

```

1 #include <iostream>
2 #include <string>
3 #include <stdexcept>
4 #include <conio.h>
5 #include <windows.h>
6 #include <unordered_map>
7 #include <queue>
8 #include <vector>
9 #include <climits>
10 #include <algorithm>
11 #include <fstream>
12 #include <sstream>
13
14 using namespace std;
15
16 class LoginSystem
17 {
18 private:
19     string username;
20     string password;
21
22 public:
23     void showWelcomeScreen()
24     {
25         system("cls");
26         cout << "\n\n";
27         cout << "\\t\\t\\t===== ";
28         cout << "\\n\\n\\t\\t\\tWELCOME TO LIBRARY MANAGEMENT SYSTEM";
29         cout << "\\n\\n\\t\\t\\t===== ";
30         getch();
31     }
32
33     void authenticateUser()
34     {
35         const string validUsernames[] = {"treena", "takia", "ariina"};
36         const string validPasswords[] = {"123456", "123456", "123456"};
37         const int numUsers = sizeof(validUsernames) / sizeof(validUsernames[0]);
38
39         while (true)
40         {
41             try
42             {
43                 system("cls");
44                 cout << "\n\n";
45                 cout << "\\t\\t\\t===== ";
46                 cout << "\\n\\n\\t\\t\\tLOGIN PANEL";
47                 cout << "\\n\\n\\t\\t\\t===== ";
48                 cout << "\\n\\n\\n ENTER USER NAME: ";
49                 cin >> username;
50                 cout << "\\n\\n ENTER PASSWORD: ";
51                 password = "";
52                 for (int i = 1; i <= 6; i++)
53                     {
54                         password += getch();
55                         cout << "*";
56                     }
57                 bool authenticated = false;
58                 for (int i = 0; i < numUsers; i++)
59                     {
60                         if (username == validUsernames[i] && password == validPasswords[i])
61                             {
62                                 authenticated = true;
63                                 break;
64                             }
65                     }
66
67                 if (authenticated)
68                     {
69                         cout << "\\n\\n\\n\\t\\t\\tCONGRATULATIONS! LOGIN SUCCESSFUL";
70                         cout << "\\n\\n\\n\\t\\t\\tLOADING";
71                         for (int i = 1; i <= 6; i++)
72                             {
73                                 Sleep(100);
74                                 cout << ".";
75                             }
76                         Sleep(100);
77                         system("cls");
78                         cout << "\n\n";
79                         cout << "\\t\\t\\t===== ";
80                         cout << "\\n\\n\\t\\t\\tMAIN MENU";
81                         cout << "\\n\\n\\t\\t\\t===== \\n";
82                         break;
83                     }
84                 else

```

```

85         {
86             throw invalid_argument("Invalid login credentials");
87         }
88     }
89     catch (const invalid_argument& e)
90     {
91         cout << "\n\n\n\t\t\t" << e.what();
92     }
93     catch (const exception& e)
94     {
95         cout << "\n\n\n\t\t\tAn error occurred: " << e.what();
96     }
97     getch();
98     getch();
99 }
100 }
101 };
102
103 struct Book
104 {
105     int id;
106     string title;
107     string author;
108     bool isBorrowed;
109     Book* next;
110 };
111
112 struct Member
113 {
114     int id;
115     string name;
116     Book* borrowedBooks;
117     int borrowedCount;
118     Member* next;
119 };
120
121 class BookList
122 {
123 private:
124     Book* head;
125     const string bookDataFile = "books.txt";
126
127 public:
128     BookList() : head(nullptr) {}
129
130     void addBook(int id, string title, string author)
131     {
132         try
133         {
134             Book* newBook = new Book{id, title, author, false, nullptr};
135             newBook->next = head;
136             head = newBook;
137
138             ofstream outFile(bookDataFile, ios::app);
139             if (outFile.is_open())
140             {
141                 outFile << id << "," << title << "," << author << endl;
142                 outFile.close();
143             }
144             else
145             {
146                 cerr << "Unable to open file for writing." << endl;
147             }
148         }
149         catch (const bad_alloc& e)
150         {
151             cerr << "Memory allocation failed: " << e.what() << endl;
152         }
153     }
154
155     void displayBooks()
156     {
157         ifstream inFile(bookDataFile);
158         if (inFile.is_open())
159         {
160             string line;
161             while (getline(inFile, line))
162             {
163                 stringstream ss(line);
164                 string idStr, title, author;
165                 getline(ss, idStr, ',');
166                 getline(ss, title, ',');
167                 getline(ss, author, ',');
168

```

```

169         int id = stoi(idStr);
170
171         cout << "Name: " << title << " , Author: " << author << " , ID: " << id << endl;
172     }
173     inFile.close();
174 }
175 else
176 {
177     cerr << "Unable to open file for reading." << endl;
178 }
179 }
180
181 Book* searchBook(int id)
182 {
183     Book* temp = head;
184     while (temp != nullptr)
185     {
186         if (temp->id == id)
187         {
188             return temp;
189         }
190         temp = temp->next;
191     }
192     return nullptr;
193 }
194
195 void quickSort(Book* low, Book* high)
196 {
197     if (low == nullptr || high == nullptr || low == high || low->next == high) return;
198
199     Book* pivot = partition(low, high);
200     quickSort(low, pivot);
201     quickSort(pivot->next, high);
202 }
203
204 Book* partition(Book* low, Book* high)
205 {
206     string pivot = high->title;
207     Book* i = low->next;
208     Book* j = low;
209     while (i != high)
210     {
211         if (i->title < pivot)
212         {
213             j = (j == nullptr) ? low : j->next;
214             swap(j->title, i->title);
215             swap(j->id, i->id);
216             swap(j->author, i->author);
217             swap(j->isBorrowed, i->isBorrowed);
218         }
219         i = i->next;
220     }
221     j = (j == nullptr) ? low : j->next;
222     swap(j->title, high->title);
223     swap(j->id, high->id);
224     swap(j->author, high->author);
225     swap(j->isBorrowed, high->isBorrowed);
226     return j;
227 }
228
229 void sortBooks()
230 {
231     if (head == nullptr) return;
232
233     Book* tail = head;
234     while (tail->next != nullptr)
235     {
236         tail = tail->next;
237     }
238
239     quickSort(head, tail);
240 }
241
242
243 Book* getHead()
244 {
245     return head;
246 }
247
248 Book* binarySearch(string title)
249 {
250     sortBooks();
251     Book* low = head;
252     Book* high = nullptr;

```

```

253
254     while (low != high)
255     {
256         Book* mid = low;
257         int count = 0;
258         while (mid->next != high)
259         {
260             mid = mid->next;
261             ++count;
262         }
263         for (int i = 0; i < count / 2; ++i)
264         {
265             mid = mid->next;
266         }
267         if (mid->title == title)
268         {
269             return mid;
270         }
271
272         else if (mid->title < title)
273         {
274             low = mid->next;
275         }
276         else
277         {
278             high = mid;
279         }
280     }
281     return nullptr;
282 }
283 };
284
285 class MemberList
286 {
287 private:
288     Member* head;
289     const string memberDataFile = "members.txt";
290
291 public:
292     MemberList() : head(nullptr) {}
293
294     void addMember(int id, string name)
295     {
296         try
297         {
298             Member* newMember = new Member{id, name, nullptr, 0, nullptr};
299             newMember->next = head;
300             head = newMember;
301
302             ofstream outFile(memberDataFile, ios::app);
303             if (outFile.is_open())
304             {
305                 outFile << id << "," << name << endl;
306                 outFile.close();
307             }
308             else
309             {
310                 cerr << "Unable to open file for writing." << endl;
311             }
312         }
313         catch (const bad_alloc& e)
314         {
315             cerr << "Memory allocation failed: " << e.what() << endl;
316         }
317     }
318
319     void displayMembers()
320     {
321         ifstream inFile(memberDataFile);
322         if (inFile.is_open())
323         {
324             string line;
325             while (getline(inFile, line))
326             {
327                 stringstream ss(line);
328                 string idStr, name;
329                 getline(ss, idStr, ',');
330                 getline(ss, name, ',');
331
332                 int id = stoi(idStr);
333
334                 cout << "Name: " << name << " , ID: " << id << " , Borrowed books: ";
335
336                 Member* member = searchMember(id);
337                 if (member)

```

```

337         {
338             cout << member->borrowedCount;
339         }
340         else
341         {
342             cout << "0";
343         }
344         cout << endl;
345     }
346     inFile.close();
347 }
348 else
349 {
350     cerr << "Unable to open file for reading." << endl;
351 }
352 }
353
354
355 Member* searchMember(int id)
356 {
357     Member* temp = head;
358     while (temp != nullptr)
359     {
360         if (temp->id == id)
361         {
362             return temp;
363         }
364         temp = temp->next;
365     }
366     return nullptr;
367 }
368
369 void borrowBook(int memberId, Book* book)
370 {
371     Member* member = searchMember(memberId);
372     if (member && book && !book->isBorrowed)
373     {
374         book->isBorrowed = true;
375         Book* borrowedBook = new Book(book->id, book->title, book->author,
book->isBorrowed, member->borrowedBooks);
376         member->borrowedBooks = borrowedBook;
377         member->borrowedCount++;
378         cout << "Book borrowed successfully." << endl;
379     }
380     else
381     {
382         cout << "Error: Member not found or book already borrowed." << endl;
383     }
384 }
385
386 void returnBook(int memberId, int bookId)
387 {
388     Member* member = searchMember(memberId);
389     if (!member)
390     {
391         cout << "Member not found." << endl;
392         return;
393     }
394
395     Book* prev = nullptr;
396     Book* curr = member->borrowedBooks;
397     while (curr != nullptr && curr->id != bookId)
398     {
399         prev = curr;
400         curr = curr->next;
401     }
402
403     if (curr != nullptr)
404     {
405         if (prev != nullptr)
406         {
407             prev->next = curr->next;
408         }
409         else
410         {
411             member->borrowedBooks = curr->next;
412         }
413
414         delete curr;
415         member->borrowedCount--;
416         cout << "Book returned successfully." << endl;
417     }
418     else
419     {

```

```

420         cout << "Book not found in member's borrowed books." << endl;
421     }
422 }
423
424 Member* bfsSearchByID(int id)
425 {
426     if (!head)
427         return nullptr;
428
429     queue<Member*> q;
430     unordered_map<int, Member*> visited;
431
432     q.push(head);
433     visited[head->id] = head;
434
435     while (!q.empty())
436     {
437         Member* current = q.front();
438         q.pop();
439
440         if (current->id == id)
441             return current;
442
443         Member* neighbor = current->next;
444         while (neighbor)
445         {
446             if (visited.find(neighbor->id) == visited.end())
447             {
448                 q.push(neighbor);
449                 visited[neighbor->id] = neighbor;
450             }
451             neighbor = neighbor->next;
452         }
453     }
454
455     return nullptr;
456 }
457 };
458
459 class LibraryGraph
460 {
461 private:
462     unordered_map<int, unordered_map<int, int>> adjacencyList;
463
464 public:
465
466     void addEdge(int memberId1, int memberId2, int weight)
467     {
468         adjacencyList[memberId1][memberId2] = weight;
469         adjacencyList[memberId2][memberId1] = weight;
470     }
471
472     vector<int> findShortestPath(int memberId1, int memberId2)
473     {
474         unordered_map<int, int> distance;
475         unordered_map<int, int> parent;
476         priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
477
478         for (auto it = adjacencyList.begin(); it != adjacencyList.end(); ++it)
479         {
480             int member = it->first;
481             distance[member] = INT_MAX;
482             parent[member] = -1;
483         }
484
485         distance[memberId1] = 0;
486         pq.push({0, memberId1});
487
488         while (!pq.empty())
489         {
490             int u = pq.top().second;
491             pq.pop();
492
493             for (auto it = adjacencyList[u].begin(); it != adjacencyList[u].end(); ++it)
494             {
495                 int v = it->first;
496                 int weight = it->second;
497
498                 if (distance[v] > distance[u] + weight)
499                 {
500                     distance[v] = distance[u] + weight;
501                     parent[v] = u;
502                     pq.push({distance[v], v});
503                 }
504             }
505         }
506     }
507 };

```

```

504     }
505 }
506
507 vector<int> shortestPath;
508 int current = memberId2;
509 while (current != -1)
510 {
511     shortestPath.push_back(current);
512     current = parent[current];
513 }
514 reverse(shortestPath.begin(), shortestPath.end());
515
516 return shortestPath;
517 }
518 };
519
520 void printMenu()
521 {
522     cout << "\n1. Add Book" << endl;
523     cout << "2. Add Member" << endl;
524     cout << "3. Display Books" << endl;
525     cout << "4. Display Members" << endl;
526     cout << "5. Borrow Book" << endl;
527     cout << "6. Return Book" << endl;
528     cout << "7. Search Book by Title" << endl;
529     cout << "8. Search Member by ID" << endl;
530     cout << "9. Find Shortest Path Between Members" << endl;
531     cout << "10. Exit" << endl;
532     cout << "Enter your choice: ";
533 }
534
535 int main()
536 {
537     LoginSystem loginSystem;
538     loginSystem.showWelcomeScreen();
539     loginSystem.authenticateUser();
540
541     BookList books;
542     MemberList members;
543     LibraryGraph libraryGraph;
544
545     libraryGraph.addEdge(1, 2, 1);
546     libraryGraph.addEdge(1, 3, 1);
547     libraryGraph.addEdge(2, 4, 1);
548     libraryGraph.addEdge(3, 4, 1);
549     libraryGraph.addEdge(4, 5, 1);
550
551     int choice;
552     do
553     {
554         printMenu();
555         cin >> choice;
556         switch (choice)
557         {
558             case 1:
559             {
560                 int id;
561                 string title, author;
562                 cout << "\nEnter Book ID: ";
563                 cin >> id;
564                 cout << "Enter Book Title: ";
565                 cin.ignore();
566                 getline(cin, title);
567                 cout << "Enter Book Author: ";
568                 getline(cin, author);
569                 books.addBook(id, title, author);
570                 cout << "Book added successfully.\n" << endl;
571                 break;
572             }
573             case 2:
574             {
575                 int id;
576                 string name;
577                 cout << "\nEnter Member ID: ";
578                 cin >> id;
579                 cout << "Enter Member Name: ";
580                 cin.ignore();
581                 getline(cin, name);
582                 members.addMember(id, name);
583                 cout << "Member added successfully.\n" << endl;
584                 break;
585             }
586             case 3:
587             {

```

```

588         cout << "\nList of Books:\n" << endl;
589         books.displayBooks();
590         cout << "\n\n";
591         break;
592     }
593     case 4:
594     {
595         cout << "\nMembers of the library:\n" << endl;
596         members.displayMembers();
597         cout << "\n\n";
598         break;
599     }
600     case 5:
601     {
602         int memberId, bookId;
603         cout << "\nEnter Member ID: ";
604         cin >> memberId;
605         cout << "Enter Book ID: ";
606         cin >> bookId;
607         Book* book = books.searchBook(bookId);
608         if (book)
609         {
610             members.borrowBook(memberId, book);
611         }
612         else
613         {
614             cout << "Book not found.\n" << endl;
615         }
616         break;
617     }
618     case 6:
619     {
620         int memberId, bookId;
621         cout << "\nEnter Member ID: ";
622         cin >> memberId;
623         cout << "Enter Book ID: ";
624         cin >> bookId;
625         members.returnBook(memberId, bookId);
626         break;
627     }
628     case 7:
629     {
630         string title;
631         cout << "\nEnter Book Title: ";
632         cin.ignore();
633         getline(cin, title);
634         Book* book = books.binarySearch(title);
635         if (book)
636         {
637             cout << "Book found: ID=" << book->id << ", Title=" << book->title << ",
Author=" << book->author << endl;
638         }
639         else
640         {
641             cout << "Book not found.\n" << endl;
642         }
643         break;
644     }
645     case 8:
646     {
647         int id;
648         cout << "\nEnter Member ID to search: ";
649         cin >> id;
650         Member* member = members.searchMember(id);
651         if (member)
652         {
653             cout << "Member found: ID=" << member->id << ", Name=" << member->name << ",
Borrowed Books=" << member->borrowedCount << endl;
654         }
655         else
656         {
657             cout << "Member not found.\n" << endl;
658         }
659         break;
660     }
661     case 9:
662     {
663         int memberId1, memberId2;
664         cout << "\nEnter ID of first member: ";
665         cin >> memberId1;
666         cout << "Enter ID of second member: ";
667         cin >> memberId2;
668
669         vector<int> shortestPath = libraryGraph.findShortestPath(memberId1, memberId2);

```



```

670
671         if (shortestPath.empty())
672         {
673             cout << "There is no path between the given members." << endl;
674         }
675         else
676         {
677             cout << "Shortest path between members: ";
678             for (int memberId : shortestPath)
679             {
680                 cout << memberId << " ";
681             }
682             cout << endl;
683         }
684         break;
685     }
686     case 10:
687     {
688         cout << "Exiting program.\n";
689         break;
690     }
691     default:
692     {
693         cout << "Invalid choice. Please enter a number between 1 and 10.\n";
694         break;
695     }
696 }
697
698 while (choice != 10);
699
700 return 0;
701 }

```