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

Bài tập lớn 1 - Phần 1

DANH SÁCH LIÊN KẾT PHÂN MẢNH

TP. HỒ CHÍ MINH, THÁNG 09/2020

ĐẶC TẢ BÀI TẬP LỚN 1

Phiên bản 2.0

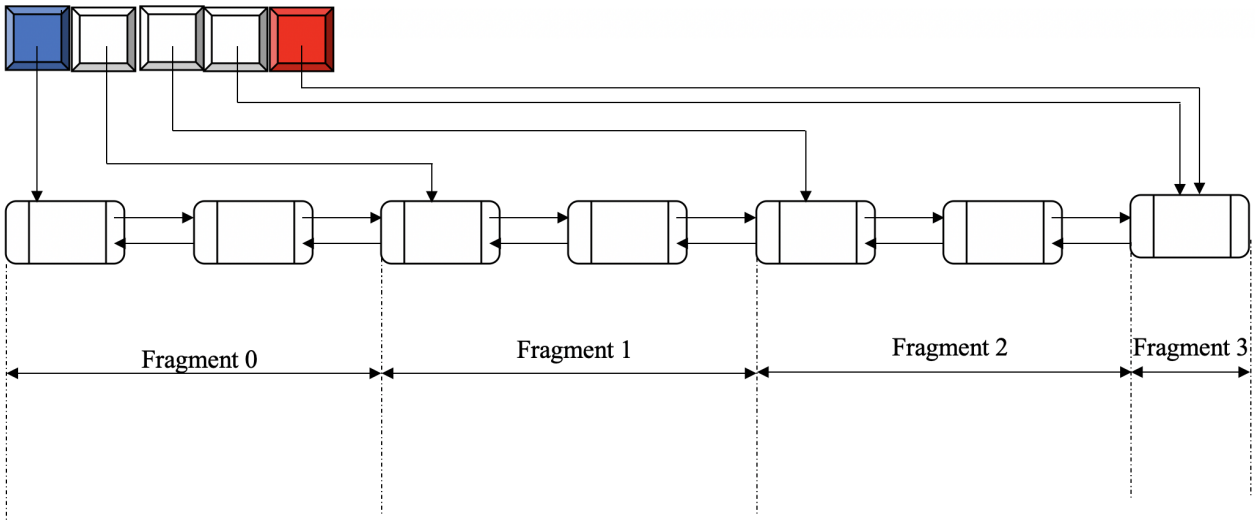
1 Chuẩn đầu ra

Sau khi hoàn thành bài tập lớn này, sinh viên sẽ có khả năng:

- Hiện thực được một phái sinh của danh sách liên kết đôi, cụ thể là danh sách liên kết phân mảnh.
- Biết cách sử dụng một cấu trúc dữ liệu danh sách.

2 Nhiệm vụ

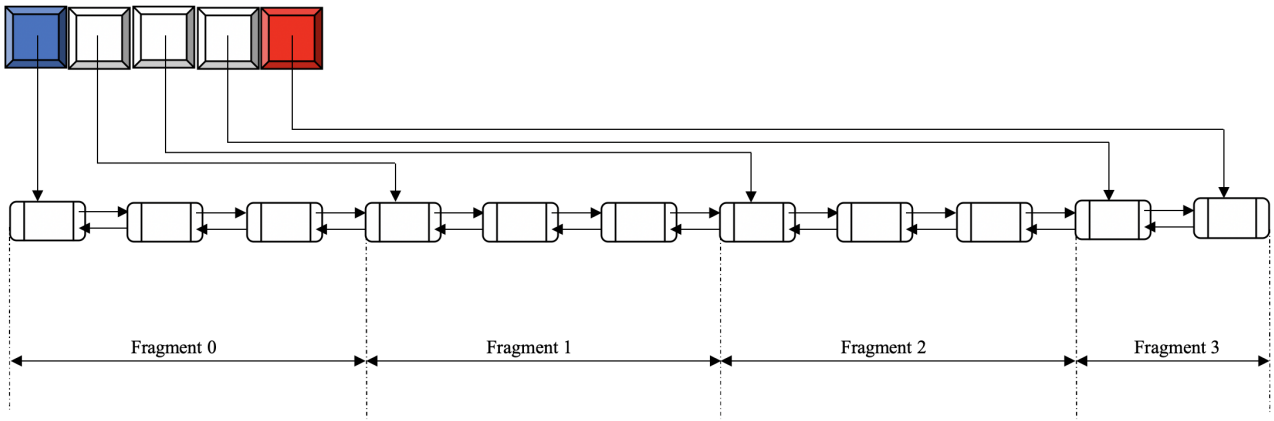
Sinh viên được yêu cầu xây dựng một chương trình trên C++ để hiện thực ý tưởng danh sách liên kết đôi phân mảnh (Fragment Linked List - FLL). Một FLL được minh họa bằng hình 1 và hình 2.



Hình 1: FLL với 7 phần tử và mỗi phân mảnh có kích thước tối đa 2 phần tử

Một danh sách liên kết phân mảnh được hiện thực dựa trên một danh sách liên kết đôi, trong đó danh sách được phân thành các phân mảnh ($fragment$) có kích thước tối đa là $fragment\_max\_size$.

Để lưu trữ các phân mảnh này, người ta dùng một danh sách các con trỏ lưu địa chỉ các nút nằm ở đầu phân mảnh, trong đó:



Hình 2: FLL với 11 phần tử và mỗi phân mảnh có kích thước tối đa 3 phần tử

- Con trỏ đầu tiên (màu xanh) trong danh sách này trỏ đến nút đầu tiên trong danh sách các phần tử cũng là nút đầu tiên trong phân mảnh đầu tiên.
- Con trỏ cuối cùng (màu đỏ) trong danh sách này trỏ đến nút cuối cùng trong danh sách các phần tử.
- Các con trỏ khác (màu trắng) trong danh sách này trỏ đến các nút đầu của mỗi phân mảnh theo thứ tự các phân mảnh.

Sinh viên được yêu cầu hiện thực cách thức lưu trữ và các phương thức cần có của cấu trúc danh sách liên kết đôi phân mảnh trên.

3 Hướng dẫn chung

Sinh được được cung cấp hai file 201.dsa-a1p1-des.pdf để mô tả bài tập lớn và file FragmentLinkedList.cpp chứa code khởi tạo, trong đó:

- Lớp `IList` (với template `T`) thể hiện interface (như một bản thiết kế) của cấu trúc danh sách nói chung bao gồm các phương thức:
  - `void add(const T& e)`: thêm một phần tử mới vào cuối danh sách.
  - `void add(int index, const T& e)`: thêm một phần tử mới vào vị trí `index`.
  - `T removeAt(int index)`: xóa phần tử tại vị trí `index` và trả về phần tử vừa xóa.
  - `bool removeItem(const T& item)`: xóa phần tử có giá trị `item` và trả về kết quả kiểm tra có tìm thấy và xóa hay không?
  - `bool empty()`: trả về kết quả kiểm tra danh sách hiện tại có rỗng hay không?
  - `int size()`: trả về kích thước (hay số phần tử) của danh sách hiện tại.

  - `void clear()`: xóa toàn bộ các phần tử hiện tại có trong danh sách.
  - `T get(int index)`: tìm và trả về phần tử tại vị trí `index`.
  - `void set(int index, const T& element)`: thiết lập giá trị mới cho phần tử tại `index`.
  - `int indexOf(const T& item)`: tìm và trả về vị trí của phần tử mang giá trị `item` trong danh sách.
  - `bool contains(const T& item)`: trả về kết quả kiểm tra trong danh sách có chứa phần tử mang giá trị `item` hay không?
  - `string toString()`: trả về một thể hiện của danh sách dưới dạng một chuỗi.

- Lớp `FragmentLinkedList` (với template `T`) là lớp cấu trúc danh sách phân mảnh cần hiện thực (sinh viên không được phép thay đổi tên lớp này và các phương thức đã nêu ở trên), trong đó:
  - Hai lớp lồng bên trong (nested class) `Node` và `Iterator` thể hiện nút trong danh sách và đối tượng sử dụng cho các hành động lặp đi lặp lại trên danh sách. Sinh viên có thể sử dụng các định nghĩa đã viết sẵn hoặc thay đổi nếu cần thiết.
  - Các phương thức được override dựa trên interface `IList`.
  - Hai phương thức phục vụ cho lớp `Iterator`:
    * `Iterator begin(int fragmentIndex = 0)`: trả về `Iterator` đầu tiên tương ứng với với `fragmentIndex`.
      Ví dụ: với `fragmentIndex = 1` trả về Iterator tương ứng với phần tử đầu tiên trong fragment 1.
    * `Iterator end(int fragmentIndex = -1)`: trả về `Iterator` phần tử kế tiếp của phần tử cuối cùng tương ứng với với `fragmentIndex`. Với `fragmentIndex = -1` trả về Iterator tương ứng với phần tử kế tiếp của phần tử cuối cùng trong danh sách (tức NULL).
      Ví dụ: với `fragmentIndex = 1` trả về Iterator tương ứng với phần tử kế tiếp của phần tử cuối cùng trong fragment 1, tức là phần tử đầu của fragment 2.
  - Các phương thức của lớp `Iterator`:
    * `Iterator(FragmentLinkedList<T>*, bool)`: thiết lập `pNode` trỏ đến node đầu tiên (`index = 0`) trong danh sách được trỏ bởi `pList` khi `begin = true`, ngược lại trỏ đến `NULL` (`index = pList->size()`).
    * `Iterator(int, FragmentLinkedList<T>*, bool)`: `pNode` trỏ đến node đầu tiên của fragment trong danh sách được trỏ bởi `pList` khi `begin = true`, ngược lại trỏ đến node kế của node cuối cùng của fragment.
    * `Iterator &operator=(const Iterator &iterator)`: toán tử gán cho lớp It-

erator thực hiện việc gán các thuộc tính tương ứng với iterator nhận vào, đồng thời trả về giá trị `Iterator` hiện tại đang giữ.

* `T &operator*()`: trả về giá trị `data` tương ứng với node đang trỏ tới. Trong trường hợp đang trỏ tới NULL, trả ra một exception `std::out_of_range("Segmentation fault!")`
* `bool operator!=(const Iterator &iterator)`: toán tử so sánh khác của lớp `Iterator`, trả về `true` khi hoặc khác nhau vùng nhớ đang trỏ đến của `pNode` hoặc khác nhau giá trị `index`.
* `void remove()`: xóa node tương ứng mà iterator đang trỏ tới. Sau khi xóa, node trỏ tới vị trí trước vị trí xóa 1 node. Trong trường hợp xóa vị trí bắt đầu, `pNode` trỏ tới NULL tương ứng với `index = -1`.
* `void set(const T& element)`: thiết lập giá trị mới cho node mà `pNode` đang trỏ tới.
* `Iterator &operator++()`: là prefix operator++ thiết lập node tiếp theo cho `pNode` và tăng giá trị index tương ứng.
* `Iterator operator++(int)`: là postfix operator++ thiết lập node tiếp theo cho `pNode` và tăng giá trị index tương ứng.

4 Quy định về thắc mắc và nộp bài

Sinh viên viết các phần hiện thực trong phần `STUDENT_ANSWER` và nộp phần này lên trên site môn học BK E-learning. Hình thức nộp bài và thời hạn nộp được thông báo cụ thể trên site môn học.

Sinh viên được giải đáp thắc mắc trên diễn đàn trên site môn học, TUYỆT ĐỐI KHÔNG GỬI EMAIL cho giảng viên phụ trách để đặt câu hỏi.

——————————HẾT——————————