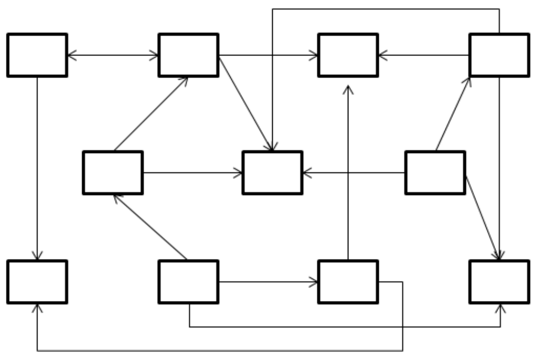
HW-11 CS330 50 points Due: December 12

We have a problem. We want to hang out with our friends, but we also need to find a way home (no Lyft or Uber as Covid-Mu+++ is spreading). So, we will create a graph of all establishments in town that you may venture out to. This graph will contain the names of the establishments and the cost (distance) of traversing amongst them (to keep things tractable we will limit the number of paths between establishments).

Part 1:
Your code will create a graph similar to that below. You can use this struct for it:

```
struct graphNode {
   char name[42];      // Establishment name
   int  arcCnt;        // Number of outgoing arc from this node
   int  weights[10];   // Weights of each outgoing arc from this node
   struct graphNode *arcs[10];
};
```

Note that each node is an establishment. The arcs connecting establishments will be the weighted paths you are allowed to take. So, the graph will look something like this:



To further simplify the creation of the graph, the data file (hw10.data) will consist of three parts: a list of establishments (one to a line) and a list of arcs. For example:

```
Applebees
GroundRound
BlueMoose
DrunkenNoodle
Home
STOP
Applebees BlueMoose 10
Applebees DrunkenNoodle 13
GroundRound Applebees 2
GroundRound DrunkenNoodle 7
GroundRound Home 52
STOP STOP 0
GroundRound
```

Notes:

- Establishments with multi-word names will be truncated into a single word (e.g. Ground Round -> GroundRound). So each line will have 1 word with a maximum character length of 100 (static arrays are OK and so is fscanf()).

- The word STOP will indicate the end this section.

- The list of arcs will contain 2 words per line and an integer. First will be the start-point of the arc, followed by the end point, followed by the cost/weight. All arcs will be one-direction (where the diagram shows bi-direction it is actually 2 one-direction arcs). The maximum number of arcs possible from any node is 10. But, not all nodes will have 10 outgoing arcs.

- The words STOP STOP 0 will indicate the end this section (to be compatible with the rest of this section).

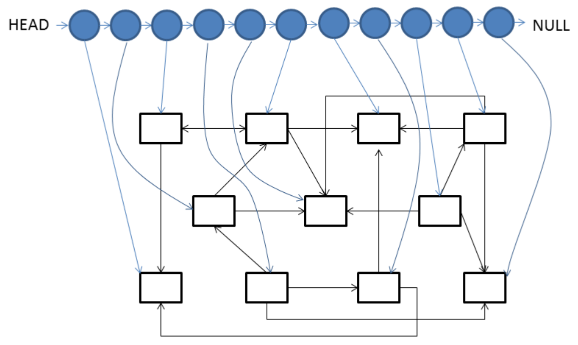
- The last section will be the name of one of the establishments. This will be the starting point for our journey home. Note that "Home" will also be a node in the graph.

Part 2:
To further simplify the creation of the graph, I suggest that you use (modify) your linked list code to build up an index. Trust me this will make the rest of the coding MUCH easier. You can use this struct for it:

```
struct listNode {
   char  name[42];            // Establishment name
   struct listNode *next;
   struct graphNode *graph;   // Pointer into graph
};
```

For example, the blue circles are your linked-list code modified such that the payload contains the name of the establishment and a pointer for that establishment into the graph.

You construct the linked-list and the nodes of the graph with the first part of the data file and then use the linked-list to find the nodes in the graph such that you can create the pointer references (arcs). And people say traversing a graph is hard. Not when you have pointers!



Part 2:
Take me Home One-Direction graph! Given a starting point for our journey home, implement a random walk algorithm (aka drunkard's walk). Given a node, the algorithm randomly selects the next path to take. Repeat until you get home. Accumulate the costs along the way. I suggest you implement this part as a separate function. The following code will generate a random number.

```
index = (random() % Start->arcCnt);   // Generates random number between 0 and Start->arcCnt
```

In oprder to make sure you generate the same results I do, you MUST seed your random number generator with 2021. The following code will seed the random number generator WITH 2021:

```
srand(2021);
```

Finally, your program must display the starting node, the ending node, and the accumulated cost.

REQUIREMENTS:
1. Your program must run on the Linux Mint machines in Engineering.

2. Your full name must appear as a comment at the beginning of your program.
3. Your source code must be named hw11-yourname.c
4. Your source MUST compile using "gcc hw11-yourname.c"
5. Failure to follow the above directions will result in loss of points.