LỜI CẢM ƠN

Lời đầu tiên, cho em xin dành lời cảm ơn chân thành nhất gửi đến giảng viên hướng dẫn, Ths. Trương Thị Thanh Tuyền đã gợi ý và đề xuất cho em đề tài này, nhờ đó em đã tích luỹ được nhiều kiến thức và kinh nghiệm hơn trong quá trình thực hiện đề tài. Và trên hết, con cũng xin gửi lời cảm ơn đến cha mẹ đã luôn luôn bên cạnh, động viên và nhắc nhỡ, khuyến kích và là hậu phương vững chắc cho con trên con đường học tập. Em xin cảm ơn các anh chị khoá trên trong Câu lạc bộ Tin học Khoa Công nghệ thông tin và Truyền thông đã cho em lời khuyên trong quá trình thực hiện khi em gặp khó khăn.

Cần Thơ, ngày tháng năm 2020

Người viết

Đào Minh Trung Thuận

MỤC LỤC NỘI DUNG

[1 TỔNG QUAN 6](#_Toc39814599)

[1.1 MÔ TẢ BÀI TOÁN 6](#_Toc39814600)

[1.2 MỤC TIÊU CẦN ĐẠT ĐƯỢC 6](#_Toc39814601)

[1.3 HƯỚNG GIẢI QUYẾT VÀ KẾ HOẠCH THỰC HIỆN 6](#_Toc39814602)

[1.3.1 Hướng giải quyết 6](#_Toc39814603)

[1.3.2 Kế hoạch thực hiện 7](#_Toc39814604)

[2 CƠ SỞ LÝ THUYẾT 8](#_Toc39814605)

[2.1 CÁC KHÁI NIỆM 8](#_Toc39814606)

[2.1.1 Kiểu dữ liệu trừu tượng Danh sách (List) 8](#_Toc39814607)

[2.1.2 Tập hợp 9](#_Toc39814608)

[2.1.3 Kiểu dữ liệu trừu tượng Tự điển (Dictionary) 10](#_Toc39814609)

[2.1.4 Cấu trúc Bảng băm (HashTable) 10](#_Toc39814610)

[2.1.5 Bảng băm mở 10](#_Toc39814611)

[2.1.6 Các phương pháp xác định hàm băm 11](#_Toc39814612)

[2.2 KẾT QUẢ VẬN DỤNG LÝ THUYẾT 11](#_Toc39814613)

[3 GIẢI QUYẾT VẤN ĐỀ 12](#_Toc39814614)

[3.1 Phân tích yêu cầu bài toán, xây dựng cấu trúc dữ liệu 12](#_Toc39814615)

[3.1.1 Phân tích yêu cầu bài toàn 12](#_Toc39814616)

[3.1.2 Xây dựng Cấu trúc dữ liệu 12](#_Toc39814617)

[3.2 SƠ ĐỒ CHỨC NĂNG, LƯU ĐỒ GIẢI THUẬT 15](#_Toc39814618)

[3.2.1 Sơ đồ chức năng 15](#_Toc39814619)

[3.3 CÁCH THỨC XÂY DỰNG CHƯƠNG TRÌNH 17](#_Toc39814620)

[3.3.1 Cấu trúc dự án 17](#_Toc39814621)

[3.3.2 Xây dựng chương trình chính 18](#_Toc39814622)

[4 KẾT LUẬN VÀ ĐÁNH GIÁ 24](#_Toc39814623)

[4.1 KẾT QUẢ ĐẠT ĐƯỢC 24](#_Toc39814624)

[4.2 HẠN CHẾ, NGUYÊN NHÂN 24](#_Toc39814625)

[4.3 HƯỚNG PHÁT TRIỂN ĐỀ TÀI 24](#_Toc39814626)

[5 PHẦN PHỤC LỤC 25](#_Toc39814627)

[5.1 Các khái niệm bổ sung 25](#_Toc39814628)

[5.1.1 Từ bài toán đến chương trình 25](#_Toc39814629)

[5.1.2 Kiểu dữ liệu, Kiểu dữ liệu trừu tượng, Cấu trúc dữ liệu 26](#_Toc39814630)

[5.2 Cài đặt các kiểu dữ liệu 27](#_Toc39814631)

[5.2.1 Chuỗi ký tự (String) 27](#_Toc39814632)

[5.2.2 Từ (Word) 29](#_Toc39814633)

[5.3 Xây dựng chương trình 42](#_Toc39814634)

[5.4 Hướng dẫn cài đặt chương trình 49](#_Toc39814635)

[5.5 Hướng dẫn sử dụng 49](#_Toc39814636)

[6 TÀI LIỆU THAM KHẢO 56](#_Toc39814637)

MỤC LỤC SƠ ĐỒ

[Sơ đồ 2.1: Sơ đồ Cài đặt danh sách bằng mảng 6](#_Toc39814272)

[Sơ đồ 2.2: Sơ đồ Danh sách liên kết đơn 7](#_Toc39814273)

[Sơ đồ 2.3: Sơ đồ Cài đặt Danh sách liên kết đôi 7](#_Toc39814274)

[Sơ đồ 2.4: Sơ đồ cài đặt Tự điển bằng Bảng băm mở 9](#_Toc39814275)

[Sơ đồ 3.1: Sơ đồ DFD mức ngữ cảnh 13](#_Toc39814276)

[Sơ đồ 3.2: Sơ đồ DFD mức 0 14](#_Toc39814277)

[Sơ đồ 3.3: Sơ đồ DFD mức 1 15](#_Toc39814278)

[Sơ đồ 3.4: Sơ đồ cấu trúc dự án 16](#_Toc39814279)

[Sơ đồ 3.5: Sơ đồ hoạt động chương trình chính 17](#_Toc39814280)

[Sơ đồ 3.6: Sơ đồ hoạt động chức năng tra cứu từ 18](#_Toc39814281)

[Sơ đồ 3.7: Sơ đồ hoạt động chức năng thêm từ mới 19](#_Toc39814282)

[Sơ đồ 3.8: Sơ đồ hoạt động chức năng xoá từ 20](#_Toc39814283)

[Sơ đồ 3.9: Sơ đồ hoạt động chức năng xem lịch sử tra cứu 21](#_Toc39814284)

[Sơ đồ 5.1: Sơ đồ các bước tiếp cận bài toán 23](#_Toc39814285)

MỤC LỤC HÌNH ẢNH

[Hình 5.1: Giao diện chính của chương trình 48](#_Toc39814304)

[Hình 5.2: Chức năng tra cứu từ 48](#_Toc39814305)

[Hình 5.3: Kết quả tra cứu từ nếu tìm thấy từ 49](#_Toc39814306)

[Hình 5.4: Thông báo không tìm thấy từ 49](#_Toc39814307)

[Hình 5.5: Nhập từ mới cần thêm 50](#_Toc39814308)

[Hình 5.6: Kết quả thêm từ mới 50](#_Toc39814309)

[Hình 5.7: Thông báo thêm từ không thành công 51](#_Toc39814310)

[Hình 5.8: Nhập từ cần xoá 51](#_Toc39814311)

[Hình 5.9: Thông báo xoá từ thành công 52](#_Toc39814312)

[Hình 5.10: Thông báo xoá từ không thành công 52](#_Toc39814313)

[Hình 5.11: Chức năng xem lịch sử tra cứu 53](#_Toc39814314)

[*Hình 5.12: Lịch sử trống* 53](#_Toc39814315)

MỤC LỤC BIỂU BẢNG

[Bảng 1.1: Bảng kế hoạch thực hiện 7](#_Toc39814334)

[Bảng 5.1: Định nghĩa kiểu String 27](#_Toc39814335)

[Bảng 5.2: Danh sách các hàm của kiểu String 28](#_Toc39814336)

[Bảng 5.3: Định nghĩa kiểu Word 29](#_Toc39814337)

[Bảng 5.4: Định nghĩa kiểu Element 29](#_Toc39814338)

[Bảng 5.5: Danh sách các hàm của kiểu Element 29](#_Toc39814339)

[Bảng 5.6: Định nghĩa kiểu NodeLinkedList 30](#_Toc39814340)

[Bảng 5.7: Danh sách hàm kiểu NodeLinkedList 30](#_Toc39814341)

[Bảng 5.8: Định nghĩa kiểu LinkedList 31](#_Toc39814342)

[Bảng 5.9: Các hàm kiểu LinkedList 31](#_Toc39814343)

[Bảng 5.10: Kỹ thuật băm cho Bảng băm 32](#_Toc39814344)

[Bảng 5.11: Đinh nghĩa kiểu NodeHashTable 33](#_Toc39814345)

[Bảng 5.12: Các hàm kiểu NodeHashTable 33](#_Toc39814346)

[Bảng 5.13: Định nghĩa kiểu HashTable 33](#_Toc39814347)

[Bảng 5.14: Các Hàm trong HashTable 37](#_Toc39814348)

[Bảng 5.15: Cài đặt kiểu History 37](#_Toc39814349)

[Bảng 5.16: Các hàm của kiểu History 38](#_Toc39814350)

[Bảng 5.17: Cài đặt kiểu Dictionary 38](#_Toc39814351)

[Bảng 5.18: Các hàm kiểu Dictionary 41](#_Toc39814352)

[Bảng 5.19: Xây dựng các hàm xử lý giao diện 42](#_Toc39814353)

[Bảng 5.20: Xây dựng giao diện chức năng chương trình 44](#_Toc39814354)

[Bảng 5.21: Xây dựng chức năng tra cứu từ 44](#_Toc39814355)

[Bảng 5.22: Xây dựng chức năng thêm từ mới 45](#_Toc39814356)

[Bảng 5.23: Xây dựng chức năng xoá từ 46](#_Toc39814357)

[Bảng 5.24: Xây dựng chức năng xem lịch sử tra cứu 47](#_Toc39814358)

[Bảng 5.25: Cú pháp lệnh đi đến đường dẫn 48](#_Toc39814359)

[Bảng 5.26: Ví dụ đi đến đường dẫn chương trình 48](#_Toc39814360)

[Bảng 5.27: Lệnh chạy chương trình 48](#_Toc39814361)

# TỔNG QUAN

## MÔ TẢ BÀI TOÁN

Xây dựng và cài đặt chương trình tự điển Anh – Việt với yêu cầu kỹ thuật sử dụng thuật toán trên bảng băm để thực hiện các thao tác cơ bản:

* Tra cứu từ.
* Thêm từ mới và tự điển.
* Xoá từ khỏi tự điển.

Ngoài ra, bổ sung thêm một số chức năng khác:

* Xem lại từ đã tra cứu (lịch sử tra cứu từ).

## MỤC TIÊU CẦN ĐẠT ĐƯỢC

Với nội dung yêu cầu của đề tài, mục tiêu cần đạt được bao gồm:

* Nắm vững được các khái niệm về kiểu dữ liệu, kiểu dữ liệu trừu tượng.
* Năm vững và cài đặt được các kiểu dữ liệu trừu tượng, cụ thể gồm có: Bảng băm, Danh sách liên kết.
* Vận dụng kiến thức để giải quyết bài toán xây dựng chương trình tự điển Anh – Việt với các chức năng theo yêu cầu.

Ngoài những mục tiêu yêu cầu về kỹ năng chuyên môn, còn có các mục tiêu khác cần phải đạt được:

* Biết cách hiểu và giải quyết vấn đề yêu cầu của đề tài đã đặt ra.
* Biết cách lên kế hoạch và đề sắp xếp thời gian cho quá trình thực hiện đề tài.
* Rèn luyện được các kỹ năng viết báo cáo, tài liệu, kỹ năng thuyết trình.

## HƯỚNG GIẢI QUYẾT VÀ KẾ HOẠCH THỰC HIỆN

### Hướng giải quyết

Để có thể thực hiện đề tài một cách tốt nhất, cần tìm hiểu lại kỹ các kiến thức về Cấu trúc dữ liệu, các kiểu dữ liệu và kiểu dữ liệu trừu tượng, cách cài đặt cũng như thuật toán đã được học ở môn Cấu trúc dữ liệu.

Vì đề tài yêu cầu cao về kỹ thuật cài đặt cũng như vận dụng thuật toán, nên hướng giải quyết cho đề tài sẽ là sử dụng ngôn ngữ lập trình C/C++ làm ngôn ngữ để phát triển. Xây dựng cài đặt các kiểu dữ liệu trừu tượng Danh sách liên kết, Bảng băm và từ đó áp dụng để xây dựng chương trình.

### Kế hoạch thực hiện

|  |  |  |
| --- | --- | --- |
| **STT** | **Thời gian** | **Nội dung** |
| 1 | Tuần 2 | Tập trung, nghe giảng viên hướng dẫn, nhân đề tài. |
| 2 | Tuần 3 | Đăng ký đề tài qua Email. |
| 3 | Tuần 4 | Tìm hiểu tài liệu, lên phương án, hướng giải quyết đề tài. |
| 4 | Tuần 5 | Viết tài liệu đặt tả yêu cầu bài toán và hướng giải quyết. |
| 5 | Tuần 6 | Chỉnh sử tài liệu đặt tả, và nộp cho giảng viên hướng dẫn. |
| 6 | Tuần 7 | Viết cơ sở lý thuyết, khái niệm và vận dụng. |
| 7 | Tuần 8 | Viết tài liệu phân tích yêu cầu, tài liệu xây dựng kiểu dữ liệu. |
| 8 | Tuần 9 | Viết tài liệu sơ đồ chức năng, vẽ lưu đồ giải thuật. |
| 9 | Tuần 10 | Nộp tài liệu thiết kế giải thuật và dữ liệu sử dụng. |
| 10 | Tuần 11 | Viết tài liệu cách thức xây dựng chương trình. |
| 11 | Tuần 12 | Cài đặt kiểu dữ liệu trừu tượng Danh sách liên kết, Bảng băm. |
| 12 | Tuần 13 | Cài đặt chương trình, xây dựng giao diện. |
| 13 | Tuần 14 | Kiểm thử, điều chỉnh tài liệu, viết tài liệu hướng sử dụng và cài đặt chương trình cho người sử dụng. |
| 14 | Tuần 15 | Hoàn thiện chương trình, kiểm thử lần cuối và hoàn thiện tài liệu báo cáo. |

Bảng 1.1: Bảng kế hoạch thực hiện

# CƠ SỞ LÝ THUYẾT

## CÁC KHÁI NIỆM

### Kiểu dữ liệu trừu tượng Danh sách (List)

#### Khái niệm danh sách

Mô hình toán học của danh sách là một tập hợp hữu hạn các phần tử có cùng một kiểu, mà tổng quát ta gọi là kiểu phần tử (ElementType). Ta biểu diễn danh sách như là một chuỗi các phần tử của nó: a1, a2, …, an-1, an (với n ≥ 0). Nếu n = 0 ta nói danh sách rỗng (empty list). Nếu n > 0 ta gọi a1 là phần tử đầu tiên và an là phần tử cuối cùng của danh sách. Số phần tử của danh sách ta gọi là độ dài của danh sách.

#### Cài đặt Danh sách

* **Cài đặt danh sách bằng mảng (danh sách đặc):** Dùng một mảng để lưu giữ liên tiếp các phần tử của danh sách từ vị trí đầu tiên của mảng. Với cách cài đặt này, ta phải ước lượng số phần tử của danh sách để khai báo số phần tử của mảng cho thích hợp. Dễ thấy rằng số phần tử của mảng phải được khai báo không ít hơn số phần tử của danh sách. Nói chung là mảng còn thừa một số chỗ trống. Mặt khác ta phải lưu giữ độ dài hiện tại của danh sách, độ dài này cho biết danh sách có bao nhiêu phần tử và cho biết phần nào của mảng còn trống. Ta định nghĩa vị trí của một phần tử trong danh sách là chỉ số của mảng tại vị trí lưu trữ phần tử đó + 1 (vì phần tử đầu tiên trong mảng là chỉ số 0).

Sơ đồ 2.1: Sơ đồ Cài đặt danh sách bằng mảng

* **Cài đặt danh sách bằng con trỏ (danh sách liên kết, LinkedList):** Dùng con trỏ để liên kết các ô chứa các phần tử. Trong cách cài đặt này các phần tử của danh sách được lưu trữ trong các ô, mỗi ô có thể chỉ đến ô chứa phần tử kế tiếp trong danh sách. Để quản lý danh sách ta chỉ cần một biến giữ địa chỉ ô chứa phần tử đầu tiên của danh sách, tức là một con trỏ trỏ đến phần tử đầu tiên trong danh sách. Biến này gọi là chỉ điểm đầu danh sách (Header). Để đơn giản hóa vấn đề, trong chi tiết cài đặt, Header là một biến cùng kiểu với các ô chứa các phần tử của danh sách và nó có thể được cấp phát ô nhớ y như một ô chứa phần tử của danh sách. Tuy nhiên Header là một ô đặc biệt nên nó không chứa phần tử nào của danh sách, trường dữ liệu của ô này là rỗng, chỉ có trường con trỏ Next trỏ tới ô chứa phần tử đầu tiên thật sự của danh sách. Nếu danh sách rỗng thì  
  Header → next trỏ tới NULL. Việc cấp phát ô nhớ cho Header như là một ô chứa dữ liệu bình thường nhằm tăng tính đơn giản của các giải thuật thêm, xoá các phần tử trong danh sách. Ta có Danh sách liên kết đơn

Position

a1

a2

…

an

NULL

1

2

n

HEADER

Sơ đồ 2.2: Sơ đồ Danh sách liên kết đơn

Từ đó ta mở rộng cặt đặt Danh sách bằng các phần tử gồm 2 con trỏ Next và Previous để thuận tiên cho việc di chuyển tới và lùi giữa các phần tử trong danh sách. Lúc này, ta sẽ có thêm một ô đặt biệt nữa là Tail đặt ở cuối danh sách. Ta có danh sách Liên kết đôi

Position

1

HEAD

a1

a2

…

an

TAIL

NULL

2

n

NULL

Sơ đồ 2.3: Sơ đồ Cài đặt Danh sách liên kết đôi

### Tập hợp

Khái niệm tập hợp cũng giống như trong toán học, đó là sự tập hợp các thành viên (Members) hoặc phần tử (Elements). Tất cả các phần tử của tập hợp là khác nhau. Tập hợp có thể có thứ tự hoặc không có thứ tự, tức là, có thể có quan hệ thứ tự xác định trên các phần tử của tập hợp hoặc không. Tuy nhiên, để đơn giản hoá, chúng ta giả sử rằng các phần tử của tập hợp có thứ tự tuyến tính, tức là, trên tập hợp S có quan hệ < và = thoả mản hai tính chất:

* Với mọi a, b ∈ S thì a < b hoặc b < a hoặc a = b
* Với mọi a, b, c ∈ S, a < b và b < c thì a < c

### Kiểu dữ liệu trừu tượng Tự điển (Dictionary)

Từ điển là một kiểu dữ liệu trừu tượng tập hợp đặc biệt, trong đó chúng ta chỉ quan tâm đến các phép toán thêm phần tử (InsertSet), xoá phần tử (DeleteSet), lấy phần tử (Member) và tạo Tự điển trống (MakeNullSet). Sở dĩ chúng ta nghiên cứu từ điển là do trong nhiều ứng dụng không sử dụng đến các phép toán hợp, giao, hiệu của hai tập hợp. Ngược lại ta cần một cấu trúc làm sao cho việc tìm kiếm, thêm và bớt phần tử có phần hiệu quả nhất gọi là từ điển. Chúng ta cũng chấp nhận MakeNullSet như là phép khởi tạo cấu trúc.

Cài đặt Tự điển: Thực chất việc cài đặt từ điển bằng mảng hoàn toàn giống với việc cài đặt danh sách đặc không có thứ tự. Có rất nhiều kỹ thuật cài đặt Tự điển: cài đặt Tự điển bằng mảng, cài đặt bằng Danh sách liên kết hoặc có thể áp dụng kỹ thuật xây dựng bảng băm để cài đặt…

### Cấu trúc Bảng băm (HashTable)

#### Kỹ thuật băm (Hashing)

Kỹ thuật băm hay phương pháp bămlà một phương pháp tính toán trực tiếp vị trí của mảng lưu trữ một phần tử của tập hợp dựa vào giá trị khoá của phần tử này, tức là tính toán “địa chỉ” trực tiếp từ khoá.

Giả sử, ta có một mảng gồm n phần tử được đánh số từ 0 đến n - 1 và một tập hợp A muốn lưu trữ vào trong mảng này. Như vậy với mỗi phần tử x ∈ A ta phải dựa vào khoá này để suy ra một giá trị số nguyên thuộc khoảng [0, n - 1] là vị trí cất giữ khoá này. Nói cách khác, ta chọn một hàm h(x), khi tìm kiếm phần tử x ta chỉ cần tính h(x):

h: A → 0..n-1

x → h(x)

### Bảng băm mở

Bảng băm mở là một ý tưởng đơn giản cho việc giải quyết đụng độ là tổ chức thêm một danh sách cho từng tập hợp các khoá có cùng giá trị băm. Tức là chia một tập hợp đang xét thành một số hữu hạn các lớp con bên trong. Nếu bảng băm có n phần từ được đánh số từ 0 đến n - 1 và h là hàm băm thì lớp thứ k là một danh sách gồm tất cả các phần tử x sao cho h(x) = k.

Tổ chức danh sách có thể theo bất kỳ cách nào, nhưng do số lượng các khoá trong mỗi lớp là không biết trước, nên mỗi lớp ta tổ chức một Danh sách liên kết cho các phần tử của có cùng kết quả băm khoá. Mỗi lớp như vậy gọi là một lô chứa (Bucket).

0

1

2

n-1

a00

a01

a03

a04

NULL

a10

NULL

NULL

…

an0

an1

…

anm

NULL

Sơ đồ 2.4: Sơ đồ cài đặt Tự điển bằng Bảng băm mở

### Các phương pháp xác định hàm băm

Có rất nhiều cách để xác định hàm băm, ví dụ như:

* Phương pháp đồng dư (phương pháp chia)
* Phương pháp nhân
* Phương pháp tách

## KẾT QUẢ VẬN DỤNG LÝ THUYẾT

Vận dụng các kiến thức trong Cấu trúc dữ liệu, ta mô hình hoá bài toán xây dựng chương trình Tự điển Anh – Việt dựa trong cấu trúc Kiểu dữ liệu trừu tượng Tự điển là một tập hợp gồm các phần tử từ (Word) gồm hai thành phần là trường từ tiếng Anh và trường nghĩa tiếng Việt. Áp dụng kỹ thuật cài đặt Bảng băm để xây dựng Tự điển. Sử dụng Danh sách liên kết làm lịch sử tra cứu.

# GIẢI QUYẾT VẤN ĐỀ

## Phân tích yêu cầu bài toán, xây dựng cấu trúc dữ liệu

### Phân tích yêu cầu bài toàn

Với yêu cầu từ bài toán đặt ra cụ thể là xây dựng Tự điển Anh – Việt bằng Bảng băm, ta có các phân tích như sau:

* Đối với kỹ thuật băm để xây dụng Bảng băm, để đơn giản ta sử dụng phương pháp chia, tức là vị trí được xác định bằng cách lấy giá trị của Khoá đem chia cho số Bucket của Tự điển và lấy phần dư của nó, ta sẽ có hàm băm với công thức: **h(x) = x mod n** (với số Bucket n = 100).
* Đối với từ (Word) trong Tự điển, ta sẽ lấy trường từ tiếng Anh làm khoá đại diện, để thuận tiện cho việc băm, ta sẽ chuyển đổi từ tiếng Anh dưới dạng chuỗi (String) trở thành dạng số nguyên dương bằng cách lấy tổng mã Ascii của các ký tự của từ. Mỗi Bucket sẽ gồm một dánh sách các từ có cùng giá trị băm, ta sẽ dùng Danh sách liên kết đơn để xây dựng danh sách đó.
* Dữ liệu Word sẽ được lưu vào một tệp dữ liệu (Data file) dưới dạng file text (phần mở rộng của file là .txt). Khi chương trình chạy sẽ tự động nạp dữ liệu bằng cách đọc file. Các chức năng như thêm, xoá từ sẽ thay đổi lên file.
* Ngoài ra, để lưu lại lịch sử tra cứu từ, ta sẽ sử dụng Danh sách liên kết đôi để làm lịch sử. Bằng cách đó, ta có thể xem lại từ đã tra cứu một cách nhanh hơn thay vì sử dụng Danh sách liên kết đơn.
* Chương trình sẽ lấy ngôn ngữ lập trình C/C++ làm ngôn ngữ phát triển, các mã lệnh (code) sẽ được phân chia và quản lý theo Kiểu dữ liệu thành các header include file trong C/C++.

Ngoài ra, còn phải có các yêu cầu như sau:

* Chương trình được gọi là hoàn thiện khi đáp ứng đầy đủ các năng tra cứu, thêm từ, xoá từ, xem lịch sử tra cứu.
* Chương trình được xây dựng hướng đến giao diện, người sử dụng thao tác dễ dàng nhất dù chương trình chạy trên nền Console.
* Chương trình chạy mượt, không quá chậm, không có lỗi.
* Dễ dàng bảo trì, nâng cấp và thiết kế sao cho dễ gỡ lỗi nếu có lỗi xảy ra.

### Xây dựng Cấu trúc dữ liệu

Ta sẽ xây dựng Cấu trục dữ liệu, Kiểu dữ liệu mang tính gần tương đồng nhất với Lập trình hướng đối tượng. Một Kiểu dữ liệu giống với một Lớp, gồm có hàm tạo (Create), hàm huỷ (Destroy) và các hàm và thủ tục khác của nó. Đối với các kiểu dữ liệu trừu tượng chỉ cài đặt các hàm và phép tính cần thiết cho chương trình.

#### Chuỗi ký tự (String)

Kiểu String là một dãy các ký tự Char, ta sẽ dụng con trỏ để xây dụng kiểu String vì đối với bài toàn, ta không biết chính xác số ký tự của một chuỗi là bao nhiêu.

Các hàm trong String gồm có:

* Hàm khởi tạo chuỗi rỗng.
* Hàm huỷ chuỗi.
* Hàm kiểm tra hai chuỗi.
* Hàm thêm ký tự vào chuỗi.
* Hàm nhập chuỗi từ bàn phím.

#### Từ (Word)

Kiểu Word đại diện cho từ có cấu trúc gồm hai thành phần là trường từ tiếng Anh và trường nghĩa tiếng Việt dựa trên kiểu String vừa định nghĩa.

#### Phần tử Tự điển (Element)

Để tương quan hơn trong thiết kế, ta xây dựng Kiểu Phần tử cho Tự điện sẽ là kiểu Word, sử dụng trường từ tiếng Anh để làm khoá đại diện cho phần tử.

Như vậy ta sẽ có một hàm của kiểu Element là Hàm lấy khoá.

#### Danh sách liên kết (Linked List)

Cài đặt Danh sách dạng Danh sách liên kết đôi, trước hết ta cài đặt kiểu Nút liên kết (NodeLinkedList)cho Danh sách liên kết và các hàm của kiểu Node. Định nghĩa kiểu Node có cấu trúc gồm ba thành phần là phần dữ liệu, phần con trỏ chỉ tới và con trỏ chỉ lùi.

Các hàm của kiểu NodeLinkedList này gồm có hai hàm:

* Hàm khởi tạo.
* Hàm huỷ.

Sau đó, định nghĩa kiểu LinkedList với cấu trúc gồm hai thành phần là trường Head và trường Tail dựa trên kiểu NodeLinkedList.

Các hàm của kiểu LinkedList cũng gồm có hai hàm:

* Hàm khởi tạo.
* Hàm huỷ.

#### Bảng băm (Hash Table)

Thiết kế thuật toán băm bằng phương pháp chia với công thức là: **h(x) = x % 100.**

Sử dụng Bảng băm mở cho cài đặt Tự điển, cho nên ta thiết kết Node dưới dạng Node cho danh sách liên kết đơn, gồm một trường dữ liệu data và một trường con trỏ trỏ tới Node kế tiếp.

Các hàm của NodeHashTable gồm có:

* Hàm khởi tạo.
* Hàm huỷ.

Định nghĩa kiểu HashTable là một tập hợp các Danh sách liên kết gồm **100 phần tử** (số phần tử bằng với số chia trong hàm băm). Mỗi phần tử này được gọi là Bucket. Và ta cũng có các hàm cho kiểu HashTable:

* Hàm khởi tạo.
* Hảm huỷ.
* Hàm lấy Node.
* Hàm duyệt qua các Node.
* Hàm thêm Node mới.
* Hàm xoá Node.

#### Lịch sử (History)

Sử dụng Danh sách liên kết đôi để cài đặt History cho dễ dàng việc di chuyển tới lùi thông qua liên kết của các Node trong LinkedList. Ngoài ra ta định nghĩa thêm một kiểu con trỏ để truy xuất History.

Các hàm của History gồm:

* Hàm khởi tạo.
* Hàm huỷ.
* Hàm thêm lịch sử.
* Hàm lấy dữ liệu lịch sử.
* Hàm di chuyển con trỏ cuộn tới.
* Hàm di chuyển con trỏ cuộn lùi.

#### Tự điển (Dictionary)

Sau khi đã cài đặt hết các kiểu dữ liệu, ta chỉ cần định nghĩa kiểu Tự điển là kiểu HashTable. Và cũng xây dựng thêm các hàm cho Dictionary:

* Hàm khởi tạo.
* Hàm huỷ.
* Hàm viết dữ liệu ra file data.txt.
* Hàm đọc file data.txt và nạp dữ liệu vào từ điển.
* Hàm thêm từ mới.
* Hàm xoá từ.
* Hàm tra cứu từ.

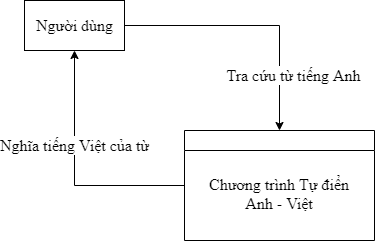
## SƠ ĐỒ CHỨC NĂNG, LƯU ĐỒ GIẢI THUẬT

### Sơ đồ chức năng

Để khái quát hơn về chương trình và chức năng của nó, ta sử dụng Sơ đồ luồng dữ liệu (Data Flow Diagram) để minh hoạ.

#### Mức ngữ cảnh (Context level)

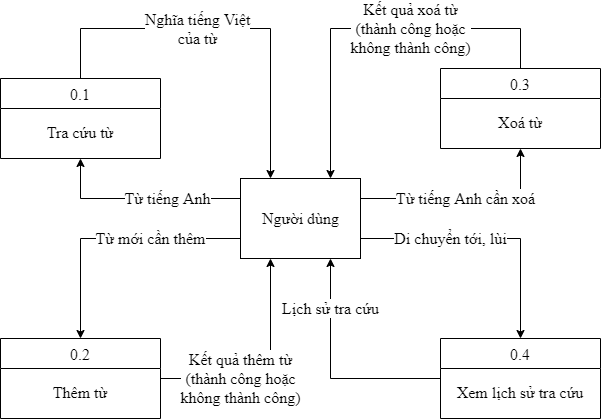
Ở đây, chức năng chính của chương trình sẽ là tra cứu từ tiếng Anh, chương trình sẽ trả kết quả về là nghĩa tiếng Việt của từ đó.



Sơ đồ 3.1: Sơ đồ DFD mức ngữ cảnh

#### Mức 0 (Level 0)

Đi sâu hơn, chương trình sẽ gồm các chức năng: Tra cứu từ, thêm từ, xoá từ và xem lịch sử tra cứu.

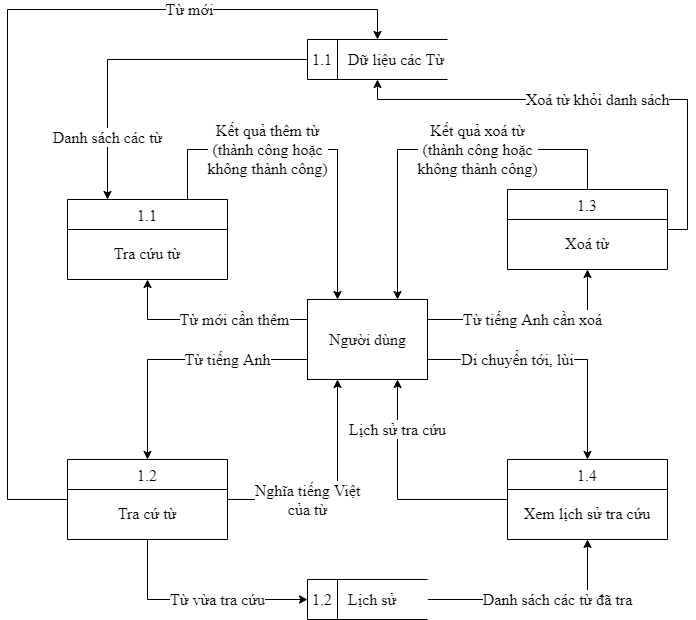


Sơ đồ 3.2: Sơ đồ DFD mức 0

#### Mức 1 (Level 1)

Đi sâu hơn nữa, các chức năng sẽ tương tác đến nơi lưu trữ dữ liệu từ. Chương trình sẽ có dữ liệu tạm thời được lưu trong quá trình chạy chương trình, và dữ liệu cứng sẽ được lưu trữ dưới dạng file text. Khi thêm từ mới và xoá từ sẽ tác động lên cả hai loại dữ liệu, bao gồm dữ liệu cứng và dữ liệu tạm thời (kiểu dữ liệu trừu tượng Tự điển bảng băm).

Khi tra cứu từ, từ vừa tra cứu sẽ được lưu tự động vào Lịch sử dưới dạng dữ liệu tạm thời (được cài đặt bằng Danh sách liên kết).



Sơ đồ 3.3: Sơ đồ DFD mức 1

## CÁCH THỨC XÂY DỰNG CHƯƠNG TRÌNH

### Cấu trúc dự án

Dự án được đặt trong thư mục gốc dictionary gồm các thư mục con:

* Thư mục docs: chứa các tệp tài liệu cho dự án.
* Thư mục src: chức các code dự án, được chia làm:
  + Thư mục data: chứa file data của chương trình (file dữ liệu nạp lúc chương trình khởi động).
  + Thư mục includes: chứa các file header (các file này thương có phần mở rộng là .h) cài đặt các kiểu dữ liệu và các code khác của chương trình.
  + File Main.c: file main chương trình.

dictionary

docs

src

data

includes

Main.c

Sơ đồ 3.4: Sơ đồ cấu trúc dự án

### Xây dựng chương trình chính

Trước khi xây dựng chương trình, ta viết cả hàm để xử lý giao diện cho việc xây dựng giao diện nhanh hơn mà không cần viết lại hàm printf, bao gồm các hàm:

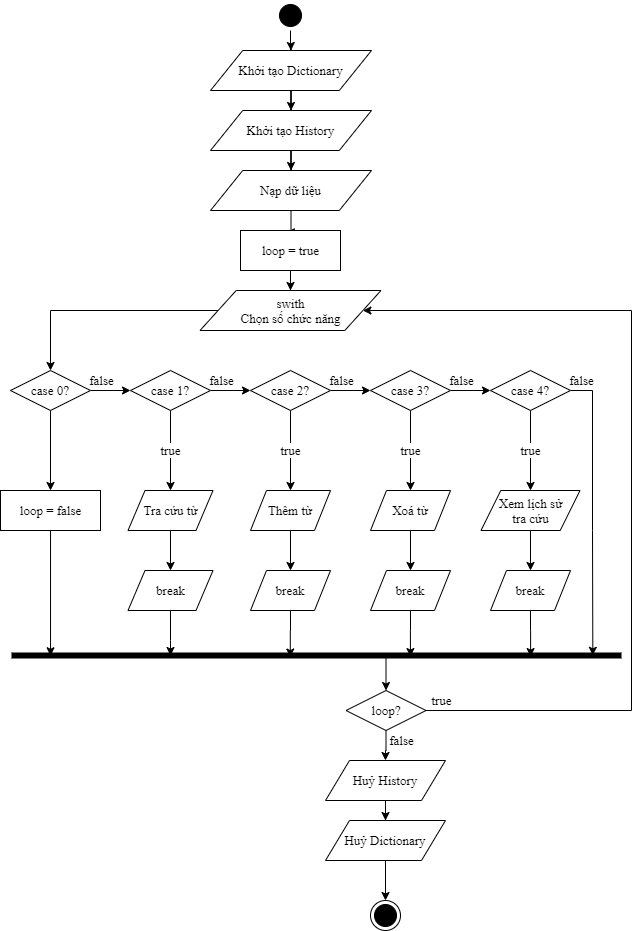
* Hàm in tên chương trình.
* Hàm in danh sách chức năng.
* Hàm in tên chức năng khi người dùng chọn.
* Hàm in kết quả tra cứu từ.

#### Cách thức hoạt động

Chương trình chạy với giao diện chức năng cho người dừng dễ thao tác với các chức năng: tra cứu từ, thêm từ, xoá từ và xem lịch sử tra cứu sẽ ứng với các phím số 1, 2, 3, 4 và để thoát chương trình, người dùng chỉ cần ấn phím 0.

Để xây dựng được chương trình vậy, ta sử dụng switch…case… và đọc ký tự từ bàn phím để biết người dùng chọn chức năng gì. Nhưng trước đó, ta phải khởi tạo Tự điển và Lịch sử, sau đó nạp dữ liệu từ data.txt vào Tự điển.

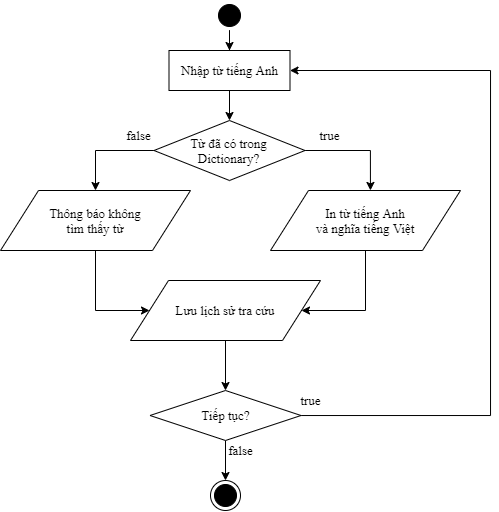
Ứng với mỗi chức năng ta chỉ cần gọi hàm đã cài đặt của kiểu Dictionary.



Sơ đồ 3.5: Sơ đồ hoạt động chương trình chính

#### Chức năng tra cứu từ

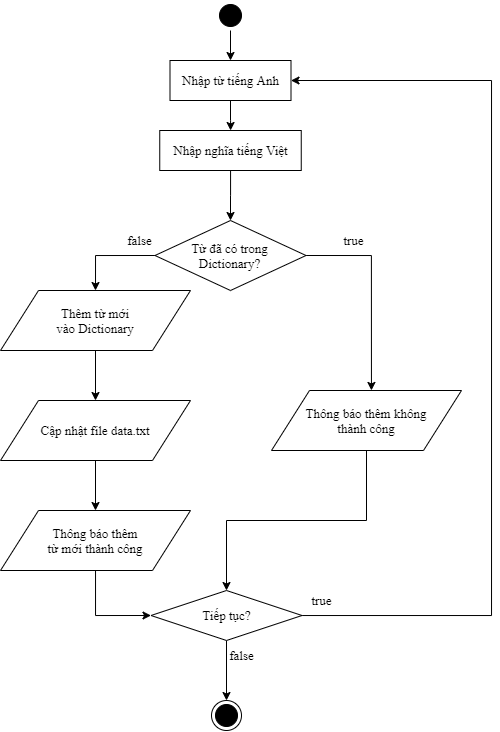
Chức năng tra cứu từ sẽ cho phép người dùng nhập từ tiếng Anh vào để tra cứu sau đó trả kết về nghĩa tiếng Việt nếu tìm thấy từ, còn nếu không thì trả về thông báo không tìm thấy từ.



Sơ đồ 3.6: Sơ đồ hoạt động chức năng tra cứu từ

#### Chức năng thêm từ mới

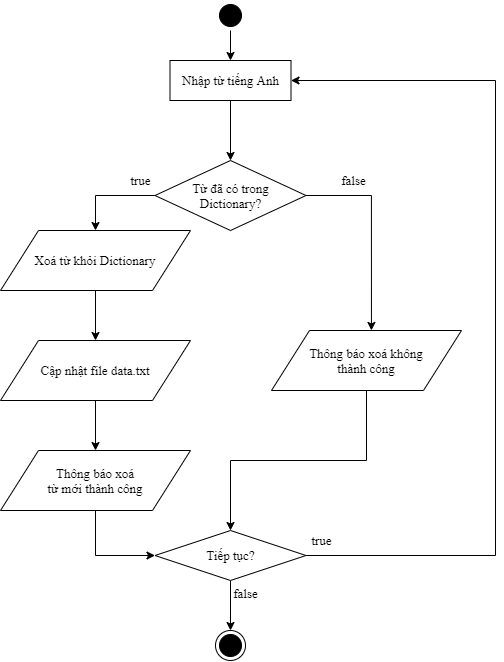
Chức năng thêm từ mới cho phép nhập từ tiếng Anh mới và nghĩa tiếng Việt của nó. Nếu thêm từ mới thành công, chương trình sẽ thông báo kết quả thành công. Nếu từ mà người dùng nhập vào đã tồn tại trong Từ điển thì chương trình sẽ thông báo từ đã tồn tại và thêm không thành công.



Sơ đồ 3.7: Sơ đồ hoạt động chức năng thêm từ mới

#### Chức năng xoá từ

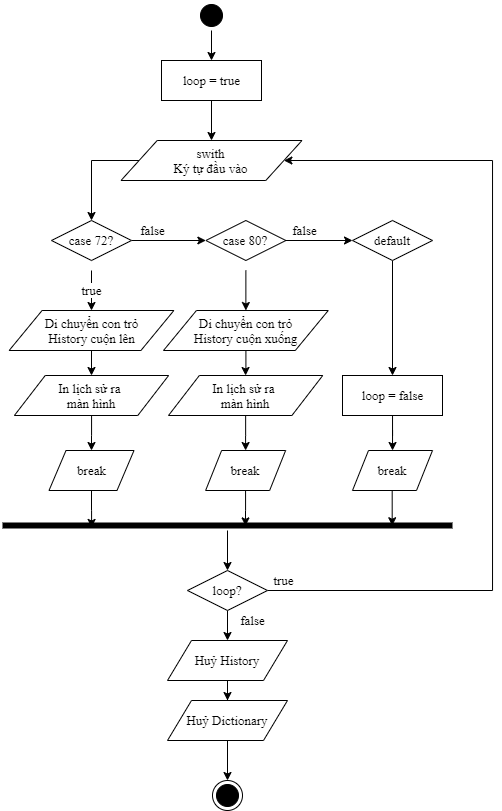
Chức năng xoá từ cho phép người dùng nhập từ tiếng Anh cần xoá để xoá từ đó ra khỏi Tự điển. Nếu từ đó tồn tại trong Tự điển, thông báo xoá từ thành công sẽ được hiện lên. Nếu từ đó không có trong Tự điển, chương trình sẽ thông báo xoá không thành công.



Sơ đồ 3.8: Sơ đồ hoạt động chức năng xoá từ

#### Xem lịch sử tra cứu

Chức năng xem lịch sử tra cứu cho phép người dùng có thể xem lại những từ mà mình đã tra cứu. Bằng cách sử dụng hai phím mũi tên lên và xuống mà cuộn con trỏ để có thể xem lịch sử. Lịch sử sẽ được ghi lại theo trình tự từ vừa tra cứu gần nhất đến xa nhất, ghi lại từ tiếng Anh và cả nghĩa tiếng Việt của nó. Nếu từ tiếng Anh không có trong Tự điển, mục nghĩa tiếng Việt sẽ để trống.



Sơ đồ 3.9: Sơ đồ hoạt động chức năng xem lịch sử tra cứu

# KẾT LUẬN VÀ ĐÁNH GIÁ

## KẾT QUẢ ĐẠT ĐƯỢC

* Hiểu và nắm rõ về cấu trúc dữ liệu, các cài đặt và ứng dụng cấu trúc dữ liệu, kiểu dữ liệu trừu tượng.
* Thiết kế kiểu dữ liệu trừu tượng theo hướng gần với hướng đối tượng.
* Về phía chương trình thì được hoàn thiện đây đủ các chức năng cơ bản của một Tự điển: tra cứu từ, thêm từ mới, xoá từ và xem lịch sử tra cứu.
* Giao diện tương đối dễ sử dụng cho người dùng.

## HẠN CHẾ, NGUYÊN NHÂN

Về thiết kế thuật toán:

* Cách đọc và ghi dữ liệu ra file data vẫn chưa được tối ưu, vì mỗi lần thêm một từ mới hoặc xoá một từ, chương trình phải ghi lại toàn bộ các từ trong Tự điện ra file, nếu số lượng từ trong Tự điển tương đối lớn, quá trình ghi và đọc file sẽ diễn ra lâu, gây chậm trễ, ảnh hưởng trãi nghiệm người dùng.
* Thuật toán tìm kiếm trên Tự điển ở Danh sách liên kết trên Bucket là cách tìm kiếm tuần tự tuyến tính (Linear Search), chưa được tối ưu, nếu danh sách có nhiều phần tử, khả năng thời gian chờ tìm kiếm sẽ lâu.

Về giao diện được xây dựng trên nền Console nên không đẹp và bắt mắt như trên giao diện Form hiện đại, nguyên nhân do chương trình được viết thuần bằng C/C++ với trình biên dịch GCC ra nền Console.

## HƯỚNG PHÁT TRIỂN ĐỀ TÀI

* Cãi thiện thuật toán, giải quyết vấn đề ghi và đọc file data, không ghi lại toàn bộ file nếu có thay đổi. Hoặc chuyển sang một hệ quản trị cơ sở dữ liệu, khi đó hệ quản trị cơ sở dữ liệu sẽ đảm nhận việc tối ưu việc truy vấn, chỉnh sửa dữ liệu.
* Cải thiện thuật toán tìm kiếm trên Tự điện (cụ thể là trên Danh sách). Có thể áp dụng các thuật toán tìm kiếm khác tối ưu hơn như: Tìm kiếm nhị phân (Binary Search) hay Tìm kiếm nội suy (Interpolation Search).
* Cải thiện giao diện người dùng, xây dựng giao diện người dụng đẹp mắt và dễ sử dụng hơn, người dùng có thể thao tác bằng chuột thay vì bằng bàn phím.
* Chuyển nền tảng và ngôn ngữ xây dựng chương trình sang một nền tảng khác như chương trình Tự điển Online với nền tảng Web app, hoặc cho điện thoại di động với App Mobile Android và IOS, hoặc vẫn trên máy tính nhưng với giao diện đồ hoạ với Winform viết bằng C# hay Java Application với Java…

# PHẦN PHỤC LỤC

## Các khái niệm bổ sung

### Từ bài toán đến chương trình

Để giải một bài toán trong thực tế bằng máy tính ta phải bắt đầu từ việc xác định bài toán. Nhiều thời gian và công sức bỏ ra để xác định bài toán cần giải quyết, tức là phải trả lời rõ ràng câu hỏi "phải làm gì?" sau đó là "làm như thế nào?". Thông thường, khi khởi đầu, hầu hết các bài toán là không đơn giản, không rõ ràng. Để giảm bớt sự phức tạp của bài toán thực tế, ta phải hình thức hóa nó, nghĩa là phát biểu lại bài toán thực tế thành một bài toán hình thức (hay còn gọi là mô hình toán).

Khi đã có mô hình thích hợp cho một bài toán ta cần cố gắng tìm cách giải quyết bài toán trong mô hình đó. Khởi đầu là tìm một giải thuật, đó là một chuỗi hữu hạn các chỉ thị mà mỗi chỉ thị có một ý nghĩa rõ ràng và thực hiện được trong một lượng thời gian hữu hạn.

Một khi đã có mô hình thích hợp cho bài toán, ta cần hình thức hoá một giải thuật trong thuật ngữ của mô hình đó. Khởi đầu là viết những mệnh đề tổng quát rồi tinh chế dần thành những chuỗi mệnh đề cụ thể hơn, cuối cùng là các chỉ thị thích hợp trong một ngôn ngữ lập trình.

Sơ đồ 5.1: Sơ đồ các bước tiếp cận bài toán

Tóm lại, để tiếp cận với một bài toán ta cần trãi qua các bước:

* Mô hình hoá bài toán bằng một mô hình toán học thích hợp.
* Tìm giải thuật trên mô hình này. Giải thuật có thể mô tả một cách không hình thức, tức là nó chỉ nêu phương hướng giải hoặc các bước giải một cách tổng quát.
* Phải hình thức hoá giải thuật bằng cách viết một thủ tục bằng ngôn ngữ giả, rồi chi tiết hoá dần các bước giải tổng quát ở trên, kết hợp với việc dùng các kiểu dữ liệu trừu tượng và các cấu trúc điều khiển trong ngôn ngữ lập trình để mô tả giải thuật. Ở bước này, nói chung, ta có một giải thuật tương đối rõ ràng, nó gần giống như một chương trình được viết trong ngôn ngữ lập trình, nhưng nó không phải là một chương trình chạy được vì trong khi viết giải thuật ta không chú trọng nặng đến cú pháp của ngôn ngữ và các kiểu dữ liệu còn ở mức trừu tượng chứ không phải là các khai báo cài đặt kiểu trong ngôn ngữ lập trình.
* Cài đặt giải thuật trong một ngôn ngữ lập trình cụ thể. Ở bước này ta dùng các cấu trúc dữ liệu được cung cấp trong ngôn ngữ để thể hiện các kiểu dữ liệu trừu tượng, các bước của giải thuật được thể hiện bằng các lệnh và các cấu trúc điều khiển trong ngôn ngữ lập trình được dùng để cài đặt giải thuật.

### Kiểu dữ liệu, Kiểu dữ liệu trừu tượng, Cấu trúc dữ liệu

Trong tin học, trừu tượng hóa nghĩa là đơn giản hóa, làm cho nó sáng sủa hơn và dễ hiểu hơn. Cụ thể trừu tượng hóa là che đi những chi tiết, làm nổi bật cái tổng thể. Trừu tượng hóa có thể thực hiện trên hai khía cạnh:

* Trừu tượng hóa chương trình là sự định nghĩa các chương trình con để tạo ra các phép toán trừu tượng (sự tổng quát hóa của các phép toán nguyên thủy). Trừu tượng hóa chương trình cho phép phân chia chương trình thành các chương trình con.
* Trừu tượng hóa dữ liệu là định nghĩa các kiểu dữ liệu trừu tượng. Một kiểu dữ liệu trừu tượng là một mô hình toán học cùng với một tập hợp các phép toán trừu tượng được định nghĩa trên mô hình đó.

Mặc dù các thuật ngữ kiểu dữ liệu (data type), cấu trúc dữ liệu (data structure), kiểu dữ liệu trừu tượng (abstract data type) nghe như nhau, nhưng chúng có ý nghĩa rất khác nhau. Cụ thể như sau:

* Kiểu dữ liệu là một tập hợp các giá trị và một tập hợp các phép toán trên các giá trị đó. Ví dụ kiểu Boolean là một tập hợp có 2 giá trị True, False và các phép toán trên nó như Or, And, Not…
* Kiểu dữ liệu có hai loại:
  + Kiểu dữ liệu sơ cấp là kiểu dữ liệu mà giá trị dữ liệu của nó là đơn nhất. Ví dụ: kiểu Boolean, Integer…
  + Kiểu dữ liệu có cấu trúc hay còn gọi là cấu trúc dữ liệu là kiểu dữ liệu mà giá trị dữ liệu của nó là sự kết hợp của các giá trị khác. Ví dụ: ARRAY là một cấu trúc dữ liệu.
* Một kiểu dữ liệu trừu tượng là một mô hình toán học cùng với một tập hợp các phép toán trên nó. Có thể nói kiểu dữ liệu trừu tượng là một kiểu dữ liệu do chúng ta định nghĩa ở mức khái niệm (conceptual), nó chưa được cài đặt cụ thể bằng một ngôn ngữ lập trình. Khi cài đặt một kiểu dữ liệu trừu tượng trên một ngôn gnữ lập trình cụ thể, chúng ta phải thực hiện hai nhiệm vụ:
  + Biểu diễn kiểu dữ liệu trừu tượng bằng một cấu trúc dữ liệu hoặc một kiểu dữ liệu trừu tượng khác đã được cài đặt.
  + Viết các chương trình con thực hiện các phép toán trên kiểu dữ liệu trừu tượng mà ta thường gọi là cài đặt các phép toán.

## Cài đặt các kiểu dữ liệu

### Chuỗi ký tự (String)

Định nghĩa kiểu:

|  |
| --- |
| // String type  typedef char \*String; |

Bảng 5.1: Định nghĩa kiểu String

Các hàm của kiểu gồm có:

|  |
| --- |
| **Hàm khởi tạo chuỗi rỗng:** Cấp phát vùng nhớ, và gán chuỗi rỗng. |
| String String\_CreateEmpty()  {    String string = (String)malloc(sizeof(char));    strcpy(string, "");    return string;  } |
| **Hàm huỷ:** Giải phóng con trỏ. |
| void String\_Destroy(String *string*)  {    free(string);  } |
| **Hàm kiểm tra hai chuỗi String có bằng nhau hay không, bỏ qua ký tự in hoa và in thường:** Kiểm tra về độ dài chuỗi và duyệt qua từng ký tự trong chuỗi. |
| bool String\_IsEqualIgnoreCase(*const* String *string1*, *const* String *string2*)  {  *if* (strlen(string1) == strlen(string2))    {      size\_t index = *0*;  *while* (index < strlen(string1))      {  *if* (tolower(string1[index]) == tolower(string2[index]))        {          index++;        }  *else*        {          return false;        }      }      return true;    }    else    {      return false;    }  } |
| **Hàm thêm ký tự vào sau chuỗi:** Cấp phát thêm vùng nhớ, và thêm ký tự vào cuối chuỗi. |
| void String\_Push(String *\*string*, *const* char *c*)  {  *const* size\_t length = strlen(\*string);    (\*string) = (String)realloc((\*string),  sizeof(char) \* (length + *2*));    (\*string)[length] = c;    (\*string)[length + *1*] = '*\0*';  } |
| **Hàm nhập chuỗi từ bàn phím:** Đọc ký tự từ bàn phím đến khi gặp ký tự xuống dòng. |
| String String\_Input()  {    String string = (String)malloc(sizeof(char));    strcpy(string, "");    fflush(stdin);    char cursor;  *while* ((cursor = getchar()) != '*\n*' && cursor != EOF)    {      String\_Push(&string, cursor);    }  *return* string;  } |

Bảng 5.2: Danh sách các hàm của kiểu String

### Từ (Word)

Định nghĩa cấu trúc và kiểu:

|  |
| --- |
| // Word struct  struct WordStruct  {    // English Word    String english;    // Vietnamese mean of Word    String vietnamese;  };  // Word type  typedef struct WordStruct \*Word; |

Bảng 5.3: Định nghĩa kiểu Word

#### Phần tử Tự điển (Element)

Định nghĩa kiểu Element:

|  |
| --- |
| // Element type  typedef Word Element; |

Bảng 5.4: Định nghĩa kiểu Element

Các hàm của kiểu gồm có:

|  |
| --- |
| **Hàm lấy khoá đại diện:** Sử dụng chuỗi tiếng Anh làm khoá. |
| String Element\_GetKey(*const* Element *element*)  {  *return* element->english;  } |

Bảng 5.5: Danh sách các hàm của kiểu Element

#### Danh sách liên kết (Linked List)

Định nghĩa cấu trúc và kiểu:

|  |
| --- |
| // Node Linked list struct  struct NodeLinkedListStruct  {    // Element    Word data;    // Next Node    struct NodeLinkedListStruct \*next;    // Previous Node    struct NodeLinkedListStruct \*previous;  };  // Node Linked list type  typedef struct NodeLinkedListStruct \*NodeLinkedList; |

Bảng 5.6: Định nghĩa kiểu NodeLinkedList

Các hàm của kiểu NodeLinkedList:

|  |
| --- |
| **Hàm khởi tạo:** Khởi tạo một Node với data, 2 con trỏ của Node trở tới NULL. |
| NodeLinkedList NodeLinkedList\_Create(Word *data*)  {    NodeLinkedList node = (NodeLinkedList)malloc(sizeof(  struct NodeLinkedListStruct));    node->data = data;    node->next = *NULL*;    node->previous = *NULL*;  *return* node;  } |
| **Hàm huỷ:** Huỷ dữ liệu và giải phóng con trỏ. |
| void NodeLinkedList\_Destroy(NodeLinkedList *node*)  {    Word\_Destroy(node->data);    free(node);  } |

Bảng 5.7: Danh sách hàm kiểu NodeLinkedList

#### Danh sách liên kết (Linked List)

Định nghĩa cấu trúc và kiểu:

|  |
| --- |
| // Linked list struct  struct LinkedListStruct  {    // Header cursor node    NodeLinkedList head;    // Last cursor node    NodeLinkedList tail;  };  // Linked list type  typedef struct LinkedListStruct \*LinkedList; |

Bảng 5.8: Định nghĩa kiểu LinkedList

Các hàm của kiểu LinkedList:

|  |
| --- |
| **Hàm khởi tạo:** Cấp phát vùng nhớ, ban đầu phần Head và Tail đều trỏ tới NULL. |
| LinkedList LinkedList\_Create()  {    LinkedList list = (LinkedList)malloc(sizeof(  struct LinkedListStruct));    list->head = *NULL*;    list->tail = *NULL*;  *return* list;  } |
| **Hàm huỷ:** Huỷ từng Node và giải phóng con trỏ. |
| void LinkedList\_Destroy(LinkedList *list*)  {  *while* (list->head != *NULL*)    {      NodeLinkedList node = list->head;      list->head = list->head->next;      NodeLinkedList\_Destroy(node);    }    free(list);  } |

Bảng 5.9: Các hàm kiểu LinkedList

#### Bảng băm (Hash Table)

Xây dựng hàm băm

|  |
| --- |
| **Hàm băm với khoá là số:** Công thức hàm băm là **h(x) = x % 100.** |
| // Mod of separate  #define HASH\_SOLUTION\_MOD\_SEPARATE 100  size\_t HashSolution\_HashNumber(size\_t *code*, size\_t *length*)  {    size\_t result = *0*;  *while* (code > *0*)    {      result += code % HASH\_SOLUTION\_MOD\_SEPARATE;      code /= HASH\_SOLUTION\_MOD\_SEPARATE;    }  *return* result % length;  } |
| **Hàm băm với khoá là chuỗi:** Chuyển chuỗi thành số bằng cách tính tổng mã Ascii các ký tự của chuỗi, sau đó thực hiện hàm băm với khoá là số. |
| size\_t HashSolution\_HashString(*const* String *key*, size\_t *length*)  {    size\_t index = *0*, code = *0*;  *while* (index < strlen(key))    {      code += tolower(key[index]);      index++;    }  *return* HashSolution\_HashNumber(code, length);  } |

Bảng 5.10: Kỹ thuật băm cho Bảng băm

Định nghĩa Node:

|  |
| --- |
| // Node Hash Table struct  struct NodeHashTableStruct  {    // Element    Element data;    // Next Node    struct NodeHashTableStruct \*next;  };  // Node Hash table type  typedef struct NodeHashTableStruct \*NodeHashTable; |

Bảng 5.11: Đinh nghĩa kiểu NodeHashTable

Các hàm của kiểu NodeHashTable:

|  |
| --- |
| **Hàm khởi tạo:** Khởi tạo một Node mới với dữ liệu và trỏ tới Node kế tiếp. |
| NodeHashTable NodeHashTable\_Create(*const* Element *data*,  *const* NodeHashTable *nextNode*)  {    NodeHashTable node = (NodeHashTable)malloc(sizeof(  struct NodeHashTableStruct));    node->data = data;    node->next = nextNode;  *return* node;  } |
| **Hàm huỷ:** Huỷ dữ liệu, giải phóng con trỏ. |
| void NodeHashTable\_Destroy(NodeHashTable *node*)  {    Word\_Destroy(node->data);    free(node);  } |

Bảng 5.12: Các hàm kiểu NodeHashTable

Định nghĩa kiểu HashTable:

|  |
| --- |
| // Length of Hash Table  #define LENGTH\_HASHTABLE 100  // Hash Table type  typedef NodeHashTable \*HashTable; |

Bảng 5.13: Định nghĩa kiểu HashTable

Kiểu HashTable gồm có các hàm:

|  |
| --- |
| **Hàm khởi tạo:** Khởi tại Bảng băm, ban đầu các Bucket sẽ bằng NULL. |
| HashTable HashTable\_Create()  {    HashTable hashtable = (HashTable)malloc(sizeof(NodeHashTable) \*  LENGTH\_HASHTABLE);    size\_t index = *0*;  *while* (index < LENGTH\_HASHTABLE)    {      hashtable[index] = *NULL*;      index++;    }  *return* hashtable;  } |
| **Hàm huỷ:** Huỷ từng Node trong Danh sách, giải phóng con trỏ. |
| void HashTable\_Destroy(HashTable *hashtable*)  {    size\_t index = *0*;  *while* (index < LENGTH\_HASHTABLE)    {  *while* (hashtable[index] != *NULL*)      {        NodeHashTable node = hashtable[index]->next;        NodeHashTable\_Destroy(hashtable[index]);        hashtable[index] = node;      }      index++;    }    free(hashtable);  } |
| **Hàm lấy Node:** Lấy Node theo khoá. Băm khoá để xác định vị trí của Bucket, rồi tìm Node có khoá giống với khoá yêu cầu |
| NodeHashTable HashTable\_GetNode(*const* HashTable *hashtable*, *const* String *key*)  {    NodeHashTable node = hashtable[  HashSolution\_HashString(key, LENGTH\_HASHTABLE)];  *while* (node != *NULL*)    {  *if* (String\_IsEqualIgnoreCase(key, Element\_GetKey(node->data)))      {  *return* node;      }  *else*      {        node = node->next;      }    }  *return* *NULL*;  } |
| **Hàm thêm Node mới:** Xác định vị trí của Bucket, sau đó chèn Node mới vào trước các Node khác trong danh sách. |
| bool HashTable\_Insert(HashTable *hashtable*, *const* Element *element*)  {    size\_t index = HashSolution\_HashString(Element\_GetKey(element),   LENGTH\_HASHTABLE);    NodeHashTable currentNode = hashtable[index];    hashtable[index] = NodeHashTable\_Create(element, currentNode);  *return* *true*;  } |
| **Hàm xoá Node theo khoá:** Băm khoá tìm vị trí Bucket, nếu Node đầu trong danh sách Bucket thì huỷ Node đó và trỏ Bucket vào Node phía sau. Ngược lại thì đi Node trong danh sách rồi Huỷ và trỏ Node trước với Node phía sau. |
| bool HashTable\_Delete(HashTable *hashtable*, *const* String *key*)  {    size\_t index = HashSolution\_HashString(key, LENGTH\_HASHTABLE);  *if* (hashtable[index] != *NULL*)    {      NodeHashTable currentNode = hashtable[index];  *if* (String\_IsEqualIgnoreCase(key,  Element\_GetKey(currentNode->data)))      {        hashtable[index] = hashtable[index]->next;        NodeHashTable\_Destroy(currentNode);  *return* *true*;      }  *else*      {        bool deleted = *false*;  *while* (currentNode->next != *NULL* && !deleted)        {  *if* (String\_IsEqualIgnoreCase(key,  Element\_GetKey(currentNode->next->data)))          {            NodeHashTable nextNode = currentNode->next;            currentNode->next = nextNode->next;            NodeHashTable\_Destroy(nextNode);            deleted = *true*;          }  *else*          {            currentNode = currentNode->next;          }        }  *return* deleted;      }    }  *return* *false*;  } |
| **Hàm duyệt qua các Node:** sử dụng con trỏ hàm để tuỳ biến hàm hành động trong việc duyệt qua các Node mà không phải viết lại kỹ thuật duyệt. |
| void HashTable\_ForEach(*const* HashTable *hashtable*, void (\**action*)  (Element *\_element*, size\_t *\_index*))  {    NodeHashTable node;    size\_t indexBucket = *0*;    size\_t index = *0*;  *while* (indexBucket < LENGTH\_HASHTABLE)    {      node = hashtable[indexBucket++];  *while* (node != *NULL*)      {        action(node->data, index++);        node = node->next;      }    }  } |

Bảng 5.14: Các Hàm trong HashTable

#### Lịch sử (History)

Định nghĩa kiểu:

|  |
| --- |
| // History type  typedef LinkedList History;  // CursorHistory type  typedef NodeLinkedList CursorHistory; |

Bảng 5.15: Cài đặt kiểu History

Kiểu History gồm có các hàm:

|  |
| --- |
| **Hàm khởi tạo:** Khởi tạo một Danh sách liên kết đại diện cho History. |
| History History\_Create()  {  *return* LinkedList\_Create();  } |
| **Hàm huỷ:** Huỷ Danh sách liên kết. |
| void History\_Destroy(History *history*)  {    LinkedList\_Destroy(history);  } |
| **Hàm thêm lịch sử:** Tạo một từ mới và thêm vào Danh sách liên kết |
| void History\_Insert(History *history*, String *english*, String *vietnamese*)  {    LinkedList\_Insert(history, Word\_Create(english, vietnamese));  } |
| **Hàm lấy dữ liệu:** Lấy dữ liệu của con trỏ hiện tại. |
| Word History\_GetData(CursorHistory *cursor*)  {  *return* cursor->data;  } |
| **Hàm di chuyển con trỏ đi tới:** Lấy con trỏ kế tiếp từ con trỏ hiện tại (di chuyển tới) |
| CursorHistory History\_Next(CursorHistory *cursor*)  {  *return* cursor->next;  } |
| **Hàm di chuyển con trỏ đi lùi:** Lấy con trỏ lùi lại từ con trỏ hiện tại (di chuyển lùi) |
| CursorHistory History\_Previous(CursorHistory *cursor*)  {  *return* cursor->previous;  } |

Bảng 5.16: Các hàm của kiểu History

#### Tự điển (Dictionary)

Định nghĩa:

|  |
| --- |
| // Dictionary type  typedef HashTable Dictionary; |

Bảng 5.17: Cài đặt kiểu Dictionary

Các hàm của kiểu Dictionary:

|  |
| --- |
| **Hàm khởi tạo:** Khởi tạo một Bảng băm HashTable đại diện cho Dictionary. |
| Dictionary Dictionary\_Create()  {  *return* HashTable\_Create();  } |
| **Hàm huỷ:** Huỷ Bảng băm. |
| void Dictionary\_Destroy(Dictionary *dictionary*)  {    HashTable\_Destroy(dictionary);  } |
| **Hàm viết dữ liệu ra file:** Viết dữ liệu trong Dictionary ra file text. |
| void Dictionary\_WirteData(Element *element*, size\_t *index*)  {    FILE \*file = fopen(CONFIG\_DATA\_FILE, index == *0* ? "w" : "a");    fputs(element->english, file);    fputc('*\n*', file);    fputs(element->vietnamese, file);    fputc('*\n*', file);    fclose(file);    file = *NULL*;  } |
| **Hàm thêm từ:** Kiểm tra từ có tồn tại không rồi thêm phần tử vào HashTable, đồng thời viết ra file text. |
| bool Dictionary\_Insert(Dictionary *dictionary*, *const* String *english*, *const* String *vietnamese*)  {  *if* (HashTable\_GetNode(dictionary, english) == *NULL*)    {  *if* (HashTable\_Insert(dictionary,  Word\_Create(english, vietnamese)))      {        HashTable\_ForEach(dictionary, Dictionary\_WirteData);  *return* *true*;      }    }  *return* *false*;  } |
| **Hàm xoá từ:** Kiểm tra từ có tồn tại hay không rồi xoá từ ra khỏi Dictionary. |
| bool Dictionary\_Delete(Dictionary *dictionary*,  *const* String *english*)  {  *if* (HashTable\_GetNode(dictionary, english) != *NULL*)    {  *if* (HashTable\_Delete(dictionary, english))      {        HashTable\_ForEach(dictionary, Dictionary\_WirteData);  *return* *true*;      }    }  *return* *false*;  } |
| **Hàm nạp dữ liệu:** Đọc dữ liệu từ file text, rồi nạp từ vào Dictionary. |
| void Dictionary\_LoadData(Dictionary *dictionary*)  {    FILE \*file = fopen(CONFIG\_DATA\_FILE, "r+");    char cursor = fgetc(file);  *while* (cursor != EOF)    {      int line = *0*;      String english = String\_CreateEmpty();      String vietnamese = String\_CreateEmpty();  *while* (line++ < *2*)      {  *while* (cursor != '*\n*')        {          String\_Push(line == *1* ? &english : &vietnamese, cursor);          cursor = fgetc(file);        }        cursor = fgetc(file);      }      HashTable\_Insert(dictionary,  Word\_Create(english, vietnamese));      String\_Destroy(english);      String\_Destroy(vietnamese);    }    fclose(file);  } |
| **Hàm tra cứu từ:** Tìm kiếm từ trong Dictionary. |
| Word Dictionary\_Search(Dictionary *dictionary*,  *const* String *english*)  {    NodeHashTable node = HashTable\_GetNode(dictionary, english);  *if* (node == *NULL*)    {  *return* *NULL*;    }  *else*    {  *return* node->data;    }  } |

Bảng 5.18: Các hàm kiểu Dictionary

## Xây dựng chương trình

#### Xử lý giao diện

Các hàm xử lý giao diện:

|  |
| --- |
| **Hàm in tên chương trình** |
| void Console\_ShowBrand()  {    system("cls");    printf("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");    printf("\t\*                                           \*\n");    printf("\t\*      CHƯƠNG TRÌNH TỰ ĐIỂN ANH - VIỆT      \*\n");    printf("\t\*                                           \*\n");    printf("\t\*      Họ và tên: Đào Minh Trung Thuận      \*\n");    printf("\t\*      Mã sinh viên: B1704855               \*\n");    printf("\t\*                                           \*\n");    printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");  } |
| **Hàm in menu chức năng** |
| void Console\_ShowMenu()  {    printf("\tDanh sách chức năng:\n");    printf("\t\t0. Thoát chương trình.\n");    printf("\t\t1. Tra cứu từ.\n");    printf("\t\t2. Thêm từ mới.\n");    printf("\t\t3. Xoá từ.\n");    printf("\t\t4. Lịch sử tra cứu.");  } |
| **Hàm in tên chức năng mà người dùng chọn** |
| void Console\_ShowFunction(String name)  {    Console\_ShowBrand();    printf("\t%s\n", name);    printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");  } |
| **Hàm in kết quả tìm kiếm từ:** Nếu từ đưa vào khác NULL thì in ra tiếng Anh và nghĩa tiếng Việt, ngược lại sẽ in ra thông báo không tìm thấy từ. |
| void Console\_ShowSearchResult(const Word word)  {    if (word != NULL)    {      printf("\tTiếng Anh: %s\n", word->english);      printf("\tTiếng Việt: %s\n\n", word->vietnamese);    }    else    {      printf("\tTra từ không thành công. Không tìm thấy từ.\n\n");    }  } |

Bảng 5.19: Xây dựng các hàm xử lý giao diện

#### Lựa chọn chức năng

Trước tiên ta xây dựng hàm main sẽ có cấu trúc mỗi một case tương ứng với một chức năng như sau:

|  |
| --- |
| *#include* <conio.h>  *#include* "./includes/Dictionary.h"  *#include* "./includes/History.h"  int main()  {    bool loop = *true*;    Dictionary dictionary = Dictionary\_Create();    History history = History\_Create();    char cursor;    Dictionary\_LoadData(dictionary);  *while* (loop)    {      bool loopHistory;      Console\_ShowBrand();      Console\_ShowMenu();  *switch* (getch())  {  *case* '0':         loop = *false*;         Console\_ShowBrand();         printf("*\t*Tạm biệt!");         getch();  *break*;  *case* '1':         //...  *break*;  *case* '2':         //...  *break*;  *case* '3':         //...  *break*;  *case* '4':         //...  *break*;      }    }    History\_Destroy(history);    Dictionary\_Destroy(dictionary);  *return* *0*;  } |

Bảng 5.20: Xây dựng giao diện chức năng chương trình

#### Chức năng tra cứu từ

Xây dựng chức năng tra cứu từ như sau:

|  |
| --- |
| case '1':  do  {  Console\_ShowFunction("TRA CỨU TỪ");  printf("\tNhập từ tiếng Anh muốn tra cứu: ");  String english = String\_Input();  Console\_ShowFunction("TRA CỨU TỪ");  Word result = Dictionary\_Search(dictionary, english);  Console\_ShowSearchResult(result);  History\_Insert(history, english,  result == NULL ? "" : result->vietnamese);  String\_Destroy(english);  printf("\tNhấn phím Enter để tiếp tục chức năng Tra cứu từ.\n\t  Nhấn phím khác để trở về màn hình danh sách chức năng.");  } while (getch() == 13);  break; |

Bảng 5.21: Xây dựng chức năng tra cứu từ

#### Chức năng thêm từ mới

Xây dựng chức năng thêm từ như sau:

|  |
| --- |
| case '2':  do  {  Console\_ShowFunction("THÊM TỪ MỚI");  printf("\n\tNhập từ mới tiếng Anh: ");  String english = String\_Input();  printf("\tNhập nghĩa tiếng Việt: ");  String vietnamese = String\_Input();  Console\_ShowFunction("THÊM TỪ MỚI");  if (strlen(english) \* strlen(vietnamese) > 0)  {  if (Dictionary\_Insert(dictionary, english, vietnamese))  {  printf("\n\tThêm từ mới thành công.\n\n");  }  else  {  printf("\n\tThêm từ mới không thành công. Từ này đã  có trong tự điển.\n\n");  }  }  else  {  printf("\n\tThêm từ mới không thành công. Từ tiếng Anh  và nghĩa tiếng Việt không được để trống.\n\n");  }  String\_Destroy(english);  String\_Destroy(vietnamese);  printf("\tNhấn phím Enter để tiếp tục chức năng Thêm từ mới.\n\t  Nhấn phím khác để trở về màn hình danh sách chức năng.");  } while (getch() == 13);  break; |

Bảng 5.22: Xây dựng chức năng thêm từ mới

#### Chức năng xoá từ

Xây dựng chức năng xoá từ như sau:

|  |
| --- |
| case '3':  do  {  Console\_ShowFunction("XOÁ TỪ");  printf("\n\tNhập từ tiếng Anh cần xoá: ");  String english = String\_Input();  Console\_ShowFunction("XOÁ TỪ");  if (strlen(english) > 0)  {  if (Dictionary\_Delete(dictionary, english))  {  printf("\n\tXoá từ thành công.\n\n");  }  else  {  printf("\n\tXoá từ không thành công. Từ này không  có trong tự điển.\n\n");  }  }  else  {  printf("\n\tXoá từ không thành công. Vui lòng nhập từ tiếng  Anh cần xoá.\n\n");  }  String\_Destroy(english);  printf("\tNhấn phím Enter để tiếp tục chức năng Xoá từ.\n\t Nhấn phím khác để trở về màn hình danh sách chức năng.");  } while (getch() == 13);  break; |

Bảng 5.23: Xây dựng chức năng xoá từ

#### Xem lịch sử tra cứu

Xây dựng chức năng xem lại lịch sử tra cứu:

|  |
| --- |
| case '4':  loopHistory = true;  CursorHistory cursor = History\_GetCursor(history);  while (loopHistory)  {  Console\_ShowBrand();  printf("\tLỊCH SỬ TRA CỨU TỪ\n");  printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");  if (cursor != NULL)  {  Console\_ShowSearchResult(History\_GetData(cursor));  printf("\tDùng 2 phím mũi tên cuộn lên và xuống để xem  lịch sử.\n\tNhấn phím khác để trở về màn hình danh sách chức năng.");  switch (getch())  {  case 224:  switch (getch())  {  case 72:  if (History\_Next(cursor) != NULL)  {  cursor = History\_Next(cursor);  }  break;  case 80:  if (History\_Previous(cursor) != NULL)  {  cursor = History\_Previous(cursor);  }  break;  }  break;  default:  loopHistory = false;  break;  }  }  else      {        printf("\tLịch sử trống\n\n");        printf("\tNhấn phím bất kì để trở về màn hình danh sách chức năng.");        getch();        loopHistory = false;      }  }    break; |

Bảng 5.24: Xây dựng chức năng xem lịch sử tra cứu

## Hướng dẫn cài đặt chương trình

Chương trình có giao diện tiếng Việt và chạy trên nền Console nhưng do Command Prompt của Windows muốn hiển thị được tiếng Việt UTF-8 thì phải điều chỉnh thông số máy rất phức tạp. Nên có cách đơn giản hơn là thay vì dùng Command Prompt có sẳn ta sử dụng phần mềm Cmder.

* Trang chủ của Cmder: <https://cmder.net/>
* Hướng dẫn cài đặt và tài liệu về Cmder: <https://github.com/cmderdev/cmder>

Sau khi cài đặt Cmder. Kế tiếp ta tải file nén chương trình Tự điển qua đường dẫn sau: <https://github.com/daomtthuan/dictionary/blob/master/install/DictionaryInstall.zip?raw=true>

Sau khi tải về ta có file DictionaryInstall.zip, giải nén file tại thư mục bất kỳ theo ý muốn, ta được một thư mục có tên Dictionary.

Sau đó mở phần mềm Cmder và dẫn tới đường dẫn nơi đã giải nén bằng lệnh:

|  |
| --- |
| cd <tên đường dẫn> |

Bảng 5.25: Cú pháp lệnh đi đến đường dẫn

Ví dụ: nơi giải nén của chương trình ở ổ đĩa D, giải nén được thư mục Dictionary, ta gõ:

|  |
| --- |
| cd D:\\Dictionary |

Bảng 5.26: Ví dụ đi đến đường dẫn chương trình

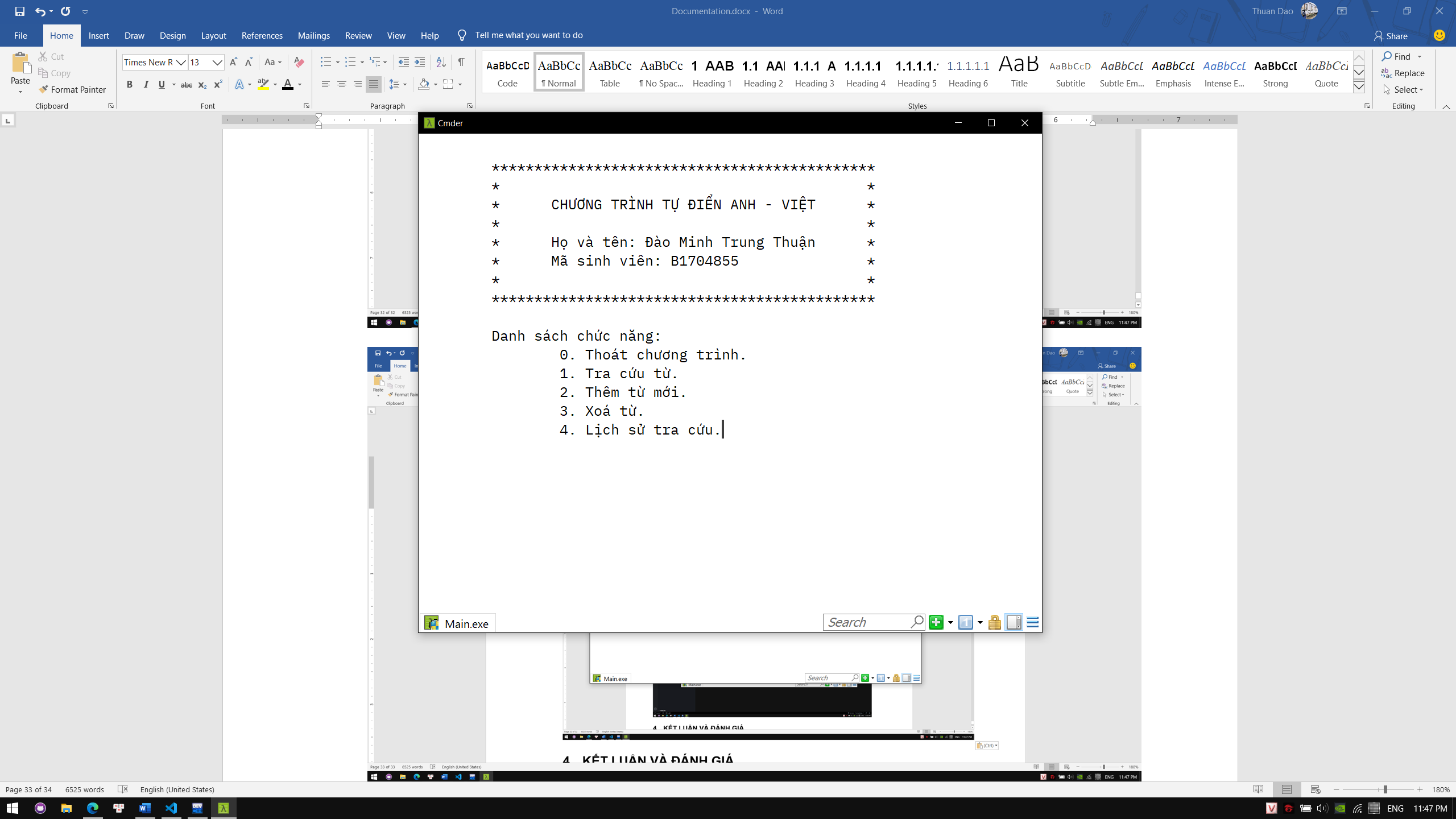
Sau đó ta gõ lệnh sau để chạy chương trình:

|  |
| --- |
| start Dictionary |

Bảng 5.27: Lệnh chạy chương trình

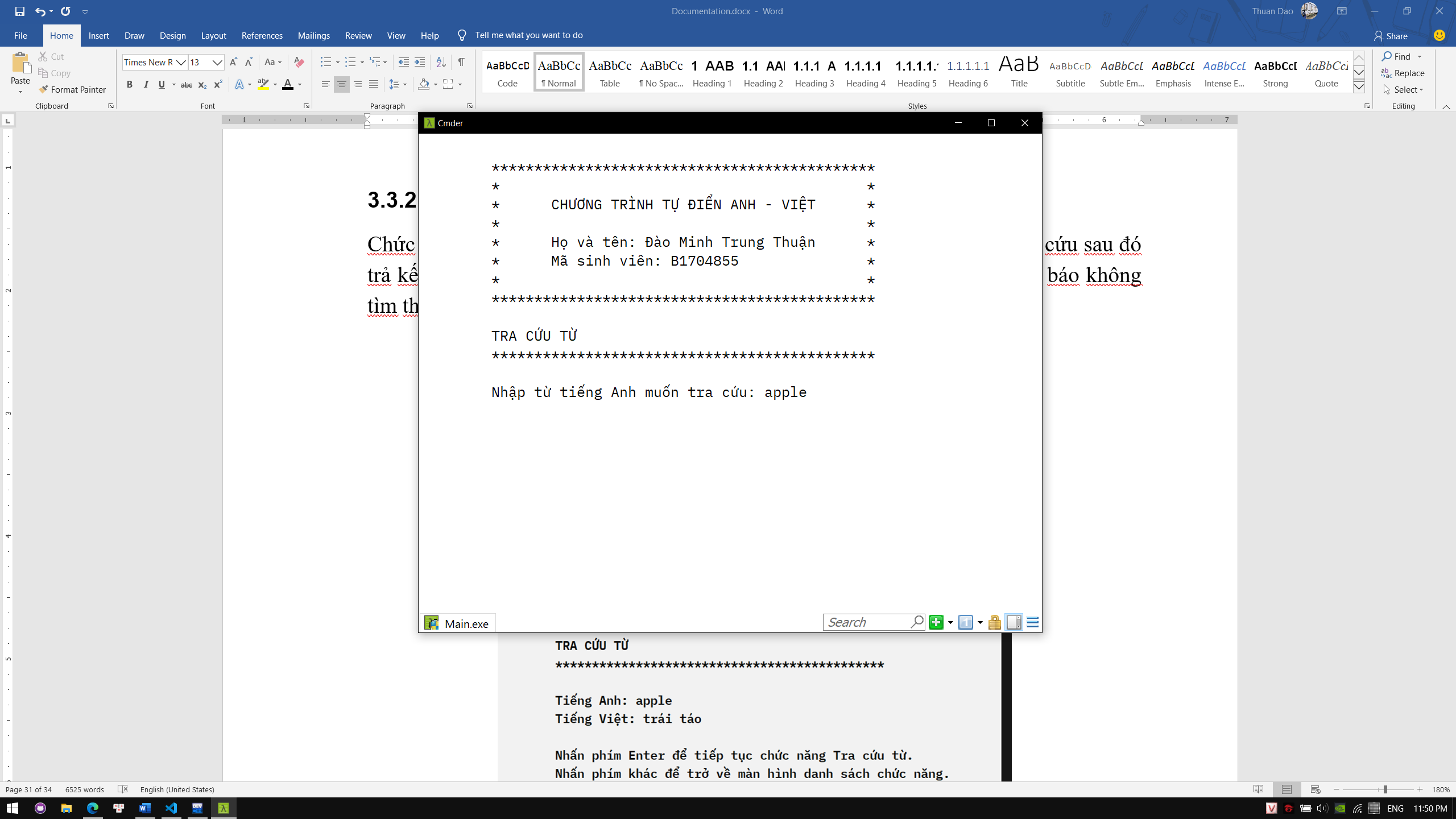
## Hướng dẫn sử dụng

Khi chạy chương trình, màn hình sẽ hiện lên danh sách chức năng, bấm phím số 0, 1, 2, 3 và 4 để lựa chọn chức năng.



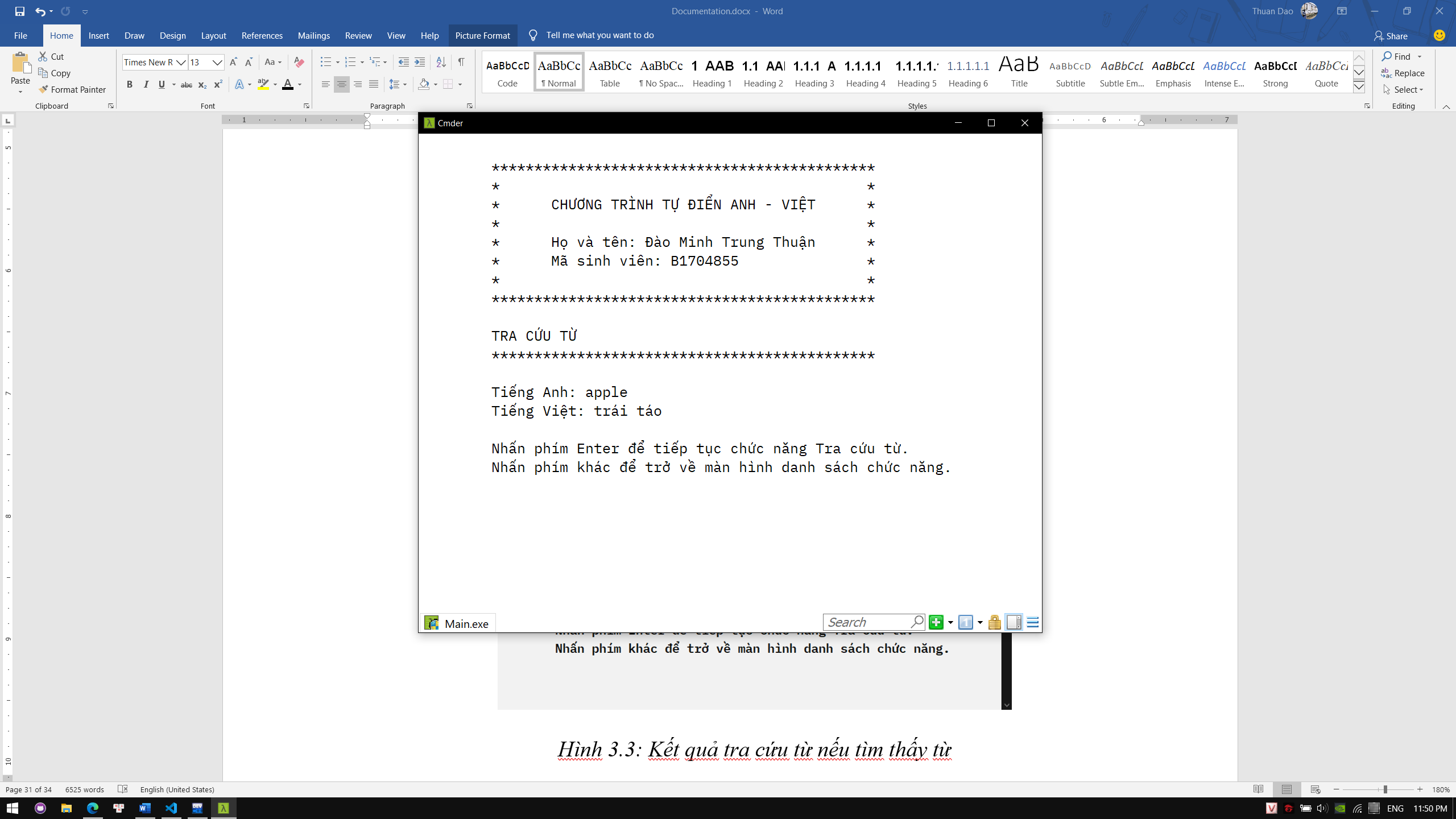
Hình 5.1: Giao diện chính của chương trình

Khi vào chức năng tra cứu từ, nhập từ tiếng Anh muốn tra cứu và nhấn Enter.



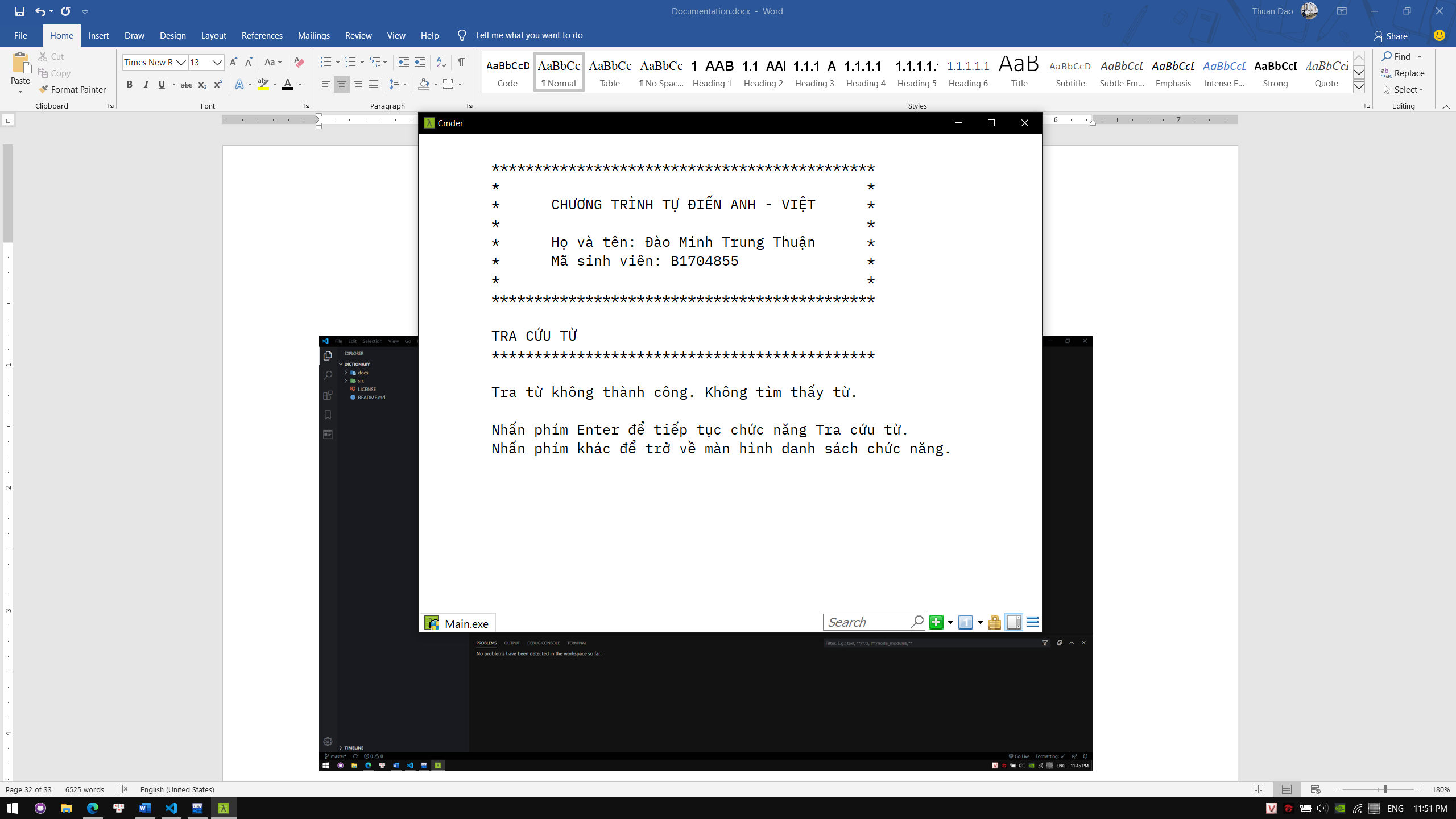
Hình 5.2: Chức năng tra cứu từ

Chương trình sẽ trả về kết quả nghĩa tiếng Việt của từ đó.



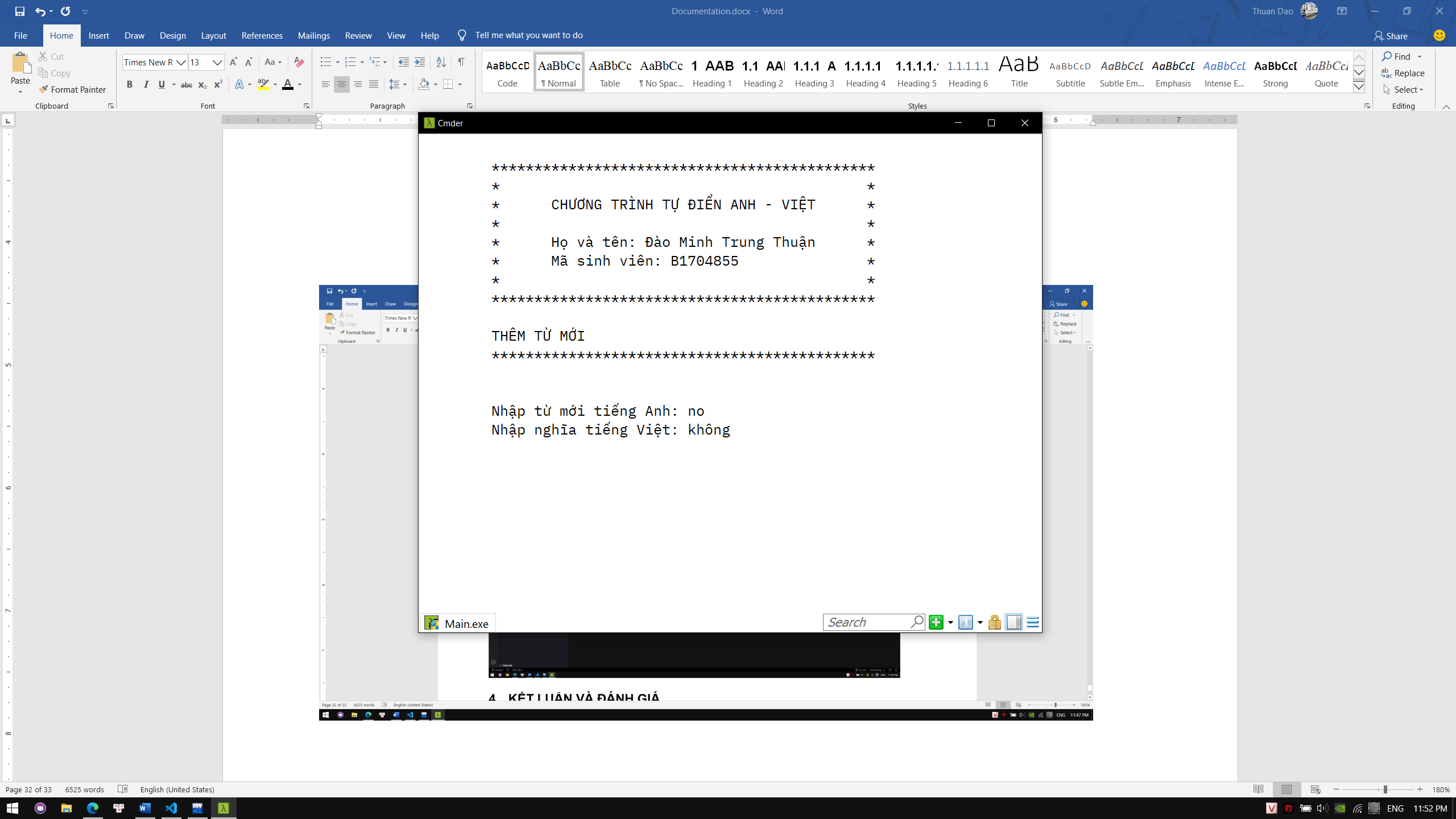
Hình 5.3: Kết quả tra cứu từ nếu tìm thấy từ

Nếu từ tra cứu không có trong Tự điển, chương trình sẽ thông báo không tìm thấy từ.



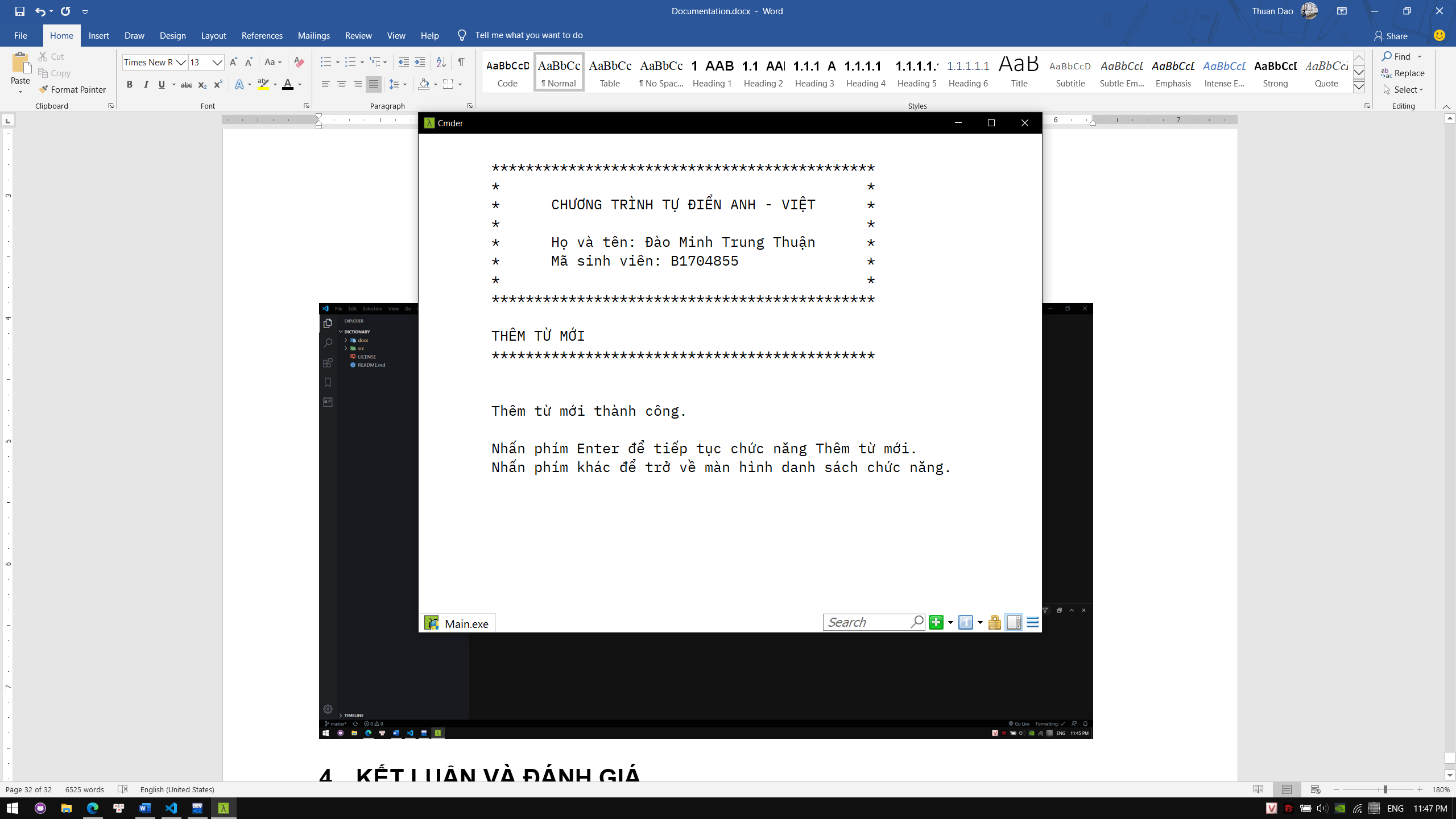
Hình 5.4: Thông báo không tìm thấy từ

Sang chức năng thêm từ mới, nhập từ tiếng Anh và nghĩa tiếng Việt của từ và nhấn Enter.



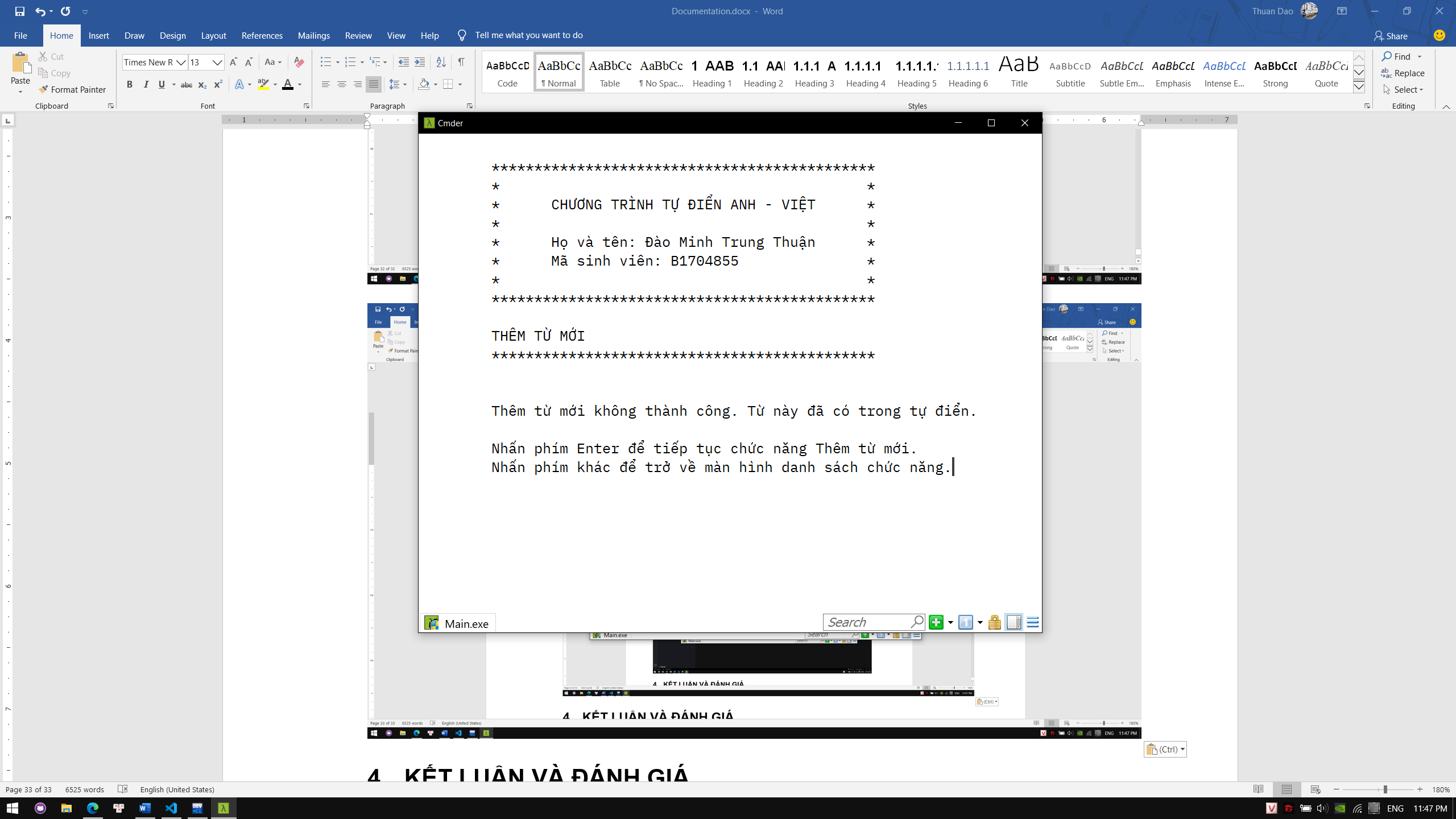
Hình 5.5: Nhập từ mới cần thêm

Nếu thêm từ mới thành, chương trình sẽ thông báo kết quả thành công.



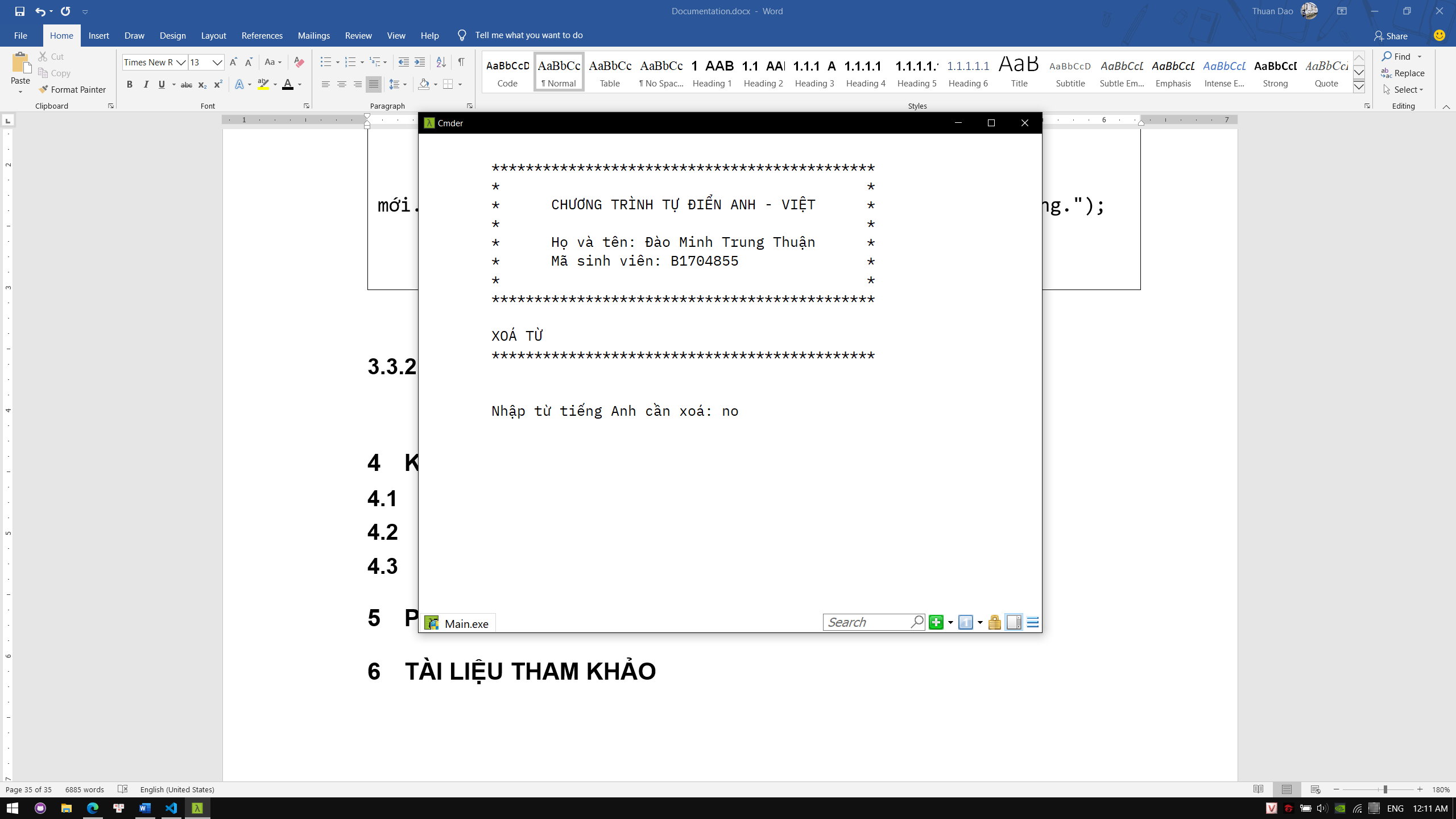
Hình 5.6: Kết quả thêm từ mới

Nếu từ mà người dùng nhập vào đã tồn tại trong Từ điển thì chương trình sẽ thông báo từ đã tồn tại và thêm không thành công. Trong trường hợp này, buộc phải xoá từ đã có trong tự điển rồi mới thêm từ mới.



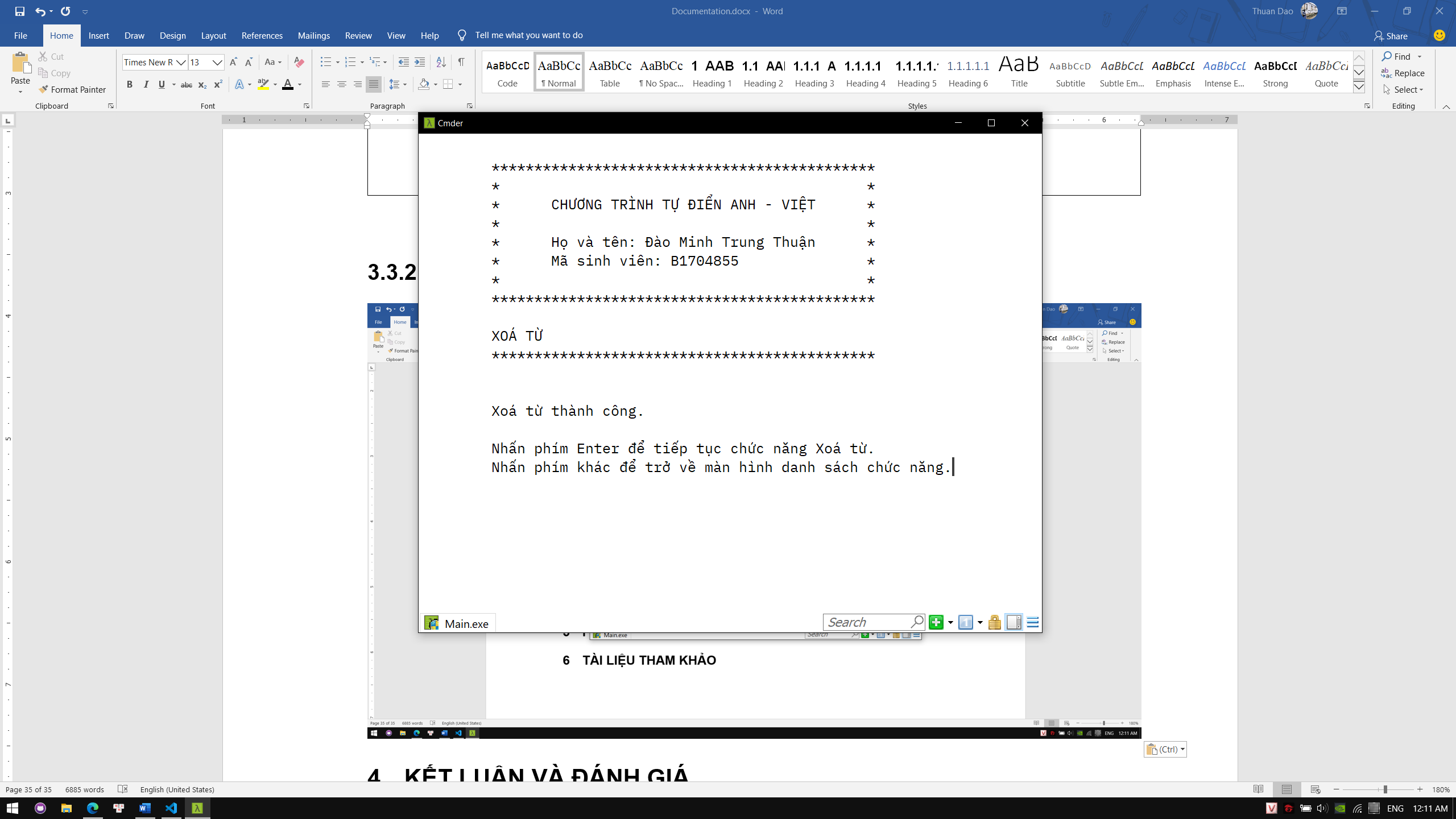
Hình 5.7: Thông báo thêm từ không thành công

Khi vào chức năng xoá từ, chỉ cần nhập từ tiếng Anh cần xoá và nhấn Enter.



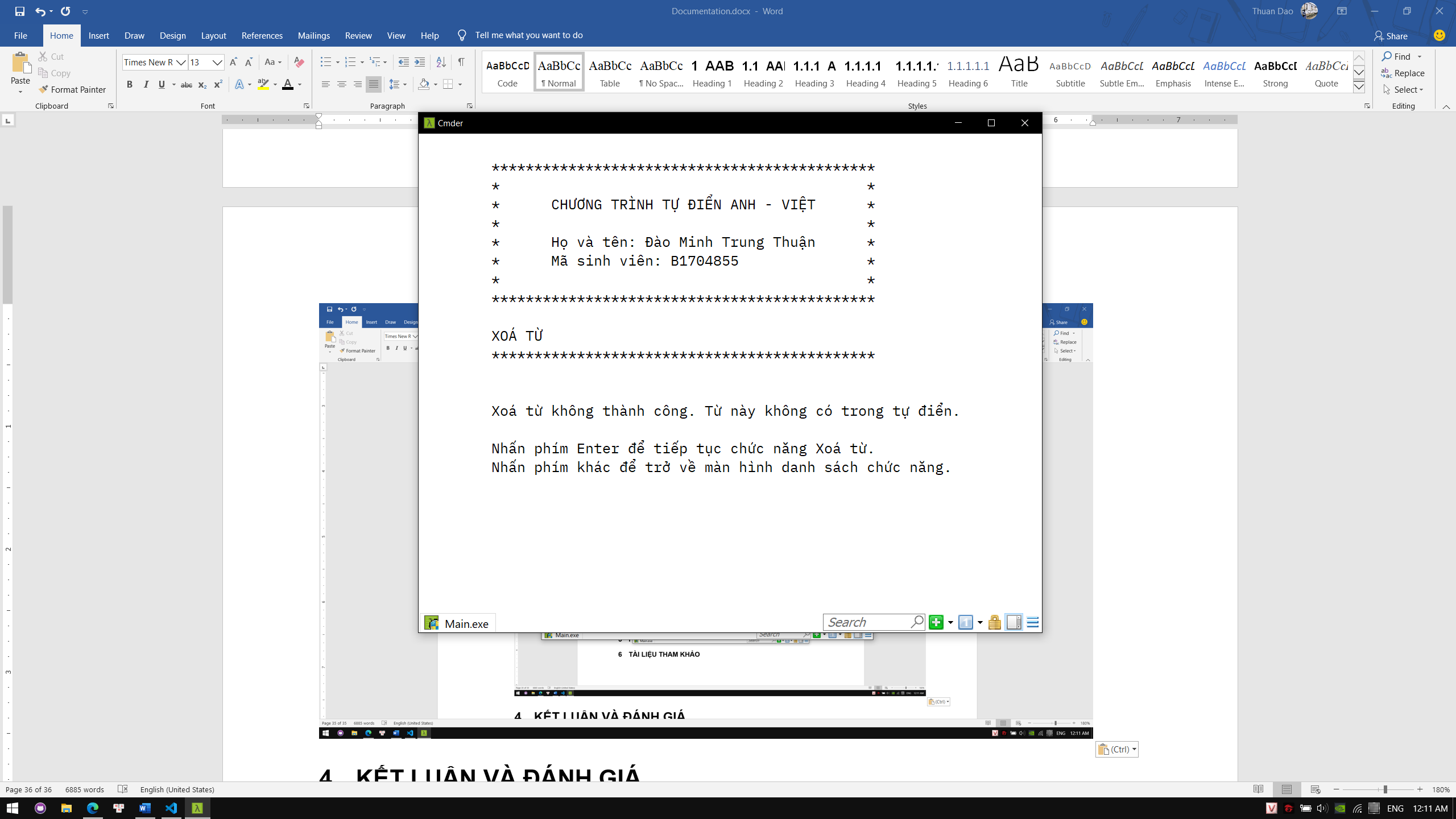
Hình 5.8: Nhập từ cần xoá

Nếu từ đó tồn tại trong Tự điển, thông báo xoá từ thành công sẽ được hiện lên.



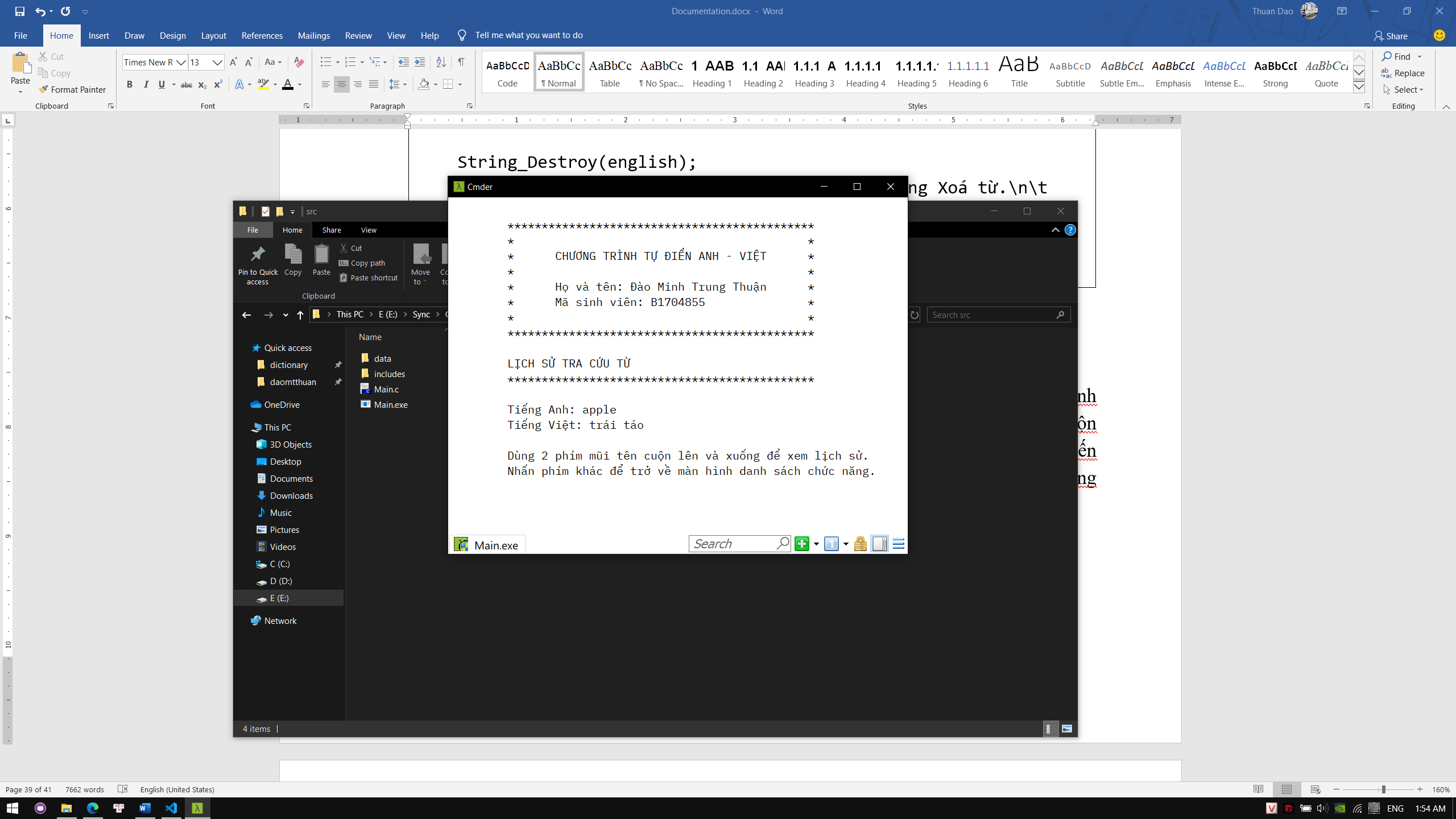
Hình 5.9: Thông báo xoá từ thành công

Nếu từ đó không có trong Tự điển, chương trình sẽ thông báo xoá không thành công.



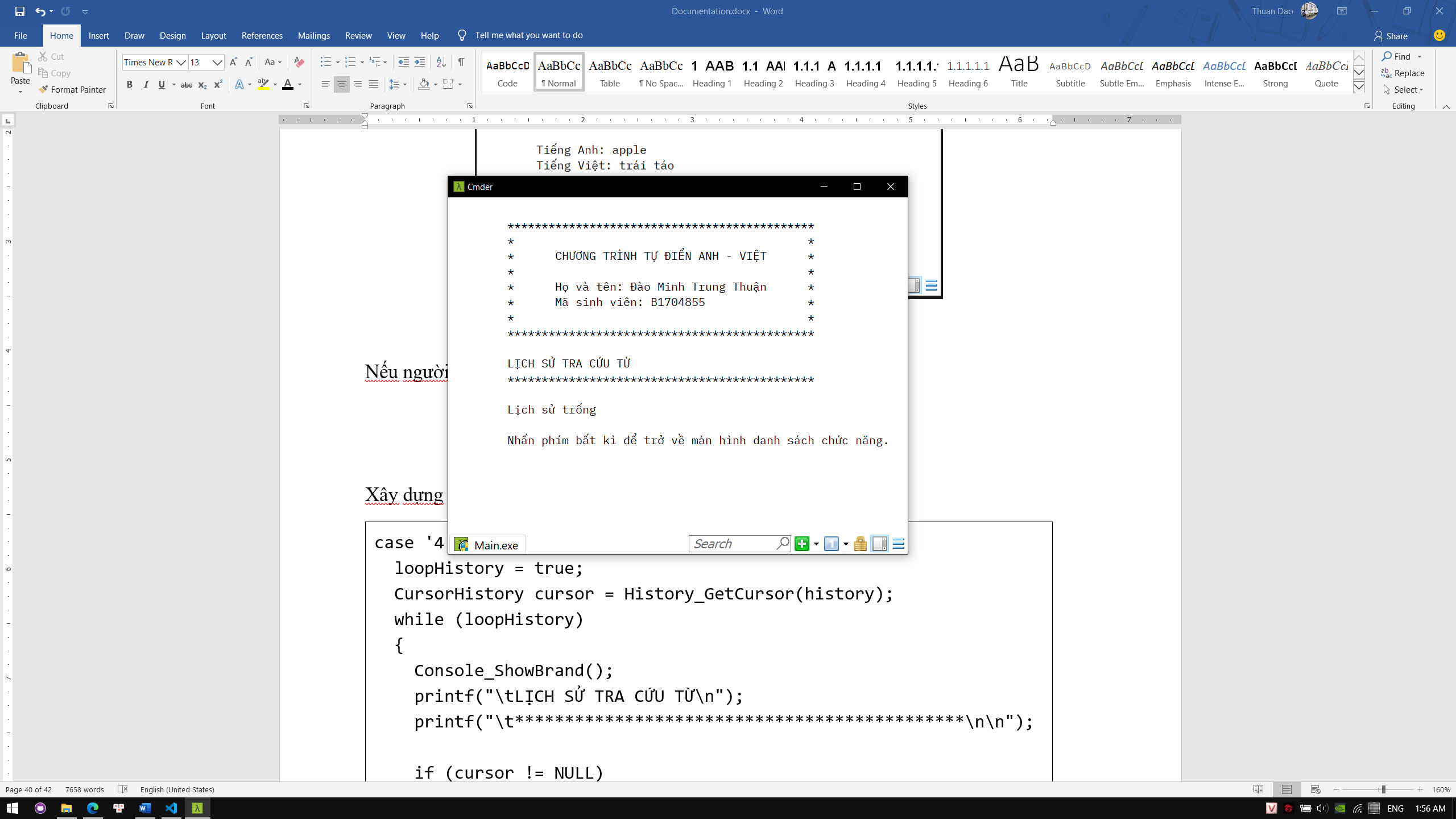
Hình 5.10: Thông báo xoá từ không thành công

Khi vào chức năng xem lại lịch sử đã tra cứu từ, sử dụng hai phím mũi tên lên và xuống để điều kiển cuộn lịch sử.



Hình 5.11: Chức năng xem lịch sử tra cứu

Nếu chưa tra cứu từ nào, lịch sử sẽ trống.



*Hình 5.12: Lịch sử trống*

# TÀI LIỆU THAM KHẢO

1. Trần Cao Đệ; Giáo trình Cấu trúc dữ liệu Data Structures; Nhà xuất bản Đại học Cần Thơ; 2012.
2. Nguyễn Văn Linh, Trần Cao Đệ, Trương Thị Thanh Tuyền, Lâm Hoài Bảo, Phan Huy Cường, Trần Ngân Bình; Giáo trình Cấu trúc dữ liệu; Nhà xuất bản Đại học Cần Thơ; 2003.
3. Bài giảng Lập trình căn bản, Nhà xuất bản Đại học Cần Thơ; Lưu hành nội bộ
4. Lê Trần Đạt; Tài liệu Khoá học C++; <https://cpp.daynhauhoc.com/>