ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Cấu trúc dữ liệu và giải thuật - CO2003

Bài tập lớn 1

XÂY DỰNG CONCAT_STRING BẰNG DANH SÁCH

Tác giả: Vũ Văn Tiến

TP. HỒ CHÍ MINH, THÁNG 08/2022

ĐẶC TẢ BÀI TẬP LỚN

Phiên bản 1.0

1 Chuẩn đầu ra

Sau khi hoàn thành bài tập lớn này, sinh viên ôn lại và sử dụng thành thục:

- Lập trình hướng đối tượng.
- Các cấu trúc dữ liệu danh sách.
- Giải thuật sắp xếp.

2 Dẫn nhập

Chuỗi ký tự (string) thường được sử dụng để biểu diễn cho một đoạn văn bản, từ đó hiển thị các thông tin có ý nghĩa đến người dùng. Thông thường, chuỗi ký tự được hiện thực bằng cách sử dụng một danh sách đặc để lưu trữ các ký tự liền kề nhau. Tuy nhiên, với cách lưu trữ của mảng đặc, thao tác nối (operation concatenate/join) 2 chuỗi có độ dài lần lượt là m và n có độ phức tạp là $O(m + n)$.

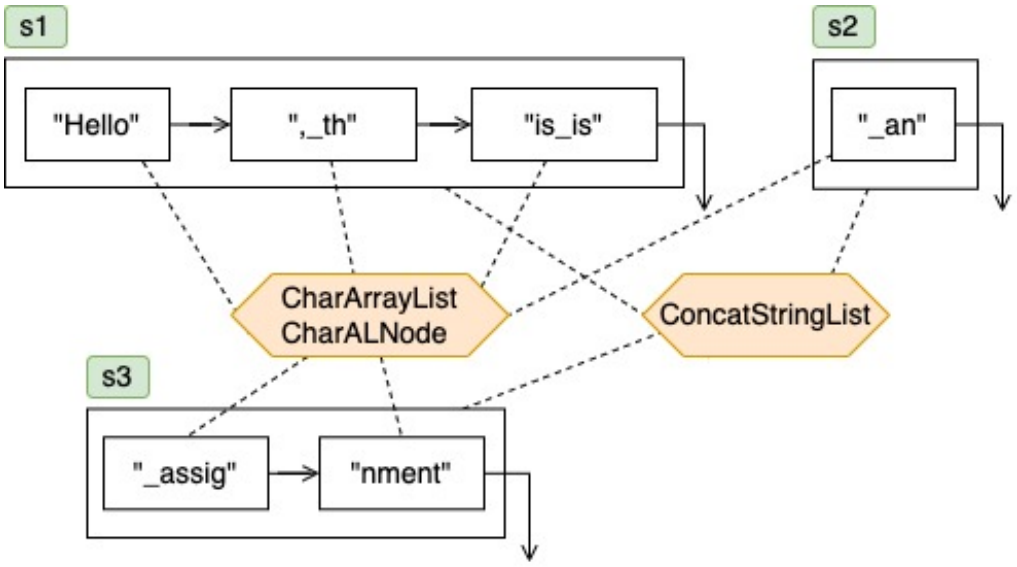
Mặt khác, danh sách liên kết là một cấu trúc dữ liệu có thể thực hiện thao tác nối 2 danh sách với độ phức tạp thấp hơn.

Trong bài tập lớn này, sinh viên được yêu cầu hiện thực một lớp chuỗi ký tự hỗ trợ thao tác nối chuỗi một cách hiệu quả sử dụng các cấu trúc dữ liệu danh sách (trong BTL này sau đây sẽ gọi chuỗi hỗ trợ thao tác nối là ConcatStringList).

3 Mô tả

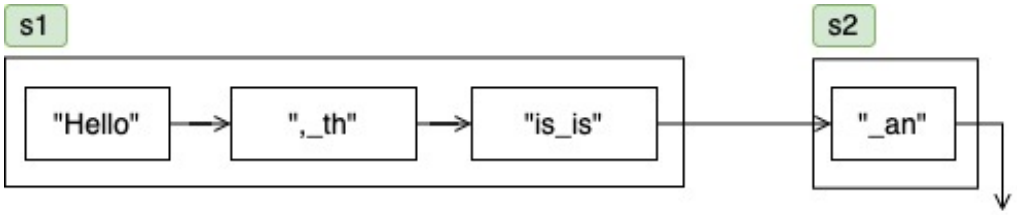
3.1 Tổng quan

Hình 1 biểu diễn cách hiện thực ConcatStringList. ConcatStringList s1 có 3 CharALNode, mỗi CharALNode có thông tin: chuỗi ký tự và liên kết đến CharALNode tiếp theo. Chuỗi ký tự được lưu trong CharArrayList, đây là 1 danh sách đặc (Array List) chứa các ký tự của chuỗi, danh sách này giúp thao tác truy cập ký tự tại vị trí ngẫu nhiên hiệu quả.



Hình 1: Tổng quan cách biểu diễn chuỗi ký tự

Mặt khác, thao tác nối chuỗi được thực hiện hiệu quả bằng cách nối liên kết của CharALNode cuối cùng của chuỗi trước với CharALNode đầu tiên của chuỗi sau. Ví dụ: trong Hình 1, khi thực hiện nối chuỗi s1 với s2, ta chỉ cần trỏ liên kết của CharALNode "is_is" đến CharALNode "_an". Kết quả của phép nối được biểu diễn trong Hình 2.



Hình 2: Kết quả của phép nối 2 chuỗi

Các phần sau sẽ mô tả chi tiết về các class cần được hiện thực.

3.2 class ConcatStringList (6 điểm)

Các phương thức cần hiện thực cho class ConcatStringList:

1. `ConcatStringList(const char * s)`
   - Khởi tạo đối tượng ConcatStringList với 1 CharALNode, trong CharALNode có CharArrayList được khởi tạo bởi chuỗi ***s***.
   - Độ phức tạp (mọi trường hợp): O(n) với n là độ dài của chuỗi ***s***.

2. `int length() const`

   - Trả về độ dài của chuỗi đang được lưu trữ trong đối tượng ConcatStringList.
   - Độ phức tạp (mọi trường hợp): O(1).

   Ví dụ 3.1

   Trong Hình 1:

   – s1.length() trả về 14.
   – s2.length() trả về 3.

3. `char get(int index) const`

   - Trả về ký tự tại vị trí ***index***.
   - Ngoại lệ: Nếu ***index*** có giá trị là 1 vị trí không hợp lệ trong chuỗi, ném ra ngoại lệ (thông qua lệnh **throw** trong ngôn ngữ C++): `out_of_range("Index of string is invalid!")`. ***index*** có giá trị là vị trí hợp lệ nếu `index` nằm trong đoạn $[0, l-1]$ với l là độ dài của chuỗi.
   - Độ phức tạp (trường hợp tệ nhất): O(k) với k là số lượng CharALNode của ConcatStringList.

   Ví dụ 3.2

   Trong Hình 1:

   – s1.get(14) ném ra ngoại lệ
   `out_of_range("Index of string is invalid!")`
   – s2.get(1) trả về ký tự 'a'.

4. `int indexOf(char c) const`

   - Trả về vị trí xuất hiện đầu tiên của ***c*** trong ConcatStringList. Nếu không tồn tại ký tự c thì trả về giá trị -1.
   - Độ phức tạp (trường hợp tệ nhất): $O(l)$ với l là chiều dài của chuỗi ConcatStringList.

   Ví dụ 3.3

   Trong Hình 1:

   – s1.indexOf('i') trả về 9.
   – s2.indexOf('b') trả về -1.

5. `string toString() const`

   - Trả về chuỗi biểu diễn cho đối tượng ConcatStringList.
   - Độ phức tạp (mọi trường hợp): $O(l)$ với l là chiều dài của chuỗi ConcatStringList.

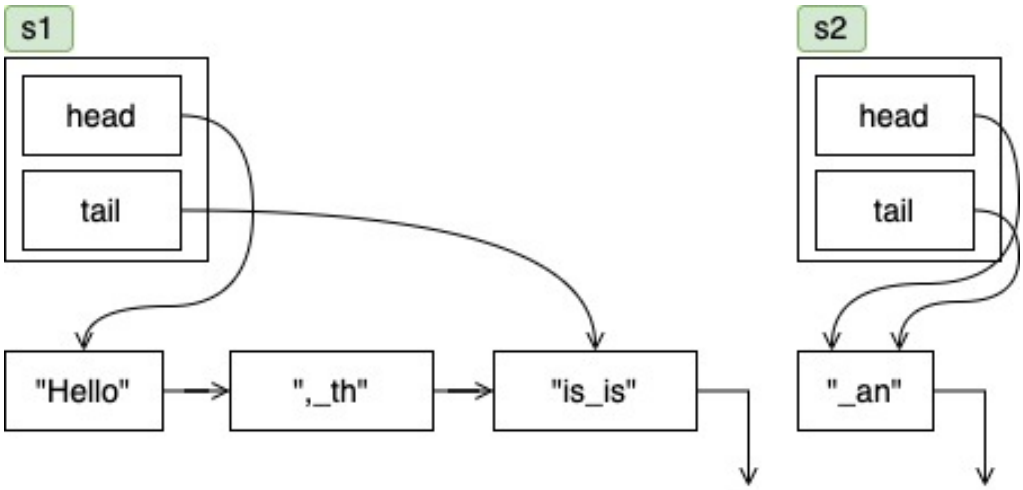
   Ví dụ 3.4

   Trong Hình 1:

   – s1.toString() trả về `"ConcatStringList["Hello,_this_is"]"`
   – s2.toString() trả về `"ConcatStringList["_an"]"`

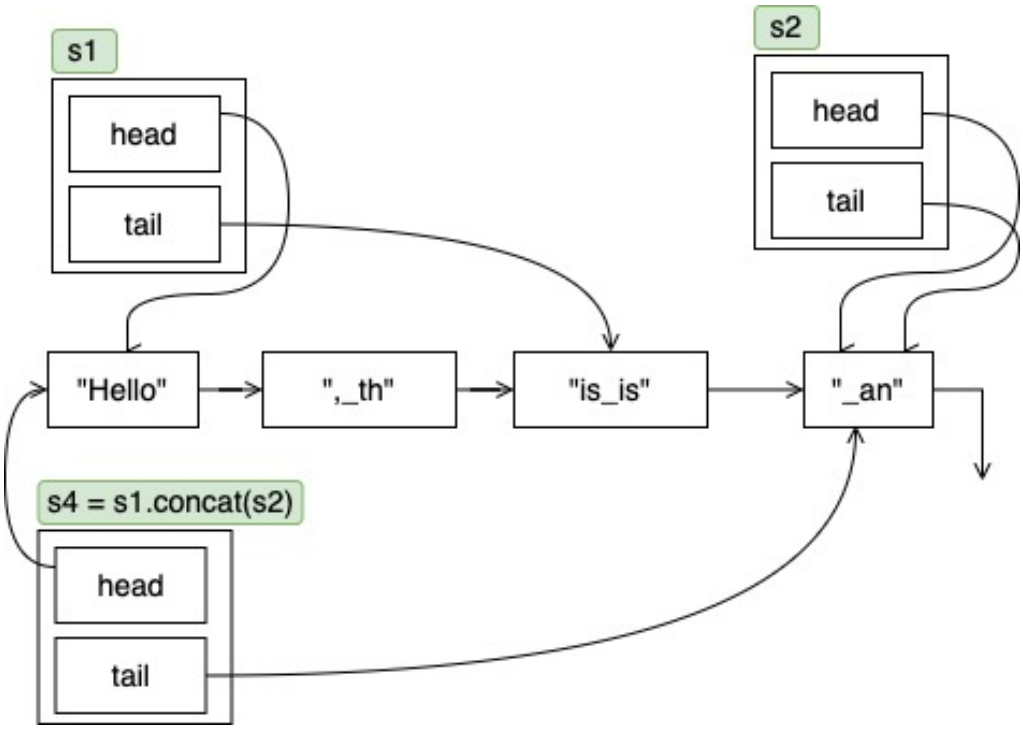
6. `ConcatStringList concat(const ConcatStringList & otherS) const`

   - Trả về đối tượng ConcatStringList mới và thực hiện trỏ liên kết của CharALNode cuối cùng của đối tượng hiện tại đến CharALNode đầu tiên trong đối tượng `otherS`.
   - Độ phức tạp (mọi trường hợp): $O(1)$
   - Ví dụ: Hình 3 và 4 lần lượt biểu diễn các chuỗi trước và sau khi thực hiện phép nối.
   - Lưu ý: class ConcatStringList cần đảm bảo có hai con trỏ đến CharALNode đầu và cuối. Hai con trỏ này sẽ được sử dụng trong một số yêu cầu sau của BTL này.
   - Lưu ý 2: Để tránh một số trường hợp mất node khi xóa theo mục 3.3, testcases đảm bảo **thao tác concat chỉ thực hiện trên các chuỗi không được tạo ra bởi concat**, và **mỗi chuỗi được tạo ra bởi hàm khởi tạo chỉ tham gia vào 1 thao tác concat**.



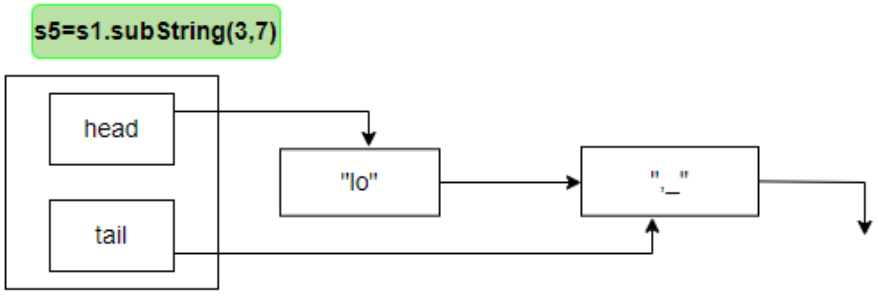
Hình 3: Minh hoạ chuỗi trước khi thực hiện phép nối

7. `ConcatStringList subString(int from, int to) const`



Hình 4: Minh hoạ chuỗi sau khi thực hiện phép nối

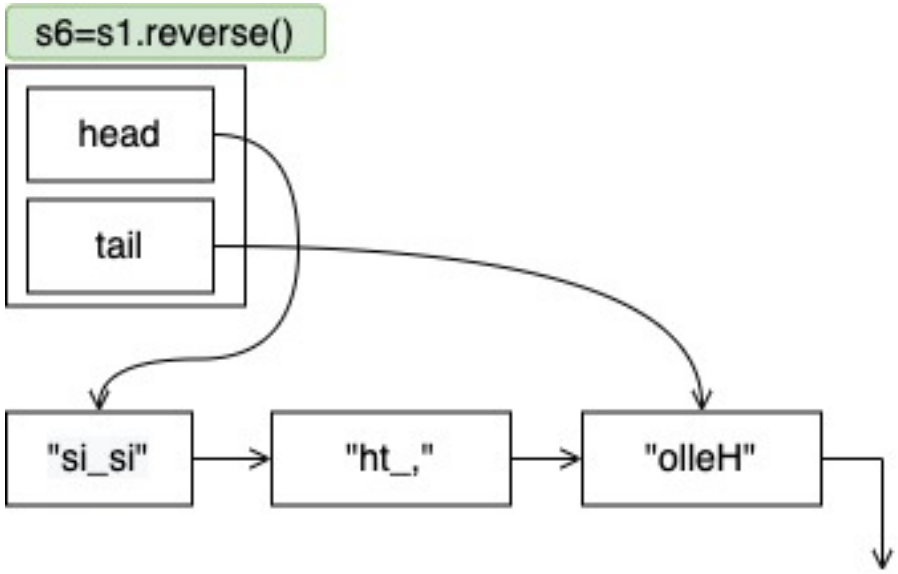
- Trả về đối tượng ConcatStringList mới chứa các ký tự bắt đầu từ vị trí `from` (bao gồm `from`) đến vị trí `to` (không bao gồm `to`).
- Ngoại lệ: Nếu **from** hoặc **to** là một vị trí không hợp lệ trong chuỗi thì ném ra ngoại lệ `out_of_range("Index of string is invalid")`. Nếu $from >= to$ thì ném ra ngoại lệ `logic_error("Invalid range")`.
- Ví dụ: Hình 5 minh hoạ cho thao tác subString.
- Lưu ý: Để có thể thực hiện các yêu cầu ở mục 3.3, cũng như đảm bảo khớp kết quả của testcase chấm điểm, chuỗi mới cần *tạo mới các* CharALNode (không sử dụng lại chuỗi gốc) và *có cấu trúc liên kết giống như chuỗi gốc* (hình 5 giữ lại liên kết giữa 2 node, thay vì gộp thành 1 node "lo,_").



Hình 5: Minh hoạ chuỗi sau khi thực hiện phép subString

8. `ConcatStringList reverse() const`

   - Trả về đối tượng ConcatStringList mới biểu diễn một chuỗi nghịch đảo của chuỗi gốc.
   - Ví dụ: Hình 6 minh hoạ thao tác reverse.



Hình 6: Minh hoạ thao tác reverse

9. `~ConcatStringList()`

   - Hiện thực hàm huỷ để tất cả vùng nhớ được cấp phát động phải được thu hồi sau khi chương trình kết thúc. Một hàm huỷ thông thường sẽ thu hồi các CharALNode nằm giữa head và tail. Điều này không phù hợp vì một số CharALNode vẫn có thể còn tham khảo đến do kết quả của thao tác nối chuỗi. Tham khảo thêm Mục 3.3 để hiện thực hàm huỷ cho phù hợp.

3.3 class ReferencesList và class DeleteStringList (4 điểm)

Xem xét lại Hình 4 minh hoạ kết quả của thao tác nối. Giả sử ta muốn thực hiện xoá chuỗi s2, nếu CharALNode "_an" bị xoá thì chuỗi s4 chỉ còn 3 CharALNode và không còn biểu diễn đúng kết quả của thao tác nối chuỗi. Để giải quyết vấn đề này, ta sẽ duy trì một danh sách các số lượng tham khảo đến các CharALNode đầu và cuối, được biểu diễn bởi class ReferencesList. Ví dụ: trong Hình 4, CharALNode của "is_is" có số lượng tham khảo đến nó là 1 (từ tail của s1), CharALNode của "_an" có số lượng tham khảo là 3. Đồng thời, ta duy trì một danh sách các chuỗi đã được xoá, được biểu diễn bởi class DeleteStringList. Mỗi node của DeleteStringList

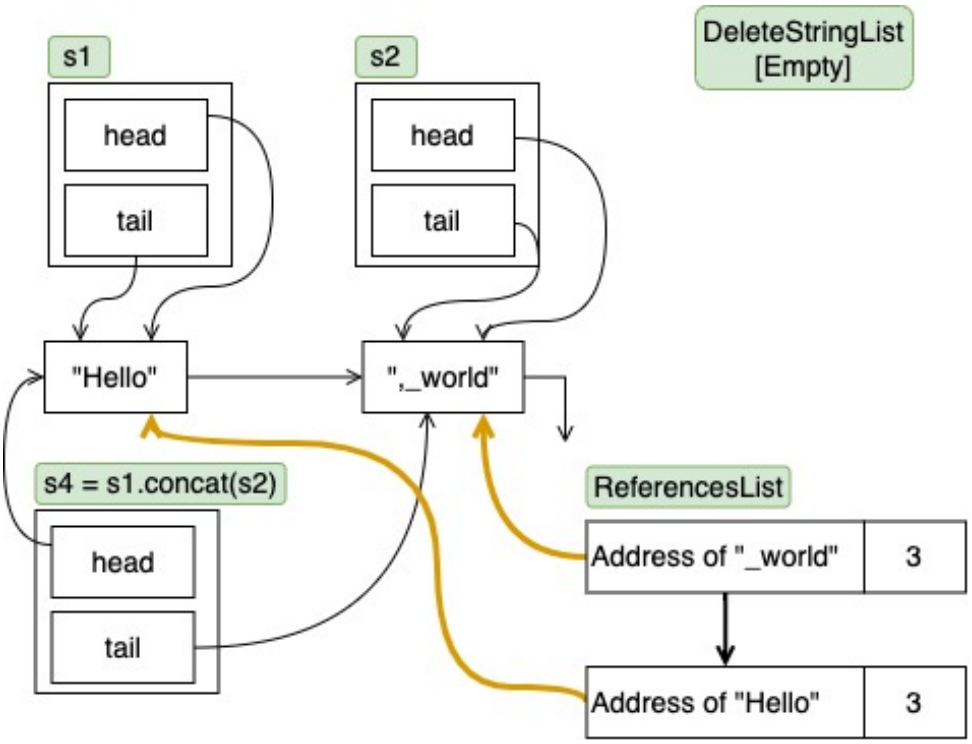
giữ thông tin của CharALNode đầu và cuối của một chuỗi đã được xoá. Ta sẽ duyệt toàn bộ danh sách này, kiểm tra nếu cả head và tail có tổng số lượng tham khảo bằng 0 thì thực hiện xoá các CharALNode nằm giữa head và tail (bao gồm cả head và tail). Sau đây là một số yêu cầu khi hiện thực class ReferencesList và class DeleteStringList:

- Mỗi khi có một chuỗi **mới** được tạo ra, CharALNode đầu và cuối của chuỗi này sẽ được thêm vào ReferencesList để theo dõi. Trong 9 phương thức được yêu cầu hiện thực cho class ConcatStringList, các phương thức sau sẽ tạo ra một đối tượng mới:
  - **ConcatStringList(const char *)**
  - **ConcatStringList concat(const ConcatStringList & otherS) const**
  - **ConcatStringList subString(int from, int to) const**
  - **ConcatStringList reverse() const**
- Mỗi khi xoá một chuỗi, ta sẽ giảm số lượng tham khảo trong ReferencesList tương ứng với CharALNode head và tail của chuỗi xuống 1 đơn vị. Đồng thời, thêm 1 node mới gồm thông tin của head và tail này vào cuối DeleteStringList. Sau đó, ta duyệt qua các node trong DeleteStringList, nếu node nào có tổng số lượng tham khảo đến head và tail bằng 0 thì ta xoá các CharALNode nằm giữa head và tail, sau đó xoá node ra khỏi DeleteStringList. Nếu mọi phần tử của ReferencesList đều có tổng bằng 0, ta xóa tất cả các node trong ReferencesList. Lưu ý, khi xoá các CharALNode nằm giữa head và tail, cần kiểm tra head và tail đã bị xoá trước đó hay chưa.
- Khi thực hiện xoá một chuỗi, ta cần quan tâm đến các CharALNode mà có số tham khảo thấp. Vì sau khi giảm số tham khảo thì có thể xảy ra trường hợp số lượng tham khảo của chúng bằng 0 (kéo theo việc có thể ta phải xóa các CharALNode theo mô tả trên). Để cải thiện hiệu quả tìm kiếm trong ReferencesList, ta sẽ luôn duy trì **thứ tự không giảm** trên danh sách này. Với thứ tự đó, các node có số lượng tham khảo bằng 0 sẽ luôn ở đầu danh sách. Tuy nhiên, các node này không còn cần thiết cho việc tìm kiếm nữa. Do vậy, các node có số lượng tham khảo bằng 0 **cần nằm ở cuối** ReferencesList.

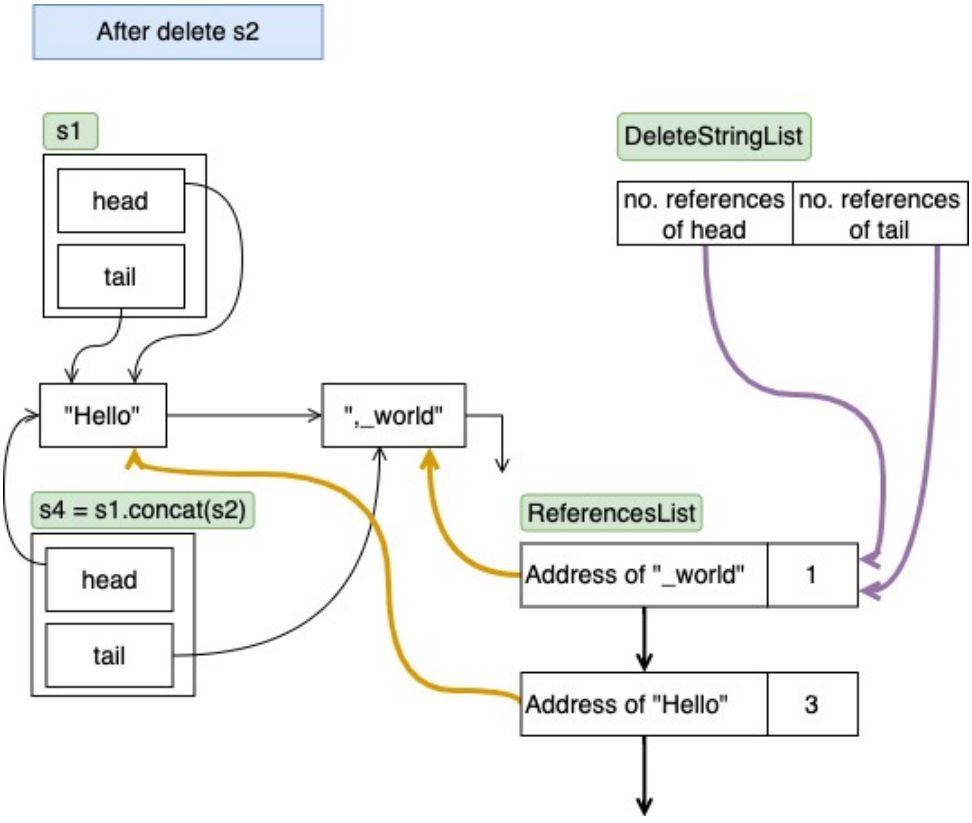
Xem thêm các Hình 7, 8, 9 minh hoạ về ReferenceList và DeleteStringList qua các bước xoá chuỗi. Lưu ý: Hình 9, DeleteStringList chưa minh hoạ kết quả cuối cùng. Vì số lượng tham khảo đến "_world" bằng 0 nên node đầu tiên của DeleteStringList (gồm "no. references of s1 head" và "no. references of s1 tail") sẽ bị xoá. DeleteStringList sau đó chỉ có 1 node duy nhất.

Các phương thức cần hiện thực cho **class ReferenceList**:

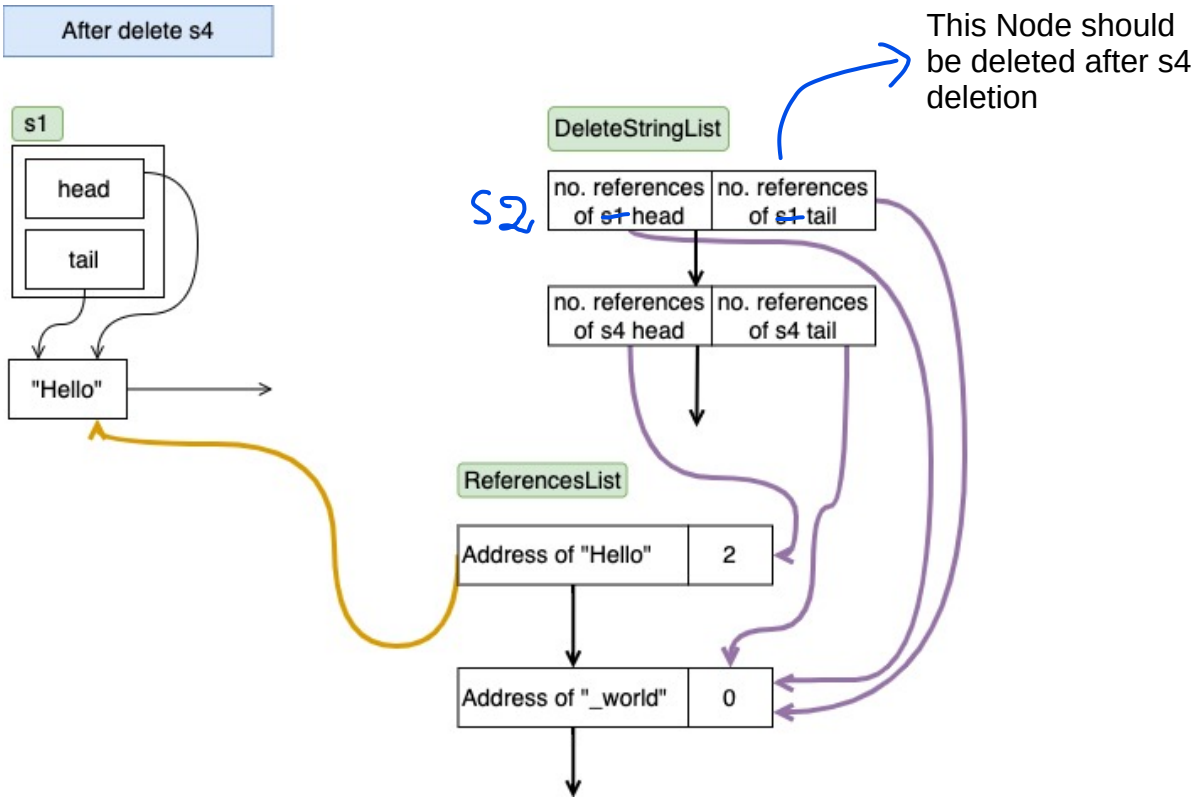
1. **int size() const**
   - Trả về số node trong danh sách tham khảo.



Hình 7: Minh hoạ ReferenceList



Hình 8: Minh hoạ ReferenceList sau khi xoá s2



Hình 9: Minh hoạ ReferenceList sau khi xoá s4

- Độ phức tạp: O(1).

Ví dụ 3.5

Gọi size() với danh sách tham khảo ở Hình 7 trả về 2.

2. **int refCountAt(int index) const**

- Trả về số lượng ứng với tham khảo ở vị trí **index**.
- Ngoại lệ: Nếu **index** có giá trị là 1 vị trí không hợp lệ, ném ra ngoại lệ: `out_of_range("Index of references list is invalid!")`.

Ví dụ 3.6

Gọi refCountAt(1) với danh sách tham khảo ở Hình 7 trả về 3.
Gọi refCountAt(0) với danh sách tham khảo ở Hình 9 trả về 1.

3. **string refCountsString() const**

- Trả về chuỗi biểu diễn của các số lượng tham khảo trong danh sách tham khảo.
- Độ phức tạp: $O(n)$.

Ví dụ 3.7

Gọi refCountsString() với danh sách tham khảo ở Hình 8 trả về **"RefCounts[1,3]"**.
Gọi refCountsString() với danh sách tham khảo ở Hình 9 trả về **"RefCounts[2]"**.

Các phương thức cần hiện thực cho **class DeleteStringList**:

1. **int size() const**
   - Trả về số node trong danh sách.
   - Độ phức tạp: O(1).

   Ví dụ 3.8

   Gọi size() với DeleteStringList ở Hình 7 trả về 0.
   Gọi size() với DeleteStringList ở Hình 9 trả về 1.

2. **string totalRefCountsString() const**
   - Trả về chuỗi biểu diễn các tổng số lượng tham khảo của các node.
   - Độ phức tạp: O(n).

   Ví dụ 3.9

   Gọi totalRefCountsString() với danh sách ở Hình 8 trả về:
   "TotalRefCounts[1]"
   Gọi totalRefCountsString() với danh sách ở Hình 9 trả về:
   "TotalRefCounts[2]"

3.4 Yêu cầu

Để hoàn thành bài tập lớn này, sinh viên phải:

1. Đọc toàn bộ tập tin mô tả này.
2. Tải xuống tập tin initial.zip và giải nén nó. Sau khi giải nén, sinh viên sẽ nhận được các tập tin: main.cpp, main.h, ConcatStringList.h, ConcatStringList.cpp và thư mục sample_output. Sinh viên sẽ chỉ nộp 2 tập tin là ConcatStringList.h và ConcatStringList.cpp nên không được sửa đổi tập tin main.h khi chạy thử chương trình.

3. Sinh viên sử dụng câu lệnh sau để biên dịch:
   g++ -o main main.cpp ConcatStringList.cpp -I . -std=c++11
   Câu lệnh trên được dùng trong command prompt/terminal để biên dịch chương trình. Nếu sinh viên dùng IDE để chạy chương trình, sinh viên cần chú ý: thêm đầy đủ các tập tin vào project/workspace của IDE; thay đổi lệnh biên dịch của IDE cho phù hợp. IDE thường cung cấp các nút (button) cho việc biên dịch (Build) và chạy chương trình (Run). Khi nhấn Build IDE sẽ chạy một câu lệnh biên dịch tương ứng, thông thường, chỉ biên dịch phải main.cpp. Sinh viên cần tìm cách cấu hình trên IDE để thay đổi lệnh biên dịch: thêm file ConcatStringList.cpp, thêm option -std=c++11, -I .
4. Chương trình sẽ được chấm trên nền tảng Unix. Nền tảng và trình biên dịch của sinh viên có thể khác với nơi chấm thực tế. Nơi nộp bài trên BKeL được cài đặt giống với nơi chấm thực tế. Sinh viên phải chạy thử chương trình trên nơi nộp bài và phải sửa tất cả các lỗi xảy ra ở nơi nộp bài BKeL để có đúng kết quả khi chấm thực tế.
5. Sửa đổi các file ConcatStringList.h, ConcatStringList.cpp để hoàn thành bài tập lớn này và đảm bảo hai yêu cầu sau:

   - Tất cả các phương thức trong mô tả này đều phải được hiện thực để việc biên dịch được thực hiện thành công. Nếu sinh viên chưa thể hiện thực được phương thức nào, hãy cung cấp một hiện thực rỗng cho phương thức đó. Mỗi testcase sẽ gọi một số phương thức đã mô tả để kiểm tra kết quả trả về.
   - Chỉ có 1 lệnh **include** trong tập tin ConcatStringList.h là **#include "main.h"** và một include trong tập tin ConcatStringList.cpp là **#include "ConcatStringList.h"**. Ngoài ra, không cho phép có một **#include** nào khác trong các tập tin này.

6. Trong tập tin main.cpp có cung cấp một số testcases đơn giản trong các hàm có định dạng **"tc<x>()"**. Kết quả chạy của hàm **"tc<x>()"** được ghi lại trong tập tin **"tc<x>.txt"** trong thư mục **sample_output**.
7. Sinh viên được phép viết thêm các phương thức và thuộc tính khác trong các class được yêu cầu hiện thực.
8. Sinh viên được yêu cầu thiết kế và sử dụng các cấu trúc dữ liệu dựa trên các loại danh sách đã học.
9. Sinh viên phải giải phóng toàn bộ vùng nhớ đã xin cấp phát động khi chương trình kết thúc.

4 Nộp bài

Sinh viên chỉ nộp 2 tập tin: ConcatStringList.h và ConcatStringList.cpp, trước thời hạn được đưa ra trong đường dẫn "Assignment 1 - Submission". Có một số testcase đơn giản được sử dụng để kiểm tra bài làm của sinh viên nhằm đảm bảo rằng kết quả của sinh viên có thể biên dịch và chạy được. Sinh viên có thể nộp bài bao nhiêu lần tùy ý nhưng chỉ có bài nộp cuối cùng được tính điểm. Vì hệ thống không thể chịu tải khi quá nhiều sinh viên nộp bài cùng một lúc, vì vậy sinh viên nên nộp bài càng sớm càng tốt. Sinh viên sẽ tự chịu rủi ro nếu nộp bài sát hạn chót. Khi quá thời hạn nộp bài, hệ thống sẽ đóng nên sinh viên sẽ không thể nộp nữa. Bài nộp qua các phương thức khác đều không được chấp nhận.

5 Một số quy định khác

- Sinh viên phải tự mình hoàn thành bài tập lớn này và phải ngăn không cho người khác đánh cắp kết quả của mình. Nếu không, sinh viên sẽ bị xử lý theo quy định của trường vì gian lận.
- Mọi quyết định của giảng viên phụ trách bài tập lớn là quyết định cuối cùng.
- Sinh viên không được cung cấp testcase sau khi chấm bài mà chỉ được cung cấp thông tin về chiến lược thiết kế testcase và phân bố số lượng sinh viên đúng theo từng testcase.
- Nội dung Bài tập lớn sẽ được Harmony với một câu hỏi trong bài kiểm tra với nội dung tương tự.

6 Thay đổi so với phiên bản trước

- Bổ sung phần "Lưu ý" cho phương thức **ConcatStringList subString(int from, int to) const**
- Chỉnh sửa đáp án ví dụ 3.9 trong mô tả phương thức **string totalRefCountsString() const**
- Chỉnh sửa điều kiện dẫn đến việc xóa tất cả các node trong ReferencesList ở mục 3.3.
- Bổ sung phần "Lưu ý 2" cho phương thức **ConcatStringList concat(const ConcatStringList & otherS) const**.