# 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

# 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 .: Bảng kế hoạch thực hiện

# CƠ SỞ LÝ THUYẾT

## CÁC KHÁI NIỆM

### 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.

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.

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

### 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.

### 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.

#### Các phép toán trên danh sách

* **Insert(x,p,L)** xen phần tử x (kiểu ElementType) tại vị trí p (kiểu position) trong danh sách L.Tức là nếu danh sách là a1, a2, …, ap-1, ap, …, an thì sau khi xen ta có kết quả a1, a2, …, ap-1, x, ap, …, an. Nếu vị trí p không tồn tại trong danh sách thì phép toán không được xác định.
* **Locate(x,L)** thực hiện việc định vị phần tử có nội dung x đầu tiên trong danh sách L. Locate trả kết quả là vị trí (kiểu position) của phần tử x trong danh sách. Nếu x không có trong danh sách thì vị trí sau phần tử cuối cùng của danh sách được trả về, tức là **Endlist(L)**.
* **Retrieve(p,L)** lấy giá trị của phần tử ở vị trí p (kiểu position) của danh sách L; nếu vị trí p không có trong danh sách thì kết quả không xác định (có thể thông báo lỗi).
* **Delete(p,L)** chương trình con thực hiện việc xoá phần tử ở vị trí p (kiểu position) của danh sách. Nếu vị trí p không có trong danh sách thì phép toán không được định nghĩa và danh sách L sẽ không thay đổi **Next(p,L)** cho kết quả là vị trí của phần tử (kiểu position) đi sau phần tử p; nếu p là phần tử cuối cùng trong danh sách L thì **Next(p,L)** cho kết quả là **Endlist(L)**. Next không xác định nếu p không phải là vị trí của một phần tử trong danh sách.
* **Previous(p,L)** cho kết quả là vị trí của phần tử đứng trước phần tử p trong danh sách. Nếu p là phần tử đầu tiên trong danh sách thì **Previous(p,L)** không xác định. Previous cũng không xác định trong trường hợp p không phải là vị trí của phần tử nào trong danh sách.
* **First(L)** cho kết quả là vị trí của phần tử đầu tiên trong danh sách. Nếu danh sách rỗng thì **Endlist(L)** được trả về.
* **EmtyList(L)** cho kết quả TRUE nếu danh sách có rỗng, ngược lại nó cho giá trị FALSE.
* **MakeNullList(L)** khởi tạo một danh sách L rỗng.

#### 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 .: 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 .: 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 .: 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 chung. Mỗi lớp 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 .: 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 từ tiếng Anh và 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 thành phần 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++.

Ngoại 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.

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

Bảng .: 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 .: Danh sách các hàm của kiểu String

#### Từ (Word)

Kiểu Word đại diện cho từ có cấu trúc gồm hai thành phần là phần từ tiếng Anh và phần nghĩa tiếng Việt.

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

Bảng .: Định nghĩa kiểu Word

#### Phần tử Tự điển (Element)

Kiểu Phần tử cho Tự điện sẽ là kiểu Word, sử dụng thành phần chuỗi từ tiếng Anh để làm khoá đại diện cho phần tử.

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

Bảng .: 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 .: Danh sách các hàm của kiểu Element

#### 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.

|  |
| --- |
| // 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 .: 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 .: Danh sách hàm kiểu NodeLinkedList

Định nghĩa kiểu LinkedList với cấu trúc gồm hai thành phần là Head và Tail.

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

Bảng .: 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 .: Các hàm kiểu LinkedList

#### Bảng băm (Hash Table)

Thiết kế thuật toán băm bằng phương pháp chia.

|  |
| --- |
| **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 .: Kỹ thuật băm cho Bảng băm

Sử dụng Bảng băm mở cho cài đặt từ điển, cho nên ta thiết kết Node dưới dạng Node cho danh sách liên kết đơn.

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

Bảng .: 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 .: Các hàm kiểu NodeHashTable

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).

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

Bảng .: 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 .: Các Hàm trong HashTable

#### Lịch sử (History)

Sử dụng Danh sách liên kết đôi để cài đặt History. Và kiểu con trỏ để truy xuất History.

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

Bảng .: 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 .: Các hàm của kiểu History

#### Tự điển (Dictionary)

Kiểu dữ liệu Diction được cài đặt trên kiểu HashTable

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

Bảng .: 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 .: Các hàm kiểu Dictionary

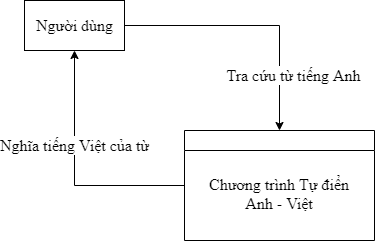
## SƠ ĐỒ CHỨC NĂNG, LƯU ĐỒ GIẢI THUẬT

### 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)

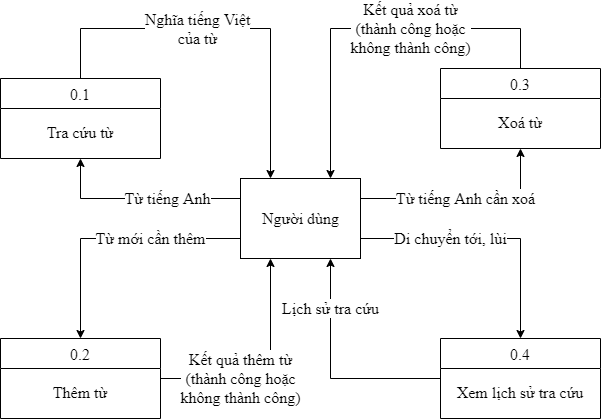
Ở đây, chức năng chính của chương trình sẽ là tra cứu từ tiếng Anh, chương trình sẽ trả kết quả về là nghĩa tiếng Việt của từ đó.



Sơ đồ .: 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.

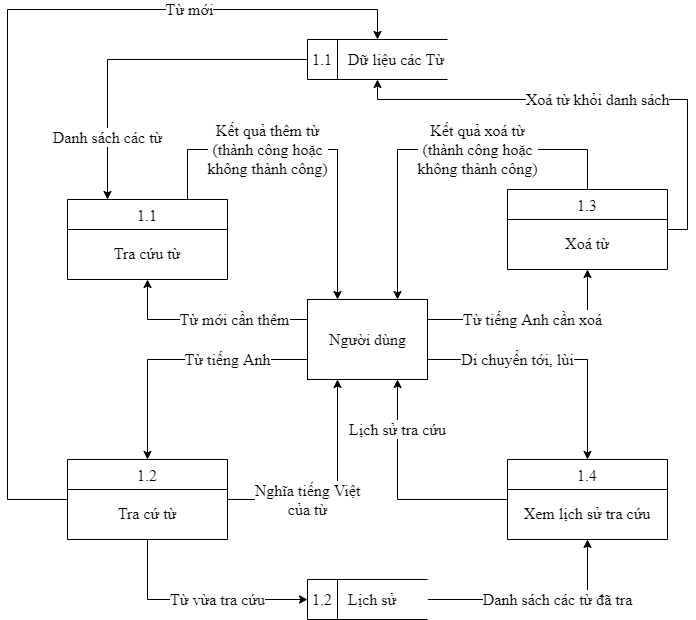


Sơ đồ .: Sơ đồ DFD mức 0

#### 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 .: Sơ đồ DFD mức 1

## CÁCH THỨC XÂY DỰNG CHƯƠNG TRÌNH

### Cấu trúc dự án

dictionary

docs

src

data

includes

Main.c

Data.txt

# KẾT LUẬN VÀ ĐÁNH GIÁ

## KẾT QUẢ ĐẠT ĐƯỢC

## HẠN CHẾ, NGUYÊN NHÂN

## HƯỚNG PHÁT TRIỂN ĐỀ TÀI

# PHẦN PHỤC LỤC

# TÀI LIỆU THAM KHẢO