CS215 Spring 2010 Program 4: Lost in the Grid

Primary learning objectives:
- Using Advanced ADTs (Trees, Linked Lists)
- Searching and Sorting ADTs

Ancillary learning objectives:
- Selecting an optimal implementation
- Using additional C++ STL

Expanding on our game motif from the previous exercise, you will be provided various input files which, if interpreted correctly will build a sort of 'map' where the Explorers from the previous assignment can maneuver.

Ships will travel between areas known as Sectors. Each Sector is connected to up to 4 other Sectors; one in each cardinal direction: N, S, E, and W (aka Up, Right, Left, and Down).

In addition to their connections, Sectors will have other attributes that may be useful in game play, as described in ***Figure 1***. You will need to create a Sector class capable of managing the links between Sectors and these attributes.

You will also need to establish one or more ADTs to hold references to these Sectors. You should choose ADTs which can facilitate the following actions on the Sectors:
- find Sector by Name
- find Sector via X & Y co-ordinate location
- find the Sector with the largest Size

Loading the Grid:

You will be given a file which contains an initial list of Sector names and links. The items will be presented in ‘grid order’. That is, the first entry will represent location 0,0 (top left) of the grid and each additional item on that line will represent another column in that row (increments x). Each new line in the file will begin a new row (increments y).

Items in the file are presented as pairs of elements. The first half of the pair is the name of the Sector, the second half is a single digit hexadecimal number. This number indicates which of the cardinal directions is connected to this Sector (i.e. which ways you can go from this Sector)

Thus a sample file of a 3x3 grid might look like:

```
a-1 6 b-2 A c-3 C
d-4 6 e-5 A f-6 D
g-7 2 h-8 A i-9 9
```

Where grid position 0,0 has a name of a-1, and a direction value of 0x6, and position 2,1 has a name of h-8 and a direction value of 0xA.

The direction value is a hexadecimal representation of a 4 bit binary number. Each bit in the number represents a direction (see ***Figure 2 & 3***).

Note that this only tells you that a link exists in a particular direction, not where it links. You must infer this information and link the proper Sector. In order to accomplish this, you will need first store the read sectors in a data structure which will allow you to access them via their X,Y grid position. Then you can iterate through each Sector, and link it to the appropriate other Sector(s) in the corresponding direction.

You should implement the following:
- a mechanism to find a Sector by Name
- a mechanism to find a Sector by its (X,Y) grid location
- a mechanism to print the Sectors in alphabetical order by name in a format similar to this example:

```
Name:d-4 Size:0 Outlets:E(m-5) S(q-7)
Name:f-6 Size:0 Outlets:N(k-3) S(i-9) W(m-5)
Name:h-8 Size:0 Outlets:E(i-9) W(q-7)
Name:i-9 Size:0 Outlets:N(f-6) W(h-8)
```

Several sample map data files will be provided on the class web site in the Examples area, along with sample code implementing various ADTs (including the BST from the book).

You may use any class from the C++ STL which we have already discussed in class, such as string, vector, stack and queue.

For this assignment submit your Sector class implementation, the code used to implement any ADTs you have used, your driver code, any additional sample maps you have used. Document which ADT(s) you are using any why.

Figure 1: Required Sector Attributes

Each Sector is comprised of several pieces of information:

Name	<string> name of Sector
Northern Link	Each directional link is a pointer to another Sector object. A NULL pointer indicates that there is no connection in this direction.
Eastern Link	
Southern Link	
Western Link	
Size	Size of the sector as an integer
Directions	*Single Hex digit containing direction information, not strictly needed, but may be useful.*

Size information will be provided in a separate file, and loaded at a later time.

Figure 2: Binary Coded Directions

Each sector can be entered/exited via one of 4 directions. If each available direction is expressed as a single bit, with 1 indicating that a path exists, and 0 meaning that none does. Then the available connections from a given sector can be represented as a single Hex digit.
For this assignment assume that the bit numbering starts at the North (or Up) position, and increments with the directions in a clockwise fashion as described below:

```
  N        1
W * E    8 * 2
  S        4
```

Hence, a direction value of 0xF, has all 4 directions linked to other Sectors, 0xA has only the West and East linked, 0x3 has the North and East linked, etc. And, of course, 0x0 implies that the Sector is not linked to any other Sector.

Figure 3: Binary and Hex

Binary is base 2: numbers range from 0 to 1
Decimal is base 10, numbers range from 0 to 9
Hexadecimal is base 16, numbers range from 0-15

Often shortened to just ‘Hex’, in C and many other programming languages, Hex values are usually prefaced by 0x.

The Bitwise logical AND and OR operators can be used to determine what bits are ‘on’ or ‘off’ in a particular value. For Example:

```
int x=0x13;
if (x & 0xA) printf("TEN ");
if (x & 1) printf("ONE ");
if (x & 11) printf(" TEN AND ONE");
if (x & 0xF) printf("none");
```

will print TEN ONE TEN AND ONE

BitPosition	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	1	1
Decimal	128	64	32	16	8	4	2	1
Hex	80	40	20	10	8	4	2	1

Decimal	Hex	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111