

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**KHOA CÔNG NGHỆ THÔNG TIN**

----o0o----



**BÁO CÁO BÀI TẬP LỚN**

**CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

***Đề tài:Ứng dụng lớp vector và list để quản lý sinh viên.***

***Xây dựng lớp stack và ứng dụng.***

***Giảng viên hướng dẫn:***

***TS. Phạm Xuân Tích***

***Sinh viên thực hiện:***

***Nguyễn Tiến Mạnh – 231230836***

***Lớp:***

***Công nghệ thông tin 3 – K64***

**Hà Nội, tháng 10 năm 2024**

MỤC LỤC

[I. GIỚI THIỆU 4](#_Toc149771016)

[1. Lý do và động lực 4](#_Toc149771017)

[2. Định nghĩa bài toán 4](#_Toc149771018)

[3. Các nhiệm vụ và mức độ hoàn thành 5](#_Toc149771019)

[II. PHƯƠNG PHÁP LỰA CHỌN 6](#_Toc149771020)

[1. Cấu trúc dữ liệu 6](#_Toc149771021)

[III. TRIỂN KHAI CÀI ĐẶT 10](#_Toc149771022)

[1. Ngôn ngữ lập trình và thư viện 12](#_Toc149771023)

[2. Tổ chức chương trình và đóng gói 12](#_Toc149771024)

[IV. PHÂN TÍCH CHƯƠNG TRÌNH 13](#_Toc149771025)

[1. Sử dụng vector hoặc list để tạo chương trình quản lý sinh viên 13](#_Toc149771026)

[1.1 Xây dụng class sinh viên và hàm nhập xuất cơ bản 13](#_Toc149771027)

[1.2 Quản lý danh sách sinh viên sử dụng vector và linked list 14](#_Toc149771028)

[2. Cài đặt cấu trúc dữ liệu Stack 17](#_Toc149771030)

[V. KẾT QUẢ THỰC NGHIỆM 2](#_Toc149771031)3

[1. Dữ liệu 2](#_Toc149771032)3

[2. Các kết quả 2](#_Toc149771033)4

[VI. KẾT LUẬN 2](#_Toc149771034)8

[1. Đánh giá mức độ hoàn thành 2](#_Toc149771035)8

[2. Bài học rút ra 2](#_Toc149771036)8

[3. Khó khăn với môn học 2](#_Toc149771037)8

[VII. LỜI CẢM ƠN 2](#_Toc149771038)9

**DANH SÁCH HÌNH ẢNH**

|  |  |  |
| --- | --- | --- |
| **Hình** | **Nội dung** | **Trang** |
| 1 | Minh họa vector và một số phương thức cơ bản | 6 |
| 2 | Minh họa Liked List | 8 |
| 3 | Minh họa Stack | 10 |
| 4 | Chương trình đọc và in kết quả ra màn hình | 24 |
| 5 | Tìm sinh viên theo mã sinh viên hoặc tên và in kết quả ra màn hình | 25 |
| 6 | Tìm sinh viên có điểm thấp nhất và in kết quả ra màn hình | 25 |
| 7 | In ra màn hình kết quả mô phỏng | 27 |
| 8 | In ra màn hình kết quả bài toán | 28 |

# GIỚI THIỆU

## Lý do và động lực

Để đáp ứng yêu cầu môn học và mở rộng hiểu biết về ngôn ngữ lập trình nói chung và môn Cấu trúc dữ liệu và giải thuật nói riêng, em đã hướng tới đề tài với các tiêu chí sau:

* Có tính ứng dụng cao.
* Áp dụng được kiến thức đã học.
* Học hỏi được kiến thức mới.

## Định nghĩa bài toán

*Bài toán A: (Bài toán quản lý)*

*Input:* Chương trình cho phép người dùng quản lý thông tin sinh viên, bao gồm thêm, xóa, tìm kiếm, sắp xếp, và hiện thị danh sách sinh viên bằng cách sử dụng ***App Menu***.

*Ouput:* Chương trình giúp quản lý và thao tác với thông tin sinh viên một cách hiệu quả.

*Bài toán B: (Bài số 32 trong danh sách BTL)*

1. Xây dựng stack bằng 2 cách : Mảng và danh sách liên kết.
2. Mô phỏng hoạt động của stack :

*Input:* Dòng đầu chứa số nguyên dương T là số truy vấn

T dòng tiếp theo, mỗi dòng chứa một truy vấn

*Ouput:* Ứng với mỗi truy vấn, loại 3 in kết quả ra 1 dòng

1. Ứng dụng stack lập trình giải bài toán sau :

*Input:* Gồm một dòng duy nhất chứa xâu ký tự biểu diễn trong dãy ngoặc

*Ouput:* Gồm *n* dòng, mỗi dòng gồm hai số u, v là chỉ số của các cặp ngoặc tương ứng nhau, thứ tự liệt kê tăng dần theo chỉ số của dấu ngoặc đóng.

## Các nhiệm vụ và mức độ hoàn thành

|  |  |
| --- | --- |
| Nhiệm vụ | Mức độ hoàn thành |
| * Nghiên cứu lớp vector, list * Lên ý tưởng và phân tích bài toán A. * Triển khai một cấu trúc dữ liệu trừu tượng vector (list). * Sử dụng vector hoặc list để tạo chương trình quản lý sinh viên hỗ trợ thao tác thêm, xóa, tìm kiếm, sắp xếp và hiển thị thông tin sinh viên. | Đã hoàn thành |
| * Tìm hiểu về Stack. * Lên ý tưởng bài toán B. * Triển khai và cài đặt cấu trúc Stack (bằng mảng và bằng danh sách liên kết đơn). * Mô phỏng cách hoạt động và ứng dụng vào giải bài toán đã cho. | Đã hoàn thành |
| * Làm báo cáo. * Lưu trữ code bằng Github. | Đã hoàn thành |

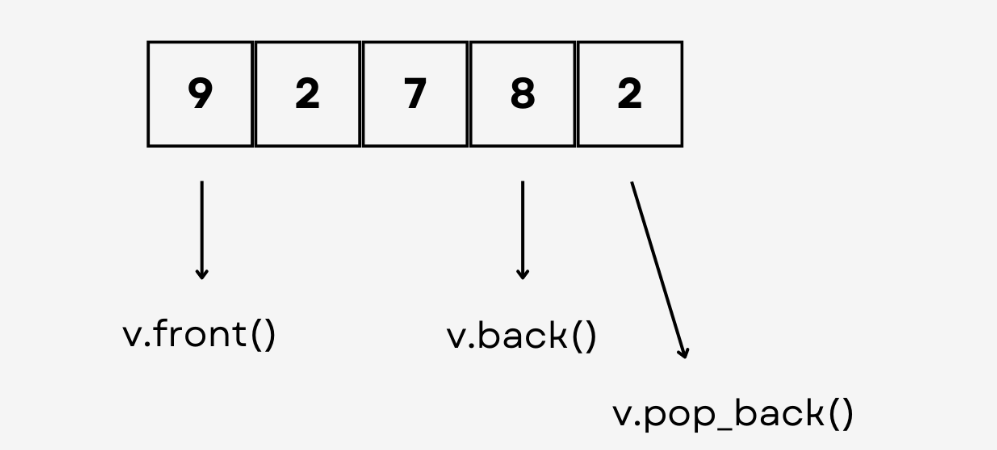
# PHƯƠNG PHÁP LỰA CHỌN

## Cấu trúc dữ liệu

* **VECTOR:**

Vector C++ là một lớp, một cấu trúc dữ liệu có trong thư viện của C++ Standard Template Library dùng để lưu các đối tượng khác giống với mảng (Array). C++ rất mạnh về tính linh hoạt và cấu trúc chính vì thế vector được sinh ra nhằm khắc phục các mặt hạn chế của mảng. Nếu như mảng chỉ có một kích thước cố định khi khai báo thì vector hoàn toàn có thể thay đổi trong khi chương trình đang hoạt động. Tức là một đối tượng vector có thể linh động về kích thước tương tự như con trỏ.

Kích thước không cố định chính là điểm nổi bật nhất của vector. Tuy nhiên không dừng lại ở đó, đối tượng này còn được trang bị thêm khá nhiều phương thức xử lý thông dụng như sort, insert, delete, legth, … Chính nhờ tính linh hoạt của nó, đôi khi sẽ hữu ích hơn trong việc giải quyết các bài toán lập trình.



Hình 1 Minh họa vector và một số phương thức cơ bản.

Các thao tác trên Vector:

**Modifiers**

1.**push\_back()**: Hàm đẩy một phần tử vào vị trí sau cùng của vector.

2. **assign()**: Nó gán một giá trị mới cho các phần tử vector bằng cách thay thế các giá trị cũ.

3.**pop\_back()**: Hàm pop\_back () được sử dụng để xóa đi phần tử cuối cùng một vector.

4. **insert()**: Hàm này chèn các phần tử mới vào trước phần tử trước vị trí được trỏ bởi vòng lặp.

5. **erase()**: Hàm được sử dụng để xóa các phần tử tùy theo vị trí vùng chứa.

6. **swap()**: Hàm được sử dụng để hoán đổi nội dung của một vector này với một vector khác cùng kiểu. Kích thước có thể khác nhau.

7. **clear()**: Hàm được sử dụng để loại bỏ tất cả các phần tử của vùng chứa vector.

**Iterators**

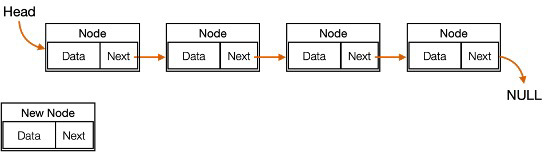
1. **begin()**: đặt iterator đến phần tử đầu tiên trong vector.
2. **end()**: đặt iterator đến sau phần tử cuối cùng trong vector.
3. **rbegin()**: đặt reverse iterator (trình lặp đảo) đến phần tử cuối cùng trong vector (reverse begin). Nó di chuyển từ phần tử cuối cùng đến phần tử đầu tiên.
4. **rend()**: đặt reverse iterator (trình lặp đảo) đến phần tử đầu tiên trong vector (reverse end).
5. **cbegin()**: đặt constant iterator (trình vòng lặp) đến phần tử đầu tiên trong vector.
6. **cend()**: đặt constant iterator (trình vòng lặp) đến phần tử cuối cùng trong vector.
7. **crbegin()**: đặt constant reverse iterator (trình lặp đảo liên tục) đến phần tử cuối cùng trong vector (reverse begin). Nó di chuyển từ phần tử cuối cùng đến phần tử đầu tiên.
8. **crend()**: đặt constant reverse iterator (trình lặp đảo liên tục) đến phần tử đầu tiên trong vector.

**Capacity**

1. **size()**: hàm sẽ trả về số lượng phần tử đang được sử dụng trong vector.
2. **max\_size()**: hàm trả về số phần tử tối đa mà vector có thể chứa.
3. **capacity()**: hàm trả về số phần tử được cấp phát cho vector nằm trong bộ nhớ.
4. **resize(n)**: Hàm này thay đổi kích thước vùng chứa để nó chứa đủ n phần tử. Nếu kích thước hiện tại của vector lớn hơn n thì các phần tử phía sau sẽ bị xóa khỏi vector và ngược lại nếu kích thước hiện tại nhỏ hơn n thì các phần tử bổ sung sẽ được chèn vào phía sau vector.
5. **empty()**: Trả về liệu vùng chứa có trống hay không, nếu trống thì trả về True, nếu có phần tử thì trả về False.
6. **shrink\_to\_fit()**: Giảm dung lượng của vùng chứa để phù hợp với kích thước của nó và hủy tất cả các phần tử vượt quá dung lượng.
7. **reserve(n)**: hàm cấp cho vector số dung lượng vừa đủ để chứa n phần tử.

**Element access**

1. **at(g)**: Trả về một tham chiếu đến phần tử ở vị trí ‘g’ trong vector.
2. **data()**: Trả về một con trỏ trực tiếp đến (memory array) bộ nhớ mảng được vector sử dụng bên trong để lưu trữ các phần tử thuộc sở hữu của nó.
3. **front()**: hàm dùng để lấy ra phần tử đầu tiên của vector.
4. **back()**: hàm dùng để lấy ra phần tử cuối cùng của vector.



Hình 2 Minh họa linked list.

* **LINKED LIST:**

Lớp List cung cấp một cấu trúc dữ liệu tuyến tính cho phép lưu trữ và quản lý một tập hợp các phần tử. Với khả năng thêm, xóa, và truy xuất các phần tử tại bất kỳ vị trí nào trong danh sách, lớp List cho phép bạn dễ dàng thao tác với dữ liệu. Lớp này hỗ trợ các phương thức như tìm kiếm, sắp xếp và lặp qua các phần tử, giúp người dùng quản lý và xử lý tập dữ liệu một cách hiệu quả.

### Các thao tác trên List:

**Modifiers:**

1. **push\_back()**: Thêm phần tử vào cuối danh sách.
2. **push\_front()**: Thêm phần tử vào đầu danh sách.
3. **pop\_back()**: Xóa phần tử ở cuối danh sách.
4. **pop\_front()**: Xóa phần tử ở đầu danh sách.
5. **insert()**: Chèn các phần tử mới trước vị trí được chỉ định.
6. **erase()**: Xóa một hoặc nhiều phần tử khỏi danh sách tại vị trí được chỉ định.
7. **clear()**: Xóa tất cả các phần tử của danh sách.
8. **remove(val)**: Xóa tất cả các phần tử có giá trị bằng val.
9. **remove\_if(cond)**: Xóa tất cả các phần tử thỏa mãn điều kiện cond.
10. **resize(n)**: Thay đổi kích thước danh sách sao cho chứa đủ n phần tử.
11. **swap()**: Hoán đổi nội dung của danh sách này với một danh sách khác.

**Iterators:**

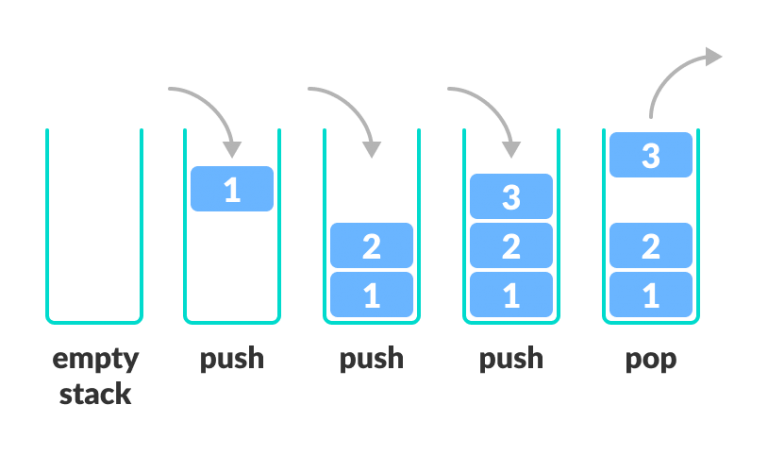
1. **begin()**: Đặt iterator đến phần tử đầu tiên của danh sách.
2. **end()**: Đặt iterator đến sau phần tử cuối cùng của danh sách.
3. **rbegin()**: Đặt reverse iterator đến phần tử cuối cùng của danh sách.
4. **rend()**: Đặt reverse iterator đến trước phần tử đầu tiên của danh sách.
5. **cbegin()**: Đặt constant iterator đến phần tử đầu tiên của danh sách.
6. **cend()**: Đặt constant iterator đến phần tử cuối cùng của danh sách.
7. **crbegin()**: Đặt constant reverse iterator đến phần tử cuối cùng của danh sách.
8. **crend()**: Đặt constant reverse iterator đến trước phần tử đầu tiên của danh sách.

**Capacity:**

1. **size()**: Trả về số lượng phần tử trong danh sách.
2. **max\_size()**: Trả về số phần tử tối đa mà danh sách có thể chứa.
3. **empty()**: Trả về true nếu danh sách trống, ngược lại trả về false.

**Operations:**

1. **sort()**: Sắp xếp các phần tử của danh sách theo thứ tự tăng dần.
2. **reverse()**: Đảo ngược thứ tự các phần tử trong danh sách.
3. **merge()**: Hợp nhất hai danh sách đã được sắp xếp.
4. **unique()**: Xóa các phần tử trùng liên tiếp trong danh sách.



Hình 3 Minh họa stack.

* **STACK:**

Lớp Stack là một cấu trúc dữ liệu kiểu LIFO (Last In, First Out), cho phép thêm và xóa các phần tử theo thứ tự mà chúng được thêm vào. Khi một phần tử được thêm vào stack, nó sẽ được đặt ở trên cùng, và khi xóa, phần tử này sẽ được loại bỏ đầu tiên. Lớp Stack cung cấp các phương thức như push để thêm phần tử và pop để xóa phần tử, giúp quản lý các tác vụ theo thứ tự ngược lại, rất hữu ích trong các ứng dụng như xử lý biểu thức, quay lui (backtracking), và quản lý các trạng thái tạm thời.

### Các thao tác trên Stack:

#### **Modifiers:**

1. **push(val)**: Thêm phần tử val vào đỉnh của stack.
2. **pop()**: Xóa phần tử ở đỉnh của stack.
3. **emplace(val)**: Tạo và thêm một phần tử val vào đỉnh của stack mà không cần tạo bản sao.
4. **clear()**: Xóa tất cả các phần tử trong stack.

#### **Accessors:**

1. **top()**: Trả về giá trị của phần tử ở đỉnh của stack mà không xóa nó.
2. **empty()**: Trả về true nếu stack trống, ngược lại trả về false.
3. **size()**: Trả về số lượng phần tử hiện có trong stack.

### **Các thao tác khác:**

1. **swap(stack)**: Hoán đổi nội dung của stack này với một stack khác.
2. **copy**: Tạo bản sao của stack hiện tại.

# TRIỂN KHAI CÀI ĐẶT

## Ngôn ngữ lập trình và thư viện

* **Ngôn ngữ lập trình:** Sử dụng C++
* C ++ là một ngôn ngữ lập trình được phát triển bởi Bjarne Stroustrup vàonăm 1979 tại Bell Labs.
* Tính phổ biến: C++ là một trong những ngôn ngữ lập trình phổ biến nhất trên thế giới.
* Tính thực thi nhanh
* Thư viện đầy đủ: Có rất nhiều tài nguyên cho người lập trình bằng C++
* Đa mô hình: C++ cho phép lập trình theo cấu trúc tuyến tính, hướng chức năng, hướng đối tượng đa dạng tùy theo nhu cầu người lập trình.
* **Thư viện:** Sử dụng thư viện trong C++

## Tổ chức chương trình và đóng gói

* **Tổ chức chương trình:**
* Viết chương trình C++ bằng DEVC++ hoặc Visual Studio Code, mỗi đoạn code xử lý một chức năng.
* Lưu trữ code bằng Github.
* **Đóng gói:**
* Các phần code và liên quan đến chương trình đóng gói ở Github (Link: https://github.com/B1CTommy/Takatori-demo/tree/main/btlctdl%26gt)
* Bài báo cáo: Báo cáo Bài Tập Lớn.docx

# PHÂN TÍCH CHƯƠNG TRÌNH

# Sử dụng vector hoặc list để tạo chương trình quản lý sinh viên :

### Xây dựng class sinh viên và hàm nhập xuất cơ bản

|  |
| --- |
| /\*-----MANAGEMENT PROGRAM-----\*/      class SV {  private:  string msv;  string ten;  long tuoi;  double diem;  public:  SV() {  msv = "";  ten = "";  tuoi = 0;  diem = 0.0;  }  ~SV() {}  friend istream& operator>>(istream &inp, SV &sv) {  inp >> sv.msv;  inp.ignore();  getline(inp, sv.ten);  inp >> sv.tuoi;  inp >> sv.diem;  return inp;  }  friend ostream& operator<<(ostream &oup, const SV &sv) {  oup << left << setw(10) << sv.msv  << left << setw(20) << sv.ten  << left << setw(10) << sv.tuoi  << left << setw(10) << sv.diem << endl;  return oup;  }  bool operator<(const SV &other) const {  return this->diem < other.diem;  }  string getMsv() const {  return msv;  }  string getTen() const {  return ten;  }  double getDiem() const {  return diem;  }  }; |

### 1.2 Quản lý danh sách sinh viên sử dụng vector và linked list

Quản lý sinh viên sử dụng vector

|  |
| --- |
| class VectorSV {  private:  vector<SV> svs;  public:  void addStudent(const SV& sv) {  svs.push\_back(sv);  }  void opList() const {  cout << left << setw(10) << "Ma SV"  << left << setw(20) << "Ten SV"  << left << setw(10) << "Tuoi"  << left << setw(10) << "Diem" << endl;  for (const auto &sv : svs) {  cout << sv;  }  }  // Hàm thêm sinh viên vào danh sách  void addSV() {  SV sv;  cout << "Nhap thong tin sinh vien moi:\n";  cin >> sv;  svs.push\_back(sv);  }  void sortList() {  sort(svs.begin(), svs.end());  }  SV findMax() const {  if (svs.empty()) {  cout << "Danh sach sinh vien trong!\n";  return SV();  }  return \*max\_element(svs.begin(), svs.end());  }  SV findMin() {  if (svs.empty()) {  cout << "Danh sach sinh vien trong!\n";  return SV();  }  return \*min\_element(svs.begin(), svs.end());  }  // Tìm kiếm sinh viên theo mã hoặc tên  void searchStudent(const string &query) const {  bool found = false;  for (const auto &sv : svs) {  if (sv.getMsv() == query || sv.getTen() == query) {  cout << "Sinh vien tim thay:\n" << sv;  found = true;  }  }  if (!found) {  cout << "Khong tim thay sinh vien nao.\n";  }  }  // Xóa sinh viên theo mã hoặc tên  void deleteStudent(const string &query) {  auto it = remove\_if(svs.begin(), svs.end(), [&](const SV &sv) {  return sv.getMsv() == query || sv.getTen() == query;  });  if (it != svs.end()) {  svs.erase(it, svs.end());  cout << "Da xoa sinh vien co ma hoac ten: " << query << endl;  } else {  cout << "Khong tim thay sinh vien de xoa.\n";  }  }  }; |

Quản lý sinh viên sử dụng linked list

|  |
| --- |
| class ListSV {  private:  list<SV> svs;  public:  void addStudent(const SV& sv) {  svs.push\_back(sv);  }  void opList() const {  cout << left << setw(10) << "Ma SV"  << left << setw(20) << "Ten SV"  << left << setw(10) << "Tuoi"  << left << setw(10) << "Diem" << endl;  for (const auto &sv : svs) {  cout << sv;  }  }  // Hàm thêm sinh viên vào danh sách  void addSV() {  SV sv;  cout << "Nhap thong tin sinh vien moi:\n";  cin >> sv;  svs.push\_back(sv);  }  void sortList() {  svs.sort();  }  SV findMax() const {  if (svs.empty()) {  cout << "Danh sach sinh vien trong!\n";  return SV();  }  return \*max\_element(svs.begin(), svs.end());  }  SV findMin() {  if (svs.empty()) {  cout << "Danh sach sinh vien trong!\n";  return SV();  }  return \*min\_element(svs.begin(), svs.end());  }  // Tìm kiếm sinh viên theo mã hoặc tên  void searchStudent(const string &query) const {  bool found = false;  for (const auto &sv : svs) {  if (sv.getMsv() == query || sv.getTen() == query) {  cout << "Sinh vien tim thay:\n" << sv;  found = true;  }  }  if (!found) {  cout << "Khong tim thay sinh vien nao.\n";  }  }  // Xóa sinh viên theo mã hoặc tên  void deleteStudent(const string &query) {  auto it = remove\_if(svs.begin(), svs.end(), [&](const SV &sv) {  return sv.getMsv() == query || sv.getTen() == query;  });  if (it != svs.end()) {  svs.erase(it, svs.end());  cout << "Da xoa sinh vien co ma hoac ten: " << query << endl;  } else {  cout << "Khong tim thay sinh vien de xoa.\n";  }  }  }; |

Code đầy đủ có thể xem đầy đủ ở Github.

Link: https://github.com/B1CTommy/Takatori-demo/tree/main/btlctdl%26gt

## Cài đặt cấu trúc dữ liệu Stack

Với cấu trúc dữ liệu Stack, chúng ta có 2 cách cài đặt chính :

* Dùng mảng.

#### Đặc điểm:

**Dễ triển khai**: Việc dùng mảng cho phép thao tác nhanh chóng vì các phần tử được lưu liên tiếp nhau trong bộ nhớ.

**Kích thước cố định**: Stack triển khai bằng mảng cần xác định kích thước ban đầu. Điều này có nghĩa là nếu Stack vượt quá kích thước này, chúng ta có thể gặp lỗi tràn (**stack overflow**).

**Thời gian truy cập nhanh**: Do mảng lưu trữ các phần tử liên tiếp nhau, việc truy cập phần tử trên đỉnh Stack (top) rất nhanh.

|  |
| --- |
| //Tm  #include<bits/stdc++.h>  #define endl "\n"  using namespace std;  #ifndef \_\_STACK\_\_  #define \_\_STACK\_\_  template<typename T>  class Stack{  private:  int Size;  int Space;  T\* elem;  public:  Stack() {  Size = Space = 0;  elem = NULL;  }  ~Stack() {  if (elem != NULL) delete[] elem;  }  Stack<T>& operator=(Stack<T>& s) {  if (this == &s) return \*this;  this->Size = s.Size;  this->Space = s.Space;  if (elem != NULL) delete[] elem;  elem = new T[Space];  for(int i = 0; i < Size; i++){  elem[i] = s.elem[i];  }  return \*this;  }  Stack(Stack<T>& s) {  \*this = s;  }  int size() {  return Size;  }  bool empty() {  return Size == 0;  }  T& top() {  return elem[Size - 1];  }  void pop() {  if (empty()) return;  Size--;  }  void clear() {  Size = 0;  }    void push(T val) {  if (Size == Space) {  Space = Size == 0 ? 1 : Size \* 2;  T\* temp = new T[Space];  for (int i = 0; i < Size; i++) {  temp[i] = elem[i];  }  if (elem != NULL) delete[] elem;  elem = temp;  }  elem[Size] = val;  ++Size;  }  };  #endif |

* Dùng danh sách liên kết.

#### Đặc điểm:

**Không giới hạn kích thước**: Không giống như mảng, Stack dùng danh sách liên kết không cần xác định kích thước ban đầu. Điều này cho phép Stack mở rộng hoặc thu hẹp linh hoạt tuỳ theo yêu cầu.

**Chiếm nhiều bộ nhớ hơn**: So với mảng, danh sách liên kết chiếm nhiều bộ nhớ hơn do cần lưu trữ thêm con trỏ để liên kết các phần tử.

**Truy cập không liên tục trong bộ nhớ**: Các phần tử của Stack được lưu ở các vị trí bộ nhớ không liền kề nhau.

|  |
| --- |
| //Tm  #include<bits/stdc++.h>  using namespace std;  #ifndef \_\_STACKLL\_\_  #define \_\_STACKLL\_\_  template<typename T>  class Node {  public:  T data;  Node\* next;  Node(T val) : data(val), next(nullptr) {}  };  template<typename T>  class StackLinkedList {  private:  Node<T>\* top;  int Size;  public:  StackLinkedList() {  top = nullptr;  Size = 0;  }  ~StackLinkedList() {  clear();  }  StackLinkedList(const StackLinkedList<T>& s) {  top = nullptr;  Size = 0;  Node<T>\* current = s.top;  while (current != nullptr) {  push(current->data);  current = current->next;  }  }  StackLinkedList<T>& operator=(const StackLinkedList<T>& s) {  if (this == &s) return \*this;  clear();  top = nullptr;  Size = 0;  Node<T>\* current = s.top;  while (current != nullptr) {  push(current->data);  current = current->next;  }  return \*this;  }  int size() const {  return Size;  }  bool empty() const {  return Size == 0;  }  T& getTop() {  if (empty()) {  throw runtime\_error("Stack is empty. Cannot access top element.");  }  return top->data;  }  void pop() {  if (empty()) return;  Node<T>\* temp = top;  top = top->next;  delete temp;  Size--;  }  void clear() {  while (!empty()) {  pop();  }  }  void push(T val) {  Node<T>\* newNode = new Node<T>(val);  newNode->next = top;  top = newNode;  Size++;  }  };  #endif |

Mô phỏng hoạt động của stack cài đặt bằng mảng:

|  |
| --- |
| //Tm  #include<bits/stdc++.h>  using namespace std;  #include "D:\Codeing\btlctdl&gt\BTL-pB\StackArray\stack.cpp"  int main()  {  int T;  cin >> T;  Stack<long long> stack;  while (T--) {  int query;  cin >> query;  if (query == 1) {  long long n;  cin >> n;  stack.push(n);  }  else if (query == 2) {  stack.pop();  }  else if (query == 3) {  if (stack.empty()) {  cout << "Empty!" << endl;  } else {  cout << stack.top() << endl;  }  }  }  return 0;  } |

Mô phỏng hoạt động của stack cài đặt bằng danh sách liên kết

|  |
| --- |
| //Tm  #include<bits/stdc++.h>  using namespace std;  #include "D:\Codeing\btlctdl&gt\BTL-pB\StackLinkedList\stacklinkedlist.cpp"  int main() {  int T;  cin >> T;  StackLinkedList<long long> stack;  while (T--) {  int query;  cin >> query;  if (query == 1) {  long long n;  cin >> n;  stack.push(n);  }  else if (query == 2) {  stack.pop();  }  else if (query == 3) {  if (stack.empty()) {  cout << "Empty!" << endl;  } else {  cout << stack.getTop() << endl;  }  }  }  return 0;  } |

Code đầy đủ có thể xem ở Github.

Link: https://github.com/B1CTommy/Takatori-demo/tree/main/btlctdl%26gt

# KẾT QUẢ THỰC NGHIỆM

## Dữ liệu

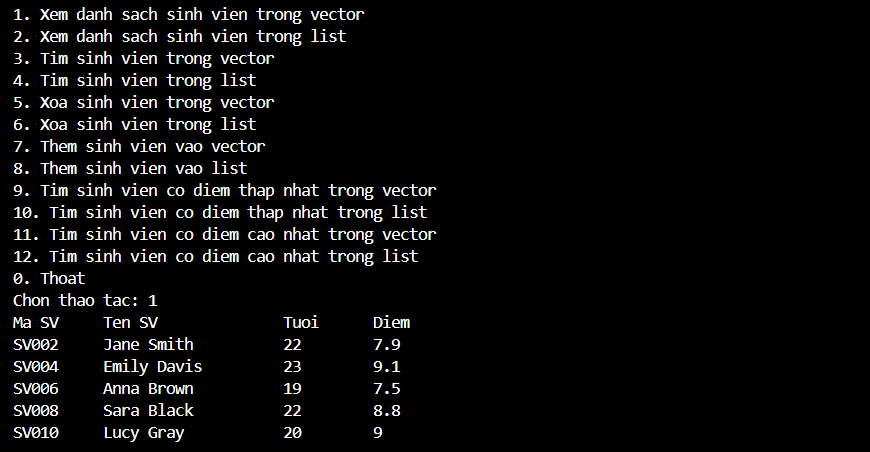
## Bộ dữ liệu thông tin sinh viên được lưu trữ trong file input.txt. Chương trình sẽ đọc dữ liệu từ file input.txt và ghi kết quả ra màn hình. Dữ liệu sẽ bao gồm các sinh viên với mã số sinh viên, tên, tuổi, và điểm số. Mã số sinh viên sẽ quyết định nơi lưu trữ: sinh viên có mã kết thúc bằng chữ số chẵn sẽ được lưu trong VectorSV, còn sinh viên có mã kết thúc bằng chữ số lẻ sẽ được lưu trong ListSV.Các kết quả:

* Các chức năng trong chương trình:
* Bài toán A:
* Đọc bộ dữ liệu sinh viên từ file input.txt, lưu thông tin sinh viên vào VectorSV hoặc ListSV tùy thuộc vào mã số sinh viên.
* Hiển thị danh sách sinh viên trong VectorSV và ListSV.
* Thực hiện các thao tác quản lý sinh viên, bao gồm:
  + Tìm kiếm sinh viên dựa trên mã hoặc tên.
  + Xóa sinh viên khỏi danh sách.
  + Thêm sinh viên mới vào danh sách.
  + Tìm sinh viên có điểm cao nhất hoặc thấp nhất.
* Bài toán B:
* Xây dựng Stack.
* Mô phỏng hoạt động và Ứng dụng Stack.

## Các kết quả:

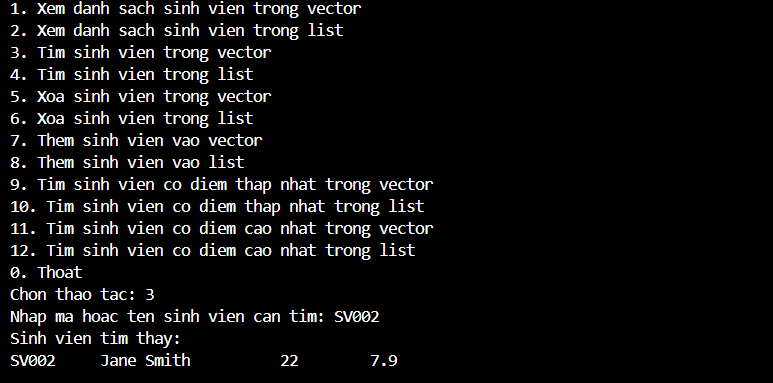
* **Kết quả 1:**

Tính năng 1: Đọc file input, ghi kết quả ra màn hình

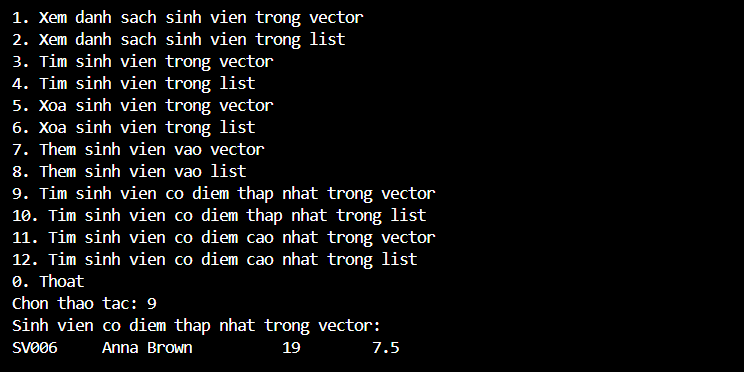


Hình 4 Chương trình đọc và in kết quả ra màn hình

Tính năng 2: Thực hiện vài chức năng quản lý:



Hình 5 Tìm sinh viên theo mã sinh viên hoặc tên và in kết quả ra màn hình



Hình 6 Tìm sinh viên có điểm thấp nhất và in kết quả ra màn hình

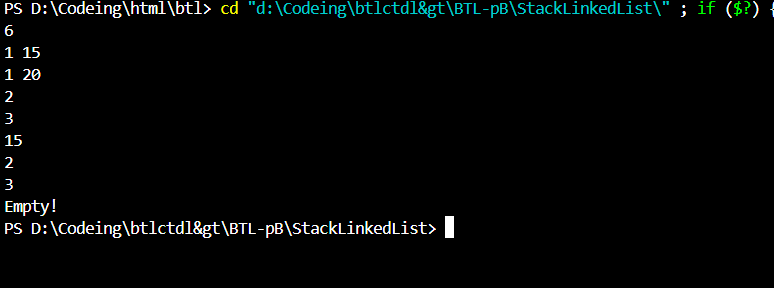
* **Kết quả 2:**

Tính năng 1.1: Mô phỏng cách hoạt động của stack (cài đặt bằng mảng)

|  |
| --- |
| //Tm  #include<bits/stdc++.h>  using namespace std;  #include "D:\Codeing\btlctdl&gt\BTL-pB\StackArray\stack.cpp"  int main()  {  int T;  cin >> T;  Stack<long long> stack;  while (T--) {  int query;  cin >> query;  if (query == 1) {  long long n;  cin >> n;  stack.push(n);  }  else if (query == 2) {  stack.pop();  }  else if (query == 3) {  if (stack.empty()) {  cout << "Empty!" << endl;  } else {  cout << stack.top() << endl;  }  }  }  return 0;  } |

Tính năng 1.2: Mô phỏng cách hoạt động của stack (cài đặt bằng danh sách liên kết)

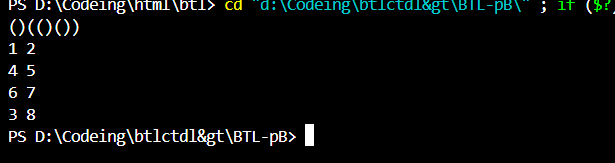
|  |
| --- |
| //Tm  #include<bits/stdc++.h>  using namespace std;  #include "D:\Codeing\btlctdl&gt\BTL-pB\StackLinkedList\stacklinkedlist.cpp"  int main() {  int T;  cin >> T;  StackLinkedList<long long> stack;  while (T--) {  int query;  cin >> query;  if (query == 1) {  long long n;  cin >> n;  stack.push(n);  }  else if (query == 2) {  stack.pop();  }  else if (query == 3) {  if (stack.empty()) {  cout << "Empty!" << endl;  } else {  cout << stack.getTop() << endl;  }  }  }  return 0;  } |



Hình 7 In ra màn hình kết quả mô phỏng

Tính năng 2: Ứng dụng stack giải bài toán

|  |
| --- |
| //Tm  #include<bits/stdc++.h>  using namespace std;  #include "D:\Codeing\btlctdl&gt\BTL-pB\StackArray\stack.cpp"  int main() {  string s;  cin >> s;  Stack<int> stack;  for (int i = 0; i < s.size(); i++) {  if (s[i] == '(') {  stack.push(i + 1);  } else if (s[i] == ')') {  if (!stack.empty()) {  int openIndex = stack.top();  stack.pop();  cout << openIndex << " " << i + 1 << endl;  }  }  }  return 0;  } |



Hình 8 In ra màn hình kết quả bài toán

# KẾT LUẬN

## Đánh giá mức độ hoàn thành

* Hoàn thành đúng với mục tiêu ban đầu đề ra.
* Đã ứng dụng được kiến thức của môn học, kết hợp với một số kiến thức tự tìm hiểu để làm sản phẩm.
* Cần nghiên cứu thêm kiến thức để hình thức sản phẩm trở thành một web hay app đặc thù.

## Bài học rút ra

* Môn học Cấu trúc dữ liệu và giải thuật là vô cùng quan trọng, là phần kiến thức không thể thiếu đối với sinh viên khoa Công nghệ thông tin.
* Cần sử dụng kiến thức đã học để áp dụng vào một việc nào đó để giúp nắm vững kiến thức và tìm ra những điều mới mẻ.
* Sau khi hoàn thành sản phẩm, cần nghĩ thêm để phát triển sản phẩm, khắc phục những hạn chế của sản phẩm.

## Khó khăn với môn học

* Để học tốt môn Cấu trúc dữ liệu và giải thuật, cần có sự chuẩn bị tốt từ các môn học nền tảng như Kỹ thuật lập trình và Lập trình hướng đối tượng, vì đây là cơ sở giúp hiểu và áp dụng các thuật toán một cách hiệu quả.
* Lượng kiến thức trong môn học này rất lớn, cần thời gian để nắm vững và áp dụng vào các bài toán thực tế.
* Việc hiểu rõ bản chất của các cấu trúc dữ liệu là yếu tốc quan trọng giúp giải quyết các vấn đề phức tạp trong lập trình.

# LỜI CẢM ƠN

Em xin chân thành cảm ơn thầy đã tận tình hướng dẫn và truyền đạt kiến thức trong suốt môn học Cấu trúc dữ liệu và Giải thuật, một môn học vô cùng quan trọng đối với sinh viên ngành Công nghệ thông tin. Những ví dụ minh họa cụ thể và sự chỉ dạy của thầy đã giúp em hiểu sâu hơn về lý thuyết cũng như cách áp dụng vào thực tế.

Em cũng xin cảm ơn thầy đã giao cho chúng em một bài tập lớn, giúp em có cơ hội thử thách bản thân không chỉ trong việc viết code mà còn trong tinh thần làm việc và tìm tòi những kiến thức mới mẻ. Qua project này, em đã có dịp phát triển thêm nhiều kỹ năng quan trọng.

Sản phẩm của em vẫn còn nhiều thiếu sót do thiếu kinh nghiệm và kỹ năng thực tiễn. Em mong nhận được những nhận xét và góp ý từ thầy để có thể hoàn thiện hơn trong tương lai.

Em xin chân thành cảm ơn!