

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM**

**KHOA ĐÀO TẠO CHẤT LƯỢNG CAO**

**🕯✡🕮🕮✡🕯**

****

**MÔN HỌC: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

**Đồ án cuối kì STACK**

**GVHD: Thầy Nguyễn Thiên Bảo**

**SVTT MSSV**

**Thiều Văn Hoàng 18110114**

**Đặng Minh Khang 18110130**

**Trần Nhật Thành 18110199**

**Lê Phước Hưng 18110297**

**Tp. Hồ Chí Minh, tháng 10 năm 2018**

**NHẬN XÉT CỦA GIÁO VIÊN**

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

Ngày………..Tháng……….Năm……

**Giáo Viên**

(Ký và ghi rõ họ tên)

**Giáo Viên**

**(Ký và ghi**

**Mục lục**

[**DANH MỤC CÁC HÌNH** 1](#_Toc27187982)

[**GIỚI THIỆU VỀ BỘ MÔN** 2](#_Toc27187983)

[**I. TỔNG QUAN VỀ LÝ THUYẾT** 6](#_Toc27187984)

[**1.** **Stack là gì?** 6](#_Toc27187985)

[**2.** **Các thao tác cơ bản trên stack** 7](#_Toc27187986)

[**3.** **Biểu diễn Stack dùng mảng** 7](#_Toc27187987)

[**3.1. Khởi tạo stack dùng mảng** 7](#_Toc27187988)

[**3.2. Thêm một phần tử vào Stack (Push)** 8](#_Toc27187989)

[**3.3. Lấy thông tin phần tử đầu ở đỉnh stack (Top))** 8](#_Toc27187990)

[**3.4. Xóa phần tử đầu từ đỉnh stack (Pop)** 9](#_Toc27187991)

[**4.** **Biểu diễn Stack dung danh sách liên kết** 10](#_Toc27187992)

[**4.1. Khai báo cấu trúc một node** 10](#_Toc27187993)

[**4.2. Khai báo cấu trúc Stack** 10](#_Toc27187994)

[**4.3. Tạo một stack rỗng (Init)** 11](#_Toc27187995)

[**4.4. Kiểm tra stack rỗng (IsEmpty)** 11](#_Toc27187996)

[**4.5. Thêm một phần tử vào stack (Push)** 11](#_Toc27187997)

[**4.6. Xóa một phần tử từ đỉnh stack (Pop)** 12](#_Toc27187998)

[**4.7. Lấy thông tin phần tử đầu của stack (Top)** 12](#_Toc27187999)

[**5.** **Ứng dụng** 13](#_Toc27188000)

[**II. GIAO DIỆN ĐỒ HỌA (DEMO)** 14](#_Toc27188001)

[**1. Giao diện** 14](#_Toc27188002)

[**2. Phần Code** 15](#_Toc27188003)

[**III. ỨNG DỤNG MINH HỌA STACK** 21](#_Toc27188004)

[**1. Giao diện** 21](#_Toc27188005)

[**2. Phần Code** 22](#_Toc27188006)

[**2.1. Khởi tạo Stack cho đĩa** 22](#_Toc27188007)

[**2.2. Các thao tác** 22](#_Toc27188008)

[**IV. PHÂN CÔNG CÔNG VIỆC** 25](#_Toc27188009)

[**Tài liệu kham khảo** 26](#_Toc27188010)

# **DANH MỤC CÁC HÌNH**

[Hình 1: Cấu trúc dữ liệu Ngăn xếp(Stack) 8](#_Toc27183371)

[Hình 2:Các thao tác cơ bản khỏi tạo Stack bằng mảng 9](#_Toc27183372)

[Hình 3:Các thao tác cơ bản khỏi tạo Stack bằng mảng 9](#_Toc27183373)

[Hình 4:Thêm một phần tử vào Stack 10](#_Toc27183374)

[Hình 5: Lấy thông tin phần tử đầu ở đỉnh stack 10](#_Toc27183375)

[Hình 6: Xóa phần tử đầu từ đỉnh Stack 11](#_Toc27183376)

[Hình 7 : Khai báo node 11](#_Toc27183377)

[Hình 8: Khai báo cấu trúc Stack 12](#_Toc27183378)

[Hình 9: Tạo Stack rỗng 12](#_Toc27183379)

[Hình 10: Kiểm tra Stack rỗng 13](#_Toc27183380)

[Hình 11: Thêm phần tử vào Stack 13](#_Toc27183381)

[Hình 12: Xóa phần tử đỉnh Stack 14](#_Toc27183382)

[Hình 13: Lấy thông tin phần tử đầu của stack 14](#_Toc27183383)

[Hình 14: Giao diện Demo Stack 16](#_Toc27183384)

[Hình 15:Khởi tạo Class stack 17](#_Toc27183385)

[Hình 16: Code khởi tạo node 18](#_Toc27183386)

[Hình 17:Code btnPush\_Click 19](#_Toc27183387)

[Hình 18: Code btnPeak\_Click 19](#_Toc27183388)

[Hình 19:Code btnPop\_Click 19](#_Toc27183389)

[Hình 20:Code vẽ mô phỏng 1 node 20](#_Toc27183390)

[Hình 23:code vẽ Pointer 21](#_Toc27183391)

[Hình 24:Code timer của pointer 21](#_Toc27183392)

[Hình 25:Code vẽ top 22](#_Toc27183393)

[Hình 26:Code timer của top 22](#_Toc27183394)

[Hình 27:Code UpdateTop 22](#_Toc27183395)

[Hình 28: Giao diện game tháp Hà Nội 23](#_Toc27183396)

[Hình 29: Khởi tạo Stack cho 3 đĩa 23](#_Toc27183397)

[Hình 30:Khai báo và contructor 25](#_Toc27183398)

[Hình 31:Event click vào các cột 26](#_Toc27183399)

[Hình 32:Event xử lý di chuyển khi click ProcessMovingDisk 26](#_Toc27183400)

[Hình 33:xử lý trao đỗi đĩa giữa các cột 27](#_Toc27183401)

[Hình 34:Event click button “Chơi” 27](#_Toc27183402)

# **GIỚI THIỆU VỀ BỘ MÔN**

Trong [khoa học máy tính](https://vi.wikipedia.org/wiki/Khoa_h%E1%BB%8Dc_m%C3%A1y_t%C3%ADnh), **cấu trúc dữ liệu** là một cách lưu [dữ liệu](https://vi.wikipedia.org/wiki/D%E1%BB%AF_li%E1%BB%87u) trong [máy tính](https://vi.wikipedia.org/wiki/M%C3%A1y_t%C3%ADnh) sao cho nó có thể được sử dụng một cách hiệu quả.

Trong thiết kế nhiều loại chương trình, việc chọn cấu trúc dữ liệu là vấn đề quan trọng. Kinh nghiệm trong việc xây dựng các hệ thống lớn cho thấy khó khăn của việc triển khai chương trình, chất lượng và hiệu năng của kết quả cuối cùng phụ thuộc rất nhiều vào việc chọn cấu trúc dữ liệu tốt nhất.

Mỗi loại cấu trúc dữ liệu phù hợp với một vài loại ứng dụng khác nhau, một số cấu trúc dữ liệu dành cho những công việc đặc biệt. Ví dụ, các [B-tree](https://vi.wikipedia.org/wiki/B-tree) đặc biệt phù hợp trong việc thiết kế cơ sở dữ liệu. Sau khi cấu trúc dữ liệu được chọn, người ta thường dễ nhận thấy [thuật toán](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n) cần sử dụng. Đôi khi trình tự công việc diễn ra theo thứ tự ngược lại: cấu trúc dữ liệu được chọn do những bài toán quan trọng nhất định có thuật toán chạy tốt nhất với một số cấu trúc dữ liệu cụ thể. Trong cả hai trường hợp, việc lựa chọn cấu trúc dữ liệu là rất quan trọng.

**Thuật toán**, còn gọi là **giải thuật**, là một [tập hợp hữu hạn](https://vi.wikipedia.org/wiki/T%E1%BA%ADp_h%E1%BB%A3p#L%E1%BB%B1c_l%C6%B0%E1%BB%A3ng_c%E1%BB%A7a_t%E1%BA%ADp_h%E1%BB%A3p_-_H%E1%BB%AFu_h%E1%BA%A1n_v%C3%A0_v%C3%B4_h%E1%BA%A1n) hay một dãy các qui tắc chặt chẽ của các chỉ thị, phương cách hay 1 trình tự các thao tác trên một đối tượng cụ thể được xác định và định nghĩa rõ ràng cho việc hoàn tất một số sự việc từ một trạng thái ban đầu cho trước; khi các chỉ thị này được áp dụng triệt để thì sẽ dẫn đến kết quả sau cùng như đã dự đoán trước.

**Cấu trúc dữ liệu (Data Structure) là gì ?**

Cấu trúc dữ liệu là cách lưu trữ, tổ chức dữ liệu có thứ tự, có hệ thống để dữ liệu có thể được sử dụng một cách hiệu quả.

Dưới đây là hai khái niệm nền tảng hình thành nên một cấu trúc dữ liệu:

* **Interface**: Mỗi cấu trúc dữ liệu có một Interface. Interface biểu diễn một tập hợp các phép tính mà một cấu trúc dữ liệu hỗ trợ. Một Interface chỉ cung cấp danh sách các phép tính được hỗ trợ, các loại tham số mà chúng có thể chấp nhận và kiểu trả về của các phép tính này.
* **Implementation** (có thể hiểu là sự triển khai): Cung cấp sự biểu diễn nội bộ của một cấu trúc dữ liệu. Implementation cũng cung cấp phần định nghĩa của giải thuật được sử dụng trong các phép tính của cấu trúc dữ liệu.

**Đặc điểm của một Cấu trúc dữ liệu**

* **Chính xác**: Sự triển khai của Cấu trúc dữ liệu nên triển khai Interface của nó một cách chính xác.
* **Độ phức tạp về thời gian (Time Complexity)**: Thời gian chạy hoặc thời gian thực thi của các phép tính của cấu trúc dữ liệu phải là nhỏ nhất có thể.
* **Độ phức tạp về bộ nhớ (Space Complexity)**: Sự sử dụng bộ nhớ của mỗi phép tính của cấu trúc dữ liệu nên là nhỏ nhất có thể.

**Tại sao Cấu trúc dữ liệu là cần thiết ?**

Ngày nay, các ứng dụng ngày càng phức tạp và lượng dữ liệu ngày càng lớn với nhiều kiểu đa dạng. Việc này làm xuất hiện 3 vấn đề lớn mà mỗi lập trình viên phải đối mặt:

* **Tìm kiếm dữ liệu**: Giả sử có 1 triệu hàng hóa được lưu giữ vào trong kho hàng hóa. Và giả sử có một ứng dụng cần để tìm kiếm một hàng hóa. Thì mỗi khi thực hiện tìm kiếm, ứng dụng này sẽ phải tìm kiếm 1 hàng hóa trong 1 triệu hàng hóa. Khi dữ liệu tăng lên thì việc tìm kiếm sẽ càng trở lên chậm và tốn kém hơn.
* **Tốc độ bộ vi xử lý**: Mặc dù bộ vi xử lý có tốc độ rất cao, tuy nhiên nó cũng có giới hạn và khi lượng dữ liệu lên tới hàng tỉ bản ghi thì tốc độ xử lý cũng sẽ không còn được nhanh nữa.
* **Đa yêu cầu**: Khi hàng nghìn người dùng cùng thực hiện một phép tính tìm kiếm trên một Web Server thì cho dù Web Server đó có nhanh đến mấy thì việc phải xử lý hàng nghìn phép tính cùng một lúc là thực sự rất khó.

Để xử lý các vấn đề trên, các cấu trúc dữ liệu là một giải pháp tuyệt vời. Dữ liệu có thể được tổ chức trong cấu trúc dữ liệu theo một cách để khi thực hiện tìm kiếm một phần tử nào đó thì dữ liệu yêu cầu sẽ được tìm thấy ngay lập tức.

**Độ phức tạp thời gian thực thi trong cấu trúc dữ liệu và giải thuật**

Có 3 trường hợp thường được sử dụng để so sánh thời gian thực thi của các cấu trúc dữ liệu khác nhau:

* **Trường hợp xấu nhất (Worst Case)**: là tình huống mà một phép tính của cấu trúc dữ liệu nào đó tốn thời gian tối đa (thời gian dài nhất). Ví dụ với ba số 1, 2, 3 thì nếu sắp xếp theo thứ tự giảm dần thì thời gian thực thi sẽ là dài nhất (và đây là trường hợp xấu nhất); còn nếu sắp xếp theo thứ tự tăng dần thì thời gian thực thi sẽ là ngắn nhất (và đây là trường hợp tốt nhất).
* **Trường hợp trung bình (Average Case)**: miêu tả thời gian thực thi trung bình một phép tính của một cấu trúc dữ liệu.
* **Trường hợp tốt nhất (Best Case)**: là tình huống mà thời gian thực thi một phép tính của một cấu trúc dữ liệu là ít nhất. Ví dụ như trên.

**Thuật ngữ cơ bản trong Cấu trúc dữ liệu**

* **Dữ liệu**: Dữ liệu là các giá trị hoặc là tập hợp các giá trị.
* **Phần tử dữ liệu**: Phần tử dữ liệu là một đơn vị đơn lẻ của giá trị.
* **Các phần tử nhóm**: Phần tử dữ liệu mà được chia thành các phần tử con thì được gọi là các phần tử nhóm.
* **Các phần tử cơ bản**: Phần tử dữ liệu mà không thể bị chia nhỏ thành các phần tử con thì gọi là các phần tử cơ bản.
* **Thuộc tính và Thực thể**: Một thực thể là cái mà chứa một vài thuộc tính nào đó, và các thuộc tính này có thể được gán các giá trị.
* **Tập hợp thực thể**: Các thực thể mà có các thuộc tính tương tự nhau thì cấu thành một tập hợp thực thể.
* **Trường**: Trường là một đơn vị thông tin cơ bản biểu diễn một thuộc tính của một thực thể.
* **Bản ghi**: Bản ghi là một tập hợp các giá trị trường của một thực thể đã cho.
* **File**: Là một tập hợp các bản ghi của các thực thể trong một tập hợp thực thể đã cho.

**Giải thuật là gì ?**

Giải thuật (hay còn gọi là thuật toán - tiếng Anh là **Algorithms**) là một tập hợp hữu hạn các chỉ thị để được thực thi theo một thứ tự nào đó để thu được kết quả mong muốn. Nói chung thì giải thuật là độc lập với các ngôn ngữ lập trình, tức là một giải thuật có thể được triển khai trong nhiều ngôn ngữ lập trình khác nhau.

Xuất phát từ quan điểm của cấu trúc dữ liệu, dưới đây là một số giải thuật quan trọng:

* **Giải thuật Tìm kiếm**: Giải thuật để tìm kiếm một phần tử trong một cấu trúc dữ liệu.
* **Giải thuật Sắp xếp**: Giải thuật để sắp xếp các phần tử theo thứ tự nào đó.
* **Giải thuật Chèn**: Giải thuật để chèn phần từ vào trong một cấu trúc dữ liệu.
* **Giải thuật Cập nhật**: Giải thuật để cập nhật (hay update) một phần tử đã tồn tại trong một cấu trúc dữ liệu.
* **Giải thuật Xóa**: Giải thuật để xóa một phần tử đang tồn tại từ một cấu trúc dữ liệu.

**Đặc điểm của giải thuật**

Không phải tất cả các thủ tục có thể được gọi là một giải thuật. Một giải thuật nên có các đặc điểm sau:

* **Tính xác định**: Giải thuật nên rõ ràng và không mơ hồ. Mỗi một giai đoạn (hay mỗi bước) nên rõ ràng và chỉ mang một mục đích nhất định.
* **Dữ liệu đầu vào xác định**: Một giải thuật nên có 0 hoặc nhiều hơn dữ liệu đầu vào đã xác định.
* **Kết quả đầu ra**: Một giải thuật nên có một hoặc nhiều dữ liệu đầu ra đã xác định, và nên kết nối với kiểu kết quả bạn mong muốn.
* **Tính dừng**: Các giải thuật phải kết thúc sau một số hữu hạn các bước.
* **Tính hiệu quả**: Một giải thuật nên là có thể thi hành được với các nguồn có sẵn, tức là có khả năng giải quyết hiệu quả vấn đề trong điều kiện thời gian và tài nguyên cho phép.
* **Tính phổ biến**: Một giải thuật có tính phổ biến nếu giải thuật này có thể giải quyết được một lớp các vấn đề tương tự.
* **Độc lập**: Một giải thuật nên có các chỉ thị độc lập với bất kỳ phần code lập trình nào.

**Độ phức tạp giải thuật (Algorithm Complexity)**

Về bản chất, độ phức tạp giải thuật là một hàm ước lượng (có thể không chính xác) số phép tính mà giải thuật cần thực hiện (từ đó dễ dàng suy ra thời gian thực hiện của giải thuật) đối với bộ dữ liệu đầu vào (Input) có kích thước **n**. Trong đó, n có thể là số phần tử của mảng trong trường hợp bài toán sắp xếp hoặc tìm kiếm, hoặc có thể là độ lớn của số trong bài toán kiểm tra số nguyên tố, …

Giả sử X là một giải thuật và n là kích cỡ của dữ liệu đầu vào. Thời gian và lượng bộ nhớ được sử dụng bởi giải thuật X là hai nhân tố chính quyết định hiệu quả của giải thuật X:

* **Nhân tố thời gian**: Thời gian được đánh giá bằng việc tính số phép tính chính (chẳng hạn như các phép so sánh trong thuật toán sắp xếp).
* **Nhân tố bộ nhớ**: Lượng bộ nhớ được đánh giá bằng việc tính lượng bộ nhớ tối đa mà giải thuật cần sử dụng.

Độ phức tạp của một giải thuật (một hàm f(n)) cung cấp mối quan hệ giữa thời gian chạy và/hoặc lượng bộ nhớ cần được sử dụng bởi giải thuật.

**Độ phức tạp bộ nhớ (Space complexity) trong phân tích giải thuật**

Nhân tố bộ nhớ của một giải thuật biểu diễn lượng bộ nhớ mà một giải thuật cần dùng trong vòng đời của giải thuật. Lượng bộ nhớ (giả sử gọi là **S(P)**) mà một giải thuật cần sử dụng là tổng của hai thành phần sau:

* Phần cố định (giả sử gọi là **C**) là lượng bộ nhớ cần thiết để lưu giữ dữ liệu và các biến nào đó (phần này độc lập với kích cỡ của vấn đề). Ví dụ: các biến và các hằng đơn giản, …
* Phần biến đổi (giả sử gọi là **SP(I)**) là lượng bộ nhớ cần thiết bởi các biến, có kích cỡ phụ thuộc vào kích cỡ của vấn đề. Ví dụ: cấp phát bộ nhớ động, cấp phát bộ nhớ đệ qui, …

**Cách viết một giải thuật ?**

Bạn đừng tìm, bởi vì sẽ không có bất kỳ tiêu chuẩn nào cho trước để viết các giải thuật. Như bạn đã biết, các ngôn ngữ lập trình đều có các vòng lặp (**do, for, while**) và các lệnh điều khiển luồng (**if-else**), … Bạn có thể sử dụng những lệnh này để viết một giải thuật.

Chúng ta viết các giải thuật theo cách thức là theo từng bước một. Viết giải thuật là một tiến trình và được thực thi sau khi bạn đã định vị rõ ràng vấn đề cần giải quyết. Từ việc định vị vấn đề, chúng ta sẽ thiết kế ra giải pháp để giải quyết vấn đề đó và sau đó là viết giải thuật.

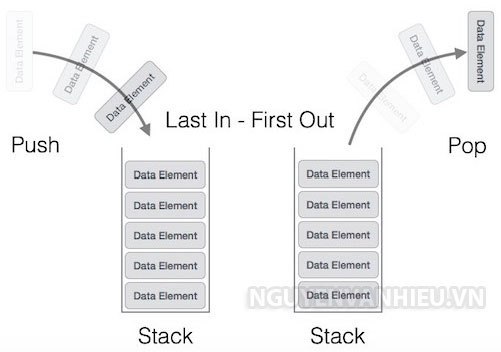
**I. TỔNG QUAN VỀ LÝ THUYẾT**

1. **Stack là gì?**

Trong khoa học máy tính, một ngăn xếp (còn gọi là bộ xếp chồng, tiếng Anh: Stack) là một cấu trúc dữ liệu trừu tượng hoạt động theo nguyên lý “vào sau ra trước” (**L**ast **I**n **F**irst **O**ut (LIFO). Tức là, phần tử cuối cùng được chèn vào ngăn xếp sẽ là phần tử đầu tiên được lấy ra khỏi ngăn xếp.

Một ví dụ trực quan, bạn có một chồng sách và bạn để nó trong một cái hộp như hình phía dưới. Giả sử hộp này vừa khít các cuốn sách. Khi đó, bạn có các thao tác:

* Thêm một cuốn sách vào hộp(push của stack)
* Lấy một cuốn sách khỏi hộp, bạn chỉ lấy được thằng trên cùng(pop của stack)

[](https://i1.wp.com/nguyenvanhieu.vn/wp-content/uploads/2018/12/ngan-xep.jpg?ssl=1)

Hình 1: Cấu trúc dữ liệu Ngăn xếp(Stack)

1. **Các thao tác cơ bản trên stack**

Stack là một CTDL trừu tượng (ADT) tuyến tình hỗ trợ hai thao tác chính:

Push(x): Thêm đối tượng x vào đầu stack

Pop(): Lấy đối tượng ở đầu stack ra khỏi stack và trả về giá trị của nó. Nếu stack rỗng thì lỗi sẽ xảy ra.

Ngoài ra, stack cũng hỗ trợ một số thao tác khác:

isEmpty(): Kiểm tra xem stack có rỗng không

Top(): Trả về giá trị của phần tử nằm ở đầu stack mà không hủy nó khỏi stack. Nếu stack rỗng thì lỗi sẽ xảy ra.

**Size():** Lấy số lượng phần tử stack đang có.

**Web designĐặc điểm**

Các thao tác thêm, trích và hủy một phần tử chỉ được thực hiện ở cùng một phía của stack do đó hoạt đồng của stack được thực hiện thao nguyên tác LIFO(Last In First Out – vào sau ra trước). Để biểu diễn stack, ta có thể dung mảng một chiều hoặc dùng danh sách liên kết.

1. **Biểu diễn Stack dùng mảng**

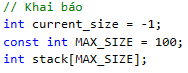
### **3.1. Khởi tạo stack dùng mảng**

- Ta có thể tạo một stack bằng cách khai báo một mảng một chiều với kíh thước tối đa là N (VD: N =1000).

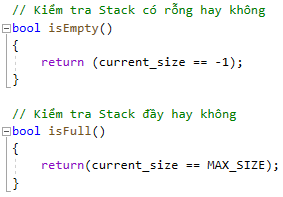
- Như vậy stack có thể chứa tối đa N phần tử đánh số từ 0 đến (N-1). Phần tử nằm ở đầu stack sẽ có chỉ số t (lúc đó trong stack đang chứa t+1 phần tử).

- Để khai báo một stack, ta cần một mảng một chiều S, biến nguyên t cho biết chỉ số của đầu stack và hằng số N cho biết kích thước tối đa của stack.

Các thao tác cơ bản:



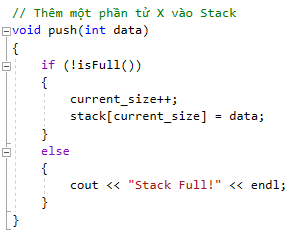
Hình 2:Các thao tác cơ bản khỏi tạo Stack bằng mảng



Hình 3:Các thao tác cơ bản khỏi tạo Stack bằng mảng

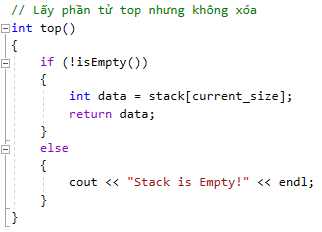
### **3.2. Thêm một phần tử vào Stack (Push)**

Chúng ta sẽ chỉ có thể push(thêm phần tử) vào đỉnh stack khi stack chưa đầy. Nếu stack đầy, chúng ta sẽ đưa ra thông báo và không thực hiện push. Ngược lại, ta sẽ tăng top lên một đơn vị và gán giá trị cho phần tử tại chỉ số top.



Hình 4:Thêm một phần tử vào Stack

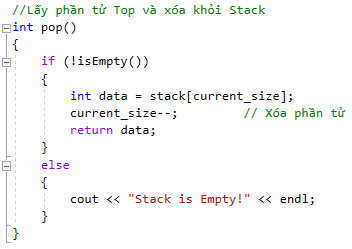
### **3.3. Lấy thông tin phần tử đầu ở đỉnh stack (Top))**

Để lấy giá trị phần tử ở đỉnh stack, ta có thao tác rất đơn giản:

Hình 5: Lấy thông tin phần tử đầu ở đỉnh stack

### **3.4. Xóa phần tử đầu từ đỉnh stack (Pop)**

Chúng ta sẽ chỉ có thể pop(xóa phần tử) khỏi đỉnh stack khi stack không trống. Nếu stack trống, chúng ta sẽ đưa ra thông báo và không thực hiện pop. Ngược lại, ta sẽ giảm giá trị top đi một đơn vị.

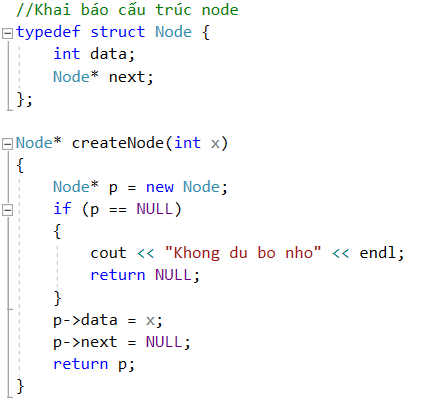


Hình 6: Xóa phần tử đầu từ đỉnh Stack

1. **Biểu diễn Stack dung danh sách liên kết**

### **4.1. Khai báo cấu trúc một node**

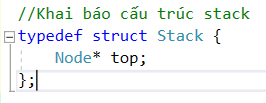
Ý tưởng: ta tạo struct gồm 2 thành phần gồm data ( dữ liệu ) và một con trỏ next. Sau đó ta khỏi tạo một node gồm dữ liệu và con trỏ tới NULL (do ban đầu chỉ có 1 node nên ta trỏ tới NULL).



Hình 7 : Khai báo node

### **4.2. Khai báo cấu trúc Stack**

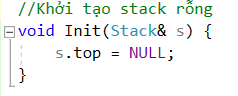
Nó là danh sách liên kết nhưng chỉ cho phép thao tác ở 1 đầu danh sách nên ta chỉ cần sử dụng 1 con trỏ top để quản lý toàn bộ danh sách.



Hình 8: Khai báo cấu trúc Stack

### **4.3. Tạo một stack rỗng (Init)**

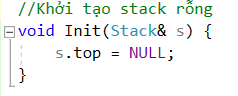
Khởi tạo ban đầu, stack chưa có phần tử nào nên ta gán con trỏ top tới NULL.



Hình 9: Tạo Stack rỗng

### **4.4. Kiểm tra stack rỗng (IsEmpty)**

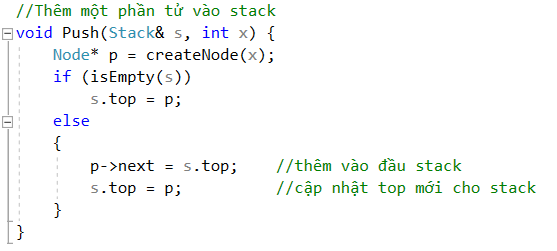
Nếu con trỏ top trỏ về NULL thì stack rỗng ngược lại stack không rỗng.



Hình 10: Kiểm tra Stack rỗng

### **4.5. Thêm một phần tử vào stack (Push)**

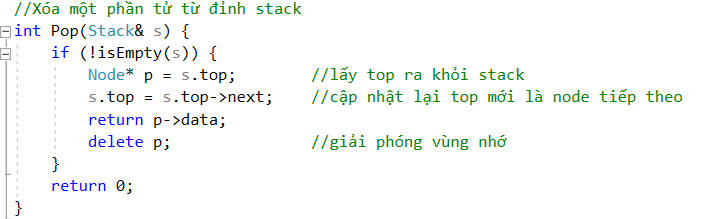
Nếu stack rỗng thì node them vào là node đầu tiên của stack, nếu stack không rỗng thì ta thêm giống như them một node vào đầu danh sách liên kết đơn.



Hình 11: Thêm phần tử vào Stack

### **4.6. Xóa một phần tử từ đỉnh stack (Pop)**

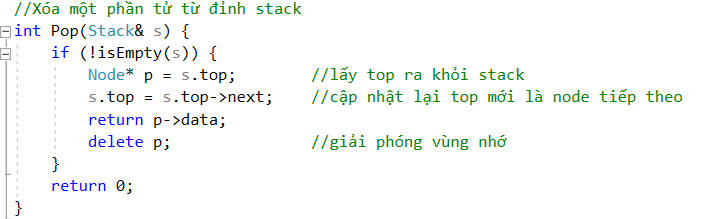
Nếu stack rỗng thì thoát vì không có phần tử nào để xóa, nếu stack không rỗng thì xóa giống như xóa phần tử đầu danh sách liên kết đơn.



Hình 12: Xóa phần tử đỉnh Stack

### **4.7. Lấy thông tin phần tử đầu của stack (Top)**

Cho con trỏ top trỏ tới phần tử đầu của stack sau đó in thông tin ra.



Hình 13: Lấy thông tin phần tử đầu của stack

1. **Ứng dụng**

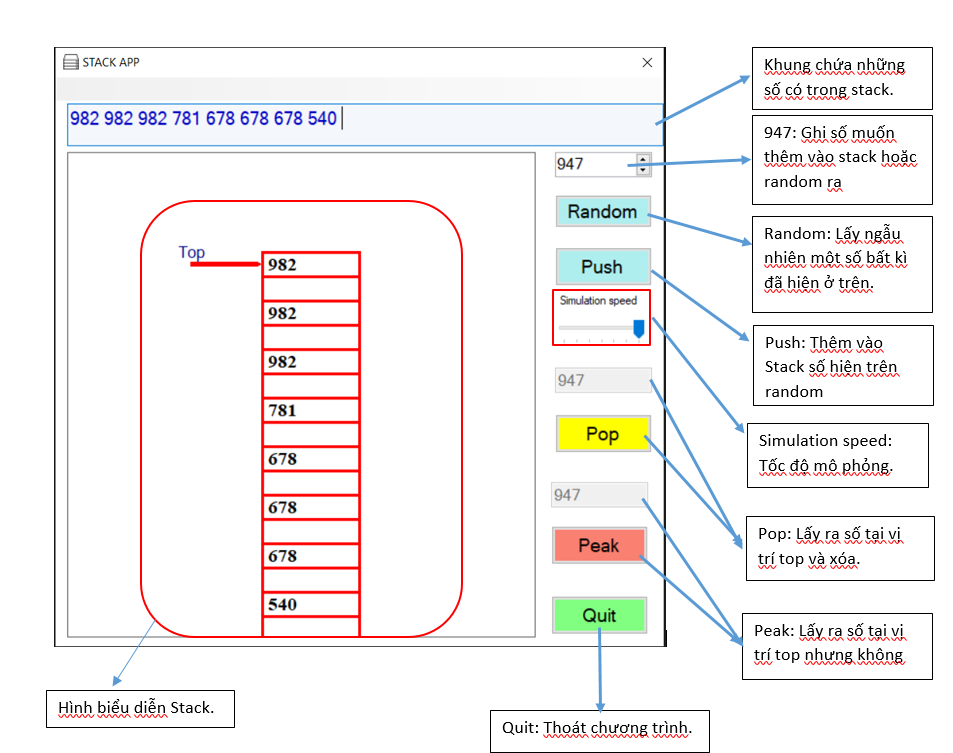
Stack được ứng dụng trong nhiều bài toán cụ thể, chẳng hạn:

* Bài toán tháp Hà Nội.
* Bài toán tính giá trị biểu thức.
* Bài toán tính giai thừa.
* Bài toán chuyển đổi cơ số từ thập phân sang các hệ khác.
* v…v

Đa số các bài toán trên đều dùng đệ quy để giải nên stack được dùng để khử đệ quy.

**II. GIAO DIỆN ĐỒ HỌA (DEMO)**

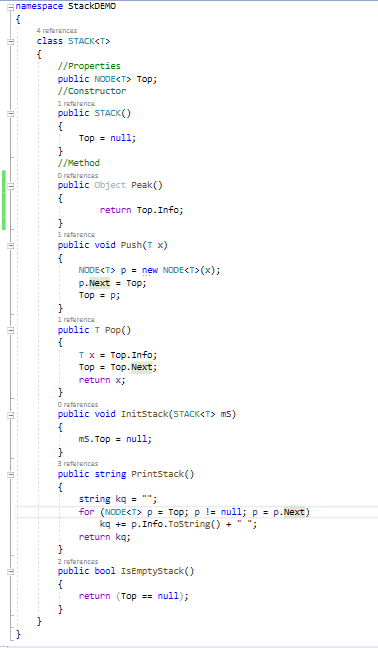
## **1. Giao diện**

****

Hình 14: Giao diện Demo Stack

## **2. Phần Code**

-Code khởi tạo class Stack:



Hình 15:Khởi tạo Class stack

-Code khởi tạo node



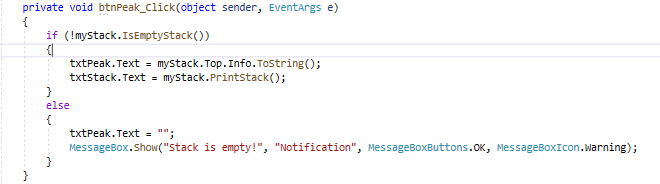
Hình 16: Code khởi tạo node

-Code xử lý khi click Button Push

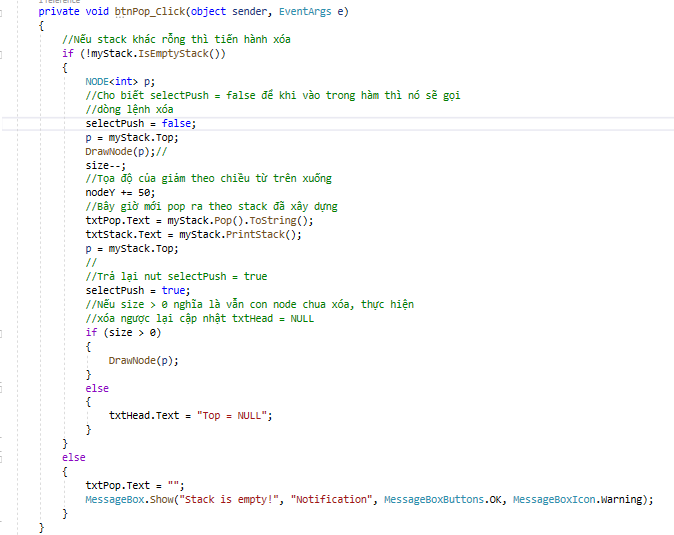
**

Hình 17:Code btnPush\_Click

-Code xử lý khi click vào button Peak

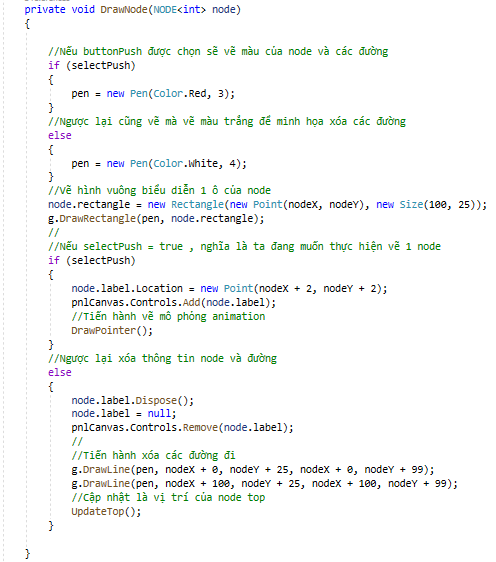


Hình 18: Code btnPeak\_Click

-Code xử lý khi click vào button Pop

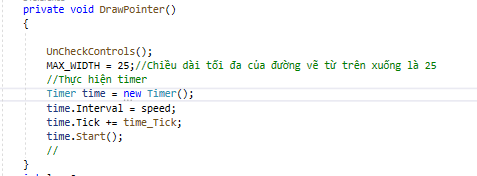
Hình 19:Code btnPop\_Click

*-Code vẽ node*

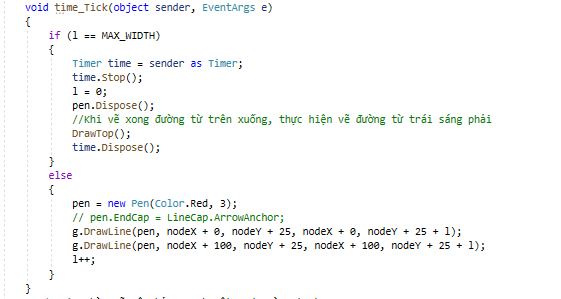


Hình 20:Code vẽ mô phỏng node

*-Code vẽ Pointer bằng Timer*

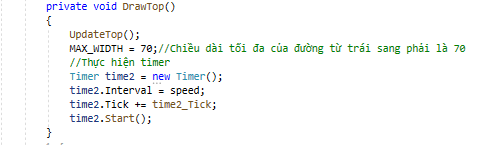
**

Hình 23:code vẽ Pointer

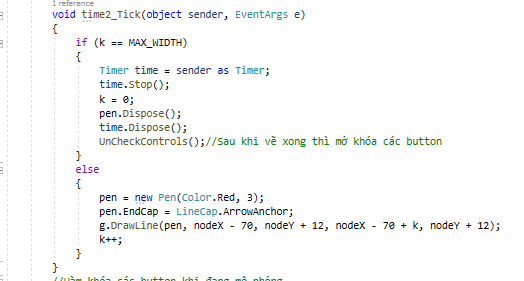
**

Hình 24:Code timer của pointer

*-Code vẽ Top bằng Timer*

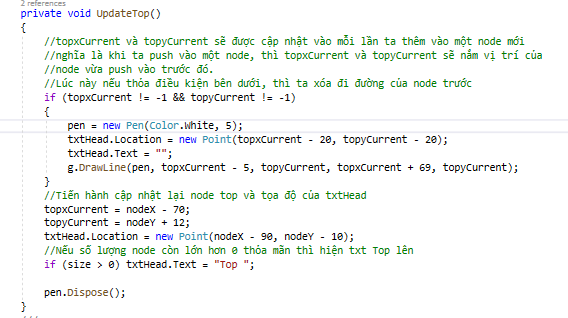
**

Hình 25:Code vẽ top

**

Hình 26:Code timer của top

*-Code UpdateTop*

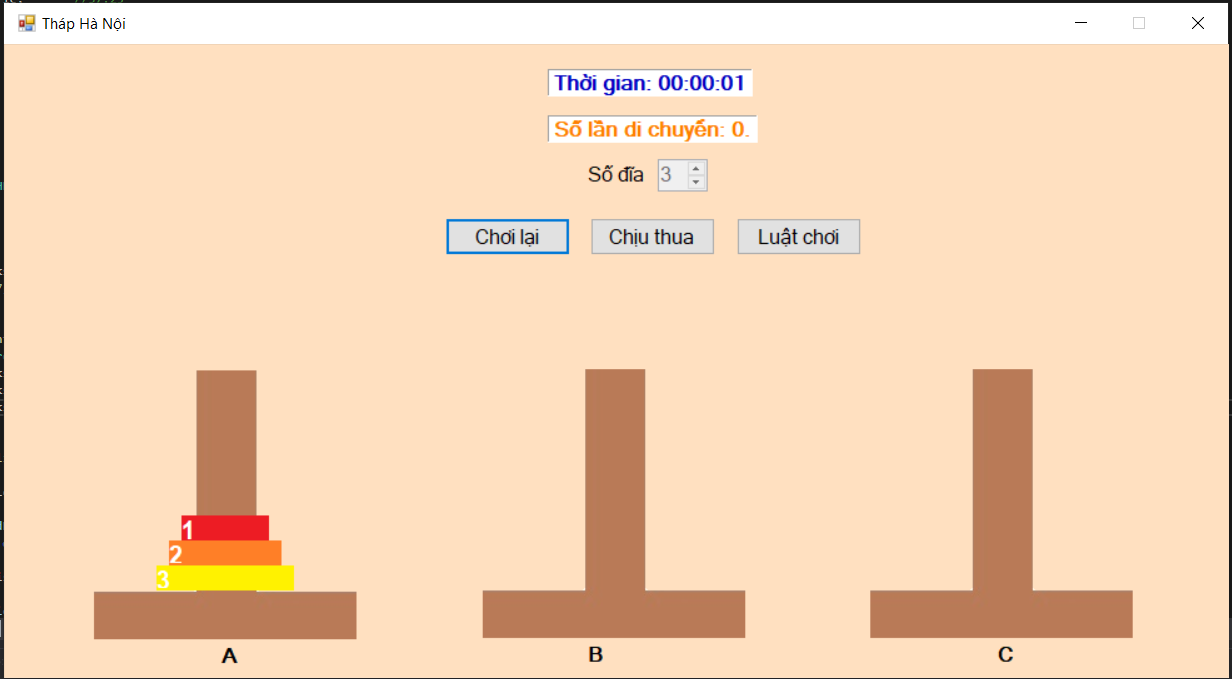
**

Hình 27:Code UpdateTop

# **III. ỨNG DỤNG MINH HỌA STACK**

## **1. Giao diện**

Ứng dụng tháp hà nội trên Winforms:

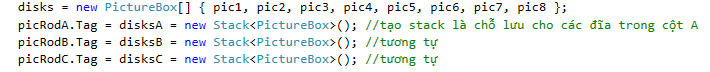


Hình 28: Giao diện game tháp Hà Nội

## **2. Phần Code**

### **2.1. Khởi tạo Stack cho đĩa**

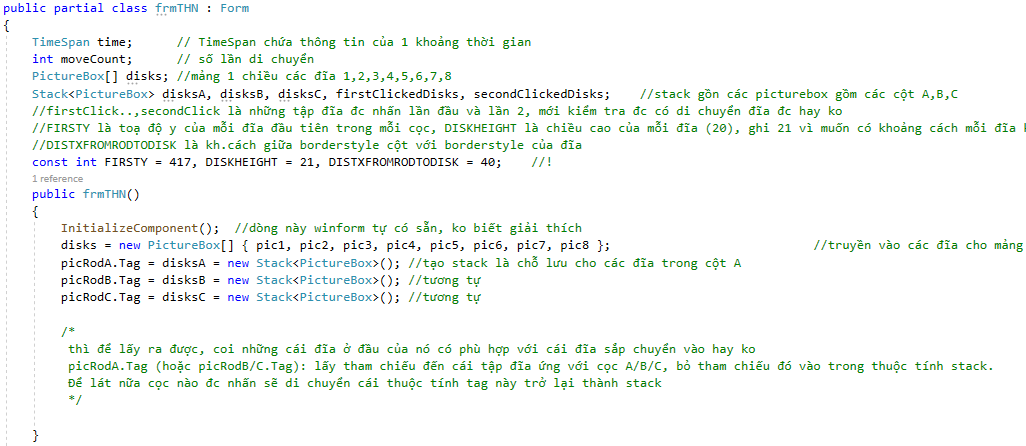
****



Hình 29: Khởi tạo Stack cho 3 đĩa

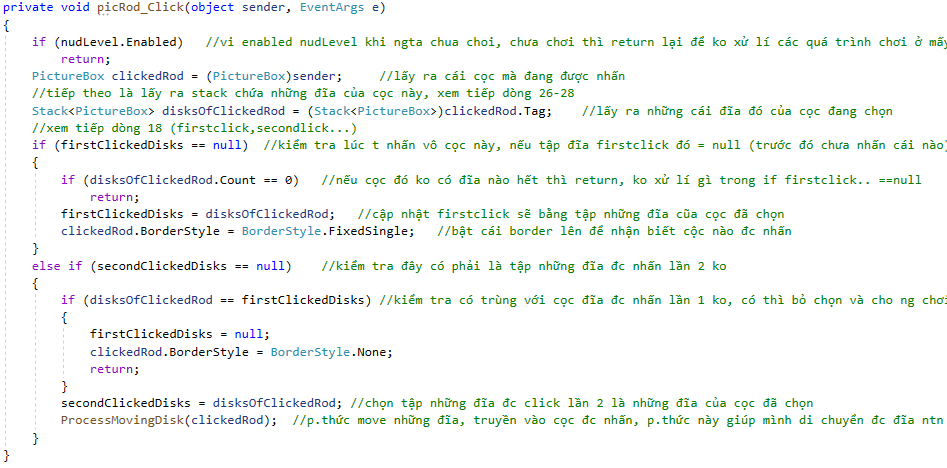
### **2.2. Các thao tác**

- Tạo dựng các biến, khai báo mảng 1 chiều các đĩa (PictureBox), Stack là những cột A,B và C chứa các phần tử là đĩa (tối đa 8 đĩa và tối thiểu 2 đĩa)



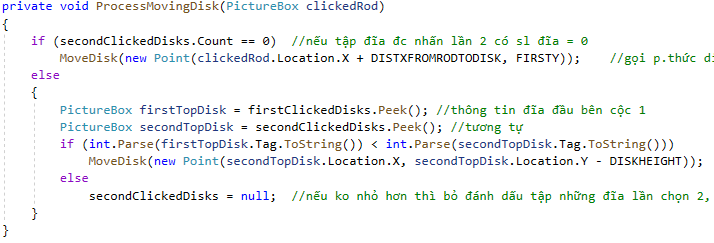
Hình 30:Khai báo và contructor

- Tạo thao tác khi click vào các cột:



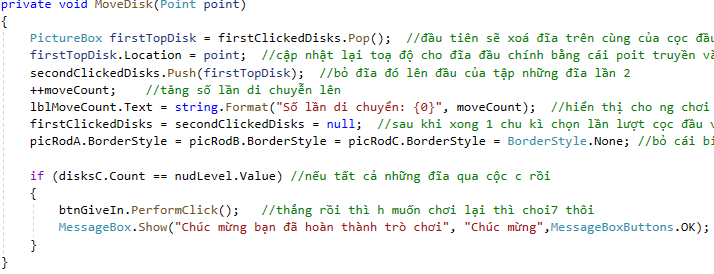
Hình 31:Event click vào các cột

- Thao tác xử lí việc di chuyển đĩa giữa các cột:



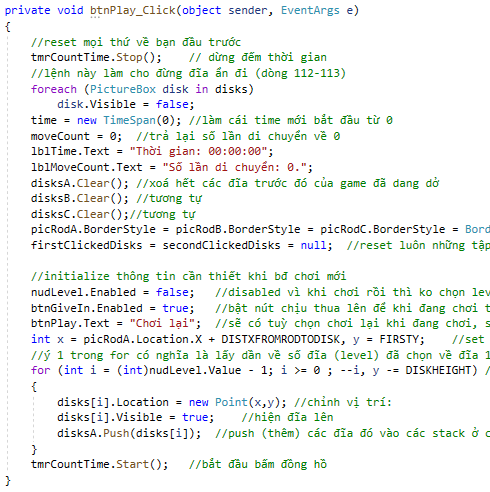
Hình 32:Event xử lý di chuyển khi click ProcessMovingDisk

- Thao tác cho các thức di chuyển đĩa:



Hình 33:xử lý di chuyển các dĩa

- Thao tác khi nhấn nút “Chơi”:



Hình 34:Event click button “Chơi”

# **IV. PHÂN CÔNG CÔNG VIỆC**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Giai đoạn | Công việc chính | Thiều Văn Hoàng | Đặng Minh Khang | Lê Phước Hưng | Trần Nhật Thành |
| Tìm hiểu yêu cầu và đưa ra ý tưởng thiết kế | Phân tích đề tài Project | X | X | X | X |
| Lên ý tưởng thiết kế | X | X | X | X |
| Tìm và bổ sung lý thuyết | X | X | X | X |
| Tìm Source code mẫu |  |  | X | X |
| Đọc bài Source Code mẫu | X | X | X | X |
| Tiến hành thực hiện đề tài | Xây dựng các đối tượng trong game | X |  | X | X |
| Thiết kế giao diện cho game bằng form |  | X | X | X |
| Viết code cho các đối tượng | X | X | X | X |
| Kiểm thử và sửa lỗi | X | X | X | X |
| Hoàn thành đề tài | Viết báo cáo | X | X |  |  |

# **Tài liệu kham khảo**

Nguồn kham khảo :

[http://github.com](http://github.com/)

<https://www.youtube.com/watch?v=vk1tf0EZAgY&t=2781s>

<https://dotnet.microsoft.com/>,

[https://www.howkteam.vn](https://www.howkteam.vn/)

[https://www.youtube.com](https://www.youtube.com/)