**TRƯỜNG ĐẠI HỌC KINH TẾ - KỸ THUẬT CÔNG NGHIỆP**

**KHOA CÔNG NGHỆ THÔNG TIN**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***



**BÀI TẬP LỚN CUỐI KỲ**  
**MÔN: CẤU TRÚC DỮ LIỆU & GIẢI THUẬT**

**ĐỀ TÀI: DANH SÁCH LIÊN KẾT KÉP**

***Nhóm sinh viên thực hiện:***

Nhóm trưởng: Nguyễn Kim Cương - 20103100772

Thành viên 1: Vũ Khương Duy - 20103100764

Thành viên 2: Phan Thế Vũ - 20103100437

***Giảng viên hướng dẫn:***

*Th.S. Trần Thị Lan Anh*

**Hà Nội - 2021**

MỤC LỤC

[LỜI CẢM ƠN 4](#_Toc91427041)

[ĐẶT VẤN ĐỀ 5](#_Toc91427042)

[PHẦN I: GIỚI THIỆU VỀ DANH SÁCH LIÊN KẾT KÉP. 6](#_Toc91427043)

[1. Khái niệm cơ bản: 6](#_Toc91427044)

[1.1. Danh sách liên kết kép (Doubly Linked List) 6](#_Toc91427045)

[1.2. Tổ chức dữ liệu 7](#_Toc91427046)

[2. Các thao tác trên danh sách liên kết kép: 7](#_Toc91427047)

[2.1. Các thao tác khởi tạo 7](#_Toc91427048)

[2.1.1. Init - Khởi tạo danh sách 7](#_Toc91427049)

[2.1.2. CreateNode - Khởi tạo node 7](#_Toc91427050)

[2.2. Các thao tác duyệt 8](#_Toc91427051)

[2.2.1. IsEmpty - Kiểm tra nếu danh sách rỗng 8](#_Toc91427052)

[2.2.2. ShowNode - Duyệt node 8](#_Toc91427053)

[2.2.3. ShowList - Duyệt từ đầu đến cuối danh sách 8](#_Toc91427054)

[2.2.4. ShowReverse - Duyệt từ cuối về đầu danh sách 8](#_Toc91427055)

[2.3. Các thao tác thêm bớt 9](#_Toc91427056)

[2.3.1. InsertFirst - Bổ sung node vào đầu danh sách. 9](#_Toc91427057)

[2.3.2. InsertLast - Bổ sung node vào cuối danh sách. 9](#_Toc91427058)

[2.3.3. InsertBefore - Bổ sung node vào trước node có con trỏ p. 9](#_Toc91427059)

[2.3.4. InsertAfter - Bổ sung node vào sau node có con trỏ p. 10](#_Toc91427060)

[2.3.5. DeleteFirst - Loại bỏ node ở đầu danh sách. 10](#_Toc91427061)

[2.3.6. DeleteLast - Loại bỏ node ở cuối danh sách. 10](#_Toc91427062)

[2.3.7. DeleteNode - Loại bỏ node có con trỏ p. 11](#_Toc91427063)

[2.4. Các thao tác khác 11](#_Toc91427064)

[2.4.1. Search - Tìm kiếm dựa trên giá trị node, trả về con trỏ node đó. 11](#_Toc91427065)

[2.4.2. ClearList - Xóa toàn bộ danh sách 12](#_Toc91427066)

[PHẦN II. XỬ LÝ BÀI TOÁN VỚI LIÊN KẾT KÉP. 12](#_Toc91427067)

[1. Đề bài 12](#_Toc91427068)

[2. Các hàm cần xây dựng 14](#_Toc91427069)

[2.1. Các hàm khởi tạo 14](#_Toc91427070)

[2.1.1. Khai báo struct SinhVien 14](#_Toc91427071)

[2.1.2. Các hàm nhập xuất cho struct SinhVien 14](#_Toc91427072)

[2.1.3. Khai báo và khởi tạo DSLK kép 15](#_Toc91427073)

[2.1.4. Init 16](#_Toc91427074)

[2.1.5. CreateNode 16](#_Toc91427075)

[2.2. Các hàm duyệt 16](#_Toc91427076)

[2.2.1. IsEmpty 16](#_Toc91427077)

[2.2.2. ShowNode 16](#_Toc91427078)

[2.2.3. ShowList 16](#_Toc91427079)

[2.3. Các hàm thêm bớt 17](#_Toc91427080)

[2.3.1. InsertLast 17](#_Toc91427081)

[2.3.2. DeleteFirst 17](#_Toc91427082)

[2.3.3. DeleteLast 17](#_Toc91427083)

[2.3.4. DeleteNode 18](#_Toc91427084)

[2.4. Các hàm khác 18](#_Toc91427085)

[2.4.1. Search - Tìm kiếm node dựa vào mã sinh viên 18](#_Toc91427086)

[2.4.2. SearchByClass - Tìm kiếm node dựa vào tên lớp 19](#_Toc91427087)

[2.4.3. SearchByMathScoreMoreThan - Tìm kiếm node dựa vào điểm Toán 19](#_Toc91427088)

[2.4.4. SearchByName - Tìm kiếm node dựa vào tênã sinh viên 19](#_Toc91427089)

[2.4.5. CompareNodes - So sánh node dựa vào biến tham số tiêu chí so sánh 19](#_Toc91427090)

[2.4.6. SortedInsert - Thêm node vào vị trí được sắp xếp trong danh sách 20](#_Toc91427091)

[2.4.7. InsertionSort 21](#_Toc91427092)

[2.4.8. BubbleSort 21](#_Toc91427093)

[2.4.9. BubbleSort2 - chỉ so sánh khi điểm Toán bằng nhau 22](#_Toc91427094)

[2.4.10. BubbleSort3 - chỉ so sánh khi điểm Toán và Lý bằng nhau 22](#_Toc91427095)

[2.5. Các hàm theo yêu cầu đề bài 23](#_Toc91427096)

[2.5.1. Ham1 - Hiển thị toàn bộ danh sách 23](#_Toc91427097)

[2.5.2. Ham2 - Tìm một sinh viên theo mã sinh viên (nhập vào) 23](#_Toc91427098)

[2.5.3. Ham3 - Thêm một sinh viên vào cuối danh sách. Đảm bảo không có 2 sinh viên nào trùng mã. 24](#_Toc91427099)

[2.5.4. Ham4 - Xóa một sinh viên khỏi danh sách từ mã sinh viên (nhập vào) 24](#_Toc91427100)

[2.5.5. Ham5 - Nhập một lớp. Hiển thị danh sách sinh viên thuộc về lớp đó 24](#_Toc91427101)

[2.5.6. Ham6 - Tính tổng số sinh viên có điểm toán >=5 25](#_Toc91427102)

[2.5.7. Ham7 - Hiển thị toàn bộ danh sách sinh viên chứa tên nhập vào. (Ví dụ: nhập tên cần tìm là Khuong, hiển thị toàn bộ sinh viên chứa tên Khuong) 25](#_Toc91427103)

[2.5.8. Ham8 - Sắp xếp danh sách tăng dần theo mã sinh viên 26](#_Toc91427104)

[2.5.9. Ham9 - Sắp xếp danh sách tăng dần theo điểm toán, rồi đến điểm lý, rồi đến điểm hóa. 26](#_Toc91427105)

[2.5.10. Ham10 - Nhập một lớp. Hủy toàn bộ các sinh viên thuộc về lớp đó. 26](#_Toc91427106)

[2.5.11. pause - Tạm dừng để hiển thị kết quả 27](#_Toc91427107)

[2.5.12. input - Bước nhập xuất ở đầu chương trình 27](#_Toc91427108)

[2.5.13. menu - Hiển thị menu và nhập lựa chọn 28](#_Toc91427109)

[3. Full Code 29](#_Toc91427110)

[a. Code 29](#_Toc91427111)

[b. Các kết quả sau khi chạy chương trình 45](#_Toc91427112)

[Phần III: Tổng Kết 52](#_Toc91427113)

# LỜI CẢM ƠN

Lời đầu tiên, chúng em xin chân thành cảm ơn cô giáo Trần Thị Lan Anh đã trang bị cho chúng em những kiến thức cần thiết để chúng em có thể thực hiện tốt bài tập. Cám ơn thầy cô luôn tận tình giúp đỡ, hướng dẫn và chỉ bảo chúng em trong suốt quá trình học tập, nghiên cứu.

Trong thời gian làm bài tập, chúng em cũng đã nhận được rất nhiều sự giúp đỡ, đóng góp ý kiến và chỉ bảo nhiệt tình của thầy cô, gia đình và bạn bè. Tuy nhiên, do còn hạn chế về mặt kiến thức, cũng như còn thiếu kinh nghiệm về nghiên cứu, bài báo cáo của chúng em không thể tránh khỏi những sai sót. Chính vì vậy, chúng em rất mong nhận được sự góp ý từ các thầy cô giáo và bạn bè.

Chúng em xin chân thành cám ơn quý thầy cô giáo!

Hà Nội, ngày 23 tháng 12 năm 2021

Sinh viên thực hiện

*Nguyễn Kim Cương*

*Vũ Khương Duy*

*Phan Thế Vũ*

# ĐẶT VẤN ĐỀ

Mục đích bài báo cáo là nhằm giới thiệu các khái niệm cơ bản, về danh sách liên kết kép và cách xử lí bài toán với danh sách liên kết kép.

Bài báo cáo gồm 2 nội dung chính:

Phần I: Giới thiệu danh sách liên kết kép.

Phần II: Xử lý bài toán với liên kết kép.

# PHẦN I: GIỚI THIỆU VỀ DANH SÁCH LIÊN KẾT KÉP.

## 1. Khái niệm cơ bản:

### *1.1. Danh sách liên kết kép (Doubly Linked List)*

Danh sách liên kết kép (Doubly Linked List) là một biến thể của Danh sách liên kết (Linked List), trong đó hoạt động duyệt qua các node có thể được thực hiện theo hai chiều: về trước và về sau một cách dễ dàng so với Danh sách liên kết đơn. Dưới đây là một số khái niệm quan trọng cần ghi nhớ về Danh sách liên kết kép.

Trong node có:

**Info**: là giá trị của node.

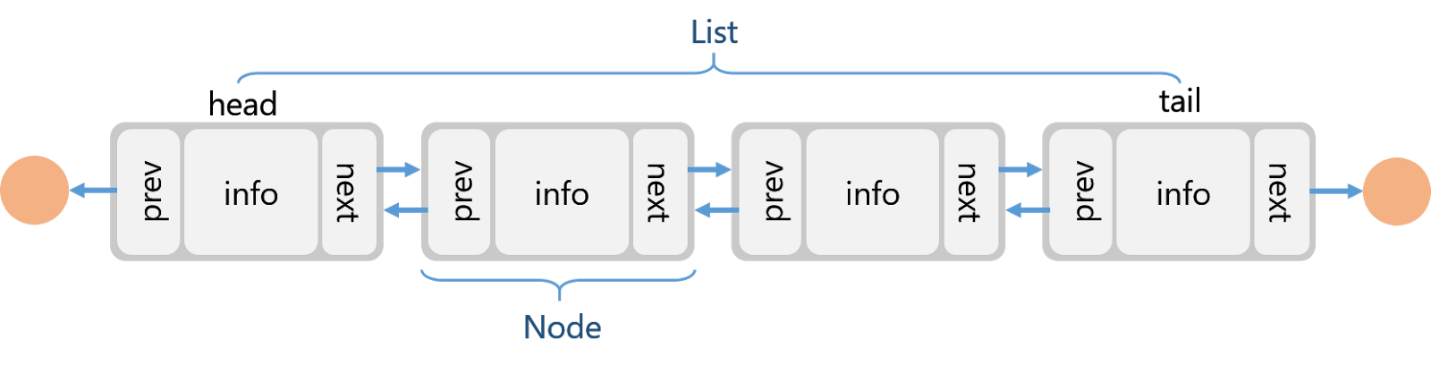
**Prev**: là con trỏ, trỏ tới phần tử liền trước.

**Next**: là con trỏ, trỏ tới phần tử liền sau.

**pHead**: là node đầu tiên trong DSLK kép, nó luôn quản lý node đầu.

**pTail**: là node cuối cùng trong DSLK kép, nó luôn quản lý node cuối.

Có thể minh họa danh sách liên kết kép như sau:



Với DSLK kép ta cần biết một vài thao tác cơ bản đủ dùng và các thao tác này sẽ được minh họa bằng C++.

### *1.2. Tổ chức dữ liệu*

Tổ chức biểu diễn một node của danh sách liên kết kép (giả sử kiểu dữ liệu cho giá trị là int):

|  |  |
| --- | --- |
| 1  2  3  4  5 | struct Node {      int info;      struct Node\* prev;      struct Node\* next;  }; |

Tạo bí danh NodePtr cho con trỏ node:

|  |  |
| --- | --- |
| 1 | typedef Node\* NodePtr; |

Khai báo con trỏ đầu và cuối danh sách:

|  |  |
| --- | --- |
| 1  2  3  4 | struct List {      NodePtr pHead;  NodePtr pTail;  }; |

## 2. Các thao tác trên danh sách liên kết kép:

### *2.1. Các thao tác khởi tạo*

#### 2.1.1. Init - Khởi tạo danh sách

|  |  |
| --- | --- |
| 1  2  3  4 | void Init(List &l) {      l.pHead = NULL;      l.pTail = NULL;  } |

#### 2.1.2. CreateNode - Khởi tạo node

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | NodePtr CreateNode(int x) {      NodePtr node = new Node;      node->info = x;      node->next = NULL;      node->prev = NULL;      return node;  } |

### *2.2. Các thao tác duyệt*

#### 2.2.1. IsEmpty - Kiểm tra nếu danh sách rỗng

|  |  |
| --- | --- |
| 1  2  3 | int IsEmpty(List l) {      return (l.pHead == NULL && l.pTail == NULL);  } |

#### 2.2.2. ShowNode - Duyệt node

|  |  |
| --- | --- |
| 1  2  3 | void ShowNode(NodePtr q) {      cout<<q->info;  } |

#### 2.2.3. ShowList - Duyệt từ đầu đến cuối danh sách

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | void ShowList(List l) {      if (IsEmpty(l)) return;      NodePtr p = l.pHead;      while (p != NULL) {          ShowNode(p);          p = p->next;      }  } |

#### 2.2.4. ShowReverse - Duyệt từ cuối về đầu danh sách

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | void ShowReverse(List l) {      if (IsEmpty(l)) return;      NodePtr p = l.pTail;      while (p != NULL) {          ShowNode(p);          p = p->prev;      }  } |

### *2.3. Các thao tác thêm bớt*

#### 2.3.1. InsertFirst - Bổ sung node vào đầu danh sách.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | void InsertFirst(List &l, int x) {      NodePtr node = CreateNode(x);      if (IsEmpty(l)) {          l.pHead = l.pTail = node;      } else {          node->next = l.pHead;          l.pHead->prev = node;          l.pHead = node;      }  } |

#### 2.3.2. InsertLast - Bổ sung node vào cuối danh sách.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | void InsertLast(List &l, int x) {      NodePtr node = CreateNode(x);      if (IsEmpty(l)) {          l.pHead = l.pTail = node;      } else {          l.pTail->next = node;          node->prev = l.pTail;          l.pTail = node;      }  } |

#### 2.3.3. InsertBefore - Bổ sung node vào trước node có con trỏ p.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | void InsertBefore(List &l, NodePtr &p, int x) {      if (p == NULL) return;      if (p == l.pHead) InsertFirst(l, x);      else {          NodePtr node = CreateNode(x);          NodePtr before = p->prev;          before->next = node;          node->prev = before;          node->next = p;          p->prev = node;      }  } |

#### 2.3.4. InsertAfter - Bổ sung node vào sau node có con trỏ p.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | void InsertAfter(List &l, NodePtr &p, int x) {      if (p == NULL) return;      if (p == l.pHead) InsertLast(l, x);      else {          NodePtr node = CreateNode(x);          NodePtr after = p->next;          after->prev = node;          node->next = after;          node->prev = p;          p->next = node;      }  } |

#### 2.3.5. DeleteFirst - Loại bỏ node ở đầu danh sách.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | void DeleteFirst(List &l) {  if (IsEmpty(l)) return;  else if (l.pHead == l.pTail) {  delete l.pHead;  l.pHead = NULL;  l.pTail = NULL;  } else {  NodePtr p = l.pHead;  l.pHead = l.pHead->next;  l.pHead->prev = NULL;  delete p;  }  } |

#### 2.3.6. DeleteLast - Loại bỏ node ở cuối danh sách.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | void DeleteLast(List &l) {  if (IsEmpty(l)) return;  else if (l.pHead == l.pTail) {  delete l.pHead;  l.pHead = NULL;  l.pTail = NULL;  } else {  NodePtr p = l.pTail;  l.pTail = l.pTail->prev;  l.pTail->next = NULL;  delete p;  }  } |

#### 2.3.7. DeleteNode - Loại bỏ node có con trỏ p.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | void DeleteNode(List &l, NodePtr &p) {  if (p == NULL) return;  if (p == l.pHead) {  DeleteFirst(l);  return;  }  if (p == l.pTail) {  DeleteLast(l);  return;  }  else {  NodePtr left = p->prev;  NodePtr right = p->next;  left->next = right;  if (right != NULL) right->prev = left;  delete p;  }  } |

### *2.4. Các thao tác khác*

#### 2.4.1. Search - Tìm kiếm dựa trên giá trị node, trả về con trỏ node đó.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | NodePtr Search(List l, int x) {      if (IsEmpty(l)) return NULL;      NodePtr p = l.pHead;      while (p->info != x && p->next != NULL) p = p->next;      if (p->info == x) return p;      else return NULL;  } |

#### 2.4.2. ClearList - Xóa toàn bộ danh sách

|  |  |
| --- | --- |
| 1  2  3  4  5 | void ClearList(List &l) {      while (l.pHead != NULL) {          DeleteFirst(l);      }  } |

# PHẦN II. XỬ LÝ BÀI TOÁN VỚI LIÊN KẾT KÉP.

## 1. Đề bài

Viết chương trình xây dựng và quản lý danh sách liên kết kép. Thành phần quản lý gồm con trỏ pHead, pTail. Thành phần dữ liệu trong mỗi node là thông tin một sinh viên, bao gồm các trường:

* Mã sinh viên (int)
* Họ tên sinh viên (string)
* Lớp (string)
* Điểm toán (float)
* Điểm lý (float)
* Điểm hóa (float)

Hiển thị menu thực hiện các chức năng sau (mỗi chức năng thực hiện bằng hàm).

1. Hiển thị toàn bộ danh sách
2. Tìm một sinh viên theo mã sinh viên (nhập vào)
3. Thêm một sinh viên vào cuối danh sách. Đảm bảo không có 2 sinh viên nào trùng mã.
4. Xóa một sinh viên khỏi danh sách từ mã sinh viên (nhập vào)
5. Nhập một lớp. Hiển thị danh sách sinh viên thuộc về lớp đó
6. Tính tổng số sinh viên có điểm toán >=5
7. Hiển thị toàn bộ danh sách sinh viên chứa tên nhập vào. (Ví dụ: nhập tên cần tìm là Khuong, hiển thị toàn bộ sinh viên chứa tên Khuong)
8. Sắp xếp danh sách tăng dần theo mã sinh viên
9. Sắp xếp danh sách tăng dần theo điểm toán, rồi đến điểm lý, rồi đến điểm hóa.
10. Nhập một lớp. Hủy toàn bộ các sinh viên thuộc về lớp đó.

## 2. Các hàm cần xây dựng

### 2.1. Các hàm khởi tạo

#### 2.1.1. Khai báo struct SinhVien

|  |  |
| --- | --- |
|  | struct SinhVien {      int ma;      string ten;      string lop;      float toan, ly, hoa;  }; |

#### 2.1.2. Các hàm nhập xuất cho struct SinhVien

|  |  |
| --- | --- |
|  | void show(SinhVien sv) {      cout<<left;      cout<<setw(MAL)<<sv.ma;      cout<<setw(TENL)<<sv.ten;      cout<<setw(LOPL)<<sv.lop;      cout<<right;      cout<<setw(DIEML)<<sv.toan;      cout<<setw(DIEML)<<sv.ly;      cout<<setw(DIEML)<<sv.hoa;      cout<<endl;  }  void header() {      cout<<left;      cout<<setw(MAL)<<"Ma";      cout<<setw(TENL)<<"Ho ten";      cout<<setw(LOPL)<<"Lop";      cout<<right;      cout<<setw(DIEML)<<"Toan";      cout<<setw(DIEML)<<"Ly";      cout<<setw(DIEML)<<"Hoa";      cout<<endl;  }  void catchInvalid(string field, float &score) {      while (score < 0 || score > 10) {          cout<<"Khong hop le. Nhap lai."<<endl;          cout<<"\tNhap diem "<<field<<": ";          cin>>score;      }  }  SinhVien insert(int ma) {      SinhVien sv;      sv.ma = ma;      cout<<"\tNhap ho ten: ";      cin.ignore();      fflush(stdin);      getline(cin, sv.ten);      cout<<"\tNhap ten lop: ";      fflush(stdin);      getline(cin, sv.lop);      cout<<"\tNhap diem Toan: ";      cin>>sv.toan;      catchInvalid("Toan", sv.toan);      cout<<"\tNhap diem Ly: ";      cin>>sv.ly;      catchInvalid("Ly", sv.ly);      cout<<"\tNhap diem Hoa: ";      cin>>sv.hoa;      catchInvalid("Hoa", sv.hoa);      return sv;  } |

#### 2.1.3. Khai báo và khởi tạo DSLK kép

|  |  |
| --- | --- |
|  | struct Node {      SinhVien info;      struct Node\* prev;      struct Node\* next;  };  typedef Node\* NodePtr;  struct List {      NodePtr pHead;      NodePtr pTail;  }; |

#### 2.1.4. Init

|  |  |
| --- | --- |
|  | void Init(List &l) {      l.pHead = NULL;      l.pTail = NULL;  } |

#### 2.1.5. CreateNode

|  |  |
| --- | --- |
|  | NodePtr CreateNode(SinhVien sv) {      NodePtr node = new Node;      node->info = sv;      node->next = NULL;      node->prev = NULL;      return node;  } |

### 2.2. Các hàm duyệt

#### 2.2.1. IsEmpty

|  |  |
| --- | --- |
|  | int IsEmpty(List l) {      return (l.pHead == NULL && l.pTail == NULL);  } |

#### 2.2.2. ShowNode

|  |  |
| --- | --- |
|  | void ShowNode(NodePtr q) {      show(q->info);  } |

#### 2.2.3. ShowList

|  |  |
| --- | --- |
|  | void ShowList(List l) {      if (IsEmpty(l)) return;      NodePtr p = l.pHead;      while (p != NULL) {          ShowNode(p);          p = p->next;      }  } |

### 2.3. Các hàm thêm bớt

#### 2.3.1. InsertLast

|  |  |
| --- | --- |
|  | void InsertLast(List &l, SinhVien sv) {      NodePtr node = CreateNode(sv);      if (IsEmpty(l)) {          l.pHead = l.pTail = node;      } else {          l.pTail->next = node;          node->prev = l.pTail;          l.pTail = node;      }  } |

#### 2.3.2. DeleteFirst

|  |  |
| --- | --- |
|  | void DeleteFirst(List &l) {  if (IsEmpty(l)) return;  else if (l.pHead == l.pTail) {  delete l.pHead;  l.pHead = NULL;  l.pTail = NULL;  } else {  NodePtr p = l.pHead;  l.pHead = l.pHead->next;  l.pHead->prev = NULL;  delete p;  }  } |

#### 2.3.3. DeleteLast

|  |  |
| --- | --- |
|  | void DeleteLast(List &l) {  if (IsEmpty(l)) return;  else if (l.pHead == l.pTail) {  delete l.pHead;  l.pHead = NULL;  l.pTail = NULL;  } else {  NodePtr p = l.pTail;  l.pTail = l.pTail->prev;  l.pTail->next = NULL;  delete p;  }  } |

#### 2.3.4. DeleteNode

|  |  |
| --- | --- |
|  | void DeleteNode(List &l, NodePtr &p) {  if (p == NULL) return;  if (p == l.pHead) {  DeleteFirst(l);  return;  }  if (p == l.pTail) {  DeleteLast(l);  return;  }  else {  NodePtr left = p->prev;  NodePtr right = p->next;  left->next = right;  if (right != NULL) right->prev = left;  delete p;  }  } |

### 2.4. Các hàm khác

#### 2.4.1. Search - Tìm kiếm node dựa vào mã sinh viên

|  |  |
| --- | --- |
|  | NodePtr Search(List l, int ma) {      if (IsEmpty(l)) return NULL;      NodePtr p = l.pHead;      while (p->info.ma != ma && p->next != NULL) p = p->next;      if (p->info.ma == ma) return p;      else return NULL;  } |

#### 2.4.2. SearchByClass - Tìm kiếm node dựa vào tên lớp

|  |  |
| --- | --- |
|  | NodePtr SearchByClass(NodePtr pHead, string lop) {      if (pHead == NULL) return NULL;      NodePtr p = pHead;      while (p->info.lop.compare(lop) != 0 && p->next != NULL)  p = p->next;      if (p->info.lop.compare(lop) == 0) return p;      else return NULL;  } |

#### 2.4.3. SearchByMathScoreMoreThan - Tìm kiếm node dựa vào điểm Toán

|  |  |
| --- | --- |
|  | NodePtr SearchByMathScoreMoreThan(NodePtr pHead, int toan) {      if (pHead == NULL) return NULL;      NodePtr p = pHead;      while (p->info.toan != toan && p->next != NULL)  p = p->next;      if (p->info.toan >= toan) return p;      else return NULL;  } |

#### 2.4.4. SearchByName - Tìm kiếm node dựa vào tênã sinh viên

|  |  |
| --- | --- |
|  | NodePtr SearchByName(NodePtr pHead, string ten) {      if (pHead == NULL) return NULL;      NodePtr p = pHead;      while (p->info.ten.find(ten) == string::npos && p->next != NULL) p = p->next;      if (p->info.ten.find(ten) != string::npos) return p;      else return NULL;  } |

#### 2.4.5. CompareNodes - So sánh node dựa vào biến tham số tiêu chí so sánh

|  |  |
| --- | --- |
|  | int CompareNodes(NodePtr q, NodePtr p, string s, string exp) {      if (s.compare("ma") == 0) {          if (exp.compare(">=") == 0) {              if (q->info.ma >= p->info.ma) return 1;          } else if (exp.compare("<") == 0) {              if (q->info.ma < p->info.ma) return 1;          }      } else if (s.compare("toan") == 0) {          if (exp.compare(">=") == 0) {              if (q->info.toan >= p->info.toan) return 1;          } else if (exp.compare("==") == 0) {              if (q->info.toan == p->info.toan) return 1;          } else if (exp.compare(">") == 0) {              if (q->info.toan > p->info.toan) return 1;          }      } else if (s.compare("ly") == 0) {          if (exp.compare("==") == 0) {              if (q->info.ly == p->info.ly) return 1;          } else if (exp.compare(">") == 0) {              if (q->info.ly > p->info.ly) return 1;          }      } else if (s.compare("hoa") == 0) {          if (exp.compare(">") == 0) {              if (q->info.hoa > p->info.hoa) return 1;          }      }      return 0;  } |

#### 2.4.6. SortedInsert - Thêm node vào vị trí được sắp xếp trong danh sách

|  |  |
| --- | --- |
|  | void SortedInsert(List &l, NodePtr &p, string s) {      if (IsEmpty(l)) l.pHead = p;      else if (CompareNodes(l.pHead, p, s, ">=")) {          p->next = l.pHead;          p->next->prev = p;          l.pHead = p;      } else {          NodePtr q = l.pHead;          while (q->next != NULL && CompareNodes(l.pHead, p, s, "<")) q = q->next;          p->next = q->next;          if (q->next != NULL) p->next->prev = p;          q->next = p;          p->prev = q;      }  } |

#### 2.4.7. InsertionSort

|  |  |
| --- | --- |
|  | void InsertionSort(List &l, string s) {      if (IsEmpty(l)) return;      List res;      Init(res);      NodePtr p = l.pHead;      while (p != NULL) {          NodePtr next = p->next;          p->prev = p->next = NULL;          SortedInsert(res, p, s);          p = next;      }      l = res;  } |

#### 2.4.8. BubbleSort

|  |  |
| --- | --- |
|  | void BubbleSort(List &l, string s) {      if (IsEmpty(l)) return;      bool swapped;      NodePtr ptr1;      NodePtr lptr = NULL;      do {          swapped = false;          ptr1 = l.pHead;          while (ptr1->next != lptr) {              if (CompareNodes(ptr1, ptr1->next, s, ">")) {                  swap(ptr1->info, ptr1->next->info);                  swapped = true;              }              ptr1 = ptr1->next;          }          lptr = ptr1;      }      while (swapped);  } |

#### 2.4.9. BubbleSort2 - chỉ so sánh khi điểm Toán bằng nhau

|  |  |
| --- | --- |
|  | void BubbleSort2(List &l, string s1, string s) {      if (IsEmpty(l)) return;      bool swapped;      NodePtr ptr1;      NodePtr lptr = NULL;      do {          swapped = false;          ptr1 = l.pHead;          while (ptr1->next != lptr) {              if (CompareNodes(ptr1, ptr1->next, s1, "==") && CompareNodes(ptr1, ptr1->next, s, ">")) {                  swap(ptr1->info, ptr1->next->info);                  swapped = true;              }              ptr1 = ptr1->next;          }          lptr = ptr1;      }      while (swapped);  } |

#### 2.4.10. BubbleSort3 - chỉ so sánh khi điểm Toán và Lý bằng nhau

|  |  |
| --- | --- |
|  | void BubbleSort3(List &l, string s1, string s2, string s) {      if (IsEmpty(l)) return;      bool swapped;      NodePtr ptr1;      NodePtr lptr = NULL;      do {          swapped = false;          ptr1 = l.pHead;          while (ptr1->next != lptr) {              if (CompareNodes(ptr1, ptr1->next, s1, "==") && CompareNodes(ptr1, ptr1->next, s2, "==") && CompareNodes(ptr1, ptr1->next, s, ">")) {                  swap(ptr1->info, ptr1->next->info);                  swapped = true;              }              ptr1 = ptr1->next;          }          lptr = ptr1;      }      while (swapped);  } |

### 2.5. Các hàm theo yêu cầu đề bài

#### 2.5.1. Ham1 - Hiển thị toàn bộ danh sách

|  |  |
| --- | --- |
|  | void Ham1(List l) {      if (!IsEmpty(l)) {          cout<<"Danh sach sinh vien:"<<endl;          header();          ShowList(l);      } else {          cout<<"Khong co du lieu hien thi."<<endl;      }  } |

#### 2.5.2. Ham2 - Tìm một sinh viên theo mã sinh viên (nhập vào)

|  |  |
| --- | --- |
|  | void Ham2(List l) {  int ma;  cout<<"Nhap ma sinh vien ban can tim: ";  cin>>ma;  NodePtr p = Search(l, ma);  if (p != NULL) {  cout<<"Thong tin sinh vien voi ma "<<ma<<":"<<endl;  header();  ShowNode(p);  } else {  cout<<"Khong co sinh vien nao voi ma "<<ma<<"."<<endl;  }  } |

#### 2.5.3. Ham3 - Thêm một sinh viên vào cuối danh sách. Đảm bảo không có 2 sinh viên nào trùng mã.

|  |  |
| --- | --- |
|  | void Ham3(List &l) {  bool check;  int ma;  cout<<"Nhap thong tin sinh vien moi:"<<endl;  do {  check = false;  cout<<"\tNhap ma: ";  cin>>ma;  NodePtr p = Search(l, ma);  if (p != NULL) {  cout<<"Ma "<<ma<<" da ton tai!"<<endl;  check = true;  }  } while (check);  SinhVien sv = insert(ma);  InsertLast(l, sv);  cout<<"Them sinh vien thanh cong."<<endl;  } |

#### 2.5.4. Ham4 - Xóa một sinh viên khỏi danh sách từ mã sinh viên (nhập vào)

|  |  |
| --- | --- |
|  | void Ham4(List &l) {  int ma;  cout<<"Nhap ma sinh vien can xoa: ";  cin>>ma;  NodePtr p = Search(l, ma);  if (p != NULL) {  DeleteNode(l, p);  cout<<"Xoa sinh vien thanh cong."<<endl;  } else {  cout<<"Khong co sinh vien nao voi ma "<<ma<<"."<<endl;  }  } |

#### 2.5.5. Ham5 - Nhập một lớp. Hiển thị danh sách sinh viên thuộc về lớp đó

|  |  |
| --- | --- |
|  | void Ham5(List l) {  string lop;  cout<<"Nhap ten lop can tim: ";  cin.ignore();  fflush(stdin);  getline(cin, lop);  NodePtr p = SearchByClass(l.pHead, lop);  if (p != NULL) {  cout<<"Danh sach sinh vien lop "<<lop<<":"<<endl;  header();  while (p != NULL) {  ShowNode(p);  p = SearchByClass(p->next, lop);  }  } else {  cout<<"Khong co sinh vien nao thuoc lop "<<lop<<"."<<endl;  }  } |

#### 2.5.6. Ham6 - Tính tổng số sinh viên có điểm toán >=5

|  |  |
| --- | --- |
|  | void Ham6(List l) {  int toan = 5, count = 0;  cout<<"Tong so sinh vien co diem Toan >= "<<toan<<": ";  NodePtr p = SearchByMathScoreMoreThan(l.pHead, toan);  while (p != NULL) {  count++;  p = SearchByMathScoreMoreThan(p->next, toan);  }  cout<<count<<endl;  } |

#### 2.5.7. Ham7 - Hiển thị toàn bộ danh sách sinh viên chứa tên nhập vào. (Ví dụ: nhập tên cần tìm là Khuong, hiển thị toàn bộ sinh viên chứa tên Khuong)

|  |  |
| --- | --- |
|  | void Ham7(List l) {  string ten;  cout<<"Nhap ten sinh vien can tim: ";  cin.ignore();  fflush(stdin);  getline(cin, ten);  NodePtr p = SearchByName(l.pHead, ten);  if (p != NULL) {  cout<<"Danh sach sinh vien co ten "<<ten<<":"<<endl;  header();  while (p != NULL) {  ShowNode(p);  p = SearchByName(p->next, ten);  }  } else {  cout<<"Khong co sinh vien nao co ten "<<ten<<"."<<endl;  }  } |

#### 2.5.8. Ham8 - Sắp xếp danh sách tăng dần theo mã sinh viên

|  |  |
| --- | --- |
|  | void Ham8(List &l) {  if (IsEmpty(l)) {  cout<<"Khong co sinh vien nao de sap xep."<<endl;  } else {  InsertionSort(l, "ma");  cout<<"Sap xep sinh vien thanh cong."<<endl;  }  } |

#### 2.5.9. Ham9 - Sắp xếp danh sách tăng dần theo điểm toán, rồi đến điểm lý, rồi đến điểm hóa.

|  |  |
| --- | --- |
|  | void Ham9(List &l) {  if (IsEmpty(l)) {  cout<<"Khong co sinh vien nao de sap xep."<<endl;  } else {  BubbleSort(l, "toan");  BubbleSort2(l, "toan", "ly");  BubbleSort3(l, "toan", "ly", "hoa");  cout<<"Sap xep sinh vien thanh cong."<<endl;  }  } |

#### 2.5.10. Ham10 - Nhập một lớp. Hủy toàn bộ các sinh viên thuộc về lớp đó.

|  |  |
| --- | --- |
|  | void Ham10(List &l) {  string lop;  cout<<"Nhap ten lop can huy sinh vien: ";  cin.ignore();  fflush(stdin);  getline(cin, lop);  NodePtr p = SearchByClass(l.pHead, lop);  if (p != NULL) {  while (p != NULL) {  NodePtr next = p->next;  DeleteNode(l, p);  p = SearchByClass(next, lop);  }  cout<<"Xoa sinh vien thanh cong."<<endl;  } else {  cout<<"Khong co sinh vien nao thuoc lop "<<lop<<"."<<endl;  }  } |

#### 2.5.11. pause - Tạm dừng để hiển thị kết quả

|  |  |
| --- | --- |
|  | void pause() {  string s;  cout<<"Nhan Enter de tiep tuc."<<endl;  cin.ignore();  fflush(stdin);  getline(cin, s);  } |

#### 2.5.12. input - Bước nhập xuất ở đầu chương trình

|  |  |
| --- | --- |
|  | void input(List &l) {  int n;  do {  cout<<"Nhap so sinh vien: ";  cin>>n;  if (n < 0) cout<<"Khong hop le. Nhap lai."<<endl;  } while (n < 0);  for (int i = 0; i < n; i++) {  Ham3(l);  }  } |

#### 2.5.13. menu - Hiển thị menu và nhập lựa chọn

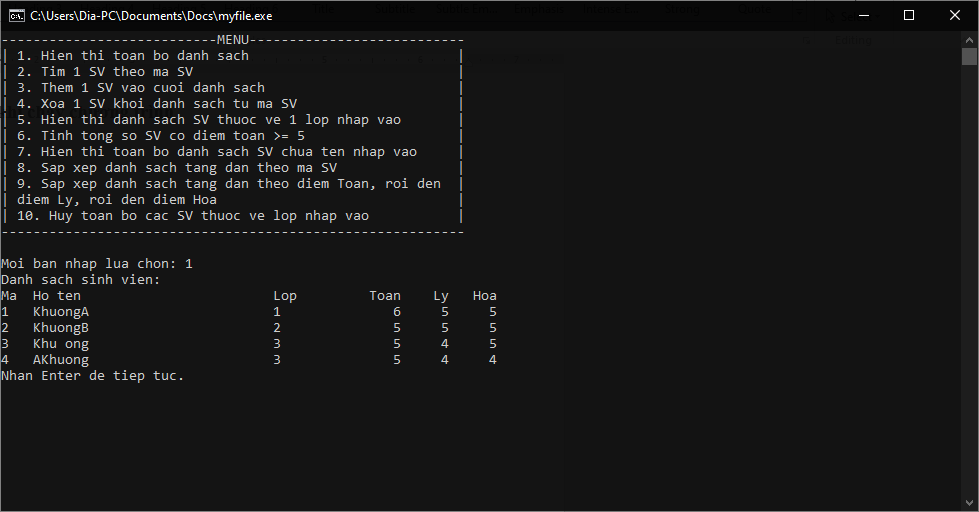
|  |  |
| --- | --- |
|  | void menu(List l) {  bool check = true;  do {  if (system("CLS")) system("clear");  cout<<"---------------------------MENU---------------------------"<<endl;  cout<<"| 1. Hien thi toan bo danh sach |"<<endl;  cout<<"| 2. Tim 1 SV theo ma SV |"<<endl;  cout<<"| 3. Them 1 SV vao cuoi danh sach |"<<endl;  cout<<"| 4. Xoa 1 SV khoi danh sach tu ma SV |"<<endl;  cout<<"| 5. Hien thi danh sach SV thuoc ve 1 lop nhap vao |"<<endl;  cout<<"| 6. Tinh tong so SV co diem toan >= 5 |"<<endl;  cout<<"| 7. Hien thi toan bo danh sach SV chua ten nhap vao |"<<endl;  cout<<"| 8. Sap xep danh sach tang dan theo ma SV |"<<endl;  cout<<"| 9. Sap xep danh sach tang dan theo diem Toan, roi den |"<<endl;  cout<<"| diem Ly, roi den diem Hoa |"<<endl;  cout<<"| 10. Huy toan bo cac SV thuoc ve lop nhap vao |"<<endl;  cout<<"----------------------------------------------------------"<<endl;  cout<<endl;  int choice;  cout<<"Moi ban nhap lua chon: ";  cin>>choice;  switch(choice) {  case 1: Ham1(l); break;  case 2: Ham2(l); break;  case 3: Ham3(l); break;  case 4: Ham4(l); break;  case 5: Ham5(l); break;  case 6: Ham6(l); break;  case 7: Ham7(l); break;  case 8: Ham8(l); break;  case 9: Ham9(l); break;  case 10: Ham10(l); break;  default: check = false; break;  }  if (check) pause();  } while (check);  } |

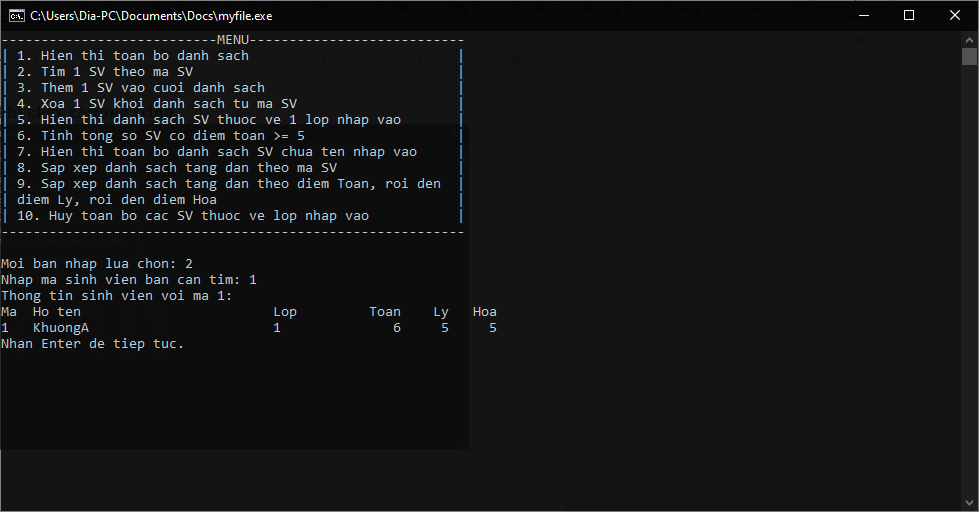
## 3. Full Code

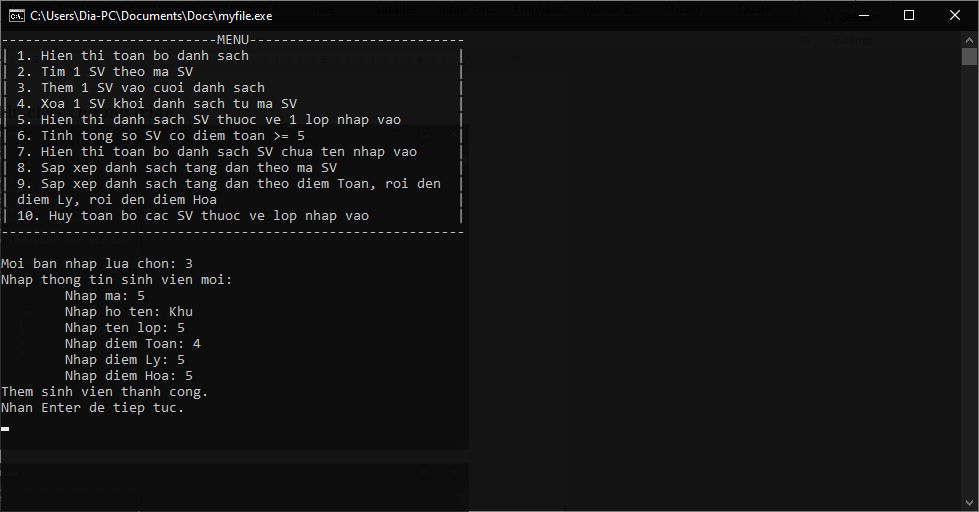
### a. Code

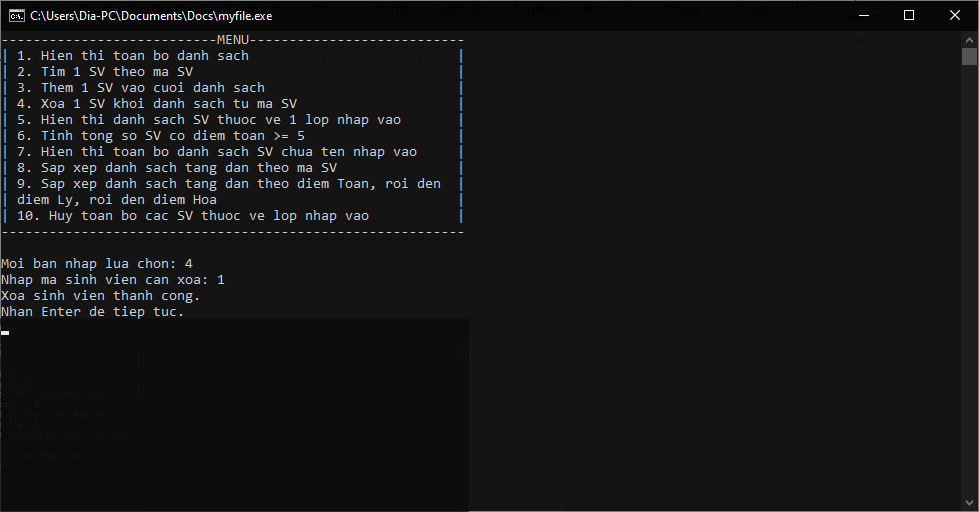
|  |  |
| --- | --- |
|  | #include <iostream>  #include <string>  #include <iomanip>  #ifdef \_\_cplusplus\_\_  #include <cstdlib>  #else  #include <stdlib.h>  #endif  using namespace std;  struct SinhVien {  int ma;  string ten;  string lop;  float toan, ly, hoa;  };  const int MAL = 4;  const int TENL = 30;  const int LOPL = 10;  const int DIEML = 6;  void show(SinhVien sv) {  cout<<left;  cout<<setw(MAL)<<sv.ma;  cout<<setw(TENL)<<sv.ten;  cout<<setw(LOPL)<<sv.lop;  cout<<right;  cout<<setw(DIEML)<<sv.toan;  cout<<setw(DIEML)<<sv.ly;  cout<<setw(DIEML)<<sv.hoa;  cout<<endl;  }  void header() {  cout<<left;  cout<<setw(MAL)<<"Ma";  cout<<setw(TENL)<<"Ho ten";  cout<<setw(LOPL)<<"Lop";  cout<<right;  cout<<setw(DIEML)<<"Toan";  cout<<setw(DIEML)<<"Ly";  cout<<setw(DIEML)<<"Hoa";  cout<<endl;  }  void catchInvalid(string field, float &score) {  while (score < 0 || score > 10) {  cout<<"Khong hop le. Nhap lai."<<endl;  cout<<"\tNhap diem "<<field<<": ";  cin>>score;  }  }  SinhVien insert(int ma) {  SinhVien sv;  sv.ma = ma;  cout<<"\tNhap ho ten: ";  cin.ignore();  fflush(stdin);  getline(cin, sv.ten);  cout<<"\tNhap ten lop: ";  fflush(stdin);  getline(cin, sv.lop);  cout<<"\tNhap diem Toan: ";  cin>>sv.toan;  catchInvalid("Toan", sv.toan);  cout<<"\tNhap diem Ly: ";  cin>>sv.ly;  catchInvalid("Ly", sv.ly);  cout<<"\tNhap diem Hoa: ";  cin>>sv.hoa;  catchInvalid("Hoa", sv.hoa);  return sv;  }  struct Node {  SinhVien info;  struct Node\* prev;  struct Node\* next;  };  typedef Node\* NodePtr;  struct List {  NodePtr pHead;  NodePtr pTail;  };  void Init(List &l) {  l.pHead = NULL;  l.pTail = NULL;  }  NodePtr CreateNode(SinhVien sv) {  NodePtr node = new Node;  node->info = sv;  node->next = NULL;  node->prev = NULL;  return node;  }  int IsEmpty(List l) {  return (l.pHead == NULL && l.pTail == NULL);  }  void ShowNode(NodePtr q) {  show(q->info);  }  void ShowList(List l) {  if (IsEmpty(l)) return;  NodePtr p = l.pHead;  while (p != NULL) {  ShowNode(p);  p = p->next;  }  }  void ShowReverse(List l) {  if (IsEmpty(l)) return;  NodePtr p = l.pTail;  while (p != NULL) {  ShowNode(p);  p = p->prev;  }  }  void InsertFirst(List &l, SinhVien sv) {  NodePtr node = CreateNode(sv);  if (IsEmpty(l)) {  l.pHead = l.pTail = node;  } else {  node->next = l.pHead;  l.pHead->prev = node;  l.pHead = node;  }  }  void InsertLast(List &l, SinhVien sv) {  NodePtr node = CreateNode(sv);  if (IsEmpty(l)) {  l.pHead = l.pTail = node;  } else {  l.pTail->next = node;  node->prev = l.pTail;  l.pTail = node;  }  }  void InsertBefore(List &l, NodePtr &p, SinhVien sv) {  if (p == NULL) return;  if (p == l.pHead) InsertFirst(l, sv);  else {  NodePtr node = CreateNode(sv);  NodePtr before = p->prev;  before->next = node;  node->prev = before;  node->next = p;  p->prev = node;  }  }  void InsertAfter(List &l, NodePtr &p, SinhVien sv) {  if (p == NULL) return;  if (p == l.pHead) InsertLast(l, sv);  else {  NodePtr node = CreateNode(sv);  NodePtr after = p->next;  after->prev = node;  node->next = after;  node->prev = p;  p->next = node;  }  }  void DeleteFirst(List &l) {  if (IsEmpty(l)) return;  else if (l.pHead == l.pTail) {  delete l.pHead;  l.pHead = NULL;  l.pTail = NULL;  } else {  NodePtr p = l.pHead;  l.pHead = l.pHead->next;  l.pHead->prev = NULL;  delete p;  }  }  void DeleteLast(List &l) {  if (IsEmpty(l)) return;  else if (l.pHead == l.pTail) {  delete l.pHead;  l.pHead = NULL;  l.pTail = NULL;  } else {  NodePtr p = l.pTail;  l.pTail = l.pTail->prev;  l.pTail->next = NULL;  delete p;  }  }  void DeleteNode(List &l, NodePtr &p) {  if (p == NULL) return;  if (p == l.pHead) {  DeleteFirst(l);  return;  }  if (p == l.pTail) {  DeleteLast(l);  return;  }  else {  NodePtr left = p->prev;  NodePtr right = p->next;  left->next = right;  if (right != NULL) right->prev = left;  delete p;  }  }  NodePtr Search(List l, int ma) {  if (IsEmpty(l)) return NULL;  NodePtr p = l.pHead;  while (p->info.ma != ma && p->next != NULL) p = p->next;  if (p->info.ma == ma) return p;  else return NULL;  }  NodePtr SearchByClass(NodePtr pHead, string lop) {  if (pHead == NULL) return NULL;  NodePtr p = pHead;  while (p->info.lop.compare(lop) != 0 && p->next != NULL) p = p->next;  if (p->info.lop.compare(lop) == 0) return p;  else return NULL;  }  NodePtr SearchByMathScoreMoreThan(NodePtr pHead, int toan) {  if (pHead == NULL) return NULL;  NodePtr p = pHead;  while (p->info.toan != toan && p->next != NULL) p = p->next;  if (p->info.toan >= toan) return p;  else return NULL;  }  NodePtr SearchByName(NodePtr pHead, string ten) {  if (pHead == NULL) return NULL;  NodePtr p = pHead;  while (p->info.ten.find(ten) == string::npos && p->next != NULL) p = p->next;  if (p->info.ten.find(ten) != string::npos) return p;  else return NULL;  }  int CompareNodes(NodePtr q, NodePtr p, string s, string exp) {  if (s.compare("ma") == 0) {  if (exp.compare(">=") == 0) {  if (q->info.ma >= p->info.ma) return 1;  } else if (exp.compare("<") == 0) {  if (q->info.ma < p->info.ma) return 1;  }  } else if (s.compare("toan") == 0) {  if (exp.compare(">=") == 0) {  if (q->info.toan >= p->info.toan) return 1;  } else if (exp.compare("==") == 0) {  if (q->info.toan == p->info.toan) return 1;  } else if (exp.compare(">") == 0) {  if (q->info.toan > p->info.toan) return 1;  }  } else if (s.compare("ly") == 0) {  if (exp.compare("==") == 0) {  if (q->info.ly == p->info.ly) return 1;  } else if (exp.compare(">") == 0) {  if (q->info.ly > p->info.ly) return 1;  }  } else if (s.compare("hoa") == 0) {  if (exp.compare(">") == 0) {  if (q->info.hoa > p->info.hoa) return 1;  }  }  return 0;  }  void SortedInsert(List &l, NodePtr &p, string s) {  if (IsEmpty(l)) l.pHead = p;  else if (CompareNodes(l.pHead, p, s, ">=")) {  p->next = l.pHead;  p->next->prev = p;  l.pHead = p;  } else {  NodePtr q = l.pHead;  while (q->next != NULL && CompareNodes(l.pHead, p, s, "<")) q = q->next;  p->next = q->next;  if (q->next != NULL) p->next->prev = p;  q->next = p;  p->prev = q;  }  }  void InsertionSort(List &l, string s) {  if (IsEmpty(l)) return;  List res;  Init(res);  NodePtr p = l.pHead;  while (p != NULL) {  NodePtr next = p->next;  p->prev = p->next = NULL;  SortedInsert(res, p, s);  p = next;  }  l = res;  }  void BubbleSort(List &l, string s) {  if (IsEmpty(l)) return;  bool swapped;  NodePtr ptr1;  NodePtr lptr = NULL;  do {  swapped = false;  ptr1 = l.pHead;  while (ptr1->next != lptr) {  if (CompareNodes(ptr1, ptr1->next, s, ">")) {  swap(ptr1->info, ptr1->next->info);  swapped = true;  }  ptr1 = ptr1->next;  }  lptr = ptr1;  }  while (swapped);  }  void BubbleSort2(List &l, string s1, string s) {  if (IsEmpty(l)) return;  bool swapped;  NodePtr ptr1;  NodePtr lptr = NULL;  do {  swapped = false;  ptr1 = l.pHead;  while (ptr1->next != lptr) {  if (CompareNodes(ptr1, ptr1->next, s1, "==") && CompareNodes(ptr1, ptr1->next, s, ">")) {  swap(ptr1->info, ptr1->next->info);  swapped = true;  }  ptr1 = ptr1->next;  }  lptr = ptr1;  }  while (swapped);  }  void BubbleSort3(List &l, string s1, string s2, string s) {  if (IsEmpty(l)) return;  bool swapped;  NodePtr ptr1;  NodePtr lptr = NULL;  do {  swapped = false;  ptr1 = l.pHead;  while (ptr1->next != lptr) {  if (CompareNodes(ptr1, ptr1->next, s1, "==") && CompareNodes(ptr1, ptr1->next, s2, "==") && CompareNodes(ptr1, ptr1->next, s, ">")) {  swap(ptr1->info, ptr1->next->info);  swapped = true;  }  ptr1 = ptr1->next;  }  lptr = ptr1;  }  while (swapped);  }  void Ham1(List l) {  if (!IsEmpty(l)) {  cout<<"Danh sach sinh vien:"<<endl;  header();  ShowList(l);  } else {  cout<<"Khong co du lieu hien thi."<<endl;  }  }  void Ham2(List l) {  int ma;  cout<<"Nhap ma sinh vien ban can tim: ";  cin>>ma;  NodePtr p = Search(l, ma);  if (p != NULL) {  cout<<"Thong tin sinh vien voi ma "<<ma<<":"<<endl;  header();  ShowNode(p);  } else {  cout<<"Khong co sinh vien nao voi ma "<<ma<<"."<<endl;  }  }  void Ham3(List &l) {  bool check;  int ma;  cout<<"Nhap thong tin sinh vien moi:"<<endl;  do {  check = false;  cout<<"\tNhap ma: ";  cin>>ma;  NodePtr p = Search(l, ma);  if (p != NULL) {  cout<<"Ma "<<ma<<" da ton tai!"<<endl;  check = true;  }  } while (check);  SinhVien sv = insert(ma);  InsertLast(l, sv);  cout<<"Them sinh vien thanh cong."<<endl;  }  void Ham4(List &l) {  int ma;  cout<<"Nhap ma sinh vien can xoa: ";  cin>>ma;  NodePtr p = Search(l, ma);  if (p != NULL) {  DeleteNode(l, p);  cout<<"Xoa sinh vien thanh cong."<<endl;  } else {  cout<<"Khong co sinh vien nao voi ma "<<ma<<"."<<endl;  }  }  void Ham5(List l) {  string lop;  cout<<"Nhap ten lop can tim: ";  cin.ignore();  fflush(stdin);  getline(cin, lop);  NodePtr p = SearchByClass(l.pHead, lop);  if (p != NULL) {  cout<<"Danh sach sinh vien lop "<<lop<<":"<<endl;  header();  while (p != NULL) {  ShowNode(p);  p = SearchByClass(p->next, lop);  }  } else {  cout<<"Khong co sinh vien nao thuoc lop "<<lop<<"."<<endl;  }  }  void Ham6(List l) {  int toan = 5, count = 0;  cout<<"Tong so sinh vien co diem Toan >= "<<toan<<": ";  NodePtr p = SearchByMathScoreMoreThan(l.pHead, toan);  while (p != NULL) {  count++;  p = SearchByMathScoreMoreThan(p->next, toan);  }  cout<<count<<endl;  }  void Ham7(List l) {  string ten;  cout<<"Nhap ten sinh vien can tim: ";  cin.ignore();  fflush(stdin);  getline(cin, ten);  NodePtr p = SearchByName(l.pHead, ten);  if (p != NULL) {  cout<<"Danh sach sinh vien co ten "<<ten<<":"<<endl;  header();  while (p != NULL) {  ShowNode(p);  p = SearchByName(p->next, ten);  }  } else {  cout<<"Khong co sinh vien nao co ten "<<ten<<"."<<endl;  }  }  void Ham8(List &l) {  if (IsEmpty(l)) {  cout<<"Khong co sinh vien nao de sap xep."<<endl;  } else {  InsertionSort(l, "ma");  cout<<"Sap xep sinh vien thanh cong."<<endl;  }  }  void Ham9(List &l) {  if (IsEmpty(l)) {  cout<<"Khong co sinh vien nao de sap xep."<<endl;  } else {  BubbleSort(l, "toan");  BubbleSort2(l, "toan", "ly");  BubbleSort3(l, "toan", "ly", "hoa");  cout<<"Sap xep sinh vien thanh cong."<<endl;  }  }  void Ham10(List &l) {  string lop;  cout<<"Nhap ten lop can huy sinh vien: ";  cin.ignore();  fflush(stdin);  getline(cin, lop);  NodePtr p = SearchByClass(l.pHead, lop);  if (p != NULL) {  while (p != NULL) {  NodePtr next = p->next;  DeleteNode(l, p);  p = SearchByClass(next, lop);  }  cout<<"Xoa sinh vien thanh cong."<<endl;  } else {  cout<<"Khong co sinh vien nao thuoc lop "<<lop<<"."<<endl;  }  }  void pause() {  string s;  cout<<"Nhan Enter de tiep tuc."<<endl;  cin.ignore();  fflush(stdin);  getline(cin, s);  }  void input(List &l) {  int n;  do {  cout<<"Nhap so sinh vien: ";  cin>>n;  if (n < 0) cout<<"Khong hop le. Nhap lai."<<endl;  } while (n < 0);  for (int i = 0; i < n; i++) {  Ham3(l);  }  }  void menu(List l) {  bool check = true;  do {  if (system("CLS")) system("clear");  cout<<"---------------------------MENU---------------------------"<<endl;  cout<<"| 1. Hien thi toan bo danh sach |"<<endl;  cout<<"| 2. Tim 1 SV theo ma SV |"<<endl;  cout<<"| 3. Them 1 SV vao cuoi danh sach |"<<endl;  cout<<"| 4. Xoa 1 SV khoi danh sach tu ma SV |"<<endl;  cout<<"| 5. Hien thi danh sach SV thuoc ve 1 lop nhap vao |"<<endl;  cout<<"| 6. Tinh tong so SV co diem toan >= 5 |"<<endl;  cout<<"| 7. Hien thi toan bo danh sach SV chua ten nhap vao |"<<endl;  cout<<"| 8. Sap xep danh sach tang dan theo ma SV |"<<endl;  cout<<"| 9. Sap xep danh sach tang dan theo diem Toan, roi den |"<<endl;  cout<<"| diem Ly, roi den diem Hoa |"<<endl;  cout<<"| 10. Huy toan bo cac SV thuoc ve lop nhap vao |"<<endl;  cout<<"----------------------------------------------------------"<<endl;  cout<<endl;  int choice;  cout<<"Moi ban nhap lua chon: ";  cin>>choice;  switch(choice) {  case 1: Ham1(l); break;  case 2: Ham2(l); break;  case 3: Ham3(l); break;  case 4: Ham4(l); break;  case 5: Ham5(l); break;  case 6: Ham6(l); break;  case 7: Ham7(l); break;  case 8: Ham8(l); break;  case 9: Ham9(l); break;  case 10: Ham10(l); break;  default: check = false; break;  }  if (check) pause();  } while (check);  }  int main() {  List l;  Init(l);  input(l);  menu(l);  return 0;  } |

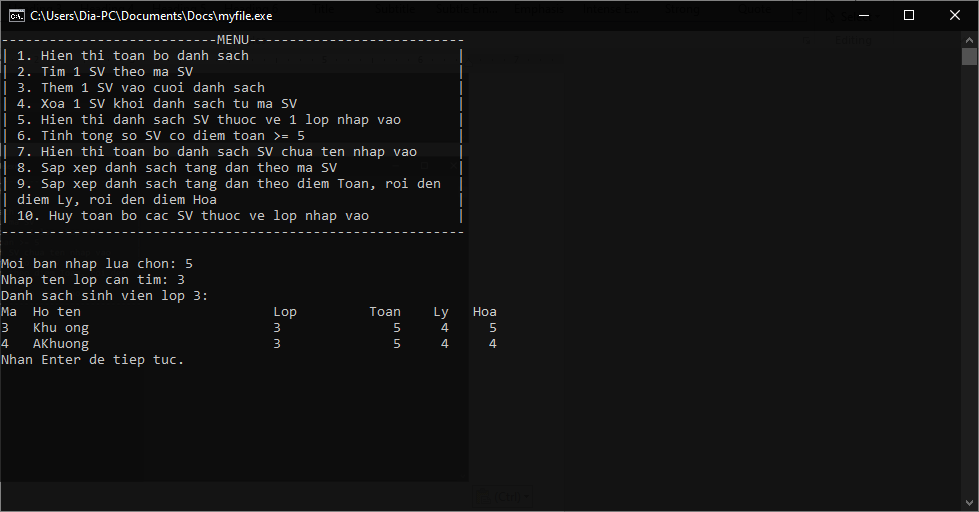
### b. Các kết quả sau khi chạy chương trình

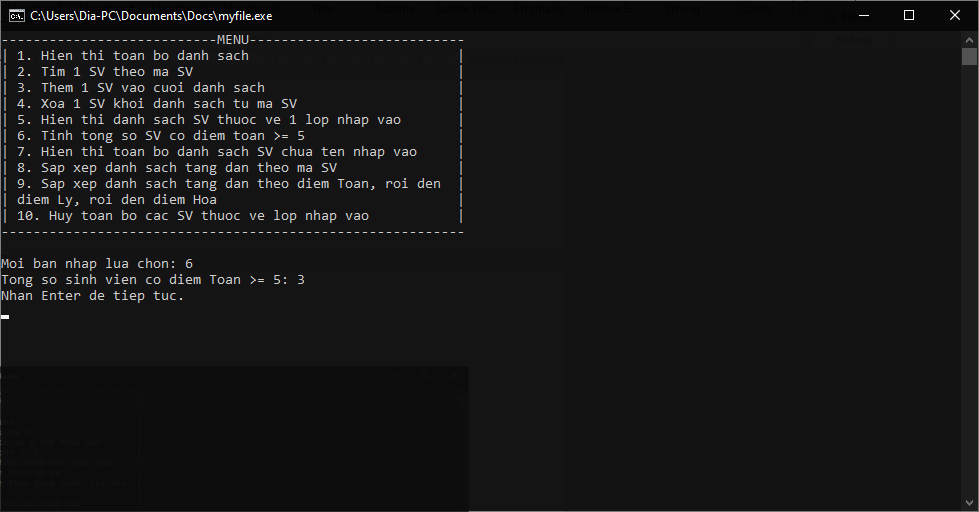


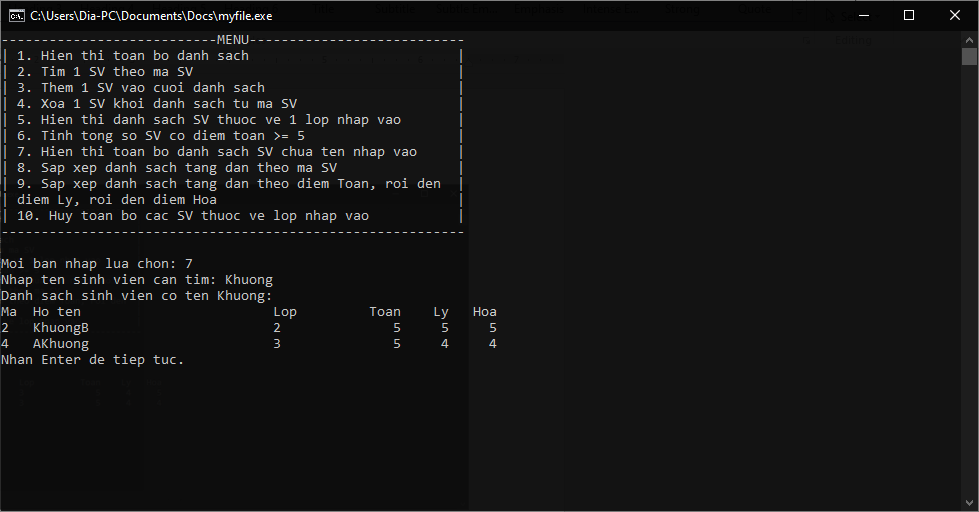


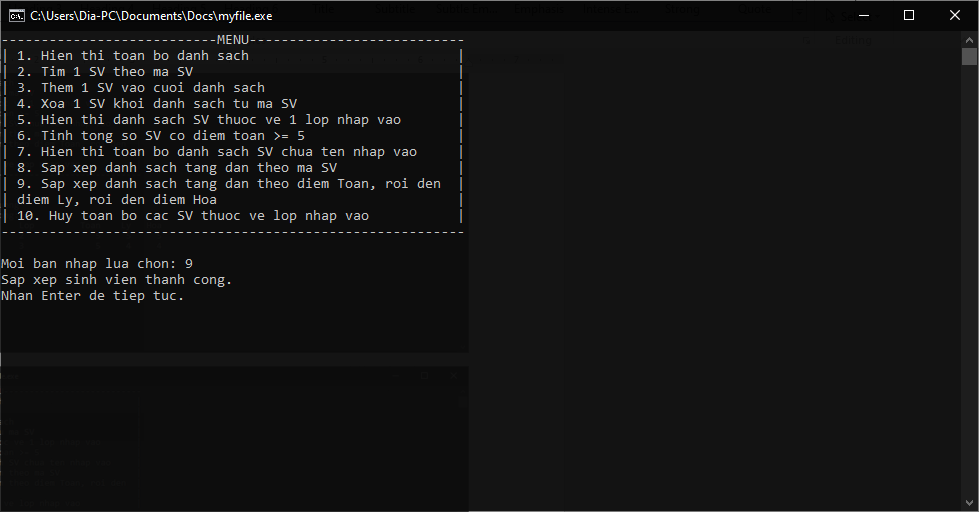


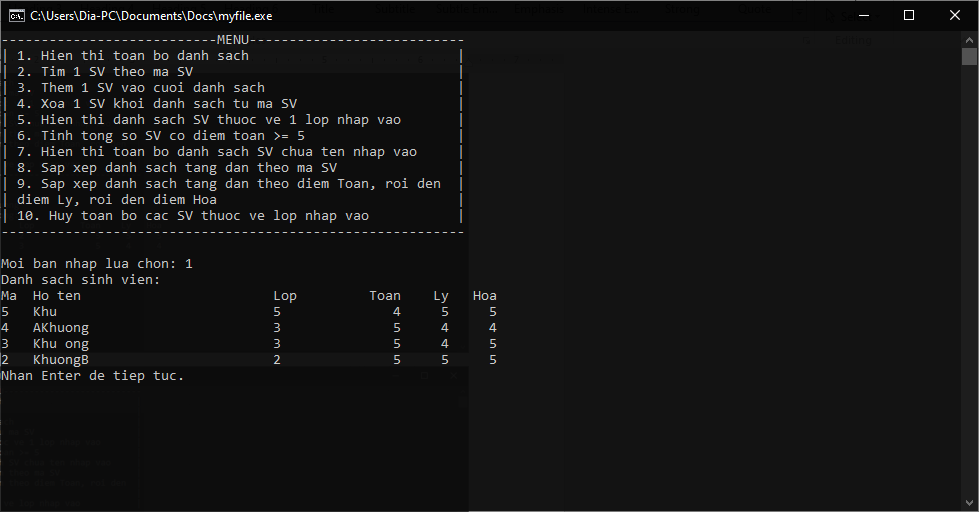


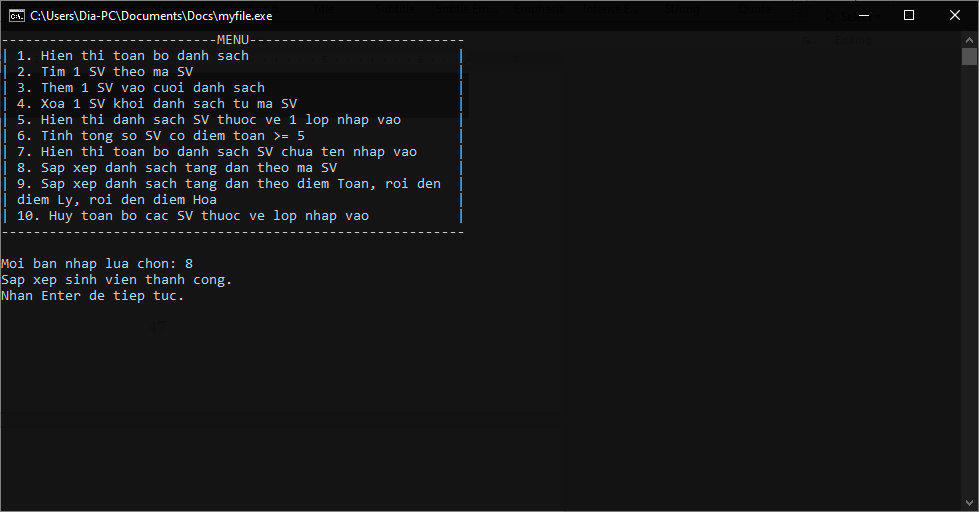


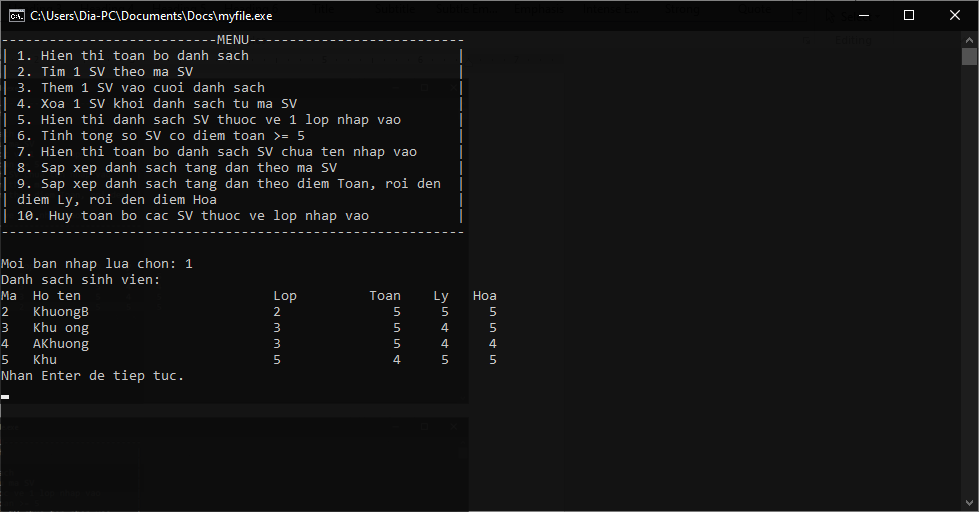


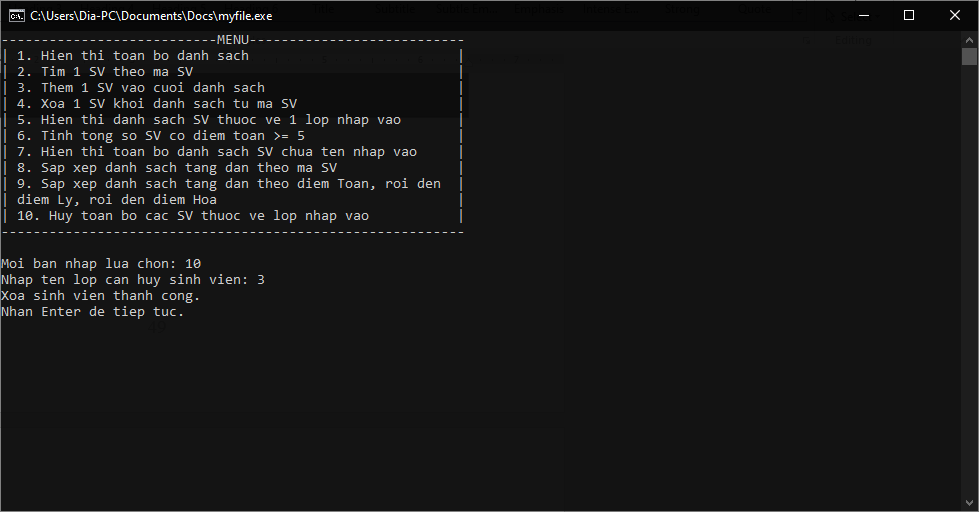


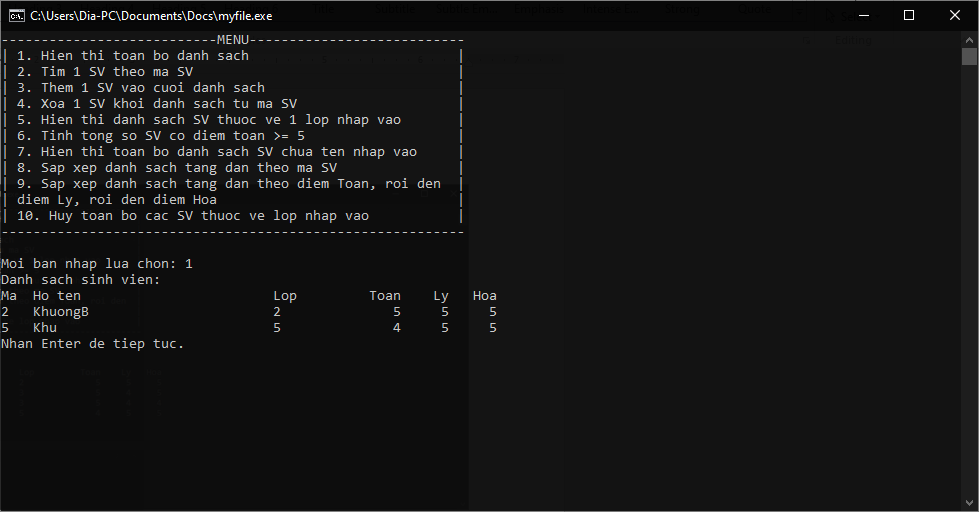












# Phần III: Tổng Kết

Danh sách liên kết là một trong những cấu trúc dữ liệu đơn giản và phổ biến nhất. Nó có thể được dùng để hiện thực một số [kiểu dữ liệu trừu tượng](https://vi.wikipedia.org/wiki/Ki%E1%BB%83u_d%E1%BB%AF_li%E1%BB%87u_tr%E1%BB%ABu_t%C6%B0%E1%BB%A3ng) phổ biến khác, bao gồm [danh sách](https://vi.wikipedia.org/w/index.php?title=Danh_s%C3%A1ch_(ki%E1%BB%83u_d%E1%BB%AF_li%E1%BB%87u_tr%E1%BB%ABu_t%C6%B0%E1%BB%A3ng)&action=edit&redlink=1) (list), [ngăn xếp](https://vi.wikipedia.org/wiki/Ng%C4%83n_x%E1%BA%BFp) (stack), [hàng đợi](https://vi.wikipedia.org/wiki/H%C3%A0ng_%C4%91%E1%BB%A3i), [mảng kết hợp](https://vi.wikipedia.org/w/index.php?title=M%E1%BA%A3ng_k%E1%BA%BFt_h%E1%BB%A3p&action=edit&redlink=1), và [S-expression](https://vi.wikipedia.org/w/index.php?title=S-expression&action=edit&redlink=1), mặc dù không có gì lạ khi hiện thực các cấu trúc dữ liệu đó mà không dựa trên nền tảng của danh sách liên kết.

Lợi ích chính của danh sách liên kết so với [mảng](https://vi.wikipedia.org/wiki/M%E1%BA%A3ng_(c%E1%BA%A5u_tr%C3%BAc_d%E1%BB%AF_li%E1%BB%87u)) thông thường là các phần tử danh sách có thể được chèn hay xóa một cách dễ dàng mà không cần phân bổ lại hoặc sắp xếp lại toàn bộ cấu trúc vì các mục dữ liệu không cần được lưu trữ [liên tục](https://vi.wiktionary.org/wiki/contiguous) trong bộ nhớ hay trên đĩa, trong khi tái cấu trúc một mảng tại [thời gian chạy](https://vi.wikipedia.org/w/index.php?title=Run_time_(program_lifecycle_phase)&action=edit&redlink=1) là một hoạt động tốn kém hơn nhiều. Danh sách liên kết cho phép chèn hay xóa nút tại bất kì điểm nào trong danh sách.

Mặc khác, vì bản thân danh sách liên kết được liên kết đơn giản nên không cho phép [truy cập ngẫu nhiên](https://vi.wikipedia.org/w/index.php?title=Truy_c%E1%BA%ADp_ng%E1%BA%ABu_nhi%C3%AAn&action=edit&redlink=1) tới dữ liệu hoặc bất kì hình thức đánh chỉ mục hiệu quả nào, nhiều toán tử cơ bản như lấy nút cuối cùng của danh sách, tìm một nút có chứa dữ liệu đã cho, hay tìm vị trí của nút để chèn một nút mới sẽ yêu cầu lặp qua hầu hết hoặc tất cả các phần tử của danh sách. Những ưu điểm và nhược điểm của danh sách liên kết được đưa ra dưới đây. Danh sách liên kết là động, vì vậy độ dài của nó có thể tăng hay giảm khi cần thiết. Mỗi nút không cần phải theo nút trước đó trong bộ nhớ.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Em xin chân thành cảm ơn nhà trường, khoa chuyên môn và Giảng viên Trần Thị Lan Anh đã tạo điều kiện thuận lợi cho chúng em hoàn thành bài tập lớn cuối kỳ. Em cũng cảm ơn cô vì đã theo dõi hết bài tập của em.

Em xin chân thành cảm ơn!

*Hà Nội, ngày 23 tháng 12 năm 2021*