```
Algorithm: import(path)
 Input: string path

 inputFile ← Open file at path

 If inputFile is not open then
   Print "We can’t open the file you have provided me with, which is  " + path
   Return -1

 Declare line as string
 Skip the header line in inputFile
 bookCount ← 0

 While there is a new line in inputFile do
   line ← Read line from inputFile
   bookData ← Empty vector
   currentField ← ""
   insideQuotes ← false
   Declare currentField as string

   For each character ch in line do
     If ch is a double quote then
       Toggle insideQuotes
     Else if ch is a comma and not insideQuotes then
       Add currentField to bookData
       Clear currentField
     Else
       Append ch to currentField

   If currentField is not empty then
     Add currentField to bookData

   If bookData has fewer than 7 elements then
     Print "Error has occurred. Invalid format in line: " + line
     Continue

   title ← bookData[0]
   author ← bookData[1]
   isbn ← bookData[2]
   publicationYear ← bookData[3]
   category ← bookData[4]

   Try
     totalCopies ← Convert bookData[5] to integer
```

```
    availableCopies ← Convert bookData[6] to integer
  Catch conversion error then
    Print "Error has occurred. Invalid total or available copies in line: " + line
    Continue

  Try
    pubYearInteger ← Convert publicationYear to integer
  Catch conversion error then
    Print "Error: Invalid publication year in line: " + line
    Continue

  Book* newBook ← New Book(title, author, isbn, pubYearInteger, totalCopies, availableCopies)
  Increment bookCount by 1

  categoryStream ← New stringstream with category
  Node* currentNode ← libTree.getRoot()

  While there is a categoryToken in categoryStream separated by '/' do
    childNode ← libTree.getChild(currentNode, categoryToken)

    If childNode does not exist then
      Insert categoryToken as a child of currentNode in libTree
      childNode ← libTree.getChild(currentNode, categoryToken)

    currentNode ← childNode

  Add newBook to currentNode.books
  Increment currentNode.bookCount by 1

  Node* parentNode ← currentNode.parent
  While parentNode is not null do
    Increment parentNode.bookCount by 1
    parentNode ← parentNode.parent

Print bookCount + " records have been imported successfully."
Close inputFile
Return bookCount
```

```
Algorithm: exportData(path)
 Input: string path

 outputFile ← Open file at path for writing

 If outputFile is not open then
   Print "We can’t open the provided file, which is  " + path
   Return

 Write "Title,Author,ISBN,Publication Year,Total Copies,Available Copies" to outputFile

 nodes_stack ← New MyVector
 Push libTree.getRoot() onto nodes_stack

 While nodes_stack is not empty do
   Node* currentNode ← Pop last element from nodes_stack

   For each book in currentNode.books do
     Book* book ← currentNode.books[i]
     Write book details (title, author, ISBN, publication year, total copies, available copies) to outputFile

   For each child in currentNode.children do
     Push child onto nodes_stack

 Close outputFile
 Print "Data has been exported successfully to: " + path
```

```
Algorithm: findAll(category)
 Input: string category

 Node* categoryNode ← libTree.getNode(category)

 If categoryNode is null then
   Print "Category is not found!"
   Return

 libTree.printAll(categoryNode)
```

```
Algorithm: findBook(bookTitle)
  Input: string bookTitle

  Node* current ← libTree.getRoot()
  Bool flag ← false

  For i from 0 to current.children.size - 1 do
    Book* book ← libTree.findBook(current, bookTitle)

    If book is not null then
      book.display()
      flag ← true
      Break the loop

  If flag is false then
    Print "The book was not found!"
```

```
Algorithm: addBook()
 Input: none

 Declare title, author, isbn, publicationYear, category as strings
 Declare totalCopies and availableCopies as integers

 Prompt "Enter Title: " and store in title
 Prompt "Enter Author: " and store in author
 Prompt "Enter ISBN: " and store in isbn
 Prompt "Enter Publication Year: " and store in publicationYear
 Prompt "Enter Category: " and store in category
 Prompt "Enter Total Copies: " and store in totalCopies
 Ignore newline in input buffer
 Prompt "Enter Available Copies: " and store in availableCopies
 Ignore newline in input buffer

 Try to convert publicationYear to an integer and store in pubYearInteger
   If conversion fails then
     Print "Invalid publication year entered. Please enter a number."
     Return

 Book* newBook ← new Book(title, author, isbn, pubYearInteger, totalCopies, availableCopies)

 Node* categoryNode ← libTree.getNode(category)

 If categoryNode is null then
   Print "Category '" + category + "' not found. Creating new category."
   categoryNode ← libTree.createNode(category)

 Append newBook to categoryNode.books
 libTree.updateBookCount(categoryNode, 1)

 Print title + " has been successfully added to the catalog."
```

```
Algorithm: editBook(bookTitle)
 Input: string bookTitle

 Node* currentNode ← libTree.getRoot()
 Book* book ← libTree.findBook(currentNode, bookTitle)

 If book exists then
   Do
     Prompt user for detail to edit (1: Title, 2: Author, 3: ISBN, 4: Publication Year, 5: Total Copies, 6: Available Copies, 7: Exit)
     choice ← user's choice

     Switch choice do
       Case 1:
         Prompt and set newTitle
         If newTitle is not empty then
           book.title ← newTitle
         Print "Title is now updated!"

       Case 2:
         Prompt and set newAuthor
         If newAuthor is not empty then
           book.author ← newAuthor
         Print "Author is now updated!"

       Case 3:
         Prompt and set newISBN
         If newISBN is not empty then
           book.isbn ← newISBN
         Print "ISBN is now updated!"

       Case 4:
         Prompt and set newPublicationYear
         If valid then
           book.publication_year ← newPublicationYear
         Print "Publication year is now updated!"

       Case 5:
         Prompt and set newTotalCopies
         If valid then
           book.total_copies ← newTotalCopies
         Print "Total copies are now updated!"

       Case 6:
         Prompt and set newAvailableCopies
         If valid then
           book.available_copies ← newAvailableCopies
         Print "Available copies are now updated!"

       Case 7:
         Print "Exiting the edit menu."

       Default:
         Print "Invalid choice. Please enter a number between 1 and 7."

   While choice is not 7
 Else
   Print "Book cannot be found."
```

```
Algorithm: borrowBook (bookTitle)
  Input: string bookTitle

  Declare name and id as strings
  Node* currentNode ← libTree.getRoot()
  Book*  book ← libTree.findBook(currentNode, bookTitle)

  If book exists and book.available_copies > 0 then
    Prompt "Enter Borrower's name: " and store in name
    Prompt "Enter Borrower's id: " and store in id

    borrower ← null
    For i from 0 to borrowers.size - 1 do
      If borrowers[i].name = name and borrowers[i].id = id then
        borrower ← borrowers[i]
        Break

    If borrower is null then
      borrower ← new Borrower(name, id)
      Append borrower to borrowers list

    Append borrower to book.currentBorrowers
    Append book to borrower.books_borrowed
    Decrement book.available_copies by 1

    Print "Book '" + bookTitle + "' has been successfully issued to " + name + " (ID: " + id + ")."
  Else
    Print "Book not found or no copies available!"
```

```
Algorithm: returnBook(bookTitle)
 Input: string bookTitle

 Node* currentNode ← libTree.getRoot()
 Book* book ← libTree.findBook(currentNode, bookTitle)

 If book exists then
  Prompt "Enter borrower's name: " and store in name
  Prompt "Enter borrower's id: " and store in id

  flag ← false
  For i from 0 to book.currentBorrowers.size - 1 do
   borrower ← book.currentBorrowers[i]

   If borrower.name = name and borrower.id = id then
    Remove borrower from book.currentBorrowers
    Increment book.available_copies by 1

    For j from 0 to borrower.books_borrowed.size - 1 do
     If borrower.books_borrowed[j] = book then
      Remove book from borrower.books_borrowed
      Break

    Print "Book has been successfully returned."
    flag ← true
    Break

  If flag is false then
   Print "Borrower's information does not match any current borrower for this book."
 Else
  Print "Book cannot be found!"
```

```
Algorithm: listCurrentBorrowers(bookTitle)
 Input: string bookTitle

 Node* root ← libTree.getRoot()
 Book*  book ← libTree.findBook(root, bookTitle)

 If book exists then
   For i from 0 to book.currentBorrowers.size - 1 do
     Print i, book.currentBorrowers[i].name, "(ID: " + book.currentBorrowers[i].id + ")"
 Else
   Print "Book cannot be found!"
```

```
Algorithm: listAllBorrowers
  Input: bookTitle

  Node* root ← libTree.getRoot()
  Book* book ← libTree.findBook(root, bookTitle)

  If book exists then
    Print "All borrowers of " + bookTitle + ":"
    For i from 0 to book.allBorrowers.size - 1 do
      Print book.allBorrowers[i].name, "(ID: " + book.allBorrowers[i].id + ")"
  Else
    Print "Book cannot be found!"
```

```
Algorithm: listBooks(borrower_name_id)
  Input: string borrower_name_id

  Create stringstream ss initialized with borrower_name_id
  Declare string variables name and id
  Use getline to read from ss into name (up to the first comma)
  Use getline to read from ss into id (the remainder after the comma)

  name ← Remove leading whitespace from name
  name ← Remove trailing whitespace from name
  id ← Remove leading whitespace from id
  id ← Remove trailing whitespace from id

  Print "Books borrowed by" + name + “(ID: “ + id + “) are listed below:”, newline

  flag ← false


  For i ← 0 to borrowers.size() - 1 do
      If borrowers[i].name = name AND borrowers[i].id = id then
          Call borrowers[i].listBooks()
          flag ← true
          Break


  If flag = false then
      Print "Borrower with name 'name' and ID 'id' cannot be found!"
```

```
Algorithm: removeBook(bookTitle)
 Input: string bookTitle

 Node* currentNode ← libTree.getRoot()
 Book* book ← libTree.findBook(currentNode, bookTitle)

 If book exists then
  Prompt "Are you sure you want to delete the book '" + bookTitle + "' from the catalog? (yes/no): " and store in confirm

  If confirm = "yes" then
   stack ← new MyVector
   Push currentNode onto stack
   flag ← false

   While stack is not empty and flag is false do
    Node* node ← stack.pop()

    For i from 0 to node.books.size - 1 do
     If node.books[i].title = bookTitle then
      Delete node.books[i]
      Remove book from node.books
      flag ← true

      Node* parentNode ← node
      While parentNode exists do
       Decrement parentNode.bookCount by 1
       parentNode ← parentNode.parent
      Print "Book '" + bookTitle + "' has been removed from the catalog."
      Break

    If flag is false then
     For each child in node.children do
      Push child onto stack

   If flag is false then
    Print "Failed to remove the book from the catalog."
  Else
   Print "Book removal has been canceled."
 Else
  Print "Book cannot be found!"
```

```
Algorithm: addCategory(category)
  Input: string category

  libTree.createNode(category)
  Print "Category has been added!"
```

```
Algorithm: findCategory(category)
 Input: string category

 Node* categoryNode ← libTree.getNode(category)

 If categoryNode exists then
   Print "Category '" + category + "' is found!"
 Else
   Print "Category '" + category + "' is not found!"
```

```
Algorithm: removeCategory(category)
 Input: string category

 Node* categoryNode ← libTree.getNode(category)

 If categoryNode exists and categoryNode has a parent then
  booksRemove ← categoryNode.books.size
  nodes_stack ← new MyVector
  Push categoryNode onto nodes_stack

  While nodes_stack is not empty do
   Node* currentNode ← nodes_stack.pop()
   booksRemove ← booksRemove + currentNode.books.size

   For each child in currentNode.children do
    Push child onto nodes_stack

  libTree.updateBookCount(categoryNode.parent, -booksRemove)
  libTree.remove(categoryNode.parent, categoryNode.name)
  Print "Category '" + category + "' removed!"
 Else if categoryNode does not exist then
  Print "Category '" + category + "' not found!"
 Else
  Print "Cannot remove the root category!"
```

```
Algorithm: editCategory(category)
  Input: string category

  Node* oldCategoryNode ← libTree.getNode(category)

  If oldCategoryNode exists then
    Prompt "Enter new category name: " and store in newCategory
    Node* newCategoryNode ← libTree.createNode(newCategory)

    For i from 0 to oldCategoryNode.books.size - 1 do
      Book* book ← oldCategoryNode.books[i]
      Append book to newCategoryNode.books

    libTree.remove(oldCategoryNode, category)
    Print "Category is now updated to " + newCategory + "!"
  Else
    Print "Category cannot be found!"
```