Computer Science 1 - Program 2
Grid Computing
Assigned: 1/30/2012
Due: 2/17/2012 11:55pm on Webcourses

Objective:

1) To use the Linked List data structure taught in class.
2) Practice insertion, deletion, search, and freeing dynamically allocated memory.

The Problem:

Grid computing is a term that refers to the combination of computer resources from multiple domains to reach a common goal. Each computer in the grid only communicates with a centralized server, which makes the grid easy to grow or shrink in size. This is useful for analyzing large amounts of data that would otherwise be too much work to do alone, and allows users to freely join and leave the network. SETI@Home, MilkyWay@Home, and Folding@Home are examples of programs that utilize grid computing to analyze massive amounts data.

As the lead programmer for NASA's new planetary exploration team, you are in charge of creating a program to analyze the massive amounts of survey data coming back from a series of state of the art probes that were launched earlier this year. You have realized that this data would be much more than your super computers at NASA are able to keep up with, and have decided to use grid computing to get volunteers to help. So far your system is able to send work to all of the computers in the grid, and gathers the results for each day in a text file.

You will need to keep track of a current list of the contributing users, a list of the discovered planets, and a list of the verified planets. A discovered planet requires 3 separate users to discover it before it is a verified planet. It is important to keep track of the number of GB each user has contributed, since the website will update the user rankings each day. One of the reasons users love volunteering is so that they can brag to all of their friends about their ranking on the website!

Implementation:

You must use the exact structures as follows:

```
typedef struct GeocentricCelestialReferenceFrame {
        int x;
        int y;
        int z;
} GCRF;

typedef struct DiscoveredPlanet {
        int usersConfirmed;
        int type;

        GCRF*coordinates;
        struct DiscoveredPlanet* next;
```

```
} DiscPlanet;

typedef struct VerifiedPlanet {
        int type;
        GCRF*coordinates;
        struct VerifiedPlanet* next;
} VerPlanet;

typedef struct UserWorkCompleted {
        char firstName[20];
        char lastName[20];
        int gbAnalyzed;
        int planetsDiscovered;
        struct UserWorkCompleted* next;
} UserWork;
```

An enum is also provided for the planet types:

```
typedef enum {
        NONE = 0,
        GAS = 1,
        ICE = 2,
        STORM = 3,
        BARREN = 4,
        TEMPERATE = 5,
        LAVA = 6,
        OCEANIC = 7,
        PLASMA = 8,
        UNKNOWN = 9
        } Planets;
```

Input File Specifications (GridComputing.in):

The input file will contain an integer n which will specify the number of user entries in the file. The number of user entries, n, will be followed by 3n lines, where each user entry consists of 3 lines: (1) the last name of the user followed by the first name, (2) the GB analyzed, (3) the type of planet discovered (if any), if a planet is discovered it will be followed by the position x, y, z.

The input file format will be as follows:

n
Last First
GBAnalyzed
planetsDiscovered
etc ...

Example Input:

```
16
Buchanan Sarah
45
0
```

```
Brown Charlie
1
0
Buchanan Ryan
22
0
Trump Ivanna
33
0
Buchanan Sarah
98
1 40 40 40
Duck Donald
12
0
Brown Charlie
1
0
Buchanan Ryan
22
0
Buchanan Sarah
78
2 95 89 73
Trump Ivanna
33
0
Trump Ivanna
33
5 12 13 14
Brown Charlie
33
5 12 13 14
Brown Charlie
78
2 95 89 73
Brown Charlie
98
1 40 40 40
Buchanan Ryan
98
1 40 40 40
Buchanan Ryan
33
5 12 13 14
```

For each user entry you will need to update the following:

1) Check if the user already exists in the UserWorkCompleted Linked List
    a) If the user does NOT exist, add that user to the list such that it is sorted by last name. Sort again by the first name if any duplicate last names are found.
    b) If the user does exist, update the user's GBAnalyed and planetsDiscovered
2) If a planet was discovered, see if the planet exists in the DiscoveredPlanet Linked List
    b) If the planet does NOT exist, add it to the list - the discovered planet list acts as a temporary queue and should not be sorted. New elements should be added to the end of the list. If the planet does exist, update its number of users confirmed.

a) If the number of users confirmed > 2, the planet should be removed from the discovered planets linked list and added to the verified planets linked list. You can assume that if a planet is added to the verified list users will no longer be working to discover a planet in that same area, meaning that it will not show up again in the input data.

b) All planets in the verified planet list should be sorted by distance from the Earth (where Earth is 0,0,0 in GCRF)

3) Once the you have finished reading from the file and completed updating all lists, you will need to print the following statistics:
   1) The list of users sorted by their name with the number of GBAnalyzed.
   2) The list of planets discovered with the current number of users confirmed.
   3) The list of verified planets, sorted by the distance from the Earth.

Output File Specifications (GridComputing.out):

The output file will contain the following format. Please use the partially completed fprintf calls to model your output.

```
fprintf(fp, "Users:\n");
fprintf(fp, "\t%s, %s - Analyzed:%d Discovered:%d\n", …);

fprintf(fp, "\nDiscovered Planets:\n");
fprintf(fp, "\t(%d,%d,%d) - Confirmed:%d Type:%d\n", …);

fprintf(fp, "\nVerified Planets:\n");
fprintf(fp, "\t(%d,%d,%d) - Type:%d\n", …);
```

Example Output:

```
Users:
      Brown, Charlie - Analyzed:211 Discovered:3
      Buchanan, Ryan - Analyzed:175 Discovered:2
      Buchanan, Sarah - Analyzed:221 Discovered:2
      Duck, Donald - Analyzed:12 Discovered:0
      Trump, Ivanna - Analyzed:99 Discovered:1

Discovered Planets:
      (95,89,73) - Confirmed:2 Type:ICE

Verified Planets:
      (12,13,14) - Type:TEMPERATE
      (40,40,40) - Type:GAS
```

Requirements:

1. Solution file name GridComputing.c
   a. Reads from GridComputing.in
   b. Writes to GridComputing.out
   c. CASE SENsITIVe, and no .txt files!!!!
2. Header comments, use of comments throughout, and whitespace for readability.
3. No system commands, for example no system("PAUSE")
4. Must use Linked Lists, and must use the Structures provided.
5. Must match the output format exactly.