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

HIỆN THỰC TEXT BUFFER ĐƠN GIẢN SỬ DỤNG DANH SÁCH

TP. HỒ CHÍ MINH, THÁNG 06/2025

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 (OOP)
- Cấu trúc dữ liệu danh sách
- Các thuật toán sắp xếp

2 Dẫn nhập

Trong Bài tập lớn 1, sinh viên được yêu cầu hiện thực Text Buffer sử dụng cấu trúc dữ liệu Danh sách liên kết đôi (Doubly Linked List - DLL), nhằm mô phỏng hoạt động của một trình soạn thảo văn bản đơn giản.

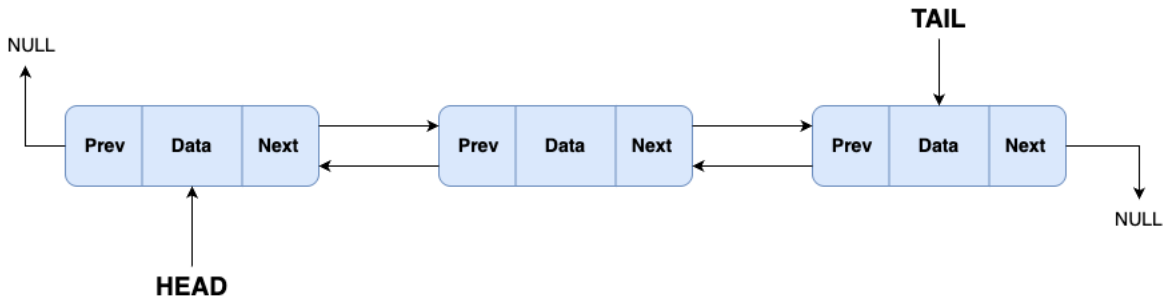
Việc sử dụng danh sách liên kết đôi mang lại nhiều ưu điểm nổi bật so với cách quản lý chuỗi bằng mảng tĩnh truyền thống, đặc biệt ở khả năng tối ưu thao tác chèn hoặc xóa ký tự tại các vị trí tùy ý với độ phức tạp thấp hơn. Điều này giúp cải thiện hiệu suất khi thao tác trên các văn bản có kích thước lớn hoặc có nhiều chỉnh sửa liên tục ở giữa văn bản.

Thông qua bài tập này, sinh viên không chỉ rèn luyện kỹ năng hiện thực cấu trúc dữ liệu cơ bản, mà còn phát triển tư duy lập trình hướng đối tượng (OOP), hiểu rõ các thuật toán tìm kiếm, sắp xếp và quản lý lịch sử thao tác. Đây là những kỹ năng quan trọng, đóng vai trò nền tảng trong việc xây dựng và phát triển các ứng dụng xử lý văn bản trong thực tế.

3 Mô tả

3.1 Danh sách liên kết đôi

Danh sách liên kết đôi (*Doubly Linked List*, viết tắt DLL) là cấu trúc dữ liệu trong đó mỗi phần tử lưu trữ một phần tử kiểu tổng quát `T`, kèm theo hai con trỏ: `next` (trỏ đến phần tử kế tiếp) và `prev` (trỏ đến phần tử trước đó). DLL hỗ trợ chèn, xóa ở mọi vị trí với chi phí thấp, phù hợp cho bài toán quản lý buffer văn bản.



Hình 1: Minh hoạ Danh sách liên kết đôi

Sinh viên tự đề xuất các thuộc tính thành viên của Lớp `DoublyLinkedList`. Các phương thức cần phải được hiện thực:

- `DoublyLinkedList()`
  - **Chức năng:** Khởi tạo một danh sách rỗng.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(1)$.
- `~DoublyLinkedList()`
  - **Chức năng:** Giải phóng bộ nhớ của toàn bộ các phần tử, tránh rò rỉ bộ nhớ.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.
- `void insertAtHead(T data)`
  - **Chức năng:** Chèn một phần tử chứa `data` vào đầu danh sách.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(1)$.
- `void insertAtTail(T data)`
  - **Chức năng:** Chèn một phần tử chứa `data` vào cuối danh sách.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(1)$.
- `void insertAt(int index, T data)`
  - **Chức năng:** Chèn một phần tử chứa `data` tại vị trí `index`.
  - **Ngoại lệ:** Ném `out_of_range("Index is invalid!")` nếu `index` không hợp lệ (nhỏ hơn 0 hoặc lớn hơn hoặc bằng kích thước danh sách).
  - **Độ phức tạp:** $O(n)$
- `void deleteAt(int index)`

  - **Chức năng:** Xóa phần tử tại vị trí `index`.
  - **Ngoại lệ:** Ném `out_of_range("Index is invalid!")` nếu `index` không hợp lệ.
  - **Độ phức tạp:** $O(n)$

- `T& get(int index)`
  - **Chức năng:** Trả về tham chiếu đến phần tử tại vị trí chỉ định `index` trong danh sách
  - **Ngoại lệ:** Ném `out_of_range("Index is invalid!")` nếu `index` không hợp lệ.
  - **Độ phức tạp:** $O(n)$

- `int indexOf(T item)`
  - **Chức năng:** Trả về chỉ số của node đầu tiên có giá trị bằng với `item` trong danh sách. Nếu không tìm thấy, trả về `-1`.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.

- `bool contains(T item)`
  - **Chức năng:** Kiểm tra xem danh sách có chứa node có giá trị bằng với `item` hay không. Trả về `true` nếu có, ngược lại trả về `false`.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.

- `int size()`
  - **Chức năng:** Trả về số lượng phần tử hiện có trong danh sách.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(1)$.

- `void reverse()`
  - **Chức năng:** Đảo ngược toàn bộ danh sách.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.

- `string toString(string (*convert2str)(T&) = 0) const;`
  - **Chức năng:** Trả về chuỗi biểu diễn dữ liệu của danh sách. Con trỏ hàm convert2str được dùng để chuyển đổi mỗi phần tử sang dạng chuỗi. Nếu con trỏ hàm không được cung cấp, sinh viên sử dụng dạng biểu diễn chuỗi mặc định của phần tử.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.

– **Định dạng in ra:** `[<element 1>, <element 2>, <element 3>, ...]`.

Ví dụ 3.1

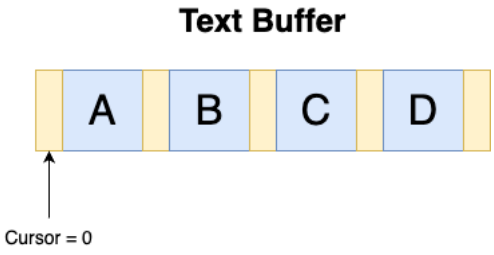
Nếu danh sách bao gồm: 1, 2, 3, 4, 5, con trỏ hàm được cung cấp để định dạng số nguyên thành chuỗi.
Kết quả trả về của toString: `[1, 2, 3, 4, 5]`

3.2 TextBuffer và HistoryManager

TextBuffer là cấu trúc dữ liệu mô phỏng trình soạn thảo văn bản cơ bản. Cấu trúc này cho phép chèn, xóa ký tự, di chuyển con trỏ, tìm kiếm và sắp xếp nội dung văn bản. TextBuffer lưu trữ dữ liệu ký tự (`char`) và vị trí con trỏ hiện tại (`cursor`). Con trỏ cho phép người dùng thực hiện các thao tác chỉnh sửa ở bất kỳ vị trí nào trong văn bản, tương tự các trình soạn thảo thực tế.

Mô tả tổng quan:

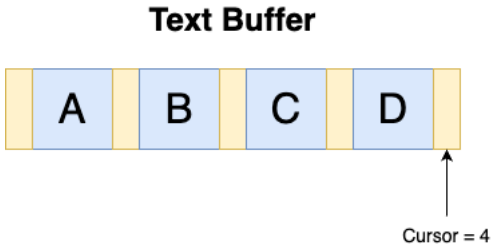
- Mỗi ký tự trong buffer được lưu bằng kiểu `char`.
- Con trỏ (`cursor`) dùng để xác định vị trí để thêm hoặc xoá một ký tự trong buffer, có thể di chuyển trái, phải, hoặc nhảy đến một vị trí bất kỳ, bắt đầu bằng 0. Vị trí cursor cuối cùng là sau ký tự cuối của TextBuffer.
- TextBuffer hỗ trợ undo/redo, giúp khôi phục hoặc làm lại các thao tác chỉnh sửa.



Hình 2: Minh hoạ TextBuffer với cursor ở vị trí đầu tiên

Class `TextBuffer` bao gồm các phương thức:

- `TextBuffer()`
  – **Chức năng:** Khởi tạo buffer rỗng, con trỏ ở vị trí đầu.

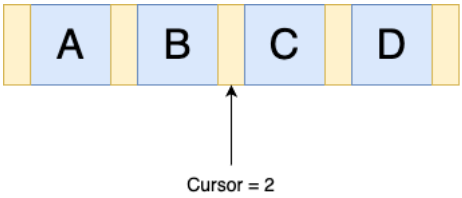


Hình 3: Minh hoạ TextBuffer với cursor ở vị trí cuối cùng

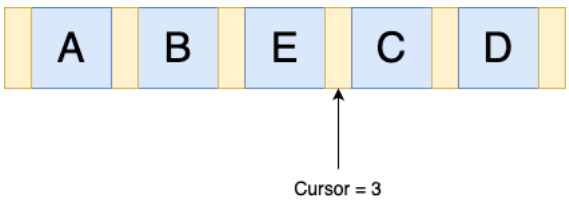
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(1)$.

- `~TextBuffer()`
  - **Chức năng:** Giải phóng toàn bộ bộ nhớ, tránh rò rỉ.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.

- `void insert(char c)`
  - **Chức năng:** Chèn ký tự `c` ngay trước vị trí con trỏ. Ví dụ, hình 4 và hình 5 mô tả buffer trước và sau khi chèn thêm 1 ký tự.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.



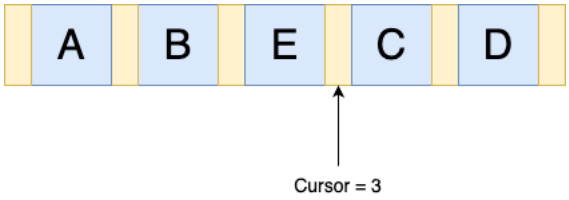
Hình 4: Minh hoạ TextBuffer trước khi chèn ký tự



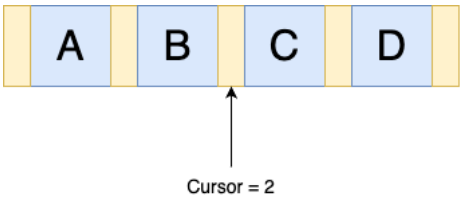
Hình 5: Minh hoạ TextBuffer sau khi chèn ký tự

- `void deleteChar()`

- **Chức năng:** Xóa ký tự ngay phía trước vị trí con trỏ. Ví dụ, hình 6 và hình 7 mô tả buffer trước và sau khi chèn xoá 1 ký tự.
- **Ngoại lệ:** Không có ngoại lệ.
- **Độ phức tạp:** $O(n)$.



Hình 6: Minh hoạ TextBuffer trước khi xoá ký tự



Hình 7: Minh hoạ TextBuffer sau khi xoá ký tự

- `void moveCursorLeft()`
  - **Chức năng:** Di chuyển con trỏ sang trái một đơn vị.
  - **Ngoại lệ:** Ném `cursor_error()` nếu con trỏ đang ở đầu.
  - **Độ phức tạp:** $O(1)$.
- `void moveCursorRight()`
  - **Chức năng:** Di chuyển con trỏ sang phải một đơn vị.
  - **Ngoại lệ:** Ném `cursor_error()` nếu con trỏ đang ở cuối.
  - **Độ phức tạp:** $O(1)$.
- `void moveCursorTo(int index)`
  - **Chức năng:** Di chuyển con trỏ đến vị trí `index`.
  - **Ngoại lệ:** Ném `out_of_range("Index is invalid!")` nếu `index` không hợp lệ.
  - **Độ phức tạp:** $O(1)$.
- `string getContent()`
  - **Chức năng:** Trả về toàn bộ nội dung buffer dưới dạng chuỗi.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.

- `int getCursorPos()`
  - **Chức năng:** Trả về vị trí hiện tại của con trỏ.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(1)$.
- `int findFirstOccurrence(char c)`
  - **Chức năng:** Trả về vị trí đầu tiên của ký tự `c`, hoặc -1 nếu không tồn tại.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.
- `int* findAllOccurrences(char c, int &count)`
  - **Chức năng:** Trả về mảng các vị trí chứa ký tự `c`, và số lượng vị trí lưu vào `count`.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.
- `void sortAscending()`
  - **Chức năng:** Sắp xếp toàn bộ buffer theo quy tắc bên dưới. Sau khi sắp xếp, cursor được đưa về vị trí đầu tiên.
  - Quy tắc sắp xếp: Sắp xếp theo thứ tự alphabet. Nếu có tồn tại cả chữ hoa và chữ thường, thì ký tự chữ hoa sẽ đứng trước ký tự chữ thường, sau đó sẽ tới ký tự alphabet tiếp theo.

    Ví dụ 3.2

    Buffer đang chứa các ký tự: **abcdfDA**
    Sau khi sắp xếp bằng phương thức `sortAscending`, buffer sẽ trở thành: **AabcDdf**

  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n \log n)$. Sinh viên bắt buộc phải hiện thực phương thức này bằng các giải thuật có độ phức tạp như yêu cầu.
- `void deleteAllOccurrences(char c)`
  - **Chức năng:** Xóa toàn bộ ký tự `c` xuất hiện trong buffer.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.
- `void undo()`
  - **Chức năng:** Thực hiện thao tác undo, khôi phục trạng thái trước đó. Chỉ áp dụng trên một số thao tác được mô tả kỹ bên dưới.

  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.

- `void redo()`
  - **Chức năng:** Thực hiện thao tác redo (làm lại thao tác đã undo). Sau khi chèn thêm 1 ký tự vào buffer thì không thể redo các hành động trước đó, mà bắt buộc phải chờ đến khi có các thao tác undo tiếp theo được thực hiện.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.

  Lưu ý: Với những thao tác có thay đổi chiều dài của buffer, phải cập nhật vi trí cursor phù hợp.

Ví dụ 3.3

```
Giả sử buffer ban đầu rỗng:
|
Thao tác 1: insert('A')
A|
Thao tác 2: insert('B')
AB|
Thao tác 3: insert('C')
ABC|
Thao tác 4: moveCursorLeft()
AB|C
Thao tác 5: insert('X')
ABX|C
Thao tác 6: moveCursorRight()
ABXC|
Thao tác 7: deleteChar()
ABX|
Thao tác 8: undo()  (khôi phục ký tự C đã xóa)
ABXC|
Thao tác 9: undo()  (xóa ký tự X vừa thêm)
AB|C
Thao tác 10: redo()  (thêm lại ký tự X)
ABX|C
Thao tác 11: redo() (xóa ký tự C lại)
ABX|
```

Lớp `HistoryManager` là lớp con (nested class) bên trong lớp `TextBuffer`. Lớp này được sử dụng để quản lý lịch sử các thao tác trên buffer, đồng thời hỗ trợ cho chức năng undo/redo của buffer.

Chức năng chính:

- Lưu trữ danh sách các hành động đã thực hiện, bao gồm thông tin về tên hành động, vị trí con trỏ, và ký tự liên quan (nếu có).
- Cho phép in toàn bộ lịch sử thao tác để phục vụ kiểm tra và minh họa.

- Hỗ trợ kiểm tra số lượng thao tác đang được lưu.

Phương thức chính:

- `HistoryManager()`
  - **Chức năng:** Khởi tạo danh sách lịch sử rỗng.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(1)$.
- `~HistoryManager()`
  - **Chức năng:** Giải phóng bộ nhớ của lịch sử, xoá toàn bộ các hành động đã lưu.
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.
- `void addAction(const string &actionName, int cursorPos, char c)`
  - **Chức năng:** Ghi lại một hành động mới vào danh sách lịch sử, cần lưu các thông tin: tên hành động, vị trí cursor trước khi thực hiện hành động, ký tự liên quan.
  - **Ngoại lệ:** Không có ngoại lệ.
  - Quy tắc định dạng 1 hành động được trình bày bên dưới.
  - **Độ phức tạp:** $O(1)$.
- `void printHistory()`
  - **Chức năng:** In toàn bộ danh sách các thao tác đã lưu ra màn hình, giúp kiểm tra và minh họa.
  - **Định dạng:** `[(<action name>, <cursor position>, <char>), (<action name>, <cursor position>, <char>, ...]`
  - **Ngoại lệ:** Không có ngoại lệ.
  - **Độ phức tạp:** $O(n)$.

Ví dụ 3.4

Với ví dụ 3.3, khi gọi hàm `printHistory()`, chương trình sẽ in ra:
`[(insert, 0, A), (insert, 1, B), (insert, 2, C), (move, 3, L), (insert, 2, X), (move, 3, R), (delete, 5, C)]`

- `int size()`
  - **Chức năng:** Trả về số lượng thao tác đang được lưu trong lịch sử.
  - **Ngoại lệ:** Không có ngoại lệ.

  - **Độ phức tạp:** $O(1)$.

Trong buffer, chỉ một số hành động có thể thực hiện undo/redo, đây cũng là các hành động cần được ghi vào lịch sử hành động, cụ thể:

- Hành động chèn ký tự:
  - **Tên hành động:** `insert`
  - **Ký tự liên quan:** ký tự được chèn vào.
- Hành động xoá ký tự:
  - **Tên hành động:** `delete`.
  - **Ký tự liên quan:** ký tự bị xoá đi.
  - **Lưu ý:** Chỉ lưu lại hành động xoá 1 ký tự.
- Hành động di chuyển cursor:
  - **Tên hành động:** `move`.
  - **Ký tự liên quan:**
    * Nếu di chuyển sang trái, lưu ký tự `'L'`.
    * Nếu di chuyển sang phải, lưu ký tự `'R'`.
    * Nếu di chuyển đến một vị trí `index`, lưu ký tự `'J'`.
- Hành động sắp xếp buffer:
  - **Tên hành động:** `sort`.
  - **Ký tự liên quan:** lưu ký tự '\0'.

4 Yêu cầu

Để hoàn thành bài tập này, sinh viên cần:

1. Đọc toàn bộ file mô tả này.
2. Tải file **initial.zip** và giải nén. Sau khi giải nén, sinh viên sẽ nhận được các file bao gồm: `main.cpp`, `main.h`, `TextBuffer.h`, `TextBuffer.cpp` và thư mục chứa output mẫu. Sinh viên chỉ nộp 2 file, đó là `TextBuffer.h` và `TextBuffer.cpp`. Do đó, không được phép chỉnh sửa file `main.h` khi kiểm tra chương trình.
3. Sinh viên sử dụng lệnh sau để biên dịch:
   `g++ -o main main.cpp TextBuffer.cpp -I . -std=c++17`
   Lệnh trên được sử dụng trong Command Prompt/Terminal để biên dịch chương trình.

Nếu sinh viên sử dụng IDE để chạy chương trình, cần lưu ý: thêm tất cả các file vào project/workspace của IDE; chỉnh lại lệnh biên dịch trong IDE cho phù hợp. IDE thường cung cấp nút Build (biên dịch) và Run (chạy). Khi bấm Build, IDE sẽ chạy câu lệnh biên dịch tương ứng, thông thường chỉ biên dịch file `main.cpp`. Sinh viên cần tìm cách cấu hình để thay đổi câu lệnh biên dịch, cụ thể: thêm file `TextBuffer.cpp`, thêm tùy chọn `-std=c++17`, và `-I .`

4. Chương trình sẽ được chấm trên nền tảng Unix. Môi trường của sinh viên và trình biên dịch có thể khác với môi trường chấm thực tế. Khu vực nộp bài trên LMS được thiết lập tương tự môi trường chấm thực tế. Sinh viên bắt buộc phải kiểm tra chương trình trên trang nộp bài, đồng thời sửa tất cả lỗi phát sinh trên hệ thống LMS để đảm bảo kết quả chính xác khi chấm cuối cùng.
5. Chỉnh sửa file `TextBuffer.h` và `TextBuffer.cpp` để hoàn thành bài tập, đồng thời đảm bảo hai yêu cầu sau:
   - Tất cả các phương thức được mô tả trong file hướng dẫn phải được cài đặt để chương trình có thể biên dịch thành công. Nếu sinh viên chưa hiện thực một phương thức nào đó, cần cung cấp phần hiện thực rỗng cho phương thức đó. Mỗi testcase sẽ gọi một số phương thức để kiểm tra kết quả trả về.
   - Trong file `TextBuffer.h` chỉ được phép có đúng một dòng `#include "main.h"`, và trong file `TextBuffer.cpp` chỉ được phép có một dòng `#include "TextBuffer.h"`. Ngoài hai dòng này, không được phép thêm bất kỳ lệnh `#include` nào khác trong các file này.
6. Khuyến khích sinh viên được phép viết thêm các lớp, phương thức và thuộc tính phụ trợ trong các lớp yêu cầu hiện thực. Nhưng sinh viên phải đảm bảo các lớp, phương thức này không làm thay đổi yêu cầu của các phương thức được mô tả trong đề bài.
7. Sinh viên phải thiết kế và sử dụng các cấu trúc dữ liệu đã học.
8. Sinh viên bắt buộc phải giải phóng toàn bộ vùng nhớ cấp phát động khi chương trình kết thúc.

5 Harmony cho Bài tập lớn

Bài kiểm tra cuối kì của môn học sẽ có một số câu hỏi Harmony với nội dung của BTL.

Sinh viên phải giải quyết BTL bằng khả năng của chính mình. Nếu sinh viên gian lận trong BTL, sinh viên sẽ không thể trả lời câu hỏi Harmony và nhận điểm 0 cho BTL.

Sinh viên **phải** chú ý làm câu hỏi Harmony trong bài kiểm tra cuối kỳ. Các trường hợp

không làm sẽ tính là 0 điểm cho BTL, và bị không đạt cho môn học. **Không chấp nhận giải thích và không có ngoại lệ.**

6 Quy định và xử lý gian lận

Bài tập lớn phải được sinh viên TỰ LÀM. Sinh viên sẽ bị coi là gian lận nếu:

- Có sự giống nhau bất thường giữa mã nguồn của các bài nộp. Trong trường hợp này, TẤT CẢ các bài nộp đều bị coi là gian lận. Do vậy sinh viên phải bảo vệ mã nguồn bài tập lớn của mình.
- Sinh viên không hiểu mã nguồn do chính mình viết, trừ những phần mã được cung cấp sẵn trong chương trình khởi tạo. Sinh viên có thể tham khảo từ bất kỳ nguồn tài liệu nào, tuy nhiên phải đảm bảo rằng mình hiểu rõ ý nghĩa của tất cả những dòng lệnh mà mình viết. Trong trường hợp không hiểu rõ mã nguồn của nơi mình tham khảo, sinh viên được đặc biệt cảnh báo là KHÔNG ĐƯỢC sử dụng mã nguồn này; thay vào đó nên sử dụng những gì đã được học để viết chương trình.
- Nộp nhầm bài của sinh viên khác trên tài khoản cá nhân của mình.
- Sinh viên sử dụng các công cụ AI trong quá trình làm bài tập lớn dẫn đến các mã nguồn giống nhau.

Trong trường hợp bị kết luận là gian lận, sinh viên sẽ bị điểm 0 cho toàn bộ môn học (không chỉ bài tập lớn).

KHÔNG CHẤP NHẬN BẤT KỲ GIẢI THÍCH NÀO VÀ KHÔNG CÓ BẤT KỲ NGOẠI LỆ NÀO!

Sau mỗi bài tập lớn được nộp, sẽ có một số sinh viên được gọi phỏng vấn ngẫu nhiên để chứng minh rằng bài tập lớn vừa được nộp là do chính mình làm.

Một số quy định khác:

- 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.
- Nội dung Bài tập lớn sẽ được Harmony với các câu hỏi trong bài kiểm tra cuối kì với nội dung tương tự.

——————————HẾT——————————