Assignment 8, TCSS 480 Winter 2016
Due Friday, Mar. 4, 2016, 9 a.m.

OBJECTIVE

The objective of this project is to apply your newly learned SML knowledge to solve an interesting puzzle.

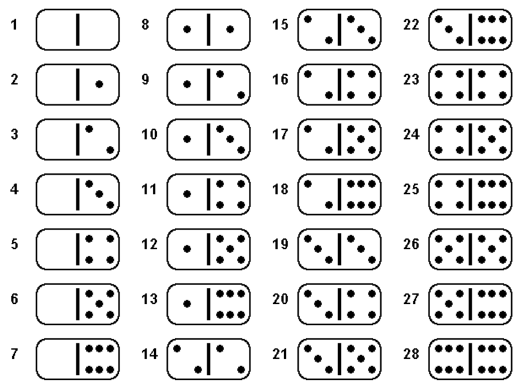
ASSIGNMENT SUBMISSION

You will need to submit your finished sml file (90 points), the test file (if modified), and an executive summary (10 points) through **Canvas**. Your code needs to include comments, follow the prescribed naming conventions, and has to be compatible with **SML N/J**.

ASSIGNMENT DESCRIPTION

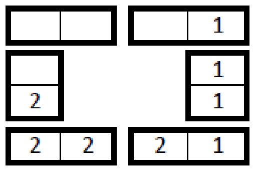
For this assignment, you are to write a program that finds a circular train constructed from all domino tiles from the given set of dominos.

Dominoes are small rectangular game tiles divided into two squares and embossed with dots on the top surface of the tile. You can think of dots as integers. The traditional set of dominoes contains one unique piece for each possible combination of two numbers – such sets are called double-N domino sets. A double-N domino set consists of (N+1)(N+2)/2 tiles, e.g. a standard double-six domino set has 28 tiles (T): one for each possible pair of values from (0.0) to (6.6):



(image from http://www.diy-enthusiasts.com/diy-gifts/diy-kids-games-dominoes-pebbles/)

Dominoes are used to play a variety of games involving patterns. One possible pattern to make with dominoes is a circular train (or a ring), in which all the tiles are laid in a circle, end-to-end, with identical numbers on all adjacent ends. In a double-two domino set, with six tiles, (0 , 0) (0 , 1) (1 , 1) (1 , 2) (2 , 2) (2 , 0) is an example of a ring that uses all the tiles from that set:

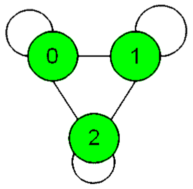


Dominoes and Graphs

There are many possible ways to write the program. Perhaps the simplest way is to generate all possible tile permutations, and to check if any one of them forms a circular train. Unfortunately, you will run out of memory very quickly since the number of permutations is a factorial of the number of tiles (T), e.g. if N=6, then T = 28, and 28! is … well, it is too big of a data structure to be held in memory. How can we improve on this? Perhaps we only generate one permutation at a time and check whether that permutation forms a ring – we will not run out of memory for sure, but using this approach we will most likely run out of patience, as the program will run forever before the match is found.

Fortunately for us, the circular train problem in a double-N domino set can be modeled using a graph, and as such, the problem was solved by Leonhard Euler in the 18th century. The problem is known as Bridges of Königsberg problem and the question that we are really asking is whether there is a Eulerian circuit in a double-N domino set.

Dominoes could be represented as a graph in the following fashion: dots are used as nodes, whereas tiles are used as edges between them. A double tile is shown as an arc that starts and finishes at the same node. The graphs for all double-N domino sets will have the same number of edges as there are dominoes in the set they model. For example, the following graph models a double-2 domino set:



As shown by Euler, for a circuit path to exist, a graph must be connected, and each node must have an even number of edges (not counting self-referential edges). It follows that the only double-N domino sets that will have a solution, are the sets where N is even (i.e. there will be no solution for double-1, double-3, double-5, etc).

Moreover, there have been a number of algorithms already written for walking such a graph. You will need to use one of them in this program (look into Fleury's algorithm or Hierholzer's algorithm).

Program Specs

Your program is to include the following functions (follow the naming conventions if you want your program to be graded):

- **dominoes(N : int)** – given an integer, returns a list of tuples containing all the tiles in the set of double-N, e.g. dominoes(2) returns some variation of [(2 , 2), (2 , 1), (2 , 0), (1 , 1), (1 , 0), (0 , 0)] – order does not matter; any helper function that you use should be an inner function
- **Eulers(L : (int * int) list)** – given a list of tuples, returns a reordered list of tuples that forms one of possible circular trains, e.g. given a list from the bullet above, the function may return [(2 , 1), (1 , 1), (1 , 0), (0 , 0), (2 , 0), (2 , 2)] (this is the function in which you will use Fleury's or Hierholzer's algorithm).
- **flip(L: (int * int) list)** – given a list of tuples that form a circular train, returns a list with tiles that are flipped where appropriate to make the loop self-evident, e.g. if flip is applied to [(2 , 1), (1 , 1), (1 , 0), (0 , 0), (2 , 0), (2 , 2)], it results in [(2 , 1), (1 , 1), (1 , 0), (0 , 0), (0 , 2), (2 , 2)]
- **solution(N: int)** – given an integer, generates a solution – ties all the functions given above in a functional manner; note that there are <u>no domino rings when N is odd</u> and you should return an empty list in such a case
- **listAsString(L : (int * int) list)** – given a list of tuples, returns a string representing that list
- **driver(F1, F2) N** – given **listAsString** and **solution** functions, as well as **N** - the initial highest number of dots for the set, returns a string representing a solution to the program, i.e. it should be enough to call this function to get the answer to the circular train question.

In addition, your program must fulfill the following requirements:

- it is to be written in a functional manner, i.e. no loops – only recursion, minimal side effects, if any, etc.
- any helper function should be nested inside the function it is supposed to be helping
- include comments – the header with your name, program name, date; function headers that describe the purpose of each function; code comments that describe the high level logic
- your program is to be contained within one file called **domino.sml**

Your program will be graded by running it through a test file, so it is extremely important that you meet all the specifications and naming conventions given above. The sample test file is provided in **Canvas**. It tests the program as a whole and it tests each function separately. If you do not get all the functionality in your program to work correctly, comment out appropriate test file portions and submit it through **Canvas** as well.

Executive summary

You are to write an executive summary similar to the ones you used to write in TCSS 305. It should be written as a txt file. In the file, describe what you have done to complete the assignment in 200-500 words. The word count is not strict, so do not worry about going slightly over; however, summaries that do not meet the minimum length requirement or are trivial in nature (representing little thought or effort) will not get full credit. You can share your personal experiences, things that particularly frustrated you about the assignment, things that particularly interested you about the assignment, etc. It is especially important that you document any difficulties you had with SML, the libraries, etc.