МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №02

по курсу “Объектно-ориентированное программирование» 1 семестр,

2021/22 уч. год

Студент: *Колпакова Диана Саргаевна, группа М8О-208Б-20*
Преподаватель: *Дорохов Евгений Павлович*

Задание

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод. Операции над объектами реализовать в виде перегрузки операторов. Реализовать пользовательский литерал для работы с константами объектов созданного класса.

Вариант 9:

Создать класс BritishMoney для работы с денежными суммами в старой британской системе. Сумма денег должна быть представлена тремя полями: типа unsigned long long для фунтов стерлингов, типа unsigned char – для шиллингов, unsigned char – для пенсов (пенни). Реализовать сложение сумм, вычитание, деление сумм, деление суммы на дробное число, умножение на дробное число и операции сравнения. 1 фунт = 20 шиллингов, 1 шиллинг = 12 пенни.

Описание программы

Исходный код лежит в 3 файлах:

1. main.cpp: часть программы, отвечающая за взаимодействие с пользователем через консоль. В ней происходит инициализация объектов и вызов функций работы с ними;
2. BritishMoney.h: описание класса сумм британских денег BritishMoney;
3. BritishMoney.cpp: реализация класса BritishMoney.

Также используется файл CMakeLists.txt с конфигурацией CMake для автоматизации сборки программы.

Дневник отладки

Проблем не было.

Вывод

В данной лабораторной работе я продолжила знакомство с ООП в языке C++.

Впервые реализовала перегрузку операторов в C++, а также задала пользовательские литералы – константы класса BritishMoney.

И переопределила пользовательский класс BritishMoney, реализовав функции-операции над экземплярами класса в виде перегрузки операторов.

Как и в предыдущей лабораторной, в программе не поддерживаются отрицательные суммы, и потому в программу добавлены соотвествующие проверки

Исходный код

BritishMoney.h:

```
#pragma once
#include <iostream>

using namespace std;

class BritishMoney
{
private:
    unsigned long long pounds;
    unsigned char shillings;
    unsigned char pennies;

public:
    static const unsigned char penniesPerShilling = 12;
    static const unsigned char shillingsPerPound = 20;

    BritishMoney();
    BritishMoney(unsigned long long pounds, unsigned char shillings, unsigned
char pennies);

    friend BritishMoney BritishMoneyFromPennies(unsigned long long pennies);
    friend unsigned long long BritishMoneyToPennies(const BritishMoney& money);

    friend BritishMoney operator+ (const BritishMoney& money1, const
BritishMoney& money2);
    friend BritishMoney operator- (const BritishMoney& money1, const
BritishMoney& money2);
    friend BritishMoney operator* (const BritishMoney& money, const double
factor);
    friend BritishMoney operator* (const double factor, const BritishMoney&
money);
    friend BritishMoney operator/ (const BritishMoney& money, const double
factor);
    friend double operator/ (const BritishMoney& money1, const BritishMoney&
money2);
```

```
    friend bool operator== (const BritishMoney& money1, const BritishMoney&
money2);
    friend bool operator!= (const BritishMoney& money1, const BritishMoney&
money2);
    friend bool operator> (const BritishMoney& money1, const BritishMoney&
money2);
    friend bool operator>= (const BritishMoney& money1, const BritishMoney&
money2);
    friend bool operator< (const BritishMoney& money1, const BritishMoney&
money2);
    friend bool operator<= (const BritishMoney& money1, const BritishMoney&
money2);

    friend ostream& operator<< (ostream& stream, const BritishMoney& money);
    friend istream& operator>> (istream& stream, BritishMoney& money);
};

BritishMoney operator "" _pound(unsigned long long pounds);
BritishMoney operator "" _shilling(unsigned long long shillings);
BritishMoney operator "" _penny(unsigned long long pennies);
```

BritishMoney.cpp:

```
#include <stdexcept>
#include "BritishMoney.h"

BritishMoney::BritishMoney()
{
    this->pounds = 0;
    this->shillings = 0;
    this->pennies = 0;
}

BritishMoney::BritishMoney(unsigned long long pounds, unsigned char shillings,
unsigned char pennies)
{
    if (shillings >= shillingsPerPound || pennies >= penniesPerShilling)
        throw std::out_of_range("BritishMoney constructor: invalid values of
shillings or pennies");
    this->pounds = pounds;
    this->shillings = shillings;
    this->pennies = pennies;
}

BritishMoney BritishMoneyFromPennies(unsigned long long pennies)
{
    unsigned char pennies2 = pennies % BritishMoney::penniesPerShilling;
    unsigned long long shillings = pennies / BritishMoney::penniesPerShilling;
    unsigned char shillings2 = shillings % BritishMoney::shillingsPerPound;
    unsigned long long pounds = shillings / BritishMoney::shillingsPerPound;

    return BritishMoney(pounds, shillings2, pennies2);
}

unsigned long long BritishMoneyToPennies(const BritishMoney& money)
```

```
{
    unsigned long long pennies = (money.pounds *
BritishMoney::shillingsPerPound + money.shillings) *
BritishMoney::penniesPerShilling + money.pennies;
    return pennies;
}

BritishMoney operator+(const BritishMoney& money1, const BritishMoney& money2)
{
    unsigned long long pennies1 = BritishMoneyToPennies(money1);
    unsigned long long pennies2 = BritishMoneyToPennies(money2);
    unsigned long long pennies3 = pennies1 + pennies2;
    return BritishMoneyFromPennies(pennies3);
}

BritishMoney operator-(const BritishMoney& money1, const BritishMoney& money2)
{
    unsigned long long pennies1 = BritishMoneyToPennies(money1);
    unsigned long long pennies2 = BritishMoneyToPennies(money2);
    if (pennies1 < pennies2)
        throw std::out_of_range("BritishMoney operator-: money1 less than
money2");
    unsigned long long pennies3 = pennies1 - pennies2;
    return BritishMoneyFromPennies(pennies3);
}

BritishMoney operator*(const BritishMoney& money, const double factor)
{
    if (factor < 0.0)
        throw std::out_of_range("BritishMoney operator*: second parameter is
less than zero");
    unsigned long long pennies = BritishMoneyToPennies(money);
    unsigned long long pennies2 = (unsigned long long)(pennies * factor);
    return BritishMoneyFromPennies(pennies2);
}

BritishMoney operator*(const double factor, const BritishMoney& money)
{
    if (factor < 0.0)
        throw std::out_of_range("BritishMoney operator*: first parameter is
less than zero");
    return money * factor;
}

BritishMoney operator/(const BritishMoney& money, const double factor)
{
    if (factor == 0.0)
        throw std::out_of_range("BritishMoney operator/: second parameter is
zero or less");

    unsigned long long pennies = BritishMoneyToPennies(money);
    unsigned long long pennies2 = (unsigned long long)(pennies / factor);
    return BritishMoneyFromPennies(pennies2);
}

double operator/(const BritishMoney& money1, const BritishMoney& money2)
{
```

```
    unsigned long long pennies1 = BritishMoneyToPennies(money1);
    unsigned long long pennies2 = BritishMoneyToPennies(money2);
    if (pennies2 == 0)
        throw std::out_of_range("BritishMoney operator-: second parameter is
zero");
    double factor = (double)pennies1 / (double)pennies2;
    return factor;
}

bool operator==(const BritishMoney& money1, const BritishMoney& money2)
{
    unsigned long long pennies1 = BritishMoneyToPennies(money1);
    unsigned long long pennies2 = BritishMoneyToPennies(money2);
    bool result = pennies1 == pennies2;
    return result;
}

bool operator!=(const BritishMoney& money1, const BritishMoney& money2)
{
    unsigned long long pennies1 = BritishMoneyToPennies(money1);
    unsigned long long pennies2 = BritishMoneyToPennies(money2);
    bool result = pennies1 != pennies2;
    return result;
}

bool operator>(const BritishMoney& money1, const BritishMoney& money2)
{
    unsigned long long pennies1 = BritishMoneyToPennies(money1);
    unsigned long long pennies2 = BritishMoneyToPennies(money2);
    bool result = pennies1 > pennies2;
    return result;
}

bool operator>=(const BritishMoney& money1, const BritishMoney& money2)
{
    unsigned long long pennies1 = BritishMoneyToPennies(money1);
    unsigned long long pennies2 = BritishMoneyToPennies(money2);
    bool result = pennies1 >= pennies2;
    return result;
}

bool operator<(const BritishMoney& money1, const BritishMoney& money2)
{
    unsigned long long pennies1 = BritishMoneyToPennies(money1);
    unsigned long long pennies2 = BritishMoneyToPennies(money2);
    bool result = pennies1 < pennies2;
    return result;
}

bool operator<=(const BritishMoney& money1, const BritishMoney& money2)
{
    unsigned long long pennies1 = BritishMoneyToPennies(money1);
    unsigned long long pennies2 = BritishMoneyToPennies(money2);
    bool result = pennies1 <= pennies2;
    return result;
}
```

```
ostream& operator<<(ostream& stream, const BritishMoney& money)
{
    unsigned long long pounds = money.pounds;
    unsigned int shillings = money.shillings;
    unsigned int pennies = money.pennies;

    stream << "(" << pounds << "," << shillings << "," << pennies << ")";
    return stream;
}

istream& operator>>(istream& stream, BritishMoney& money)
{
    unsigned long long pounds;
    unsigned int shillings;
    unsigned int pennies;
    char leftBracket, rightBracket, comma1, comma2;

    stream >> leftBracket >> pounds >> comma1 >> shillings >> comma2 >> pennies
>> rightBracket;
    money.pounds = pounds;
    money.shillings = shillings;
    money.pennies = pennies;

    return stream;
}

BritishMoney operator "" _pound(unsigned long long pounds)
{
    unsigned long long shillings = pounds * BritishMoney::shillingsPerPound;
    unsigned long long pennies = shillings * BritishMoney::penniesPerShilling;
    return BritishMoneyFromPennies(pennies);
}

BritishMoney operator "" _shilling(unsigned long long shillings)
{
    unsigned long long pennies = shillings * BritishMoney::penniesPerShilling;
    return BritishMoneyFromPennies(pennies);
}

BritishMoney operator ""_penny(unsigned long long pennies)
{
    return BritishMoneyFromPennies(pennies);
}
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.21)
project(oop_exercise_02)

set(CMAKE_CXX_STANDARD 14)

include_directories(.)

add_executable(oop_exercise_02
        BritishMoney.cpp
```

```
BritishMoney.h
main.cpp)
```