Có cần phải backup, header của linked list vào dữ liệu back up nào ko, nếu chẳng may làm mất con header đó (thằng đầu tiên) (ko qtr).

Nhỡ header trỏ đi đâu mà làm mất linked list.

Mất là có thể.

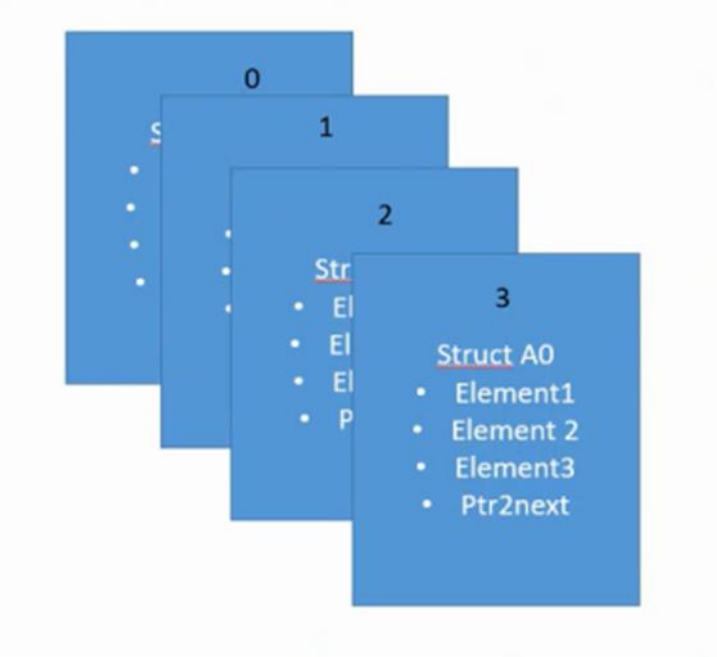
Làm sao để biết dc phần tử nào trong list đầu tiên, dùng con trỏ trỏ đến phần tử đầu tiên.

Linked list có 1 nhược điểm là ko index, mỗi 1 linked list dc mô tả bởi struct.

Không giống mảng vì mảng có các phần tử liên tiếp nhau trong bộ nhớ có thể truy xuất liên tiếp được còn linked list thì rời rạc. giả sử nó ở cạnh nhau thì có cách nào truy cập được ko

Load từ đầu, load thế nào mà ko có index, vậy đâu là phần tử đầu tiên?

Đâu là phần tử cuối cùng

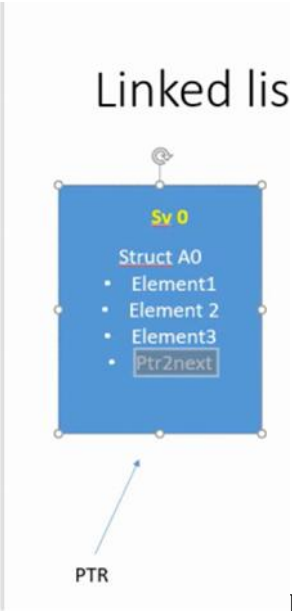


Đâu là phần tử thứ n

Phần tử đầu tiên chỉ có 1 con trỏ, có data

phần tử đầu tiên là phần tử được header chỉ vào, có 1 pointer để chỉ nó bắt đầu.

pointer đùng để chỉ thằng bắt đầu từ list và mình bắt đầu duyệt từ pointer



pointer này sẽ trỏ đến đâu, trỏ đến phần tử thứ 2, vậy bản chất là nó vẫn có index.

Index là chỉ thị. Khi ta gọi index thì nó là 1 cách truy cập nhanh dữ liệu. khi có 1 quyển tử điển, tra từ điển theo chữ cái đầu tiên. Chữ cái đầu tiên đó người ta gọi là index, nghĩa là phương pháp truy cập nhanh phần tử.

Cách thứ 2 là nếu ko muốn truy cập nhanh là dùng con trỏ.

Bản chất của mảng thì complier sẽ biến tấu nó, thực ra bản chất mảng chính là 1 con trỏ và khi truyền mảng vào trong function thì truyền tham trị hay tham chiếu? truyền cả mảng vào trong hàm hay chỉ truyền tham chiếu nó vào thôi? Truyền tham chiếu địa chỉ của mảng đó. Tại vì trong hàm đó có thể thay đổi dữ liệu, đó là trường hợp ngoại lệ.

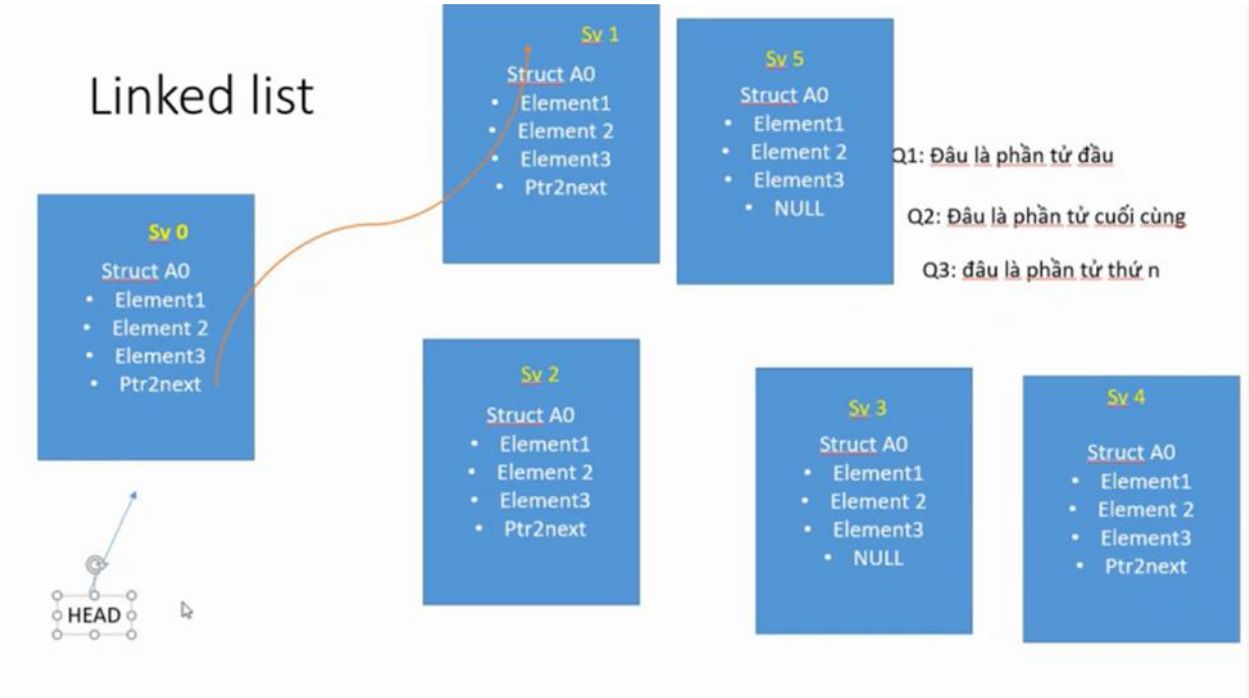
Complier hỗ trợ việc đó.

Còn với linked list thì mình hoàn toàn quản lý dưới các thuật toán của mình.

Thứ nhất ta thấy danh sách liên kết này ko có index cho nên phải tìm cách xác định điểm đầu của nó. Điểm đầu thì dùng 1 pointer được gọi là pointer head.

Vậy đâu là phần tử cuối cùng là con trỏ trỏ tới NULL, phần tử cuối cùng là NULL.

Ko có dữ liệu, ko có 1 dữ liệu gì cả, do người dùng tự định nghĩa tất cả rule của nó. Nếu bảo phần tử cuối cùng là NULL thì khi gặp phần tử cuối cùng sẽ là NULL. Giả sử list có 5 phần tử. trong 5 phần tử đó lại có 2 cái NULL



Sử dụng 1 con trỏ nữa gọi là tail để chỉ vị trí cuối cùng, liệu có cần thiết ko. Tail là để biết được điểm cuối. Liệu có cần tail ko? Có cách nào cần biết phần tử cuối cùng mà ko cần tail ko?

Struct này có 1 pointer PTR 2 next sẽ trỏ đến phần tử tiếp theo, nếu mà pointer của phần tử thứ 3 này đang là NULL tức là nó đang không chỉ vào phần tử tiếp theo nào đấy, tức là list đang bị cắt đứt ở chỗ này thì phần tử thứ 3 này là đuôi.

Câu hỏi: viết 1 chương trình để tìm ra 1 phần tử cuối cùng? Duyệt đến khi nào ptr next bằng NULL thì thôi.

Duyệt từ lúc bắt đầu, có pointer là head thì sẽ biết được địa chỉ bắt đầu, cách mà nó sẽ sử dụng, là nó sẽ duyệt từ sv0 trước (phần tử thứ 0), rồi đọc đến pointer next, rồi pointer này trỏ đến phần tử thứ 2, đọc dữ liệu ở đó ra. Current pointer sẽ chuyển sang trỏ tới phần tử thứ 2 là thằng sv1. Sv1 sẽ đọc và check pointer next là valid tương tự trỏ đến địa chỉ kế tiếp. tiếp tục trỏ đến thằng sv2. Thằng 2 này tiếp tục đọc check tiếp pointer này của nó là valid thì nó sẽ tiếp tục đọc thằng sv3. Thằng 3 check được pointer của nó là NULL mà nó sẽ dừng lại nếu thuật toán của mình cho thằng 3 biết rằng nó là thằng cuối của list mà ko cần có pointer là tail.

Câu hỏi: nếu mình có tail này thì mình sẽ update tail này như thế nào? Khi nào nó thêm phần tử mới nào thì nó sẽ cho cái tail đó trỏ vào phần tử cuối là xong. Được ko? Khi list nó được update, khi màlist đó được thêm phần tử thì cái tail đó mới được bắt đầu update giá trị chứ ko tự dưng mà nó có. Tự dưng cho nó 1 cái list sẵn có. Giả sử ta có danh sách của các bạn học sinh trong lớp thì mình sẽ khởi tạo linked list như thế nào?

Mình biết cái head rồi thì mình vẫn phải duyệt để mình biết tail vì mình không có phần tử cuối cùng. Tại vì khoá cứng thẳng tail lại, sau đó lại chèn thêm các phần tử mới vào linked list của mình, ý là khoá cứng cái đuôi của linked list lại. khoá cứng tức là mình sẽ tạo 1 linked list đầu tiên, sau đó cho con trỏ next trỏ đến NULL thì mình sẽ mặc định thẳng đấy là tail luôn.

Câu hỏi trên có 2 trường hợp tức là mình được cho 1 cục dữ liệu, bây giờ sẽ làm thế nào để khởi tạo 1 linked list. Trong ngôn ngữ lập trình nó được định nghĩa 2 khái niệm là runtime và init và pre-load.

Init có nghĩa là gì, preload có nghĩa là khi ta complie 1 chương trình rồi, nạp xuống mô trường để nó chạy thì object đấy đã có dữ liệu hay giá trị chưa; trường hợp thứ 2 là cái data dữ liệu nó vốn chưa được khởi tạo nó nằm ở đâu đấy, đến khi nào chương trình nó bắt đầu chạy thì dữ liệu đó nó được khởi tạo. ví dụ mình khai báo 1 mảng a bằng

Int A[5] = {0,1,2,3,4}

Kiểu này người ta gọi là khởi tạo luôn, tức là khởi tạo ngay từ đầu, tức là khi chương trình đó được nạp xuống môi trường chạy, hay xuống con chip, hay để trên máy tính, … thì mảng A đấy đã có dữ liệu ngay lập tức là 0 1 2 3 4 rồi. thế nhưng làm thế này thì nó sẽ khác:

Int A[5]

For (I = 0;I <5; i++)

{

A[i]=I;

}

Thì nó sẽ khác, khi chạy chương trình thì nó sẽ có

Chương trình nó là run time thôi, khi mà nó chạy thì dữ liệu trên object mới được khỏi tạo. nếu có 1 cục dữ liệu lớn thì sẽ làm thế nào để khởi tạo linked list

Câu hỏi để chỉ để cho ta phân biệt được 2 trường hợp trên thôi.

Thực sự thì linked list có được khởi tạo theo cách này ko? Ko vì nó ko có index. Sai vì ko liên quan. Index là khái niệm của mảng. khi mà đã nói chuyện linked list thì ko nói gì về index ở đây cả mặc dù là mình có thể implement index cho nó bằng cách cho 1 element. index vào thôi

Ko làm dc vì bản chất của linked list là mình dùng software control là cái giải pháp phần mềm thì chương trình của mình, cái linked list đấy phải được khởi tạo. tức là linked list là

cấu trúc dữ liệu mà nó động chứ ko phải nó là dữ liệu đã được fix cứng. như ở mảng, khi mình khai báo mảng thì ngay lập tức complier sẽ đặt chỗ cho ta hết các dữ liệu rồi.

Cái này ko được support trong bản C hiện tại:

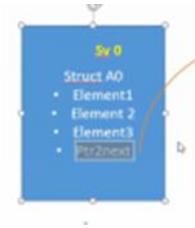
X = size(x) + func();

Int A[x];

Cái này được gọi là dynamic array, về bản chất thì cần có thư viện để hỗ trợ cái này.

Mảng khi mà khai báo thì ngay lập tức nó chiếm ô nhớ, ko cần biết là nó chiếm bao nhiêu cả nhưng mà sẽ chiếm bằng size mà ta khai báo

Còn linked list ko có cơ chế đó, bản chất của nó là các struct rời rạc và nó được liên kết bằng thông tin nó chứa trong chính nó



Thông tin về vị trí thằng tiếp theo ở đâu. Nếu mà có cách nào chỉ ra được thông tin thằng tiếp theo nó nằm ở đâu thì

Thí dụ có 1 mảng

Muốn triển khai nó dạng linked list ntn. Giả sử có danh sách sinh viên của lớp được gửi vào dưới dạng là các ô nhớ ở cạnh nhau ntn. Cấu trúc dữ liệu này tương tự như

mảng.

Có thể ko khai báo là mảng nhưng bản chất là mảng.

Bây giờ muốn truy cập theo index của những phần tử này thì sẽ làm thế nào để truy cập nhanh vào phần tử thứ 4 thì sử dụng con trỏ để trỏ tới phần tử đầu tiên rồi mình trỏ đến

phần tử mình muốn. con trỏ số nguyên cụ thể là con trỏ gì chứ? Using pointer được ko, quan trọng là dùng con trỏ gì

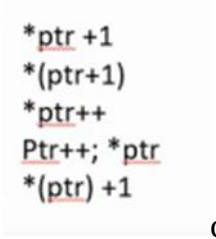
Mình nói con trỏ thì con trỏ có kiểu dữ liệu, có thể dùng typedef được ko, hay struct kiểu sinh viên chẳng hạn, kiểu dữ liệu là kiểu dữ liệu sinh viên chẳng hạn thì lúc đấy mình sẽ có 1 con trỏ sinh viên ptr kiểu này, mình sẽ khai báo 1 con trỏ mà nó trỏ lên block dữ liệu đầu tiên thì nó sẽ lấy hết tất cả các dữ liệu trong đó lên.

Nhưng nếu con trỏ mà truy cập đến ô nhớ đầu tiên của thằng này thì nó có khả năng lấy hết dữ liệu ở đây, nếu nó trỏ đến thứ 2 thì tức là mình cộng con trỏ lên 1, cộng như thế nào:

*ptr + 1

*(ptr+1)

Ptr++ hay là Ptr++ ;ptr* hay là *(ptr) +1



cú pháp nào là đúng, dòng thứ 4 đúng nhưng mà có 1 vấn đề là câu lệnh đấy sẽ tăng giá trị của ptr lên thành cộng 1, còn đúng thì truy xuất theo câu lệnh số 2 là nó sẽ ko tăng địa chỉ đầu tiên thì mình sẽ mất dấu địa chỉ đầu tiên cảu linked list, nó cộng 1 theo kiểu con trỏ struct tức có nghĩa là cộng 1 kiểu struct tức là size của struct là bao nhiêu thì nó cộng bấy nhiêu.

Có nghĩa là nếu pp này thì mình sẽ mất ngay vị trí đầu tiên của block dữ liệu này, cụ thể là mình mất luôn thằng head tức là mình chót tăng con trỏ đầu tiên trỏ đến dữ liệu của block này thì ngay lập tức mất luôn dữ liệu chứa thằng đầu tiên của nó, tại vì mình ko biết nó nằm ở đâu cả, pointer này nó tăng lên là mình ko có cơ hội để mình quay lại duyệt thằng đầu tiên nữa, mình cứ tăng lên nữa thì làm sao mà mình biết được thằng nào là thằng đầu. vậy thì mình phải tạo ra 1 con trỏ khác, ko cần thiết.

Cách thứ 2 là p+1 hay là p+I và tại sao, I ở đây là vị trí mà mình cộng lên, nếu I = 1 là vị trí mà mình muốn cộng tiếp theo, I = 2 thì mình sẽ nhảy tiếp lên từ vị trí header ban đầu thì lên 2 vị trí. Vị trí ở đây là cục của sinh viên 1 hay là sv1, bằng luôn size của kiểu dữ liệu đó, là cách nên dùng.

Vấn đề của mảng và linked list

Thứ nhất đặc điểm bản chất của linked list là nó ko thể truy cập theo index được tại vì là cấu trúc dữ liệu của mảng bản chất là do complier đặt các block dữ liệu ở cạnh nhau nên

là nó có thể dùng các index để truy cập. gốc gác của index chỉ là liên quan đến con trỏ ở dưới.

Có thể dùng con trỏ để truy cập lên trên mảng được, tuy nhiên mình nhìn thấy 1 điểm là nếu lúc bắt đầu thuật toán của mình, nếu mình gán con trỏ thì dùng con trỏ đầu tiên để trỏ đến hoặc cộng lên thì nó sẽ mất ngay điểm bắt đầu của nó ở đâu dẫn tới khi mình sử dụng linked list thì sẽ cần thông tin để lưu trữ thằng bắt đầu ở đâu và kế tiếp của câu chuyện linked list là linked list thường khi khởi tạo có ngay dữ liệu ở đầu không thì câu trả lời là có thể thế nhưng mà thông tin để nó đánh các thông tin cho nó, về bản chất thì các linked list thì bản chất là những object mà các linked list thì nó chưa được khởi tạo nhưng mà data thì có sẵn trên bộ nhớ. Nghĩa là khi người ta đưa cho mình 1 block dữ liệu mà nó có sẵn trong bộ nhớ thì mình vẫn có thể khởi tạo linked list được

Đơn giản hơn là trên bộ nhớ nó có sẵn các dữ liệu đó rồi. giả sử như là:

Ngày trước có 1 chương trình ghi trên Flash thì dữ liệu trên này vốn là nó có sẵn. nếu mà nó có sẵn thì mình có cần khởi tạo lại các dữ liệu này không. Có sẵn rồi thì ko cần

Vấn đề là khi mà nó có sẵn rồi thì mình ko cần khởi tạo lại nhưng vấn đề ở đây là linked list của mình chưa được ini, vì đơn giản linked list của mình chỉ là 1 tập hợp các struct hay những cái biến.

Các khái niệm rất là thực tế, khi mà chương trình chạy, các biến global, local được lưu ở đâu? Trên RAM. Lưu trên Ram thì có nghĩa là khi mình mất điện thì ko còn gì, mất hết dữ liệu thế nên khi mà các cấu trúc dữ liệu mà người ta muốn lưu lại để lần sau người ta đỡ phải làm công việc mà người ta lặp lại hay là những dữ liệu này có thể được update hay là liên quan đến những chương trình liên quan đến setting của game,…

Thì người ta bắt buộc phải lưu trên ổ cứng. nếu mình dùng cấu trúc dữ liệu này thì mỗi lần cái máy tính của mình khởi động lại thì mình phải làm thêm công việc là load hết tát cả các dữ liệu đã lưu từ trước đó lên cái vùng nhớ của mình. Đúng là nó load dữ liệu nhưng mà ko phải load hết tại vì nhìn đơn giản thì nhìn vào game

Có những con game hàng chục GB, rất nặng, làm sao mà có thể load dữ liệu lên Ram của mình được. mình giờ phổ thông đang là 8GB còn nếu có điều kiện hơn thì nâng thành 16GB. Khi mà đến hàng chục GB thì làm sao mà đủ chỗ cho các giá trị đó được. vấn đề ở đây là gì, là các dữ liệu luôn được load lên trên phần bộ nhớ của chương trình, chương trình để thực thi nó bản chất là tất cả các phần mềm chạy trên window hay là chạy trên các hệ điều hành thì nó sẽ 1 những chương trình nhỏ, là các hàm main nhỏ hoặc là những function nhỏ thôi.

Thì trong những function đấy sẽ có những biến global hay local của nó. Khi mà ko muốn sử dụng nữa thì nó sẽ load và upload cái chương trình đấy, các hệ điều hành đấy có 1 tính năng là tính năng load INMIT, tức là nó load ảnh lộ trình lên xong và nó lại có thể gỡ bỏ

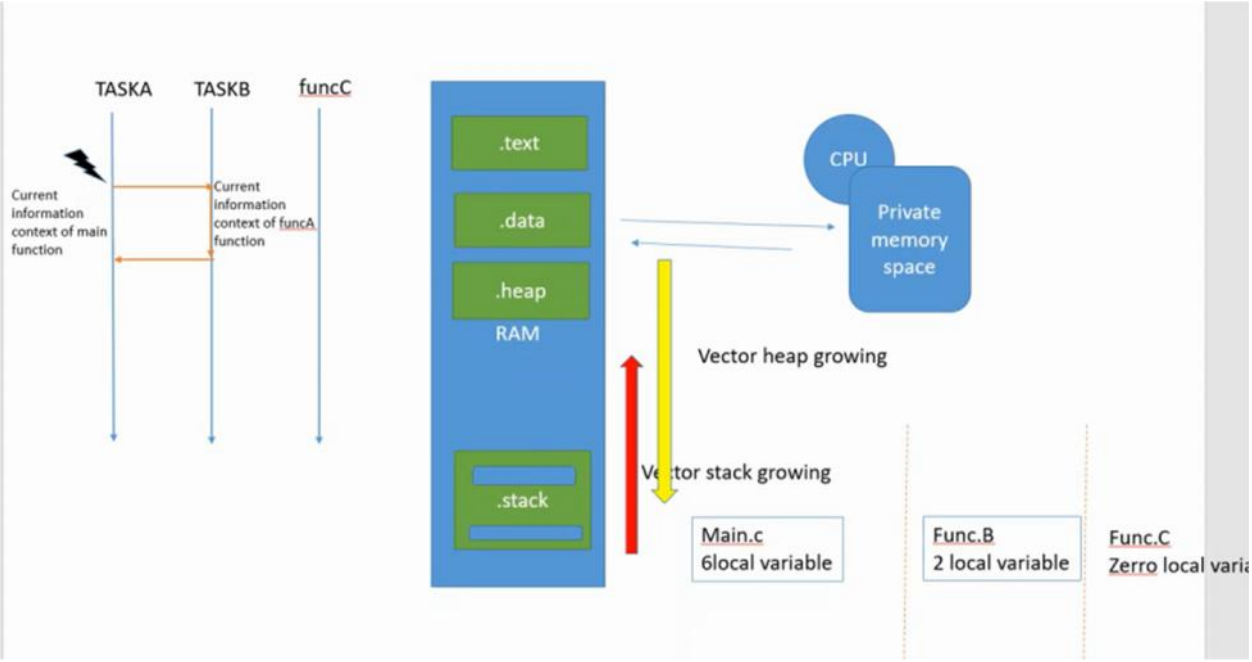
cái đấy đi. Thì mỗi lần nó load đấy thì các biến global hay là những cái runtime variable nó sẽ được khởi tạo trên RAM và nó có thể sử dụng các cái data nằm ở đâu đấy thì ví dụ với linked list này thì có thể khối data này có thể nó đã nằm ở đâu đấy rồi và công việc của mình là nó sẽ khởi tạo cái linked list thực sự mà người dùng có thể dùng để truy cập và khi đấy thì công việc đấy có thể là mình sẽ phải làm gì để khởi tạo 1 cái linked list mà trong khi khối block dữ liệu mà nó sẵn có để mình có thể dùng linked list và mình có thể dùng cái phương pháp, các thuật toán mà mình có thể truy cập hay là sắp xếp hay là đọc dữ liệu ra. Là phải biết được dạng dữ liệu của cái dữ liệu được lưu trước ở trên flash. Giả sử mình biết và mình viết chương trình ra để lưu dữ liệu mà trên chính chương trình đó nó lưu dữ liệu lên trên và chính chương trình đấy cũng là chương trình mà nó load dữ liệu mà mình đã lưu ở FLASH, chương trình đấy là chương trình của mình, kiểu dữ liệu mà mình sẽ biết.

Mình biết thì mình sẽ khởi tạo cái linked list theo cái dạng dữ liệu đó. Quan trọng là mình khởi tạo như thế nào, là viết thứ tự sắp xếp các phần tử. thứ tự sắp xếp sao mà mình biết được, tại vì lúc trước mình đang sử dụng thuật toán linked list thì các dữ liệu có thể mình đang nằm rải rác trong bộ nhớ.

Có thể lưu tuần tự từng node này lên trên file và sau này lấy lại, cứ thế mình cứ lấy.

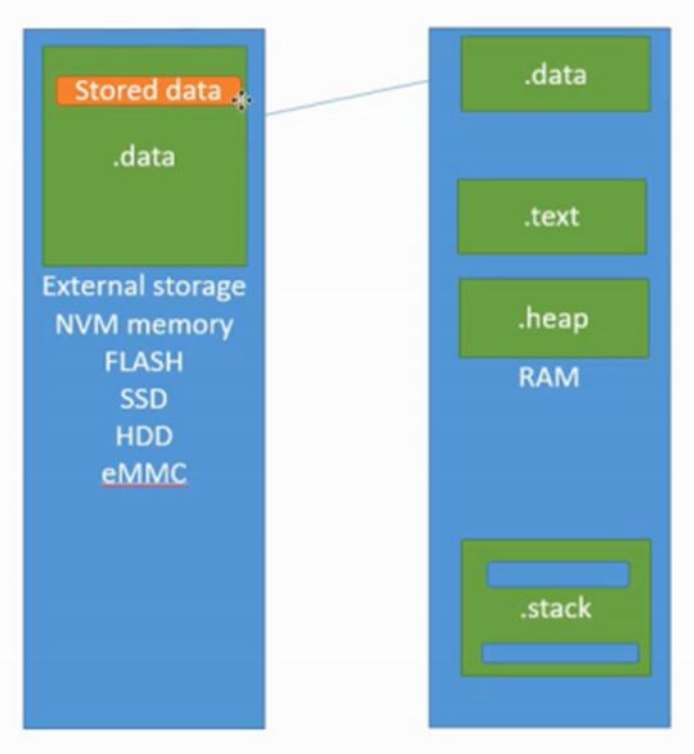
Cứ lưu hết các thứ tự ở đây thành các địa chỉ (ko phải lưu địa chỉ mà lưu thông tin) của nó trên file. Thông tin là lưu cái gì, là lưu hẳn cái struct như thế này lên file nằm ở ổ cứng. nhưng thế thì bản thân nó tất cả nằm trong ổ cứng rồi. tức là mình biết cấu trúc xong mình đọc về kiểu gì?

Lấy ví dụ:

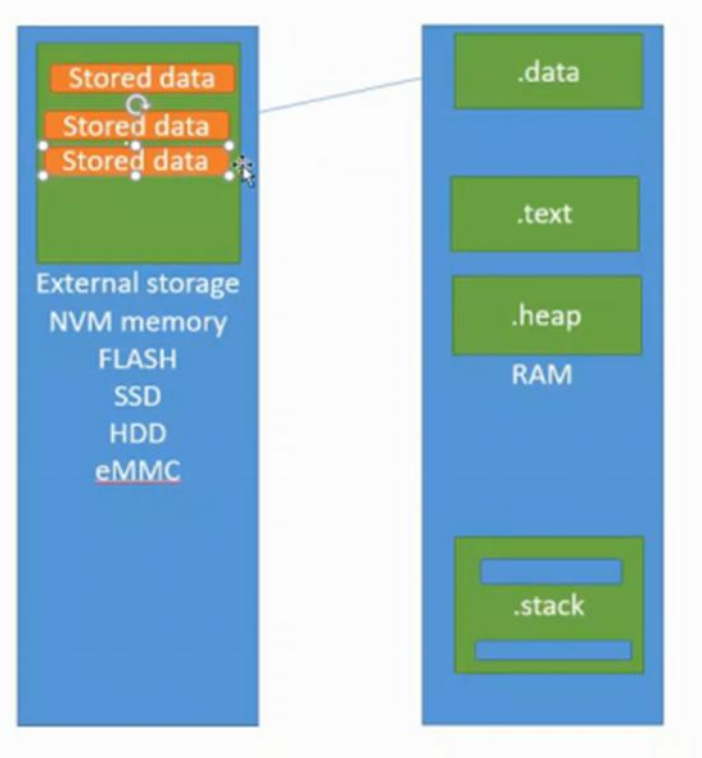


Bộ nhớ của mình là RAM, có vấn đề ở chỗ khi mà nó mất điện thì nó sẽ cần 1 thằng khác được gọi là external storage hay còn gọi là runtime memory hay còn gọi là flash, hdd, ssd, immc, rất nhiều loại bộ nhớ mà nó ko mất điện

Vấn đề ở đây là khi mà nó dữ liệu này bản chất nằm ở trên extenal storage và khi được thực thi thì nó sẽ được đẩy lên trên RAM và hình vuông to hơn có nghĩa là dữ liệu trên này rất là lớn nên đây chỉ là phần nhỏ của dữ liệu đó thôi. Bản chất là khi dữ liệu ở trên này đã được xử lý xong hết rồi mà nó ko cần nữa thì nó sẽ ghi toàn bộ dữ liệu này xuống extenal storage và nó sẽ chiếm ô nhớ khoảng nhỏ nhỏ (hình vuông xanh trong hình chữ nhật lục).



Vấn đề ở đây là các linked list khi mà nó được ghi ở trên này thì ghi thành các cục rời rạc ở trên này, tất nhiên là địa chỉ trên này sẽ khác cho nên là về mặt liên kết thì nó sẽ giống nhau và khi chương trình nó load cục này thì nó sẽ load tất cả lên trên này, đại khái là vậy



Vấn đề là mình ko có cách nào để truy cập lên trên các element data nhỏ nhỏ này, các cục dữ liệu rời rạc như thế này. Tại sao? Khi nói đến software thì mình cần có 1 cái gì đó để biểu diễn nó và cái biểu diễn đó là gì? Kiểu dữ liệu int hay kiểu dữ liệu uint8_t, uint32_t, kiểu dữ liệu struct, kiểu dữ liệu con trỏ, tất cả những cái object dùng để access lên những cái lên trên bộ nhớ. Làm sao mà có thể tự dưng đọc được data này không nếu mà không có kiểu dữ liệu tương đồng với nó?

Đọc ra sẽ bị sai mất thông tin. Sai vì ko có gì để mà đọc cả mà phải theo 1 kiểu dữ liệu nào đấy để cho nó đọc được. có bao giờ mình khai báo kiểu dữ liệu mà nó có thể đọc được kiểu dữ liệu đó ra là gì không, có trường hợp đấy không? Không, nghĩa là mình luôn luôn có 1 khái niệm gì đấy để có thể mô tả kiểu dữ liệu ở đây, ko cần biết nó là linked list hay là struct hay là biến hay là cái gì cả mà bắt buộc nó phải có 1 kiểu dữ liệu. đối với trường hợp này, linked list là bản chất mà mình hiểu là các block dữ liệu này nó nằm rải rác trong bộ nhớ và các block dữ liệu này có thông tin của các block dữ liệu kế tiếp. vấn đề ở đây là mình biết được kiểu dữ liệu này vốn là bản chất đã được liên kết với nhau rồi thì mình ko cần thiết là tìm hết các kiểu dữ liệu đấy.

Ngay từ lúc bắt đầu mình chỉ cần có dữ liệu của thằng đầu tiên thôi, chỉ cần biết thằng đầu tiên nó nằm ở đâu trên bộ nhớ thôi. Khi mà chương trình sẽ load lại thì chỉ tìm 1 cách nào đấy mà chương trình có thể load nhanh nhất các thằng storage data đầu tiên nó xuất hiện và được tìm thấy đầu tiên và từ thằng đấy mình có thể khởi tạo ngay lập tức linked list, mình chỉ cần biết thằng đầu tiên thôi.

Và linked list kế tiếp nó sẽ làm gì? Mình có cần khởi tạo trước thằng kế tiếp không, thằng tiếp theo ko, cần ko? Tuỳ vào việc mình muốn lấy bao nhiêu để xử lý. Thực ra nếu mình khởi tạo thì nếu mình chỉ muốn khai báo lập tức cái linked list thì lập tức chỉ có 2 trường hợp khi mà mình muốn scan tiếp những thằng kế tiếp và 1 khi muốn biết thằng tail thực sự như thế nào và đang nằm ở đâu. Nếu ko dùng tail hay không dùng tail thì lại là 1 chuyện khác.

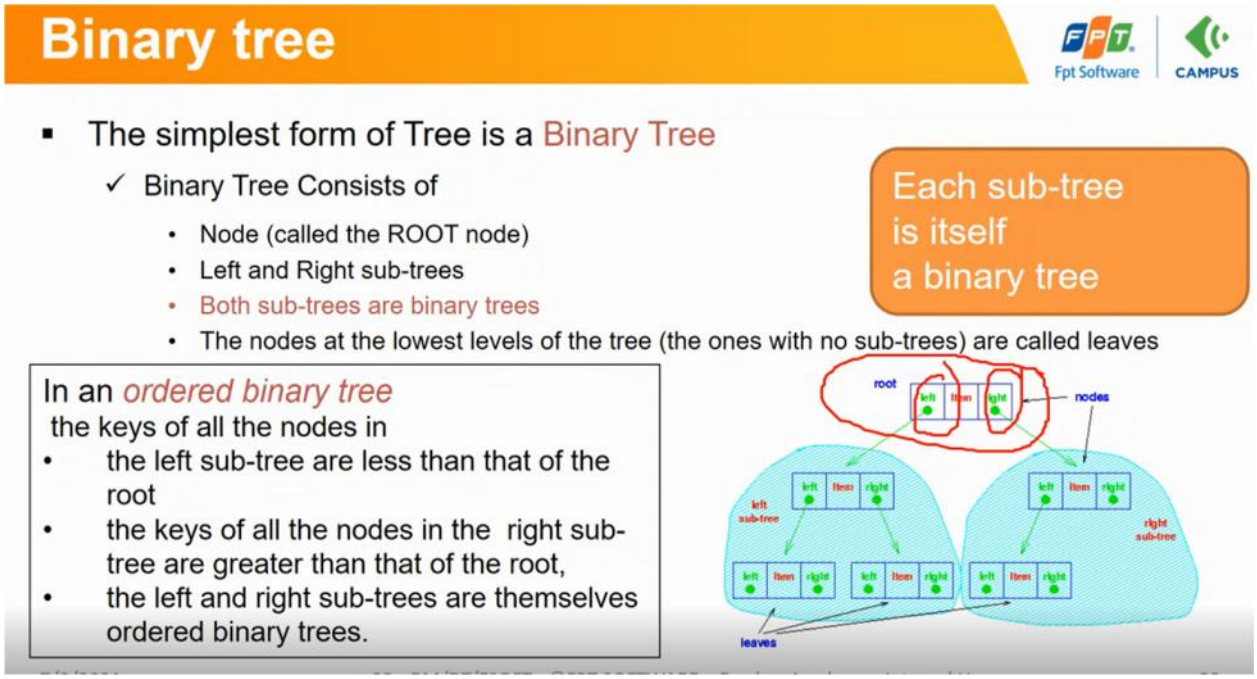
Cái thứ 2 là muốn check xem liệu chuỗi này liệu nó có lỗi hay không tại vì rất có thể là các struct này có thể nó sẽ có issue, lỗi do hỏng bộ nhớ hay hỏng cell, đang có vấn đề hay là copy bị lỗi, có rất nhiều vấn đề.

Còn nếu không thì chỉ cần lấy thằng đầu tiên là được. nếu mình làm như thế này thì ?

Với linked list này thì mình không thể tự truy xuất được ngẫu nhiên phải ko, tức là yêu cầu ?, có nghĩa là đến luôn 1 cái thông tin của 1 thằng nào đó. Có nghĩa là mình phải lặp từ đầu cho đến chỗ đấy. nhưng ko có cách nào truy cập được. thế nên là mình muốn truy xuất ngay lập tức phần tử thứ 3 của chuỗi của list này thì ko có cách nào được cả, ta phải truy xuất từ cái thằng đầu tiên.

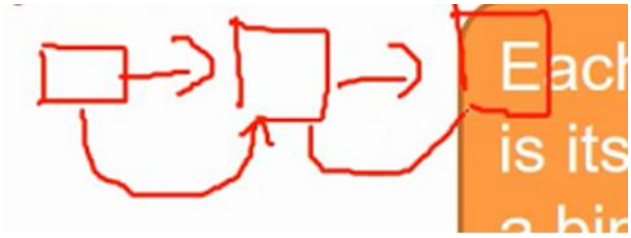
Muốn truy xuất từ thằng phần tử thứ 3 này thì phải truy xuất được thằng từ phần tử thứ 2 từ dưới lên, muốn truy xuất phần tử thứ 2 thì phải truy xuất từ phần tử thứ 1, rồi đến phần thứ đầu tiên là thằng thứ 0 tại vì mình ko biết thằng thứ 3 này nằm ở đâu cả. mà thông tin của thằng thứ 3 lại nằm ở thằng thứ 2. Thằng thứ 2 nằm đâu thì lại nằm ở thằng thứ 1 và đến thằng thứ 0 và thằng thứ 0 là mình biết chứ mình ko bao giờ biết thằng thứ 2 nằm ở đâu hay là thằng thứ 1 nằm ở đâu cả trừ khi là có thuật toán gì đấy để mà giúp mình truy xuất nhanh được cái đó nó chỉ đơn giản là software là chính. Đơn giản là cách con người nghĩ ra để nó quản lý 1 cái dữ liệu thôi, tất cả những cái concept, tất cả những cái kiểu dữ liệu khác thí dụ binary tree chẳng hạn nó cũng chỉ là 1 concept để mà cách quản lý dữ liệu.

ở binary tree thì ta có 2 cái struct để quản lý 2 cái con trỏ, cái mũi tên rất đặc trưng để thể hiện con trỏ khi mà người ta biểu diễn cái gì thì sử dụng mũi tên thì cái đó gần như chắc chắn là con trỏ. Mình sẽ nhìn thấy ở đây là gì, cái root này bản thân nó là 1 struct

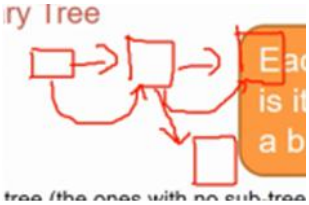


Bản thân nó là 1 con trỏ, con trỏ này bản thân nó trỏ đến chính kiểu dữ liệu của nó giống linked list

Con trỏ thì vào phần tử kế tiếp của nó nó cũng có 1 con trỏ để trỏ vào phần tử kế tiếp. chủ yếu mình sẽ nhìn vào đây và thấy rằng sẽ có 2 con trỏ. 2 con trỏ này sẽ chỉ đến sub tree của nó. Ý tưởng xây dựng của nó khác linked list ở chỗ là thằng linked list nó xây dựng 1 kiến trúc dữ liệu/ cấu trúc dữ liệu là từ thằng này sẽ nối tới thằng kia và nó sẽ chỉ nối đến 1 phần tử thôi

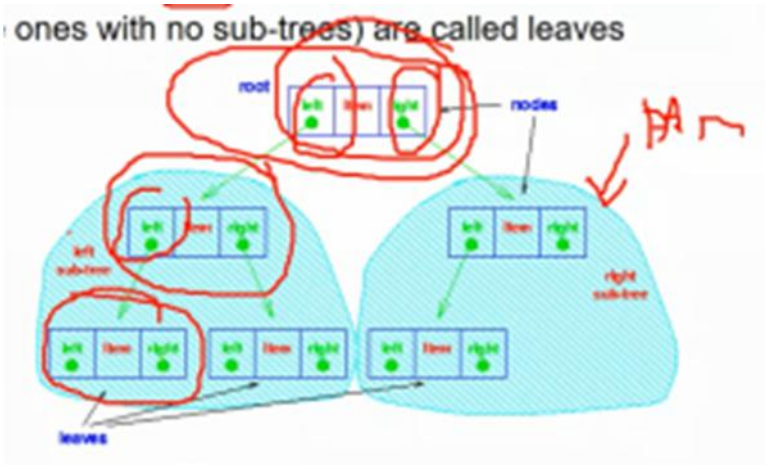


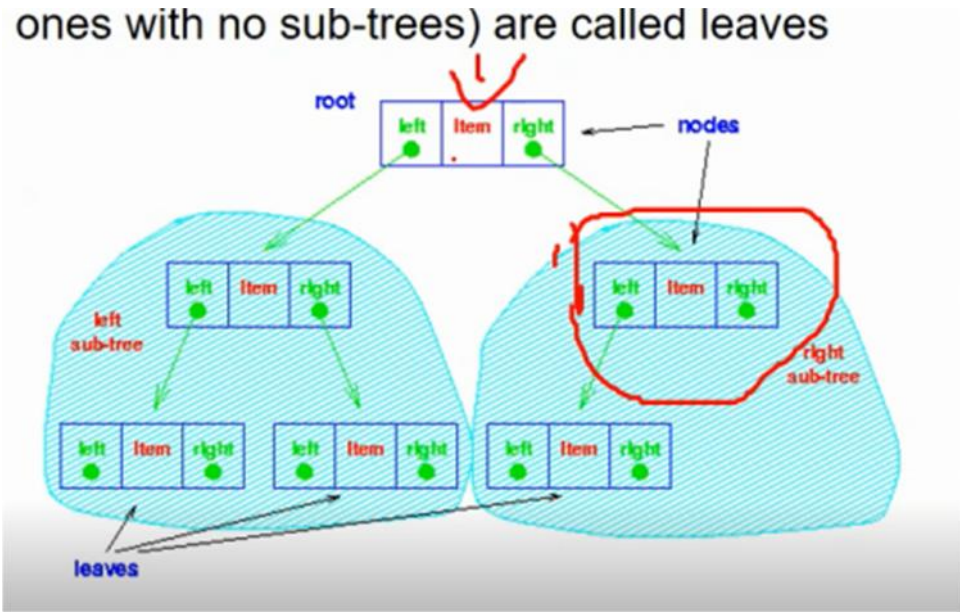
Nhưng nếu linked list thì nó có thể rẽ nhánh sang thằng khác đươc nữa



Thì cái này được người ta gọi là cây nhị phân

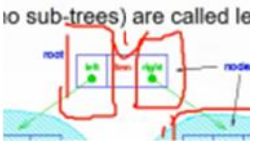
Câu hỏi đặt ra cho cây nhị phân là có 1 pointer đang ở đây (tree con phải) có thể truy cập lên root được ko





Nếu đang ở đây thì có thể truy xuất lên root được ko (thằng con truy xuất lên thằng bố) có được ko? Nếu có tức là có địa chỉ của vùng đó phải ko? Nếu mà mình truy ngược lên địa chỉ của vùng đó lên được ko? Ko được

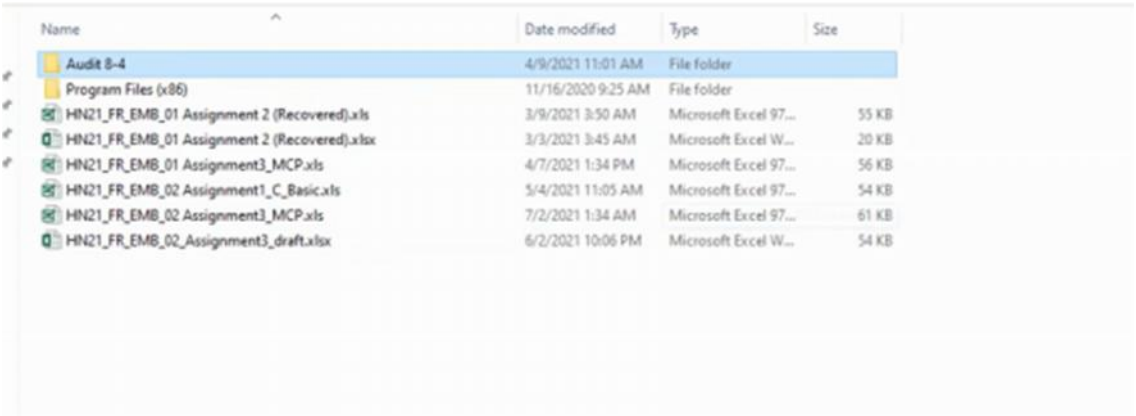
Ko được vì thằng bên trên chứa địa chỉ của thằng bên dưới, địa chỉ của thằng bên trên xuống thằng bên dưới chứ thằng bên dưới đâu có biết thằng bên trên là thằng nào đâu



Cái này chứa thông tin của thằng kế tiếp ko chứa thằng ở trên nó. Linked list cũng là 1 chiều thôi

Tiếp theo của nó ko chứa thông tin gì của thằng kế tiếp nó cả hay ở trước nó cả, tức là nó sẽ ko bao giờ có kiểu truy xuất dữ liệu dùng chính data ở đây để truy xuất ngược chiều được

Tuy nhiên thì vẫn có cách để truy xuất ngược lại thằng này được bằng cách thêm con trỏ, biến tạm thời để lưu trữ. Đặc trưng của các cây là



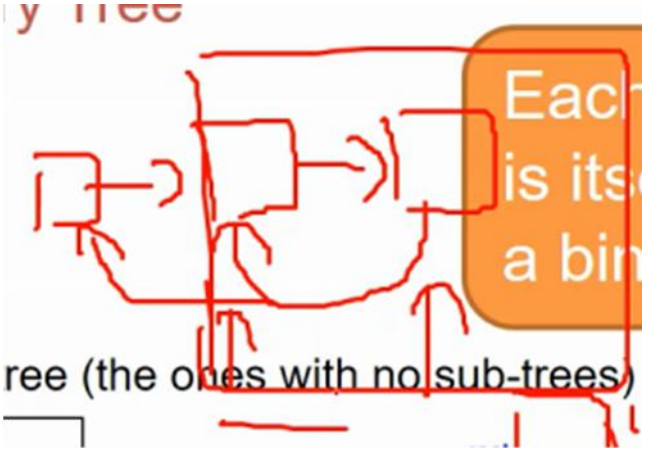
Nhìn ntn là hiểu

Muốn quay ngược lại thì bấm backspace

Có nghĩa là ở đây sẽ thêm 1 cấu trúc dữ liệu gọi là kiểu stack tức là nó sẽ lưu trữ lại thằng ở trước đấy mình sẽ, cái pointer mà ở trước đấy mình đã truy cập hay có thể kết hợp kiểu dữ liệu binary tree này dùng với kiểu dữ liệu stack thì mình có thể làm được 1 kiểu cấu trúc có dữ liệu file gần giống như kiểu window vậy. nhìn ở chỗ này thì mình có 1 thuật toán gì liên quan đến việc truy xuất tree này bằng tên hay là thứ tự. tuy nhiên là mình ko có cách nào để truy xuất nhanh được ngang hàng

Tức là nó sẽ có phương pháp để duyệt tất cả các nhánh ngang hàng với thằng này.

Khi mà mình đang xử lý 2 cái block dữ liệu hay là 2 cái phần tử dữ liệu trong 1 chuỗi phần mềm, chuỗi data khi mà đang ở trong data structure thì mình sẽ vẫn chứa thông tin của thằng này

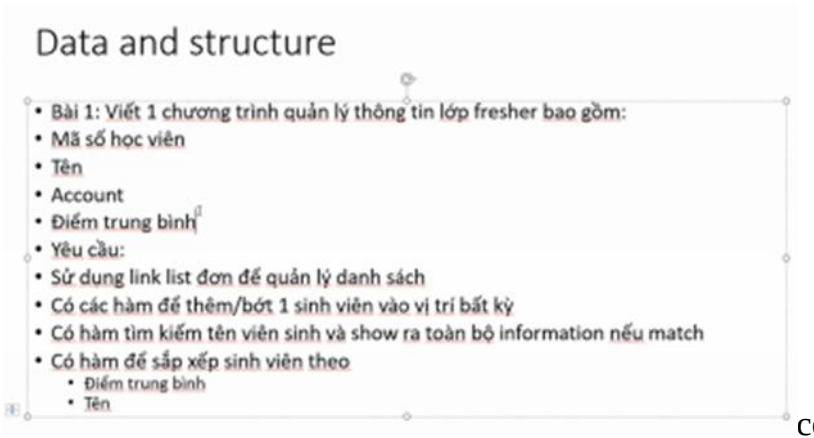


Tại 1 thời điểm thì mình sẽ backup dữ liệu này vào 1 con trỏ khác hay là 1 thông tin nào đấy ở ngoài thì dẫn đến việc khi mà mình truy cập đến node kế tiếp thì đâu đó mình vẫn còn dữ liệu của thằng này để mình có thể thoát, quay lại cái, tức là mình có thể comeback lại dữ liệu đằng trước đấy nhưng mà chỉ ở 1 thời điểm. khi mà block dữ liệu chuyển sang 1 block dữ liệu kế tiếp này rồi thì những thông tin này chắc chắn sẽ mất đi, cách này được người ta gọi là tempory data hay tempory pointer chẳng hạn. thì đây là ví dụ về cây nhị phân thôi

Cây nhị phân về bản chất giống như là mở rộng của linked list thế nên cây nhị phân lại nằm ở phía sau thằng linked list

Assignment này thì dữ liệu được lấy ở đâu hay là khởi tại trong RAM. Tất cả bài tập đều chạy trên Ram chứ ko chạy trên Flash. Dữ liệu đấy sẽ được nhập từ bàn phím hay là hard code luôn ở trong code

Có cách nào xây dựng để suy ra hard code ko hay binary ko. Như vậy là phải nhập vào từ bàn phím hoặc là có thể xây dựng dữ liệu dưới dạng kiểu mảng nhưng khi khởi tạo chương trình có thể in ra linked list này từ mảng đó



có thể lưu 3 dòng đầu tiên ở trong mảng từ a0 đến a thứ n nhưng mà mình yêu cầu là khởi tạo bằng linked list có nghĩa là mình sẽ tạo 1 linked list mới, mình sẽ cop dữ liệu từ mảng đó sang linked list của

mình chứ ko nhất thiết là phải khởi tạo dùng cái danh sách liên kết sẵn có, tức là từ danh sách liên kết đã được khởi tạo ở trên data sẵn có, đấy là 1 bộ đệm thuộc phạm trù khác gọi là khởi tạo danh sách liên kết mới copy dữ liệu từ data sẵn có thì lại là 1 phạm trù khác

Nên là khởi tạo 1 linked list tuỳ ý nhưng mà data này dựa trên data sẵn có, có thể là nhập từ bàn phím hay là dữ liệu mà mình tự định nghĩa sẵn ở trong bộ nhớ nên là mình sẽ copy sang linked list của mình.

Dữ liệu mình ko nhập từ bàn phím này ko phải test mà mỗi lần nhập đi và nhập nhiều lần thì dữ liệu đầu vào đấy từ ngoài ổ cứng hay là mình sẽ khởi tạo ngay trên đầu chương trình?

Ko ổ cứng, ko thể truy cập vào ổ cứng theo cách bình thường mà phải dùng các API. ổ cứng ko có địa chỉ nên ko đơn giản là truy cập như thế được thế nên từ cái sẵn có mình nên khởi tạo 1 cái dữ liệu của mình coi như nó là ổ cứng (giả lập ổ cứng) khi mà trong chương trình có sẵn dữ liệu đấy rồi và dùng dữ liệu đấy để khởi tạo 1 linked list còn khởi tạo như thế nào thì ko quan tâm

Tức là dùng mảng để lưu thông tin của cả lớp vào đấy, ko quan tâm đến cái mảng đấy nhưng mà sẽ dùng mảng đó để copy thông tin trên mảng đó để khởi tạo linked list của mình khi mà chương trình bắt đầu chạy

Tức là khởi tạo như

mảng struct, linked list cứ bám theo những cái đó để init các element của nó lên thôi

Trên ssd và hdd

mình chỉ cần có 1 thằng dữ liệu đầu tiên là mình sẽ có được những thằng dữ liệu sau nhưng mà khi save vào thì cũng save vào như thế này tức là mình khi tạo được 1 linked list xong là nó sẽ lưu đúng lại là mình chỉ cần lấy 1 thằng đầu xong là sẽ có những thằng sau phải ko?

Khi nó lưu thì nó sẽ ôm cả cục data ném thẳng lên external storage chứ ko nhặt từng chỗ nhặt vào chỗ này

Mặc dù trên Ram 2 địa chỉ này khác nhau nhưng trên external storage thì 2 địa chỉ này là 1. Chương trình của mình chỉ chạy trên RAM thôi còn lưu trữ thì nằm ở ES, chỉ để lưu trữ chứ ko truy cập thẳng vào chương trình