Movie Database

Team: 11

Members: Giovany Calleja, Han Qiang,

Zohreh Ashtarilarki, Sabrina Diaz-Erazo

Welcome!

This project utilizes Hashing and Binary Search Trees in order to sort a database of different movies in order of Imdb code(unique) and the Title. The imdb code generally signifies the order in which a movie has been released, the higher the number, the more recent it is. For example:

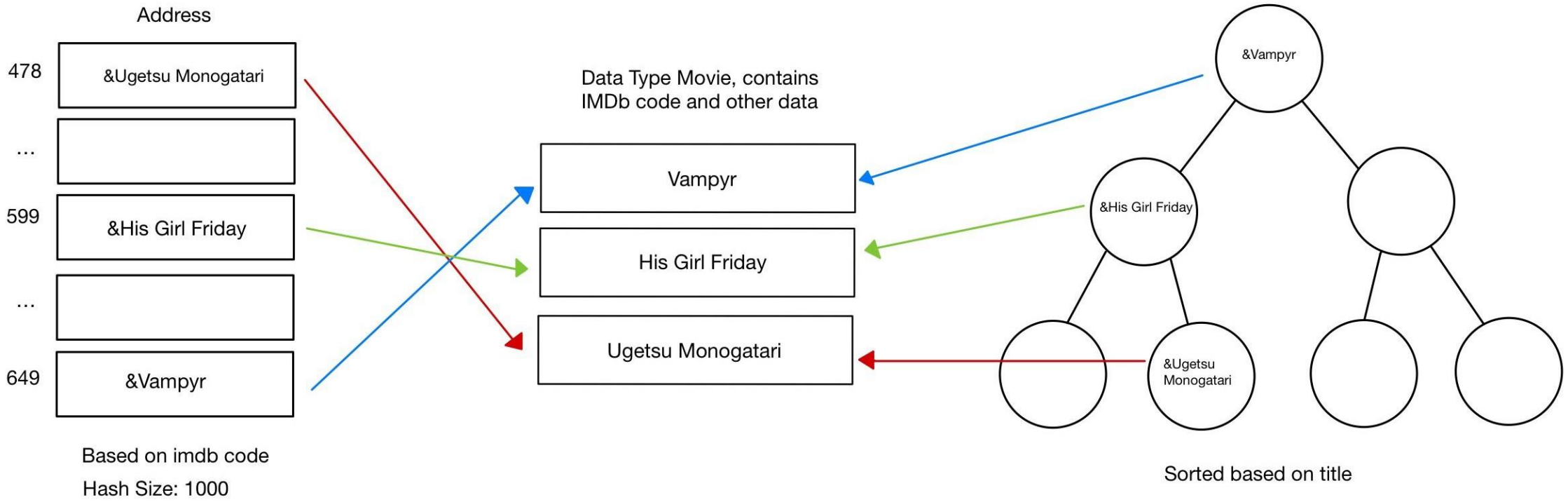
tt0068646 The Godfather; 1972... <-- Imdb Code has numbers 68646(Oldest)

tt0110912 Pulp Fiction; 1994... <-- Imdb Code has numbers 110912(Youngest)

tt0109830 Forrest Gump; 1994... <-- Imdb Code has numbers 109830

The project will also save movies into an output file for later usage. The outputted file will be named "NewMovies.txt"

Data Structure Diagram



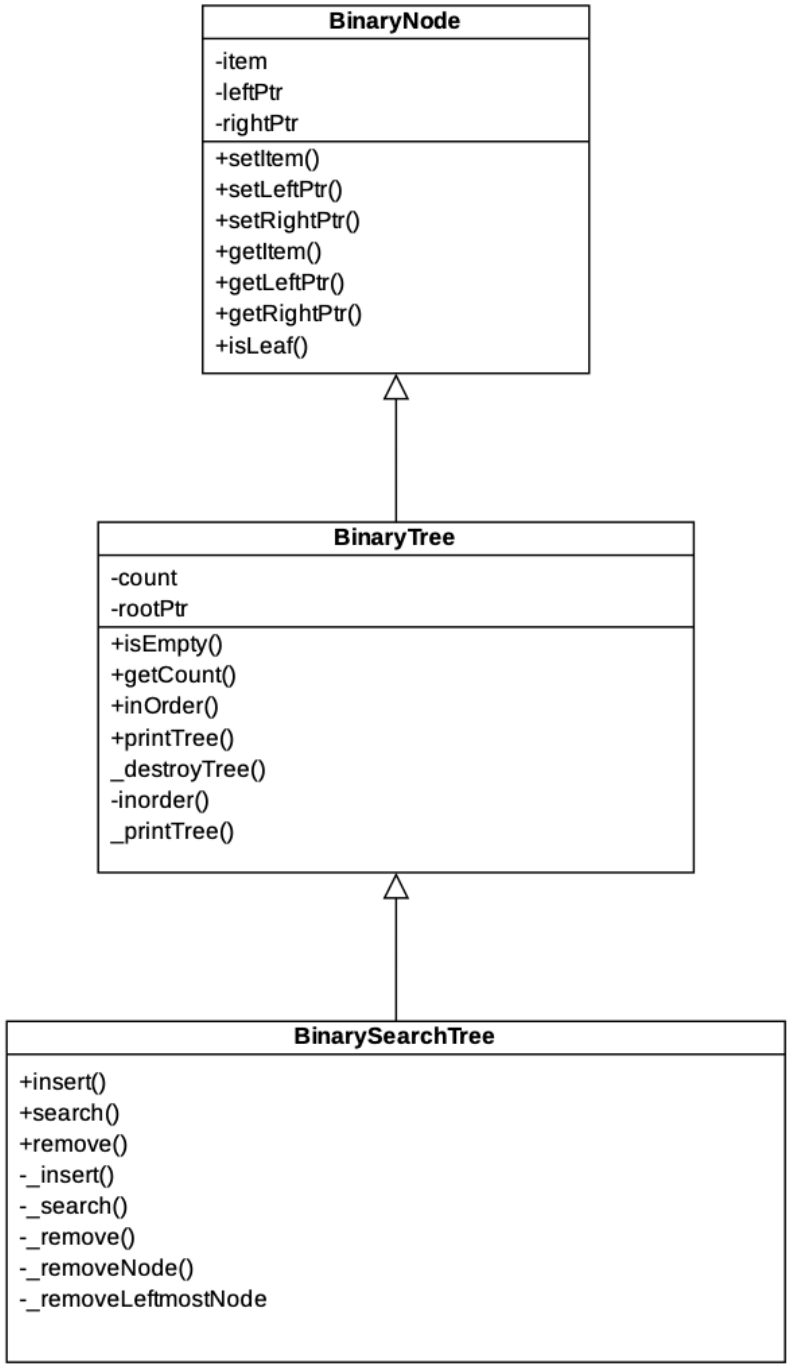
Data Members of Movie: imdb, title, director, genre, year, rating

Main

Includes various functions
For outputting, inserting, deletion,
And more.

Binary Search Tree

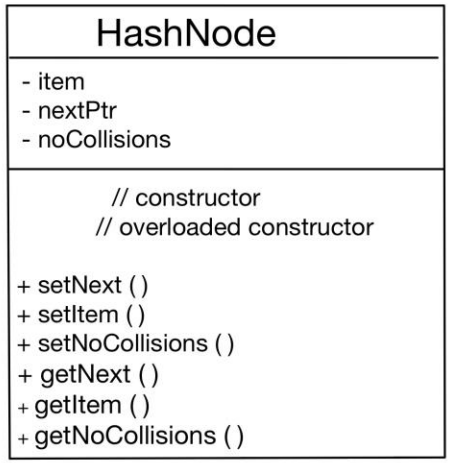
Contains:
Binary Node,
Binary Tree,
Binary Search Tree



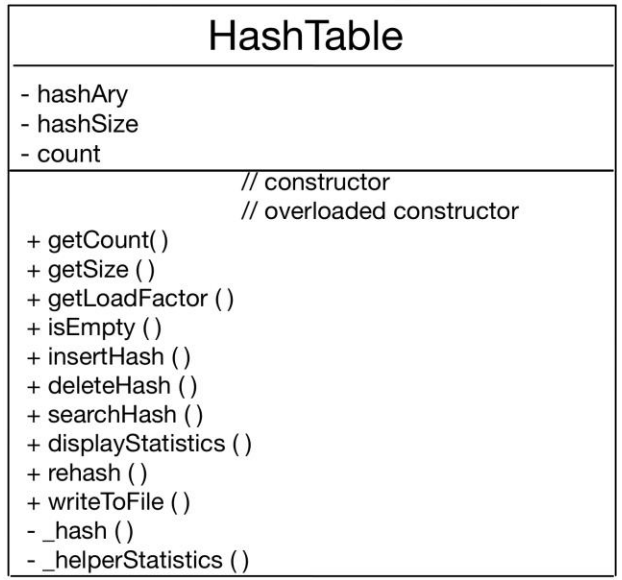
HashTable

Contains: HashNode, HashTable

UML Diagram for the HashNode class

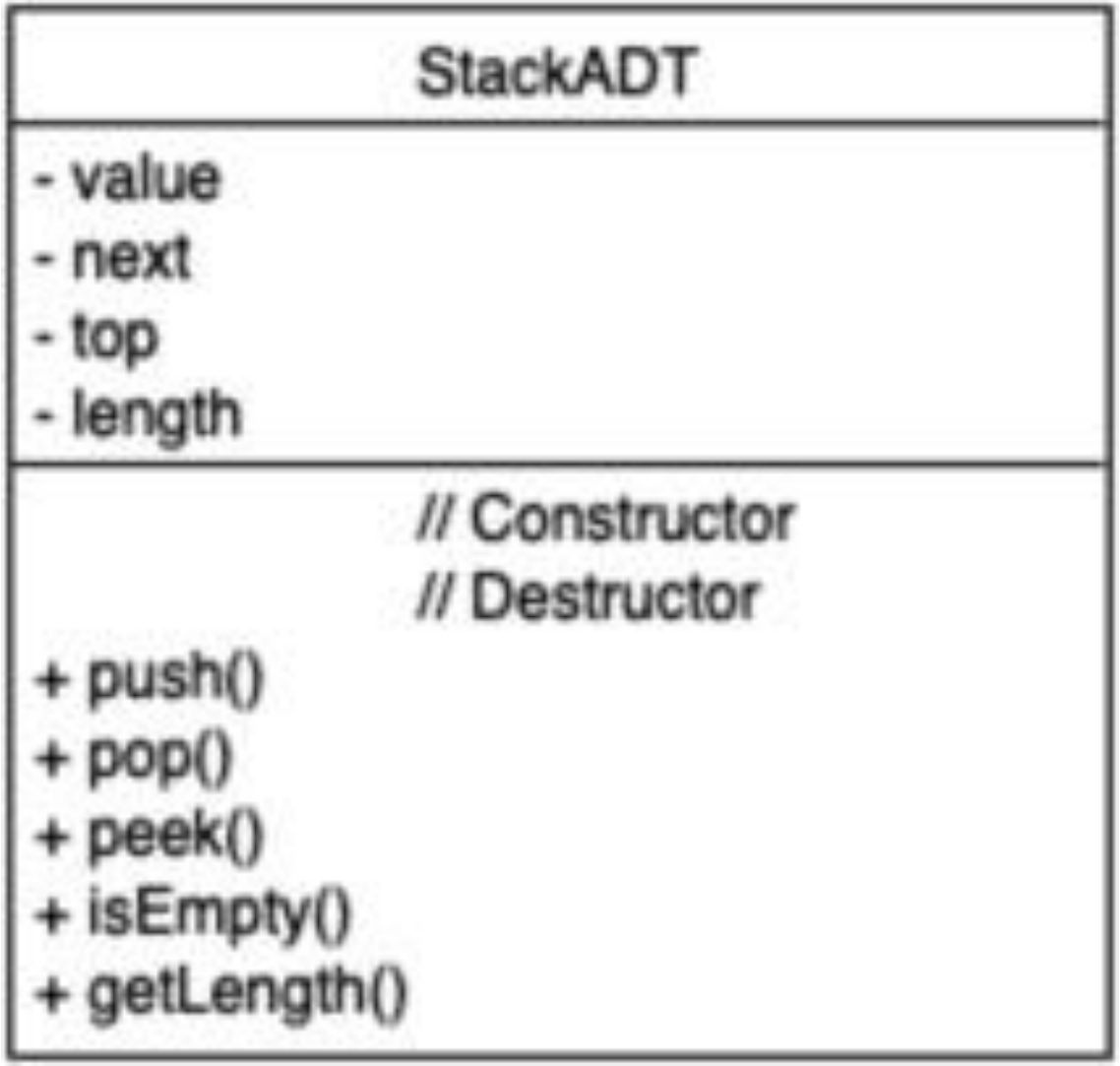


UML Diagram for the HashTable Class



Stack ADT

*used for undo delete



Structure Chart

Contains all functions used in main.

main calls the following functions:

- readFile
  - insert
  - insertHash
  - getLoadFactor
  - getSize
  - getPrime
  - rehash
- menu
- insertMovie
  - imdbValidate
  - titleValidate
  - validateGenre
  - validateDirector
  - calculateLongest
  - insert
  - compare
  - insertHash
- deleteManager
  - setImdb
  - deleteHash
  - remove
  - push
- undoDelete
  - isEmpty
  - pop
  - insert
  - insertHash
- searchManagerPrimary
  - setImdb
  - searchHash
  - vDisplay
- searchManagerSecondary
  - setTitle
  - search
  - vDisplay
- displayManager
  - header
  - inOrder
  - printTail
  - printTree
  - iDisplay
- help
- writeFile
  - writeToFile

Each Assignment

- Giovany Calleja: Unit 1(Team Coordination), File Documentation, Project Presentation
  - Mostly debugging and setting up how the output looks in the program. Wrote menu and some functions of main. Set up the file documentation and project presentation.
  - Wrote some functions for output in main
- Han Qiang: Unit 2(BST Algorithms), Unit 5(File I/O)
  - Created files BinaryNode.h, BinaryTree.h, BinarySearchTree.h
  - Created hash size function, and rehash function in Hash
  - Created readFromFile and saveToFile function in main
  - Debugging for Unit 2, 3, 5

- Zohreh Ashtarilarki: Unit 3(Hash List Algoritms)
  - Wrote HashNode.h and HashTable.h minus functions implemented from Unit 5. Fixed some errors in Main.
  - Debugging for Unit 5

- Sabrina Diaz-Erazo: Unit 4 (Screen Output)
  - Wrote Primary Search Manager in main.
  - Wrote Secondary Key Manager in main
  - Wrote Display Manager in main
  - Wrote Undo Delete function in Main
  - Wrote StackADT for the undo delete function

Hash Function

```
template<class ItemType>
int HashTable<ItemType>::_hash(string key) const
{
    int sum = 0;
    for (int i = 0; key[i]; i++)
        sum += key[i];
    return sum % hashSize;
};
```

Collision Resolution Method

```
if (table.getLoadFactor() >= 75)
   {
      int newSize = table.getSize();
      newSize = getPrime(newSize * 2);
      table.rehash(newSize);
   }
```

A new hashtable will be created through the rehash function, in which all items of the old hashtable will be added onto the new one. This will occur if the hashtable begins to run out of room for the movies. It will also ensure there are less collisions for the movies.