Jeff Hughes -- 0026991582

CS390Lang - CPP Homework

**Question 1**

Write the files Student.h and Student.cpp that implement a class **Student** that stores the name of a student and a map of string to ints with the lab and exam grades of that student. Also define the methods:

* Student(string name) constructor - Constructs a Student object
* setGrade(string lab, int grade) - sets the grade of a lab.
* int getGrade(string lab) - Returns the grade of a given lab.
* string toString() - Retuns a string with the name of the student and the labs and grades for that student.
* ~Student() // Destructor

Student.h:

#ifndef **CS390CPP\_FINAL\_STUDENT\_H**

#define **CS390CPP\_FINAL\_STUDENT\_H**

#include **<map>**

#include **<string>**

**using namespace** std;

**class** Student {

**private**:

string name;

map grades;

**public**:

Student(string name);

~Student();

**void** setGrade(string lab, **int** grade);

**int** getGrade(string lab);

string toString();

};

#endif *//CS390CPP\_FINAL\_STUDENT\_H*

Student.cpp:

#include **<sstream>**

#include **<map>**

#include **"Student.h"**

*// Constructor*

Student::Student(string name)

{

**this**->\_name = name;

}

*// Destructor*

Student::~Student() {

}

**void** Student::setGrade(string lab, **int** grade) {

**this**->\_grades.at(lab) = grade;

}

**int** Student::getGrade(string lab) {

**return this**->\_grades.at(lab);

}

string Student::toString() {

stringstream stream;

stream << **this**->\_name << **": "**;

map::iterator it;

**for**(it = **this**->\_grades.begin(); it != **this**->\_grades.end(); it++) {

stream << it->second << **" "**;

}

**return** stream.str();

}

**Question 2 and 3**

Write the files Lecture.h and Lecture.cpp that implement a class **Lecture** that stores the name of the lecture and a map with the students that are in the class.

Define the methods:

* Lecture(string name) - Constructs a Lecture Object of that name
* string getName() - Returns the lecture name
* addStudent(string studentName) - Adds a new student to the class.
* Student & getStudent(string studentName) - Returns a reference to the Student object for this student
* string toString() - Returns a string with the name of the class and the students in this lecture.
* save(string fileName) - Saves the Lecture to disk under this fileName
* open(string fileName) -Loads the Lecture to disk from this fileName
* ~Lecture() - Destructor.
* Create a copy constructor and an assignment operator for the class Lecture described before.

Lecture.h:

#ifndef **CS390CPP\_FINAL\_LECTURE\_H**

#define **CS390CPP\_FINAL\_LECTURE\_H**

#include **<string>**

#include **"Student.h"**

**using namespace** std;

**class** Lecture

{

**private**:

string \_name;

map \_students;

**public**:

Lecture **operator**=(Lecture &lecture);

Lecture(Lecture& lecture);

Lecture(string name);

~Lecture();

string getName() **const**;

**void** addStudent(string studentName);

Student &getStudent(string studentName) **const**;

string toString() **const**;

**void** save(string filename);

**void** open(string filename);

};

#endif *//CS390CPP\_FINAL\_LECTURE\_H*

Lecture.cpp:

#include **<sstream>**

#include **<fstream>**

#include **"Lecture.h"**

Lecture::Lecture(string name) {

**this**->\_name = name;

}

Lecture::Lecture(Lecture &lecture) {

**this**->\_name = lecture.\_name;

**this**->\_students = lecture.\_students;

}

Lecture **operator**=(Lecture &lecture) {

**return** Lecture(lecture);

}

Lecture::~Lecture() {

map::iterator it;

**for**(it = **this**->\_students.begin(); it != **this**->\_students.end(); it++) {

**delete** (it->second);

}

}

string Lecture::getName() **const** {

**return this**->\_name;

}

**void** Lecture::addStudent(string studentName) {

**this**->\_students.at(studentName) = **new** Student(studentName);

}

Student &Lecture::getStudent(string studentName) **const** {

**return this**->\_students.at(studentName);

}

string Lecture::toString() **const** {

stringstream stream;

stream << **this**->\_name << **": "**;

map::iterator it;

**for**(it = **this**->\_students.begin(); it != **this**->\_students.end(); it++) {

stream << it->first << **","**;

}

**return** stream.str();

}

**void** Lecture::save(string filename) {

ofstream out(filename.c\_str());

out << **this**->toString() << endl;

out.close();

}

**void** Lecture::open(string filename) {

ifstream in(filename.c\_str());

string str;

in >> str;

size\_t loc = str.find(**": "**);

string s = str.substr(0, loc);

**this**->\_name = s;

**while**( loc != -1 ) {

loc = str.find(**","**, loc+1);

size\_t l2 = str.find(**","**, loc+1);

**this**->addStudent(str.substr(loc, l2-loc));

}

}

**Question 4**

Create a template in Stack.h that creates a Stack of any type and any size passed as parameter. The stack should have methods such as pop(), push(), isEmpty(), isFull(). This is an example of how to use it.

Stack<int, 100> sint; // Creates a stack of 100 ints max.

**template** <**typename** T>

**class** Stack {

**private**:

T \*stack;

**int** max;

**int** ndx;

**public**:

Stack();

T pop();

**void** push(T item);

**bool** isEmpty() **const**;

**bool** isFull() **const**;

};

**Question 4 (second one)**

What is a dynamic cast and give an example code of how to use it?

A dynamic cast allows you to cast a variable from one type to another, given that the object is of the type.

Animal\* a = new Dog(“Jeff”);

Dog\* jeff = dynamic\_cast(a);

**Question 5**

Explain what is a virtual method.

A virtual method is a method that can be overwritten by a subclass.

**Question 6**

Explain what is an abstract class and explain how you can tell if a class is abstract or not.

Abstract classes are classes that contain only abstract methods, or methods with no code.

**Question 7**

When do you need virtual destructors?

Virtual destructors are used when you need to delete an instance of a derived class through a pointer to base class.

**Question 8**

What is public, private, and protected inheritance?

* Public: everything that is aware of Base and Child is also aware that Child inherits from Base.
* Protected: only Child, and its children, are aware that they inherit from Base.
* Private: no one other than Child is aware of the inheritance.

**Question 9**

What are the 5 memory allocation errors? Give a description and an example of each of them.

Memory Leaks: while (1) { ptr = new int; }  
Premature Free: int \* p = new int; \* p = 8; delete p; \*p = 9;  
Double Free: delete p; delete p;  
Wild Free: int q; int \* p = &q; delete p;  
Memory Smashing: char \* s = new char[8]; strcpy(s, “hello world”);

**Question 10**

When are constructors and destructors called in objects defined as local variables?

When they go out of scope

**Question 11**

Write a SmartPointer class that will wrap an object and it will keep a reference count on the object similar to the one covered in class.

#ifndef **CS390CPP\_FINAL\_SMARTPOINTER\_H**

#define **CS390CPP\_FINAL\_SMARTPOINTER\_H**

#include **<iostream>**

**using namespace** std;

**template**<**class** T>

**class** RefCnt {

**private**:

T \*\_obj;

**int** \_cnt;

**public**:

RefCnt(T \*p) : \_obj(p), \_cnt(0) {}

~RefCnt() {

**delete** \_obj;

}

T \*object() **const** { **return** \_obj; }

**int** inc() {

\_cnt++;

**return** \_cnt;

}

**int** dec() {

\_cnt--;

**return** \_cnt;

}

};

**template**<**class** T> **class** SmartPtr {

RefCnt<T>\* \_ptr;

**static void**\* **operator new**(size\_t) {}

**static void operator delete**(**void**\*) {}

**public**:

SmartPtr(T\* p) {

\_ptr = **new** RefCnt<T>(p);

\_ptr->inc();

}

SmartPtr(**const** SmartPtr<T>& p) {

\_ptr = p.\_ptr;

\_ptr->inc();

}

~SmartPtr() { **if** (\_ptr->dec() == 0) { **delete** \_ptr; } }

**void operator**=(**const** SmartPtr<T>& p) {

**if** (**this** != &p) {

**if** (\_ptr->dec() == 0) **delete** \_ptr;

\_ptr = p.\_ptr;

\_ptr->inc();

}

}

T\* **operator**->() **const** { **return** \_ptr->object(); }

T& **operator**\*() **const** { **return** \*(\_ptr->object()); }

};

#endif *//CS390CPP\_FINAL\_SMARTPOINTER\_H*