LỜI NÓI ĐẦU

Cấu trúc dữ liệu và giải thuật là học phần rất quan trọng đối với người lập trình. Môn học này được xem như là nền tảng của lập trình máy tính. Nó là cơ sở vững chắc để giải quyết một số bài toán, đồng thời cung cấp cho chúng ta hiểu biết về các giải thuật tác động lên dữ liệu như thế nào, cũng như tổ chức dữ liệu để giải quyết các bài toán sao cho hiệu quả nhất, tối ưu hóa nhất. Và cũng là môn học tạo ra môi trường khởi đầu để các em sáng tạo.

Chúng em nghiên cứu và thực hiện đồ án này như một cách để cũng cố và mở rộng kiến thức về môn cấu trúc dữ liệu. Thông qua quá trình thực hiện đồ án, chúng em đã nắm bắt được những kỹ thuật quan trọng của việc xây dựng cấu trúc dữ liệu và cách xây dựng thuộc toán sao cho tối ưu nhất.

Bài Toán “Sắp xếp danh sách liên kết đơn bằng phương pháp merge sort” mà chúng em nghiên cứu và trình bày trong báo cáo sau đây là một trong các ví dụ của môn cấu trúc dữ liệu và giải thuật.

GIỚI THIỆU VỀ ĐỀ TÀI

**Đề bài**

***“Sắp xếp danh sách liên kết bằng phương pháp merge sort”***

Khởi tạo một danh sách liên kết đơn ngẫu nhiên gồm n phần tử với n nhập từ bàn phím. Hãy dùng phương pháp sắp xếp merge sort (sắp xếp bằng con trỏ ID) để sắp xếp sao cho danh sách liên kết đơn giảm dần.

Lý do chọn đề tài

Các kiểu cấu trúc dữ liệu như danh sách liên kết đơn, ngăn xếp, hàng đợi, cây nhị phân tìm kiếm…cùng với các giải thuật chiếm một vị thế rất quan trọng tronhg khoa học máy tính. Ngày nay, với sự phá triển của công nghệ thông tin, các thuật toán mới ra đời giúp cho con người giải quyết những bài toán phức tạp. nhưng vai trò của kiểu cấu trúc dữ liệu link list không hề giảm bớt, nó chính là kiểu dữ liệu cơ sở để áp dụng các thuật toán phức tạp khác. Nhờ có danh sách liên kết mà một số bài toán phúc tạp được giải quyết một cách dễ dàng hơn.Trong kiểu link list thì các thuật toán sắp xếp luôn thông dụng, nó có thể giải quyết một số bài toán liên quan đến thục tế như quản lý nhân viên, quản lý một chuỗi cung ứng dịch vụ nào đó mà đòi hỏi phải có sự sắp xếp theo một tiêu chí nào đó để thuận tiện trong việc quản lý.Một số phương pháp sắp xếp danh sách liên kêt có thể kể ra như selection sort, interchange sort,insertion sort, build sort….Nhưng điều quan trọng với một lượng dữ liệu cực lớn thì các phương pháp sắp xếp trên có còn hiệu quả, có còn đảm bảo tiết kiệm chi phí hay không… Merge sort là cách giải quyết có thể nói là tối ưu trong việc dữ liệu cực lớn nó có thể sắp xếp nhanh rất nhiều lần so với các phương pháp kể trên. Không những thế bài toán “sắp xếp danh sách liên kết bằng phương pháp merge sort” có thể giúp chúng em tìm hiểu sâu hơn về môn cấu trúc dữ liệu và giải thuật. Chính vì nhứng lý do trên nhóm em chọn đề tài cho đồ án là “sắp xếp danh sách liên kết đơn bằng phương pháp merge sort”

Mục tiêu và nhiệm vụ

* Nghiên cứu và viết code để thấy rõ tác vai trò của merge sort trong danh sách liên kết đơn.
* Hiểu rõ bản chất của thuật toán, quy trình hoặc động của merge trong danh sách liên kết đơn
* Hoàn thành tất cả các quy trình của thuật toán, có thể nghiên cứu và phát triển thêm những phần liên quan đến thuật toán.

Đối tượng nghiên cứu

* Lý thuyết về cấu trúc dữ liệu trừu tượng link list
* Cơ chế hoạt động của danh sách liên kết và một số thao tác cơ bảng trên danh sách liên kết đơn
* Ngôn ngữ lập trình Visual studio C/C++ dùng để phục vụ cho việc hoàn thành bài toán.

PHẦN ĐỊNH NGHĨA

Lý thuyết về Danh sách liên kết đơn

Cũng giống như mảng danh sách liên kết đơn cũng bao gồm các phần tử nhưng có liên kết vơi nhau bởi các địa chỉ gọi tắt là con trỏ.

Mỗi phần tử của danh sách liên kết đơn là một kiểu cấu trúc chứa 2 thông tin

* Thành phần dữ liệu: lưu trữ thông tin cơ bản của phần của phần tử có kiểu dữ liệu là int, char, …. Tùy vào mục đich của bài toán.
* Thành phần liên kết: lưu trữ địa chỉ của phần tử kế tiếp nó trong danh sách liên kết, hoặc lưu trữ giá trị NULL nếu như phần tử đó là phần tử cuối cùng của danh sách liên kết đơn.
* Ta có định nghĩa tổng quát

Typedef struct Node

{

data Info; //data là kiểu dữ liệu từ ý mà người tạo muốn có để xử lý bài toán

Struct Node\* pnext;  
 };

* Một phần tử trong danh sách đơn là một biến động được yêu cầu cấp phát khi cần và danh sách đơn chính là sự liên kết các biến động này lại với nhau
* Nếu biết địa chỉ của phần tử đầu tiên trong danh sách liên kết đơn ta có thể truy xuất đến phần tứ thứ 2 dựa vào thông tin của pnext.
* Về danh sách liên kết đơn ta có 2 biến địa chỉ Node\* phead và Node \* ptail để quản lý phần tử đầu tiên và phần tử cuối cùng của danh sách liên kết đơn

## ***Lý thuyết về MERGE SORT***

Định nghĩa MERGE SORT

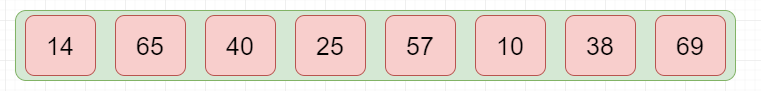
* Thuật toán sắp xếp trộn (Merge Sort) là một thuật toán sắp xếp dùng để sắp xếp các danh sách, luồng tập tin, ….. vv. theo một trật tự vào đó. Thuật toán này được xếp vào thể loại sắp xếp so sánh. Thuật toán này gần giống với thuật toán sắp xếp nhanh (Quick Sort) là tách danh sách thành 2 danh sách con, nhưng với Quick Sort là dùng phần tử nằm giữa mảng làm mốc để so sánh còn Merge Sort là chia hẳn ra thành các danh sách con nhỏ có số phần tử lớn hơn 1 và sắp xếp từng danh sách con đó rồi trộn lại.
* Sắp xếp trộn là thuật toán sắp xếp nâng cao, sử dụng để sắp xếp các danh sách, luồng tậptin… theo một trật tự nào đó. Sắp xếp trộn được xếp vào loại sắp xếp so sánh. Thuật toán này là một ví dụ tương đối điển hình cho lối thuật toán chia để trị mà John Von Neumann đưa ra vao những năm 1945.
* Sắp xếp trộn được thực hiện dựa trên việc chia danh sách ra sau đó hội những danh sách lại theo những tiêu chí của đề bài.

Tư tưởng thuật toán

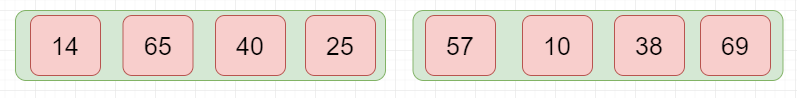
* Thuật toán Merge Sort tách danh sách liên kết cần sắp xếp thành 2 danh sách con. (first và second)
* Nếu danh sách liên kết 1 có nhiều hơn 1 phần tử thì sắp xếp danh sách liên kết con này bằng thuật toán Merge Sort tức là lại tiếp tục tách ra thành 2 mảng con nữa.
* Nếu danh sách liên kết 2 có nhiều hơn 1 phần tử thì sắp xếp danh sách liên kết con này bằng thuật toán Merge Sort tức là lại tiếp tục tách ra thành 2 mảng con nữa.
* Trộn 2 danh sách liên kết con đã sắp xếp lại với nhau cứ trộn đến khi nào số phần tử của danh sách sau khi sắp xếp bằng số phần tử danh sách ban đầu. Khi này danh sách của ta đã được sắp xếp tăng dần.

### Ví dụ hình dung về cách giải thuật toán sắp xếp Merge Sort

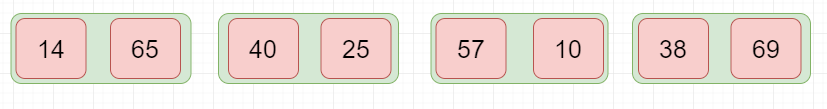
* Đây là danh sách đầu tiên.



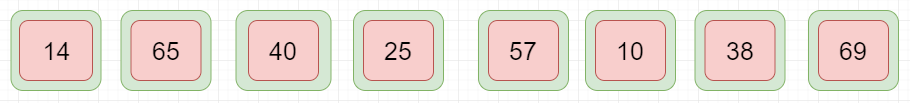
* Đầu tiên, giải thuật sắp xếp trộn chia danh sách thành hai nửa. Tiến trình chia này tiếp tục diễn ra cho đến khi danh sách chỉ còn 1 phàn tử. Trong hình dưới, đầu tiên chúng ta chia mảng kích cỡ 8 thành hai mảng kích cỡ 4.



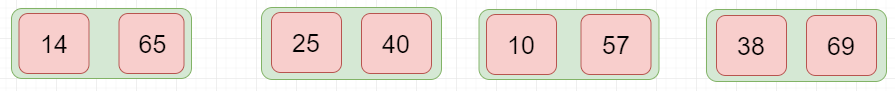
* Tiến trình chia này không làm thay đổi thứ tự các phần tử trong danh sách ban đầu. Bây giờ chúng ta tiếp tục chia các danh sách này thành 2 nửa.



* Tiến hành chia tiếp cho tới khi không còn chia được nữa.



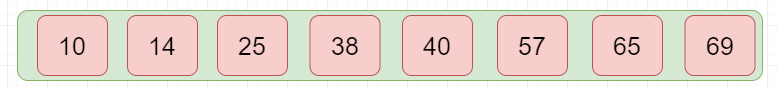
* Bây giờ chúng ta trộn chúng theo như đúng cách thức mà chúng được chia ra.
* Đầu tiên chúng ta so sánh hai phần tử trong mỗi danh sách và sau đó tổ hợp chúng vào trong một danh sách khác theo cách thức đã được sắp xếp. Ví dụ, 14 và 65 là trong các vị trí đã được sắp xếp. Chúng ta so sánh 25 và 40 và trong danh sách khác chúng ta đặt 25 ở đầu và sau đó là 40. Tương tự, chúng ta thay đổi vị trí của 10 và 57, 38 và 69 thì giữ nguyên.



* Vòng lặp tiếp theo là để kết hợp từng cặp danh sách một ở trên. Chúng ta so sánh các giá trị và sau đó hợp nhất chúng lại vào trong một danh sách chứa 4 giá trị, và 4 giá trị này đều đã được sắp thứ tự.



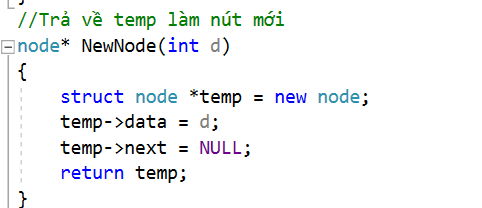
* Sau bước kết hợp cuối cùng, danh sách sẽ trông giống như sau:



PHẦN ĐỒ ÁN

Một Số Hàm Khởi tạo

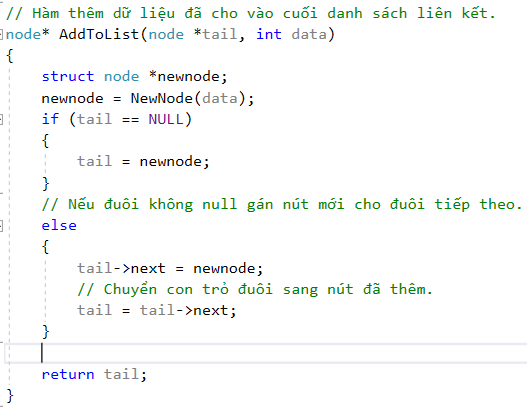
Hàm tạo một node mới.



* Hàm sẽ trả về một Node mới với thông tin dữ liệu là một số nguyên. Ban đầu ta xem như danh sách chưa có một phần tử nào muốn có phần tử để tiến hành sắp xếp ta phải khởi tạo 1 Node mới bằng cách khai báo 1 struct node\*temp. chúng ta có thể dùng con trỏ để trỏ đến data của nó gán nó bằng giá trị d mà mình tạo từ hàm ngẫu nhiên ở trên.
* Chúng ta xem như phần tử vừa tạo là 1 phần tử đơn nên temp 🡪 next =NULL
* Sau khi làm xong ta trả về giá trị của biến temp.

Hàm thêm Node vào danh sách liên kết đơn

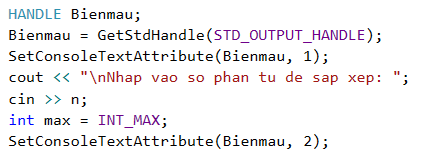
* Ở đây chúng em thực hiện hàm thêm Node vào cuối danh sách.



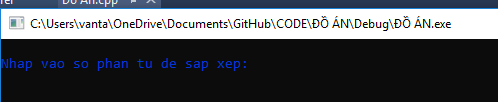
* Ở Hàm thêm Node vào cuối danh sách liên kết đơn đầu tiên chúng ta phải gọi lại hàm khởi tạo Node ở trên để ta có 1 Node hoàn chỉnh.
* Khi đã có Node ta tiến hành thêm Node vào cuối danh sách bằng cách kiểm tra xem danh sách đó có rỗng hay không nếu rỗng thì xem như khi thêm Node vừa mới khởi tạo vào mặc nhiên nó là phần tử duy nhất trong danh sách và nó nằm ở vị trí cuối cùng trong danh sách (tail).
* Ngược lại nếu trong danh sách đã có ít nhất 1 phần tử thì ta tiến hành gán con trỏ next của phần tử cuối cùng trong danh sách đến biến con trỏ next của phần tử ta muốn thêm vào.
* Tiếp theo ta gán lại biến con trỏ tail vào Node đã thêm. Và cuối cùng là trả về giá trị của Node mới thêm vào tức là tail.
* Kết quả là Node vừa mới thêm vào sẽ có giá trị là data nằm cuối cùng danh sách và biên tail 🡪next sẽ là NULL.

Hàm tạo màu bổ Trợ ( không bắt buộc phải có)

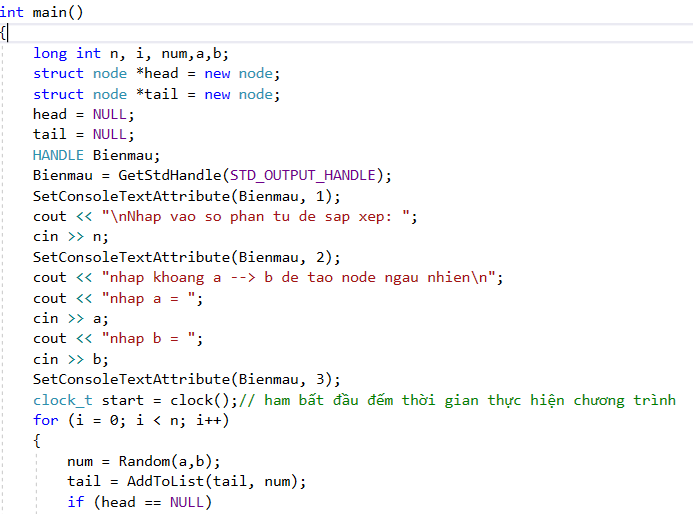
* Để làm được code màu trước hết ta phải khai báo thư viện windows.h thư viện này cho phép ta có thể sử dụng một số code màu.

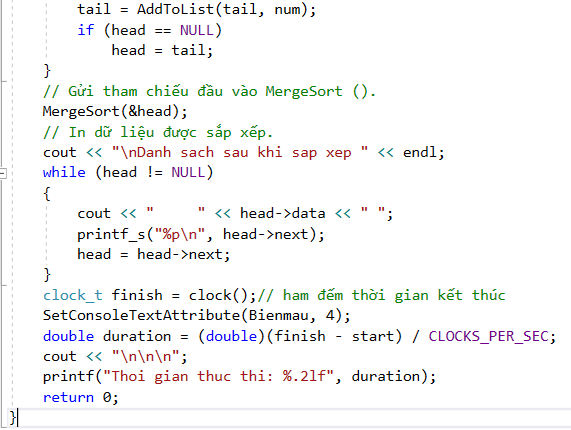


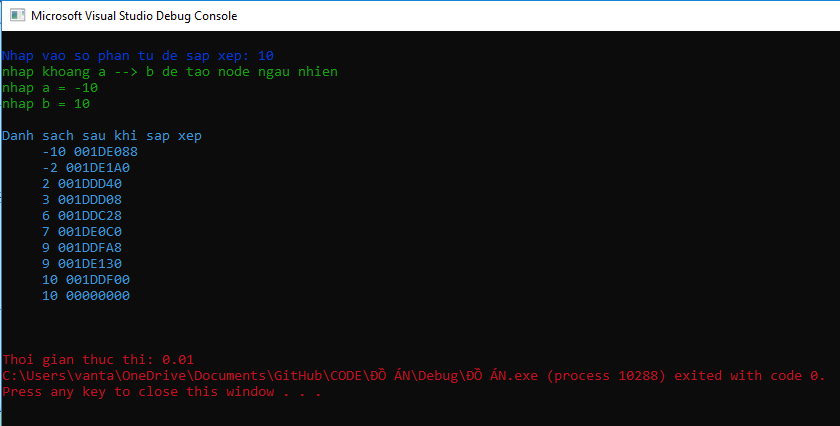
* Ta khai báo một biến màu Bienmau có kiểu dữ liệu HANDLE
* Gọi hàm GetStdHandle(STD\_OUTPUT\_HANDLE) này cho phép ta sử dụng 255 màu có sẵn trong thư viện Windows
* Tiếp đến ta gọi hàm SetConsoleTextAttribute để tô màu cho những dòng chữ xuất ra ở đây chọn màu 1 để in dòng chữ “Nhap vao so phan tu de sap xep:”



Hàm main và màn hình console



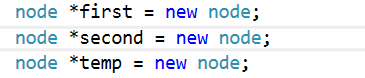




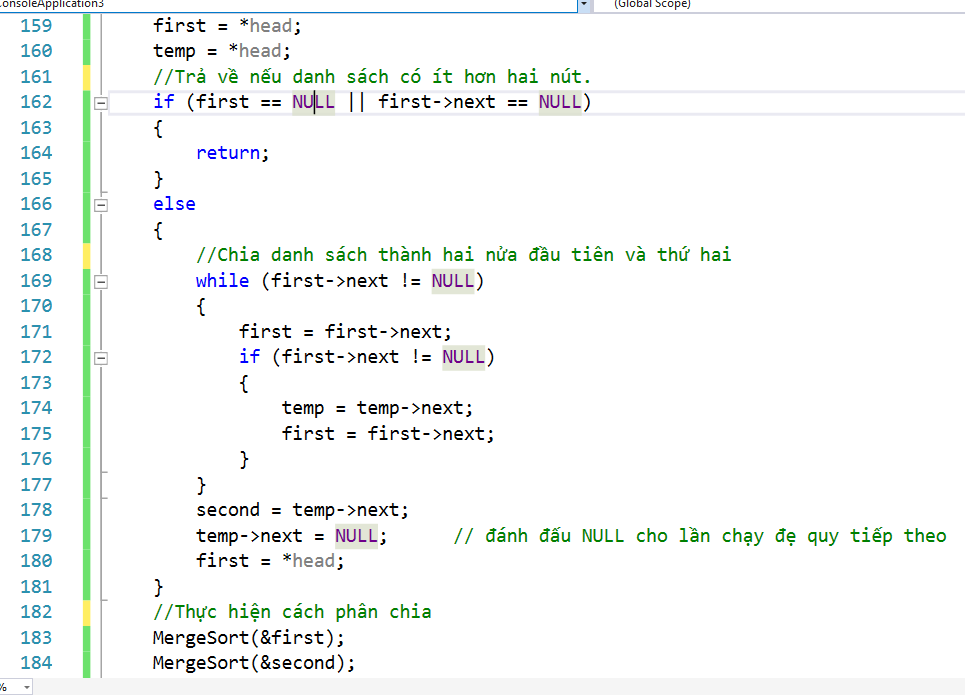
## ***Phân tích thiết kế thuật toán.***

Hàm thực hiện hợp nhất danh sách liên kết bằng cách sử dụng tham chiếu hàm MergeSort(node \*\*head)

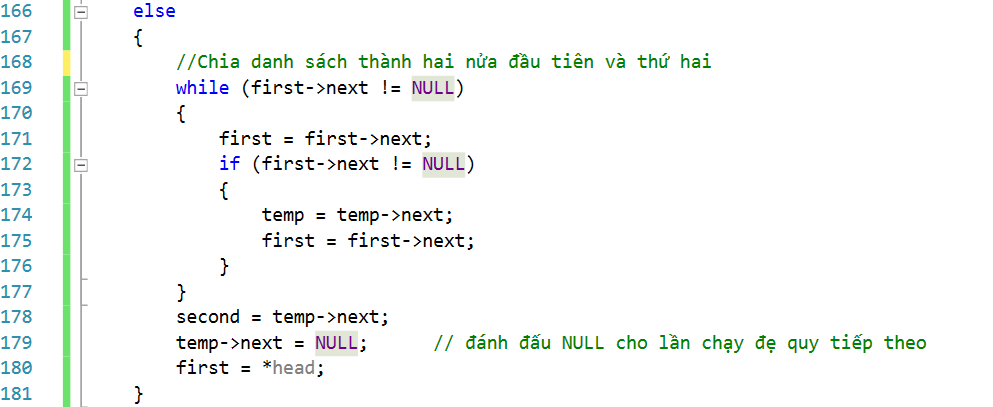
* Ta khai báo 3 danh sách mới, tại vì ta phải tách danh sách đã tạo thành 2 danh sách con và tách đến khi nào mà có ít hơn 2 phần tử trong danh sách.



* node \*first là khao báo con trỏ thứ nhất.
* node \*second là khao báo con trỏ thứ hai.
* node \*temp là con trỏ đánh dấu nằm giữa danh sách tạo ban đầu.



* Ở đây ta thấy ban đầu gán cho con trỏ 1(first) và con trỏ đánh dấu giữa danh sách (temp) là phần tử đầu của danh sách chính.
* Tiếp đến so điều kiện nếu mà phát hiện danh sách mà có ít hơn 2 phần tử thì ta dừng lại.
* Còn có nhiều hơn 2 phần tử thì ta phân tách thành 2 danh sách



* Ta phân tích hàm này:

Ví dụ ta cho một danh sách là:

1 5 9 7 8

- Giờ ta chạy tay:

Lúc này con trỏ first được gán bằng Head là bằng 1.

Giờ vào vòng while thấy first -> next không bằng NULL

Giờ thì gán first bằng con trỏ kế tiếp có data là 5

Ta lại so sánh và thấy địa chỉ của phần tử kế tiếp khác NULL

Nên ta lại chạy vào và gán con trỏ temp bằng data con trỏ kế tiếp của temp là bằng 5 vì lúc này temp bằng phần tử đầu.

Lại một lần nữa ta gán first bằng con trỏ kế tiếp có data là 9

Quay lại vong lặp ta kiểm tra thấy chưa đi hết danh sách nên ta chạy lại while và

Ta gán first bằng con trỏ kế tiếp có data là 7

Ta lại so sánh và thấy địa chỉ của phần tử kế tiếp khác NULL

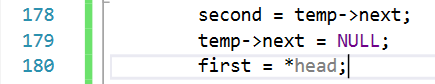
Nên ta lại chạy vào if và gán con trỏ temp bằng data con trỏ kế tiếp của temp (temp = 5) là bằng 9

Ta gán first bằng con trỏ kế tiếp có data là 8

Và đến đây ta thấy first -> next = NULL vi phạm nên thoát vòng lặp while

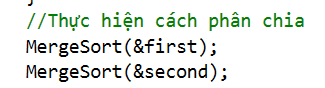
Xong hàm này ta đã tìm được phần tử giữ danh sách là temp

Việc này cũng sẽ tìm được phần tử giữ danh sách nếu có nhiều phần tử hơn mỗi lần lặp lại while thì con trỏ first sẽ cách con trỏ temp 2n. vì vậy ta có thể tìm thấy phần tử giữa của danh sách dễ dàng.



Ra khỏi while ta gán con trỏ thứ 2(second) bằng phần tử kế tiếp của phần tử giữ danh sách ( temp ) để đánh dấu phần tử đầu của danh sách thứ 2 là second để chạy đệ quy việc phân chia danh sách thứ 2 thành 2 phần nữa.

Ta thấy temp -> next =NULL để đánh dấu kết thúc danh sách đầu là phần tử này.

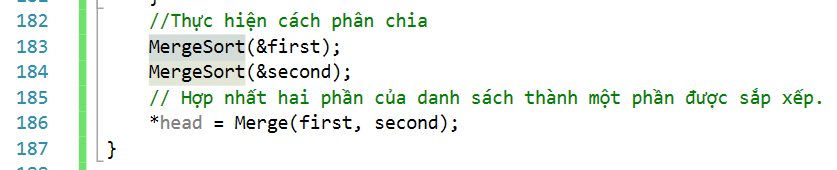


Ta gán first bằng phần tử đầu để chạy đệ quy lại phần danh sách thứ nhất

Ta cứ lặp lại hàm MergeSort đến khi nào mà danh sách có ít hơn 2 phần tử tức là danh sách còn 1 phần tử nhảy ra.

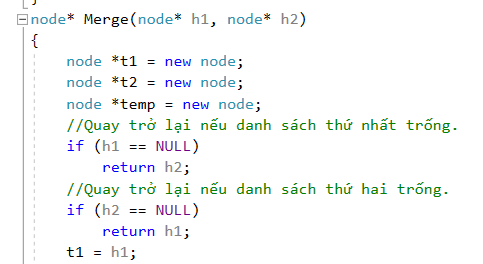
Sau khi chạy đệ quy hết danh sách 1 thì ta chạy đệ quy danh sách 2 ta cũng lặp lại hàm MergeSort đến khi nào mà danh sách có ít hơn 2 phần tử tức là danh sách còn 1 phần tử thì nhảy ra.

Lúc này thì ta đã phân rã danh sách đầu thành n danh sách con

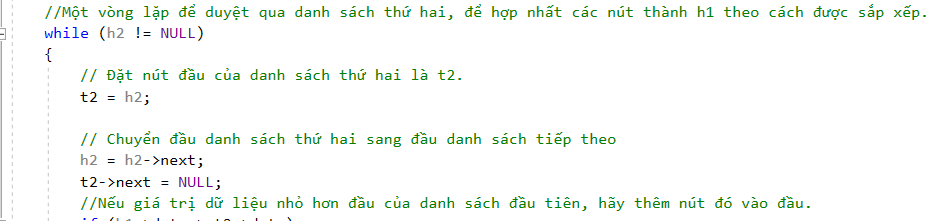


Giờ ta chạy vào hàm tập hợp và sắp xếp lại tăng dần.

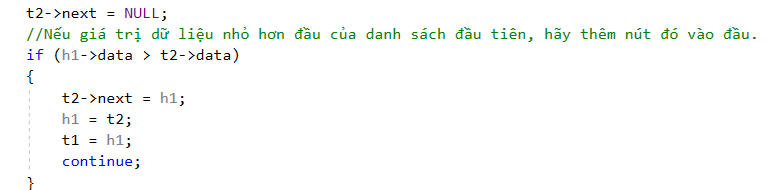
Hàm phân hoạch node\* Merge(node\* h1, node\* h2)

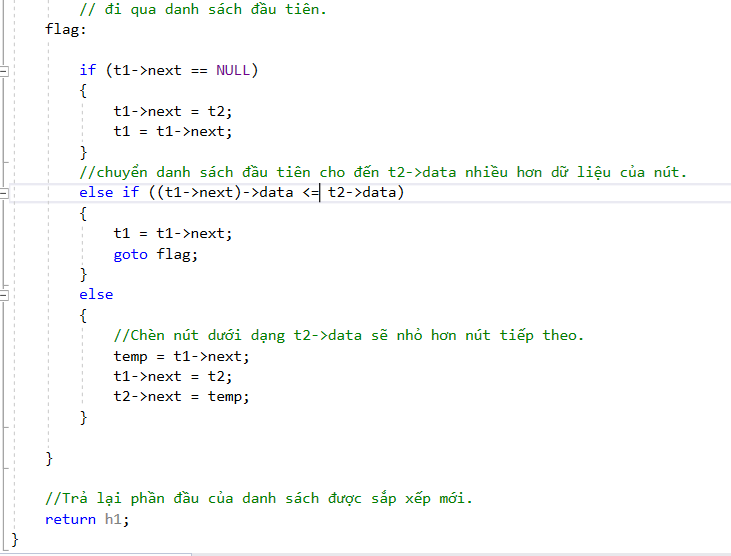


* Hàm phân hoạch gồm có hai danh sách là h1 và h2 mục đích của 2 danh sách này là để phân danh sách ban đầu chưa sắp xếp thành 2 danh sách con h1 và h2.
* Nội dung hàm Merge ta khai báo mới 2 danh sách mới nữa là t1 và t2 danh sách này hoàn toàn mới rỗng. Ban đầu ta kiểm tra xem h1 có rỗng hay không nếu rỗng thì mình sẽ trả về danh sách h2 để tiếp tục bước tiếp theo.
* Tiếp tục nếu danh sách h1 đã rỗng thì mình kiểm tra xem danh sách h2 có rỗng không nếu danh sách h2 cũng rỗng thì ta trả về danh sách h1 và sau đó gán danh sách t1=h1.



* Ở đây chúng ta sử dụng 1 vòng lặp While với điều kiện dừng là khi danh sách h2 đã rỗng.
* Ta đặt Node đầu tiên của danh sách h2 là t2. Đó ta chuyển đầu danh sách h2 sang danh sách tiếp theo. Để thực hiện điều đó ta thực hiện chuyển địa chỉ trong h2 sang next và gán t2🡪next = NULL xem như danh sách t2 vào cuối danh sách h2. Và địa chỉ là NULL.





* Ta kiểm tra xem giá trị có trong danh sách h1 có giá trị như thế nào với giá trị t2. Nếu giá trị trong h1🡪data lớn hơn t2🡪data thì ta sẽ thực hiện gán địa chỉ con trỏ của h1 sang cho t2. Chỗ này quan trọng t2🡪next=h1. Đầu tiên nó sẽ nhận diện chỉ sau khi so sánh 2 cái dữ liệu data bên trên. Nó sẽ đổi địa chỉ next trước sau đó mới đổi chỗ data.\* Ở đây lệnh continue chương trình sẽ quay lại từ đầu thực hiện đến khi điều kiện sai nó sẽ qua bước tiếp theo. Continue có tác dụng là đảm bảo các Node trong danh sách h2 được sắp xếp hoàn toàn.
* Flag là một biến cầm canh để thực hiện lệnh goto. Khi gặp goto thì nó sẽ trở lại dòng lệnh cầm canh flag.
* Ở đây ta kiểm tra xem địa chỉ của Node của t1 có bằng NULL hay không tức là t1 có phải là phần tử cuối cùng hay không nếu đã là phần tử cuối cùng.thì khi thêm phần tử t2. Lấy địa chỉ của phần tử t1 gán cho t2. Và có nghĩa là t2 lúc này là phần tử cuối cùng của danh sách và t1 lúc này sẽ có địa chỉ mới là next.
* Ngược lại nếu phía sau phần tử t1 có 1 Node khác thì ta sẽ so sánh xem giá trị của t2 ta định phân hoạch vào có lớn hơn hay bằng giá trị nằm sau t1 hay không nếu lớn hơn hoặc bằng thì ta vẫn giữ địa chỉ của t1. Và thực hiện lệnh goto flag.
* Lệnh goto flag ; có nghĩa là ta sẽ quay lên dòng lệnh có biến cờ hiệu để thực hiện.lệnh goto sẽ dừng khi điều kiện ràng buộc không còn đúng nữa.
* Trường hợp cuối cùng là khi mà node t2 thêm vào có giá trị nhỏ hơn giá trị của t1 thì đầu tiện ta sẽ dùng một node trung gian để lữu lại địa chỉ của t1 sau đó ta tiến hành hoán vị địa chỉ 2 node này. Khi hóa vị 2 địa chỉ thì giá trị của nó tự thay đổi.
* Kết thúc vòng while ta trả về danh sách h1. Danh sách đã được sắp xếp trong lúc phân hoạch

## ***4.3 Mô tả công việc và Nhận xét đồ án***

|  |  |  |
| --- | --- | --- |
| Tên Sinh Viên | Đánh Giá Chung Phần Trăm đóng góp | Mô tả khái quát công việc |
| * Nguyễn Văn Tây * Võ Tấn Nguyên | 100% | tìm hiểu danh sách liên kết các thuật toán cơ bản, tham khảo code liên quan đên mergesort, lên ý tưởng cho bài toán. |
| * Nguyễn Văn Tây * Võ Tấn Nguyên | 100% | tham khảo code trên internet,thiết kế thuật toán, lên danh sách các hàm quan trọng để làm trong bài toán |
| * Võ Tấn Nguyên | 100% | khởi tạo danh sách, sinh ngẫu nhiên các node trong danh sách để sắp xếp, debug chạy thử |
| * Nguyễn Văn Tây | 100% | tạo Node mới và thêm Node vào danh sách liên kết |
| * Nguyễn Văn Tây * Võ Tấn Nguyên | 100% | tách danh sách liên kế vừa tạo ngẫu nhiên ra thành các dãy có giá trị tăng dần, phân hoạch chúng vào 2 danh sách. |
| * Nguyễn Văn Tây | 100% | viết hàm phân hoạch cho merge sort luân phiên tăng giảm |
| * Võ Tấn Nguyên | 100% | viết hàm để hợp nhất các dãy liên kết đã phân hoạch. Trộn lại để ra danh sách liên kết tăng |
| * Võ Tấn Nguyên | 100% | viết 1 vòng lặp để có thể duyệt danh sách thứ 2 để hợp các node thành 1 danh sách |
| * Nguyễn Văn Tây | 100% | duyệt qua danh sách đầu tiên để chèn node nhỏ hơn vào danh sách. |
| * Nguyễn Văn Tây * Võ Tấn Nguyên | 100% | liên kết các hàm đã làm lại ở hàm main. Chạy thử chương trình.-> debug lỗi. |
| * Nguyễn Văn Tây * Võ Tấn Nguyên | 100% | xem xét các lỗi lần lượt sửa lỗi để hoàn thiện |
| * Nguyễn Văn Tây * Võ Tấn Nguyên | 100% | chạy chương trình lần cuối, thử tất cả các trường hợp xảy ra.--> hoàn thiện code |
| * Nguyễn Văn Tây * Võ Tấn Nguyên | 100% | Viết báo cáo bằng file word, làm power point chương trình |
| * Nguyễn Văn Tây * Võ Tấn Nguyên | 100% | tổng hợp kiểm duyệt mọi thứ trước khi kết thúc đề tài |

Kết Luận

## ***Nhận xét:***

* Cơ bản nhóm đã hoàn thành mục tiêu của đồ án. Đồ án hoàn thành 100% mục tiêu đề ra.
* Nhưng bên cạnh đó Nhóm cũng gặp một số khó khăn trong quá trình thực hiện đồ án.

+ Thời gian hoàn thành đồ án không được hoàn thành như thời gian kế hoạch đề ra.

+ Trong quá trình làm Nhóm không thể in ra địa chỉ con trỏ để thấy rõ sự sắp xếp merge sort được thực hiện trên con trỏ địa chỉ ID.Nhưng nhóm cũng đã tìm ra cách và khắc phục.

+ Trong quá trình làm trong môi trường github khi làm 2 laptop riêng thì 2 laptop khác fame work nên code của 2 thành viên mỗi lần commit thì người commit thì chạy được chương trình còn người kia thì không ( dù là chương trình đã đúng hoàn toàn)

* Nhóm sẽ có gắng khắc phục lỗi fame work bằng cách là cập nhật famework mới để có thể thuận tiện trong các thao tác nhóm.
* Ý tưởng để phát triển đồ án là có thể vận dụng C# hoặc graphics để có thể tạo giao diện giúp người đọc hình dung rõ hơn về thuật toán.
* Ưu điêm: Thuật toán chạy rất nhanh với số lượng phần tử lớn, và rất ổn đình
* Nhược điểm: Thuận toán tốn bộ nhớ phụ.

## ***Tài Liệu Tham Khảo.***

<https://cachhoc.net/2015/03/04/thuat-toan-sap-xep-tron-merge-sort/>

<https://vi.wikipedia.org/wiki/S%E1%BA%AFp_x%E1%BA%BFp_tr%E1%BB%99n>

<https://ilaptrinh.wordpress.com/2013/01/14/merge-sort/>