Initialize Libraries and Define Structures:

1. Start by loading the necessary libraries and headers for text parsing.
2. Define a Course structure that contains:
   1. courseID: a string to hold the course identifier.
   2. courseName: a string for the course name.
   3. preCount: an integer to track the number of prerequisites.
   4. preList: a string to store a comma-separated list of prerequisite courses.
   5. Constructor to initialize courseID and courseName as empty strings, preCount to zero, and preList to an empty string.

Define the Binary Tree Class:

1. Create a BinaryTree class to store and manage courses:
   1. Inside it, define a Node structure, which contains:
      1. A Course object.
      2. Pointers to the left and right child nodes (left, right).
   2. The BinaryTree class will have:
      1. Use root pointer that points to the root node of the tree.
      2. Use method printCourse() to display course information.
      3. Use constructor BinaryTree() to initialize the tree.

Main Function:

1. In the Main() function, perform the following:
   1. Create an instance of the BinaryTree class, named courseTree, which will store Course objects.
   2. Prompt the user to input the path of a CSV file containing course information. If no input is provided, default to a pre-configured file location.
   3. Call the txtParser(filePath) function, passing the CSV file path to read and parse the course data.
   4. After parsing, call validateList(courseTree) to ensure all prerequisites listed in the tree are valid.
   5. Request a course ID from the user for searching, store it in a variable userSearch.
   6. Use printCourse(userSearch) to search for the course in the tree and display its details.

txtParser(filePath):

1. In the txtParser() function:
   1. Open the file at the provided filePath.
   2. Loop through the file rows until reaching the end of the file (EOF):
      1. For each row, if the first and second columns contain values:
         1. Set the first column as courseID and the second as courseName in the Course struct.
      2. Loop through subsequent columns to identify any prerequisites:
         1. Increment preCount for each prerequisite found.
         2. Append the prerequisite name to the preList string, separated by commas.
   3. Once all rows are processed, return a list of parsed course data (tempList).

searchList(courseID):

1. The searchList() function is responsible for searching a course by its ID:
   1. Create a temporary node tempCourse of type Node.
   2. Use a hashing mechanism to set tempCourse to the node at the hashed location of courseID.
   3. Loop through the list of courses in the tree:
      1. If a match is found between the courseID and the current course in the node, assign that course to tempCourse.
   4. Return tempCourse, which now holds the matched course details.

printCourse(courseID):

1. The printCourse() function searches the binary tree and displays a course's information:
   1. Start by setting tempCourse to the root node of the tree.
   2. Traverse the tree, comparing the courseID provided with the courseID in the current node:
      1. If a match is found, output the courseID, courseName, and list of prerequisites (preList).
      2. For each prerequisite listed, recursively call printCourse() to display the prerequisite course details.
   3. If courseID is less than the current node's courseID, move to the left child node.
   4. If courseID is greater, move to the right child node.
   5. Continue traversing until the matching course is found or the tree is fully searched.

validateList():

1. The validateList() function ensures all prerequisites listed for each course are valid:
   1. Start by initializing a valid flag to true.
   2. Loop through each course in the binary tree:
      1. If the valid flag is set to false, exit the loop.
      2. For each course, check the prerequisites listed in preList:
         1. Use searchList() to look up each prerequisite in the binary tree.
         2. If a prerequisite is not found (i.e., the course ID is empty), set valid to false.
   3. Once all courses and prerequisites are checked, return the valid flag to indicate whether all prerequisites are valid.