TRƯỜNG ĐẠI HỌC GIAO THÔNG VÂN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

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



**Bài tập lớn môn học**

**CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

Giảng viên hướng dẫn: ThS. Phạm Xuân Tích

Sinh viên thực hiện: Nguyễn Hải Hiếu – Lớp CNTT 3-K64

Đề tài số: 29

**Hà Nội tháng 11 năm 2024**

Phần A: Bài toán quản lý

# Đề Tài: Quản lý sinh viên bằng vector

Ứng dụng lớn vector để quản lý các đối tượng là các sinh viên. Mỗi sinh viên cần quản lý các thông tin sau: Mã sinh viên, Tên sinh viên, Ngày sinh, Điểm sinh viên. Trong lớp sinh viên xây dựng toán tử gán.

Viết chương trình cho phép thực hiện các chức năng sau:

* Thêm danh sách sinh viên từ tệp vừa chọn bằng cách nhập tên tệp.
* Thêm từng sinh viên một cách thủ công.
* Xóa sinh viên theo mã sinh viên.
* Sửa đổi thông tin của một sinh viên bất kì trong danh sách.
* Sắp xếp sinh viên theo họ tên hoặc điểm của sinh viên đó.
* Hiển thị danh sách sinh viên có trong vector.
* Tìm kiếm thông tin sinh viên theo mã sinh viên.

# Phân tích đề bài

Bài toàn yêu cầu sử dụng vector được xây dựng sẵn trong thư viện C++.

Xây dựng các lớp và thuộc tính, phương thức của chúng.

* Lớp Birthday: lớp quản lý ngày tháng năm sinh của một sinh viên.

|  |  |
| --- | --- |
| Birthday | |
| Thuộc tính | Phương thức |
| Int day, month, year: ngày, tháng, năm sinh của một sinh viên. | bool valid (Birthday day): kiểm tra ngày nhập vào có đúng không.  Birthday (): hàm tạo với mọi giá trị là 1. |

* Lớp FullName: quản lý họ và tên của sinh viên.

|  |  |
| --- | --- |
| FullName | |
| Thuộc tính | Phương thức |
| String ho, ten: chia họ và tên của sinh viên ra để dễ quản lý. | String getHo (): lấy họ của sinh viên.  String getTen (): lấy tên của sinh viên. |

* Lớp SinhVien: quản lý thông tin của một sinh viên.

|  |  |
| --- | --- |
| FullName | |
| Thuộc tính | Phương thức |
| FullName name: tên của sinh viên.  Birthday: ngày sinh của sinh viên.  Int maSV: mã phân biệt các sinh viên với nhau.  Float point: điểm gpa của sinh viên. | SinhVien (): hàm tạo sinh viên với các số liệu mặc định.  getMaSV (): lấy giá tri của mã sinh viên.  int static cmpName (SinhVien sv1, SinhVien sv2): điều kiện so sánh theo tên.  int static cmpPoint (SinhVien sv1, SinhVien sv2): điều kiện so sánh theo điểm.  static void printTable (): tiêu đề bảng. |

* Lớp Manager: lớp chứa các chức năng quản lý sinh viên.

|  |  |
| --- | --- |
| FullName | |
| Thuộc tính | Phương thức |
| vector<SinhVien> vectorSV: danh sách sinh viên. | static void printOption (): in ra các lựa chọn cho người dùng chọn.  void addFromFile (string fileName): lấy dữ liệu sinh viên từ file được truyền vào.  void addSinhVien (SinhVien sv): thêm sinh viên vào danh sách thông quá bàn phím.  void deleteSinhVien (int maSV): xóa một sinh viên thông quá mã sinh viên.  void printSinhVien (): in ra danh sách sinh viên.  void sortPoint (): sắp xếp sinh viên trong danh sách theo điểm.  void sortName (): sắp xếp sinh viên trong danh sách theo tên.  void findSinhVien (int maSV): tìm thông tin của một sinh viên thông qua mã sinh viên.  void clearData (): xóa toàn bộ dữ liệu của sinh viên trong danh sách. |

# Cài đặt các lớp và hàm main Bằng C++

## Birthday.cpp

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  #ifndef BIRTHDAY\_H  #define BIRTHDAY\_H  class Birthday  {  private:      int day, month, year;      bool valid(Birthday day)      {          if (day.month < 1 || day.month > 12)              return false;          if (day.day < 1 || day.day > 31)              return false;          if (day.month == 2)          {              if (day.year % 400 == 0 || (day.year % 4 == 0 && day.year % 100 != 0))                  return day.day <= 29;              return day.day <= 28;          }          if (day.month == 4 || day.month == 6 || day.month == 9 || day.month == 11)              return day.day <= 30;          return true;      }  public:      Birthday()      {          day = month = year = 1;      }      friend istream &operator>>(istream &is, Birthday &bd)      {          is >> bd.day >> bd.month >> bd.year;          if (!bd.valid(bd))              throw invalid\_argument("Invalid date");          return is;      }      friend ostream &operator<<(ostream &os, Birthday bd)      {          cout << bd.day << "/" << bd.month << "/" << bd.year;          return os;      }  };  #endif |

## Name.cpp

|  |
| --- |
| #include<bits/stdc++.h>  using namespace std;  #ifndef FullName\_h  #define FullName\_h  class FullName{  private:      string ho, ten;  public:      friend istream &operator>> (istream &is, FullName &fn){          string str;          getline(is, str);;          fn.ho = str.substr(0, str.find\_last\_of(" "));          fn.ten = str.substr(str.find\_last\_of(" ") + 1);          return is;      }      friend ostream &operator<< (ostream &os, FullName fn){          string str = fn.ho + " " + fn.ten;          cout << str;          return os;      }      FullName operator=(FullName fullName){          ho = fullName.ho;          ten = fullName.ten;          return \*this;      }      string getHo(){          return ho;      }      string getTen(){          return ten;      }      bool operator<(FullName fullName){return ten < fullName.ten || (ten == fullName.ten && ho < fullName.ho);}      bool operator>(FullName fullName){return ten > fullName.ten || (ten == fullName.ten && ho > fullName.ho);}      bool operator==(FullName fullName){return ten == fullName.ten && ho == fullName.ho;}      bool operator!=(FullName fullName){return !(\*this == fullName);}      bool operator<=(FullName fullName){return \*this < fullName || \*this == fullName;}      bool operator>=(FullName fullName){return \*this > fullName || \*this == fullName;}  };  #endif |

## SinhVien.cpp

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <string>  #include "Name.cpp"  #include "Birthday.cpp"  using namespace std;  #ifndef SinhVien\_h  #define SinhVien\_h  class SinhVien  {  private:      FullName name;      Birthday birthday;      int maSV;      float point;  public:      SinhVien()      {          maSV = 0;          point = 0;      }      friend istream &operator>>(istream &is, SinhVien &sv)      {          is >> sv.maSV;          is.ignore();          is >> sv.name;          is >> sv.birthday >> sv.point;          return is;      }      friend ostream &operator<<(ostream &os, SinhVien sv)      {          os << setw(30) << sv.maSV             << setw(30) << sv.name             << setw(20) << sv.birthday             << setw(30) << sv.point << endl;          return os;      }      int getMaSV() { return maSV; }      int static cmpName(SinhVien sv1, SinhVien sv2){return sv1.name < sv2.name;}      int static cmpPoint(SinhVien sv1, SinhVien sv2)      {          if (sv1.point == sv2.point)              return sv1.maSV < sv2.maSV;          return sv1.point < sv2.point;      }      static void printTable()      {          for (int i = 0; i < 120; i++)              cout << "-";          cout << endl;          cout << setw(30) << "Ma SV"               << setw(30) << "Ten"               << setw(30) << "Sinh Nhat"               << setw(30) << "Diem" << endl;          for (int i = 0; i < 120; i++)              cout << "-";          cout << endl;      }      SinhVien operator=(SinhVien sv)      {          maSV = sv.maSV;          name = sv.name;          birthday = sv.birthday;          point = sv.point;          return \*this;      }  };  #endif |

## Main.cpp

|  |
| --- |
| #include <bits/stdc++.h>  #include "SinhVien.cpp"  using namespace std;  class Manager  {  private:      vector<SinhVien> vectorSV;  public:      static void printOption()      {          cout << "1. them danh sach sinh vien tu file" << endl               << "2. them sinh vien" << endl               << "3. xoa sinh vien" << endl               << "4. in danh sach sinh vien" << endl               << "5. sap xep sinh vien theo diem" << endl               << "6. sap xep sinh vien theo ten" << endl               << "7. tim sinh vien theo ma sinh vien" << endl               << "8. clear du lieu" << endl               << "9. sua thong tin sinh vien" << endl               << "10. lam sach man hinh" << endl               << "11. thoat" << endl;          cout << "Nhap lua chon: ";      }      void addFromFile(string fileName)      {          ifstream file(fileName);          if (file.is\_open())          {              int n;              file >> n;              while (n--)              {                  SinhVien sv;                  file >> sv;                  vectorSV.push\_back(sv);              }              file.close();              cout << "Da them sinh vien tu file" << endl;          }          else          {              cout << "Khong the mo file" << endl;          }      }      void addSinhVien(SinhVien sv)      {          vectorSV.push\_back(sv);          cout << "Da them sinh vien" << endl;      }      void deleteSinhVien(int maSV)      {          bool isDeleted = false;          for (int i = 0; i < vectorSV.size(); i++)          {              if (vectorSV[i].getMaSV() == maSV)              {                  vectorSV.erase(vectorSV.begin() + i);                  isDeleted = true;                  cout << "Da xoa sinh vien" << endl;                  break;              }          }          if (!isDeleted)              cout << "Khong tim thay sinh vien" << endl;      }      void printSinhVien()      {          cout << "Danh sach sinh vien:" << endl;          SinhVien::printTable();          for (int i = 0; i < vectorSV.size(); i++)          {              cout << vectorSV[i] << endl;          }      }      void sortPoint()      {          sort(vectorSV.begin(), vectorSV.end(), SinhVien::cmpPoint);          cout << "Sinh vien da duoc sap xep theo diem" << endl;      }      void sortName()      {          sort(vectorSV.begin(), vectorSV.end(), SinhVien::cmpName);          cout << "Sinh vien da duoc sap xep theo ten" << endl;      }      void findSinhVien(int maSV)      {          bool isFound = false;          for (int i = 0; i < vectorSV.size(); i++)          {              if (vectorSV[i].getMaSV() == maSV)              {                  SinhVien::printTable();                  cout << vectorSV[i] << endl;                  isFound = true;                  break;              }          }          if (!isFound)              cout << "Khong tim thay sinh vien" << endl;      }      void clearData()      {          vectorSV.clear();          cout << "Da xoa toan bo du lieu sinh vien" << endl;      }      void editSinhVien(int maSV)      {          bool isFound = false;          for (int i = 0; i < vectorSV.size(); i++)          {              if (vectorSV[i].getMaSV() == maSV)              {                  cout << "thong tin can sua gom: maSV, hotenSV, ngaySinhNhat, DiemSv" << endl;                  SinhVien sv;                  cin >> sv;                  isFound = true;                  if (sv.getMaSV() == vectorSV[i].getMaSV()){                      vectorSV[i] = sv;                        cout << "Da sua sinh vien" << endl;                      break;                  }                  else                      cout << "Ma sinh vien khong duoc sua" << endl;                }          }          if (!isFound)              cout << "Khong tim thay sinh vien" << endl;      }  };  int main()  {      int option;      Manager manager;      do      {          Manager::printOption();          cin >> option;          if (option == 1)          {              // them danh sach sinh vien tu file              string fileName;              cout << "Nhap ten file: ";              cin >> fileName;              manager.addFromFile(fileName);          }          else if (option == 2)          {              // them sinh vien              cout << "thong tin can them gom: maSV, hotenSV, ngaySinhNhat, DiemSv" << endl;              SinhVien sv;              cin >> sv;              manager.addSinhVien(sv);          }          else if (option == 3)          {              // xoa sinh vien              int maSV;              cout << "Nhap ma sinh vien can xoa: ";              cin >> maSV;              manager.deleteSinhVien(maSV);          }          else if (option == 4)          {              // in danh sach sinh vien              manager.printSinhVien();          }          else if (option == 5)          {              // sap xep sinh vien theo diem              manager.sortPoint();          }          else if (option == 6)          {              // sap xep sinh vien theo ten              manager.sortName();          }          else if (option == 7)          {              // tim sinh vien theo so bao danh              int maSV;              cout << "Nhap ma sinh vien can tim: ";              cin >> maSV;              manager.findSinhVien(maSV);          }          else if (option == 8)          {              // clear du lieu              manager.clearData();          }          else if (option == 9){              cout << "nhap ma sinh vien muon sua thong tin: ";              int maSV;              cin >> maSV;              manager.editSinhVien(maSV);          }          else if (option == 10)          {              // lam sach man hinh              system("cls");          }          if (option > 0 && option < 11){              cout << "An enter de tiep tuc...";              cin.ignore(numeric\_limits<streamsize>::max(), '\n');              cin.get();          }      } while (option > 0 && option < 11);      system("cls");  } |

# Phân tích thời gian chạy của từng phương thức

## Lớp Birthday

|  |  |  |
| --- | --- | --- |
| Hàm | Độ phức tạp | Phân tích |
| valid (Birthday day) | O (1) | Hàm kiểm tra dữ liệu có thể tạo ra một ngày không. |
| Birthday () | O (1) | Gán giá trị ban đầu là 1 cho cả 3 thuộc tính. |
| &operator >> (Birthday &bd) | O (1) | Nhập 3 số int là 3 thuộc tính của lớp sau đó kiểm tra tính thỏa mãn bằng hàm valid có độ phức tạp O (1). |
| &operator << (Birthday bd) | O (1) | In ra lần lượt 3 thuộc tính của lớp. |

## Lớp FullName

|  |  |  |
| --- | --- | --- |
| Hàm | Độ phức tạp | Phân tích |
| &operator >> (FullName fn) | O (n) | Nhập vào một chuỗi và cắt chuỗi đó bằng 2 hàm có sẵn trong lớp string của thư viện string là substr và find\_last\_of có độ phức tạp đều là O (n) trong trường hợp tệ nhất, với n là độ dài chuỗi. |
| &operator << (FullName fn) | O (1) | In ra lần lượt 2 thuộc tính của lớp. |
| string getHo () | O (1) | Trả về giá trị của họ sinh viên. |
| string getTen () | O (1) | Trả về giá trị của tên sinh viên. |
| Các toán tử so sánh | O (1) | Đều sử dụng so sánh sẵn của 2 string để so sánh 2 thuộc tính ho và ten. |
| Operator = (FullName FullName) | O (1) | Gán cho 2 thuọc tinh của lớp với giá trị của mủa một đối tượng fullname khác. |

## Lớp SinhVien

|  |  |  |
| --- | --- | --- |
| Hàm | Độ phức tạp | Phân tích |
| SinhVien () | O (1) | Hàm tạo gán giá trị ban đầu cho các thuộc tính |
| &operator >> (SinhVien &sv) | O (1) | Nhập giá trị cho các thuộc tính trong lớp tuy có nhập cho các lớp không phải kiểu dữ liệu nguyên thủy nhưng độ phức tạp nhập vào các lớp này vẫn là O (1). |
| &operator << (SinhVien sv) | O (1) | In lần lượt các dữ liệu tuy có cả các lớp nhưng độ phức tạp in ra của các lớp này cũng là O (1), bên cạnh đó có hàm setw cho dữ liệu dễ nhìn những độ phức tạp cũng chỉ là O (1). |
| getMaSV () | O (1) | Trả về mã sinh viên có kiểu dữ liệu int của sinh viên này. |
| cmpName (SinhVien sv1, SinhVien sv2) | O (1) | Hàm này chỉ so sánh 2 tên với nhau với độ phức tạp O (1). |
| cmpPoint (SinhVien sv1, SinhVien sv2) | O (1) | Hàm so sánh điểm của 2 sinh viên nếu bằng nhau thì so sánh mã sinh viên. |
| printTable () | O (1) | In ra tên các cột trong bảng dữ liệu của sinh viên |
| operator= (SinhVien sv) | O (1) | Gán các thuộc tính của đối tượng SinhVien khác cho đối tượng này. |

## Lớp Manager

|  |  |  |
| --- | --- | --- |
| Hàm | Độ phức tạp | Phân tích |
| printOption () | O (1) | In ra các thông tin để người dùng chọn. |
| addFromFile (string fileName) | O (n) | Nhập thông tin cho n sinh viên từ file có dừng toán tử nhập cho lớp SinhVien với độ phức tạp là O (1) và push\_back của vector có độ phức tạp O (1). |
| addSinhVien (SinhVien sv) | O (1) | Thêm thông tin sinh viên được truyền vào với sử dụng push\_back của vector với độ phức tạp O (1). |
| deleteSinhVien (int maSV) | O (n) | Tìm và xóa sinh viên lần lượt từng sinh viên trong trường hợp tệ nhất thì độ phức tạp là O (n). |
| printSinhVien () | O (n) | Duyệt từng phần tử trong vector để in ra danh sách có n sinh viên. sử dụng hàm printTable của lớp SinhVien có độ phức tạp O (1) |
| sortPoint () | O (n \* log(n)) | Sắp xếp các sinh viên tăng dần theo điểm trong vector với hàm sort sẵn có độ phức tạp thấp nhất là O (n \* log(n)). |
| sortName () | O (n \* log(n)) | Sắp xếp các sinh viên theo tên trong vector với hàm sort sẵn có độ phức tạp thấp nhất là O (n \* log(n)). |
| findSinhVien (int maSV) | O (n) | Duyệt từng phần tử trong vector để tìm sinh vien có mã sinh viên bằng maSV. Trong trường hợp tệ nhất không tìm thấy sẽ có độ phức tạp là O (n). |
| void clearData () | O (n) | Sử dụng hàm clear của vector có độ phức tạp O (n) với n là số phần tử trong vector. |
| void editSinhVien (int maSV) | O (n) | Tìm kiếm sinh viên qua mã sinh viên sau đó nếu tìm thấy sẽ thay đổi thông tin sinh viên đó. Trong trường hợp tệ nhất không tìm thấy sẽ có độ phức tạp là O (n). |

# Tài liệu tham khảo

* Độ phức tạp của các hàm, phương thức được sử dụng trong bài: chatGPT, https://www.28tech.com.vn/.

Phần B: Đề tài được phân công

# Đề bài (bài số 29 trong bài tập)

* Xây dựng lớp cây tìm kiếm nhị phân.
* Viết chương trình quản lý lịch công tác trong tháng đơn giản: cho phép nhập vào nội dung công việc cần làm theo ngày, theo giờ. Trong một ngày có thể có nhiều công việc, mỗi công việc có giờ bắt đầu, tên công việc, nội dung công việc, tính chất công việc {rất quan trọng, quan trọng, bình thường, không cần thiết}, …

Chương trình có các chức năng chính như sau:

* Nhập nội dung công việc cần làm theo ngày, theo giờ.
* Xem lịch công tác theo ngày, theo giờ.
* Xem các công việc đã hoàn tất.
* Xem các công việc chưa thực hiện.
* Xem các công việc từ ngày a đến ngày b.
* Xóa hay điều chỉnh lịch công tác. Nếu sau khi điều chỉnh, ngày nào không còn việc phải làm làm sẽ xóa khỏi lịch công tác.

Yêu cầu: Cấu trúc dữ liệu sẽ được sử dụng là cây nhị phân tìm kiếm (BST), trong đó:

* Mỗi nút trên cây BST là một ngày của lịch công tác.
* Trong mỗi nút ngày trên cây lại chứa một danh sách liên kết lưu thông tin các công việc.
* Khi them một công việc vào ngày đã tồn tại trên cây thì công việc này sẽ được đưa vào danh sách liên kết chứa các công việc theo thứ tự tang dần của giờ bắt đầu.

# phân tích bài toán

## Xác định yêu cầu bài toán

* Xây dựng lớp cây nhị phân tìm kiếm.
* Xây dựng lớp danh sách liên kết.
* Mỗi nốt trên cây là ngày và danh sách liên kết chứa các công việc trong ngày nếu danh sách liên kết rỗng sẽ xóa nốt này
* Danh sách liên kết chứa thông tin công việc xếp các công việc theo thời gian bắt đầu tang dần.

## Xác định các lớp, các thuộc tính, các phương thức của lớp

* BST\_Node: cây nhị phân chứa danh sách các công việc sắp xếp các công việc sắp xếp theo ngày từ cũ nhất đến mới nhát.

|  |  |
| --- | --- |
| BST\_Node | |
| Thuộc tính | Phương thức |
| DailySchedule data: dữ liệu mỗi nút của cây nhị phân.  BST\_Node \*left: con trỏ trỏ tới nút con bên trái.  BST\_Node \*right: con trỏ trỏ tới nút con bên phải. | BST\_Node (DailySchedule data): hàm tạo của cây nhị phân.  Void insert (BST\_Node \*&curNode, Task val): thêm công việc vào cây nếu như ngày đó đã tồn tại trong cây thì them công việc vào danh sách cây đó còn không thì tạo một nút mới.  Void \*find (BST\_Node \*curNode, Day): tìm nút có ngày là day nếu không tìm thấy sẽ trả về nullptr.  Void \*find (BST\_Node \*curNode, Task task): tìm nút chứa công việc task: nếu tìm thấy thì trả về con trỏ tới nút đó.  DailySchedule max\_node (BST\_Node \*curr): trả dữ liệu có ngày lớn nhất từ nút được truyền vào.  Void remove\_day (BST\_Node \*&root, Day day): xóa nút có ngày là day.  Void remove (BST\_Node \*&root, Task task): xóa một công việc task nếu xóa xong ngày đó không còn công việc nào thì xóa luôn ngày đó.  Void inorder (BST\_Node \*root): in ra dữ liệu của từng ngày từ cũ nhất đến mới nhất.  Void getTaskByPriority (BST\_Node \*root, int tinhChatCongViec): duyệt và in ra những công việc có tính chất là tinhChatCongViec.  Void geTaskByStatus (BST\_Node \*root, int trangThaiCongViec): duyệt và in ra những công việc có trạng thái là trangThaiCongViec.  Void aPeriod (BST\_Node \*root, Day a, Day b): duyệt công việc trong khoảng thời gian từ ngày a đến ngày b. |

* List: lớp này là danh sách liên kết đôi sắp xếp dữ liệu theo thứ tự tăng dần từ bé đến lớn

|  |  |
| --- | --- |
| List<T> | |
| Thuộc tính | Phương thức |
| Int n: số phần tử có trong danh sách liên kết.  List\_node<T> \*head: nút đầu của danh sách liên kết.  List\_node<T> \*tail: nút cuối của danh sách liên kết. | List (): hàm tạo danh sách liên kết gán head, tail bằng nullptr và n = 0.  Void pop\_front (): xóa phần tử đâu tiên của danh sách liên kết.  Void push\_back (T x): them phần tử x vào cuối dãy.  Void push\_front (T x): them phần tử x vào đầu dãy.  Bool empty (): kiểm tra danh sách có rỗng hay không.  Int size (): trả về số lượng phần tử có trong danh sách.  Void push (T x): them dữ liệu vào danh sách theo điều kiện càng về sau thì dữ liệu có giá trị càng lớn.  Earase (T x): xóa phần từ có giá trị là x.  Iterator begin (): trả vè vị trí của phần tử đầu trong danh sách  Iterator end (): trả về vị trí của phần từ sau phần từ của cùng của danh sách.  Iterator rbegin (): trả về bị trí của phần từ cuối cùng trong danh sách.  Iterator rend (): trả về vị trí trước của phần tử đầu tiên trong danh sách. |

* CongViec: lớp này chứa thông tin của một công việc không bao gồm ngày làm công việc.

|  |  |
| --- | --- |
| CongViec | |
| Thuộc tính | Phương thức |
| Int gioBatDau: giờ bắt đầu làm công việc.  Sring tenCongViec: tên công việc.  String noiDungcongViec: nội dung công việc.  Int tinhChatCongViec: tính chất công việc.  Int trangThaiCongViec: trạng thái công việc. | String tinhChat (int tinhChatCongViec): trả về tính chất công việc nhưng ở dạng chuỗi.  String trangThaicongviec (int trangThaicongViec): trả về trạng thái công việc nhưng ở dạng chuỗi.  CongViec (): hàm tạo có thể truyền đối hoặc không truyền đối.  Các hàm get và set để chỉnh sửa hoặc lấy dữ liệu trong một phạm vi bị giới hạn. |

* Day: là lớp đại diện cho ngày tháng năm.

|  |  |
| --- | --- |
| Day | |
| Thuộc tính | Phương thức |
| Int d: ngày.  Int m: tháng.  Int y: năm. | Bool valid (Day day): kiểm tra xem ngày mà dữ liệu nhận được có đúng không. |

* Task: là lớp chứa thông tin công việc bao gồm cả ngày làm công việc

|  |  |
| --- | --- |
| Task | |
| Thuộc tính | Phương thức |
| Day day: ngày làm công việc.  CongViec: thông của công việc | Task (): hàm tạo rỗng.  Void setDay: chỉnh sửa ngày.  Void getDay: lấy thông tin về ngày. |

* TaskList: lớp này là danh sách liên kết làm việc với lớp công việc.

|  |  |
| --- | --- |
| TaskList | |
| Thuộc tính | Phương thức |
| List<CongViec> list: danh sách liên làm việc với lớp CongViec. | Int getSize (): lấy kích cỡ của danh sách liên kết.  Void addTask (): thêm công việc vào danh sách.  List<CongViec> getList (): lấy danh sách.  Void getTaskBypriority (int tinhChatCongViec): in ra các công việc có tính chất công việc bằng tinhChatCongViec.  Void getTaskByStatus (int trangThaiCongViec): in ra các công việc có trạng thái công việc bằng trangThaiCongViec.  Void print (): in ra danh sách các công việc.  Bool containTask (CongViec cv): kiểm tra một công việc có ở trong danh sách hay không.  Void setDone (CongViec cv): dánh dấu một công việc là hoàn thành. |

* DailySchedule: đây là lớp chứa ngày nào và có những công việc gì trong ngày này.

|  |  |
| --- | --- |
| DailySchedule | |
| Thuộc tính | Phương thức |
| Day day: ngày này có những công việc nào.  TaskList taskList: danh sách công việc trong ngày day. | DailySchedule (), DailySchedule (Day day), DailySchedule (Task task): hàm tạo  Void addTask (CongViec cv): them công việc vào danh sách.  Void removeTask (CongViec cv): loại bỏ cv trong danh danh sách.  Void getTaskPriority (int tinhChatCongViec): in ra danh sách công việc có tính chất công việc là tinhChatCongViec.  Void getTaskByStatus (int trangThaiCongViec): in ra danh sách công việc có trạng thái công việc là trangThaiCongViec.  List<CongViec> getList (): phương thức trả về danh sách công việc.  Day getDay (): lấy ngày của danh sách công việc.  Void print ():in ra danh sách công việc có trong ngày.  Void setDone (Task task): đánh dầu hoàn thành cho một công việc.  getListSize (): lấy số lượng công công việc có trong ngày.  Bool containTask (CongViec cv): kiểm tra xem trong danh sách công việc có công việc không. |

* DailyScheduleManager: lớp quản lý và xử lý các yêu cầu của người dung

|  |  |
| --- | --- |
| DailyScheduleManager | |
| Thuộc tính | Phương thức |
| BST\_Node \*scheduleTree: nút gốc của cây tìm kiếm nhị phân. | DailyScheduleManager (): hàm tạo.  BST\_Node \*getSchedule Tree (): trả về node gốc của cây.  Void addTask (Task dl): them một công việc vào cây.  Void removeTask (Task dl): xóa một công việc ra khỏi cây.  Void removeTask (Day day): xóa một ngày công việc khỏi cây.  Void viewTask (Day day): xem công việc trong ngày bất kì.  Void viewAllTask (): xem toàn bộ công việc hiện có.  Void viewTaskDone (BST\_Node \*root): xem công việc đã hoàn thành.  Void viewTaskNotDone (BST\_Node \*root): xem công việc chưa hoàn thành.  View TaskByDay (BST\_Node \*root, Day a, Day b): xem công việc trong khoảng thời gian từ ngày a đến ngày b.  Void changeSchedule (Task task, Day newDay): chuyển một công việc từ ngày a đến ngày b.  Void setDone (Task task): dánh dầu một công việc đã hoàn thành. |

# Cài đặt các lớp và hàm main bằng C++

## BSTree.cpp

|  |
| --- |
| #include <iostream>  #include "DailySchedule.cpp"  #include "Task.cpp"  using namespace std;  #ifndef BST\_H  #define BST\_H  class BST\_Node  {  private:  public:  DailySchedule data;  BST\_Node \*left, \*right;  BST\_Node(DailySchedule data)  {          this->data = data;  this->left = this->right = nullptr;      }  };  // thêm hàm vào bằng đệ quy  void insert(BST\_Node \*&curNode, Task val)  {      if (curNode == nullptr)      {          curNode = new BST\_Node(val);          return;      }      if (val.getDay() == curNode->data.getDay())      {          curNode->data.addTask(val.getCongViec());      }      else if (val.getDay() < curNode->data.getDay())          insert(curNode->left, val);      else          insert(curNode->right, val);  }  // tìm phần tử trong cây  BST\_Node \*find(BST\_Node \*curNode, Day day)  {      if (curNode == nullptr || curNode->data.getDay() == day)          return curNode;      else          return find((day < curNode->data.getDay()) ? curNode->left : curNode->right, day);  }  BST\_Node \*find(BST\_Node \*curNode, Task task)  {      if (curNode == nullptr || curNode->data.getDay() == task.getDay()){          if (curNode->data.containTask(task.getCongViec()))              return curNode;          else              return nullptr;      }      else          return find((task.getDay() < curNode->data.getDay()) ? curNode->left : curNode->right, task);  }  // tìm node có giá trị lớn nhất  DailySchedule max\_node(BST\_Node \*curr)  {      return curr->right ? max\_node(curr->right) : curr->data;  }  void remove\_day (BST\_Node \*&root, Day day) {      if (root == nullptr)          return;      if (root->data.getDay() == day){          if(!root->left && !root->right){              root = nullptr;              return;          }          if (!root->left){              BST\_Node \*p = root;              root->right;              p = nullptr;              // delete p;          }          else if (!root->right){              BST\_Node \*p = root;              root = root->left;              p = nullptr;              // delete p;          }          else {              root->data = max\_node(root->left);              remove\_day(root->left, root->data.getDay());          }      }      else          remove\_day(day < root->data.getDay()?root->left:root->right, day);  }  // xóa node  void remove(BST\_Node \*&root, Task val)  {      BST\_Node \*node = find(root, val.getDay());      if (node == nullptr)          return;      node->data.removeTask(val.getCongViec());      if (node->data.getList().size()==0)          remove\_day(root, val.getDay());  }  // duyệt cây  void inorder(BST\_Node \*root)  {      if (!root)          return;      inorder(root->left);      printLine();      cout << root->data.getDay() << ":" << endl;      root->data.print();      inorder(root->right);  }  void getTaskByPriority(BST\_Node \*root, int tinhChatCongViec){      if (!root)          return;      getTaskByPriority(root->left, tinhChatCongViec);      root->data.getTaskByPriority(tinhChatCongViec);      getTaskByPriority(root->right, tinhChatCongViec);  }  void getTaskByStatus(BST\_Node \*root, int trangThaiCongViec){      if (!root)          return;      getTaskByStatus(root->left, trangThaiCongViec);      root->data.getTaskByStatus(trangThaiCongViec);      getTaskByStatus(root->right, trangThaiCongViec);  }  void aPeriod(BST\_Node \*root, Day a, Day b)  {      if (!root)          return;      if (root->data.getDay() >= a)          aPeriod(root->left, a, b);      if (root->data.getDay() >= a && root->data.getDay() <= b)      {          printLine();          cout << root->data.getDay() << ":" << endl;          root->data.print();      }      if (root->data.getDay() <= b)          aPeriod(root->right, a, b);  }  #endif |

## List.cpp

|  |
| --- |
| #include <iostream>  using namespace std;  #ifndef list\_h  #define list\_h  // Node class  template <class T>  class list\_node  {  public:      T data;      list\_node \*prev;      list\_node \*next;      list\_node(T data, list\_node \*prev = nullptr, list\_node \*next = nullptr)      {          this->data = data;          this->prev = prev;          this->next = next;      }  };  // Iterator class  template <class T>  class list\_iterator  {  public:      list\_node<T> \*cur;      list\_iterator(list\_node<T> \*cur) { this->cur = cur; }      T &operator\*() { return cur->data; }      bool operator!=(list\_iterator it) { return cur != it.cur; }      list\_iterator<T> operator++(int)      {          cur = cur->next;          return cur;      }      list\_iterator<T> operator++()      {          list\_iterator<T> it = cur;          cur = cur->next;          return it;      }      list\_iterator<T> operator--(int)      {          cur = cur->prev;          return cur;      }      list\_iterator<T> operator--()      {          list\_iterator<T> it = cur;          cur = cur->prev;          return it;      }  };  // Linked list class  template <class T>  class List  {  private:      list\_node<T> \*head;      list\_node<T> \*tail;      int n = 0;      void push\_back(T x)      {          if (n == 0) head = tail = new list\_node<T>(x);          else tail = tail->next = new list\_node<T>(x, tail);          n++;      }      void push\_front(T x)      {          if (n == 0) head = tail = new list\_node<T>(x);          else head = head->prev = new list\_node<T>(x, nullptr, head);          n++;      }  public:      void pop\_front()      {          if (!head)              return;          n--;          if (head == tail){              head = tail = nullptr;              return;          }          list\_node<T> \*p = head;          head->next->prev = nullptr;          head = head->next;          delete p;      }      List()      {          head = tail = nullptr;          n = 0;      }      ~List(){while (n) pop\_front();}      bool empty() { return n == 0; }      int size() { return n; }      T front() { return head->data; }      T back() { return tail->data; }      // void clear (){while (n) pop\_front();}      void push(T x)      {          if (n == 0) push\_back(x);          else if (x <= head->data) push\_front(x);          else if (x >= tail->data) push\_back(x);          else          {              list\_node<T> \*cur = head->next;              list\_node<T> \*prevNode = head;              while (cur->next != nullptr && x > cur->data)              {                  prevNode = cur;                  cur = cur->next;              }              list\_node<T> \*newNode = new list\_node<T>(x, cur->prev, cur);              // cout << cur->prev << ' ' << cur->next << '\n';              prevNode->next = newNode;              cur->prev = newNode;              n++;          }      }      void erase(T x)      {          if (n == 0)              return;          if (n == 1 && head->data == x)              return pop\_front();          list\_node<T> \*p = head;          while (p->next != nullptr && p->data != x)              p = p->next;          if (p == nullptr)              return;          if (p == head)          {              head = head->next;              head->prev = nullptr;          }          else if (p == tail)          {              tail = tail->prev;              tail->next = nullptr;          }          else          {              p->prev->next = p->next;              p->next->prev = p->prev;          }          delete p;          n--;      }      typedef list\_iterator<T> iterator;      iterator begin() { return head; }      iterator end() { return tail->next; }      iterator rbegin() { return tail; }      iterator rend() { return head->prev; }      void erase(iterator it)      {          if (it.cur == head)          {              head = head->next;              head->prev = nullptr;          }          else if (it.cur == tail)          {              tail = tail->prev;              tail->next = nullptr;          }          else          {              it.cur->prev->next = it.cur->next;              it.cur->next->prev = it.cur->prev;          }          delete it.cur;          n--;      }  };  #endif |

## Day.cpp

|  |
| --- |
| #include <iostream>  #include <exception>  using namespace std;  #ifndef DAY\_H  #define DAY\_H  // đây là ngày tháng năm phục vụ cho việc thêm vào cây nhị phân tìm kiếm  class Day  {  private:      int d, m, y; // ngày tháng năm      bool valid(Day day)      {          if (day.m < 1 || day.m > 12)              return false;          if (day.d < 1 || day.d > 31)              return false;          if (day.m == 2)          {              if (day.y % 400 == 0 || (day.y % 4 == 0 && day.y % 100 != 0))                  return day.d <= 29;              return day.d <= 28;          }          if (day.m == 4 || day.m == 6 || day.m == 9 || day.m == 11)              return day.d <= 30;          return true;      }  public:      Day(int d = 1, int m = 1, int y = 1)      {          this->d = d;          this->m = m;          this->y = y;          if (!valid(\*this))              throw invalid\_argument("Invalid date");      }      // các toán tử nhập xuất      friend istream &operator>>(istream &is, Day &day)      {          is >> day.d >> day.m >> day.y;          if (!day.valid(day))              throw invalid\_argument("Invalid date");          return is;      }      friend ostream &operator<<(ostream &os, Day day)      {          os << day.d << '/' << day.m << '/' << day.y;          return os;      }      // các toán tử so sánh      Day operator=(Day day)      {          d = day.d;          m = day.m;          y = day.y;          return \*this;      }      bool operator==(Day day) { return d == day.d && m == day.m && y == day.y; }      bool operator!=(Day day) { return !(\*this == day); }      bool operator<(Day day) { return y < day.y || (y == day.y && (m < day.m || (m == day.m && d < day.d))); }      bool operator>(Day day) { return !(\*this < day || \*this == day); }      bool operator<=(Day day) { return \*this < day || \*this == day; }      bool operator>=(Day day) { return !(\*this < day); }  };  #endif |

## CongViec.cpp

|  |
| --- |
| #include <iostream>  #include <exception>  #include <stdexcept>  #include <string>  #include <iomanip>  // #include "Time.cpp"  using namespace std;  #ifndef CONGVIEC\_H  #define CONGVIEC\_H  //đây là công việc sẽ đc chứa trong TaskList và DailySchedule  class CongViec  {  private:      int gioBatDau;      string tenCongViec;      string noiDungCongviec;      int tinhChatCongViec;//rat quan trong, quan trong, binh thuong, khong can thiet      int trangThaiCongviec;      string tinhChat(int tinhChatCongViec){          switch (tinhChatCongViec)          {              case 1: return "Rat Quan Trong";              case 2: return "Quan Trong";              case 3: return "Binh Thuong";              case 4: return "Khong Can Thiet";              default: return "Khong Xac Dinh";          }      }      string trangThai(int trangThaiCongViec){          switch (trangThaiCongViec)          {              case 0: return "Chua Hoan Thanh";              case 1: return "Hoan Thanh";              default: return "Khong Xac Dinh";          }      }  public:      CongViec(int gioBatDau, string tenCongViec, string noiDungCongviec)      {          this->gioBatDau = gioBatDau;          this->tenCongViec = tenCongViec;          this->noiDungCongviec = noiDungCongviec;          this->tinhChatCongViec = 0;          this->trangThaiCongviec = 0;      }      CongViec()      {          this->gioBatDau = 0;          this->tenCongViec = "";          this->noiDungCongviec = "";          this->tinhChatCongViec = 0;          this->trangThaiCongviec = 0;      }      //toan tu nhap xuat      friend istream &operator>>(istream &is, CongViec &cv)      {          is >> cv.gioBatDau;          is.ignore();          getline(is, cv.tenCongViec);          getline(is, cv.noiDungCongviec);          is >> cv.tinhChatCongViec;          return is;      }      friend ostream &operator<<(ostream &os, CongViec cv)      {          os << setw(15) << cv.gioBatDau << '|'             << setw(20) << cv.tenCongViec << '|'             << setw(20) << cv.noiDungCongviec << '|'             << setw(20) << cv.tinhChat(cv.tinhChatCongViec) << '|'             << setw(20) << cv.trangThai(cv.trangThaiCongviec) << endl;          return os;      }      //getter and setter      int getTime(){return gioBatDau;}      int getTrangThaiCongViec(){return trangThaiCongviec;}      int getTinhChatCongViec(){return tinhChatCongViec;}      void setDone(){trangThaiCongviec = 1;}      void setRatQuanTrong(){tinhChatCongViec = 1;}      void setQuanTrong(){tinhChatCongViec = 2;}      void setBinhThuong(){tinhChatCongViec = 3;}      void setKhongCanThiet(){tinhChatCongViec = 4;}      // ghi de toan tu gan      CongViec operator=(const CongViec cv)      {          this->gioBatDau = cv.gioBatDau;          this->tenCongViec = cv.tenCongViec;          this->noiDungCongviec = cv.noiDungCongviec;          this->tinhChatCongViec = cv.tinhChatCongViec;          this->trangThaiCongviec = cv.trangThaiCongviec;          return \*this;      }      //ghi de toan tu so sanh      bool operator<(const CongViec cv){          if (this->gioBatDau == cv.gioBatDau)              return tenCongViec < tenCongViec;          return this->gioBatDau < cv.gioBatDau;      }      bool operator>(const CongViec cv){          if (this->gioBatDau == cv.gioBatDau)              return tenCongViec > tenCongViec;          return this->gioBatDau > cv.gioBatDau;      }      bool operator<=(const CongViec cv){          if (this->gioBatDau == cv.gioBatDau)              return tenCongViec <= tenCongViec;          return this->gioBatDau <= cv.gioBatDau;      }      bool operator>=(const CongViec cv){          if (this->gioBatDau == cv.gioBatDau)              return tenCongViec >= tenCongViec;          return this->gioBatDau >= cv.gioBatDau;      }      bool operator==(const CongViec cv){          if (this->gioBatDau == cv.gioBatDau)              return tenCongViec == tenCongViec;          return this->gioBatDau == cv.gioBatDau;      }      bool operator!=(const CongViec cv){          if (this->gioBatDau == cv.gioBatDau)              return tenCongViec != tenCongViec;          return this->gioBatDau != cv.gioBatDau;      }  };  void printLine(){      for (int i = 0; i <= 100; i++)      {          cout << '-';      }      cout << endl;  }  void printTile(){      printLine();      cout << setw(15) << "Gio Bat Dau" << '|'           << setw(20) << "Ten Cong Viec" << '|'           << setw(20) << "Noi Dung Cong Viec" << '|'           << setw(20) << "Tinh Chat Cong Viec" << '|'           << setw(20) << "Trang Thai Cong Viec" << endl;      printLine();  }  #endif |

## TaskList.cpp

|  |
| --- |
| #include<iostream>  #include"List.cpp"  #include"CongViec.cpp"  using namespace std;  #ifndef TASKLIST\_H  #define TASKLIST\_H  // đây là danh sách liên kiết chứa các công việc  class TaskList {  private:      List<CongViec> list;  public:      TaskList() {}      int getSize(){return list.size();}      void addTask(CongViec cv) {list.push(cv);}      void removeTask(CongViec cv) {list.erase(cv);}      List<CongViec> getList() {return list;}      void getTaskByPriority(int tinhChatCongViec) {          for(List<CongViec>::iterator it = list.begin(); it != list.end(); ++it){              if ((\*it).getTinhChatCongViec() == tinhChatCongViec){                  cout << (\*it);              }          }      }      void getTaskByStatus(int trangThaiCongviec) {          for(List<CongViec>::iterator it = list.begin(); it != list.end(); ++it){              if ((\*it).getTrangThaiCongViec() == trangThaiCongviec){                  cout << (\*it);              }          }      }      void print(){          if(list.size() > 0)              for(List<CongViec>::iterator it = list.begin(); it != list.end(); ++it)                  cout << (\*it);      }        bool containTask(CongViec cv){          for(List<CongViec>::iterator it = list.begin(); it != list.end(); ++it){              if ((\*it) == cv)                  return true;          }          return false;      }      void setDone (CongViec cv) {          for(List<CongViec>::iterator it = list.begin(); it != list.end(); ++it){              if ((\*it) == cv){                  (\*it).setDone();                  cout << "da danh dau hoan thanh.\n";                  return;              }          }          cout << "khong tim thay cong viec ban muon danh dau.\n";      }  };  #endif |

## Task.cpp

|  |
| --- |
| #include<iostream>  #include"CongViec.cpp"  #include"Day.cpp"  using namespace std;  #ifndef Task\_H  #define Task\_H  class Task {  private:      Day day;      CongViec cv;  public:      Task(){}      friend istream &operator>>(istream &is, Task &t){          is >> t.day >> t.cv;          return is;      }      friend ostream &operator<<(ostream &os, Task dl){          os << dl.day << endl << dl.cv;          return os;      }      void setDay(Day day){this->day = day;}      Day getDay(){return day;}      CongViec getCongViec () {return cv;}      // các toán tử so sánh      bool operator==(Task ds) {return this->day == ds.day;}      bool operator!=(Task ds) {return this->day != ds.day;}      bool operator<(Task ds) {return this->day < ds.day;}      bool operator>(Task ds) {return this->day > ds.day;}      bool operator<=(Task ds) {return this->day <= ds.day;}      bool operator>=(Task ds) {return this->day >= ds.day;}  };  #endif |

## DailySchedule

|  |
| --- |
| #include<iostream>  #include"TaskList.cpp"  #include"Day.cpp"  #include"Task.cpp"  using namespace std;  #ifndef DAILYSCHEDULE\_H  #define DAILYSCHEDULE\_H  // đây là lịch hàng ngày chứa ngày nào có những công việc gì trong ngày đó  class DailySchedule {  private:      Day day;      TaskList taskList;  public:      DailySchedule(Day day) {this->day = day;}      DailySchedule(){}      DailySchedule(Task task){          day = task.getDay();          taskList.addTask(task.getCongViec());      }      // các method thêm xóa công việc      void addTask(CongViec cv) {taskList.addTask(cv);}      void removeTask(CongViec cv) {taskList.removeTask(cv);}      // các method lấy ra danh sách công việc theo mức độ quan trọng hoặc trạng thái      void getTaskByPriority(int tinhChatCongViec) {          printLine();          cout << day << ":" << endl;          taskList.getTaskByPriority(tinhChatCongViec);      }      void getTaskByStatus(int trangThaiCongviec) {          printLine();          cout << day << ":" << endl;          taskList.getTaskByStatus(trangThaiCongviec);      }      List<CongViec> getList() {return taskList.getList();}      Day getDay() {return day;}      void print(){taskList.print();}      void setDone(Task task){          taskList.setDone(task.getCongViec());      }      int getListSize(){taskList.getSize();}      bool containTask(CongViec cv){return taskList.containTask(cv);}      // các toán tử so sánh      bool operator==(DailySchedule ds) {return this->day == ds.day;}      bool operator!=(DailySchedule ds) {return this->day != ds.day;}      bool operator<(DailySchedule ds) {return this->day < ds.day;}      bool operator>(DailySchedule ds) {return this->day > ds.day;}      bool operator<=(DailySchedule ds) {return this->day <= ds.day;}      bool operator>=(DailySchedule ds) {return this->day >= ds.day;}      // ghi đè toán tử IO      friend ostream &operator<<(ostream &os, DailySchedule dl){          os << dl.day << endl;          dl.taskList.print();          return os;      }      friend istream &operator>>(istream &is, DailySchedule &dl){          CongViec cv;          is >> dl.day >> cv;          dl.addTask(cv);          return is;      }      void Output () {          cout << this->day << endl;          this->taskList.print();      }      static DailySchedule Input () {          Day day;          cin >> day;          CongViec cv;          cin >> cv;          DailySchedule dl(day);          dl.addTask(cv);          return dl;      }  };  #endif |

## 8.Main.cpp

|  |
| --- |
| #include <iostream>  #include <exception>  #include <limits>  #include"BSTree.cpp"  #include"Day.cpp"  #include"DailySchedule.cpp"  #include"Task.cpp"  using namespace std;  //  Lớp quản lý và xử lý các yêu cầu của người dùng.  class DailyScheduleManager  {  private:      BST\_Node \*scheduleTree;  public:      DailyScheduleManager() {scheduleTree = nullptr;}      ~DailyScheduleManager() {delete scheduleTree;}      BST\_Node \*getScheduleTree() {return scheduleTree;}      // Thêm một công việc vào lịch hàng ngày      void addTask(Task dl) {          insert(scheduleTree, dl);      }      // Xóa một công việc khỏi lịch hàng ngày      void removeTask(Task dl) {          remove(scheduleTree, dl);      }        // Xóa một ngày khỏi lịch      void removeTask(Day day) {          remove\_day(scheduleTree, day);      }      // xem lịch công tác trong một ngày      void viewTask(Day day) {          printTile();          BST\_Node \*node = find(scheduleTree, day);          if(node)              node->data.print();      }      void viewAllTask() {          printTile();          inorder(scheduleTree);      }      //Xem các công việc theo tính chất: rất quan trọng, quan trọng, …      void viewTaskByPriority(int tinhChatCongViec) {          printTile();          getTaskByPriority(scheduleTree, tinhChatCongViec);      }      //Xem các công việc đã hoàn tất.      void viewTaskDone(BST\_Node \*root) {          printTile();          getTaskByStatus(scheduleTree, 1);      }      //Xem các công việc chưa thực hiện.      void viewTaskNotDone(BST\_Node \*root) {          printTile();          getTaskByStatus(scheduleTree, 0);      }      //Xem các công việc từ ngày a đến ngày b.      void viewTaskByDay(BST\_Node \*root,Day a,Day b) {          printTile();          aPeriod(scheduleTree, a, b);      }      //Xóa hay điều chỉnh lịch công tác. Nếu sau khi điều chỉnh, ngày nào không còn việc phải làm sẽ xóa khỏi lịch công tác.      void changeSchedule(Task task, Day newDay) {          BST\_Node \*node = find(scheduleTree, task);          if (node)          {              removeTask(task);              // remove(scheduleTree, task);              task.setDay(newDay);              insert(scheduleTree, task);          }      }      void setDone(Task task){          BST\_Node \*node = find(scheduleTree, task);          if (node)          {              node->data.setDone(task);          }        }  };  int main()  {      DailyScheduleManager manager;      int option = 0;      do      {          cout << "0. thoat\n"              << "1. nhap noi dung cong viec can lam theo ngay, theo gio\n"              << "2. xem lich cong tac hien co\n"              << "3. xem lich cong tac theo ngay yeu cau\n"              << "4. xem cac cong viec theo tinh chat: rat quan trong, quan trong, ...\n"              << "5. xem cac cong viec da hoan tat\n"              << "6. xem cac cong viec chua thuc hien\n"              << "7. xem cac cong viec tu ngay a den ngay b\n"              << "8. xoa mot ngay cong tac\n"              << "9. xoa mot cong viec\n"              << "10. chuyen mot cong viec sang ngay khac\n"              << "11. danh dau mot cong viec da hoan thanh\n"              << "12. lam sach\n"              << "Chon chuc nang: ";          cin >> option;          if (option == 1)          {              cout << "Nhap noi dung cong viec can lam theo ngay, theo gio:\n";              Task task;              cin >> task;              manager.addTask(task);          }          else if (option == 2){              manager.viewAllTask();          }          else if (option == 3)          {              cout << "Nhap ngay yeu cau: ";              Day day;              cin >> day;              manager.viewTask(day);          }          else if (option == 4)          {              // Rất quan trọng, quan trọng, bình thường, không cần thiết              cout << "Nhap tinh chat cong viec:\n"              << "1. Rat quan trong\n"              << "2. Quan trong\n"              << "3. Binh thuong\n"              << "4. Khong can thiet\n";              int tinhChat;              cin >> tinhChat;              manager.viewTaskByPriority(tinhChat);          }          else if (option == 5){manager.viewTaskDone(manager.getScheduleTree());}          else if (option == 6){manager.viewTaskNotDone(manager.getScheduleTree());}          else if (option == 7)          {              Day a, b;              cout << "Tu ngay: ";              cin >> a ;              cout << "Den ngay: ";              cin >> b;              manager.viewTaskByDay(manager.getScheduleTree(), a, b);          }          else if (option == 8)          {              cout << "Nhap ngay can xoa: ";              Day day;              cin >> day;              manager.removeTask(day);          }          else if (option == 9)          {              cout << "Nhap cong viec can xoa: ";              Task task;              cin >> task;              manager.removeTask(task);          }          else if (option == 10)          {              cout << "nhap cong viec can chuyen:\n";              Task task;              cin >> task;              cout << "nhap ngay chuyen den:\n";              Day day;              cin >> day;              manager.changeSchedule(task, day);          }          else if (option == 11)          {              cout << "nhap cong viec muon danh dau hoan thanh: ";              Task task;              cin  >> task;              manager.setDone(task);          }            else if (option == 12)          {              // lam sach man hinh              system("cls");          }            if (option > 0 && option <= 12){              cout << "An enter de tiep tuc...";              cin.ignore(numeric\_limits<streamsize>::max(), '\n');              cin.get();          }      } while (option > 0 && option <= 12);      system("cls");      return 0;  } |

# Phân tích thời gian chạy của từng phương thức có trong lớp.

## Lớp List

|  |  |  |
| --- | --- | --- |
| Hàm | Độ phức tạp | Phân tích |
| List () | O (1) | Chỉ gán con trỏ head và tail thành nullptr và gàn n bằng 0. |
| ~List () | O (n) | Xóa lần lượt từng phần tử ra khỏi dãy. |
| Push\_back (T x) | O (1) | Chỉ gắn tạo vào gắn con trỏ tail vào địa chỉ của một list\_node mới. |
| Push\_front (T x) | O (1) | Chỉ gắn tạo vào gắn con trỏ head vào địa chỉ của một list\_node mới. |
| Pop\_front () | O (1) | Nếu danh sách rỗng thì kết thúc, nếu còn phần tử thì gắn head vào địa chỉ tiếp theo và xóa con trỏ cũ. |
| Empty () | O (1) | Trả về giá trị của n. |
| Size () | O (1) | Trả về n có bằng 0 hay không. |
| Front () | O (1) | Lấy dữ liệu của con trỏ head. |
| Back () | O (1) | Lấy dữ liệu của con trỏ tail. |
| Push () | O (n) | Trong trường hợp tệ nhất thì phải duyệt hết danh sách mới them được dữ liệu vào danh sách vì phải duyệt đến khi nào giá trị truyền vào nhỏ hơn nút đang duyệt. |
| Earse (T x) | O (n) | Trong trường hợp xấu nhất ta sẽ phải duyệt hết danh sách mới tìm ra được dữ liệu mà mình cần xóa vì duyêt từng phần tử để tìm ra dữ liệu cần xóa. |
| Begin () | O (1) | Trả về con trỏ của head. |
| End () | O (1) | Trả về con trỏ sau tail. |
| Rbegin () | O (1) | Trả về con trỏ của tail. |
| Rend () | O (1) | Trả về con trỏ trước head. |
| End () | O (1) | Trả về con trỏ của tail. |

## Lớp Day

|  |  |  |
| --- | --- | --- |
| Hàm | Độ phức tạp | Phân tích |
| Valid (Day day) | O (1) | Chỉ kiểm tra dữ liệu truyền vào có thỏa mãn hay không. |
| Day (int d, int m, int y) | O (1) | Truyền dữ liệu vào để tạo ngày, tháng, năm rồi kiểm tra bằng hàm valid. |
| Operator >> (Day &day) | O (1) | Nhập dữ liệu gắn giá trị cho các thuộc tính và kiểm tra bằng hàm valid. |
| Operator << (Day day) | O (1) | In các thuộc tính trong lớp ra màn hình. |
| Operator = (Day day) | O (1) | Gán giá trị mới cho các thuộc tính. |
| Các toán tử so sánh | O (1) | So sánh giá trị hiện tại và giá trị day được truyền vào nên không tốn nhiều thời gian. |

## Lớp CongViec

|  |  |  |
| --- | --- | --- |
| Hàm | Độ phức tạp | Phân tích |
| CongViec (int gioBatDau, string tenCongViec, string noiDungCongviec) | O (1) | Hàm tạo gắn giá trị ban đầu cho các thuộc tính. |
| CongViec () | O (1) | Hàm tạo gắn giá trị ban đầu cho các thuộc tính. |
| Operator >> (CongViec &cv) | O (1) | Thay đổi giá trị các thuộc tính trong lớp. |
| Operator << (CongViec cv) | O (1) | In ra các dữ liệu theo fomat. |
| getTime () | O (1) | Trả về giá trị của gioBatdau. |
| getTrangThaiCongViec () | O (1) | Trả về giá trị của trangThaiCongviec. |
| getTinhChatCongViec () | O (1) | Trả về giá trị của tinhChatCongViec. |
| setDone () | O (1) | Đánh dấu là công việc đã hoàn thành. |
| setRatQuanTrong () | O (1) | Đánh dầu tính chất công việc là rất quan trọng. |
| setQuanTrong () | O (1) | Đánh dầu tính chất công việc là quan trọng. |
| setBinhThuong () | O (1) | Đánh dầu tính chất công việc là bình thường. |
| setKhongCanThiet () | O (1) | Đánh dầu tính chất công việc là không cần thiết. |
| Operator = (CongViec cv) | O (1) | Gián các thuộc tính của lớp thành giá trị mới. |
| Các toàn tử so sánh với CongViec | O (1) | So sánh giờ bắt đầu nếu mà giờ bắt đầu giống nhau thì bắt đầu so sánh tên công viêc. |

## Lớp TaskList

|  |  |  |
| --- | --- | --- |
| Hàm | Độ phức tạp | Phân tích |
| TaskList () | O (1) | Hàm rỗng. |
| getSize () | O (1) | Gọi hàm size của lớp List có độ phức tạp O (1). |
| addTask (CongViec cv) | O (n) | Sử dụng hàm push của lớp List có độ phức tạp O (n) trong trường hợp tệ nhất. |
| removeTask (CongViec cv) | O (n) | Sử dụng hàm erase của lớp List có độ phức tạp O (n) trong trường hợp tệ nhất. |
| List<CongViec> getList() | O (1) | Hàm trả về dữ liệu của list. |
| getTaskByPriority (int tinhChatCongViec) | O (n) | Duyệt toàn bộ danh sách để in ra các công việc cần tìm. |
| getTaskByStatus (int trangThaiCongviec) | O (n) | Duyệt toàn bộ danh sách để in ra các công việc cần tìm. |
| Print () | O (n) | In ra toàn bộ công việc có trong danh sách. |
| containTask (CongViec cv) | O (n) | Trong trường hợp tệ nhất thì phải duyệt hết danh sách để tìm. |
| void setDone (CongViec cv) | O (n) | Trong trường hợp không tìm thấy thì sẽ phải duyệt hết danh sách. |

## Lớp Task

|  |  |  |
| --- | --- | --- |
| Hàm | Độ phức tạp | Phân tích |
| Task () | O (1) | Hàm tạo rỗng. |
| Operator >> (Task &t) | O (1) | Sử dụng toán tử >> của cả lớp Day và lớp CongViec, cả hai đều có độ phức tạp lần lượt là O (1). |
| Operator << (Task t) | O (1) | Sử dụng toán tử << của cả lớp Day và lớp CongViec, cả hai đều có độ phức tạp lần lượt là O (1). |
| setDay (Day day) | O (1) | Sử dụng toán tử gán của lớp Day có độ phức tạp là O (1). |
| getDay () | O (1) | Lấy giá trị thuộc tính day. |
| getCongViec () | O (1) | Lấy giá trị thuộc tính cv. |
| Các toàn tử so sánh | O (1) | Sử dụng toán tử so sánh của lớp Day tất cả đều có độ phức tạp là O (1). |

## Lớp DailySchedule

|  |  |  |
| --- | --- | --- |
| Hàm | Độ phức tạp | Phân tích |
| DailySchedule (Day day) | O (1) | Hàm tạo với day là day được truyền vào. |
| DailySchedule () | O (1) | Hàm tạo rỗng. |
| DailySchedule (Task task) | O (1) | Hàm tạo với ngày, công việc ban đầu có các hàm con được sử dụng bên trong đều có độ phức tạp là O (1). |
| addTask (CongViec cv) | O (n) | Sử dụng hàm addTask của lớp TaskList có độ phức tạp là O (n) trong trường hợp tệ nhất. |
| removeTask (CongViec cv) | O (n) | Sử dụng hàm removeTask của lớp TaskList có độ phức tạp là O (n) trong trường hợp tệ nhất. |
| getTaskByPriority (int tinhChatCongViec) | O (n) | Sử dụng hàm getTaskByPriority của lớp TaskList có độ phức tạp là O (n), bên cạnh đó còn sử dụng hàm printLine () để hiển thị cho đẹp có độ phức tạp chủ O (1). |
| getTaskByStatus (int trangThaiCongviec) | O (n) | Sử dụng hàm getTaskByStatus của lớp TaskList có độ phức tạp là O (n), bên cạnh đó còn sử dụng hàm printLine () để hiển thị cho đẹp có độ phức tạp chủ O (1). |
| getList () | O (1) | Gọi hàm getList của lớp TaskList có độ phức tạp chỉ O (1). |
| getDay () | O (1) | Trả về giá trị của thuộc tính ngày. |
| Print () | O (n) | Sử dụng hàm print của lớp TaskList có độ phức tạp là O (n). |
| setDone (Task task) | O (n) | Gọi phương thức getCongViec của lớp Task có độ phức tạp O (1) để truyền vào hàm setDone của lớp TaskList có độ phức tạp là O (n). |
| getListSize () | O (1) | Gọi hàm getSize của lớp TaskList có độ phức tạp chỉ O (1). |
| containTask (CongViec cv) | O (n) | Gọi hàm containTask của lớp TaskList có độ phức là O (n) trong trường hợp tệ nhất. |
| Các toàn tử so sánh | O (1) | Sử dụng toán tử so sánh của lớp Day tất cả đều có độ phức tạp là O (1). |
| Operator << (DailySchedule dl) | O (n) | Sử dụng toán tử << của lớp Day có độ phức tạp là O (1) và sử dụng hàm print của lớp TaskList có độ phức tạp là O (n). |
| Operator >> (DailySchedule dl) | O (n) | Sử dụng toán tử << của lớp Day có độ phức tạp là O (1) và sử dụng hàm addTask có độ phức tạp là O (n) trong trường hợp tệ nhất. |

## Lớp BST\_Node

|  |  |  |
| --- | --- | --- |
| Hàm | Độ phức tạp | Phân tích |
| BST\_Node (DailySchedule data) | O (1) | Hàm tạo với dữ liệu data ban đầu. |
| Insert (BST\_Node \*&curNode, Task val) | O (log(m) + n) | Hàm them công việc vào cây theo ngày với m là số lượng nút có trong mảng và n là số lượng nút có trong danh sách công việc. khi tìm ngày có thể them có độ phức tạp là log(m) nào tìm được mới bắt đầu gọi hàm addTask của lớp TaskList có độ phức tạp là O (n). |
| \*find (BST\_Node \*curNode, Task task) | O (log(m) + n) | Hàm tìm và trả về con trỏ trỏ tới nút có ngày là ngày ngày mình cần tìm vì tìm theo kiểu nếu nút có giá trị nhỏ hơn ngày mình cần tìm thì tìm bên phải và ngược lại, khi tìm thấy rồi thì tiếp tục tìm xem ngày đó có chứa công việc cần tìm hay không bằng cách sử dụng thêm hàm containTask của lớp DailySchedule có độ phức tạp là O (n). |
| \*find (BST\_Node \*curNode, Day day) | O (log(m)) | Hàm tìm và trả về con trỏ trỏ tới nút có ngày là ngày ngày mình cần tìm vì tìm theo kiểu nếu nút có giá trị nhỏ hơn ngày mình cần tìm thì tìm bên phải và ngược lại, khi tìm thấy rồi thì trả về giá trị. |
| DailySchedule max\_node (BST\_Node \*curr) | O (log(m)) | Hoạt động giống với hàm find nhưng tim nút có giá trị lớn nhất. |
| remove\_day (BST\_Node \*&root, Day day) | O (log(m)) | Hoạt động giống với hàm find nhưng khi tìm thấy sẽ xóa nút đó. |
| Remove (BST\_Node \*&root, Task val) | O (2log(m) + n) | Hàm sẽ sử dụng hàm find để tìm ngày của công việc cần xóa có độ phức tạp là O (log(m)) nếu không tìm được kết thúc chương trình còn nếu tìm được sẽ xóa công việc trong danh sách với độ phức tạp là O (n) trong trường hợp tệ nhất. khi xóa xong mà ngày này không cồn công việc nào nữa thì xóa này này ra khỏi cây bằng cách gọi hàm remove\_day có độ phức tạp là O (log(m)). |
| Inorder (BST\_Node \*root) | O (m \* n) | Hàm sẽ duyệt từng nút trong cây có m nút và ở mỗi nút sẽ in gọi hàm print của lớp TaskList với độ phức tạp là n với n là danh sách có số lượng công việc lớn nhất. |
| getTaskByPriority (BST\_Node \*root, int tinhChatCongViec) | O (m \* n) | Hàm duyệt từng nút trong cây có m nút và ở mỗi nút sẽ gọi hàm getTaskByPriority có độ phức tạp là O (n). |
| getTaskByStatus (BST\_Node \*root, int trangThaiCongViec) | O (m \* n) | Hàm duyệt từng nút trong cây có m nút và ở mỗi nút sẽ gọi hàm getTaskByStatus có độ phức tạp là O (n). |
| aPeriod (BST\_Node \*root, Day a, Day b) | O (m \* n) | Hàm duyệt từ ngày a đến ngày b những ngày nào không thuộc khoảng thời gian này sẽ không duyệt tuy nhiên trong trường hợp tệ nhất thì ta sẽ thăm hết toàn bộ nút như hàm Inorder. |

## Lớp DailyScheduleManager

|  |  |  |
| --- | --- | --- |
| Hàm | Độ phức tạp | Phân tích |
| DailyScheduleManager () | O (1) | Gán gốc của cây là nullptr |
| ~DailyScheduleManager () | O (1) | Xóa gốc cây. |
| getScheduleTree () | O (1) | Trả về địa chỉ của nút gốc cây. |
| addTask (Task dl) | O (log(m) + n) | Sử dụng hàm insert của lớp BST\_Node có độ phức tạp là O (log(m) + n). |
| removeTask (Task dl) | O (2log(m) + n) | Sử dụng hàm remove của lớp BST\_Node có độ phức tạp là O (2log(m) + n). |
| removeTask (Day day) | O (log(m)) | Sử dụng hàm remove\_day của lớp BST\_Node có độ phức tạp là O (log(m)). |
| viewTask (Day day) | O (log(m) + n) | Sử dụng hàm in tiêu đề của bảng có độ phức tạp là O (1) và hàm find của lớp BST\_Node có độ phức tạp là O (log(m)), nếu tìm thấy ngày ta muốn tìm thì sẽ in ra dữ liệu trong ngày đó bằng hàm print của lớp DailySchedule có đọ phức tạp là O (n). |
| viewAllTask () | O (m \* n) | Sử dụng hàm in tiêu đề của bảng có độ phức tạp là O (1) và hàm inorder của lớp BST\_Node có độ phức tạp là O (m \* n). |
| viewTaskByPriority (int tinhChatCongViec) | O (m \* n) | Sử dụng hàm in tiêu đề của bảng có độ phức tạp là O (1) và hàm getTaskByPriority của lớp BST\_Node có độ phức tạp là O (m \* n). |
| viewTaskDone (BST\_Node \*root) | O (m \* n) | Sử dụng hàm in tiêu đề của bảng có độ phức tạp là O (1) và hàm getTaskByStatus với giá trị trangThaiCongViec cần tìm là 1 của lớp BST\_Node có độ phức tạp là O (m \* n). |
| viewTaskNotDone (BST\_Node \*root) | O (m \* n) | Sử dụng hàm in tiêu đề của bảng có độ phức tạp là O (1) và hàm getTaskByStatus với giá trị trangThaiCongViec cần tìm là 0 của lớp BST\_Node có độ phức tạp là O (m \* n). |
| viewTaskByDay (BST\_Node \*root, Day a, Day b) | O (m \* n) | Sử dụng hàm in tiêu đề của bảng có độ phức tạp là O (1) và hàm aPeriod của lớp BST\_Node có độ phức tạp là O (m \* n) trong trường hợp tệ nhất. |
| changeSchedule (Task task, Day newDay) | O (4log(m) + 3n) | Sử dụng hàm find của lớp BST\_Tree có độ phức tạp là O (log(m) + n) nếu không tìm được thì kết thúc chương trình, còn nếu tìm được sẽ gọi hàm removeTask có độ phức tạp là O (2log(m) + n) sau đó sửa ngày công tác của công việc bằng hàm setDay của lớp task có độ phức tạp O (1). Cuối cùng thì sử dụng hàm insert của lớp BST\_Tree với độ phức tạp là O (log(m) + n). |
| setDone (Task task) | O (log(m) + 2n) | Sử dụng hàm find của lớp BST\_Node có độ phức tạp là O (log(m) + n) để tìm xem công việc có trong danh sách không nếu có thì thực hiện hàm setDone của DailySchedule có độ phức tạp là O (n) trong trường hợp tệ nhất. |

# Danh sách tài liệu tham khảo

* Danh sách liên kết, Cây tìm kiếm nhị phân: ThS. Phạm Xuân Tích
* Ý tưởng các lớp: ChatGpt.