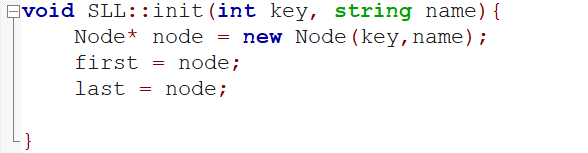
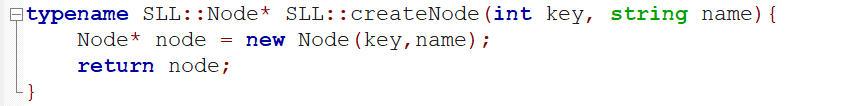
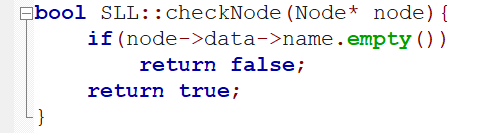
1. Giải thích ý nghĩa các hàm



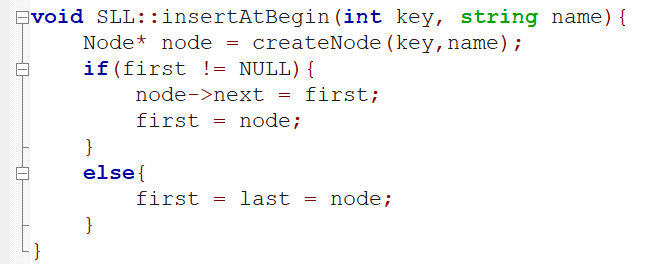
- Hàm init được dùng để khởi tạo danh sách với node đầu tiên có key và name được truyền vào. Con trỏ đầu và cuối của danh sách liên kết đơn đều được trỏ vào node đó.



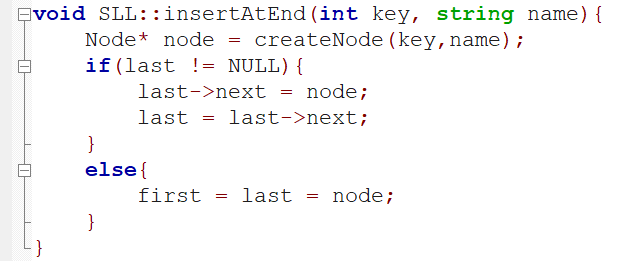
- Hàm createNode được sử dụng để tạo 1 node từ key và name được truyền vào. Hàm này trả về con trỏ trỏ vào node vừa được khởi tạo.



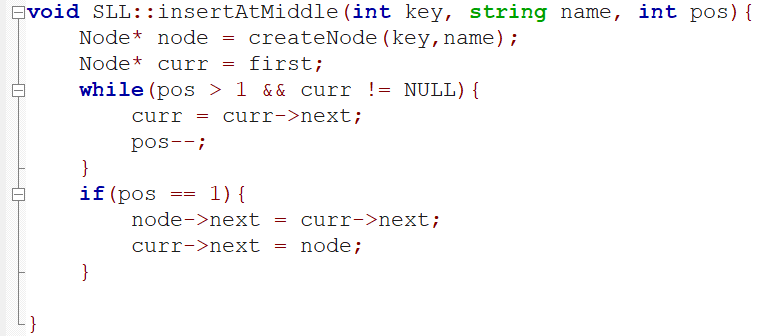
- Hàm checkNode được sử dụng để kiểm tra 1 node có phải node trống (không có name – dữ liệu của node) hay không. Giá trị trả về false có nghĩa node được kiểm tra là node trống, ngược lại nếu trả về true cho thấy node đó không phải node trống.



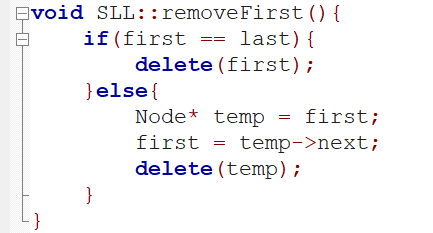
- Hàm insertAtBegin sẽ chèn 1 node vào đầu danh sách. Trước hết node chứa key và name được truyền vào sẽ được khởi tạo. Sau đó, nếu con trỏ đầu danh sách không rỗng thì thành phần next của node vừa tạo sẽ cùng trỏ node đầu tiên của danh sách, rồi con trỏ first sẽ trỏ lại vào node vừa được tạo. Ngược lại, nếu con trỏ đầu rỗng thì node vừa tạo chính là node đầu tiên của danh sách, con trỏ first và last sẽ cùng trỏ vào node đó.



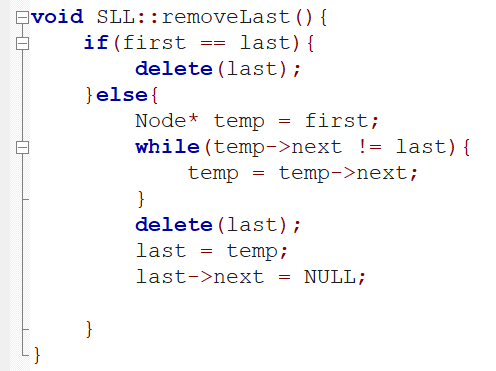
- Hàm insertAtEnd chèn 1 node vào cuối danh sách. Trước hết node chứa key và name được truyền vào sẽ được khởi tạo. Sau đó kiểm tra con trỏ last của danh sách. Nếu nó khác rỗng thì thành phần next của node cuối cùng sẽ trỏ vào node vừa được tạo, rồi con trỏ last sẽ trỏ vào node được tạo. Ngược lại, nếu con trỏ last rỗng thì node vừa tạo chính là node đầu tiên của danh sách, con trỏ first và last sẽ cùng trỏ vào node đó.

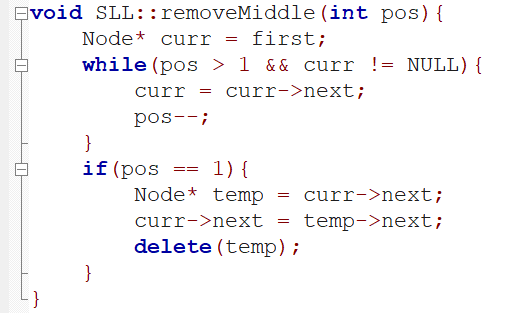


- Hàm insertAtMiddle chèn 1 node vào sau vị trí pos được truyền vào. Trước hết ta khởi tạo node cần chèn với key và name có được. Đặt con trỏ curr trỏ vào đầu danh sách. Ta sẽ di chuyển dần con trỏ curr về cuối danh sách chừng nào pos > 1 và curr != NULL. Mỗi lần dịch chuyền ta giảm pos đi 1 đơn vị. Kết thúc vòng lặp, nếu pos = 1 tức là ta đã đến đúng node đứng trước node cần chèn. Ta cho node curr liên kết đến node cần chèn, node cần chèn liên kết đến curr->next.

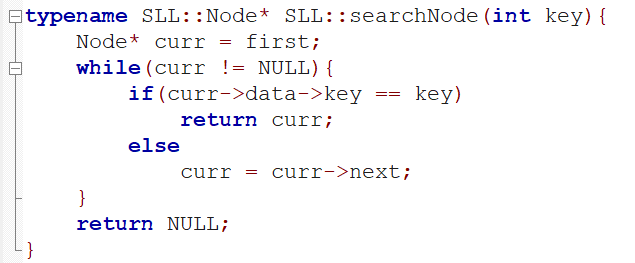


- Hàm removeFirst sẽ xoá node đầu của danh sách. Nếu con trỏ first và last cùng trỏ vào 1 node, tức là danh sách chỉ có 1 node thì ta xoá ngay node đó. Ngược lại, ta tạo node temp trỏ vào đầu danh sách, lùi con trỏ first trỏ về temp->next rồi xoá node mà temp đang trỏ vào.

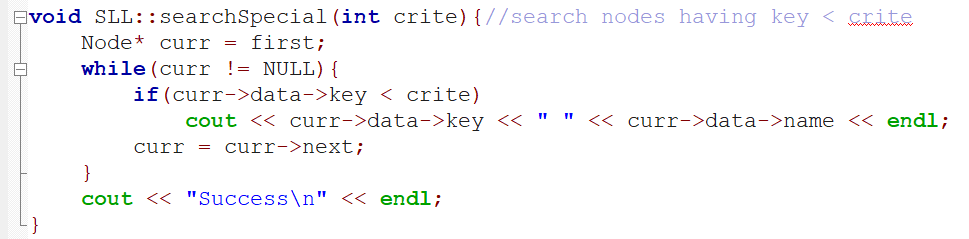
  
- Hàm removeLast xoá node cuối của danh sách. Nếu con trỏ first và last cùng trỏ vào 1 node, tức là danh sách chỉ có 1 node thì ta xoá ngay node đó. Ngược lại, ta cũng tạo con trỏ temp trỏ vào đầu danh sách. Ta tìm tới node áp chót của danh sách rồi xoá node cuối, gán con trỏ last vào node áp chót tìm được, đặt last->next = NULL.



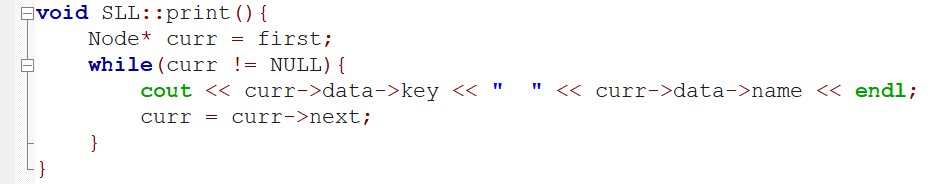
- Hàm removeMiddle xoá con trỏ sau vị trí pos được truyền vào. Ta tạo con trỏ curr trỏ vào đầu danh sách. Ta sẽ di chuyển dần con trỏ curr về cuối danh sách chừng nào pos > 1 và curr != NULL. Mỗi lần dịch chuyền ta giảm pos đi 1 đơn vị. Kết thúc vòng lặp, nếu pos = 1 tức là ta đã đến đúng node đứng trước node cần xoá. Ta trỏ con trỏ temp vào node cần xoá, liên kết node curr trỏ vào với node ngay sau node được temp trỏ rồi xoá node mà temp trỏ vào.



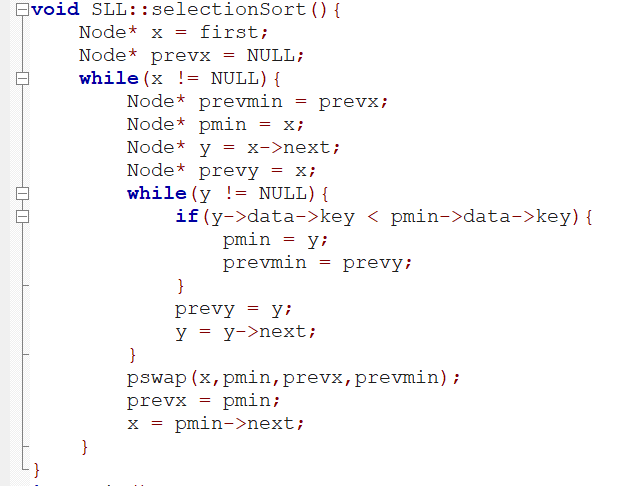
- Hàm searchNode trả về node có key bằng key được truyền vào. Đặt con trỏ curr vào đầu danh sách. Duyệt toàn bộ danh sách, nếu tìm được node có key trùng khớp thì trả về địa chỉ của node đó. Nếu không tìm được node nào thì trả về NULL.



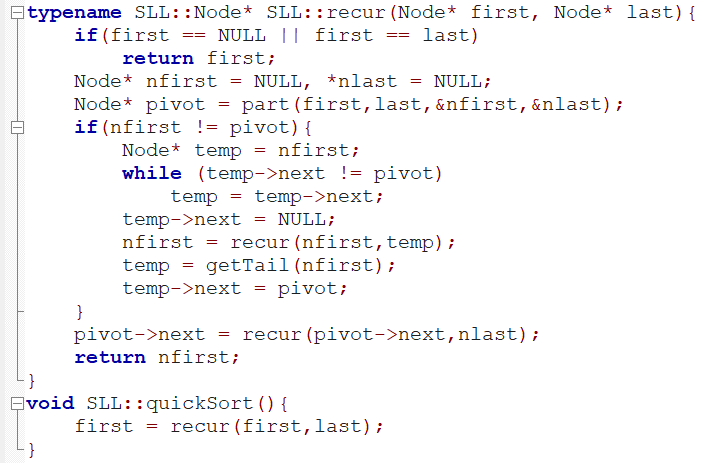
- Hàm searchSpecial in ra các node thoả mãn điều kiện nào đó, ở đây điều kiện là key < giá trị crite được truyền vào. Đặt con trỏ curr vào đầu danh sách. Duyệt toàn bộ danh sách, nếu tìm được bất kỳ node có key < crite thì in ra màn hình.



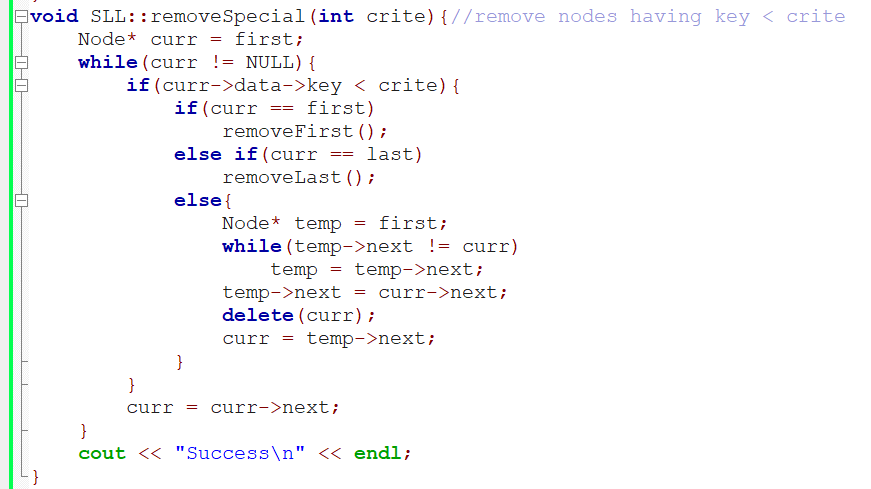
- Hàm print in toàn bộ danh sách ra màn hình. Đặt con trỏ curr vào đầu danh sách. Duyệt toàn bộ danh sách để in ra key và name của từng node.



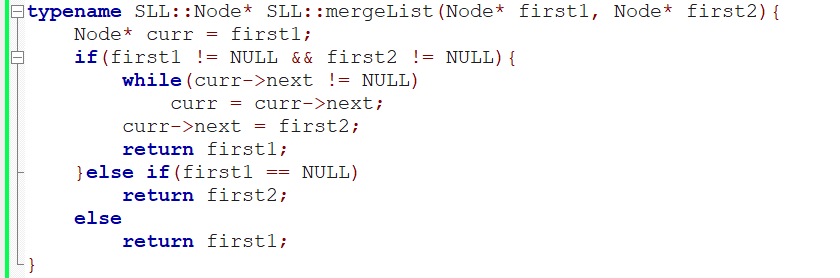
- Hàm selectionSort sắp xếp danh sách theo cách sắp xếp chọn. Đặt x trỏ vào đầu danh sách, prevx = NULL. Duyệt x từ đầu đến cuối danh sách. Tại mỗi bước duyệt, đặt y trỏ vào node ngay sau node được x trỏ, prevy trỏ cùng node với x, pmin = x, prevmin = prevx, rồi duyệt y đến cuối danh sách, so sánh các node đi qua với key của node được pmin trỏ vào, nếu key của node bé hơn key của pmin thì pmin sẽ trỏ vào node đó, prevmin trỏ vào node liền trước của node tìm được.Kết thúc bước duyệt của y, ta hoán đổi 2 node được x và pmin trỏ vào rồi tịnh tiến x sang node tiếp theo.



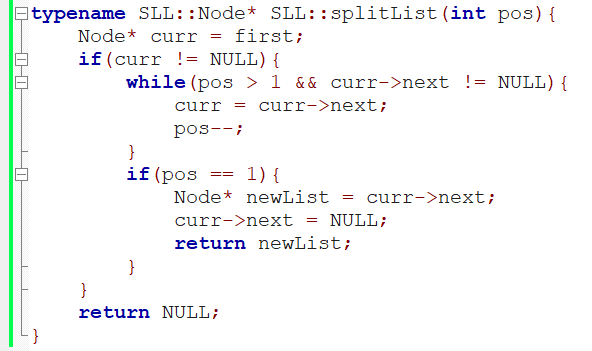
- Hàm quickSort được dùng để sắp xếp danh sách theo cách sắp xếp nhanh. Ta sử dụng đệ quy hàm recur để sắp xếp danh sách. Trong hàm recur, ta kiểm tra con trỏ first và last được truyền vào, nếu first = last hoặc first = NULL thì không cần phải sắp xếp vì đoạn danh sách đó không có hoặc chỉ có 1 phần tử. Ngược lại, ta khởi tạo con trỏ nfirst, nlast để lưu vị trí mới của đầu và cuối danh sách, con trỏ pivot được dùng để đánh dấu phân đoạn của danh sách. Từ đó, ta sắp xếp đệ quy nửa đầu của danh sách - từ đầu đến pivot. Sau đó, ghép nối 2 đoạn danh sách rồi lại sắp xếp đệ quy nửa sau của danh sách – từ pivot->next đến cuối danh sách. Kết quả trả về là con trỏ nfirst trỏ về node đầu tiên của danh sách sau sắp xếp.



- Hàm removeSpecial được sử dụng để xoá những node đặc biệt trong danh sách (ở đây là xoá các node có key bé hơn giá trị crite được truyền vào). Ta duyệt toàn bộ danh sách, nếu node thoả mãn nằm ở đầu hoặc cuối thì ta dùng 2 hàm removeFirst hoặc removeLast đã dựng ở trên. Nếu node tìm thấy ở giữa, ta tìm đến node ngay trước nó, liên kết với node liền sau node thoả mãn rồi xoá node tìm được.



- Hàm mergeList kết hợp 2 danh sách thành 1 danh sách duy nhất. Ta kiểm tra 2 node đầu của danh sách khác rỗng hay không. Nếu có, ta tìm đến node cuối của danh sách 1 rồi kết nối đến node đầu của danh sách 2, sau đó trả về node đầu của danh sách mới. Ngược lại, ta trả về node đầu của danh sách không rỗng.



- Hàm splitList được sử dụng để chia danh sách ra làm 2 phần tại node có vị trí pos. Ta duyệt danh sách nhằm tìm node liền trước node phân chia (node đầu của danh sách mới). Sau đó ta ngắt kết nối của 2 node này và trả về node đầu của danh sách mới.

2. Kỹ thuật được sử dụng: Nạp chồng constructor

