Programming Problems

1) Reversing a linked list. Given a linked list, reverse it. Input must be read from a file, "list.dat". It will just be a list of integers. The total number is not known. The program should create a linked list with the given numbers in the same order. Each node contains the value and a pointer to the next node of the list. Defining just TWO additional node pointers you must inverse the given list and print out the numbers in the list. NO OTHER VARIABLE of any type should be defined. Hence the output will be reverse of the input. eg. Input file reads, 3 1 4 2 Linked list will be, Head -> 3 -> 1 -> 4 -> 2 -> Null the the reversed list must be, Null <- 3 <- 1 <- 4 <- 2 <- Head So the output will be, 2,4,1,3.

Algo :

1) Reversing a linked list. Simple. Use two new pointer p, q and the head thats already available. Invert the first three nodes and keep iterating along the list. Else use recursion.

---

2) Koch curve

Write a code to generate the (nasent) Koch curve. A (nasent) koch curve is drawn iteratively. At any iteration the curve is a set of straight lines. In every iteration each line from the previous iteration is split into three parts and the middle part is replaced with two sides of the equilateral triangles in which the middle part is the third side. You start with a straight line _____. (0,0)-(90,0) In the next iteration you get four lines __/__ (0,0)-(30,0), (30,0)-(45,15*sqrt(3)), (45,15*sqrt(3))-(60,0) and (60,0)-(90,0) In the next iteration you apply the same division tecnique to all the four line segemetns (0,0)-(30,0), (30,0)-(45,15*sqrt(3)), (45,15*sqrt(3))-(60,0) and (60,0)-(90,0). And so on... Generate the points after n iterations starting with (0,0)-(x,0) Input: one integer for the number of iterations, n and one float for x. n is typically 6 to 8 Output: All the points of the koch curve after n iterations must be written to a file, "koch.dat" in order from left to right. Use all float values. eg. n=1, x=90 0 0 30 0 45 25.98 60 0 90 0 Note: x and y co-ordinates come alternately. A point occurs on two adjusent lines but is printed only once.

Algo :

2) Koch Curve. Sraight forward mathematical calculations gives the intermediate points. Just add them to your data structure and keep iterating.

---

3) Enemies and boats.

A list of persons represented by numbers 1 to n are given along with their enemies in that list. You must group them into two boats such that no person is in the same boat with his enemy. Assume a solution exists. eg. 1's enemies 2,3 2's enemies 1 3's enemies (none) 4's enemies 3 5's enemies 4,1 The groups will be 1,4 2,3,5 Input: Input should be read from a file, "enemies.dat". First the number of persons in the list is given. Then the enemies for each person is given, terminated by a zero. eg. for the discussed case, input will read 5 2 3 0 1 0 0 3 0 1 4 0 note: Numbers between third and fourth zeros are four's enemies. Output: Output should be printed on the screen as two lines. First line is the list of people on boat one and the next line is for boat two.

Algo :

3) Enemies and boats. Many people gave different algorithms but most didn't work. One way of doing it is to put 1 in boat one, his enemies in boat two. See where 2 can be placed. If you have an inevitable option take it. Else try both option by recursion.

---

4) Euler's knight tour.

Do an Euler knight's tour. A knight starts at one of the squares of the chessboard and visits ALL the squares EXACTLY ONCE. Number the chessboard from 1 to 64 starting from one corner and traversing row wise. Start at square one and visit all the squares. Beware!! The brute force algorithm will burst out of memory! Output: Print out the number of the squares the knight visits in the order they are visited.

Algo :

4) Euler' s knight tour. The algorithm we had in mind was back tracking when you hit a dead end. But we got a much better algo. Start a one corner. At every step try to go to a corner if not possible try to go to a edge, otherwise goto some point. More generally goto the sqaure with the minimum number of squares leading to it. This algo works terrifically and gives instantaneous results.

---

5) Cows and bulls.
Write a program that plays "Cows and bulls". The user thinks of a number less than 10000, with no two digits identical (like 1234. numbers like 0875 are allowed. But 5456 and 3922 are not because it contains 2 indentical digits). The program starts by guessing a number. The user gives the number of hits and misses. A hit is a number in the right place and a miss is a number in the wrong place. e.g If the the number you thought is 4273 the feedback for 2135 is 0 hit 2 miss 0913 is 1 hit 0 miss 4217 is 2 hit 1 miss Your program should use the feedback provided by the user for the next guess and keep guessing after every feedback till it gets the right number. A good program will get the number in about 7 guesses (on an average).

Algo :

5) Cows and bulls. This requires some explaining because nobody got it. But actually its not that difficult. You create an array of all possible numbers. Guess one randomly from that list. Based on the feedback reduce your list eliminating the numbers that can' t satisfy the feedback. Guess again from the new list. You will hit the right number in 5 to 7 trys.

---

6) Knapsack problem.
A Sack is given with a specified volume. And a set of bags are given, each with a specified mass and volume. You have to fill the sack using some of bags such that the total volume of chosen bags is less than or equal to the volume of the sack and their total mass is maximum. Input: Input will be read from a file called "bags.dat". It will contain the number of bags, the volume of the sack, followed by the volume and the mass of the bags. eg. 6 100 10 2 50 10 20 5 40 9 30 5 35 8 Output: Output should be printed on the screen. It will be the number of all the bags to be used. For the given example it will be, 3,4,6

Algo :

6) Knapsack problem. This is a straight forward algorithm. Try all possible combinations that satisfy the given volume constraint and choose the one with the maximum mass.

---

7) Zigzag An n by m matrix with all its elements between 1 to 4 is given. The left top element (1,1) is 1 and right bottom element(m,n) is 4. You have to find out a continuous zigzag path connecting these two elements, such that, 1)small parts of the zigzag are lines connecting two elements. 2)The numbers in the zigzag path should read 1-2-3-4-1-2-3-4... 3)The zigzag must touch all squares. Assume a solution exists. Example : Given: Solution should be: ;1 3 1 2 ;1 3 1-2 |/ \| | 2 1 4 3 2 1 4 3 / \ / 2 3 4 3 2-3 4 3 / /| 4 1 2 4; 4-1-2 4; Input: input should be read from a file called "matrix.dat". The file contains, row no. column no. elements. eg. 4 4 1 3 1 2..... Output: Output should be printed on the screen. Program should print out the row and column numbers of the elements in the order they are visited. eg. (1,1) (2,1) (1,2) (2,3) .....

Algo :

7) Zigzag. Many people used brute force algorithms. Their code didn' t work for a matrix of order 8. One way to do this is to maintain a list of all possible ins and outs for every square. Using the ins of the neigbouring nodes decrease the possible outs for every square. Then use the neighbouring outs to decrease the ins of every square. Do this a few times and you will get very near the solution. Then do a few check and you can get it.

---

8) Partitioning
You have "n" numbers having values from 1 to n. You have to write a code to partition it into "m" groups such that sum of numbers in each group is equal to n*(n+1)/(2*m). Assume that numbers n and m are such that n*(n+1)/(2*m) is an integer. Your program should print out the

numbers in each group. Or if it is not possible to partition the numbers into m groups, then print out that its not possible. Example : n=20 and m=3 Group 1 : 1,8,11,13,18,19 Group 2 : 2,5,12,14,17,20 Group 3 : 3,4,6,7,9,10,15,16 Input: Two integers n,m. Output: Print out to the screen the numbers in each group in a differnt line. Testing: We will run atleast 5 test cases for this problem. So if you cannot write a generalised code which will take care of all n and m, you can try to write codes based on specific conditions.

Algo :

8) Partition. An algo which worked very well, was to start by adding the number n(the largest number) to group 1. Then add the next largest number (n-1 in this case) , and check whether your sum in that group is greater than n(n+1)/(2m). If it is greater, than take out the number and then add the next greatest number and so on till you get the sum equal to n(n+1)/(2m). For the next group, start out similarly, add the largest of the remaining numbers to the group, and keep adding the next largest, till you get the sum.