浙江理工大學
ZHEJIANG SCI-TECH UNIVERSITY

School of Computer Science and Technology

Lab Report of Object-Oriented Programming A

Lab 3: Polymorphism　　　　Credit hour: 3

Student Name: 余洋　　　　Student ID: 2023337621052

1 Objective

1.1 To understand the meaning of operator overloading.
1.2 To master the characteristics of overloading operator by using member function and friend function.
1.3 To master the calling method of overloading operator function.
1.4 To master the concept of dynamic binding.
1.5 To master the concept of virtual function and pure virtual function.

2 Introduction to lab principle

Realize the inheritance of abstract base class and operator overloading by using operator function calling.

3 Lab requirement

3.1 Software: C++ compiler under Windows or linux
3.2 Hardware: main memory(>2GB), free secondary memory(>40G), monitor and printer.

4 Lab content

Assume that the employees in a company are Manager, Technician and Salesperson. Please design a program (including Employee.h, Employee.cpp, Report.h, Report.cpp and demo.cpp etc.) to meet the following requirements.

Requirement:

4.1 The salary of employees in this company are calculated by month as

✧ For the Manager:	payment = salary + bonus
✧ For the Technician:	payment = salary
✧ For the Salesperson:	payment = salary + profit * 5%

4.2 For each employee, there are ID, name, gender, enroll date, position, salary, etc.

4.3 For each employee, the interface get_pay() is employed to calculate the payment and the operator << should be overloaded to realize the output of the employee-related information.

4.4 Design a Report class to display the employee's payment of that month and provided the following interfaces:

- ✧ Insert interface that can insert employee's information to Report class.
- ✧ Print interface that can display each employee's ID, name, gender, enroll date, payment of that month and calculate the max and min payment of this employee.

4.5 Overload the operator [] for Report class, make the position as the index and can search all the employees based on the position.

4.6 Write a main() function to testify your class definition.

- ✧ Firstly, create object for each kind of employee and by using insert interface of the Report class to insert these employee's information to the report.
- ✧ Secondly, display the employee's payment report of the month by using the print interface of Report class.

5 Code list

、、、 Report.h

```
#ifndef REPORT_H
#define REPORT_H

#include "Employee.h"
#include <map>
#include <vector>

class Report {
private:
    map<string, vector<Employee*>> employees_Position;

public:
    void insert(Employee* emp);
    void print() const;
    vector<Employee*> operator[](const string position) const;
};

#endif // REPORT_H
```

、、、 Employee.h

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

class Employee {
    protected:
        string id;
        string name;
        string gender;
        string enroll_date;
        string position;
        double salary;

    public:
        Employee() {}
        Employee(string id, string name, string gender, string enroll_date, string
position, double salary);

        virtual ~Employee() {}

        virtual double get_pay() { return -1; };
        string read_position();
        friend  ostream& operator<<(ostream& out, const Employee& e);
    };

class Manager : public Employee
{
private:
    double bonus;
public:

    double get_pay();
    Manager(string id, string name, string gender, string enroll_date, string position,
double salary ,double bonus): Employee(id, name, gender, enroll_date, position,
salary), bonus(bonus){}

};
class Technician : public Employee
{
```

```
public:

    double get_pay();
    Technician(string id, string name, string gender, string enroll_date, string position, double salary) : Employee(id, name, gender, enroll_date, position, salary) {}

};


class  Salesperson : public Employee
{
private:
    double profit;
public:

    double get_pay();
    Salesperson(string id, string name, string gender, string enroll_date, string position, double salary, double profit) : Employee(id, name, gender, enroll_date, position, salary), profit(profit) {}

};
```

、、、 Report.cpp

```
#include "Report.h"
#include <iostream>

void Report::insert(Employee* emp) {
    employees_Position[emp->read_position()].push_back(emp);
}

void Report::print() const {
    for (const auto& entry : employees_Position) {
        cout << "Position: " << entry.first << endl;
        for (Employee* emp : entry.second) {
            cout << *emp << ", Payment: " << emp->get_pay() << endl;

        }
        cout << "----------------------------" << endl << endl;
    }
    double max_pay = -1;
    double min_pay = 999999999;
    for (const auto& entry : employees_Position) {
        for (Employee* emp : entry.second) {
            double pay = emp->get_pay();
            if (pay > max_pay) max_pay = pay;
            if (pay < min_pay) min_pay = pay;
        }
    }
    cout << "Max Payment: " << max_pay << endl;
    cout << "Min Payment: " << min_pay << endl<<endl;
    cout << "------------------------------------" << endl;
}

vector<Employee*> Report::operator[](const  string position) const {
    auto it = employees_Position.find(position);
    if (it != employees_Position.end()) {
        return it->second;
    }
    return  vector<Employee*>();
}
```

、、、Employee.cpp

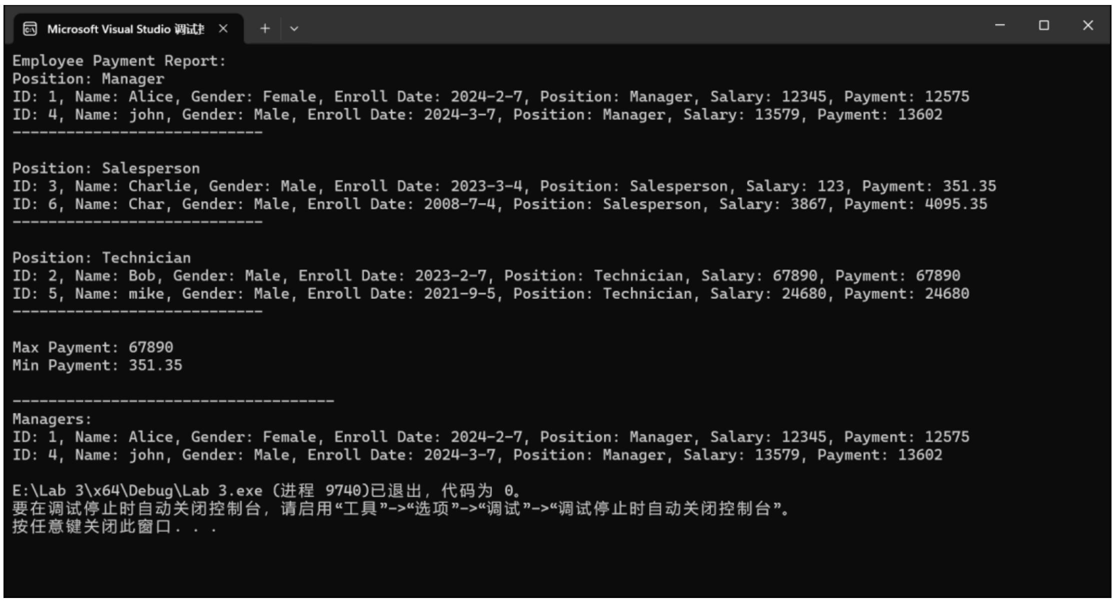
```
#include "Employee.h"


ostream& operator<<(ostream& out, const Employee& e) {
    out << "ID: " << e.id
        << ", Name: " << e.name
        << ", Gender: " << e.gender
        << ", Enroll Date: " << e.enroll_date
        << ", Position: " << e.position
        << ", Salary: " << e.salary;
    return out;
}
Employee::Employee(string id, string name, string gender, string enroll_date, string
position, double salary):id(id), name(name), gender(gender), enroll_date(enroll_date),
position(position), salary(salary) {}
string Employee::read_position()
```

```
{
    return position;
}
double Manager::get_pay()
{
    return salary + bonus;
}

double Technician::get_pay()
{
    return salary ;
}

double Salesperson::get_pay()
{
    return salary + profit*0.05;
}
```

、、、main.cpp

```
#include "Report.h"
#include <iostream>
#include<string>
using namespace std;

int main() {

    Report report;
    Manager A("1", "Alice", "Female", "2024-2-7", "Manager", 12345, 230);
    Technician B("2", "Bob", "Male", "2023-2-7", "Technician", 67890);
    Salesperson C("3", "Charlie", "Male", "2023-3-4", "Salesperson", 123, 4567);
    Manager D("4", "john", "Male", "2024-3-7", "Manager", 13579, 23);
    Technician E("5", "mike", "Male", "2021-9-5", "Technician", 24680);
    Salesperson F("6", "Char", "Male", "2008-7-4", "Salesperson", 3867, 4567);



    report.insert(&A);
    report.insert(&B);
    report.insert(&C);
    report.insert(&D);
    report.insert(&E);
    report.insert(&F);

    cout << "Employee Payment Report:" << endl;
    report.print();

    vector<Employee*> managers = report["Manager"];
    cout << "Managers:" << endl;
    for (Employee* emp : managers) {
        cout << *emp << ", Payment: " << emp->get_pay() << endl;
    }

    return 0;
}
```

6 Output



```
Employee Payment Report:
Position: Manager
ID: 1, Name: Alice, Gender: Female, Enroll Date: 2024-2-7, Position: Manager, Salary: 12345, Payment: 12575
ID: 4, Name: john, Gender: Male, Enroll Date: 2024-3-7, Position: Manager, Salary: 13579, Payment: 13602
----------------------------

Position: Salesperson
ID: 3, Name: Charlie, Gender: Male, Enroll Date: 2023-3-4, Position: Salesperson, Salary: 123, Payment: 351.35
ID: 6, Name: Char, Gender: Male, Enroll Date: 2008-7-4, Position: Salesperson, Salary: 3867, Payment: 4095.35
----------------------------

Position: Technician
ID: 2, Name: Bob, Gender: Male, Enroll Date: 2023-2-7, Position: Technician, Salary: 67890, Payment: 67890
ID: 5, Name: mike, Gender: Male, Enroll Date: 2021-9-5, Position: Technician, Salary: 24680, Payment: 24680
----------------------------

Max Payment: 67890
Min Payment: 351.35

------------------------------------
Managers:
ID: 1, Name: Alice, Gender: Female, Enroll Date: 2024-2-7, Position: Manager, Salary: 12345, Payment: 12575
ID: 4, Name: john, Gender: Male, Enroll Date: 2024-3-7, Position: Manager, Salary: 13579, Payment: 13602

E:\Lab 3\x64\Debug\Lab 3.exe (进程 9740)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

7 Analysis and conclusions

My code defines a base class Employee, which contains basic information about employees, including ID, name, gender, start date, position, and salary. This class uses C++'s protected members to define properties that can be accessed by derived classes. At the same time, it also defines a virtual function get_pay, which has different implementations in derived classes to calculate the wages of different types of employees, reflecting polymorphism.

my code defines three derived classes: Manager, Technician, and Salesperson. These classes inherit from the Employee class and add specific attributes (for example, the Manager class has a bonus attribute and the Salesperson class has a profit attribute), as well as rewrite the get_pay function to provide the logic for calculating their respective salaries.

The Employee class also defines a constructor and a destructor. The constructor is used to initialize the object, while the destructor is used to perform cleanup work when the object is destroyed. The destructor is declared virtual to ensure that derived class objects can correctly call the corresponding destructor when deleted.

In addition, my code also defines a Report class that uses a map to store vectors pointing to employees in different positions. This class provides the insert method to add employees to the corresponding positions, the print method to print information for all employees, and overloads operator [] to allow access to a specific list of employees for a position through position name indexing.