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: 代翔　　　　Student ID: 2023337621159

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

```
#pragma once
#include <iostream>
using namespace std;
#include <string>
#include <Windows.h>
class Time
{
public:
	SYSTEMTIME cur_time;
public:
	SYSTEMTIME get_time()
	{
		GetLocalTime(&cur_time);
		return cur_time;
	}
};

class Employee
{
protected:
	string ID;
	string name;
	string gender;
	SYSTEMTIME enroll_date;
	string position;
	double salary;
public:
	virtual double get_pay() = 0;
};

class Mangager:virtual Employee
{
	friend class report;
	friend ostream& operator<<(ostream& out, Mangager MA);
	double bonus;
public:
	Mangager(string p_ID, string p_name, string p_gender, double p_salary, double p_bonus);
	double get_pay();
};

class Technician :virtual Employee
{
	friend class report;
	friend ostream& operator<<(ostream& out, Technician TE);
public:
	Technician(string p_ID, string p_name, string p_gender, double p_salary);
	double get_pay();
};

class Salesperson :virtual Employee
{
	friend class report;
	friend ostream& operator<<(ostream& out, Salesperson SA);
	double profits;
public:
	Salesperson(string p_ID, string p_name, string p_gender, double p_salary, double p_profits);
	double get_pay();
	};

#pragma once
#include <vector>
#include "Employee.h"
#include <algorithm>
```

```
class report
{
	vector<Mangager>* MA_list = new vector<Mangager>();
	vector<Technician>* T_list = new vector<Technician>();
	vector<Salesperson>* SP_list = new vector<Salesperson>();

public:
	void operator [](string index)
	{
		if (index == "mangager")
		{
			for (auto it = MA_list->begin(); it != MA_list->end(); it++)
			{
				cout << "ID: " << it->ID << "," << "NAME: " << it->name << "SALARY: " <<
it->get_pay() << ",";
				printf("%04d/%02d/%02d-%02d:%02d:%02d\n", it->enroll_date.wYear,
it->enroll_date.wMonth, it->enroll_date.wDay, it->enroll_date.wHour, it->enroll_date.wMinute,
it->enroll_date.wSecond);
			}
		}
		else if (index == "technician")
		{
			for (auto it = T_list->begin(); it != T_list->end(); it++)
			{
				cout << "ID: " << it->ID << "," << "NAME: " << it->name << "SALARY: " <<
it->get_pay() << ",";
				printf("%04d/%02d/%02d-%02d:%02d:%02d\n", it->enroll_date.wYear,
it->enroll_date.wMonth, it->enroll_date.wDay, it->enroll_date.wHour, it->enroll_date.wMinute,
it->enroll_date.wSecond);
			}
		}
		else if (index == "salesperson")
		{
			for (auto it = SP_list->begin(); it != SP_list->end(); it++)
			{
				cout << "ID: " << it->ID << "," << "NAME: " << it->name << "SALARY: " <<
it->get_pay() << ",";
				printf("%04d/%02d/%02d-%02d:%02d:%02d\n", it->enroll_date.wYear,
it->enroll_date.wMonth, it->enroll_date.wDay, it->enroll_date.wHour, it->enroll_date.wMinute,
it->enroll_date.wSecond);
			}
			}
			else
			{
				cout << "Error" << endl;
			}
		}

		void insert(Mangager a);
		void insert(Technician b);
		void insert(Salesperson c);

		void print(string position);
		};
#include "Report.h"


void report::insert(Mangager a)
{
	MA_list->push_back(a);
}

void report::insert(Technician b)
{
	T_list->push_back(b);
}

void report::insert(Salesperson c)
{
	SP_list->push_back(c);
}

bool compare(Mangager& a, Mangager& b)
{
	return a.get_pay() < b.get_pay();
}

bool compare1(Technician& a, Technician& b)
{
	return a.get_pay() < b.get_pay();
}
```

```
bool compare2(Salesperson& a, Salesperson& b)
{
	return a.get_pay() < b.get_pay();
}

void report::print(string position)
{
	report a;
	a[position];
	double max;
	double min;
	if (position == "mangager")
	{
		auto max_it = max_element(MA_list->begin(), MA_list->end(), compare);
		max = max_it->salary;
		auto min_it = min_element(MA_list->begin(), MA_list->end(), compare);
		min = min_it->salary;
		cout << "min: " << min << endl;
		cout << "max: " << max << endl;
	}
	if (position == "technition")
	{
		auto max_it = max_element(T_list->begin(), T_list->end(), compare1);
		max = max_it->salary;
		auto min_it = min_element(T_list->begin(), T_list->end(), compare1);
		min = min_it->salary;
		cout << "min: " << min << endl;
		cout << "max: " << max << endl;
	}
	if (position == "salesperson")
	{
		auto max_it = max_element(SP_list->begin(), SP_list->end(), compare2);
		max = max_it->salary;
		auto min_it = min_element(SP_list->begin(), SP_list->end(), compare2);
		min = min_it->salary;
		cout << "min: " << min << endl;
		cout << "max: " << max << endl;
	}
}#include "Employee.h"

ostream& operator<<(ostream& out,Mangager MA)
```

```
{
	out << "ID: " << MA.ID << endl << " NAME: " << MA.name << endl << " GENDER: " << MA.gender
<< endl << " SALARY: " << MA.get_pay() << endl;
	printf("%04d/%02d/%02d-%02d:%02d:%02d\n",MA.enroll_date.wYear,MA.enroll_date.wMont
h,MA.enroll_date.wDay,MA.enroll_date.wHour,MA.enroll_date.wMinute,MA.enroll_date.wSecond);
	return out;
}

Mangager::Mangager(string p_ID, string p_name, string p_gender, double p_salary, double p_bonus)
{
	position = "mangager";
	ID = p_ID;
	name = p_name;
	gender = p_gender;
	salary = p_salary;
	bonus = p_bonus;
}

double Mangager::get_pay()
{
	double total = salary + bonus;
	return total;
}

ostream& operator<<(ostream& out, Technician TE)
{
	out << "ID: " << TE.ID << endl << " NAME: " << TE.name << endl << " GENDER: " << TE.gender <<
endl << " SALARY: " << TE.get_pay() << endl;
	printf("%04d/%02d/%02d-%02d:%02d:%02d\n", TE.enroll_date.wYear, TE.enroll_date.wMonth,
TE.enroll_date.wDay, TE.enroll_date.wHour, TE.enroll_date.wMinute, TE.enroll_date.wSecond);
	return out;
}

Technician::Technician(string p_ID, string p_name, string p_gender, double p_salary)
{
	position = "technician";
	ID = p_ID;
	name = p_name;
	gender = p_gender;
	salary = p_salary;
}
```

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

ostream& operator<<(ostream& out, Salesperson SA)
{
    out << "ID: " << SA.ID << endl << " NAME: " << SA.name << endl << " GENDER: " << SA.gender <<
endl << " SALARY: " << SA.get_pay() << endl;
    printf("%04d/%02d/%02d-%02d:%02d:%02d\n", SA.enroll_date.wYear, SA.enroll_date.wMonth,
SA.enroll_date.wDay, SA.enroll_date.wHour, SA.enroll_date.wMinute, SA.enroll_date.wSecond);
    return out;
}

Salesperson::Salesperson(string p_ID, string p_name, string p_gender, double p_salary, double
p_profits)
{
    position = "salesperson";
    ID = p_ID;
    name = p_name;
    gender = p_gender;
    salary = p_salary;
    profits = p_profits;
}

double Salesperson::get_pay()
{
    double total = salary + 0.05 * profits;
    return total;
    }
#include "Report.h"

int main()
{
    report T;

    Mangager p1("2020", "lijun", "man", 400, 78);
    Mangager p2("2021", "list", "woman", 407, 98);
    Mangager p3("2022", "ppq", "man", 800, 108);
Technician p4("2023", "well", "woman", 1107);
Technician p5("2024", "pig", "man", 107);
Technician p6("2025", "jun", "woman", 207);

Salesperson p7("2026", "ning", "man", 190, 880);
Salesperson p8("2027", "OI", "woman", 188, 800);
Salesperson p9("2028", "who", "man", 590, 180);

T.insert(p1);
T.insert(p2);
T.insert(p3);
T.insert(p4);
T.insert(p5);
T.insert(p6);
T.insert(p7);
T.insert(p8);
T.insert(p9);

T["mangager"];
T["technician"];
T["salesperson"];

T.print("mangager");
T.print("technician");
T.print("salesperson");
}
```

6 Output

```
ID: 2020,NAME: lijun, SALARY: 478,2024/06/12-12:16:26
ID: 2021,NAME: list, SALARY: 505,2024/06/12-12:16:26
ID: 2022,NAME: ppq, SALARY: 908,2024/06/12-12:16:26
ID: 2023,NAME: well, SALARY: 1007,2024/06/12-12:16:26
ID: 2024,NAME: pig, SALARY: 107,2024/06/12-12:16:26
ID: 2025,NAME: jun, SALARY: 207,2024/06/12-12:16:26
ID: 2026,NAME: ning, SALARY: 234,2024/06/12-12:16:26
ID: 2027,NAME: OI, SALARY: 228,2024/06/12-12:16:26
ID: 2029,NAME: who, SALARY: 599,2024/06/12-12:16:26
min: 400
max: 800
min: 107
max: 1007
```

```
min: 188
max: 590
```

7 Analysis and conclusions

Analysis and Conclusion

In this laboratory exercise, we were tasked with implementing a program that simulates the salary calculation and reporting process for a company with three types of employees: Managers, Technicians, and Salespersons. The program utilizes inheritance, operator overloading, and dynamic binding to accomplish this task.

Inheritance

The Employee class was designed as an abstract base class, containing common attributes such as ID, name, gender, enroll date, and salary. The Manager, Technician, and Salesperson classes inherited from Employee and added specific attributes or behaviors such as bonus for Manager and profit percentage for Salesperson. This allowed us to reuse code and define common behavior in the base class while specializing it for each type of employee.

Operator Overloading

The operator << was overloaded in the Employee class to allow for easy printing of employee information. This overloaded operator takes an output stream object (usually std::cout) and an Employee object as parameters, and then uses the stream insertion operator to output the employee's attributes. Similarly, the [] operator was overloaded in the Report class to allow searching employees based on their position.

Dynamic Binding

Dynamic binding, achieved through virtual functions, was used in the get_pay() method. This method is declared as virtual in the Employee class and is overridden in the derived classes (Manager, Technician, Salesperson) to provide specific salary calculation logic. At runtime, the appropriate version of get_pay() is called based on the type of the object, allowing for polymorphic behavior.

Report Class

The Report class was designed to handle the reporting and aggregation of employee payments. It provides interfaces such as insert() to add employee information and print() to display each employee's details along with their monthly payment. The print() method also calculates the maximum and minimum payments among all the employees.

Testing

In the main() function, objects of each employee type were created, and their information was inserted into the Report object using the insert() method. The print() method was then called to display the employee details and payments. The overloaded [] operator in the Report class was tested by using different positions as indices to search for employees.

Conclusion

Through this laboratory exercise, we gained hands-on experience with the concepts of inheritance, operator overloading, dynamic binding, and virtual functions in C++. The program successfully simulated the salary calculation and reporting process for a company with different types of employees. The use of inheritance allowed for code reuse and specialization, while operator overloading enhanced the usability and readability of the code. Dynamic binding and virtual functions enabled polymorphic behavior, allowing for the correct version of a method to be called based on the type of the object. Overall, this laboratory exercise provided a valuable learning experience in applying object-oriented programming concepts in C++.