**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 2**

Тема: Операторы, литералы на языке С++

Студент: Эссаулов Андрей

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. Постановка задачи

Создать класс BritishMoney для работы с денежными суммами в старой британской система. Сумма денег должна быть представлено 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.

Операции сложения, вычитания, умножения, деления, сравнения (на равенство, больше и меньше) должны быть выполнены в виде перегрузки операторов.

Необходимо реализовать пользовательский литерал для работы с константами типа BritishMoney.

1. Описание программы

Программа содержит класс BritishMoney, который имеет поля unsigned long long для фунтов и два unsigned char для шиллингов и пенни.

Также в классе определен конструктор для начальной инициализации переменных и метод WriteMoney, который выводит в консоль текущее значение всех полей класса.

Все действия над классом BritishMoney реализованы в виде перегрузки операторов, которые принимают 2 экземпляра класса(либо 1 экземпляр и 1 число) и возвращают результирующий экземпляр. В операциях суммы и вычитания реализована проверка на выход за границы размерности переменной. В операции деления реализована проверка деления на ноль. Операции сравнения переводят поля фунтов и шиллингов в пенни и сравнивают основываясь на сумме трех полей.

В программе реализована система литералов для класса BritishMoney. При помощи суффиксов \_f, \_s, \_p осуществляется возвращение экземпляра класса с фунтами, шиллингами или пенни соответсвенно.

Работа с программой осуществляется следующим образом:

Программа дважды запрашивает ввод 3 значений (фунты, шиллинги, пенни), после чего запрашивает ввод той операции, которую надо провести над введенными данными (коды операций выводятся в консоль). В итоге программа выводит результат операции и завершается.

1. Набор testcases

Тестам подаются значения 2 раза по 3 значения, которые интерпретируются как фунты, шиллинги и пенни. И в конце тесту подается код команды, которую он должен выполнить.

Коды операций:

1) + 2) - 3) / 4) / (float) 5) \* (float) 6) > 7) < 8) ==

**1-ый тест:**

Проверяем операцию сложения

2 6 17 // Первое число в формате фунты, шиллинги, пенни

4 8 20 // Второе число в формате фунты, шиллинги, пенни

1 // Код операции плюс

**2-ой тест:**

Проверяем операцию минус

0 2 10 // Первое число в формате фунты, шиллинги, пенни

0 1 12 // Второе число в формате фунты, шиллинги, пенни

2 // Код операции минус

**3-ий тест:**

Проверяем операцию деление

8 12 7 // Первое число в формате фунты, шиллинги, пенни

0 1 2 // Второе число в формате фунты, шиллинги, пенни

3 // Код операции деление

1. Результаты выполнения тестов.

1) Funts - 6, shilings - 14, pennys - 37

2) Funts - 0, shilings - 0, pennys - 10

3) Funts - 8, shilings - 12, pennys - 3

1. Листинг программы

// Лабораторная №2 Эссаулов Андрей M80-207Б-18

// Создать класс BritishMoney для работы с денежными суммами в старой британской система.

// Сумма денег должна быть представлено 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.

// Операции сложения, вычитания, умножения, деления, сравнения (на равенство, больше и меньше)

// должны быть выполнены в виде перегрузки операторов.

// Необходимо реализовать пользовательский литерал для работы с константами типа BritishMoney

#include <iostream>

class BritishMoney

{

public:

BritishMoney(unsigned long long f, unsigned char s, unsigned char p )

{

funts = f;

shilings = s;

pennys = p;

}

~BritishMoney()

{

}

void WriteMoney()

{

std::cout << "Funts - " << funts << ", shilings - " << (unsigned int)shilings << ", pennys - " << (unsigned int)pennys << std::endl;

}

unsigned long long funts;

unsigned char shilings;

unsigned char pennys;

};

BritishMoney operator+ (const BritishMoney a, const BritishMoney b)

{

BritishMoney result(0, 0, 0);

if (a.pennys + b.pennys > UINT8\_MAX)

{

unsigned int buf = a.pennys + b.pennys;

result.pennys += buf % 12;

result.shilings += buf / 12;

}

else {

result.pennys += a.pennys + b.pennys;

}

if (a.shilings + b.shilings > UINT8\_MAX)

{

unsigned int buf = a.shilings + b.shilings;

result.shilings += buf % 20;

result.funts += buf / 20;

}

else{

result.shilings += a.shilings + b.shilings;

}

result.funts += a.funts + b.funts;

return result;

}

BritishMoney operator- (const BritishMoney a, const BritishMoney b)

{

BritishMoney result(0, 0 ,0);

int p, s, f = 0;

p = a.pennys - b.pennys;

s = a.shilings - b.shilings;

f = a.funts - b.funts;

if (p < 0)

{

while (p < 0){

if (s > 0)

{

s--;

p += 12;

}

else if (f > 0)

{

f--;

s += 19;

p += 12;

}

else

{

break;

}

}

}

if (s < 0)

{

while (s < 0)

{

if (f > 0)

{

f--;

s += 12;

}

else if (p > 12)

{

p -= 12;

s++;

}

else

{

break;

}

}

}

if(f < 0)

{

while (f < 0)

{

if (s > 20)

{

s -= 20;

f++;

}

else if (p > (12 \* 20))

{

p -= (12\*20);

f++;

}

else

{

break;

}

}

}

result.pennys = p<0?0:p;

result.shilings = s<0?0:s;

result.funts = f<0?0:f;

return result;

}

BritishMoney operator/ (const BritishMoney a, const BritishMoney b)

{

BritishMoney result(0, 0, 0);

if (b.pennys != 0) result.pennys = a.pennys / b.pennys; else result.pennys = a.pennys;

if (b.shilings != 0) result.shilings = a.shilings / b.shilings; else result.shilings = a.shilings;

if (b.funts != 0) result.funts = a.funts / b.funts; else result.funts = a.funts;

return result;

}

BritishMoney operator/ (const BritishMoney a, const float b)

{

BritishMoney result(0,0,0);

if (b != 0)

{

result.pennys = a.pennys / b;

result.shilings = a.shilings / b;

result.funts = a.funts / b;

}

else

{

result.pennys = a.pennys;

result.shilings = a.shilings;

result.funts = a.funts;

}

return result;

}

BritishMoney operator\* (const BritishMoney a, const float b)

{

BritishMoney result(0,0,0);

result.pennys = a.pennys \* b;

result.shilings = a.shilings \* b;

result.funts = a.funts \* b;

return result;

}

bool operator== (const BritishMoney a, const BritishMoney b)

{

BritishMoney result(0,0,0);

unsigned long long first, second;

first = a.pennys + a.shilings \* 12 + a.funts \* 12 \* 20;

second = b.pennys + b.shilings \* 12 + b.funts \* 12 \* 20;

return (first==second?true:false);

}

bool operator> (const BritishMoney a, const BritishMoney b)

{

BritishMoney result(0,0,0);

unsigned long long first, second;

first = a.pennys + a.shilings \* 12 + a.funts \* 12 \* 20;

second = b.pennys + b.shilings \* 12 + b.funts \* 12 \* 20;

return (first>second?true:false);

}

bool operator< (const BritishMoney a, const BritishMoney b)

{

BritishMoney result(0,0,0);

unsigned long long first, second;

first = a.pennys + a.shilings \* 12 + a.funts \* 12 \* 20;

second = b.pennys + b.shilings \* 12 + b.funts \* 12 \* 20;

return (first<second?true:false);

}

bool operator>= (const BritishMoney a, const BritishMoney b)

{

BritishMoney result(0,0,0);

unsigned long long first, second;

first = a.pennys + a.shilings \* 12 + a.funts \* 12 \* 20;

second = b.pennys + b.shilings \* 12 + b.funts \* 12 \* 20;

return (first>=second?true:false);

}

bool operator<= (const BritishMoney a, const BritishMoney b)

{

BritishMoney result(0,0,0);

unsigned long long first, second;

first = a.pennys + a.shilings \* 12 + a.funts \* 12 \* 20;

second = b.pennys + b.shilings \* 12 + b.funts \* 12 \* 20;

return (first<=second?true:false);

}

BritishMoney operator "" \_f(unsigned long long funts)

{

return BritishMoney(funts, 0, 0);

};

BritishMoney operator "" \_s(unsigned long long shil)

{

return BritishMoney(0, shil, 0);

};

BritishMoney operator "" \_p(unsigned long long pen)

{

return BritishMoney(0, 0, pen);

};

int main(){

// 1 (+)

BritishMoney test11 = 2\_f + 6\_s + 17\_p;

BritishMoney test12 = 4\_f + 8\_s + 20\_p;

// 2 (-)

BritishMoney test21 = 0\_f + 2\_s + 10\_p;

BritishMoney test22 = 0\_f + 1\_s + 12\_p;

// 3 (/)

BritishMoney test31 = 8\_f + 12\_s + 7\_p;

BritishMoney test32 = 0\_f + 1\_s + 2\_p;

int tests = 0;

std::cout << "1) Start tests \n2) Start custom input" << std::endl;

std::cin >> tests;

if (tests == 1)

{

// First test

std::cout << "First test. Testing literals and operator plus \n funts shillings pennys" << std::endl;

std::cout << test11.funts << " " << (int)test11.shilings << " " << (int)test11.pennys << std::endl;

std::cout << test12.funts << " " << (int)test12.shilings << " " << (int)test12.pennys << std::endl;

BritishMoney test13 = test11 + test12;

std::cout << "Result: \n" << test13.funts << " " << (int)test13.shilings << " " << (int)test13.pennys << std::endl << std::endl;

//Second Test

std::cout << "Second test. Testing literals and operator minus \n funts shillings pennys" << std::endl;

std::cout << test21.funts << " " << (int)test21.shilings << " " << (int)test21.pennys << std::endl;

std::cout << test22.funts << " " << (int)test22.shilings << " " << (int)test22.pennys << std::endl;

BritishMoney test23 = test21 - test22;

std::cout << "Result: \n" << test23.funts << " " << (int)test23.shilings << " " << (int)test23.pennys << std::endl << std::endl;

//Third Test

std::cout << "Third test. Testing literals and operator divide \n funts shillings pennys" << std::endl;

std::cout << test31.funts << " " << (int)test31.shilings << " " << (int)test31.pennys << std::endl;

std::cout << test32.funts << " " << (int)test32.shilings << " " << (int)test32.pennys << std::endl;

BritishMoney test33 = test31 - test32;

std::cout << "Result: \n" << test33.funts << " " << (int)test33.shilings << " " << (int)test33.pennys << std::endl;

}

else if(tests == 2)

{

BritishMoney first(0, 0 , 0);

BritishMoney second(0, 0, 0);

int bufs, bufp;

std::cout << "Enter first bill\nfunts\t shilings\t pennys" << std::endl;

std::cin >> first.funts >> bufs >> bufp;

first.shilings = bufs;

first.pennys = bufp;

std::cout << "--------------------------------------------" << std::endl << std::endl;

std::cout << "Enter second bill\nfunts\t shilings\t pennys" << std::endl;

std::cin >> second.funts >> bufs >> bufp;

second.shilings = bufs;

second.pennys = bufp;

std::cout << "--------------------------------------------" << std::endl << std::endl;

std::cout << "Enter operation code: " << std::endl << "1) +" << std::endl << "2) -" << std::endl << "3) /"

<< std::endl << "4) / (float)" << std::endl << "5) \* (float)" << std::endl << "6) >"<< std::endl << "7) <"

<< std::endl << "8) ==" << std::endl;

int switch\_var;

std::cin >> switch\_var;

switch (switch\_var)

{

case 1:

(first + second).WriteMoney();

break;

case 2:

(first - second).WriteMoney();

break;

case 3:

(first / second).WriteMoney();

break;

case 4:

std::cout << "Enter divider" << std::endl;

int div;

std::cin >> div;

(first / div).WriteMoney();

break;

case 5:

std::cout << "Enter multiplier" << std::endl;

int mul;

std::cin >> mul;

(first \* mul).WriteMoney();

break;

case 6:

if (first > second) std::cout << "first > second - true" << std::endl;

else std::cout << "first > second - false" << std::endl;

break;

case 7:

if (first < second) std::cout << "first < second - true" << std::endl;

else std::cout << "first < second - false" << std::endl;

break;

case 8:

if (first == second) std::cout << "first == second - true" << std::endl;

else std::cout << "first == second - false" << std::endl;

break;

default:

std::cout << "Uncorrect enter" << std::endl;

break;

}

}

else {

std::cout << "wrong input" << std::endl;

}

return 1;

}

[Код на github](https://github.com/PromZona/oop_exercise_02)

1. Выводы

Литералы позволяют удобно представлять инстансы классов в качестве значений с суффиксом, что повышает читаемость кода и удобство использования класса.

Перегрузка операторов удобна для использования в классах, экземпляры которого взаимодействуют друг с другом. Данный функционал позволяет уменьшить количество строк кода необходимого для обозначения взаимодействия.

Данную программу можно улучшить изменив способ приема входных параметров на зацикленный, что позволит не выходя из программы вести непрерывную работу.

СПИСОК ЛИТЕРАТУРЫ

1. Справка по C++ [Электронный ресурс]. URL:<https://en.cppreference.com/w/cpp/language/user_literal>

Дата обращения: 21.09.2019

1. Документация по C++ [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/cpp/cpp/user-defined-literals-cpp?view=vs-2019>

Дата обращения 21.09.2019

1. Коллективный блог [Электронный ресурс]. URL: [https://habr.com/ru/post/132014](https://habr.com/ru/post/132014/)

Дата обращения: 20.09.2019