TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

---------------o0o---------------



**Bài tập lớn môn học**

**CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

Giảng viên hưỡng dẫn: TS. Hoàng Văn Thông

Sinh viên thực hiện: Nguyễn Hùng Cường

Lớp: CNTT VA 1

**Hà Nội tháng 10 năm 2023**

**Bài 1 (Bài số 5 trong danh sách bài tập)**

* 1. **Đề Bài**:

Xây dựng lớp xâu ký tự có các phương thức:

a. Cắt các ký trắng (dấu cách) ở hai đầu của xâu

Ví dụ: s = ” Ha Noi ”, sau khi cắt ký tự trắng s = “Ha Noi”

b. Cắt toàn bộ ký tự trắng thừa có trong xâu (giữa hai từ mà có n (n>2) ký tự trắng thì số ký tự trắng thừa là n-1.)

Ví dụ: s = ”Ha Noi”, sau khi cắt ký tự trắng s = “Ha Noi”

c. Chuyển đổi toàn bộ xâu ký tự thành xâu ký tự hoa

d. Chuyển đổi toàn bộ xâu ký tự thành xâu ký tự thường

e. Thực hiện nối thêm một xâu ký tự vào sau xâu ký tự hiện có

f. Phương thức trích ra n ký tự ở phía phải của xâu

g. Phương thức trích ra n ký tự ở phía trái của xâu

h. Phương thức trích ra n ký tự kể từ vị trí thứ i của xâu

i. Nhập

j. In

* 1. **Phân tích bài toán:**

Yêu cầu bài toán: xây dựng lớp xâu ký tự: class String

Các thuộc tính của lớp String:

* string str: chứa xâu ký tự được người dùng nhập vào

Các phương thức của lớp String:

* void trimFrontBack(): cắt các dấu cách ở hai đầu của xâu
* while (isspace(str[0])) { str.erase(0,1); }// kiểm tra xem ký tự đầu tiên của xâu có phải là dấu cách không, nếu đúng thì xóa ký tự đó
* while (isspace(str[str.length()-1])) { str.erase(str.length()-1); }// kiểm tra xem ký tự cuối cùng của xâu có phải là dấu cách không, nếu đúng thì xóa ký tự đó
* void trimExtraSpaces(): cắt các dấu cách thừa trong xâu
* for (int i=0; i<str.length(); i++) {

if (isspace(str[i]) && isspace(str[i-1])){

str.erase(i,1);

}

}

=> vòng lặp soát toàn bộ xâu ký tự, nếu một ký tự nào đó và ký tự trước nó đều là dấu cách thì xóa ký tự đó

* void uppercase(): chuyển toàn bộ xâu thành xâu ký tự hoa
* for (int i=0; i<str.length(); i++) {

str[i] = toupper(str[i]);

}

=> chuyển từng ký tự của xâu thành ký tự hoa

* void lowercase(): chuyển toàn bộ xâu thành xâu ký tự thường
* for (int i=0; i<str.length(); i++) {

str[i] = tolower(str[i]);

}

=> chuyển từng ký tự của xâu thành ký tự thường

* void append(): nối thêm một xâu ký tự vào sau xâu hiện có
* string apd;

cout << "Nhập xâu ký tự nối thêm: "; cin >> apd;

=> nhận xâu ký tự apd do người dùng nhập

* str += apd;

=> nối xâu apd vào sau xâu str hiện có

* string extractRight(int n): trích ra n ký tự ở bên phải của xâu
* if (n>str.length()){

return str;

}

=> trường hợp người dùng yêu cầu trích ra n ký tự từ một xâu có ít hơn n ký tự, trả lại kết quả là toàn bộ xâu

* else{

return str.substr(str.length()-n);

}

=> trả lại kết quả là xâu ký tự bắt đầu từ vị trí thứ n tính từ bên phải của xâu

* string extractLeft(int n): trích ra n ký tự ở bên trái của xâu
* if (n>str.length()){

return str;

}

=> tương tự với phương thức extractRight

* else{

return str.substr(0,n);

}

=> trả lại kết quả là xâu ký tự độ dài n tính từ vị trí đầu của xâu

* string extractMid(int i, int n): trích ra n ký tự kể từ vị trí i của xâu
* if (i+n>str.length()){

return str.substr(i, str.length()-i);

}

=> trường hợp người dùng yêu cầu trích ra một xâu n ký tự từ vị trí i mà từ vị trí i xâu không còn đủ n ký tự, trả lại kết quả là xâu từ vị trí i đến cuối xâu

* else{

return str.substr(i, n);

}

=> trả lại kết quả là xâu n ký tự bắt đầu từ vị trí i

* void input(): Nhập xâu ký tự
* cout << “Nhập xâu: “;

getline (cin, str);

=> nhận xâu ký tự str gồm đầy đủ các dấu cách từ bàn phím

* void output(): In xâu ký tự ra màn hình
* cout << str << endl; //In xâu str ra màn hình
  1. **Cài đặt chương trình:**

***https://github.com/XerAlix04/UTC-BTL-CTDLGT/blob/main/bai5.cpp***

* 1. **Phân tích thời gian chạy của từng phương thức có trong các lớp:**

Các phương thức của lớp String:

* void trimFrontBack(): độ phức tạp O(n)
* void trimExtraSpaces(): độ phức tạp O(n)
* void uppercase(): độ phức tạp O(n)
* void lowercase(): độ phức tạp O(n)
* void append(): độ phức tạp O(1)
* string extractRight(int n): độ phức tạp O(1)
* string extractLeft(int n): độ phức tạp O(1)
* string extractMid(int i, int n): độ phức tạp O(1)
* void input(): độ phức tạp O(1)
* void output(): độ phức tạp O(1)

**Bài 2 (Bài số 37 trong danh sách bài tập)**

* 1. **Đề Bài**:

Lập chương trình quản lý danh sách học sinh, chương trình có những chức năng sau: (Hồ sơ một học sinh giả sử có: Tên, lớp, số điện thoại, điểm TB)

a. Nhập danh sách học sinh từ bàn phím hay từ file.

b. In ra danh sách học sinh gồm có tên và xếp loại

c. In ra danh sách học sinh gồm các thông tin đầy đủ

d. Nhập vào từ bàn phím một tên học sinh và một tên lớp, tìm xem có học sinh có tên nhập vào trong lớp đó không? Nếu có thì in ra số điện thoại của học sinh đó

e. Nhập vào một hồ sơ học sinh mới từ bàn phím, bổ sung học sinh đó vào danh sách.

f. Nhập vào từ bàn phím tên một lớp, loại bỏ tắt cả các học sinh của lớp đó khỏi danh sách.

g. Sắp xếp danh sách học sinh theo thứ tự giảm dần của điểm trung bình.

h. Nhập vào hồ sơ một học sinh mới từ bàn phím, chèn học sinh đó vào danh sách mà không làm thay đổi thứ tự đã sắp xếp, in ra danh sách mới.

i. Lưu trữ lại trên đĩa danh sách học sinh khi đã thay đổi.

* 1. **Phân tích bài toán:**

Yêu cầu bài toán:

* Xây dựng cấu trúc struct Student chứa thông tin hồ sơ học sinh
* Xây dựng các lớp cài đặt cấu trúc dữ liệu danh sách liên kết kép và bộ duyệt xuôi ngược: class node, dlist\_iterator, dlist\_reverse\_iterator, dlist
* Xây dựng lớp chương trình quản lý danh sách học sinh: class Manager

Các thuộc tính của lớp node:

* Student elem: biến chứa các thông tin trong hồ sơ học sinh
* node \*next, \*prev: con trỏ trỏ đến node sau và node trước

Các phương thức của lớp node:

* node(Student x): constructor của lớp node
* next = NULL; prev = NULL; //khai báo 2 con trỏ next và prev
* elem.HoTen=x.HoTen; elem.Lop=x.Lop; elem.SDT=x.SDT; elem.Diem=x.Diem; //khai báo các thành phần của elem
* Student &getelem(): trả lại kết quả là tham chiếu đến elem
* node \*&getnext(); \*&getprev(): trả lại tham chiếu đến con trỏ next và prev
* void setelem(Student x): gán giá trị x vào biến elem
* void setnext(node \*p=nullptr); setprev(node \*p=nullptr): gán con trỏ p vào con trỏ next hoặc prev

Các thuộc tính của lớp dlist\_iterator:

* node \*curr: con trỏ trỏ đến node hiện tại

Các phương thức của lớp dlist\_iterator:

* dlist\_iterator(node \*c=0): constructor của lớp dlist\_iterator
* node \*getcurr(): trả lại con trỏ curr
* dlist\_iterator &operator=(dlist\_iterator it): gán giá trị bộ lặp it vào bộ lặp operator
* this->curr=it.getcurr(); //gán con trỏ curr trong it vào con trỏ curr của operator
* bool operator!=(dlist\_iterator it): kiểm tra xem giá trị bộ lặp operator có bằng bộ lặp it không
* Student &operator\*(): trả lại thông tin trong hồ sơ học sinh lưu trữ ở node hiện tại
* dlist\_iterator operator++(): tịnh tiến bộ lặp operator
* curr=curr->getnext(): //gán con trỏ trỏ đến node sau node hiện tại vào con trỏ curr

Các thuộc tính của lớp dlist\_reverse\_iterator:

* node \*curr: con trỏ trỏ đến node hiện tại

Các phương thức của lớp dlist\_reverse\_iterator:

* dlist\_reverse\_iterator(node \*c=0): constructor của lớp dlist\_reverse\_iterator
* node \*getcurr(); dlist\_reverse\_iterator &operator=(dlist\_reverse\_iterator it); bool operator!=(dlist\_reverse\_iterator it), Student &operator\*(): tương tự như lớp dlist\_iterator
* dlist\_reverse\_iterator operator++(): tịnh tiến bộ lặp ngược operator
* curr=curr->getprev(): //gán con trỏ trỏ đến node trước node hiện tại vào con trỏ curr

Các thuộc tính của lớp dlist:

* node \*head, \*tail: con trỏ trỏ đến node đầu tiên và node cuối cùng của danh sách
* unsigned int num: độ lớn của danh sách

Các phương thức của lớp dlist:

* typedef dlist\_iterator iterator; typedef dlist\_reverse\_iterator reverse\_iterator; định nghĩa dlist::iterator là tên gọi khác cho lớp dlist\_iterator, dlist::reverse\_iterator là tên gọi khác cho lớp dlist\_reverse\_iterator
* iterator begin(); iterator end(); reverse\_iterator rbegin(); reverse\_iterator rend(); trả lại điểm bắt đầu và điểm kết thúc cho bộ lặp xuôi và bộ lặp ngược
* dlist(): constructor của lớp dlist
* num=0; head=tail=nullptr; //khai báo biến num lưu độ lớn của danh sách và con trỏ trỏ đến node đầu và node cuối của danh sách; tạo một danh sách liên kết kép rỗng
* dlist(int k, Student x): tạo danh sách có k phần tử giá trị x
* while (k--) push\_back(x); //vòng lặp chạy k lần, dùng phương thức void push\_back(Student x) thêm k phần tử giá trị x vào danh sách
* unsigned size(): trả lại kết quả là độ lớn của danh sách
* void push\_back(Student x): thêm một phần tử giá trị x vào cuối danh sách
* if (num==0) head=tail=new node(x); // nếu danh sách là danh sách rỗng, tạo phần tử mới giá trị x, gán cho hai con trỏ trỏ vào phần tử đầu và cuối danh sách cùng trỏ vào phần tử mới này
* else{

tail->setnext(new node(x));

tail->getnext()->setprev(tail);

tail=tail->getnext();

}

=> tạo phần tử mới giá trị x, liên kết phần tử mới này với phần tử cuối trước đó của danh sách, đặt lại con trỏ tail để trỏ vào phần tử mới này

* num++; //update độ lớn của danh sách
* void push\_front(Student x): thêm một phần tử giá trị x vào đầu danh sách
* if (num==0) head=tail=new node(x); tương tự với push\_back(Student x)
* else{

head->setprev(new node(x));

head->getprev()->setnext(head);

head=head->getprev();

}

=> tạo phần tử mới giá trị x, liên kết phần tử mới này với phần tử đầu trước đó của danh sách, đặt lại con trỏ head để trỏ vào phần tử mới này

* num++; //update độ lớn của danh sách
* void pop\_back(): xóa một phần tử ở cuối danh sách
* if(num==0) return; if(num==1) head=tail=nullptr; //nếu danh sách rỗng thì trả lại kết quả trống, nếu danh sách chỉ có một phần tử thì gán cho 2 con trỏ head và tail là con trỏ null, biến danh sách thành danh sách rỗng
* else{

tail=tail->getprev();

tail->setnext(0);

}

=> gán cho con trỏ tail trỏ vào phần tử trước phần tử cuối cần xóa, cắt liên kết giữa phần tử cuối hiện tại với phần tử cần xóa

* num--; //update độ lớn của danh sách
* void pop\_front(): xóa một phần tử ở đầu danh sách
* if(num==0) return; if(num==1) head=tail=nullptr; tương tự pop\_back()
* else{

head=head->getnext();

head->setprev(0);

}

=> gán cho con trỏ head trỏ vào phần tử sau phần tử đầu cần xóa, cắt liên kết giữa phần tử đầu hiện tại với phần tử cần xóa

* num--; //update độ lớn của danh sách
* void insert(node \*p, Student x): chèn một phần tử giá trị x vào sau phần tử được trỏ vào bởi con trỏ p ở giữa danh sách
* if (num==0) head=tail=new node(x); //nếu danh sách rỗng, tạo phần tử mới giá trị x, gán cho hai con trỏ trỏ vào phần tử đầu và cuối danh sách cùng trỏ vào phần tử mới này
* else{

node \*q; //tạo con trỏ node mới q

q->setelem(x); //gán giá trị x vào phần tử được trỏ vào bởi q

q->setnext(p->getnext());

p->getnext()->setprev(q); //liên kết phần tử được trỏ vào bởi q với phần tử sau phần tử được trỏ vào p

q->setprev(p);

p->setnext(q); //liên kết phần tử được trỏ vào bởi p với phần tử được trỏ vào bởi p

}

* num++; update độ lớn danh sách
* void remove(node \*p): xóa một phần tử được trỏ vào bởi con trỏ p ở giữa danh sách
* p->getprev()->setnext(p->getnext());
* p->getnext()->setprev(p->getprev());
* p->setnext(NULL);
* p->setprev(NULL);
* delete p;
* num--;

=> cắt liên kết giữa phần tử cần xóa với các phần tử trước và sau nó, liên kết phần tử trước và sau phần tử cần xóa với nhau, xóa con trỏ p, cuối cùng update độ lớn danh sách

Các thuộc tính của lớp Manager:

* node \*head, \*tail; con trỏ đến phần tử đầu và cuối danh sách
* dlist HS; danh sách liên kết kép chứa hồ sơ học sinh

Các phương thức của lớp Manager:

* void NhapDanhSachHS(int n): nhập danh sách học sinh bằng bàn phím hoặc bằng file, biến n để chọn giữa 2 cách nhập danh sách
* Student x; int m; //khai báo biến x lưu thông tin học sinh và biến số nguyên m
* if (n==1) { //nhập bằng bàn phím

cout << "Số học sinh cần nhập: "; cin >> m; // nhập số học sinh cần nhập, đưa vào biến m

for (int i=0; i<m; i++){ // vòng lặp m lần

cout << "Nhập tên học sinh thứ " << m << ": "; getline (cin, x.HoTen);

cout << "Nhập lớp: "; getline (cin, x.Lop);

cout << "Nhập số điện thoại: "; cin >> x.SDT;

cout << "Nhập điểm trung bình: "; cin >> x.Diem;

HS.push\_back(x);

}

=> đưa thông tin học sinh do người dùng nhập vào biến x, dùng phương thức dlist::push\_back(Student x) thêm hồ sơ học sinh có thông tin là x vào danh sách

}

* if (n==2){ // nhập bằng file

string tenfile; // khai báo biến xâu tenfile để nhận tên file

cout << "Nhập tên file: "; cin >> tenfile; //người dùng nhập tên file

vào biến tenfile

ifstream file(tenfile); // lớp ifstream để đọc từ file

if (!file.is\_open()) {

cout << "Không thể mở tệp" << endl;

}

=> thông báo nếu file không mở được

while (file >> x.HoTen >> x.Lop >> x.SDT >> x.Diem){

HS.push\_back(x);

}

=> đọc hồ sơ của từng học sinh từ file, đưa vào biến x, dùng phương thức push\_back thêm hồ sơ vào danh sách

file.close(); // đóng file

}

* void InXepLoaiHS(): in danh sách học sinh gồm tên và xếp loại
* Student x; //khai báo biến x lưu thông tin học sinh
* for (dlist::iterator it = HS.begin(); it != HS.end(); it++){ //dùng bộ lặp xuôi it thực hiện vòng lặp từ đầu đến cuối danh sách HS

node \*t = it.getcurr();

x = t->getelem(); // đưa thông tin từng học sinh vào x

cout << x.HoTen << "\n" << "Xếp loại: ";

if (x.Diem>=8.5) cout << "Học sinh Giỏi";

else if (x.Diem>=7.0) cout << "Học sinh Khá";

else if (x.Diem>=5.5) cout << "Học sinh Trung bình";

else if (x.Diem>=4.0) cout << "Học sinh Trung bình yếu";

else cout << "Học sinh Kém"; //in họ tên và xếp loại từng học sinh ra màn hình

}

* void InDanhSachHS(): in danh sách học sinh đầy đủ
* for (dlist::iterator it = HS.begin(); it != HS.end(); it++){//vòng lặp soát từ đầu đến cuối danh sách HS

node \*t = it.getcurr();

x = t->getelem(); // đưa thông tin từng học sinh vào x

cout << x.HoTen << "\n" << x.Lop << "\n" << x.SDT << "\n" << x.Diem;

}

=> In thông tin từng học sinh ra màn hình

* void TimHS(string name, string cl): tìm học sinh có tên trùng với xâu name và tên lớp trùng với xâu cl, in ra số điện thoại của học sinh đó
  + Student x; int i=0; //khai báo biến thông tin học sinh x, biến i kiểm tra xem có tìm được học sinh cần tìm không
  + for (dlist::iterator it=HS.begin(); it != HS.end(); it++){//soát danh sách HS

node \*t = it.getcurr();

x = t->getelem(); //đưa thông tin từng học sinh vào x

if (x.HoTen==name && x.Lop==cl) cout << x.SDT; i++;

break; //nếu học sinh nào trùng khớp thì in số điện thoại, tăng biến i rồi thoát khỏi vòng lặp

}

* if (i==0) cout << "Không có học sinh nào tên " << name << " trong lớp " << cl; //nếu biến i=0, tức là không tìm được học sinh nào trùng khớp, thì báo cho người dùng
* void AddHS(string name, string cl, int phone, float grade): thêm học sinh mới có tên name, lớp cl, số điện thoại phone, điểm trung bình grade vào danh sách; sử dụng phương thức dlist::push\_back(Student x)
* void DeleteLop(string cl): xóa tất cả các học sinh của lớp cl khỏi danh sách
  + for (dlist::iterator it=HS.begin(); it != HS.end(); it++){//soát danh sách HS

node \*t = it.getcurr();

x = t->getelem(); //đưa thông tin từng học sinh vào x

if (x.Lop==cl) HS.remove(t); // nếu học sinh nào thuộc lớp cl, sử dụng phương thức dlist::remove(node \*p) để xóa học sinh đó khỏi danh sách

}

* void SortHS(): bắt đầu thực hiện sắp xếp danh sách học sinh; gọi phương thức void quickSort (node \*h, node \*t)
  + node \*h=head; node \*t=tail; con trỏ h và t trỏ vào 2 đầu danh sách
* node \*Partition (node \*h, node \*t): thuật toán phân hoạch danh sách liên kết, đặt phần tử chốt vào đúng vị trí, tất cả các phần tử có điểm trung bình thấp hơn đặt về bên phải, cao hơn đặt về bên trái chốt
  + Student x = h->getelem(); //đặt phần tử chốt là phần tử đầu danh sách
  + Student y; //khai báo biến y
  + node \*i = h->getprev(); //con trỏ i trỏ về trước phần tử đầu danh sách
  + int inum=0; int jnum=HS.size()+1; //để so sánh vị trí i và j trong danh sách
  + for (dlist::reverse\_iterator it = HS.rbegin(); it != HS.rend(); it++){ //bộ lặp ngược it soát danh sách HS

node \*j = it.getcurr(); //con trỏ j chạy theo it

jnum--;

y = j->getelem(); //đưa thông tin học sinh vào y

if (y.Diem>=x.Diem){

i = (i==NULL)? h : i->getnext(); inum++;

swap(&(i->getelem()), &(j->getelem()));

}

if (inum>jnum) break;

=> j chạy cho đến khi điểm trung bình của y >= chốt, khi đó i chạy sang phần tử kế tiếp, đổi giá trị giữa i và j. Quá trình lặp lại cho đến khi i>j.

}

* i = (i==NULL)? h : i->getnext(); inum++; //i chạy sang phần tử kế tiếp

swap(&(h->getelem()), &(j->getelem())); //đổi giá trị giữa j và phần tử đầu

return i;//trả lại kết quả là con trỏ i

* void quickSort (node \*h, node \*t): thuật toán sắp xếp nhanh
  + node \*p = Partition(h,t); //thuật toán phân hoạch chia danh sách làm 2 phần
  + quickSort(h, p->getprev());
  + quickSort(p->getnext(), t); //thực hiện đệ qui với 2 phần đó, tiếp tục chia làm 2 phần nữa cho đến khi danh sách cần phân hoạch chỉ có 1 phần tử
* void swap(Student \*a, Student \*b): đổi giá trị giữa 2 phần tử
* void InsertHS(string name, string cl, int phone, float grade): chèn một học sinh mới tên name, lớp cl, số điện thoại phone, điểm trung bình grade vào giữa danh sách mà không làm thay đổi thứ tự đã sắp xếp
  + Student x, y, z;// khai báo biến x, y, z
  + for (dlist::iterator it=HS.begin(); it != HS.end(); it++){//soát danh sách HS

node \*t = it.getcurr();

if (t!=tail){

node \*v = t->getnext();

y = t->getelem();

z = v->getelem(); //y và z là cặp 2 phần tử liên tiếp

if (x.Diem<=y.Diem&&x.Diem>=z.Diem){

HS.insert(t, x);

break;

}

=> nếu điểm trung bình của x ở giữa y và z, thực hiện chèn x vào danh sách bằng phương thức dlist::insert(node \*p, Student x)

}

}

* void Save(): Lưu danh sách liên kết vào file
  + string filename; Student x; //khai báo biến
  + cout << "Nhập tên file: "; cin >> filename;
  + ofstream file(filename);
  + if (!file.is\_open()) {

cout << "Không thể mở tệp" << endl;

}

* + for (dlist::iterator it=HS.begin(); it != HS.end(); it++){//soát danh sách HS

node \*t = it.getcurr();

x = t->getelem(); //đưa thông tin từng học sinh vào x

file << x.HoTen << "\n" << x.Lop << "\n" << x.SDT << "\n" << x.Diem; //nhập thông tin từng học sinh vào file

* + file.close(); // đóng file
  1. **Cài đặt chương trình:**

***https://github.com/XerAlix04/UTC-BTL-CTDLGT/blob/main/bai37.cpp***

* 1. **Phân tích thời gian chạy của từng phương thức có trong các lớp:**

Các phương thức của lớp node:

* node(Student x): độ phức tạp O(1)
* Student &getelem(): độ phức tạp O(1)
* node \*&getprev(): độ phức tạp O(1)
* node \*getnext(): độ phức tạp O(1)
* void setelem(Student x): độ phức tạp O(1)
* void setprev(): độ phức tạp O(1)
* void setnext(): độ phức tạp O(1)

Các phương thức của lớp dlist\_iterator:

* dlist\_iterator(node \*c=0): độ phức tạp O(1)
* node \*getcurr(): độ phức tạp O(1)
* dlist\_iterator &operator=(dlist\_iterator it): độ phức tạp O(1)
* bool operator!=(dlist\_iterator it): độ phức tạp O(1)
* Student &operator\*(): độ phức tạp O(1)
* dlist\_iterator operator++(): độ phức tạp O(1)

Các phương thức của lớp dlist\_reverse\_iterator:

* dlist\_reverse\_iterator(node \*c=0): độ phức tạp O(1)
* node \*getcurr(): độ phức tạp O(1)
* dlist\_reverse\_iterator &operator=(dlist\_reverse\_iterator it): độ phức tạp O(1)
* bool operator!=(dlist\_reverse\_iterator it): độ phức tạp O(1)
* Student &operator\*(): độ phức tạp O(1)
* dlist\_reverse\_iterator operator++(): độ phức tạp O(1)

Các phương thức của lớp dlist:

* iterator begin(): độ phức tạp O(1)
* iterator end(): độ phức tạp O(1)
* reverse\_iterator rbegin(): độ phức tạp O(1)
* reverse\_iterator rend(): độ phức tạp O(1)
* dlist(): độ phức tạp O(1)
* dlist(int k, Student x): độ phức tạp O(n)
* unsigned size(): độ phức tạp O(1)
* void push\_back(Student x): độ phức tạp O(1)
* void push\_front(Student x): độ phức tạp O(1)
* void pop\_back(): độ phức tạp O(1)
* void pop\_front(): độ phức tạp O(1)
* void insert(node \*p, Student x): độ phức tạp O(1)
* void remove(node \*p): độ phức tạp O(1)

Các phương thức của lớp Manager:

* void NhapDanhSachHS(int n): độ phức tạp O(n)
* void InXepLoaiHS(): độ phức tạp O(n)
* void InDanhSachHS(): độ phức tạp O(n)
* void TimHS(string name, string cl): độ phức tạp O(n)
* void AddHS(string name, string cl, int phone, float grade): độ phức tạp O(1)
* void DeleteLop(string cl): độ phức tạp O(n)
* node \*Partition (node \*h, node \*t): độ phức tạp O(n)
* void quickSort (node \*h, node \*t): độ phức tạp O(nlogn)
* void swap(Student \*a, Student \*b): độ phức tạp O(1)
* void InsertHS(string name, string cl, int phone, float grade): độ phức tạp O(n)
* void Save(): độ phức tạp O(n)