Final Project Report

Name: Mohammed Bajaman

Date: 8th December 2022

Contents

[Running the project using g++ 3](#_Toc121430960)

[Note 3](#_Toc121430961)

[Code Overview 3](#_Toc121430962)

[GIS 3](#_Toc121430963)

[CommandProcessor 3](#_Toc121430964)

[NameIndex 4](#_Toc121430965)

[BufferPool 4](#_Toc121430966)

[Logger 4](#_Toc121430967)

[HashTable 4](#_Toc121430968)

[Screenshots 5](#_Toc121430969)

[References 7](#_Toc121430970)

# Running the project using g++

To compile **and** run the project, extract the files in Cygwin (or equivalent) and run the below command:



The headers and source files are located inside their respective folders. The command line arguments for running GIS.exe should be located in the same folder as GIS.exe

# Note

The *what\_is\_in*, *what\_is\_at*, *debug quad* and *debug world* commands have not been implemented due to time constraints. The buffer pool is implemented but hasn’t been setup to redirect searches appropriately (Search -> NameIndex -> Buffer pool -> Database).

# Code Overview

## GIS

This is the main file and begins the application, instantiating a logger and a Command processor

## CommandProcessor

The CommandProcess reads each line of the script file and parses it into a vector<string> with each item being one word/argument in the command function in the script. I did it this way thinking it will make it easier to access and manipulate the command arguments across the program.

The CommandProcessor then reads what kind of command was triggered and does the appropriate action. The CommandProcessor handles the following commands:

1. WorldCommand

Sets world boundaries

1. ImportCommand

Imports GIS Record file and puts valid entries into a database

1. WhatIsCommand

Checks if a given feature name and state abbreviation exists in the NameIndex and returns the full GIS record. Also stores it in the Buffer pool

1. DebugCommand

Handles debug for hash and pool. Debug for quad has not been implemented

The following commands have not been implemented

1. WhatIsAtCommand
2. WhatIsInCommand

The following functions are helper functions in CommandProcessor:

1. addCommand

Adds the command to a commandList

1. RunCommands

Executed commands in the commandList one by one

1. DMStoDDConverter

Converts DMS entry to degree decimals.

## NameIndex

The NameIndex uses a HashTable implementation to store all valid record name and state abbreviation and is supported by the following functions:

1. addToIndex

Adds entry to NameIndex which uses a hashTable implementation

1. searchIndex

Searched the NameIndex and returns the full GIS record

1. str

Pretty print the contents of the NameIndex

## BufferPool

The BufferPool stores recent search results and uses two data structures: a HashMap STL implementation and a LinkedList. The following functions are used by the BufferPool:

1. insert

Inserts the file offset and GIS record to the BufferPool

1. str

Pretty print the contents of the BufferPool

## Logger

The logger handles all input into the logfile and has several methods to do this. An instance is passed from GIS to CommandProcessor so script commands and lines along with results can be logged accordingly. The following functions are used by the Logger:

1. addLine

Adds a string and a new line to the log file.

1. addText

Adds a string without a new line to the log file.

1. addLine(vector<string>)

Takes a vector<string> and prepares a properly formatted string then adds it to the log file along with a new line.

1. addLineBreak

Creates a formatted line break and appends it to the log file.

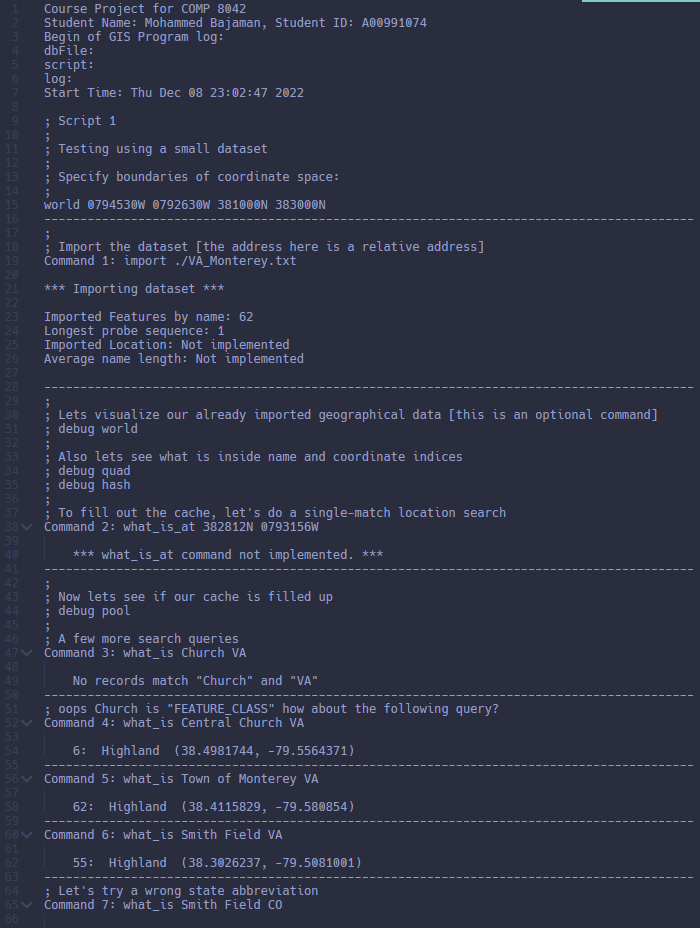
## HashTable

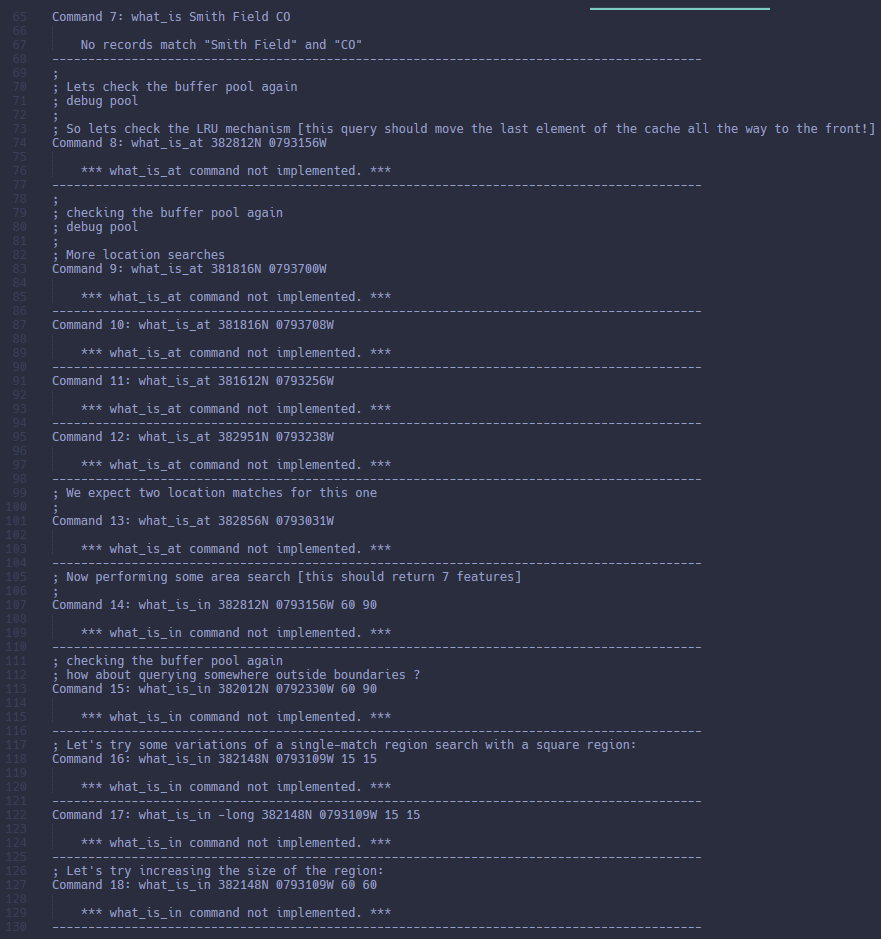
The HashTable uses an ElfStringHash algorithm to compute the initial hash value then runs it through a prime number function that find the nearest prime. This reduces the number of collisions that occur and ensures that entries are put into the HashTable. Additionally in the event that an insert command from NameIndex is unable to add an entry into the HashTable. The HashTable is then expanded and rehashed.

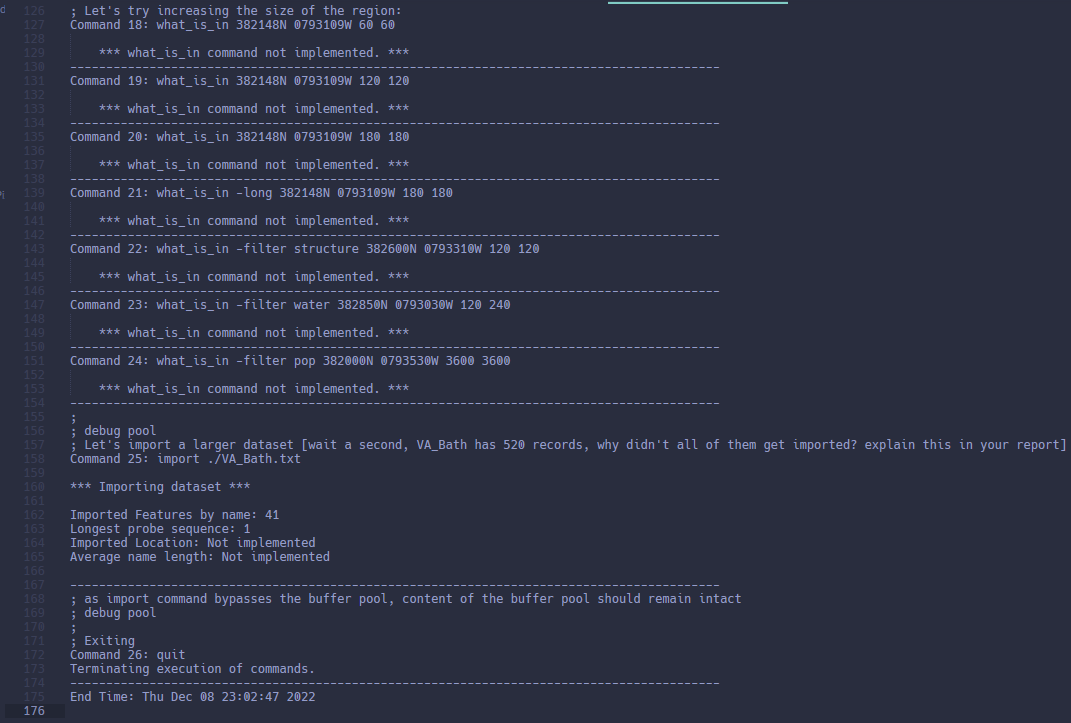
The HashTable utilizes Quadratic probing when dealing with collisions to find the next available bucket.

# Screenshots

The following screenshots are from running scrip01.txt with all debug commands commented out:







# References

**Buffer Pool – LRU Cache Implementation**<https://leetcode.com/problems/lru-cache/solutions/355310/lru-cache-simple-c-code-with-clear-comments/>

**DMS to DD Conversion**  
<https://gsp.humboldt.edu/olm/Lessons/GIS/01%20SphericalCoordinates/Reporting_Geographic_Coordinates.html#:~:text=Degrees%20Minutes%20Seconds%20(DMS),give%20their%20location%20in%20DMS>

**HashTable**  
COMP 8042 – Week 6 – hashtable-solution.cc   
Modified to use ElfHash  
<https://www.programmingalgorithms.com/algorithm/elf-hash/cpp/>

**NameIndex (Reading a file from a given file offset)**  
<https://stackoverflow.com/questions/5207550/in-c-is-there-a-way-to-go-to-a-specific-line-in-a-text-file>

**Nearest Prime Number**  
<https://www.geeksforgeeks.org/nearest-prime-less-given-number-n/>