实验报告

东北林业大学

信息与计算机科学技术实验中心

|  |
| --- |
| 1. 实验目的   1·掌握类中静态成员的定义方法；  2·掌握静态数据成员的初始化方法、静态数据成员和静态成员函数的访问和使用方法；  3·掌握友元函数和友元类的说明方法；  4·理解友元函数和友元类的使用特点。 |
| 二、实验环境  Codeblocks |
| 三、实验内容及结果  1.编写程序，统计某旅馆住宿客人的总数。要求输入客人的姓名，输出客人编号（按先后顺序自动生成）、姓名以及总人数。  2.编写程序，统计歌手大奖赛的参赛人数，每名选手的平均得分。要求依次输入选手的编号、姓名、各评委的打分；输入参赛总人数，并按平均得分由高到低的顺序输出每位选手的名次、姓名、平均得分。  3.利用友元函数求两个数的平方差  4.编写程序，设计一个友元函数比较学生成绩的高低，并输出所有学生中最高分和最低分学生的姓名。  5.日期类Date包含日期的年、月、日三个数据成员，编写一个友元函数，求两个日期之间相差的天数。  6.点类Point，包含点的坐标x、y两个数据成员，设计两个友元函数，分别计算两点间的水平距离和垂直距离。  7. 采用友元类方式重新编写以上各题的程序。 |

|  |
| --- |
| 四、实验过程分析与讨论  1.编写程序，统计某旅馆住宿客人的总数。要求输入客人的姓名，输出客人编号（按先后顺序自动生成）、姓名以及总人数。  #include <iostream>  #include <vector>  #include <string>  class Guest {  public:  Guest(int id, const std::string& name) : id(id), name(name) {}  void print() const {  std::cout << "Guest ID: " << id << ", Name: " << name << std::endl;  }  static int getTotalGuests() {  return totalGuests;  }  static int totalGuests;  private:  int id;  std::string name;    };  int Guest::totalGuests = 0;  int main() {  std::vector<Guest> guests;  std::string name;  while (true) {  std::cout << "Enter guest name (or 'done' to finish): ";  std::cin >> name;  if (name == "done") break;  guests.push\_back(Guest(++Guest::totalGuests, name));  }  std::cout << "Total guests: " << Guest::getTotalGuests() << std::endl;  for (size\_t i = 0; i < guests.size(); ++i) {  guests[i].print();  }  return 0;  }  2.编写程序，统计歌手大奖赛的参赛人数，每名选手的平均得分。要求依次输入选手的编号、姓名、各评委的打分；输入参赛总人数，并按平均得分由高到低的顺序输出每位选手的名次、姓名、平均得分。  #include <iostream>  #include <vector>  #include <string>  #include <algorithm>  class Singer {  public:  Singer(int id, const std::string& name) : id(id), name(name) {}  void addScore(double score) {  scores.push\_back(score);  }  double getAverageScore() const {  double sum = 0;  for (size\_t i = 0; i < scores.size(); ++i) {  sum += scores[i];  }  return sum / scores.size();  }  void print() const {  std::cout << "ID: " << id << ", Name: " << name  << ", Average Score: " << getAverageScore() << std::endl;  }  int getId() const { return id; }  const std::string& getName() const { return name; }  private:  int id;  std::string name;  std::vector<double> scores;  };  bool compareSingers(const Singer& a, const Singer& b) {  return a.getAverageScore() > b.getAverageScore();  }  int main() {  int n;  std::cout << "Enter the number of singers: ";  std::cin >> n;  std::vector<Singer> singers;  for (int i = 0; i < n; ++i) {  int id;  std::string name;  std::cout << "Enter singer ID: ";  std::cin >> id;  std::cout << "Enter singer name: ";  std::cin >> name;  Singer singer(id, name);  std::cout << "Enter the number of scores: ";  int scoresCount;  std::cin >> scoresCount;  for (int j = 0; j < scoresCount; ++j) {  double score;  std::cout << "Enter score " << j + 1 << ": ";  std::cin >> score;  singer.addScore(score);  }  singers.push\_back(singer);  }  std::sort(singers.begin(), singers.end(), compareSingers);  std::cout << "Singers sorted by average score:\n";  for (size\_t i = 0; i < singers.size(); ++i) {  singers[i].print();  }  return 0;  }  3.利用友元函数求两个数的平方差  #include <iostream>  class Number {  private:  double value;  public:  Number(double val) : value(val) {}  friend double squareDifference(const Number& a, const Number& b);  };  double squareDifference(const Number& a, const Number& b) {  return (a.value \* a.value) - (b.value \* b.value);  }  int main() {  Number num1(5.0);  Number num2(3.0);  std::cout << "Square difference: " << squareDifference(num1, num2) << std::endl;  return 0;  }  4.编写程序，设计一个友元函数比较学生成绩的高低，并输出所有学生中最高分和最低分学生的姓名。  #include <iostream>  #include <vector>  #include <string>  class Student {  private:  std::string name;  double score;  public:  Student(const std::string& name, double score) : name(name), score(score) {}  friend void findHighLow(const std::vector<Student>& students, Student& highest, Student& lowest);  void print() const {  std::cout << "Name: " << name << ", Score: " << score << std::endl;  }  };  void findHighLow(const std::vector<Student>& students, Student& highest, Student& lowest) {  if (students.empty()) return;  highest = students[0];  lowest = students[0];  for (size\_t i = 1; i < students.size(); ++i) {  if (students[i].score > highest.score) {  highest = students[i];  }  if (students[i].score < lowest.score) {  lowest = students[i];  }  }  }  int main() {  std::vector<Student> students = {  {"Alice", 85.5},  {"Bob", 92.3},  {"Charlie", 78.4},  {"David", 88.1}  };  Student highest("", 0.0);  Student lowest("", 100.0);  findHighLow(students, highest, lowest);  std::cout << "Highest score:\n";  highest.print();  std::cout << "Lowest score:\n";  lowest.print();  return 0;  }  5.日期类Date包含日期的年、月、日三个数据成员，编写一个友元函数，求两个日期之间相差的天数。  #include <iostream>  #include <ctime>  class Date {  private:  int year, month, day;  public:  Date(int y, int m, int d) : year(y), month(m), day(d) {}  friend int daysBetween(const Date& date1, const Date& date2);  void print() const {  std::cout << year << "-" << month << "-" << day << std::endl;  }  };  int daysBetween(const Date& date1, const Date& date2) {  std::tm a = {0, 0, 0, date1.day, date1.month - 1, date1.year - 1900};  std::tm b = {0, 0, 0, date2.day, date2.month - 1, date2.year - 1900};  std::time\_t x = std::mktime(&a);  std::time\_t y = std::mktime(&b);  double difference = std::difftime(y, x) / (60 \* 60 \* 24);  return std::abs(static\_cast<int>(difference));  }  int main() {  Date date1(2023, 5, 30);  Date date2(2024, 5, 30);  std::cout << "Days between: " << daysBetween(date1, date2) << " days" << std::endl;  return 0;  }  6.点类Point，包含点的坐标x、y两个数据成员，设计两个友元函数，分别计算两点间的水平距离和垂直距离。  #include <iostream>  #include <cmath>  class Point {  private:  double x, y;  public:  Point(double x\_val, double y\_val) : x(x\_val), y(y\_val) {}  friend double horizontalDistance(const Point& p1, const Point& p2);  friend double verticalDistance(const Point& p1, const Point& p2);  void print() const {  std::cout << "(" << x << ", " << y << ")" << std::endl;  }  };  double horizontalDistance(const Point& p1, const Point& p2) {  return std::abs(p1.x - p2.x);  }  double verticalDistance(const Point& p1, const Point& p2) {  return std::abs(p1.y - p2.y);  }  int main() {  Point p1(3.0, 4.0);  Point p2(7.0, 1.0);  std::cout << "Horizontal distance: " << horizontalDistance(p1, p2) << std::endl;  std::cout << "Vertical distance: " << verticalDistance(p1, p2) << std::endl;  return 0;  }  7. 采用友元类方式重新编写以上各题的程序。#include <iostream>  #include <cmath>  class Point;  class DistanceCalculator {  public:  static double horizontalDistance(const Point& p1, const Point& p2);  static double verticalDistance(const Point& p1, const Point& p2);  };  class Point {  private:  double x, y;  public:  Point(double x\_val, double y\_val) : x(x\_val), y(y\_val) {}  friend class DistanceCalculator;  void print() const {  std::cout << "(" << x << ", " << y << ")" << std::endl;  }  };  double DistanceCalculator::horizontalDistance(const Point& p1, const Point& p2) {  return std::abs(p1.x - p2.x);  }  double DistanceCalculator::verticalDistance(const Point& p1, const Point& p2) {  return std::abs(p1.y - p2.y);  }  int main() {  Point p1(3.0, 4.0);  Point p2(7.0, 1.0);  std::cout << "Horizontal distance: " << DistanceCalculator::horizontalDistance(p1, p2) << std::endl;  std::cout << "Vertical distance: " << DistanceCalculator::verticalDistance(p1, p2) << std::endl;  return 0;  } |
|  |
| 五、指导教师意见  指导教师签字：  年 月 日 |