//Reading the File

//Open course information

file.open(courseInformation)

open.list(courseNumber)

open.list(courseTitle)

open.list(prerequisite)

count.lines(numberOfLines)

While numberOfLines !=0;

numberOfLines = TempNumber

If length (TempNumber != CourseNumber && courseTitle);

Break

Set Num and Pre to [0]

Set courseNumber to [Num] = courseInformation[0]

Set courseTitle[Num] = courseInformation[0]

//Keep adding to the list

courseInformation = Num++

If length of courseInformation is > 2;

prerequisite[Pre] set equal to courseInformation

//Create the object

Class Course

Public

String courseTitle;

String courseNumber +to\_string;

//Print course information

courseInformation.open

Class Course

If course =<1

Print No information found

Else

Print courseNumber

Print courseTitle

Print prerequisite

Return 0;

////Hash table

////Reading the file

Open “course.text”

Open “course.text’

If course.text == open{

While True;

HashTable=[courseName],[courseNumber], [prerequisite];

close”course.text”

}

Else {

Display error

}

Return 0;

//// The Hash Table

Class HashTable{

Public:

HashTable();

~HashTable();

String hash(string key, const int Table\_Size);

Void addItem(string courseName, string courseNumber, string prerequisite);

////Count number of items in bucket

Int numItemsIndex(int index);

//Prints item of each bucket

Void printTable();

Private:

Struct Node{

String courseNumber;

String courseName;

String prerequisite;

Node \* next;

Node() : courseNumber(“ ”), courseName (“ “), prerequisite (“ “), next(NULL)

};

Typedef struct Node\* Nodeptr;

Static const int Table\_size = 12;

Nodeptr table[table\_size];

}

////Print Table

table.printTable();

Return 0;

////Binary Search Tree

///Reading the file

Open “course.text’

If course.text == open

While True;

bst.insert(courseNumber)

bst.insert(courseName)

bst.insert(coursePrerequisites)

Else;

Display error

Return 0;

Class bstNode{

Public:

Int key;

bstNode \* left;

bstNode \* right;

bstNode\* parent;

}

If “courseNumber” is < key;

printf(node->right, node->left)

If courseNumber == Prerequisite;

printf(Prerequisite);

Elif “courseNumber” is > key;

printf(node->left)

If courseNumber == Prerequisite;

printf(Prerequisite);

////Menu

//set up csv function

Class csvReader;

String file

String delimiter

Public

csvReader filename, delimiter

Get data

Vector = getdata

File name

Datalist

Close file

Print “Welcome to the course planner”

While (True)

Int flag; //use the flag variable to select either true or false

String flag2;

Flag = 0;

Print “1. Load Data Structures;

Print “ 2. Print Course List.;

Print “ 3. Print Course;

Print “9. Exit;

Get user input

If input is 9;

exit;

Else If input is 4-8

Error invalid option;

Else if input is 1;

Open csv file

Sort in alphanumerically order

Print list of courses.

Else if input is 2;

Print what course do you want to know about

Get user input

Print course name number and prerequisites

////Evaluation

By creating a class the file name will be given. The program will find the file and open it. Once the file is open the program will start looking for strings that are in the file. There will need to be a variable named to save the string into. Then the program will search each line and save the information to the variable. When another line is searched the push back function will kick in and will save the information after the last line was saved. After all of the lines are ran the file will be closed and the information will be saved in the variable name. The runtime for the vector will be O(N). The runtime for the hash table will be O(N) The runtime for the bst will be O(N). The runtime for the menu will be O(N).

It appears that the Big O values are the same. I will choose the vector because the list will be on the small side. I think if we were doing courses for the entire college then a bst would be the best option and it would also sort the data with no need for additional work. The vector and hashtable will need to be sorted.

////Recommendation

I will recommend that the vector is used. It will save space because there will not be any unused slots for classes. It will still need to be sorted in order to get the correct order that is needed, however this method will maintain speed and memory.