**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQG.TPHCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

Logo

Description automatically generated

**BÁO CÁO BÀI TẬP VỀ NHÀ TUẦN 2**

**LINKED LIST**

| Giảng viên thực hành |

Lê Tuấn Thu

| Nhóm thực hiện |

Nguyễn Huy Hoàng – 21120458

Nguyễn Ngọc Như Huyền - 21120475

Thành phố Hồ Chí Minh – 2022

**DANH SÁCH LIÊN KẾT ĐƠN VỚI DỮ LIỆU ĐẦU VÀO LÀ SỐ NGUYÊN, SỐ THỰC, CHUỖI KÍ TỰ, CẤU TRÚC SINH VIÊN**

* **NHÓM HÀM DATA**
* Là nơi chứa những hằng số, cấu trúc dữ liệu do người lập trình định nghĩa dùng xuyên suốt chương trình.
* **Khai báo thư viện**

|  |  |
| --- | --- |
|  | #include<iostream> |
|  | #include<fstream> |
|  | #include<string> |
|  | using namespace std; |

* **Thư viện fstream:** sử dụng để đọc/ghi file.
* **Thư viện string:** sử dụng kiểu dữ liệu string và dùng hàm getline();
* **Khai báo các cấu trúc**

|  |  |
| --- | --- |
|  | /\*Khai bao kieu du lieu sinh vien\*/ |
|  | struct sv { |
|  | string hoten; |
|  | string id; |
|  | double lt=0; |
|  | double th=0; |
|  | double tb=0; |
|  | }; |
|  | /\*Khai bao kieu du lieu Node\*/ |
|  | template <class T> |
|  | struct Node { |
|  | T data; |
|  | Node\* next; |
|  | }; |
|  |  |
|  | /\*Khai bao kieu du lieu List\*/ |
|  | template <class T> |
|  | struct List { |
|  | Node<T>\* head; |
|  | Node<T>\* tail; |
|  | }; |

* **Struct sv:** lưu các thông tin cần có của một sinh viên (họ tên, mssv, điểm lý thuyết, điểm thực hành, điểm trung bình).
* **Struct Node:** lưu thông tin của một node (dữ liệu mà node chứa, con trỏ trỏ đến node kế tiếp).
* **Struct List:** cấu trúc danh sách liên kết với hai con trỏ (đầu và cuối) trỏ đến vị trí đầu và vị trí cuối của danh sách.
* **CÁC HÀM KHỞI TẠO VÀ GIẢI PHÓNG**

|  |  |
| --- | --- |
|  | /\*Khoi tao List\*/ |
|  | template <class T> |
|  | void ListInit(List<T>& l) { |
|  | l.head = l.tail = NULL; |
|  | } |
|  |  |
|  | /\*Khoi tao Node\*/ |
|  | template <class T> |
|  | Node<T>\* getNode(T DATA) { |
|  | Node<T>\* p = new Node<T>; |
|  | if (p != NULL) { |
|  | p->data = DATA; |
|  | p->next = NULL; |
|  | } |
|  | return p; |
|  | } |
|  | /\*Giai phong danh sach lien ket\*/ |
|  | template<class T> |
|  | void GiaiPhong(List<T>& l) { |
|  | Node\* p; |
|  | while (l.head != NULL) { |
|  | p = l.head; |
|  | l.head = l.head->next; |
|  | delete p; |
|  | } |
|  | } |

* **ListInit():** khởi tạo danh sách liên kết rỗng bằng việc gán NULL cho hai con trỏ đầu và cuối.
* **getNode():** Tạo một node với các bước:

1. Khởi tạo vùng nhớ cho node và cho con trỏ kiểu Node trỏ đến vùng nhớ vừa tạo được.
2. Kiểm tra vùng nhớ tạo thành công không bằng cách kiểm tra con trỏ có khác NULL không.
3. Nếu khác NULL ta thêm dữ liệu cho node và gán NULL cho con trỏ next trong node.
4. Cuối cùng, trả về địa chỉ của node.

* **GiaiPhong():** giải phóng danh sách liên kết bằng cách giải phóng từng node duyệt từ đầu cho đến cuối danh sách.
* **CÁC HÀM NHẬP, XUẤT**
* **Hàm nhập thông tin một sinh viên**

|  |  |
| --- | --- |
|  | void nhap1Sv(sv& sinhvien) { |
|  | cout << "Nhap Id: "; |
|  | cin.ignore(); |
|  | getline(cin, sinhvien.id); |
|  | cout << "Nhap ho ten: "; |
|  | getline(cin, sinhvien.hoten); |
|  | cout << "Nhap diem ly thuyet: "; |
|  | cin >> sinhvien.lt; |
|  | cout << "Nhap diem thuc hanh: "; |
|  | cin >> sinhvien.th; |
|  | sinhvien.tb = sinhvien.lt \* 0.7 + sinhvien.th \* 0.3; |
|  | } |

* **Nạp chồng toán tử (dùng cho xuất danh sách sinh viên)**

|  |  |
| --- | --- |
|  | ostream& operator<<(ostream& COUT, sv sinhvien) { |
|  | COUT << "Ho ten: " << sinhvien.hoten << endl; |
|  | COUT << "Id: " << sinhvien.id << endl; |
|  | COUT << "Diem ly thuyet: " << sinhvien.lt << endl; |
|  | COUT << "Diem thuc hanh: " << sinhvien.th << endl; |
|  | COUT << "Diem trung binh: " << sinhvien.tb << endl; |
|  | return COUT; |
|  | } |

* Nạp chồng toán tử dùng cho việc sử dụng những toán tử căn bản cho những kiểu dữ liệu đặc biệt (do người lập trình định nghĩa).
* Nạp chồng toán tử << ta không cần phải viết hàm xuất danh sách cho kiểu dữ liệu sinh viên.
* **Hàm xuất danh sách liên kết**

|  |  |
| --- | --- |
|  | template <class T> |
|  | void printList(List<T> l) { |
|  | int i = 0; |
|  | for (Node<T>\* r = l.head; r != NULL; r = r->next) { |
|  | cout << "Node " << i << endl; |
|  | cout << r->data << endl; |
|  | i++; |
|  | } |
|  | } |

* **HÀM ĐỌC THÔNG TIN SINH VIÊN TỪ FILE**

|  |  |
| --- | --- |
|  | /\*Doc thong tin 1 sinh vien tu file\*/ |
|  | void docSv(ifstream& filein, sv& sinhvien) { |
|  | //doc file den dau ; thi ngung |
|  | getline(filein, sinhvien.hoten, ';'); |
|  | getline(filein, sinhvien.id, ';'); |
|  | //doc so thuc |
|  | filein >> sinhvien.lt; |
|  | //bo qua dau ; |
|  | filein.ignore(1, ';'); |
|  | filein >> sinhvien.th; |
|  | sinhvien.tb = sinhvien.lt \* 0.7 + sinhvien.th \* 0.3; |
|  | filein.ignore(); |
|  | } |
|  |  |
|  | /\*Doc danh sach sinh vien\*/ |
|  | void docDssv(ifstream& filein, List<sv>& l) { |
|  | sv sinhvien; |
|  | ListInit(l);//tao danh sach rong |
|  | //doc ky tu den cuoi file thi ngung |
|  | while (filein.peek() != EOF) { |
|  | docSv(filein, sinhvien); |
|  | addLast(l, sinhvien); |
|  | } |
|  | } |

* **docSv():** Hàm đọc thông tin một sinh viên với dữ liệu đầu vào là file text và biến sinhvien lưu thông tin vừa đọc.
* **getline():** Đọc hết dòng đến dấu ; thì ngừng.
* **>> :** đọc số thực.
* **ignore():** bỏ qua dấu ;
* **docDssv():** Hàm đọc danh sách sinh viên từ file text và lưu vào danh sách liên kết.
* **peek():** Đọc ký tự (không lưu ký tự) đến cuối file gặp EOF thì ngừng.
* **CÁC HÀM THÊM**
* **Hàm thêm cuối**

|  |  |
| --- | --- |
|  | template <class T> |
|  | void addLast(List<T>& l, T DATA) { |
|  | Node<T>\* p = getNode(DATA); |
|  | if (l.head == NULL) |
|  | l.head = l.tail = p; |
|  | else { |
|  | l.tail->next = p; |
|  | l.tail = p; |
|  | } |
|  | } |

* **Bước 1:** Cho một con trỏ (giả sử p) giữ địa chỉ của node mới tạo thông qua hàm getNode().
* **Bước 2:** Kiểm tra danh sách có rỗng không bằng việc kiểm tra địa chỉ mà con trỏ head đang giữ là NULL hay khác NULL.
* Nếu danh sách rỗng: địa chỉ của con trỏ p đang giữ cũng chính là địa chỉ của con trỏ head và tail đang giữ.
* Nếu danh sách không rỗng: gán địa chỉ mà con trỏ kế tiếp của con trỏ tail đang giữ bằng địa chỉ p đang giữ và gán lại địa chỉ con trỏ tail đang giữ bằng địa chỉ p đang giữ.
* **Hàm thêm đầu**

|  |  |
| --- | --- |
|  | template <class T> |
|  | void addFirst(List<T>& l, T DATA) { |
|  | Node<T>\* p = getNode(DATA); |
|  | if (l.head == NULL) |
|  | l.head = l.tail = p; |
|  | else { |
|  | p->next = l.head; |
|  | l.head = p; |
|  | } |
|  | } |

* **Bước 1:** Cho một con trỏ (giả sử p) giữ địa chỉ của node mới tạo thông qua hàm getNode().
* **Bước 2:** Kiểm tra danh sách có rỗng không bằng việc kiểm tra địa chỉ mà con trỏ head đang giữ là NULL hay khác NULL.
* Nếu danh sách rỗng: địa chỉ của con trỏ p đang giữ cũng chính là địa chỉ của con trỏ head và tail đang giữ.
* Nếu danh sách không rỗng: gán địa chỉ mà con trỏ kế tiếp của con trỏ p đang giữ bằng địa chỉ mà con trỏ head đang giữ và gán lại địa chỉ con trỏ head đang giữ bằng địa chỉ con trỏ p đang giữ.
* **Hàm thêm vào một vị trí bất kỳ**

|  |  |
| --- | --- |
|  | template <class T> |
|  | void add(List<T>& l, T DATA, int pos) { |
|  | /\*Dem so node co trong List\*/ |
|  | int node = countNode(l); |
|  |  |
|  | /\*Tao node can them\*/ |
|  | Node<T>\* newNode = getNode(DATA); |
|  |  |
|  | /\*Kiem tra vi tri them\*/ |
|  | if (pos < 0 || pos > node) { |
|  | do { |
|  | cout << "Nhap lai vi tri them: "; cin >> pos; |
|  | } while (pos < 0 || pos > node); |
|  | } |
|  | else if (pos == 0) |
|  | addFirst(l, DATA); |
|  | else if (pos == node) |
|  | addLast(l, DATA); |
|  | else { |
|  | Node<T>\* r; int i; |
|  | /\*Tao node t luu dia chi node truoc vi tri can them\*/ |
|  | Node<T>\* t = new Node<T>; |
|  | for (i = 0, r = l.head; i < pos; i++, r = r->next) |
|  | if (i == pos - 1) |
|  | t = r; |
|  | t->next = newNode; |
|  | newNode->next = r; |
|  | } |
|  | } |

* **Bước 1:** Hàm tạo node mới và thêm thông tin cho node.
* **Bước 2:** Kiểm tra vị trí cần thêm để lựa chọn cách thêm cho phù hợp. Nếu vị trí không hợp lệ thì mời nhập lại cho đến khi vị trí phù hợp.
* **Bước 3:** Tạo node t giữ địa chỉ của node trước vị trí cần thêm bằng vòng lặp for, r giữ địa chỉ node sau vị trí cần thêm. Khi đó con trỏ trỏ đến node kế tiếp của t sẽ là địa chỉ của node mới và con trỏ trỏ đến node kế tiếp của node mới sẽ là địa chỉ r.
* **CÁC HÀM XÓA**
* **Hàm xóa cuối**

|  |  |
| --- | --- |
|  | template<class T> |
|  | void delLast(List<T>& l) |
|  | { |
|  | //duyet cac phan tu co trong danh sach |
|  | for (Node<T>\* k = l.head; k != NULL; k = k->next) |
|  | { |
|  | //neu duyet den phan tu pTail cuoi trong danh sach |
|  | if (k->next == l.tail) |
|  | { |
|  | //xoa phan tu cuoi |
|  | delete l.tail; |
|  | //tro phan tu truoc phan tu cuoi ve NULL |
|  | k->next = NULL; |
|  | //thay doi lai phan tu cuoi pTail cua danh sach |
|  | l.tail = k; |
|  | } |
|  | } |
|  | } |

* **Bước 1:** Tạo một node k và thực hiện duyệt toàn bộ phần tử có trong danh sách liên kết đơn
* **Bước 2:** Kiểm tra duyệt, nếu thực hiện duyệt đến phần tử cuối của danh sách nghĩa là k == ds.pTail thì thực hiện xóa phần tử đó: delete ds.pTail;
* **Bước 3:** Trỏ phần tử đứng trước pTail bằng NULL: k->next = NULL;
* **Bước 4:** Thay đổi lại pTail của danh sách bằng node k
* **Hàm xóa đầu**

|  |  |
| --- | --- |
|  | template<class T> |
|  | void delBegin(List<T>& l) { |
|  | //tao node p |
|  | Node<T>\* p = new Node<T>; |
|  | //gan p bang node pHead dau tien cua danh sach |
|  | p = l.head; |
|  | //thay doi lai pHead cua danh sach |
|  | l.head = l.head->next; |
|  | //gan node p ban dau tro den NULL |
|  | p->next = NULL; |
|  | //xoa node p |
|  | delete p; |
|  | } |

* **Bước 1:** Tạo một node p và gán bằng pHead hiện tại của danh sách: p = ds.pHead
* **Bước 2:** Đặt lại phần tử pHead của danh sách bằng phần tử kế sau pHead hiện tại của danh sách: ds.pHead = ds.pHead->next
* **Bước 3:** Gán node p được gán bằng pHead ở bước thứ nhất trỏ đến NULL: p->next = NULL
* **Bước 4:** Xóa đi node p được gán bằng pHead ở bước thứ nhất: delete p
* **Hàm xóa ở vị trí bất kỳ**

|  |  |
| --- | --- |
|  | template <class T> |
|  | void Delete(List<T>& l, int pos) { |
|  | int node = countNode(l); |
|  | if (pos < 0 || pos >= node) { |
|  | do { |
|  | cout << "Nhap lai vi tri xoa: "; cin >> pos; |
|  | } while (pos < 0 || pos > node); |
|  | } |
|  | else if (pos == 0) |
|  | delBegin(l); |
|  | else if (pos == node - 1) |
|  | delLast(l); |
|  | else { |
|  | Node<T>\* r; int i; |
|  | Node<T>\* t = l.head; |
|  | for (i = 0, r = l.head; i < pos; i++, r = r->next) { |
|  | if (i == pos - 1) |
|  | t = r; |
|  | } |
|  | t->next = r->next; |
|  | //delete[]t; |
|  | } |
|  | } |

* **Bước 1:** Kiểm tra vị trí cần xóa để lựa chọn cách xóa cho phù hợp. Nếu vị trí không hợp lệ thì mời nhập lại cho đến khi vị trí phù hợp.
* **Bước 2:** Tạo node t giữ địa chỉ của node trước vị trí cần xóa bằng vòng lặp for, r giữ địa chỉ node sau vị trí cần xóa.
* **Bước 3:** Cho con trỏ trỏ đến node kế tiếp của t giữ địa chỉ của con trỏ trỏ đến node kế tiếp của r. Khi đó node tại vị trí xóa sẽ mất liên kết với danh sách.
* **HÀM MENU ĐỂ CHẠY CHƯƠNG TRÌNH**
* Hàm menu chính được gọi trong hàm main để chạy chương trình, trong hàm menu chính có gọi 4 hàm menu ứng với 4 kiểu dữ liệu int, float, string, struct sv

**Text

Description automatically generatedText

Description automatically generated**

* **HÀM MAIN()**
* Gọi lại hàm Menu() để thực thi chương trình

**Text

Description automatically generated**