Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа
по курсу «Объектно-ориентированное программирование»
III Семестр

Задание 2
Вариант 8
Операторы, литералы

Студент:	Жерлыгин М.А
Группа:	М8О-208Б-18
Преподаватель:	Журавлев А.А.
Оценка:	
Дата:	

Москва 2019

1. Код программы на языке C++

1. Файл money.h

```
#ifndef D_MONEY_H_
#define D_MONEY_H_

#include <iostream>

class Money {
        private:
                unsigned long long roub; // количество рублей
                uint16_t cop; // количество копеек
        public:
                Money() {
                        roub = 0;
                        cop = 0;
                }
                unsigned long long roub_get(const Money& a);
                uint16_t cop_get(const Money& a);
                void get(std::istream& is); // Получить сумму std::istream&
                void show(std::ostream& os) const; // Вывести сумму
                Money Plus(const Money& a2); // сложение сумм
                Money Minus(const Money& a2); // вычитание сумм
                double Div(const Money& a2); // Деление 2х сумм
                Money DivN(double arg); // Деление суммы на число
                Money Mult(double arg); // Умножение суммы на число
                bool operator<(const Money& a2); // <
                bool operator>(const Money& a2); // >
                bool operator<=(const Money& a2); // <=
                bool operator>=(const Money& a2); // >=
                bool operator==(const Money& a2); // ==
                bool operator!=(const Money& a2); // !=
};

long double operator ""_toDollar(long double cash);
```

2. Файл money.cpp

```
#include "money.h"

unsigned long long Money::roub_get(const Money& a) {
        return a.roub;
}

uint16_t Money::cop_get(const Money& a) {
        return a.cop;
}
void Money::get(std::istream& is) {
        unsigned long long temp_roub;
    uint16_t temp_cop;
    char temp;
    is >> temp_roub >> temp >> temp_cop;
    this->roub = temp_roub;
    this->cop = temp_cop;
}

void Money::show(std::ostream& os) const {
        os << roub << ",";
```

```
        if(cop <= 0) {
                os << "00" << std::endl;
        }
        else os << cop << std::endl;
}


Money Money::Plus(const Money& a2) {
        Money temp;
        temp.roub = this->roub + a2.roub;
        temp.cop = this->cop + a2.cop;
        if(temp.cop >= 100) {
                temp.roub++;
                temp.cop -= 100;
        }
        return temp;
}

Money Money::Minus(const Money& a2) {
        Money temp;
        temp.roub = this->roub - a2.roub;
        if(this->cop < a2.cop) {
                temp.roub--;
                this->cop += 100;
                temp.cop = this->cop - a2.cop;
                this->cop -= 100;
        }
        else temp.cop = this->cop - a2.cop;
        return temp;
}

double Money::Div(const Money& a2) {
        double temp, a, b;
        a = ((this->roub * 100) + this->cop);
        b = ((a2.roub * 100) + a2.cop);
        temp = a / b;
        return temp;
}

Money Money::DivN(double arg) {
        Money temp;
        double tempN;
        tempN = (((this->roub * 100) + this->cop) / arg) / 100;
        temp.roub = (int)tempN;
        temp.cop = (tempN - (int)tempN) * 100;
        return temp;
}

Money Money::Mult(double arg) {
        Money temp;
        double tempN;
        tempN = (((this->roub * 100) + this->cop) * arg) / 100;
        temp.roub = (int)tempN;
        temp.cop = (tempN - (int)tempN) * 100;
        return temp;
}

bool Money::operator<(const Money& a2) {
        return (this->roub < a2.roub || (this->roub == a2.roub && this->cop < a2.cop));
}

bool Money::operator>(const Money& a2) {
        return (this->roub > a2.roub || (this->roub == a2.roub && this->cop > a2.cop));
}
```

```
bool Money::operator<=(const Money& a2) {
        return (this->roub <= a2.roub && this->cop <= a2.cop);
}

bool Money::operator>=(const Money& a2) {
        return (this->roub >= a2.roub && this->cop >= a2.cop);
}

bool Money::operator==(const Money& a2) {
        return (this->roub == a2.roub && this->cop == a2.cop);
}

bool Money::operator!=(const Money& a2) {
        return (this-> roub != a2.roub || this->cop !=a2.cop);
}

long double operator ""_toDollar(long double cash) {
        return (cash * 64.27);
}
```

3. Файл main.cpp

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

#define UNUSED(variable) (void)variable

int main(int argc, char** argv) {

        Money a1;
        Money a2;
        float arg;

        std::cout << "First summ:" << std::endl;
        a1.get(std::cin);
        std::cout << "Second summ:" << std::endl;
        a2.get(std::cin);
        std::cout << "Number to div and multiply" << std::endl;
        std::cin >> arg;
        if(arg == 0){
                std::cout << "cannot be divided by zero" << std::endl;
                return 0;
        }
        std::cout << std::endl;

        a1.show(std::cout);
        a2.show(std::cout);

        a1.Plus(a2).show(std::cout);

        a1.Minus(a2).show(std::cout);

        std::cout << a1.Div(a2) << std::endl;

        a1.DivN(arg).show(std::cout);

        a1.Mult(arg).show(std::cout);

        if(a1 < a2) std::cout << "<:" << std::endl;
        else std::cout << "<: false" << std::endl;

        if(a1 > a2) std::cout << ">: true" << std::endl;
```

```
        else std::cout << ">: false" << std::endl;

        if(a1 <= a2) std::cout << "<=: true" << std::endl;
        else std::cout << "<=: false" << std::endl;

        if(a1 >= a2) std::cout << ">=: true" << std::endl;
        else std::cout << ">=: false" << std::endl;

        if(a1 == a2) std::cout << "==: true" << std::endl;
        else std::cout << "==: false" << std::endl;

        if(a1 != a2) std::cout << "!=: true" << std::endl;
        else std::cout << "!=: false" << std::endl;


        long double dollar;
        dollar = 5000.00_toDollar;

        std::cout << "5000$ = " << dollar << "P" << std::endl;

        UNUSED(argc);
        UNUSED(argv);

        return 0;
}
```

2. Ссылка на репозиторий на Github

https://github.com/mmaxim2710/oop_exercise_02

3.Набор testcases

1. 1000
500
2

2. 1000.25
50.64
0

3. 500
500
3

4. 1234.5
123
4

4. Результат выполнения тестов

1. 1000,00
500,00
1500,00
500,00
2000,00
<: false
>: true
<=: false
>=: true
==: false
!=: true
$:321350

2. cannot be divided by zero

3. 500,00
500,00
1000,00
0,00
1,00
166,66
1500,00
<: false
>: false
<=: true
>=: true
==: true
!=: false
$:321350

4. 1234,50
123,00
1357,50
1111,50
10,68
308,50
4936,00
<: false
>: true
<=: false
>=: true
==: false
!=: true
$:321350

5. Объяснение результатов программы

Данная программа создает класс Money для работы с денежными суммами. Сумма денег представлена двумя полями: типа unsigned long long для рублей и типа unsigned short int – для копеек. Дробная часть (копейки) при выводе на экран отделена от целой части запятой. Реализовано сложение сумм, вычитание, деление сумм, деление суммы на дробное число, умножение на дробное число и перегружены операторы сравнения.

В программе реализованы следующие функции:

1. Вывести количетсво рублей
2. Сложить 2 суммы
3. Вычесть из первой суммы вторую
4. Разделить первую сумму на вторую
5. Разделить первую сумму на дробное число
6. Умножить первую сумму на дробное число
7. Оператор <
8. Оператор >
9. Оператор <=
10. Оператор >=
11. Оператор ==
12. Оператор !=

При выполении первой функции программа выводит количество денег в формате <рубли>,<копейки>. Рубли и копейки разделяются в разные переменные класса Money с помощью функции shiftUp().
При выполнении второй функции программа просит ввести вторую сумму, затем складывает с первой, записывает в класс a2 для последующего использования функции shiftUp() и вывода.
Третья и четвертая функции работают аналогично второй.
Пятая и шестая функции берут на вход дробное число типа float, проводят соответствующие операции и выводят сумму аналогично пунктам 2-4.
Перегрузка операторов 7-12 — операторы сравнения. Они берут на вход вторую сумму, которая будет сравниваться с первоначальной, затем выводится результат true/false (true в случае верного сравнения, false – в противном случае).

Вывод: Проделав данную работу я изучил перегрузку операторов и пользовательские литералы. Сделал вывод, что перегрузка операторов необходима для переопределения того, что должен делать данный оператор, если это необходимо. Например, если класс реализован двумя переменными, и их необходимо сравнить как одно целое, то необходимо применить перегрузку операторов. Так же пользовательские литералы

необходимы для просчета константных переменных где это возможно, так как расчеты производятся на этапе компиляции.