John Pitkin
CS 203 – Computer Organization
Professor Liew

Project 2 Report

The most important feature of my project is the CourseBinaryTree: a binary search tree organized lexicographically by Course name. This structure provides a simple means of adding and referencing Course information; however, in my project, Courses are also referenced by other data structures.

The CourseBinaryTree is made of CourseBTNodes, all of which contain a unique Course pointer, a reference to the right CourseBTNode (if present), and a reference to the left CourseBTNode (if present). But the true effectiveness of the project's design shines through when the CourseBInaryTree is used in combination with the Department, Degree, and Student abstract data types. All Departments, Degrees, and Students require a means of referencing relevant Course data: Departments have available Courses, Degrees have required Courses, and Students have completed Courses.

Every Course read from any Department, Degree, or Student file is first inserted into the CourseBinaryTree with as much Course information as possible. Only then, the Courses read from a particular Department, Degree, or Student file are added to their respective structures. Every Department, Degree, and Student refer to their Courses with Course pointers, just as the CourseBinaryTree refers all loaded Courses with Course pointers.

Since I knew all Departments, Degrees, and Students would need their own collection of Course pointers and would all operate on their Course pointers in nearly the same way, I decided I would make a single system for organizing collections Course pointers that could be used for all three ADTs. My answer was the Vector and Element abstract data types.

The Vector abstract data type is the same as any other Vector or ArrayList, except that instead of listing integers or strings, my Vector contains Elements. Each Element has a data member for a Course pointer and a data member for an Element pointer named disjunct. The Department and Student ADTs contain conjunct Course collections, while Degrees have both conjunct and disjunct Course requirements. Every Element contains a Course pointer; however, only a Degree's disjunct Course requirements utilize the Element's disjunct pointer.

My Vector abstract data type lists Elements in contiguous memory, keeps track of the number of Elements in contiguous memory, and saves the Vector's capacity determined by the allocated heap space. On the other hand, disjunct Elements are stored in Linked Lists stemming from the Elements referrable from the contiguous Vector space. Simply put, I have a Vector of Elements and Element linked lists. The Vector does not keep count of the disjunct Elements and it does not need to for the purpose of my project.

All Vector functions I wrote had to take into account the possible presence of disjunct Elements; however, once the functions were complete, I was able to reuse the same code for all Departments, Degrees, and Students; thus, saving me a lot of time by designing a more general means of Course pointer organization.

Additionally, different Departments, Degrees, and Students need references to the same Courses but within differing collections of Course pointers. The Vector and Element ADTs are a far better alternative than linking Courses to other Courses directly.

Each Course consists of the Course's name, and if loaded from a Department file, the Course's title and prerequisite Courses. In addition to being able to access Courses from Department and Degree data structures, the Course ADT contains the name of the Department it belongs to and the names of the Degree programs it is a part of. This information may seem redundant, since Departments and Degrees already have references to their Courses, but I found that many functions were far easier to write and that the program would run faster at the expense of some additional used memory.

The Department, Degree, and Student ADTs each have their own function for loading their information from lines of a formatted file. Departments and Degrees loaded from formatted files are stored in DepartmentVector and DegreeVector respectively. All Students loaded from Student files are stored in a StudentBinaryTree, similar to the CourseBinaryTree, but organized lexicographically by Student name. DepartmentVector and DegreeVector are searched iteratively while the StudentBinaryTree is searched and traversed recursively.

I made a string linked list implementation for every Course's Degree list and prerequisite list that allows me to add, remove, and clear.

One neat feature of my project that I am personally proud of lies within my formatted file processing: I load in lines from a given file into a string buffer of a set size that's local to the function and then allocate only the heap space needed for the number and size of the file's lines.

The rest of my project consists of more functions for parsing formatted files, processing user command input, and utilizing my data structures to perform all the requested functions. All functions from the first and second phases of the project are functional including:

- finding and printing a Course's information
- finding and printing a Degree's information
- showing the effect of completing a given Course
- adding a Course to a Department (and the CourseBinaryTree)
- showing the Courses a Student still needs to complete for their Degree program
- showing the available Courses for a Student based off of their completed Courses and the prerequisites of Courses in their Degree program
- removing a Course from a Degree program
- removing a Course from its Department, all Degrees, all prerequisite requirements, the CourseBinaryTree (and Students' completed Courses)
- the 'p' commands including
    - finding and printing different Course information
    - finding and printing a Department's information
    - finding and printing a Degree in a different format
    - finding and printing a Student's information
- exiting the program

My project does not free all of the dynamically allocated memory. I have included Department, Degree, and Student files that I created.