# Struct Prerecs

//this will give us a structure to load prerequisites into

String preCourse

# Struct Course

//outlines the content of each course that will be stored in the vector of courses

String courseNumber

String courseTitle

Vector <Prerecs> coursePrerecs //this will set up the vector that the prerecs will be stored in

# Struct Node

// outlines the different nodes to store Courses in

Course course // creates a blank course

Node\* left // these create left and right sides

Node\* right

Node() // what a node comes standard with in its default constructor

Left = nullptr

Right = nullptr

Node( Course aCourse) // this is what an initialized node will come with

Node()

Course = aCourse

# Class BinarySearchTree

// this gives the Binary Search Tree something to live it. It assigns a single node of root to start populating the tree from, provides the ability to add a node and print nodes in order.

private:

Node\* root

void addNode(Node\* node, Course course)

void inOrder(Node\* node)

public:

BinarySearchTree()

virtual ~BinarySearchTree()

void InOrder()

void Insert(Course course)

void Remove(string courseName)

Bid Search(string courseName)

# BinarySearchTree()

// This creates the root of the BST and sets it to nullptr

Root = nullptr

# ~BinarySearchTree // Destructor

// this erases the BST

Delete root

# Insert (Course course)

// This will allow the creation of a node that represents a course and its contents. If the root of the BST is empty, put the course at the root. Otherwise add it to the BST.

if (root == nullptr) // if there is nothing assigned as the root

root = new Node(course) // root is equal to new node course

else //if there is something in the BST

this->addNode(root, course) //add this course

# addNode(Node\* node, Course\* course)

// this will add nodes to the bst based on their alphabetic / numeric value when compared to the existing node in the bst, if its been created at all.

if (node->course.courseNumber.compare(course.courseNumber) > 0)

if (node->left == nullptr)// if no left node

node->left = new Node(course)// this node becomes left

else

this->addNode(node->left, course)// else recurse down the left node

else// else

if (node->right == nullptr) // if no right node

node->right = new Node(course)// this node becomes right

else //else

this->addNode(node->right, course)// recurse down the right node

# Void InOrder()

// this sends the root to the actual print function, starting with the root and then traversing from the left side of the BST to the right. It will be sorted alphabetically, then within its alphabetic sorting, it will also be sorted numerically. It will only display the courseNumber and courseTitle, NOT prerequisites.

This->inOrder(root)

# Void inOrder(Node node)

//see above InOrder() comment

if (node != nullptr)

inOrder(node->left)

cout << node->course.courseNumber << “: “ <<node-> course.courseTitle << “ | “ << endl

# displayCourse(Course course)

/\* this displays the contents of each individual node in this format:

MAT123: Made up math class that’s been debunked |

No Prerequisites

OR

MAT122, MAT121, MAT(Insert number here)

\*/

cout << course.courseNumber << “: “ << course.courseTitle << “ | “ << endl

//if the course doesn’t have prerequisites…

if (coursePrerecs.size() == 0)

cout << “No Prerequisites.” <<endl;

//if the course does have prerequisites

else

for(int I = 0; I < coursePrerecs.size(); ++i)

if(I < coursePrerecs.size()-1)

cout << coursePrerecs.at(i) << “, “

else

cout << coursePrerecs.at(i) << endl;

return

# loadCourses(string somePath, BinarySearchTree\* bst)

// this loads parses the *text file*. I find this interesting because we haven’t actually, or I haven’t actually parsed a text file to extract its data. Going to give it my best shot.

Cout << “Loading TXT file “ << somePath

ifstream inputFile(“somePath\some\_file.txt”)

if(!inputfile.is\_open())

cerr << “Error opening file.” << endl

try

string line // this is to capture each line .

// while loop to capture each line

while(getline(some\_file.txt, line)//Iterator1

Course course // create a course structure

int iterator = 0// used to keep track of where we are in the line

stringstream ss(line) // get stringstream going to read the line

string info // this is to capture each course

// loop to capture each bit within the text file using a comma as a delimiter.

while( getLine(ss, info, ‘,’) // iterator 2

if(iterator == 0) // this is the course number

course.courseNumber = info

if(iterator == 1) // this is the full title

course.CourseTitle = info

if(iterator >=2) // this is where it starts to add prerecs to the vector of prerecs

course.course.Prerecs.push\_back(info)

Iterator++ // iterate the iterator

//end Iterator 2:: Meaning no more commas delimiting text in string stream

// insert the course into the binary search tree

bst->Insert(course)

// end Iterator 1:: Meaning time to move to the next line in the text file or there is no more lines in the text file

file.close() // close the text file once we’re done adding courses

# QualityControl(BinarySearchTree\* bst)

/\* The way this is supposed to work: compare two course listings simultaneously using two while loops and a for loop. First reference a node and name it cur and set it equal to root. While the current node isn’t a nullptr check to see if its got prerecs by examining its size. If its larger than zero go through a for loop to iterate through the vector of course prerecs. Point at another node, naming it comp, and setting it equal to root. Compare cur’s prerecs vector course numbers with comp’s courseNumbers and iterate down the tree as appropriate until the course is found. If it is, great! Iterate to cur->next and rinse and repeat the process. If its not, tell the user that they’ve got an invalid course prerequisite. Uses similar logic to the Search() function. \*/

Node\* cur = root

//While loop to iterate though all of the courses

While( cur != nullptr ) // iterator1

// if a course has course has course prerecs

If (if cur->course.coursePrerecs.size()>0)

// iterate through the list of course prerecs using coursePrerecs size

For (I = 0, I < cur->course.coursePrerecs.size (), i++) // iterator 2

Node\* comp = root

// Another while loop to make comparisons between cur and comp’s

While (comp != nullptr) // iterator 3

if (cur->course.coursePrerecs.at(i).compare(comp->course.courseNumber)== 0)

// if we find the prerec in the list of courses we get out of this while loop

return

//if we find that the prerec in the list of courses is smaller, alphanumerically, than the comparison, traverse down the left side of the tree

if (cur->course.coursePrerec.at(i).compare(comp->course.courseNumber) < 0)

comp = comp->left

//if we find that the prerec in the list of courses is smaller, alphanumerically, than the comparison, traverse down the right side of the tree

if (cur->course.coursePrerec.at(i).compare(comp->course.courseNumber) > 0)

comp = comp->right

//else if we have reached the end of the tree and still nothing found

else

cout << “Course not found in list of courses :: QualityControl Error” << endl;

return 0 // exit back to the menu

// end iterator 3

//end iterator 2:: Meaning either iterate to the next prerec OR if there are no more prerecs iterate to the next course to examine whether or not it has prerecs

Cur = cur->next

//end iterator 1 :: Meaning its time to examine another course if applicable or be done with Quality control

Cout << “quality control checks good! << endl;

Return

# Search(string courseNumber)

// This searches the BST for a specific course number.

Node\* cur = root

while (cur != nullptr) // keep looping downwards until bottom reached or matching bidId found

if (cur->courseNumber.compare(courseNumber) == 0) // if match found, return current course

return cur->course

if (courseNumber.compare(cur->course.courseNumber) < 0) // if course is smaller than current node then traverse left

cur = cur->left

else // else larger so traverse right

cur = cur->right

Course course

DisplayCourse(course)

Return

# main( )

String filePath, case, csearch

Int choice = 0

BinarySearchTree\* bst

Course course

bst = new BinarySearchTree()

clock\_t ticks

clock\_t tTicks

### \\ menu

while ( choice != 9 )

cout << "Menu:" << endl

cout << " 1. Load Courses" << endl

cout << " 2. Get Course Information" << endl

cout << " 9. Exit" << endl

cout << "Enter choice: "

cin >> choice;

### case 1 Load courses

cout << “What file are you looking for?”<< endl

cin << filePath

ticks = clock()

tTicks = clock()

bst = loadCourses(filePath, bst)

ticks = clock() – ticks

cout<< “Courses loaded”<< endl

cout << “time to load courses:” << ticks << “Clock ticks” << endl

cout << “time it took to load courses in seconds: << ticks\*1.0 / CLOCKS\_PER\_SEC << endl

ticks = clock()

QualityControl(bst)

cout << “Courses loaded and are valid” << endl

ticks = clock() – ticks

cout << “time to run QC:” << ticks << “Clock ticks” << endl

cout << “time it took to run QC in seconds: << ticks\*1.0 / CLOCKS\_PER\_SEC << endl

tTicks = clock() – tTicks

cout << “time to Load courses and QC them:” << tTicks << “Clock ticks” << endl

cout << “time it took in seconds: << tTicks\*1.0 / CLOCKS\_PER\_SEC << endl

break

### case 2 Get Course Information

cout << “Do you want to see all courses(a) or do you want to see just one course(b)?” << endl

getLine(cin, case)

#### if (case == “a”) // display all courses

ticks = clock()

bst->InOrder()

ticks = clock()-ticks

cout << “time it took to display sorted courses” << ticks << “ clock ticks” << endl;

cout << “ time it took in seconds: “ << ticks\*1.0 / CLOCKS\_PER\_SEC << endl;

return

#### if ( case == “b”) // search for a single course

cout << “Which course are you looking for? << endl;

getLine(cin, csearch))

cout << “searching for “ << csearch << endl;

ticks = clock()

bst->Search( csearch )

ticks = clock()-ticks

cout << “time it took to find and display the course” << ticks << “ clock ticks” << endl;

cout << “ time it took in seconds: “ << ticks\*1.0 / CLOCKS\_PER\_SEC << endl;

return

// Iterator 1 End

Cout << “Good bye!” << endl