Geographic Information System

Geographic information systems organize information pertaining to geographic features and provide various kinds of access to the information. A geographic feature may possess many attributes (see below). In particular, a geographic feature has a specific location.

There are a number of ways to specify location. For this project, we will use latitude and longitude, which will allow us to deal with geographic features at any location on earth. A reasonably detailed tutorial on latitude and longitude can be found in the Wikipedia at en.wikipedia.org/wiki/Latitude and en.wikipedia.org/wiki/Longitude.

We will employ public data obtained from the Geographic Names Information System (geonames.usgs.gov). Each data file consists of a list of ASCII records, each describing a single geographic feature with many attributes.

More precisely, each GIS record may contain the following fields in the indicated order (all are mandatory unless indicated otherwise):

Figure 1: Geographic Data Fields

Significance	**Type/Format**	**Comments**
Feature ID number (FID)	non-negative integer	unique identifier for this geographic feature
State alphabetic code	two-characters	US postal code abbreviation
Feature name	string	standard name of feature
Feature type	string	descriptive classification of feature
County name	string	county in which feature occurs
State number code	non-negative integer	numeric code for state
County number code	non-negative integer	numeric code for county
Primary latitude (DMS)	DDMMSS['N' | 'S']	feature latitude in DMS format
Primary longitude (DMS)	DDDMMSS['E' | 'W']	feature longitude in DMS format
Primary latitude (dec deg)	decimal number	feature latitude in decimal format
Primary longitude (dec deg)	decimal number	feature longitude in decimal format
Source latitude (DMS)	DDMMSS['N' | 'S']	latitude of feature source in DMS format, optional
Source longitude (DMS)	DDDMMSS['E' | 'W']	longitude of feature source in DMS format, optional
Source latitude (dec deg)	decimal number	latitude of feature source in decimal format, optional
Source longitude (dec deg)	decimal number	longitude of feature source in decimal format, optional
Feature elevation	integer	altitude above/below sea level, optional
Est. feature population	non-negative integer	estimated population of feature, optional
Federal status	string	???, optional
Cell	string	???

In the GIS record file, each record will occur on a single line, and the fields will be separated by pipe ('|') symbols. Some sample records are shown below. Note that it was necessary to wrap the lines in order to fit the data between the margins of this page. In the actual data files, each record would appear on a single line by itself.

GIS record files are guaranteed to conform to this syntax, so there is no explicit requirement that you validate the files. On the other hand, some error-checking during parsing may help you detect errors in your parsing logic.

Figure 2: Sample Geographic Data Records

Note that some record fields are optional, and that when there is no given value for a field, there are still delimiter symbols for it.

```
1495182|VA|Acre of Rocks|summit|Montgomery|51|121|371636N|0801608W|37.27667|-80.26889||||2270|||McDonalds Mill
1674451|VA|Agnew Hall|building|Montgomery|51|121|371329N|0802528W|37.22472|-80.42444||||||||Blacksburg
1481269|VA|Aiken Dam Hollow|valley|Montgomery|51|121|372056N|0801741W|37.34889|-80.29472|371932N|0801752W|37.32556|-80.29778||||McDonalds Mill
1674452|VA|Airport Acres (subdivision)|ppl|Montgomery|51|121|371238N|0802427W|37.21056|-80.4075||||2125|||Blacksburg
1462389|VA|Alburn High School|school|Montgomery|51|121|370341N|0802633W|37.06139|-80.4425||||||||Riner
1674453|VA|Alleghany Addition (subdivision)|ppl|Montgomery|51|121|370744N|0802350W|37.12889|-80.39722||||2120|||Blacksburg
1462398|VA|Alleghany Church|church|Montgomery|51|121|370726N|0801610W|37.12389|-80.26944||||||||Pilot
1462399|VA|Alleghany Church|church|Montgomery|51|121|370925N|0801519W|37.15694|-80.25528||||||||Ironto
1481276|VA|Alleghany Church|church|Montgomery|51|121|370930N|0802507W|37.15833|-80.41861||||||||Blacksburg
1674454|VA|Alleghany Heights (subdivision)|ppl|Montgomery|51|121|371509N|0802447W|37.2525|-80.41306||||2220|||Newport
1674455|VA|Alleghany School (historical)|school|Montgomery|51|121|370812N|0801519W|37.13667|-80.25528||||||||Ironto
1674456|VA|Alleghany School (historical)|school|Montgomery|51|121|370921N|0802430W|37.15583|-80.40833||||||||Blacksburg
1674457|VA|Alleghany Spring School (historical)|school|Montgomery|51|121|370706N|0801602W|37.11833|-80.26722||||||||Pilot
1462400|VA|Alleghany Springs|ppl|Montgomery|51|121|370741N|0801555W|37.12806|-80.26528||||1399|||Ironto
1481287|VA|Allen Hollow|valley|Montgomery|51|121|372001N|0801939W|37.33361|-80.3275|371849N|0801938W|37.31361|-80.32722||||McDonalds Mill
1462408|VA|Allen Hollow|valley|Montgomery|51|121|370849N|0801614W|37.14694|-80.27056|370937N|0801636W|37.16028|-80.27667||||Ironto
1674458|VA|Altoona School (historical)|school|Montgomery|51|121|370425N|0801805W|37.07361|-80.30139||||||||Pilot
1674459|VA|Ambler Johnston Hall|building|Montgomery|51|121|371323N|0802517W|37.22306|-80.42139||||||||Blacksburg
1481314|VA|Anderson Cemetery|cemetery|Montgomery|51|121|371519N|0802058W|37.25528|-80.34944||||||||McDonalds Mill
1674460|VA|Apperson Park (subdivision)|ppl|Montgomery|51|121|371425N|0802413W|37.24028|-80.40361||||2220|||Blacksburg
1674461|VA|Apple Acres (subdivision)|ppl|Montgomery|51|121|370923N|0802508W|37.15639|-80.41889||||2130|||Blacksburg
1674462|VA|Asbury Methodist Church|church|Montgomery|51|121|370800N|0802436W|37.13333|-80.41||||||||Blacksburg
1674463|VA|Atkinson Acres (subdivision)|ppl|Montgomery|51|121|370712N|0802436W|37.12|-80.41||||2060|||Riner
1481363|VA|Austin Hollow|valley|Montgomery|51|121|371621N|0801822W|37.2725|-80.30611|371617N|0801657W|37.27139|-80.2825||||McDonalds Mill
1462695|VA|Bain Chapel|church|Montgomery|51|121|370536N|0803150W|37.09333|-80.53056||||2116|||Radford South
1674464|VA|Bangs (historical)|ppl|Montgomery|51|121|370828N|0802407W|37.14111|-80.40194||||2029|||Blacksburg
1674465|VA|Barringer Hall|building|Montgomery|51|121|371334N|0802502W|37.22611|-80.41722||||||||Blacksburg
1481476|VA|Barringer Mountain|summit|Montgomery|51|121|370817N|0802809W|37.13806|-80.46917||||2342|||Blacksburg
1477097|VA|Basham|ppl|Montgomery|51|121|370210N|0802034W|37.03611|-80.34278||||2430|||Pilot
1481600|VA|Beeks School|school|Montgomery|51|121|371249N|0802423W|37.21361|-80.40639||||||||Blacksburg
```

Assignment:

You will implement a system that indexes and provides search features for a file of GIS records, as described above.

Your system will build and maintain several in-memory index data structures to support these operations:

- Retrieving GIS records matching given geographic coordinates
- Retrieving GIS records matching a given unique record ID
- Retrieving GIS records that fall within a given geographic region.
- Displaying the in-memory indices in a human-readable manner

You will implement a single C++ program to perform all system functions. Your system will incorporate a simple buffer pool to mediate disk traffic when servicing record queries.

Program Invocation:

The program will take the names of three files from the command line, like this:

```
GIS <GIS record file> <command script file name> <log file name>
```

If the GIS record file or the script file are not found the program should log an error message and exit. If a log file name is not specified, the program should echo an error message and exit.

Data and File Structures:

There is no guarantee that the GIS record file will not contain two or more distinct records that have the same geographic coordinates. In fact, this is natural since the coordinates are expressed in the usual DMS system. So, we will not treat geographic coordinates as a primary key.

The GIS records will be indexed by geographic coordinate, using a PR quadtree. The index entries will store a geographic coordinate and a collection of the file offsets at which matching records occur in the GIS record file. Since each GIS record occupies one line in the file, it is a trivial matter to locate and read a record given nothing but the file offset at which the record begins.

The GIS records will also be indexed by the FID, using a perfectly height-balanced BST. The index entries will store an FID and the file offset of the matching record. Since all of the possible records that must be indexed are known when the index is built (i.e., you will never add or remove records in this assignment), it is possible to build a BST that has the theoretically-minimal height, and that is what you will do.

When building the index structures, your program will make one complete pass through the GIS record file. Aside from where specific data structures are required, you may use any suitable STL component you like.

Each index object should have the ability to write a nicely-formatted display of itself to an output stream.

Note: your implementation will not simply store the complete GIS records in memory. The file on disk is only place that complete records will exist, aside from a small number of temporary objects created when servicing search requests.

A Note on Coordinates and Spatial Regions:

It is important to remember that there are fundamental differences between the notion that a geographic feature has a specific location (which may be thought of as a point) and the notion that each node of the PR quadtree corresponds to a particular sub-region of the coordinate space (which will usually contain many points).

In this assignment, coordinates of geographic features are specified as latitude/longitude pairs, and the minimum resolution is one second of arc. Thus, you may think of the geographic coordinates as being specified by a pair of integer values.

On the other hand, the boundaries of the sub-regions are determined by performing arithmetic operations, including division, starting with the values that define the boundaries of the "world". Unless the dimensions of the world happen to be powers of 2, this can quickly lead to regions whose boundaries cannot be expressed as integer values. So, it is important that you do not think of region boundaries, or dimensions, as being integer values.

In fact, there is an advantage to specifying the world boundaries so that the dimensions of the world are odd numbers. If we do that, then region boundaries cannot be integers. And, if geographic features only occur at integer coordinates, they can never fall on the boundary between two regions. Unfortunately it is not always convenient to pick the world boundaries this way, since there is often a natural symmetry in the coordinate system.

Your implementation should view the boundary between regions as belonging to one of those regions. The choice of a particular rule for handling this situation is left to you.

Since we cannot think of regions as having integer dimensions, we also do not need to restrict them to being square. On the other hand, such an assumption does simplify some issues, and so for this assignment we will always specify a square world.

Other System Elements:

There should be an overall controller that validates the command line arguments and manages the initialization of the indices. The controller should hand off execution to a command processor that manages retrieving commands from the script file, and making the necessary calls to carry out those commands.

Naturally, there should be a data type that models a GIS record. Since reading such records is a basic requirement for the system implementation, and it is good practice to explore basic C++ language features, you are explicitly required to implement operator>> for this data type.

There may well be additional system elements, whether data types or data structures, or system components that are not mentioned here. The fact no additional elements are explicitly identified here does not imply that you will not be expected to analyze the design issues carefully, and to perhaps include such elements.

Command File:

The execution of the program will be driven by a script file. Lines beginning with a semicolon character (`';'`) are comments and should be ignored. Each non-comment line of the command file will specify one of the commands described below. Each line consists of a sequence of tokens, which will be separated by single tab characters. A newline character will immediately follow the final token on each line. The command file is guaranteed to conform to this specification, so you do not need to worry about error-checking when reading it. The following commands must be supported:

`world<tab><westLong><tab><eastLong><tab><southLat><tab><northLat>`

> This will be the first command in the file, and will occur once. It specifies the boundaries of the coordinate space to be modeled. The four parameters will be longitude and latitudes expressed in DMS format, representing the vertical and horizontal boundaries of the coordinate space.
>
> It is likely that the GIS record file will contain records for features that lie outside the specified coordinate space. Such records should be ignored; i.e., they will not be indexed.

`what_is_at<tab>[-brief<tab>]<geographic coordinate>`

> Log all the data fields, properly labeled, in every GIS record that matches the given `<geographic coordinate>`. If the optional command switch `-brief` is used, log only the name field for each of the relevant GIS records.

`what_is<tab><FID>`

> Log all the data fields in the unique GIS record that matches the given `<FID>`.

`what_is_within<tab>[-brief<tab>]<geographic coordinate>tab<radius>`

Log all the data fields, properly labeled, in every GIS record whose location lies within a circle of `<radius>` centered at the given `<geographic coordinate>`. If the optional command switch `-brief` is used, log only the name field for each of the relevant GIS records. The `radius` will be specified in seconds.

`debug<tab>[location | FID ]`

Log the contents of the specified index structure in a fashion that makes the internal structure and contents of the index clear. It is not necessary to be overly verbose here, but it would be useful to include information like key values and file offsets where appropriate.

`quit<tab>`

Terminate program execution.

If a `<geographic coordinate>` is specified for a command, it will be expressed as a pair of latitude/longitude values, expressed in the same DMS format that is used in the GIS record files.

A sample command script is included in Figure 3 on page 7 of this document. As a general rule, every command should result in some output. In particular, error messages should be logged if searches yield no matching records.

Instrumentation:

Each index (or its aggregated container) must be instrumented so that it logs information about each search it performs. The information should identify each index record that is accessed during the index search, and should be written to the log file.

Log File Description:

Since this assignment will be graded by TAs, rather than the Curator, the format of the output is left up to you. Of course, your output should be clear, concise, well labeled, and correct. You should begin the log with a few lines identifying yourself, and listing the names of the input files that are being used.

The remainder of the log file output should come directly from your processing of the command file. You are required to echo each command that you process to the log file so that it's easy to determine which command each section of your output corresponds to. Each command should be numbered, starting with 1, and the output from each command should be well formatted, and delimited from the output resulting from processing other commands. A complete sample log will be posted shortly on the course website.

File Navigation in C++

Logically, a file is simply a sequence of bytes, each having a location or offset, just like the cells in an array. File stream objects in C++ maintain internal variables that store offsets into the associated file; these variables are called file pointers.

Input streams have a *get pointer* that stores the offset from which the next byte will be read from the file. The get pointer can be moved to any valid offset by using the `seekg()` member function. The current value of the get pointer can be obtained by using the `tellg()` member function.

Output streams have a *put pointer* that stores the offset to which the next byte will be written to the file. The put pointer can be moved to any valid offset by using the `seekp()` member function. The current value of the put pointer can be obtained by using the `tellp()` member function.

It is also possible to move either pointer a given number of bytes relative to its current location, either forward or backward within the file. Seeking to a negative offset or to an offset beyond the end of the file will produce unfortunate results.

Seeking to a location within the file and writing data will replace the corresponding bytes of the file with the new data. Seeking to the end of the file and writing data will cause the data to be appended to the end of the file.

More details and examples are given in the course notes.

Administrative Issues:

You will submit this assignment to the Curator System (read the *Student Guide*), where it will be archived for grading at a demo with a TA.

For this assignment, you must submit a gzip'd tar file containing all the source code files for your implementation (i.e., header files and cpp files). Submit only the header and cpp files. Submit nothing else.

In order to correct submission errors and late-breaking implementation errors, you will be allowed up to five submissions for this assignment. You may choose which one will be evaluated at your demo.

The Student Guide and link to the submission client can be found at: http://www.cs.vt.edu/curator/

Evaluation:

Shortly before the due date for the project, we will announce which TA will be grading your project and post signup sheets inside the McB 124 lab. You will schedule a demo with your assigned TA. At the demo, you will perform a build, and run your program on the demo test data, which we will provide to the TAs. The TA will evaluate the correctness of your results. In addition, the TA will evaluate your project for good internal documentation and software engineering practice.

Remember that your implementation will be tested in the McB 124 lab environment. If you use a different development platform, it is entirely your responsibility to make sure your implementation works correctly in the lab.

Note that the evaluation of your project will depend substantially on the quality of your code and documentation. See the Programming Standards page on the course website for specific requirements that should be observed in this course.

Pedagogic points:

The goals of this assignment include, but are not limited to:

- implementation of a BST template in C++
- implementation of a PR quadtree template in C++
- implementation of complementary internal and leaf node types to conserve memory
- design of appropriate index classes to wrap the containers
- design of appropriate data objects to store in each index
- understanding how to navigate a file in C/C++
- creation of a sensible OO design for the overall system, including the identification of a number of useful classes not explicitly named in this specification
- implementation of such an OO design into a working system
- incremental testing of the basic components of the system in isolation
- satisfaction when the entire system comes together in good working order

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the pledge statement provided on the Submitting Assignments page of the course website.

Figure 3: Sample Command Script

```
; Sample script
;
; Specify boundaries of coordinate space:
world 0810000W    0790000W    343000N     363000N
;
; Check the indices:
debug location
debug FID
;
; Perform some exact coordinate searches:
what_is_at  371349N     0802520W
what_is_at  -brief      370905N     0802542W
;
; Perform an FID search:
what_is     1472316
;
; Perform a radius search:
what_is_within    371349N     0802520W    10
;
; Exit
quit
```