 10 + 'A');
    if (num / 10)
        print_integer(num / 10);
}
int main()
{
    print_integer (1234);cout << endl;
}
```

```
EDCB
Program ended with exit code: 0
```

Figure 9: The output generated by the code shown in figure 8.

If num is a single-digit integer, then num/10 rounded-off to an int is zero. The if statement is not executed in this case, and the code prints out num%10, which is (the character corresponding to) num.

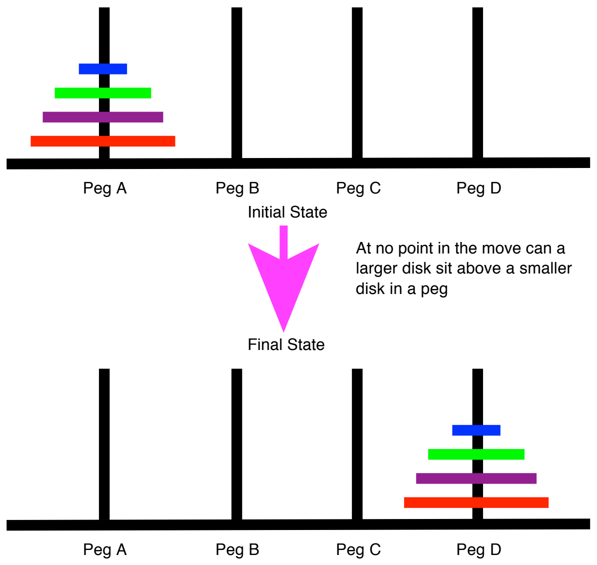
If num is a multi-digit integer, then num/10 rounded-off to an int will be a number where the right-most digit (which is num%10) is dropped off. After all the children of this parent-recursion are completed, the code prints out (the character corresponding to) num%10 in reverse-order.

(d) (*10 points*) My review of computing contains the *Frame-Stewart* solution to the *k*-peg version of the *Tower of Hanoi* problem. Consider the following solution to the 4-peg version of the problem.

Alternate Solution to 4-Peg Tower of Hanoi Problem

```
1: int $k = \lfloor\sqrt{2 * n}\rfloor$;
2: Move_Using_Four_Pegs ($n - k$, source, intermediate_1, intermediate_2, destination);
3: Move_Using_Four_Pegs ($k - 1$, source, intermediate_2, destination, intermediate_1);
4: cout << "Move the top disk from peg " << source << " to peg " << destination << endl;
5: Move_Using_Four_Pegs ($k - 1$, intermediate_2, destination, intermediate_1, source);
6: Move_Using_Four_Pegs ($n - k$, intermediate_1, destination, intermediate_2, source);
```

Comment on the correctness of this algorithm. Will this work? (**You might want to spend a little time thinking about this problem before you wrote down your answer**)


Peg A
Peg B
Peg C
Peg D
Initial State
At no point in the move can a larger disk sit above a smaller disk in a peg
Final State
Peg A
Peg B
Peg C
Peg D

This is what I have in my notes for the 4-Peg Tower of Hanoi recursion.

Method1

```
1: int $k = \lfloor\sqrt{2 * n}\rfloor$;
2: Move_Using_Four_Pegs ($n - k$, source, intermediate_1, intermediate_2, destination);
3: Move_Using_Three_Pegs ($k$, source, destination, intermediate_2);

4: Move_Using_Four_Pegs ($n - k$, intermediate_1, destination, intermediate_2, source);
```

The alternate approach was to do something along the lines of

```
Method2
1: $int\ k = \lfloor \sqrt{2 * n} \rfloor$;
2: Move_Using_Four_Pegs ($n - k$, source, intermediate_1, intermediate_2, destination);
3: Move_Using_Four_Pegs ($k - 1$, source, intermediate_2, destination, intermediate_1);
4: cout << ”Move the top disk from peg ” << source << ” to peg ” << destination << endl;
5: Move_Using_Four_Pegs ($k-1$, intermediate_2, destination, intermediate_1, source);
6: Move_Using_Four_Pegs ($n-k$, intermediate_1, destination, intermediate_2, source);
```

which is what is suggested in this problem The problem with this method is that the moves implied by steps 2, where $k-1$ disks are moved from the source-peg to the intermediate_2-peg, might result in a (larger diameter) disk being placed on (top of smaller diameter disks in) intermediate_1-peg. This is a violation of the requirements of this problem. Likewise, a similar violation of requirements can happen with the moves implied by step 4. That said, in the end, you would have moved all disks from the source-peg to the destination-peg with this recursion, perhaps with a fewer number of moves.

Here are a couple of sample runs of the two methods for different #disks.

```
Method2                                  Method1
Enter the #disks in peg A: 3             Enter the #disks in peg A: 3
Move the top disk from peg A to peg B    Move the top disk from peg A to peg B
Move the top disk from peg A to peg C    Move the top disk from peg A to peg C
Move the top disk from peg A to peg D    Move the top disk from peg A to peg D
Move the top disk from peg C to peg D    Move the top disk from peg C to peg D
Move the top disk from peg B to peg D    Move the top disk from peg B to peg D
#Steps: 5                                #Steps: 5
```

That is, the two methods yield the same set of moves for 3 disks. However, when we run the two methods on 8 disks, we get

```
Method2
Enter the #disks in peg A: 8
Move the top disk from peg A to peg B
Move the top disk from peg A to peg C
Move the top disk from peg B to peg C
Move the top disk from peg A to peg D
Move the top disk from peg A to peg B
Move the top disk from peg D to peg B
Move the top disk from peg C to peg A
Move the top disk from peg C to peg B
Move the top disk from peg A to peg B
Move the top disk from peg A to peg D
Move the top disk from peg A to peg B
Move the top disk from peg A to peg C
Move the top disk from peg B to peg C
Move the top disk from peg D to peg C
Move the top disk from peg A to peg D
Move the top disk from peg C to peg B
Move the top disk from peg C to peg A
Move the top disk from peg C to peg D
Move the top disk from peg A to peg D
Move the top disk from peg B to peg D
Move the top disk from peg B to peg D
Move the top disk from peg B to peg C
Move the top disk from peg D to peg C
Move the top disk from peg B to peg A
Move the top disk from peg B to peg D
Move the top disk from peg A to peg D
Move the top disk from peg C to peg B
Move the top disk from peg C to peg D
Move the top disk from peg B to peg D
#Steps: 29
```

```
Method1
Enter the #disks in peg A: 8
Move the top disk from peg A to peg B
Move the top disk from peg A to peg C
Move the top disk from peg B to peg C
Move the top disk from peg A to peg D
Move the top disk from peg A to peg B
Move the top disk from peg D to peg B
Move the top disk from peg C to peg A
Move the top disk from peg C to peg B
Move the top disk from peg A to peg B
Move the top disk from peg A to peg C
Move the top disk from peg A to peg D
Move the top disk from peg C to peg D
Move the top disk from peg A to peg C
Move the top disk from peg D to peg A
Move the top disk from peg D to peg C
Move the top disk from peg A to peg C
Move the top disk from peg A to peg D
Move the top disk from peg C to peg D
Move the top disk from peg C to peg A
Move the top disk from peg D to peg A
Move the top disk from peg C to peg D
Move the top disk from peg A to peg C
Move the top disk from peg A to peg D
Move the top disk from peg C to peg D
Move the top disk from peg B to peg D
Move the top disk from peg B to peg C
Move the top disk from peg D to peg C
Move the top disk from peg B to peg A
Move the top disk from peg B to peg D
Move the top disk from peg A to peg D
Move the top disk from peg C to peg B
Move the top disk from peg C to peg D
Move the top disk from peg B to peg D
#Steps: 33
```

Note that the moves identified in red for Method 2 are illegal. These moves essentially place a larger diameter disk over (an immediately) smaller disk. If you ignored these violations, it turns out Method 2 does the job with a fewer moves.

This example illustrates the need for why need to use Move_Using_Three_Pegs in Method1 – this guarantees that the peg where we temporarily moved $n - k$ disks is not touched/used during the process of moving the remaining k disks to the destination.