You can view this report online at : https://www.hackerrank.com/x/tests/1742749/candidates/57687212/report

Full Name:	Ashbah Faisal
Email:	af08271@st.habib.edu.pk
Test Name:	**CS 224 Week 11 Lab**
Taken On:	3 Nov 2023 08:48:32 PKT
Time Taken:	3710 min 50 sec/ 5500 min
Work Experience:	1 years
Invited by:	Munzir
Skills Score:	
Tags Score:	smart pointer 70/70 templates 70/70

100%

70/70

scored in **CS 224 Week 11 Lab** in 3710 min 50 sec on 3 Nov 2023 08:48:32 PKT

Recruiter/Team Comments:

No Comments.

	Question Description	Time Taken	Score	Status
Q1	**Smart Pointers Using Template Class** > **Coding**	3 hour 3 min 22 sec	70/ 70	✓

QUESTION 1

Correct Answer

Score 70

Smart Pointers Using Template Class > Coding

templates | smart pointer

QUESTION DESCRIPTION

Suppose you are designing a software application for a local hospital in Karachi, Pakistan. The application needs to manage patient records, and you decide to use smart pointers with templates to ensure efficient memory management. Write C++ code for a `SmartPointer` class template that will act as a smart pointer for dynamically allocated patient records.

The `SmartPointer` class template should have the following functionalities:

1. Allocation of memory for the patient record.
2. Deallocation of memory when the `SmartPointer` goes out of scope.
3. Overload the `->` operator to access the patient record's attributes.
4. Implement copy and move constructors and assignment operators for proper resource management.

Your task is to complete the `SmartPointer` class template and provide a `main` function that demonstrates its usage. You should use the following `PatientRecord` class to store patient information:

```
class PatientRecord {
public:
    PatientRecord(const std::string& name, int age, const std::string&
diagnosis)
        : name(name), age(age), diagnosis(diagnosis) {}
```

```
    std::string GetName() const { return name; }
    int GetAge() const { return age; }
    std::string GetDiagnosis() const { return diagnosis; }

private:
    std::string name;
    int age;
    std::string diagnosis;
};
```

Write the `SmartPointer` class template for the given `main` function, and ensure that the memory is correctly managed.

INTERVIEWER GUIDELINES

Solution from ChatGPT

```
#include <iostream>
#include <string>
#include <memory>

// Define the PatientRecord class
class PatientRecord {
public:
    PatientRecord(const std::string& name, int age, const std::string&
diagnosis)
        : name(name), age(age), diagnosis(diagnosis) {}

    std::string GetName() const { return name; }
    int GetAge() const { return age; }
    std::string GetDiagnosis() const { return diagnosis; }

private:
    std::string name;
    int age;
    std::string diagnosis;
};

// Define the SmartPointer class template
template <typename T>
class SmartPointer {
public:
    SmartPointer() : data(nullptr) {}
    SmartPointer(T* ptr) : data(ptr) {}
    ~SmartPointer() {
        delete data;
    }

    T* operator->() const {
        return data;
    }

    // Implement copy constructor
    SmartPointer(const SmartPointer& other) {
        data = new T(*other.data);
    }

    // Implement move constructor
    SmartPointer(SmartPointer&& other) noexcept {
        data = other.data;
        other.data = nullptr;
    }

    // Implement copy assignment
```

```
    SmartPointer& operator=(const SmartPointer& other) {
        if (this != &other) {
            delete data;
            data = new T(*other.data);
        }
        return *this;
    }

    // Implement move assignment
    SmartPointer& operator=(SmartPointer&& other) noexcept {
        if (this != &other) {
            delete data;
            data = other.data;
            other.data = nullptr;
        }
        return *this;
    }

private:
    T* data;
};

int main() {
    int choice;
    std::string name, diagnosis;
    int age;

    SmartPointer<PatientRecord> patientRecord;

    while (true) {
        std::cout << "Choose an option:\n1. Create Patient Record\n2.
Access Patient Record\n3. Exit\n";
        std::cin >> choice;

        if (choice == 1) {
            std::cout << "Enter patient name: ";
            std::cin >> name;
            std::cout << "Enter patient age: ";
            std::cin >> age;
            std::cout << "Enter patient diagnosis: ";
            std::cin >> diagnosis;

            // Create a new patient record and store it in the smart
pointer
            patientRecord = SmartPointer<PatientRecord>(new
PatientRecord(name, age, diagnosis));
        } else if (choice == 2) {
            if (patientRecord) {
                std::cout << "Patient Name: " << patientRecord->GetName()
<< "\n";
                std::cout << "Patient Age: " << patientRecord->GetAge()
<< "\n";
                std::cout << "Patient Diagnosis: " << patientRecord-
>GetDiagnosis() << "\n";
            } else {
                std::cout << "No patient record found.\n";
            }
        } else if (choice == 3) {
            break;
        } else {
            std::cout << "Invalid choice. Please try again.\n";
        }
    }

    return 0;
}
```

CANDIDATE ANSWER

Language used: **C++14**

```
#include <iostream>
#include <string>
using namespace std;


class PatientRecord{
    private:
        string name;
        int age;
        string diagnosis;
    public:
        PatientRecord(const string& name,int age,const string&
diagnosis):name(name),age(age),diagnosis(diagnosis){}
        string GetName()const{return name;}
        int GetAge()const{return age;}
        string GetDiagnosis()const{return diagnosis;}
};
template<class type>
class SmartPointer{
    private:
        type* pointer;
    public:
        SmartPointer():pointer(nullptr){}

        SmartPointer(type* arr):pointer(arr){}
        explicit operator bool() const{
            return pointer!=nullptr;
        }


        SmartPointer(const SmartPointer& t){
            pointer=nullptr;
            if (t.pointer){
                pointer= new type(*t.pointer);
            }

        }
        SmartPointer(SmartPointer&& t)noexcept:pointer(t.pointer){
            t.pointer=nullptr;
        }
        SmartPointer& operator=(const SmartPointer &t){
            if(this!= &t){
                delete pointer;
                pointer=nullptr;
            }
            if(t.pointer){
                pointer=new type(*t.pointer);
            }
            return *this;
        }

        SmartPointer& operator=(SmartPointer&& t)noexcept{
            if(this!= &t){
                delete pointer;
                pointer=t.pointer;
                t.pointer=nullptr;
            }

```

```
            return *this;
        }
        type* operator->(){
            return pointer;
        }
        ~SmartPointer(){
            delete pointer;
        }




};
int main() {
    int choice;
    std::string name, diagnosis;
    int age;

    SmartPointer<PatientRecord> patientRecord;

    while (true) {
        // std::cout << "Choose an option:\n1. Create Patient Record\n2.
Access Patient Record\n3. Exit\n";
        std::cin >> choice;

        if (choice == 1) {
            // std::cout << "Enter patient name: ";
            std::cin >> name;
            // std::cout << "Enter patient age: ";
            std::cin >> age;
            // std::cout << "Enter patient diagnosis: ";
            std::cin >> diagnosis;

            // TODO: Create a new patient record and store it in the smart
pointer
            patientRecord= SmartPointer<PatientRecord>(new
PatientRecord(name,age,diagnosis));

        } else if (choice == 2) {
            // TODO: If smart pointer is not null then print the patient
record stored in the pointer
            // TODO: Otherwise print the appropriate error message
            if (patientRecord){
                cout<<"Patient Name: "<<patientRecord->GetName()<<endl;
                cout<<"Patient Age: "<<patientRecord->GetAge()<<endl;
                cout<<"Patient Diagnosis: "<<patientRecord->GetDiagnosis()
<<endl;
            }
            else{
                cout<<"No patient record found."<<endl;
            }
        } else if (choice == 3) {
            // TODO: Exit the program
            break;
        } else {
            std::cout << "Invalid choice. Please try again.\n";
        }
    }

    return 0;
}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED

Testcase 0	Easy	Sample case	Success	10	0.0645 sec	8.88 KB
Testcase 1	Easy	Sample case	Success	10	0.0638 sec	8.91 KB
Testcase 2	Easy	Hidden case	Success	10	0.0594 sec	8.75 KB
Testcase 3	Easy	Hidden case	Success	10	0.0813 sec	8.83 KB
Testcase 4	Easy	Hidden case	Success	10	0.1208 sec	8.91 KB
Testcase 5	Easy	Hidden case	Success	10	0.0637 sec	8.93 KB
Testcase 6	Easy	Hidden case	Success	10	0.038 sec	8.88 KB

No Comments

PDF generated at: 5 Nov 2023 17:40:41 UTC