Đại học Quốc gia Hà Nội

**Trường Đại học Công Nghệ**

Báo cáo Bài tập lớn về:

**DANH SÁCH LIÊN KẾT ĐƠN**

**(SINGLY LINKED LIST)**

**Prepared by:**

Tên: Hoàng Ngọc Khôi

MSSV: 17021160

Lớp: INT1006 3

Hà Nội, ngày 15/04/2019

Link code: github.com/hoangkhoi-uet/C

Mục lục:

[**I.** **Mở đầu:** 3](#_Toc7906336)

[**II.** **Ứng dụng Danh sách liên kết:** 3](#_Toc7906337)

[**III.** **So sánh với Mảng:** 4](#_Toc7906338)

[**IV.** **Cấu tạo:** 4](#_Toc7906339)

[**1.** **Hàm deleteNode(Hàm này sẽ xóa 1 Node ở vị trí được chọn index):** 8](#_Toc7906340)

[**2.** **Hàm addNode(Hàm này thêm 1 Node vào vị trí được chọn với giá trị):** 9](#_Toc7906341)

[**3.** **Hàm printNode(Hàm này in danh sách các Node theo thứ tự từ head):** 10](#_Toc7906342)

[**4.** **Một số test case cho chương trình:** 10](#_Toc7906343)

1. **Mở đầu:**

* Danh sách liên kết đơn(Singly Linked List) là một dạng **lưu trữ dữ liệu một chiều** gồm các “mắt xích”(Node) nối liên tiếp với nhau.

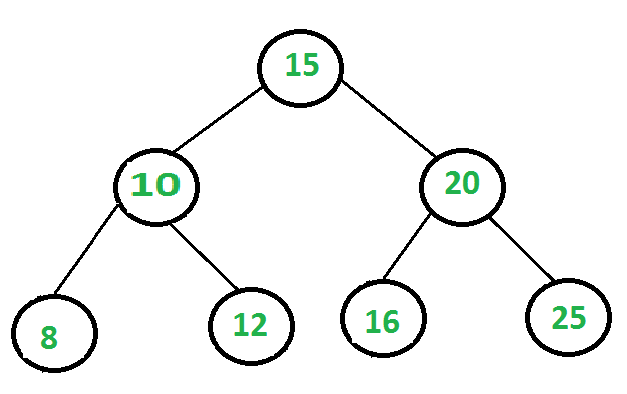


* Còn 2 kiểu danh sách liên kết nữa là danh sách liên kết đôi(hai chiều) và danh sách liên kết vòng(đầu và cuối nối với nhau thành vòng tròn).

1. **Ứng dụng Danh sách liên kết:**

* Lưu trữ dữ liệu dưới dạng tuyến tính giống như mảng. Nhưng **có thể thêm, xóa các phần tử dễ dàng hơn mảng**, do mảng bị cố định về kích thước.
* Do **tính chất dễ dàng thêm, xóa** nên danh sách liên kết có thể dùng triển khai:
* Queue(hàng đợi): Vào trước thì ra trước. Giống như việc xếp hàng để tính tiền trong siêu thị:
* Stack(Ngăn xếp): Vào càng sớm thì ra càng sớm. Giống như việc nhét những quả bóng vào 1 cái lọ dài.

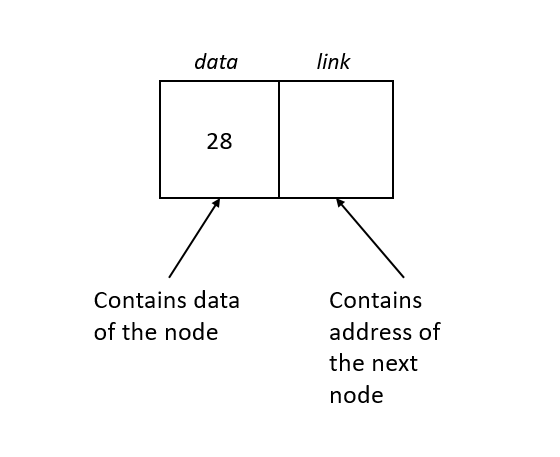
 và

* Binary Tree(Cây nhị phân): Mỗi nút của cây lưu 1 giá trị.

1. **So sánh với Mảng:**

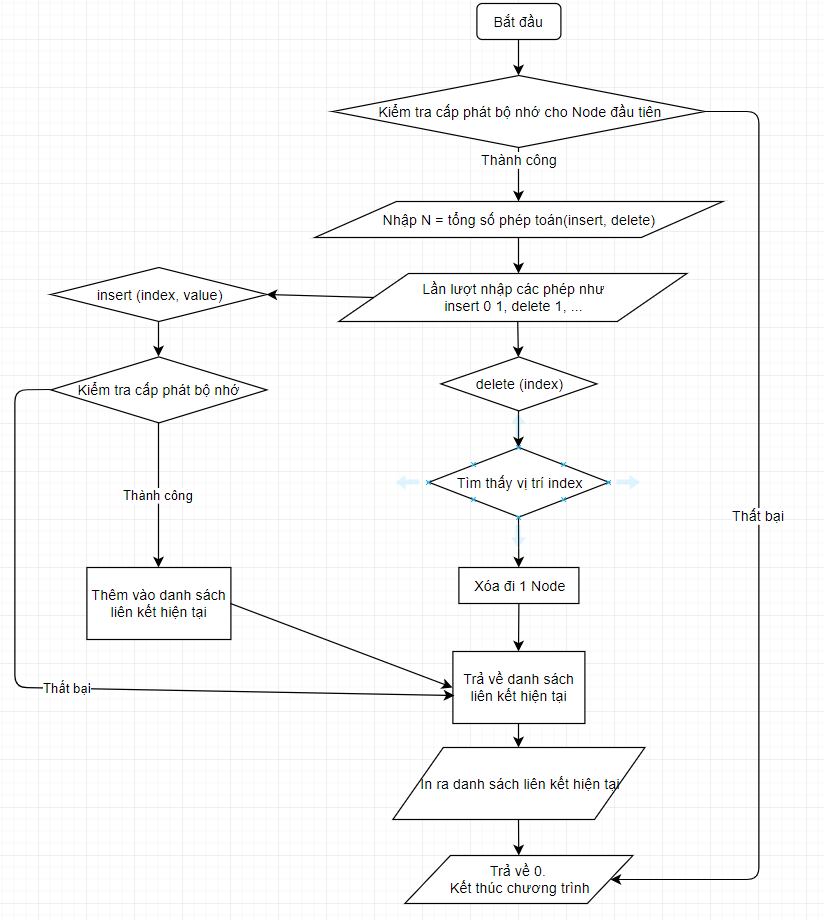
|  |  |
| --- | --- |
| Mảng | Danh sách liên kết |
| Kích thước cố định, cần khai báo rõ trước khi sử dụng. | Kích thước không cố định, có thể tùy ý thay đổi thêm xóa. |
| Lưu trữ trong các ô nhớ xếp liên tục cạnh nhau. | Có thể lưu trữ ở bất kỳ ô nhớ nào trống, tuy nhiên cần thêm bộ nhớ cho con trỏ. |
| Truy cập bằng chỉ số mảng. | Duyệt từng phần tử để truy cập tuyến tính. |

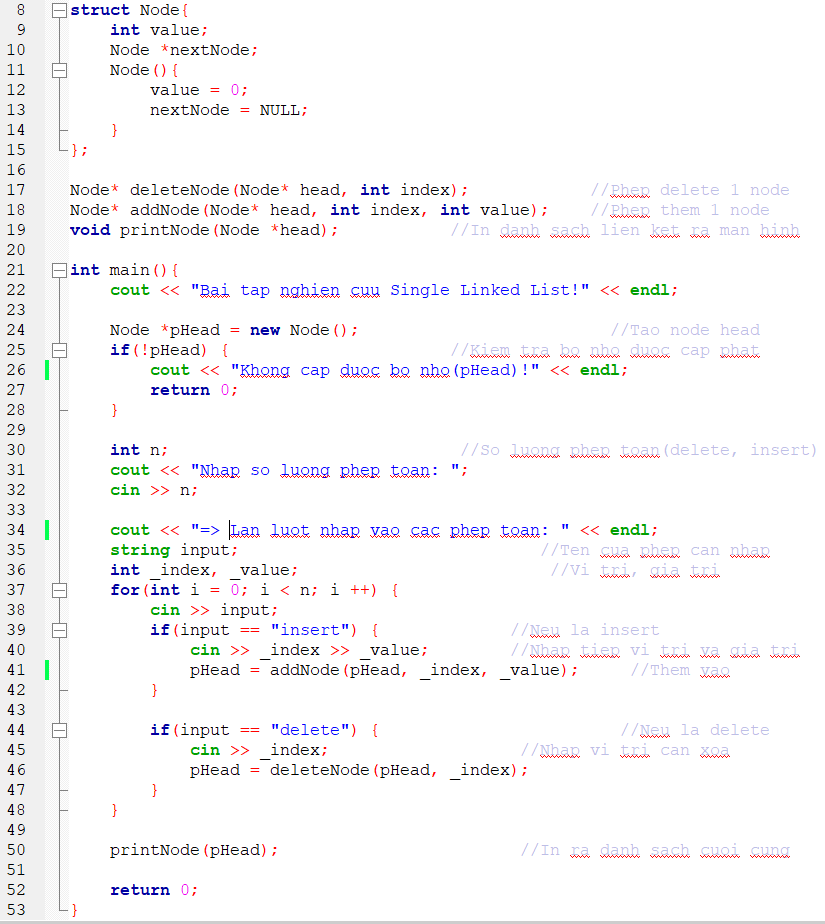
1. **Cấu tạo:**



* Mỗi “mắt xích”(Node) sẽ gồm 2 thành phần: Dữ liệu(Data) và **con trỏ** lưu **địa chỉ** của Node tiếp theo. Node đầu tiên sẽ gọi là Head, node cuối cùng trỏ NULL. Chúng ta sẽ quy định Node đầu tiên là bắt đầu từ 0, rồi đến 1, 2, 3...
* Dưới đây là 1 chương trình cơ bản mô tả Singly Linked List:

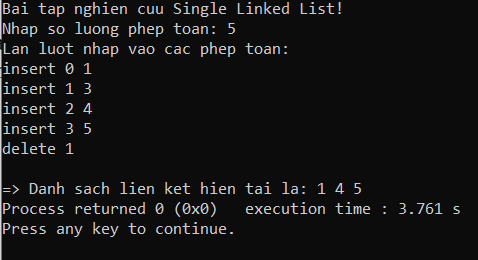
Về sơ đồ khối của chương trình:





Trong đó:

* Node mô tả bằng struct gồm 2 thành phần: **data**(kiểu int) và **nextNode**(con trỏ kiểu Node).
* Hàm khởi tạo mặc định 1 Node có: **data = 0** và **nextNode = NULL**.
* Các hàm khác: xóa 1 Node bất kỳ(deleteNode), thêm 1 Node vào vị trí bất kỳ(addNode).
* Hàm main():
* Khởi tạo 1 Node pHead(chính là Node head đầu tiên).
* Số phép toán n, bao gồm phép deleteNode và addNode.
* Đầu vào phép toán input, là viết tắt của phép, đó là “insert” và “delete”:

Ví dụ như:

* Ở đây có tất cả 5 phép toán:4 insert và 1 delete.
* Phép “insert” thì số đầu tiên là vị trí cần thêm Node trong linked list, số thứ 2 là dữ liệu cần ghi vào Node đó. Ở trên ta cần “insert” 1 linked list hoàn chỉnh gồm các vị trí từ 0 đến 3.
* Phép “delete” theo sau là vị trí của Node cần xóa.

insert 0 1 => Danh sách liên kết gồm: 1

insert 1 3 => Danh sách liên kết gồm: 1 3

insert 2 4 => Danh sách liên kết gồm: 1 3 4

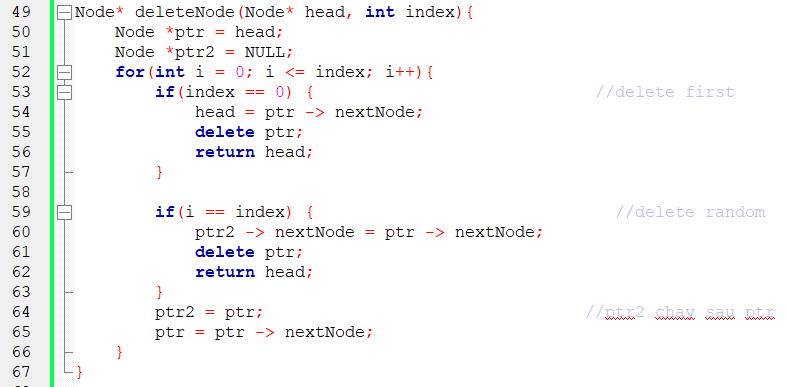
insert 3 5 => Danh sách liên kết gồm: 1 3 4 5

delete 1 => Danh sách liên kết gồm: 1 4 5

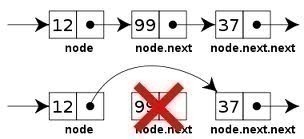
* printNode(pHead): in ra danh sách liên kết cuối cùng với con trỏ pHead.

Tiếp theo, chúng ta cùng giải thích rõ chức năng các hàm:

1. **Hàm deleteNode(Hàm này sẽ xóa 1 Node ở vị trí được chọn index):**

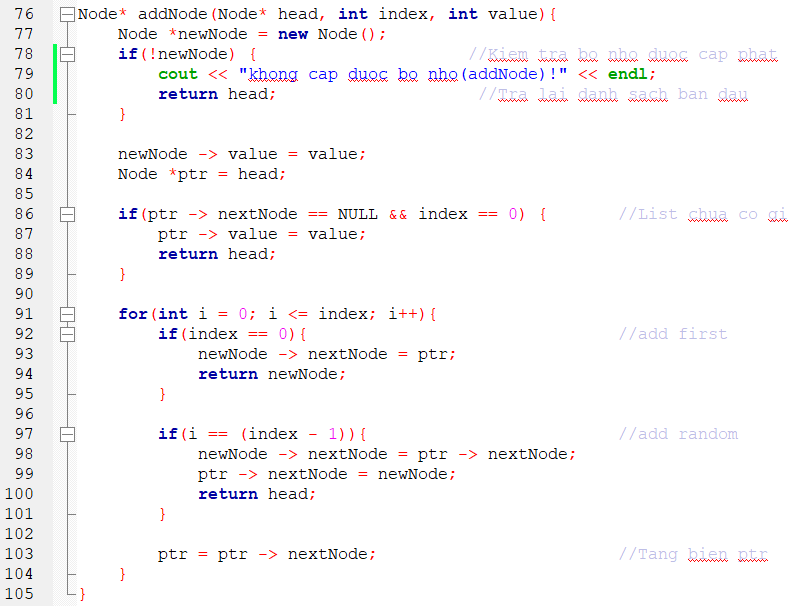


Mô tả thuật toán: Để xóa Node i thì sẽ tạo 1 con trỏ tương ứng Node đó, kết nối Node i - 1(trước) và i + 1(sau), rồi mới xóa Node i đó đi.

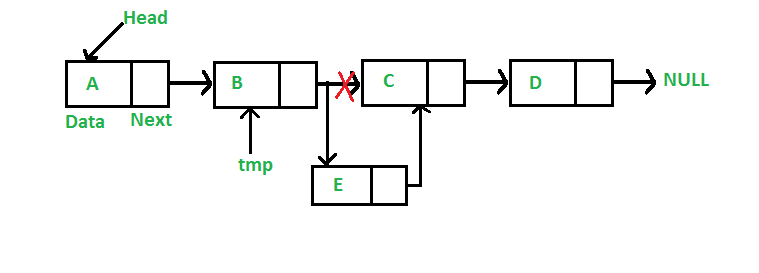


* Tham số truyền vào hàm bao gồm con trỏ Node \*head(Node đầu tiên) và index(vị trí Node cần xóa).
* Tiếp theo tạo 2 con trỏ Node gồm:
* Node \*ptr cùng trỏ đến head.
* Node \*ptr2 sẽ chạy sau ptr 1 Node tức là luôn đứng trước ptr 1 Node.
* Tiếp theo chúng ta sẽ duyệt tuyến tính từng phần tử của Node từ vị trí đầu tiên 0 đến vị trí index. Khi đến được vị trí index:
* Nếu vị trí đó là 0, tức là head, thì chỉ cần di chuyển head sang vị trí 1 rồi xóa con trỏ ptr(head lúc đầu). Trả về head.
* Nếu vị trí đó >0, thì ta nối con trỏ trước đó ptr2 với con trỏ sau nó ptr -> nextNode. Rồi xóa con trỏ vị trí đó ptr. Trả về head.

1. **Hàm addNode(Hàm này thêm 1 Node vào vị trí được chọn với giá trị):**

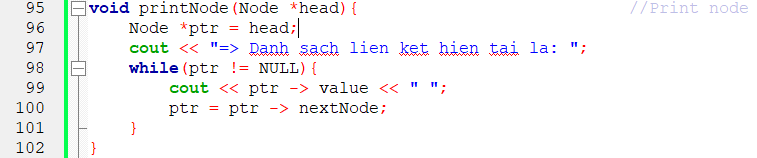


Mô tả thuật toán: Để thêm Node i vào vào vị trí index thì ta khởi tạo 1 newNode, gán giá trị cho nó -> value rồi nối Node i – 1 và Node i + 1 với Node i đó



* Tham số truyền vào là Node \*head, vị trí (index) và giá trị(value).
* Tạo con trỏ Node \*ptr trỏ đến head.
* Tạo Node mới với giá trị là value.
* Trường hợp danh sách rỗng thì ta gán ngay giá trị cho head. Trả về head.
* Trường hợp đã có ít nhất 1 phần tử:
* Nếu vị trí index là 0, tức là chèn vào trước head, thì chỉ cần nối newNode và head. Rồi trả về newNode(tức là head mới).
* Nếu Node hiện tại tiếp theo là vị trí index, thì nối newNode với node tiếp theo đó. Rồi nối Node hiện tại.

1. **Hàm printNode(Hàm này in danh sách các Node theo thứ tự từ head):**



* Đầu tiên tạo Node \*ptr trỏ đến head.
* Duyệt tuyến tính từ phần tử 0 sao cho ptr khác NULL rồi in ra giá trị của các Node đó.

1. **Một số test case cho chương trình:**

|  |  |  |
| --- | --- | --- |
| Test case1: | Test case 2: | Test case 3: |
| 5  insert 0 1  insert 1 5  insert 2 17  insert 3 4   * 1 5 17 4   delete 2   * 1 5 4 | 5  insert 0 1  insert 1 5  insert 2 17  insert 3 4   * 1 5 17 4   delete 0   * 5 17 4 | 6  insert 0 1  insert 1 5  insert 2 17  insert 3 4  insert 0 3   * 3 1 5 17 4   delete 3   * 3 1 5 4 |

Kết thúc tài liệu.