# Đề thi cuối kỳ

Môn : Lập trình C nâng cao

Thời gian: 90 phút

**Bài 1:**

Có 2 người chơi A và B đang tranh giải vô địch thế giới Oẳn Tù Tì. Mỗi round mỗi người chơi phải quyết định ra hoặc Búa (B), hoặc Bao (G), hoặc Kéo (K). Người thắng mỗi round sẽ được 1 điểm.

Nhiệm vụ:

- Viết hàm getWinner in ra màn hình ai là người thắng. Hàm nhận 2 tham số dạng chuỗi: chuỗi đầu của người chơi A, chuỗi sau của người chơi B. Mỗi ký tự trong chuỗi tương ứng với nước đi của người chơi mỗi round. Ví dụ: "BBG" nghĩa là round 1 và 2 người chơi ra Búa, round 3 ra Bao.

- Hàm trả về lỗi nếu 2 chuỗi tham số không hợp lệ (xem lỗi ở ví dụ bên dưới)

Ví dụ 1:

|  |
| --- |
| **Input:**  BBK GBB  **Output:** B  Giải thích: B thắng do có 2 điểm thắng ở round 1 (G thắng B) và round 3 (B thắng K)  **Input:**  BBKK KKBB  **Output:** Hoa  Giải thích: Trận này hòa, mỗi bên 2 điểm |

Ví dụ 2:

|  |
| --- |
| Input:  BBKB GBB  Output: Error  Giải thích: Số round không bằng nhau, A 4 round trong khi B chỉ có 3 round |

Ví dụ 3:

|  |
| --- |
| Input:  BSA BGK  Output: Error  Giải thích: "S" "A" là các ký tự lạ |

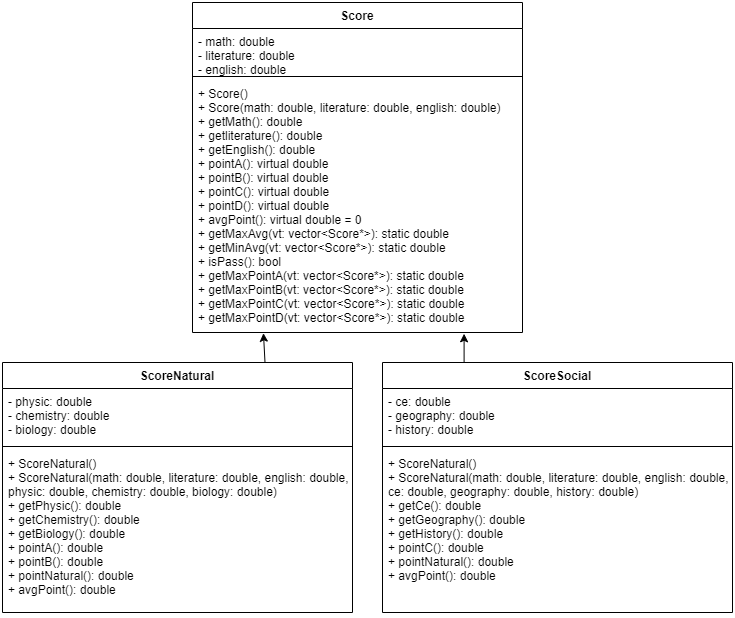
**Bài 2.** Hoàn thiện bài tập lập trình sau:

**Phần 2.1.** Bạn hãy tạo 3 lớp Score, ScoreNatual, ScoreSocial với các thông tin như phần giải thích bên dưới.

**Phần 2.2.** Viết hàm main để kiểm thử tính chính xác của chương trình.

* Khai báo vector gồm 3 đối tượng lớp ScoreNatural và khởi tạo giá trị cho các đối tượng này thông qua hàm tạo có tham số.
* Hiển thị điểm trung bình và thông tin trượt đỗ trượt/đỗ của ba đối tượng này.
* Khai báo vector gồm 3 đối tượng lớp ScoreSocial và khởi tạo giá trị cho các đối tượng này thông qua hàm tạo có tham số.
* Tạo vector các con trỏ Score \* lưu địa chỉ ít nhất 2 đối tượng lớp ScoreNatural và ít nhất 1 đối tượng lớp ScoreSocial .
* In điểm trung bình của các đối tượng nằm trong vector được tạo bên trên thông qua cơ chế hàm ảo.
* In điểm trung bình lớn nhất/nhỏ nhất sử dụng vector<Score\*>
* In điểm trung bình lớn nhất của các khối sử dụng vector<Score\*>

**Giải thích:**



Một thí sinh khi tham gia thi đại học sẽ có các môn bắt buộc phải thi là toán, văn, anh. Ngoài ra thí sinh phải chọn thêm một tổ hợp thi kèm là khoa học tự nhiên bao gồm các môn lý, hóa, sinh hoặc khoa học xã hội bao gồm các môn giáo dục công dân, lịch sử, địa lý.

**Lớp Score:**

* math, literature, english lần lượt là các thuộc tính chỉ điểm toán, văn, anh có phạm vi truy cập protected.
* Score() là phương thức khởi tạo không tham số.
* Score(math: double, literature: double, english: double) là phương thức khởi tạo 3 tham số.
* pointA(): virtual double là phương thức ảo trả về điểm của khối A tính bằng điểm math + physic + chemistry.
* pointB(): virtual double là phương thức ảo trả về điểm của khối B tính bằng điểm math + chemistry + biology.
* pointC(): virtual double là phương thức ảo trả về điểm của khối C tính bằng điểm literature + history + geography.
* pointD(): virtual double là phương thức ảo trả về điểm của khối D tính bằng điểm math+literature+english.
* avgPoint(): virtual double = 0 là phương thức thuần ảo trả về điểm trung bình của cả kì thi. Bao gồm các môn thi chính và tổ hợp kèm theo. Ví dụ thí sinh thi kèm tổ hợp khoa học tự nhiên thì avgPoint() trả về (physic + chemistry + biology + math + literature + english)/6.
* getMaxAvg(vt: vector<Score\*>): static double là phương thức tĩnh trả về avgPoint() cao nhất.
* getMinAvg(vt: vector<Score\*>): static double là phương thức tĩnh trả về avgPoint() thấp nhất.
* isPass(): bool là phương thức kiểm tra xem thí sinh có đỗ tốt nghiệp hay không. Trả về true nếu avgPoint()>5 và false ngược lại.
* getMaxPointA(vt: vector<Score\*>): static double là phương thức trả về điểm khối A cao nhất. Điểm khối A = toán + lý + hóa.
* getMaxPointB(vt: vector<Score\*>): static double là phương thức trả về điểm khối B cao nhất. Điểm khối B = toán + hóa + sinh.
* getMaxPointC(vt: vector<Score\*>): static double là phương thức trả về điểm khối C cao nhất. Điểm khối C = văn + sử + địa
* getMaxPointD(vt: vector<Score\*>): static double là phương thức trả về điểm khối D cao nhất. Điểm khối D = toán + văn + anh.
* Các phương thức còn lại là getter và setter.

**Lớp ScoreNatural:**

* Là lớp kế thừa public từ lớp Score.
* physic, chemistry, biology lần lượt là các thuộc tính chỉ điểm lý, hóa, sinh có phạm vi truy cập private.
* ScoreNatural() là phương thức khởi tạo không tham số.
* ScoreNatural(math: double, literature: double, english: double, physic: double, chemistry: double, biology: double) là phương thức khởi tạo 6 tham số.
* PointA(): double là phương thức trả về tổng điểm khối A.
* PointB(): double là phương thức trả về tổng điểm khối B.
* pointNatural(): double là phương thức trả về tổng điểm của bài khoa học tự nhiên.
* avgPoint(): double là phương thức trả về điểm trung bình của cả kì thi.
* Các phương thức còn lại là getter và setter.

**Lớp ScoreSocial**

* Là lớp kế thừa public từ lớp Score.
* ce, geography, history lần lượt là các phương thức chỉ điểm môn giáo dục công dân, địa lí, lịch sử có phạm vi truy cập private.
* ScoreNatural() là phương thức khởi tạo không tham số.
* ScoreNatural(math: double, literature: double, english: double, ce: double, geography: double, history: double) là phương thức khởi tạo 6 tham số.
* pointC(): double là phương thức trả về điểm khối C.
* pointNatural(): double là phương thức trả về tổng điểm bài thi tổ hợp khoa học xã hội.
* avgPoint(): double là phương thức trả về điểm trung bình của cả kì thi.
* Các phương thức còn lại là getter và setter.

# 

# Đáp án - Đề thi cuối kỳ

Môn : Lập trình C nâng cao

Thời gian: 90 phút

# 

**Bài 2.**

**Phần 2.1.**

Score.cpp

#pragma once

#include <iostream>

#include <vector>

using namespace std;

class Score

{

protected:

double math;

double literature;

double english;

public:

Score() {}

Score(double math, double literature, double english) : math(math), literature(literature), english(english) {}

double getMath()

{

return math;

}

double getLiterature()

{

return literature;

}

double getEnglish()

{

return english;

}

virtual double pointA()

{

return 0;

}

virtual double pointB()

{

return 0;

}

virtual double pointC()

{

return 0;

}

virtual double pointD()

{

return math + literature + english;

}

virtual double avgPoint() = 0;

virtual void display()

{

cout << "Avg = " << avgPoint() << endl;

}

static double getMaxAvg(vector<Score \*> vt)

{

double max = vt[0]->avgPoint();

for (int i = 1; i < vt.size(); i++)

{

if (vt[i]->avgPoint() > max)

{

max = vt[i]->avgPoint();

}

}

return max;

}

static double getMinAvg(vector<Score \*> vt)

{

double min = vt[0]->avgPoint();

for (int i = 1; i < vt.size(); i++)

{

if (vt[i]->avgPoint() < min)

{

min = vt[i]->avgPoint();

}

}

return min;

}

bool isPass()

{

if (avgPoint() > 5)

{

return true;

}

return false;

}

static double getMaxPointA(vector<Score \*> vt)

{

double max = vt[0]->pointA();

for (int i = 1; i < vt.size(); i++)

{

if (vt[i]->pointA() > max)

{

max = vt[i]->pointA();

}

}

return max;

}

static double getMaxPointD(vector<Score \*> vt)

{

double max = vt[0]->pointD();

for (int i = 1; i < vt.size(); i++)

{

if (vt[i]->pointD() > max)

{

max = vt[i]->pointD();

}

}

return max;

}

static double getMaxPointB(vector<Score \*> vt)

{

double max = vt[0]->pointB();

for (int i = 1; i < vt.size(); i++)

{

if (vt[i]->pointB() > max)

{

max = vt[i]->pointB();

}

}

return max;

}

static double getMaxPointC(vector<Score \*> vt)

{

double max = vt[0]->pointA();

for (int i = 1; i < vt.size(); i++)

{

if (vt[i]->pointC() > max)

{

max = vt[i]->pointC();

}

}

return max;

}

};

### ScoreNatural.cpp

#pragma once

#include "Score.cpp"

class ScoreNatural : public Score

{

private:

double physic;

double chemistry;

double biology;

public:

ScoreNatural() {}

ScoreNatural(double math, double literature, double english, double physic, double chemistry, double biology)

: Score(math, literature, english), physic(physic), chemistry(chemistry), biology(biology) {}

double getPhysic()

{

return physic;

}

double getChemistry()

{

return chemistry;

}

double getBiology()

{

return biology;

}

double pointA()

{

return math + chemistry + physic;

}

double pointB()

{

return math + chemistry + biology;

}

double pointNatual()

{

return physic + chemistry + biology;

}

double avgPoint()

{

return (physic + chemistry + biology + math + literature + english) / 6;

}

};

### ScoreSocial.cpp

#pragma once

#include "Score.cpp"

class ScoreSocial : public Score

{

private:

double ce;

double geography;

double history;

public:

ScoreSocial() {}

ScoreSocial(double math, double literature, double english, double ce, double geography, double history) : Score(math, literature, english), ce(ce), geography(geography), history(history) {}

double getCe()

{

return ce;

}

double getGeography()

{

return geography;

}

double getHistory()

{

return history;

}

double pointC()

{

return literature + history + geography;

}

double pointSocial()

{

return literature + history + geography;

}

double avgPoint()

{

return (literature + history + geography + math + literature + english) / 6;

}

};

### **Phần 2.2.**

int main()

{

vector<ScoreNatural> naturals(3);

naturals[0] = ScoreNatural(1, 1, 1, 1, 1, 1);

naturals[1] = ScoreNatural(2, 2, 2, 2, 2, 2);

naturals[2] = ScoreNatural(3, 3, 3, 3, 3, 3);

vector<ScoreSocial> socials(3);

socials[0] = ScoreSocial(1, 1, 1, 1, 1, 1);

socials[1] = ScoreSocial(2, 2, 2, 2, 2, 2);

socials[2] = ScoreSocial(3, 3, 3, 3, 3, 3);

for (auto &v : naturals)

if (v.isPass())

v.display();

for (auto &v : socials)

cout << v.avgPoint() << endl;

vector<Score \*> scores({&naturals[0], &naturals[1], &socials[0]});

for (auto &v : scores)

v->avgPoint();

cout << "Trung binh lon nhat:" << Score::getMaxAvg(scores) << endl;

cout << "Trung binh nho nhat:" << Score::getMinAvg(scores) << endl;

cout << "Trung binh lon nhat khoi A:" << Score::getMaxPointA(scores) << endl;

cout << "Trung binh lon nhat khoi B:" << Score::getMaxPointB(scores) << endl;

cout << "Trung binh lon nhat khoi C:" << Score::getMaxPointC(scores) << endl;

cout << "Trung binh lon nhat khoi D:" << Score::getMaxPointD(scores) << endl;

}