//Sucharita Das

//Desigh of Final Project movie serch

//Student ID 109103020

//11/01/2018

//Program Describtion : For many movie lovers, actors and directors, the annual Academy Awards are the highlight of the year.

//but we have to find which movies are the top movies, or who has received the most awards to reaD THE csv file.

//Here we are making our own database and design our database and analysis the data according to the requirements

**Input Requirements:**

To execute this program user need to enter the csv file where we have stored data. We will call actor-actress.csv , picture.csv and nomination.esv file. The data of the csv file are separated by comma . We need to search, sort, update, delete, add organize according to the requirements . Each line ends in a newline, not a comma.

**Output Requirements:**

Read the file name picture.csv file by inputting the name by user. Place this file in binary search to find the nominations of film. WE need to use binary search where it will find a particular single data which one we need to pull out. Binary search tree will search for particular one by one. It will start from the top and eventually go down and then will come back to whenever the search will be done.

Load ALL CSV files (it might take up to 3 min)

Display Records

Add a record

Find a record (by exact value)

Modify a record

Delete a record

Sort records by field

Search for a records (by partial match

Write to CSV file

**Problem Solution Discussion:**

BStree.h file we will declare BStree class. Then we will call those following function to fulfill all requirements

// Constructor

template <typename DATATYPE, typename KEYTYPE>

BSTree<DATATYPE, KEYTYPE>::BSTree()

// Free the node

template <typename DATATYPE, typename KEYTYPE>

// Add a node

template <typename DATATYPE, typename KEYTYPE>

void BSTree<DATATYPE, KEYTYPE>::addNode(KEYTYPE key, DATATYPE &data)

// Find a node

template <typename DATATYPE, typename KEYTYPE>

Node<DATATYPE, KEYTYPE> \* BSTree<DATATYPE, KEYTYPE>::findNode(KEYTYPE key, Node<DATATYPE, KEYTYPE> \* node)

//deleteNode (Private)

template <typename DATATYPE, typename KEYTYPE>

Node<DATATYPE, KEYTYPE> \* BSTree<DATATYPE, KEYTYPE>::deleteNode(Node<DATATYPE, KEYTYPE> \* aRoot,KEYTYPE value)

//print function

void DataSet::printInOrderBy(string key) {

this->indexedData[key]->printInorder();

// insert a single row into the DataSet

void DataSet::add(unordered\_map<string, string> row)

/\* delete a row by searching for an exact match on a field then replace it \*/

void DataSet::replaceByValue(string columnValue, string cellValue,unordered\_map<string, string> row)

/return titles

const vector<string> &DataSet::getTitles() const

// write a DataSet to a CSV output stream

void DataSet::writeDataSet(ostream &out)

//calling the write data screen function through write data

void DataSet::writedata()

//read a DataSet from a CSV input stream

DataSet readDataSet(istream &in)

// Destructor

template <typename DATATYPE, typename KEYTYPE>

BSTree<DATATYPE, KEYTYPE>::~BSTree()

**Output Requirements**

A detailed list or description of all outputs (to files) including a description of the data type and range of valid values for each output are follow

Test for binary tree search.

**Data Structures**

A description of choice of your data structures and justification. Of course the main data structure for the database is Binary Search tree. But for some internal operations you may have to use a few sub containers. So include a brief explanation for your choice. For example, "I have considered DS1, DS2, and DS3. Their pros and cons are summarized as follow... I choose DS1 over DS2 and DS3 because

User must fill in

if the root is the leaf, delete that leaf

otherwise if the leaf is not null

recursive call of the leaf's left

recursive call of the leaf's right

now delete the leaf

Problem Solution Discussion

Don't forget to set your key, Parent, then left or right

Based on the case you use you will add Node recursively to the left or right

First check if root is null

make a new node

set the key and data

set the root

Otherwise

Check to see if the key is < the leaf's key

if left is not null then

Add the node to the left (recursively)

Otherwise make a new node and attach it to the left

Check to see if the key >= leaf's key

if leaf's right is not null then

Add the node to the right (recursively)

Otherwise make new node and attach it to the right

**User Interface Scheme**

User interface scheme should show the menu items at top level and items in sub menus and how to navigate through menus.

Need to add make and read me file with the program.

**All Files:**

main.cpp

BSTDriver.cpp

BSTree.h

csv.cpp

DataSet.h

DataSet.h

MenuDisplay.cpp

MenuDisplay.h

BSTree.hpp

operators.h

Node.h

A binary search tree (BST) index will be created for each field in each table in order to optimize searches. Traversal of this tree from top to bottom will yield O(log(n)) searches.

**Status of program:**

The program runs successfully.

The program was developed and tested on Visual Studio and g++. It was

compiled, run, and tested on csegrid.ucdenver.pvt.

Main Function

Calling Menu Display and all options

Methods to Call All csv files and read data from it

Exit the program

Any option can be called after loading the file and can be executed, with database as parameter and the field to operate on

Read csv file and read it and put it in binary tree

Ask for a database and a file name and path to write the records in that file for that path

Partial search works with find By Partial Value function with display record

Sort function for a particular database through a field name and display the data on the screen

Select database and choose option to delete a particular record through the vector of binary tree

Put into vector to modify a particular record from any database – the file selection parameter comes first

Choose the picture or actor or nomination database, search for a record by exact value – findrecord function and display it

Display Function – select any file – Enter a number to display chunk by chunk – this function can be called any time

Choose either the Actor/Picture/Nomination