Module 3 – 5:

Open/read file

```
Use fstream to open file
IF return value == -1
    Return error
IF file found
    Read each line
    If parameters in line < 2
        Return error
    ELSE
        Read parameters
Close file
```

Create course objects

```
init struct Course
Loop through file
WHILE file != EOF
    FOR first course && second course
        Create temp
    IF third course
        Add to course
```

Create course objects

```
init vector course
Create hashtable class
Loop through file
While file != EOF
    For first course && second course
        Create temp
    If third course
        Add to course
```

Tree and Nodes

```
CREATE root -> null
CREATE insert
    IF root == null
        Current = root
    ELSE IF course number < than root
        Insert left
        IF left = null
            Add course number
        ELSE
            IF course number < leaf node
                Insert left
            IF course number > leaf node
                Insert right
    ELSE
        IF course number < leaf node
            Insert left
        IF course number > leaf node
            Insert right
```

Print course information

```
If input key
    Print course information
    For each prerequisite
        Print prerequisite information
    Store data in hash table
```

Module 6:

Menu Pseudocode

```
CREATE int userInput == 0

WHILE userInput != 4

cout << Welcome to the course planner

cout << 1) Load Data Structure

cout << 2) Print Course List

cout << 3) Print Course

cout << 4) Exit

cout << What would you like to do?

cin >> userInput

CREATE Switch(userInput)

        Case 1: Load Data Structure

                break

        Case 2: PRINT Alphanumeric Course List

                break

        Case 3: PRINT Course Title

                PRINT Prerequisites

                break

        Case 4: end program

                break

        Default: PRINT Error
        break
```

Alphanumeric Order

```
CREATE printSorted(courses)
CREATE int partition
    low = first element
    high = last element
    mid = (low + (high – low) / 2)
CREATE quicksort
    mid = 0
    low = start
    high = end
    IF start >= end
        Return
    Recursive quicksort
CREATE void displayCourse
cout << course information
loop through vector
    display courses
CREATE inOrder void BST::inOrder
    IF (node != Null)
        Search left leaf
        Node -> left
```

Search right leaf

Node -> right

Advantages, Disadvantages, and Recommendations:

Hash tables are fast. The information the user would be looking for would be searched and printed fast because it would be attached to a key. This seems the most promising so far, but hash functions sometimes produce duplicate keys which can cause collisions. This is one of the downsides of using hash tables. Implementing good hash functions that won't duplicate keys is not worth the time for this program as I feel there is a better data structure to use.

When it comes to vectors, elements can be inserted and removed very easily. A big disadvantage to vectors is the memory consumption. When it comes to this program, I don't see that as an issue as we're just storing courses. If there are more courses added overtime though, loading speed can become an issue.

One thing to look out for when using binary search trees is they must stay balanced to work correctly. When working properly, they are very efficient when dealing with inserting items and deletion. BST are also great at fast sorting, like hash tables. A huge part of this program is displaying lists in alphanumeric order. Because of this I think binary search trees are the way to go as it is the easiest to implement without missing out on any of the hash tables and vectors advantages.