**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**KHOA CÔNG NGHỆ THÔNG TIN**

----o0o----

Ảnh có chứa biểu tượng, Nhãn hiệu, Phông chữ, vòng tròn

Mô tả được tạo tự động

**BÁO CÁO MÔN HỌC**

**CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

Đề tài:

**Ứng dụng Vector vào bài toán quản lý**

**Xây dựng lớp cây nhị phân để quản lý dữ liệu**

|  |  |
| --- | --- |
| Giảng viên hướng dẫn: | **TS. Hoàng Văn Thông** |
| Sinh viên thực hiện: | **Trần Tiến Sơn** |
| Mã sinh viên: | **231210892** |
| Khoa: | **Công nghệ thông tin – Khóa 64** |

**MỤC LỤC**

[LỜI MỞ ĐẦU 4](#_Toc182294733)

[I. GIỚI THIỆU 4](#_Toc182294734)

[1. Lý do và động lực 4](#_Toc182294735)

[2. Định nghĩa bài toán 4](#_Toc182294736)

[a. Bài toán 1 (Phần A: Bài toán quản lí): 4](#_Toc182294737)

[b. Bài toán 2 (Phần B: Bài tập cá nhân) 4](#_Toc182294738)

[3. Nhiệm vụ và các mức độ hoàn thành 5](#_Toc182294739)

[II. PHƯƠNG PHÁP LỰA CHỌN 6](#_Toc182294740)

[1. Cấu trúc dữ liệu 6](#_Toc182294741)

[a. Vector 6](#_Toc182294742)

[III. TRIỂN KHAI CÀI ĐẶT 9](#_Toc182294743)

[1. Ngôn ngữ lập trình và thư viện 9](#_Toc182294744)

[a. Ngôn ngữ lập trình C++ 9](#_Toc182294745)

[b. Thư viện trong C++ 10](#_Toc182294746)

[c. Tổ chức chương trình và đóng gói 11](#_Toc182294747)

[1. Tổ chức chương trình: 11](#_Toc182294748)

[2. Đóng gói: 11](#_Toc182294749)

[II. PHÂN TÍCH CHƯƠNG TRÌNH – BÀI TOÁN 1 12](#_Toc182294750)

[1. Khai báo các thư viện sử dụng trong chương trình 12](#_Toc182294751)

[2. Xây dựng lớp “Car” gồm các thuộc tính và phương thức 12](#_Toc182294752)

[a. Các thuộc tính 13](#_Toc182294753)

[b. Hàm khởi tạo (Constructor) 13](#_Toc182294754)

[c. Hàm nhập liệu (Input Stream Operator Overloading) 14](#_Toc182294755)

[d. Hàm xuất dữ liệu (Output Stream Operator Overloading) 14](#_Toc182294756)

[e. Các phương thức truy cập (Getter Methods) 15](#_Toc182294757)

[3. Xây dựng lớp “CarList” gồm các hàm chức năng 15](#_Toc182294758)

[a. Khai báo lớp CarList và các thành viên 16](#_Toc182294759)

[b. Hàm expand – mở rộng dung lượng của mảng động 17](#_Toc182294760)

[c. Hàm khởi tạo (Constructor) và hàm hủy (Destructor) 17](#_Toc182294761)

[d. Hàm addCar – thêm xe vào danh sách 18](#_Toc182294762)

[e. Hàm removeCar – xóa xe khỏi danh sách theo số VIN 18](#_Toc182294763)

[f. Hàm displayAll – Hiển thị tất cả các xe trong danh sách 18](#_Toc182294764)

[g. Hàm sortByPrice – Sắp xếp danh sách xe theo giá 19](#_Toc182294765)

[h. Hàm sortByYear – Sắp xếp xe theo năm sản xuất 20](#_Toc182294766)

[4. Xây dựng lớp “app” để quản lý chương trình, gồm có menu và thực hiện các thao tác ở lớp danh sách 20](#_Toc182294767)

[a. Định nghĩa lớp Menu 20](#_Toc182294768)

[b. Lặp do-while – Vòng lặp điều khiển menu 22](#_Toc182294769)

[c. Hiển thị các lựa chọn và nhập lựa chọn từ người dùng 22](#_Toc182294770)

[d. Switch – xử lý các lựa chọn 22](#_Toc182294771)

[5. Xây dựng hàm main để chạy chương trình 25](#_Toc182294772)

[III. PHÂN TÍCH CHƯƠNG TRÌNH – BÀI TOÁN 2 25](#_Toc182294773)

[6. Khai báo các thư viện sử dụng trong chương trình 25](#_Toc182294774)

[a. Cấu trúc dữ liệu và lớp cơ bản 25](#_Toc182294775)

[a. Cấu trúc Node 25](#_Toc182294776)

[b. Lớp Binary Search Tree (BST) 26](#_Toc182294777)

[c. Lớp Dictionary 31](#_Toc182294778)

[IV. KẾT QUẢ THỰC NGHIỆM 35](#_Toc182294779)

[V. KẾT LUẬN 35](#_Toc182294780)

[VI. LỜI CẢM ƠN 35](#_Toc182294781)

[VII. TÀI LIỆU THAM KHẢO 35](#_Toc182294782)

# LỜI MỞ ĐẦU

# GIỚI THIỆU

## Lý do và động lực

Để đáp ứng yêu cầu môn học cũng như mong muốn tìm hiểu, mở rộng kiến thức 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:

* Hữu ích.
* Áp dụng được kiến thức đã học.
* Học hỏi được kiến thức mới.

## Định nghĩa bài toán

### Bài toán 1 (Phần A: Bài toán quản lí):

Với phần này, em lựa chọn bài toán quản lý một danh sách gồm ô tô.

Input: Các trường dữ liệu về ô tô, gồm có số VIN (Vehical Identification Number), tên thương hiệu, dòng xe, năm sản xuất và giá xe

Chương trình sẽ đọc thông tin từ file văn bản và lưu vào dưới dạng Vector, cho phép người dùng lựa chọn các tùy chọn: sắp xếp danh sách các xe dựa vào giá, sắp xếp danh sách xe dựa vào năm sản xuất, hiển thị xe có giá cao nhất hoặc thấp nhất.

Output: Hiển thị danh sách các ô tô đã được sắp xếp theo yêu cầu của người dùng

### Bài toán 2 (Phần B: Bài tập cá nhân)

1. Xây dựng lớp cây tìm kiếm nhị phân

2. Xây dựng một lớp quản lý một từ điển Anh-Việt, từ điển được lưu vào cây nhị phân mà

tại mỗi nút của nó lưu một từ tiếng Anh và nghĩa tiếng Việt tương ứng có các phương thức:

a. Đọc từ điển từ file để xây dựng cây

b. Ghi từ điển trong cây vào file (duyệt theo thứ tự giữa, duyệt đến nút nào thì ghi vào file

giá trị của nút đó)

c. Thêm một từ mới vào từ điển đang lưu trong cây

d. Xóa bỏ một từ của từ điển đang lưu trong cây

e. Cập nhật lại nghĩa một từ đang lưu trong cây

f. Tìm kiếm một từ trong cây.

2. Viết chương trình kiểm tra sự hoạt động của lớp từ điển đã xây dựng

## Nhiệm vụ và các mức độ hoàn thành

|  |  |
| --- | --- |
| Nhiệm vụ | Mức độ hoàn thành |
| * Tìm hiểu lớp vector * Tìm hiểu lớp list * Lên ý tưởng bài toán 1 * Lên ý tưởng cho bài toán quản lí sinh viên. * Triển khai với cấu trúc danh * sách (List) để quản lí sinh viên. * Kiểm thử và hoàn thiện * chương trình. | Đã hoàn thành |
| * Tìm hiểu cấu trúc danh sách liên kết (Linked List) * Áp dụng vào bài xây dựng đa thức * Lên ý tưởng bài toán 2 | Đã hoàn thành |
| * Làm báo cáo * Lưu trữ code bằng Github | Đã hoàn thành |

# PHƯƠNG PHÁP LỰA CHỌN

## Cấu trúc dữ liệu

### Vector

Trong C++, vector là một cấu trúc dữ liệu thuộc thư viện chuẩn (STL - Standard Template Library), cung cấp một **mảng động** có khả năng tự động thay đổi kích thước khi thêm hoặc xóa phần tử. Vector cho phép truy cập ngẫu nhiên tới các phần tử và hỗ trợ các thao tác thêm, xóa nhanh chóng ở cuối mảng. Đây là một công cụ hữu ích cho việc quản lý và thao tác trên các tập dữ liệu có kích thước không cố định.

Đặc điểm chính của Vector:

* **Mảng động**: Khác với mảng tĩnh, vector có thể thay đổi kích thước một cách linh hoạt khi cần.
* **Truy cập ngẫu nhiên**: Vector cho phép truy cập các phần tử nhanh chóng thông qua chỉ số, tương tự như mảng.
* **Tối ưu hiệu năng khi thêm/xóa phần tử ở cuối**: Việc thêm phần tử vào cuối vector có độ phức tạp trung bình là O(1).
* **Quản lý bộ nhớ tự động**: Khi cần mở rộng dung lượng, vector sẽ tự động cấp phát lại bộ nhớ để chứa thêm dữ liệu.

Một số thao tác cơ bản với Vector:

* Khởi tạo: std::vector<int> v;
* Thêm phần tử: v.push\_back(10);
* Truy cập phần tử: v[i]; hoặc v.at(i);
* Lấy kích thước: v.size();
* Xóa phần tử cuối: v.pop\_back();

Ưu điểm và nhược điểm:

* Ưu điểm: Linh hoạt, dễ sử dụng, tự động quản lý kích thước, tích hợp tốt với các thuật toán trong thư viện STL.
* Nhược điểm: Có thể gây lãng phí bộ nhớ nếu vector mở rộng vượt quá dung lượng cần thiết, thao tác chèn hoặc xóa phần tử ở giữa có thể chậm.

Các thao tác trên Vector

#### Modifers

|  |  |
| --- | --- |
| push\_back(const T& value) | * Chức năng: Thêm một phần tử value vào cuối vector. * Độ phức tạp: Trung bình là O(1); khi cần cấp phát thêm bộ nhớ, độ phức tạp có thể tăng lên O(n). |
| pop\_back() | * Chức năng: Xóa phần tử cuối cùng của vector. * Độ phức tạp: O(1). |
| insert(iterator pos, const T& value) | * Chức năng: Thêm một phần tử value vào vị trí pos. * Độ phức tạp: O(n) vì có thể phải dịch chuyển các phần tử sau pos. |
| erase(iterator pos) | * Chức năng: Xóa một phần tử tại vị trí pos. * Độ phức tạp: O(n) vì có thể phải dịch chuyển các phần tử sau pos. |
| erase(iterator first, iterator last) | * Chức năng: Xóa tất cả các phần tử từ vị trí first đến last. * Độ phức tạp: O(n). |
| clear() | * Chức năng: Xóa toàn bộ các phần tử trong vector, khiến vector trở nên rỗng. * Độ phức tạp: O(n). |
| resize(size\_type count) | * Chức năng: Thay đổi kích thước của vector thành count phần tử. Nếu count lớn hơn kích thước hiện tại, vector sẽ thêm các phần tử có giá trị mặc định. * Độ phức tạp: O(n). |
| swap(vector& other) | * Chức năng: Hoán đổi nội dung của vector hiện tại với other. Không thay đổi kích thước tổng bộ nhớ đã cấp phát. * Độ phức tạp: O(1). |

#### Capacity

|  |  |  |  |
| --- | --- | --- | --- |
| capacity() | | * Mô tả: Trả về dung lượng hiện tại của vector (số phần tử tối đa có thể lưu trữ mà không cần cấp phát thêm bộ nhớ). * Ứng dụng: Giúp kiểm tra dung lượng của vector, từ đó biết được khi nào vector sẽ cần cấp phát thêm bộ nhớ khi thêm phần tử. | |
| reserve(size\_type n) | | * Mô tả: Đặt trước dung lượng của vector ít nhất là n phần tử. Nếu n lớn hơn dung lượng hiện tại, vector sẽ cấp phát thêm bộ nhớ; nếu không, vector sẽ giữ nguyên dung lượng. * Ứng dụng: Hữu ích khi biết trước số lượng phần tử cần thêm vào vector, giúp tránh việc cấp phát lại nhiều lần và tăng hiệu suất cho các thao tác thêm phần tử. | |
| shrink\_to\_fit() | | * Mô tả: Yêu cầu giảm dung lượng của vector sao cho bằng kích thước thực tế (số phần tử hiện tại). Thao tác này có thể giúp giải phóng bộ nhớ dư thừa, nhưng không bắt buộc trình biên dịch phải thực hiện. * Ứng dụng: Sử dụng khi cần tối ưu bộ nhớ của vector, đặc biệt trong trường hợp dung lượng quá lớn so với kích thước sau các thao tác xóa phần tử. | |
| size() | | * Mô tả: Trả về số lượng phần tử thực tế hiện có trong vector. * Ứng dụng: So sánh với capacity để biết vector đã gần tới giới hạn dung lượng chưa, qua đó có thể sử dụng reserve() để tối ưu hiệu năng nếu cần thiết. | |

# TRIỂN KHAI CÀI ĐẶT

## Ngôn ngữ lập trình và thư viện

### Ngôn ngữ lập trình C++

C++ là ngôn ngữ lập trình mạnh mẽ, kế thừa từ C và hỗ trợ lập trình hướng đối tượng. Các đặc điểm nổi bật của C++ bao gồm:

* **Hướng đối tượng (OOP)**: C++ hỗ trợ các tính năng như kế thừa, đa hình, và đóng gói, giúp tổ chức mã nguồn hiệu quả.
* **Quản lý bộ nhớ trực tiếp**: C++ cho phép lập trình viên kiểm soát bộ nhớ thông qua con trỏ và cấp phát động.
* **Hiệu suất cao**: C++ biên dịch trực tiếp thành mã máy, mang lại hiệu suất tối ưu cho các ứng dụng yêu cầu tính toán phức tạp.
* **Tính linh hoạt**: C++ hỗ trợ cả lập trình thủ tục và lập trình hướng đối tượng.
* **Tính tương thích ngược**: C++ hoàn toàn tương thích với C, cho phép tái sử dụng mã nguồn C.
* **Ứng dụng rộng rãi**: C++ được sử dụng trong phát triển phần mềm hệ thống, game, ứng dụng khoa học và kỹ thuật.

### Thư viện trong C++

#### <iostream>:

* Cung cấp các chức năng nhập và xuất cơ bản (input/output).
* Các đối tượng nổi bật: std::cin (đọc dữ liệu), std::cout (xuất dữ liệu), std::cerr (xuất lỗi).

#### <vector>:

* Cung cấp cấu trúc dữ liệu mảng động (vector) với khả năng tự động thay đổi kích thước.
* Hỗ trợ các thao tác như thêm, xóa, truy xuất phần tử dễ dàng.

#### <string>:

* Cung cấp lớp std::string để làm việc với chuỗi ký tự.
* Hỗ trợ các thao tác chuỗi như nối, cắt, tìm kiếm, thay thế ký tự.

#### <algorithm>:

* Cung cấp các thuật toán cơ bản như sắp xếp, tìm kiếm, sao chép, và biến đổi dữ liệu.
* Các hàm nổi bật: std::sort(), std::find(), std::reverse().

#### <cmath>:

* Cung cấp các hàm toán học chuẩn như sin(), cos(), sqrt(), pow(), v.v.
* Hỗ trợ các phép toán với số thực.

#### <cstdlib>:

* Cung cấp các hàm chuẩn như malloc(), free(), rand(), exit(), v.v.
* Dùng cho các tác vụ quản lý bộ nhớ và các chức năng hỗ trợ hệ thống.

#### <ctime>:

* Cung cấp các hàm để làm việc với thời gian và ngày tháng, như time(), difftime(), clock().

#### <fstream>:

* Cung cấp các chức năng đọc và ghi dữ liệu từ/đến tệp tin.
* Các đối tượng nổi bật: std::ifstream (đọc từ tệp), std::ofstream (ghi vào tệp).

#### <map> và <unordered\_map>:

* Cung cấp cấu trúc dữ liệu bảng băm (map) và bảng băm không theo thứ tự (unordered\_map) để lưu trữ các cặp khóa-giá trị.
* Hỗ trợ tìm kiếm nhanh và thao tác trên dữ liệu theo cặp.

#### <queue> và <stack>:

* Cung cấp các cấu trúc dữ liệu hàng đợi (queue) và ngăn xếp (stack), hỗ trợ các thao tác như thêm, xóa phần tử theo nguyên lý FIFO (queue) và LIFO (stack).

#### <set> và <unordered\_set>:

* Cung cấp cấu trúc dữ liệu tập hợp (set) và tập hợp không theo thứ tự (unordered\_set) để lưu trữ các phần tử không trùng lặp.

## Tổ chức chương trình và đóng gói

### Tổ chức chương trình:

* Chương trình được viết bằng trình soạn thảo Visual Studio Code hoặc DevC/DevC++.
* Các đoạn mã code được lưu trữ trên Github

### Đóng gói:

* Các đoạn mã code và tệp tin liên quan được lưu trữ tại repository: [Data-Algo\_UTC](https://github.com/Maloque18705/Data-Algo_UTC.git).
* Báo cáo bài tập lớn.

# PHÂN TÍCH CHƯƠNG TRÌNH – BÀI TOÁN 1

## 1. Khai báo các thư viện sử dụng trong chương trình

|  |
| --- |
| #include <iostream>  #include <vector>  #include <string>  **using** **namespace** std**;** |

## Xây dựng lớp “Car” gồm các thuộc tính và phương thức

|  |
| --- |
| class Car  **{**  private**:**  string brand**;** // Ten thuong hieu  string model**;** // Ten mau xe  string vin**;** // So VIN cua xe  int year**;** // Nam san xuat  float price**;** // Gia xe  public**:**  Car**()** **:** brand**(**""**),** model**(**""**),** vin**(**""**),** year**(**0**),** price**(**0.0**)** **{}**  friend istream**&** **operator>>(**istream**&** in**,** Car**&** c**){**  cout **<<** "Thuong hieu: "**;**  in**.**ignore**();** // Loai bo ky tu xuong dong  getline**(**in**,** c**.**brand**);** // Cho phep dau cach trong xau  cout **<<** "Dong xe: "**;**  in**.**ignore**();** // Loai bo ky tu xuong dong  getline**(**in**,** c**.**model**);** // Cho phep dau cach trong xau  cout **<<** "So VIN: "**;** in**>>** c**.**vin**;**  cout **<<** "Nam san xuat: "**;** in**>>** c**.**year**;**  cout **<<** "Gia: "**;** in**>>** c**.**price**;**  **return** in**;**  **}**  friend ostream**&** **operator<<(**ostream**&** out**,** const Car**&** c**){**  out **<<** "Thuong hieu: " **<<** c**.**brand **<<** endl**;**  out **<<** "Dong xe: " **<<** c**.**brand **<<** endl**;**  out **<<** "So VIN: " **<<** c**.**vin **<<** endl**;**  out **<<** "Nam san xuat: " **<<** c**.**year **<<** endl**;**  out **<<** "Gia: " **<<** c**.**price **<<** endl**;**  **return** out**;**  **}**  string getBrand**()** const **{return** brand**;** **}**  string getModel**()** const **{return** model**;** **}**  string getVIN**()** const **{return** vin**;** **}**  int getYear**()** const **{return** year**;** **}**  float getPrice**()** const **{return** price**;** **}**    **};** |

### Các thuộc tính

|  |
| --- |
| private:          string brand;           // Ten thuong hieu          string model;           // Ten mau xe          string vin;             // So VIN cua xe          int year;               // Nam san xuat          float price;            // Gia xe |

* **brand**: Một chuỗi (string) chứa tên thương hiệu xe (ví dụ: Toyota, Honda).
* **model**: Một chuỗi (string) chứa tên mẫu xe (ví dụ: Corolla, Civic).
* **vin**: Một chuỗi (string) chứa số VIN, đây là mã số định danh duy nhất của xe.
* **year**: Một số nguyên (int) chứa năm sản xuất của chiếc xe.
* **price**: Một số thực (float) chứa giá trị của chiếc xe.

### Hàm khởi tạo (Constructor)

|  |
| --- |
| public:          Car() : brand(""), model(""), vin(""), year(0), price(0.0) {} |

Hàm khởi tạo mặc định của lớp Car. Nó khởi tạo tất cả các thuộc tính của đối tượng Car với các giá trị mặc định:

* **brand** là chuỗi rỗng (""),
* **model** là chuỗi rỗng (""),
* **vin** là chuỗi rỗng (""),
* **year** là 0 (năm không xác định),
* **price** là 0.0 (giá không xác định)

### Hàm nhập liệu (Input Stream Operator Overloading)

|  |
| --- |
| friend istream& operator>>(istream& in, Car& c){              cout << "Thuong hieu: ";              in.ignore();            // Loai bo ky tu xuong dong              getline(in, c.brand);   // Cho phep dau cach trong xau              cout << "Dong xe: ";              in.ignore();            // Loai bo ky tu xuong dong              getline(in, c.model);   // Cho phep dau cach trong xau              cout << "So VIN: "; in>> c.vin;              cout << "Nam san xuat: "; in>> c.year;              cout << "Gia: "; in>> c.price;              return in;          } |

* Định nghĩa lại toán tử >> để nhập thông tin cho đối tượng Car từ luồng nhập (istream).
* in.ignore() được dùng để loại bỏ ký tự xuống dòng (\n) còn lại trong bộ đệm khi người dùng nhập vào trước đó.
* getline(in, c.brand) và getline(in, c.model) cho phép nhập chuỗi có dấu cách (như tên thương hiệu hoặc mẫu xe có nhiều từ).
* in >> c.vin, in >> c.year, in >> c.price dùng để nhập các giá trị vin, year, và price.

### Hàm xuất dữ liệu (Output Stream Operator Overloading)

|  |
| --- |
| friend ostream& operator<<(ostream& out, const Car& c){              out << "Thuong hieu: " << c.brand << endl;              out << "Dong xe: " << c.model << endl;              out << "So VIN: " <<  c.vin << endl;              out << "Nam san xuat: " << c.year << endl;              out << "Gia: " << c.price << endl;              return out;          } |

Định nghĩa lại toán tử << để xuất thông tin của đối tượng Car ra màn hình hoặc bất kỳ luồng xuất nào (ostream).

Mỗi dòng xuất ra tương ứng với một thuộc tính của đối tượng:

* **c.brand** là tên thương hiệu.
* **c.model** là tên mẫu xe.
* **c.vin** là số VIN.
* **c.year** là năm sản xuất.
* **c.price** là giá xe.

### Các phương thức truy cập (Getter Methods)

|  |
| --- |
| string getBrand() const {return brand; }          string getModel() const {return model; }          string getVIN() const {return vin; }          int getYear() const {return year; }          float getPrice() const {return price; } |

Đây là các phương thức getter, được sử dụng để lấy giá trị của các thuộc tính riêng tư.

Các phương thức này không thay đổi trạng thái của đối tượng, vì vậy chúng được khai báo là const (hàm hằng – constant methods).

## Xây dựng lớp “CarList” gồm các hàm chức năng

|  |
| --- |
| class CarList  **{**  private**:**  Car**\*** cars**;**  int size**;**  int capacity**;**  void expand**(){**  capacity **\*=**2**;**  Car**\*** newArray **=** **new** Car**[**capacity**];**  **for** **(**int i **=** 0**;** i **<** size**;** **++**i**){**  newArray**[**i**]** **=** cars**[**i**];**  **}**  **delete[]** cars**;**  cars **=** newArray**;**  **}**  public**:**  CarList**(**int cap **=** 10**):** size**(**0**),** capacity**(**cap**){**  cars **=** **new** Car**[**capacity**];**  **}**  **~**CarList**()** **{delete[]** cars**;}**  void addCar**(**const Car**&** c**){**  **if** **(**size **==** capacity**)** expand**();**  cars**[**size**++]** **=** c**;**  **}**  void removeCar**(**const string**&** vin**){**  **for** **(**int i **=** 0**;** i**<**size**;** i**++){**  **if(**cars**[**i**].**getVIN**()** **==** vin**){**  **for** **(**int j **=** i**;** j **<** size**-**1**;** **++**j**){**  cars**[**j**]** **=** cars**[**j**+**1**];**  **}**  **--**size**;**  **break;**  **}**  **}**  **}**    void displayAll**()** const**{**  **for** **(**int i **=** 0**;** i **<** size**;** i**++){**  cout **<<** cars**[**i**]** **<<** endl**;**  **}**  **}**  void sortByPrice**(){**  **for** **(**int i **=** 0**;** i **<** size**-**1**;** **++**i**){**  **for** **(**int j **=** 0**;** j **<** size **-** i **-** 1**;** **++**j**){**  **if(**cars**[**j**].**getPrice**()** **>** cars**[**j**+**1**].**getPrice**()){**  swap**(**cars**[**j**],** cars**[**j**+**i**]);**  **}**  **}**  **}**  **}**    void sortByYear**(){**  **for** **(**int i **=** 0**;** i **<** size **-** 1**;** **++**i**){**  **for** **(**int j **=** 0**;** j **<** size **-** i **-** 1**;** **++**j**){**  **if(**cars**[**j**].**getYear**()** **>** cars**[**j**+**1**].**getYear**()){**  swap**(**cars**[**j**],** cars**[**j**+**1**]);**  **}**  **}**  **}**  **}**  **};** |

### Khai báo lớp CarList và các thành viên

|  |
| --- |
| class CarList  {      private:          Car\* cars;          int size;          int capacity;  … |

* CarList là lớp quản lý một danh sách các xe (Car).
* cars: Con trỏ kiểu Car trỏ đến mảng động chứa các đối tượng Car.
* size: Số lượng xe hiện tại trong danh sách.
* capacity: Dung lượng tối đa của mảng động hiện tại.

### Hàm expand – mở rộng dung lượng của mảng động

|  |
| --- |
| void expand(){              capacity \*=2;              Car\* newArray = new Car[capacity];              for (int i = 0; i < size; ++i){                  newArray[i] = cars[i];              }              delete[] cars;              cars = newArray;          } |

* Khi mảng cars đầy, hàm expand sẽ tăng gấp đôi dung lượng của mảng.
* Tạo mảng newArray với dung lượng gấp đôi (capacity \*= 2).
* Sao chép toàn bộ phần tử từ cars sang newArray.
* Giải phóng bộ nhớ của mảng cars cũ bằng delete[].
* Con trỏ cars sẽ trỏ đến newArray để cập nhật thành mảng mở rộng.

### Hàm khởi tạo (Constructor) và hàm hủy (Destructor)

|  |
| --- |
| public:          CarList(int cap = 10): size(0), capacity(cap){              cars = new Car[capacity];          }          ~CarList() {delete[] cars;} |

* **Constructor**: Khởi tạo CarList với dung lượng mặc định là 10 nếu không có tham số. Đặt size là 0 và cấp phát bộ nhớ cho mảng cars với dung lượng capacity.
* **Destructor**: Giải phóng bộ nhớ của mảng cars khi đối tượng CarList bị hủy.

### Hàm addCar – thêm xe vào danh sách

|  |
| --- |
| void addCar(const Car& c){              if (size == capacity) expand();                  cars[size++] = c;          } |

* Hàm addCar thêm một đối tượng Car vào danh sách.
* Nếu mảng cars đầy (size == capacity), gọi expand()để mở rộng dung lượng.
* Thêm đối tượng c vào vị trí cars[size], sau đó tăng size lên một đơn vị (size++).

### Hàm removeCar – xóa xe khỏi danh sách theo số VIN

|  |
| --- |
| void removeCar(const string& vin){              for (int i = 0; i<size; i++){                  if(cars[i].getVIN() == vin){                      for (int j = i; j < size-1; ++j){                          cars[j] = cars[j+1];                      }                      --size;                      break;                  }              }          } |

* Hàm removeCar tìm xe theo số VIN (vin) và xóa nó nếu tìm thấy.
* Duyệt qua từng xe trong danh sách, nếu tìm thấy xe có VIN khớp, dịch chuyển các phần tử sau đó lên một vị trí để lấp chỗ trống.
* Giảm size của danh sách xuống một đơn vị để cập nhật số lượng xe hiện tại.

### Hàm displayAll – Hiển thị tất cả các xe trong danh sách

|  |
| --- |
| void displayAll() const{              for (int i = 0; i < size; i++){                  cout << cars[i] << endl;              }          } |

 Hàm displayAll in ra tất cả các xe trong danh sách bằng cách sử dụng toán tử << đã được định nghĩa lại cho lớp Car.

 Từ khóa const sau tên hàm biểu thị rằng hàm này không làm thay đổi trạng thái của đối tượng CarList.

### Hàm sortByPrice – Sắp xếp danh sách xe theo giá

|  |
| --- |
| void sortByPrice(){              for (int i = 0; i < size-1; ++i){                  for (int j = 0; j < size - i - 1; ++j){                      if(cars[j].getPrice() > cars[j+1].getPrice()){                          swap(cars[j], cars[j+i]);                      }                  }              }          } |

* Hàm sortByPrice sắp xếp danh sách xe theo giá sử dụng thuật toán **Bubble Sort**.
* Nếu giá của cars[j] lớn hơn cars[j+1], hoán đổi hai xe này.
* **Lỗi**: Trong dòng swap(cars[j], cars[j+i]), nên là swap(cars[j], cars[j+1]) để hoán đổi hai xe liên tiếp khi điều kiện thỏa mãn.

### Hàm sortByYear – Sắp xếp xe theo năm sản xuất

|  |
| --- |
| void sortByYear(){              for (int i = 0; i < size - 1; ++i){                  for (int j = 0; j < size - i - 1; ++j){                      if(cars[j].getYear() > cars[j+1].getYear()){                          swap(cars[j], cars[j+1]);                      }                  }              }          } |

* Hàm sortByYear sắp xếp danh sách xe theo năm sản xuất cũng sử dụng **Bubble Sort**.
* Nếu năm sản xuất của cars[j] lớn hơn cars[j+1], hoán đổi hai xe này.

## Xây dựng lớp “app” để quản lý chương trình, gồm có menu và thực hiện các thao tác ở lớp danh sách

### Định nghĩa lớp Menu

|  |
| --- |
| class Menu{      private:          CarList carList;      public:          void displayMenu(){              int choice;              do {                  cout << "\n---Chon mot trong cac lua chon sau---\n";                  cout << "1. Them mot mau xe\n";                  cout << "2. Xoa mot mau xe\n";                  cout << "3. Xem danh sach cac mau xe\n";                  cout << "4. Sap xep cac mau xe theo gia\n";                  cout << "5. Sap xep cac mau xe theo nam san xuat\n";                  cout << "0. Thoat\n";                  cout << "Nhap lua chon: ";                  cin >> choice;                  switch (choice)                  {                      case 1:{                          Car c;                          cin >> c;                          carList.addCar(c);                          break;                      }                      case 2:{                          string vin;                          cout << "Nhap so VIN cua xe can xoa: ";                          cin >> vin;                          carList.removeCar(vin);                          break;                      }                      case 3:{                          carList.displayAll();                          break;                      }                      case 4:{                          carList.sortByPrice();                          cout << "Danh sach xe da sap xep theo gia: " << endl;                          carList.displayAll();                          break;                      }                      case 5:{                          carList.sortByYear();                          cout << "Danh sach xe da sap xep theo nam san xuat: " << endl;                          carList.displayAll();                          break;                      }                      case 0:{                          cout << "Thoat chuong trinh.\n";                          break;                      }                      default:                          cout << "Lua chon khong phu hop. Vui long thu lai.";                  }              } while(choice != 0);          }  }; |

Thành viên dữ liệu carList:

* CarList carList;: Đối tượng CarList này sẽ quản lý danh sách xe trong chương trình.

Phương thức displayMenu - Hiển thị và xử lý menu

* Phương thức displayMenu hiển thị một menu và chờ người dùng chọn một trong các lựa chọn. Sau khi chọn, chương trình sẽ thực hiện hành động tương ứng và tiếp tục hiển thị menu cho đến khi người dùng chọn thoát (nhập 0).

### Lặp do-while – Vòng lặp điều khiển menu

|  |
| --- |
| do {        //Hiển thị menu và xử lý lựa chọn người dùng  } while(choice != 0); |

Lặp do-while đảm bảo rằng menu sẽ hiển thị ít nhất một lần. Vòng lặp sẽ tiếp tục cho đến khi người dùng chọn thoát (choice = 0).

### Hiển thị các lựa chọn và nhập lựa chọn từ người dùng

|  |
| --- |
| cout << "\n---Chon mot trong cac lua chon sau---\n";     cout << "1. Them mot mau xe\n";    cout << "2. Xoa mot mau xe\n";     cout << "3. Xem danh sach cac mau xe\n";     cout << "4. Sap xep cac mau xe theo gia\n";     cout << "5. Sap xep cac mau xe theo nam san xuat\n";     cout << "0. Thoat\n";     cout << "Nhap lua chon: ";     cin >> choice; |

Menu hiển thị các lựa chọn khác nhau để người dùng nhập một số (choice) tương ứng với hành động mong muốn.

### Switch – xử lý các lựa chọn

Trong cấu trúc switch, mỗi lựa chọn của người dùng sẽ được xử lý bởi một khối mã khác nhau:

Case 1: Thêm một mẫu xe

|  |
| --- |
| case 1:{            Car c;            cin >> c;            carList.addCar(c);            break;        } |

* Người dùng nhập thông tin xe mới. Đối tượng Car mới (c) được tạo và đọc từ người dùng qua toán tử >> đã được định nghĩa trong Car.
* Sau đó, đối tượng c được thêm vào danh sách carList bằng phương thức addCar.

Case 2: Xóa một mẫu xe

|  |
| --- |
| case 2:{            string vin;            cout << "Nhap so VIN cua xe can xoa: ";            cin >> vin;            carList.removeCar(vin);            break;           } |

* Yêu cầu người dùng nhập số VIN của xe cần xóa.
* Sau khi nhập, phương thức removeCar của carList được gọi để xóa xe có VIN đó khỏi danh sách.

Case 3: Xem danh sách các mẫu xe

|  |
| --- |
| case 3:{            carList.displayAll();            break;           } |

* Hiển thị toàn bộ danh sách xe hiện có bằng phương thức displayAll của carList.

Case 4: Sắp xếp danh sách theo giá

|  |
| --- |
| case 4:{           carList.sortByPrice();           cout << "Danh sach xe da sap xep theo gia: " << endl;           carList.displayAll();           break;         } |

* Sắp xếp danh sách xe theo giá, sau đó hiển thị danh sách đã được sắp xếp. Phương thức sortByPrice trong carList thực hiện sắp xếp, và displayAll in ra kết quả.

Case 5: Sắp xếp danh sách theo năm sản xuất

|  |
| --- |
| case 5:{           carList.sortByYear();           cout << "Danh sach xe da sap xep theo nam san xuat: " << endl;           carList.displayAll();           break;         } |

* Sắp xếp danh sách xe theo năm sản xuất bằng phương thức sortByYear, sau đó hiển thị danh sách đã sắp xếp.

Case 0: Thoát chương trình

|  |
| --- |
| case 0:{           cout << "Thoat chuong trinh.\n";           break;         } |

* Thoát khỏi chương trình và hiển thị thông báo "Thoat chuong trinh."

Mặc định: Xử lý lựa chọn không hợp lệ

|  |
| --- |
| default:           cout << "Lua chon khong phu hop. Vui long thu lai."; |

* Nếu người dùng nhập lựa chọn không có trong danh sách, hiển thị thông báo lỗi và yêu cầu chọn lại.

## Xây dựng hàm main để chạy chương trình

|  |
| --- |
| int main(){      Menu menu;      menu.displayMenu();      return 0;  } |

* Menu menu;: Tạo một đối tượng menu của lớp Menu. Đối tượng này sẽ cung cấp các chức năng quản lý danh sách xe thông qua một giao diện đơn giản.
* menu.displayMenu();: Gọi phương thức displayMenu() của menu. Phương thức này hiển thị menu lựa chọn và cho phép người dùng nhập các lệnh để thêm, xóa, sắp xếp, hoặc hiển thị danh sách xe.
* return 0;: Trả về 0 cho hệ điều hành để báo rằng chương trình đã kết thúc thành công.

# PHÂN TÍCH CHƯƠNG TRÌNH – BÀI TOÁN 2

## Khai báo các thư viện sử dụng trong chương trình

## Cấu trúc dữ liệu và lớp cơ bản

### Cấu trúc Node

|  |
| --- |
| struct Node {      string english;      string vietnamese;      Node\* left;      Node\* right;      Node(string eng, string viet): english(eng), vietnamese(viet), left(nullptr), right(nullptr) {}  }; |

Cấu trúc Node đại diện cho một nút trong cây nhị phân tìm kiếm. Mỗi từ và nghĩa của từ sẽ được lưu trữ trong một nút riêng biệt.

Các thành phần của Node:

* string english: Đây là trường dữ liệu lưu trữ từ tiếng Anh.
* string vietnamese: Đây là trường dữ liệu lưu trữ nghĩa tiếng Việt của từ tiếng Anh.
* Node\* left: Con trỏ trỏ đến nút con bên trái. Nút con trái sẽ chứa từ có thứ tự từ điển nhỏ hơn từ ở nút hiện tại.
* Node\* right: Con trỏ trỏ đến nút con bên phải. Nút con phải sẽ chứa từ có thứ tự từ điển lớn hơn từ ở nút hiện tại.

Hàm khởi tạo Node:

* Node(string eng, string viet): Đây là hàm khởi tạo dùng để tạo một nút mới với từ tiếng Anh và nghĩa tiếng Việt được truyền vào. Khi một Node mới được tạo, các con trỏ left và right của nó được khởi tạo là nullptr, biểu thị rằng nút đó chưa có nút con.

### Lớp Binary Search Tree (BST)

Lớp BST (Binary Search Tree - Cây nhị phân tìm kiếm) dùng để quản lý và sắp xếp các từ vựng trong từ điển theo thứ tự từ điển. BST cung cấp các phương thức để chèn, xóa, tìm kiếm và duyệt qua các từ trong từ điển.

#### Thuộc tính của BST:

|  |
| --- |
| private:      Node\* root; |

* Node\* root: Con trỏ trỏ đến nút gốc của cây nhị phân tìm kiếm. Nút gốc là điểm bắt đầu của tất cả các thao tác trong cây.

#### Phương thức insert:

|  |
| --- |
| //Insert a node      Node\* insert(Node\* node, string english, string vietnamese){          if (node == nullptr) {              return new Node(english, vietnamese);          }          if (english < node->english) {              node->left = insert(node->left, english, vietnamese);          }          else if (english > node->english) {              node->right = insert(node->right, english, vietnamese);          }          return node;      } |

**Mục đích:** Chèn một từ mới vào cây nhị phân tìm kiếm.

Thao tác:

* Nếu cây trống (tức node == nullptr), hàm sẽ tạo một Node mới chứa từ tiếng Anh và nghĩa tiếng Việt được cung cấp.
* Nếu từ tiếng Anh cần chèn nhỏ hơn từ ở nút hiện tại, hàm sẽ gọi đệ quy để chèn từ vào cây con bên trái.
* Nếu từ tiếng Anh cần chèn lớn hơn từ ở nút hiện tại, hàm sẽ gọi đệ quy để chèn từ vào cây con bên phải.

**Kết quả:** Từ mới được chèn vào cây đúng vị trí theo thứ tự từ điển.

#### Phương thức findMin:

|  |
| --- |
| //Finding the minimum node      Node\* findMin(Node\* node){          while (node && node->left != nullptr) node = node->left;          return node;      } |

**Mục đích:** Tìm nút có từ nhỏ nhất trong cây hoặc cây con.

Thao tác:

* Duyệt từ nút hiện tại đến nút trái nhất (là nút nhỏ nhất trong cây theo thứ tự từ điển).

**Kết quả:** Trả về con trỏ đến nút có từ nhỏ nhất.

#### Phương thức remove:

|  |
| --- |
| //Delete a node      Node\* remove(Node \*node, string english){          if (node == nullptr) return node;          if (english < node->english){              node->left = remove(node->left, english);          }          else if (english > node->english){              node->right = remove(node->right, english);          }          else{              if (node->left == nullptr){                  Node\*temp = node->right;                  delete node;                  return temp;              }              else if (node->right == nullptr){                  Node\* temp = node ->left;                  delete node;                  return temp;              }              Node\* temp = findMin(node->right);              node->english = temp->english;              node->vietnamese = temp->vietnamese;              node->right = remove(node->right, temp->english);          }          return node;      } |

**Mục đích:** Xóa một từ khỏi cây nhị phân tìm kiếm.

Thao tác:

* Tìm vị trí của từ cần xóa trong cây.
* Khi tìm thấy từ cần xóa:
* Nếu nút không có con, xóa trực tiếp nút đó.
* Nếu nút chỉ có một con (trái hoặc phải), liên kết nút cha với nút con của nó và xóa nút hiện tại.
* Nếu nút có cả hai con, tìm nút nhỏ nhất trong cây con bên phải (sử dụng findMin) và thay thế giá trị của nút hiện tại bằng giá trị của nút này, sau đó xóa nút đó.

**Kết quả:** Từ được xóa khỏi cây.

#### Phương thức search:

|  |
| --- |
| //Searching a node      Node\* search(Node\* node, string english) {          if (node == nullptr || node-> english == english){              return node;          }          if (english < node->english){              return search(node->left, english);          }          else {              return search(node->right, english);          }      } |

**Mục đích:** Tìm một từ trong cây nhị phân tìm kiếm.

Thao tác:

* So sánh từ cần tìm với từ trong nút hiện tại.
* Nếu từ cần tìm nhỏ hơn từ trong nút hiện tại, tiếp tục tìm trong cây con bên trái.
* Nếu lớn hơn, tiếp tục tìm trong cây con bên phải.
* Nếu tìm thấy từ cần tìm, trả về nút chứa từ đó.

**Kết quả:** Trả về con trỏ đến nút chứa từ cần tìm hoặc nullptr nếu từ không tồn tại.

#### Phương thức inOrder:

|  |
| --- |
| //In-order traversal      void inOrder(Node\* node, ofstream &outFile){          if (node != nullptr){              inOrder(node->left, outFile);              outFile << node->english << ": " << node->vietnamese << endl;              inOrder(node->right, outFile);          }      } |

**Mục đích:** Duyệt cây theo thứ tự từ điển và ghi từ và nghĩa của nó ra tệp.

Thao tác:

* Duyệt qua cây nhị phân theo thứ tự trái - gốc - phải (trong cây BST, thứ tự này sẽ trả về các từ theo thứ tự từ điển).
* Ghi từ và nghĩa của nó ra tệp tin.

**Kết quả:** Xuất tất cả từ và nghĩa của chúng ra tệp theo thứ tự từ điển.

#### Phương thức clear:

|  |
| --- |
| //Clear      void clear(Node\* node){          if (node != nullptr){              clear(node->left);              clear(node->right);              delete node;          }      } |

**Mục đích:** Giải phóng bộ nhớ của toàn bộ cây nhị phân tìm kiếm.

Thao tác:

* Duyệt cây theo đệ quy để xóa từng nút.

**Kết quả:** Giải phóng tất cả bộ nhớ mà cây đã sử dụng, tránh rò rỉ bộ nhớ khi đối tượng BST bị hủy.

### Lớp Dictionary

Lớp Dictionary trong đoạn mã là lớp dùng để quản lý từ điển Anh-Việt thông qua một cây nhị phân tìm kiếm (BST). Lớp này cung cấp các chức năng cơ bản để thêm, xóa, cập nhật, tìm kiếm từ trong từ điển, cũng như nạp từ điển từ tệp tin và lưu từ điển xuống tệp tin. Nó sử dụng đối tượng BST làm cấu trúc lưu trữ chính cho các từ và nghĩa của chúng.

#### Thành phần của lớp Dictionary:

|  |
| --- |
| private:          BST bst; |

Đây là thuộc tính chính của lớp Dictionary, đại diện cho cây nhị phân tìm kiếm (BST) chứa các từ và nghĩa.

bst là một đối tượng của lớp BST, lưu trữ tất cả các từ tiếng Anh và nghĩa tiếng Việt trong từ điển.

#### Phương thức loadFromFile:

|  |
| --- |
| void loadFromFile(const string& filename){              ifstream inFile(filename);              if (!inFile){                  cout << "Error opening file" << endl;                  return;              }              string english, vietnamese;              while (inFile >> english && getline(inFile, vietnamese)){                  if (!vietnamese.empty() && vietnamese[0] == ' '){                      vietnamese.erase(0, 1);                      bst.insert(english, vietnamese);                  }              }          } |

**Mục đích**: Đọc dữ liệu từ một tệp văn bản và chèn các từ vào cây BST.

**Đầu vào**: Một string đại diện cho tên tệp cần đọc.

Thao tác:

* Mở tệp văn bản bằng cách sử dụng ifstream.
* Nếu không mở được tệp, thông báo lỗi sẽ được hiển thị.
* Đọc lần lượt từng từ tiếng Anh và nghĩa tiếng Việt trong tệp:
* Từ tiếng Anh được đọc bằng inFile >> english.
* Nghĩa tiếng Việt được đọc từ phần còn lại của dòng bằng getline(inFile, vietnamese).
* Nếu vietnamese bắt đầu bằng một khoảng trắng (có thể do format của tệp tin), khoảng trắng này sẽ bị xóa để chuẩn hóa dữ liệu.
* Sau đó, từ và nghĩa sẽ được chèn vào BST thông qua bst.insert(english, vietnamese).

#### Phương thức saveToFile:

|  |
| --- |
| void saveToFile(const string& filename){              ofstream outFile(filename);              if(!outFile){                  cout << "Error opening file!" << endl;                  return;              }              bst.inOrder(outFile);              outFile.close();          } |

**Mục đích:** Ghi dữ liệu từ BST vào một tệp văn bản.

**Đầu vào:** Một string đại diện cho tên tệp cần ghi.

Thao tác:

* Mở tệp văn bản để ghi bằng ofstream.
* Nếu không mở được tệp, thông báo lỗi sẽ được hiển thị.
* Gọi bst.inOrder(outFile), phương thức này duyệt qua tất cả các nút trong cây nhị phân tìm kiếm theo thứ tự từ điển và ghi từ và nghĩa của chúng ra tệp.
* Kết quả: Toàn bộ từ điển được ghi vào tệp dưới dạng cặp từ tiếng Anh và nghĩa tiếng Việt theo thứ tự từ điển.

#### Phương thức addWord:

|  |
| --- |
| void addWord(const string& english, const string& vietnamese){              bst.insert(english, vietnamese);          } |

**Mục đích**: Thêm một từ và nghĩa vào từ điển.

**Đầu vào**: string english là từ tiếng Anh, string vietnamese là nghĩa tiếng Việt.

Thao tác:

* Gọi bst.insert(english, vietnamese) để thêm từ vào cây nhị phân tìm kiếm.

**Kết quả**: Từ và nghĩa được thêm vào từ điển theo đúng thứ tự từ điển.

#### Phương thức removeWord:

|  |
| --- |
| void removeWord(const string& english){              bst.remove(english);          } |

**Mục đích:** Xóa một từ khỏi từ điển.

**Đầu vào:** string english là từ tiếng Anh cần xóa.

Thao tác:

* Gọi bst.remove(english) để tìm và xóa từ trong cây nhị phân tìm kiếm.

**Kết quả:** Từ và nghĩa của nó được xóa khỏi từ điển.

#### Phương thức updateWord:

|  |
| --- |
| void updateWord(const string& english, const string& newVietnamese){              bst.update(english, newVietnamese);          } |

**Mục đích:** Cập nhật nghĩa mới cho một từ có sẵn trong từ điển.

**Đầu vào:** string english là từ tiếng Anh cần cập nhật, string newVietnamese là nghĩa tiếng Việt mới.

Thao tác:

* Gọi bst.update(english, newVietnamese) để tìm từ và thay đổi nghĩa tiếng Việt của từ đó.

**Kết quả:** Nghĩa tiếng Việt của từ được cập nhật.

#### Phương thức findWord:

|  |
| --- |
| void findWord(const string& english){              Node\* result = bst.search(english);              if (result){                  cout << "Found: " << result->english << " - " << result->vietnamese << endl;              }              else {                  cout << "Word not found!" << endl;              }          } |

**Mục đích:** Tìm kiếm một từ trong từ điển và in nghĩa của nó ra màn hình nếu tìm thấy.

**Đầu vào:** string english là từ tiếng Anh cần tìm.

**Thao tác:**

* Gọi bst.search(english) để tìm từ trong cây nhị phân tìm kiếm.
* Nếu tìm thấy, in ra từ và nghĩa của từ đó.
* Nếu không tìm thấy, in thông báo "Word not found!".

**Kết quả:** Hiển thị nghĩa của từ hoặc thông báo nếu từ không tồn tại.

# KẾT QUẢ THỰC NGHIỆM

# KẾT LUẬN

# LỜI CẢM ƠN

# TÀI LIỆU THAM KHẢO