PART 1:

struct Row {

vector<string> values;

};

struct Course {

string courseId;

string name;

std::vector<std::string> courseIds;

};

HashMap<string, Course> courseMap;

vector<Row>

loadFile(string csvPath):

vector<string> linesFromFile;

fp = openfile(csvPath);

while (fp != EOF):

line = getLine();

linesFromFile.push\_back(line);

vector<Row> rows;

int i = 0;

while (i < linesFromFile.size()) :

Row row(linesFromFile[i]); //Row is create from comma separated strings as strings are pushed into vector

rows.push\_back(row);

loadCourse(vector<Row> rows):

for (unsigned int i = 0; i < rows.size(); i++) {

hash = hashOf(courseID);

auto it = courseMap.find(hash);

Course course = nullptr;

if (it != courseMap->end()) {

course = it->second;

}

else {

course = new Course();

(\*courseMap)[hash] = course;

}

course->courseId = rows[i][0];

course->name = rows[i][1];

for (int j = 2; j < rows[i].size(); j++) {

hash = hashof(rows[i][j]);

auto it = courseMap->find(hash);

if (it == courseMap->end()) {

Course dependentCourse =new Course;

dependentCourse->name = rows[i][j];

(\*courseMap)[hash] = dependentCourse;

course->courseIds.push\_back(dependentCourse->name);

}

else {

course->courseIds.push\_back(it->second->name);

}

}

}

void printCourse(courseMap, courseId)

{

size\_t hash =HashOf(courseId);

Course course = courseMap[hash];

if (course != nullptr) {

cout << course->courseId << " " << course->name << "\n";

cout << "Prerequisite Subjects are:\n";

int i = 0;

for (auto& x : course->courseIds) {

cout << i << " " << x << "\n";

i++;

}

}

cout << "\n";

}

PART 2:

Create pseudocode for a menu.

while (choice != 9) {

cout << "Menu:" << endl;

cout << " 1. Load Course" << endl;

cout << " 2. Print Course List" << endl;

cout << " 3. Print Course" << endl;

cout << " 9. Exit" << endl;

cout << "Enter choice: ";

cin >> choice;

switch (choice) {

case 1:

{

courseMap = loadCourse(csvPath);

break;

}

case 2:

{

listCoursesInSortedOrder(courseMap);

break;

}

case 3:

{

printCourse(courseMap, course);

break;

}

case 9:

cout << "Bbye";

return 0;

}

}

PART 3:pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order

for each course in hashmap :

sorted\_set.insert(course.courseName);

for each name in sorted\_set:

cout << name;

PART 4:

Evaluate the run - time and memory of data structures that could be used to address the requirements.

O(Reading file) = MAX Character in file. i.e size of file

O(Course creation) = NUmber of unique courses present in the file. This will use extra memory for hashmap to save course.

COST PER LINE OF CODE: NUmber of lines in file \* Max number of chars in each file (For row creation)

Number of rows \* hashlookup for courses + N courses creation + N insertions in Hashmap

+ N\*x insertion in vector for depenent course

PART 5:

DS Comaparisons:

vector:

insertion O(1) unless it resizes

lookup O(1)

deletion O(N) as it will move the data after every delete

Order of insertion is preserved

Hashtable:

Insertion: O(1)

LookUP: O(1)

Deletion: O(1)

Order of insertion is not preserved

Tree:

Insertion: O(logn)

LookUp: O(logn)

Deletion: O(logn)

Order is not preserved, but the out is in sorted order

Based upon above evaluation, we will use Hashmap and vectors because these 2 are fastest for storing and lookups