**LongDistStationsDatabase, BusStation, and GasStation pseudocode**

**Drake, Jeffrey, and Joseph**

Class LongDistStationsDatabase:

Let constant *DATA\_FILE* refer to file LongDistStationsText.txt

Declare a list *lDStations* that will store BusStation objects (some of which will be GasStation objects, since GasStation is a child class of BusStation). This list will be initialized in the constructor.

Void method createConnection ( integer *i0*, integer *i1* ) :

Let variables *bs0* and *bs1* store the *i0*th and *i1*th stations in *lDStations*, respectively

Using the BusStation.connect method, connect *bs0* to *bs1* and *bs1* to *bs0*

Constructor:

Let string *line* store the first line of *DATA\_FILE*. (Eventually it will iterate through all the lines in the file.)

Initialize *lDStations* with a new, empty list of BusStations

Until *line* reaches a blank line:

Let 1-dimensional, 4-element string array *splitLine* store the 4 substrings of *line* representing its 4 columns, which are separated by 3 substrings “, ”

Let real numbers *lat* and *long* store the numbers parsed from the 0th and 1st elements of *splitLine*

Let boolean *isGasStation* be false if the 2nd element of *splitLine* is “City” and true if the 2nd element of *splitLine* is “Gas”. Throw an exception if the 2nd element of *splitLine* is neither “City” nor “Gas”.

Let string *name* store the 3rd element of *splitLine*

Create a BusStation *lDStation* using either the BusStation constructor or the GasStation constructor – depending on *isGasStation*’s value – and passing *lat*, *long*, and *name* to that constructor

Add *lDStation* to *lDStations*

Overwrite *line* with the next line in the file

*line* currently stores a blank line. Overwrite *line* with the next line in the file.

Until *line* reaches another blank line or the end of the file:

Let 1-dimensional, 2-element string array *splitLine* store the 2 substrings of *line* representing its 2 columns, which are separated by a space

Call this class’s createConnection method, parsing the 2 elements of *splitLine* as integers for the 2 arguments

Overwrite *line* with the next line in the file

Function toArray() returning a 1-dimensional array of BusStations:

Return *lDStations*.toArray()

Void method update():

Overwrite *DATA\_FILE* entirely as follows:

For each BusStation in *lDStations*:

Write a line containing its latitude; its longitude; either of the strings “Gas” or “City”, depending on whether it is a GasStation or a normal BusStation, respectively; and its name. Separate each of these four items with a comma and a space – the string “, ”.

Write a black line to indicate the separation between the two parts of the file

For ( integer *i* starting at 0 and incrementing through every index valid for *lDStations* (ending at one less than the size of *lDStatiions*) ):

For ( integer *j* starting one greater than *i* and incrementing through every index greater than i valid for *lDStations* ):

If the *i*th and *j*th stations in *lDStations* are connected, then write a line containing *i* and *j*, separated by a space

Void method addLDStation ( BusStation *lDStation* ):

If *lDStation*’s name contains a comma, then throw an exception

Otherwise, add *lDStation* to *lDStations*

Void method addNewLDStation taking four parameters:

Parameters:

Real numbers *latitude* and *longitude*

Boolean *isGasStation*

String *name*

If *name* contains a comma, then throw an exception

Otherwise:

Create a BusStation *lDStation* using either the BusStation constructor or the GasStation constructor – depending on *isGasStation*’s value – and passing the other three arguments to that constructor

Add *lDStation* to *lDStations*

Void method delete ( integer *index* ):

Let string *name* store the name of the *index*th station in *lDStations*

Delete the *index*th station in *lDStations*

For each remaining station *bs* in *LDStations*:

For each station in *bs*.connectedStations:

If that station’s name is identical to the variable *name* created at the beginning of this method, then remove that station from *bs*.connectedStations.

Class BusStation, a child class of the Place class:

Declare a string *name* that will store the name of the bus station, and declare a list *connectedStations* that will store other BusStation objects. Both will be initialized in the constructor.

Function getName():

The accessor function for *name*

Constructor ( real number *latitude*, real number *longitude*, string *name* ):

Call Place’s constructor with arguments *latitude* and *longitude*

Set this object’s *name* to the argument *name* of this method

Initialize *connectedStations* with a new, empty list of BusStations

Void method connect ( BusStation *station* ):

Add *station* to *connectedStations*

Function toString() overriding Place’s toString() function:

Return a string containing this BusStation’s *name*, *latitude*, and *longitude*, in that order, with *latitude* and *longitude* in parentheses and preceded by the word at, as in this example:

Abc Xyz Station at (0, 0)

Function hoursTo( Place *place*, Bus *bus* ) returning a real number:

Use Place’s milesTo function to find the distance (in miles) between this BusStation and *place.* Divide that distance by the cruising speed of *bus* to find the time required (in hours) to travel that distance in the *bus*. Return that time.

Class GasStation, a child class of the BusStation class:

This class has no fields besides those inherited from BusStation

Constructor ( real number *latitude*, real number *longitude*, string *name* ):

Call BusStation’s constructor with these arguments

Function toString() overriding BusStation’s toString() function:

Return BusStation’s toString() value preceded by string “GAS STATION: ”, as in this example:

GAS STATION: Abc Xyz Station at (0, 0)