ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ВЫСШАЯ ШКОЛА ПЕЧАТИ И МЕДИАИНДУСТРИИ

*Институт Принтмедиа и информационных технологий*

*Кафедра Информатики и информационных технологий*

направление подготовки

09.03.02 «Информационные системы и технологии»,

ЛАБОРАТОРНАЯ РАБОТА № 7

Дисциплина: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Тема: Перегрузка операций

Выполнил(а): студент(ка) группы 181-722

Голиков Р.О.

(Фамилия И.О.)

Дата, подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_*

(Дата) (Подпись)

Проверил: Ктн Доцент Арсентьев Д.А.*\_\_\_\_\_\_\_\_\_\_*

(Фамилия И.О. степень, звание) (Оценка)

Дата, подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_*

(Дата) (Подпись)

Замечания: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва

2019

**Задание 1**

Добавьте в класс Distance из программы ENGLPLUS перегруженную

операцию -, которая вычисляет разность двух интервалов. Она должна позволять выполнение выражений типа dist3=distl-dist2;. Предполагаем, что эта операция никогда не будет использоваться для вычитания большего интервала из меньшего (так как отрицательного интервала быть не может).

**Код:**

#include <iostream>

using namespace std;

class Distance //English Distance class

{

private:

int feet;

float inches;

public: //constructor (no args)

Distance() : feet(0), inches(0.0)

{ } //constructor (two args)

Distance(int ft, float in) : feet(ft), inches(in)

{ }

void getdist() //get length from user

{

cout << "\nEnter feet: "; cin >> feet;

cout << "Enter inches: "; cin >> inches;

}

void showdist() //display distance

{

cout << feet << "\'-" << inches << '\"';

}

Distance operator + (Distance); //add two distances

Distance operator - (Distance); //subtract two distances

};

//add d2 to this distance

Distance Distance::operator + (Distance d2) //return the sum

{

int f = feet + d2.feet; //add the feet

float i = inches + d2.inches; //add the inches

if (i >= 12.0) //if total exceeds 12.0,

{ //then decrease inches

i -= 12.0; //by 12.0 and

f++; //increase feet by 1

} //return a temporary Distance

return Distance(f, i); //initialized to sum

}

//subtract d2 from this dist

Distance Distance::operator - (Distance d2) //return the diff

{

int f = feet - d2.feet; //subtract the feet

float i = inches - d2.inches; //subtract the inches

if (i < 0) //if inches less than 0,

{ //then increase inches

i += 12.0; //by 12.0 and

f--; //decrease feet by 1

} //return a temporary Distance

return Distance(f, i); //initialized to difference

}

int main()

{

Distance dist1, dist3; //define distances

dist1.getdist(); //get dist1 from user

Distance dist2(3, 6.25); //define, initialize dist2

dist3 = dist1 - dist2; //subtract

//display all lengths

cout << "\ndist1 = "; dist1.showdist();

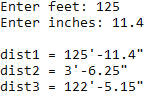
cout << "\ndist2 = "; dist2.showdist();

cout << "\ndist3 = "; dist3.showdist();

cout << endl;

return 0;

}

**Вывод:**  


**Задание 2**

Напишите программу, которая заменяет перегруженную операцию + на перегруженную операцию += в программе STRPLUS. Эта операция должна позволять записывать выражения типа:

si += s2

где s2 прибавляется (объединяется) к строке si, результат при этом остается в si. Операция должна также позволять использовать результат для других вычислений, например в выражениях типа:

s3 = si += s2

**Код:**

#include <iostream>

#include <cstring> //for strcpy(), strlen()

using namespace std;

#include <process.h> //for exit()

class String //user-defined string type

{

private:

enum { SZ = 80 }; //size of String objects

char str[SZ]; //holds a C-string

public:

String() //no-arg constructor

{

strcpy\_s(str, "");

}

String(const char s[]) //1-arg constructor

{

strcpy\_s(str, s);

}

void display() //display the String

{

cout << str;

}

String operator += (String ss) //add a String to this one

{ //result stays in this one

if (strlen(str) + strlen(ss.str) >= SZ)

{

cout << "\nString overflow"; exit(1);

}

strcat\_s(str, ss.str); //add the argument string

return String(str); //return temp String

}

};

int main()

{

String s1 = "Merry Christmas! "; //uses 1-arg ctor

String s2 = "Happy new year!"; //uses 1-arg ctor

String s3; //uses no-arg ctor

s3 = s1 += s2; //add s2 to s1, assign to s3

cout << "\ns1="; s1.display(); //display s1

cout << "\ns2="; s2.display(); //display s2

cout << "\ns3="; s3.display(); //display s3

cout << endl;

return 0;

}

**Вывод:**  


**Задание 3**

Модифицируйте класс time из упражнения 3 лабораторной работы №4 так, чтобы вместо метода add\_time() можно было использовать операцию + для складывания двух значений времени. Напишите программу для проверки класса.

**Код:**

#include <iostream>

using namespace std;

class time

{

private:

int hrs, mins, secs;

public:

time() : hrs(0), mins(0), secs(0) //no-arg constructor

{ } //3-arg constructor

time(int h, int m, int s) : hrs(h), mins(m), secs(s)

{ }

void display() //format 11:59:59

{

cout << hrs << ":" << mins << ":" << secs;

}

time operator + (time t2) //add two times

{

int s = secs + t2.secs; //add seconds

int m = mins + t2.mins; //add minutes

int h = hrs + t2.hrs; //add hours

if (s > 59) //if secs overflow,

{

s -= 60; m++;

} // carry a minute

if (m > 59) //if mins overflow,

{

m -= 60; h++;

} // carry an hour

return time(h, m, s); //return temp value

}

};

int main()

{

time time1(5, 59, 59); //create and initialze

time time2(4, 30, 30); // two times

time time3; //create another time

time3 = time1 + time2; //add two times

cout << "\ntime3 = "; time3.display(); //display result

cout << endl;

return 0;

}

**Вывод:**  


**Задание 4**

Создайте класс Int, основанный на упражнении 1 из лабораторной работы № 4. Перегрузите четыре целочисленных арифметических операции (+, -, \* и /) так, чтобы

их можно было использовать для операций с объектами класса Int. Если результат какой-либо из операций выходит за границы типа int (в 32- битной системе), имеющие значение от 2 14 7 483 648 до -2 147 483 648, то операция должна послать сообщение об ошибке и завершить программу. Такие тины данных полезны там, где ошибки могут быть вызваны арифметическим переполнением, которое недопустимо. Подсказка: для облегчения проверки переполнения выполняйте вычисления с использованием типа long double. Напишите программу для проверки этого класса.

**Код:**

#include <iostream>

using namespace std;

#include <process.h> //for exit()

class Int

{

private:

int i;

public:

Int() : i(0) //no-arg constructor

{ }

Int(int ii) : i(ii) //1-arg constructor

{ } // (int to Int)

void putInt() //display Int

{

cout << i;

}

void getInt() //read Int from kbd

{

cin >> i;

}

operator int() //conversion operator

{

return i;

} // (Int to int)

Int operator + (Int i2) //addition

{

return checkit(long double(i) + long double(i2));

}

Int operator - (Int i2) //subtraction

{

return checkit(long double(i) - long double(i2));

}

Int operator \* (Int i2) //multiplication

{

return checkit(long double(i)\*long double(i2));

}

Int operator / (Int i2) //division

{

return checkit(long double(i) / long double(i2));

}

Int checkit(long double answer) //check results

{

if (answer > 2147483647.0L || answer < -2147483647.0L)

{

cout << "\nOverflow Error\n"; exit(1);

}

return Int(int(answer));

}

};

int main()

{

Int alpha = 20;

Int beta = 7;

Int delta, gamma;

gamma = alpha + beta; //27

cout << "\ngamma="; gamma.putInt();

gamma = alpha - beta; //13

cout << "\ngamma="; gamma.putInt();

gamma = alpha \* beta; //140

cout << "\ngamma="; gamma.putInt();

gamma = alpha / beta; //2

cout << "\ngamma="; gamma.putInt();

delta = 2147483647;

gamma = delta + alpha; //overflow error

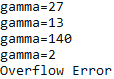
delta = -2147483647;

gamma = delta - alpha; //overflow error

cout << endl;

return 0;

}

**Вывод:**  


**Задание 5**

Пополните класс time, рассмотренный в упражнении 3, перегруженными операциями увеличения (++) и уменьшения (--), которые работают в обеих, префиксной и постфиксной, формах записи и возвращают значение. Дополните функцию main(), чтобы протестировать эти операции.

**Код:**

#include <iostream>

using namespace std;

class time

{

private:

int hrs, mins, secs;

public:

time() : hrs(0), mins(0), secs(0)

{ }

time(int h, int m, int s) : hrs(h), mins(m), secs(s)

{ }

void display()

{

cout << hrs << ":" << mins << ":" << secs;

}

time operator + (time t2)

{

int s = secs + t2.secs;

int m = mins + t2.mins;

int h = hrs + t2.hrs;

if (s > 59)

{

s -= 60; m++;

}

if (m > 59)

{

m -= 60; h++;

}

return time(h, m, s);

}

time operator++()

{

if (++secs > 59)

{

secs -= 60;

mins++;

}

if (mins > 59)

{

mins -= 60;

hrs++;

}

return \*this;

}

time operator--()

{

if (--secs < 0)

{

secs += 60;

mins--;

}

if (mins < 0)

{

mins += 60;

hrs--;

}

return \*this;

}

time operator++(int)

{

time temp(hrs, mins, secs);

++(\*this);

return temp;

}

time operator--(int)

{

time temp(hrs, mins, secs);

--(\*this);

return temp;

}

};

int main()

{

time time1(5, 59, 59);

cout << "\ntime: "; time1.display();

cout << "\ntime++: "; time1++.display();

cout << "\ntime: "; time1.display();

cout << "\n++time: "; (++time1).display();

cout << endl;

cout << "\ntime: "; time1.display();

cout << "\ntime--: "; time1--.display();

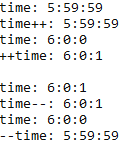
cout << "\ntime: "; time1.display();

cout << "\n--time: "; (--time1).display();

cout << endl;

return 0;

}

**Вывод:**  


**Задание 6**

Добавьте в класс time из упражнения 5 возможность вычитать значения времени, используя перегруженную операцию -, и умножать эти значения, используя тип float и перегруженную операцию \*.

**Код:**

#include <iostream>

using namespace std;

class time

{

private:

int hrs, mins, secs;

public:

time() : hrs(0), mins(0), secs(0)

{ }

time(int h, int m, int s) : hrs(h), mins(m), secs(s)

{ }

void display()

{

cout << hrs << ":" << mins << ":" << secs;

}

time operator+(time t2)

{

int s = secs + t2.secs;

int m = mins + t2.mins;

int h = hrs + t2.hrs;

if (s > 59)

{

s -= 60; m++;

}

if (m > 59)

{

m -= 60; h++;

}

return time(h, m, s);

}

time operator-(time t2)

{

int s = secs - t2.secs;

int m = mins - t2.mins;

int h = hrs - t2.hrs;

if (s < 0)

{

s += 60; m--;

}

if (m < 0)

{

m += 60; h--;

}

return h < 0 ? time() : time(h, m, s);

}

time operator\*(float f)

{

int s = secs \* f;

int m = mins \* f;

int h = hrs \* f;

if (s > 59)

{

s %= 60; m += s / 60;

}

if (m > 59)

{

m %= 60; h += m / 60;

}

return time(h, m, s);

}

time operator++()

{

if (++secs > 59)

{

secs -= 60;

mins++;

}

if (mins > 59)

{

mins -= 60;

hrs++;

}

return \*this;

}

time operator--()

{

if (--secs < 0)

{

secs += 60;

mins--;

}

if (mins < 0)

{

mins += 60;

hrs--;

}

return \*this;

}

time operator++(int)

{

time temp(hrs, mins, secs);

++(\*this);

return temp;

}

time operator--(int)

{

time temp(hrs, mins, secs);

--(\*this);

return temp;

}

};

int main()

{

time time1(5, 59, 59);

time time2(4, 30, 30);

cout << "\ntime1: "; time1.display();

cout << "\ntime2: "; time2.display();

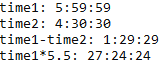
cout << "\ntime1-time2: "; (time1-time2).display();

cout << "\ntime1\*5.5: "; (time1 \* 5.5).display();

cout << endl;

return 0;

}

**Вывод:**  


**Задание 7**

Модифицируйте класс fraction в четырехфункциональном дробном калькуляторе из упражнения 11 лабораторной работы №4 так, чтобы он использовал перегруженные операции сложения, вычитания, умножения и деления. (Вспомните правила арифметики с дробями в упражнении 12 лабораторной работы №2 «Циклы и ветвления».) Также перегрузите операции сравнения == и != и используйте их для выхода из цикла, когда пользователь вводит 0/1, 0 и 1 значения двух частей дроби. Вы можете модифицировать и функцию lowterms() так, чтобы она возвращала значение ее аргумента, уменьшенное до несократимой дроби. Это будет полезным в арифметических функциях, которые могут быть выполнены сразу после получения ответа.

**Код:**

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

class fraction

{

private:

int \_dividend;

int \_divider;

public:

fraction()

{

}

fraction(int dividend, int divider) : \_dividend(dividend), \_divider(divider)

{

\*this = lowterms(\*this);

}

void parse(string str)

{

\_dividend = 0;

\_divider = 0;

int i, j;

int len = str.length();

for (i = 0; i < len && str[i] != '/'; i++);

for (j = i - 1; j >= 0; j--)

\_dividend += (str[j] - '0') \* pow(10, i - j - 1);

for (j = len - 1; j > i; j--)

\_divider += (str[j] - '0') \* pow(10, len - j - 1);

}

fraction operator+(fraction f)

{

fraction resf;

resf.\_dividend = \_dividend \* f.\_divider + \_divider \* f.\_dividend;

resf.\_divider = \_divider \* f.\_divider;

return lowterms(resf);

}

fraction operator-(fraction f)

{

fraction resf;

resf.\_dividend = \_dividend \* f.\_divider - \_divider \* f.\_dividend;

resf.\_divider = \_divider \* f.\_divider;

return lowterms(resf);

}

fraction operator\*(fraction f)

{

fraction resf;

resf.\_dividend = \_dividend \* f.\_dividend;

resf.\_divider = \_divider \* f.\_divider;

return lowterms(resf);

}

fraction operator/(fraction f)

{

fraction resf;

resf.\_dividend = \_dividend \* f.\_divider;

resf.\_divider = \_divider \* f.\_dividend;

return lowterms(resf);

}

bool operator==(fraction f)

{

return \_dividend == f.\_dividend && \_divider == f.\_divider;

}

bool operator!=(fraction f)

{

return \_dividend != f.\_dividend || \_divider != f.\_divider;

}

void display()

{

cout << \_dividend << '/' << \_divider << endl;

}

fraction lowterms(fraction f)

{

long tnum, tden, temp, gcd;

tnum = labs(f.\_dividend);

tden = labs(f.\_divider);

if (tden == 0) {

return f;

}

else if (tnum == 0) {

f.\_divider = 1;

return f;

}

while (tnum != 0) {

if (tnum < tden) {

temp = tnum;

tnum = tden;

tden = temp;

}

tnum -= tden;

}

gcd = tden;

f.\_dividend /= gcd;

f.\_divider /= gcd;

return f;

}

};

int main()

{

setlocale(LC\_ALL, "");

fraction f1, f2, fres;

char oper, ch;

string str1, str2;

do {

cout << "\nВведите выражение с дробями: " << endl;

cin >> str1 >> oper >> str2;

f1.parse(str1);

f2.parse(str2);

if (f1 == fraction(0, 1) || f1 == fraction(1, 1) || f1 == fraction(0, 0)) break;

if (f2 == fraction(0, 1) || f2 == fraction(1, 1) || f2 == fraction(0, 0)) break;

switch (oper)

{

case '+':

fres = f1 + f2;

break;

case '-':

fres = f1 - f2;

break;

case '\*':

fres = f1 \* f2;

break;

case '/':

fres = f1 / f2;

break;

}

cout << "Answer = ";

fres.display();

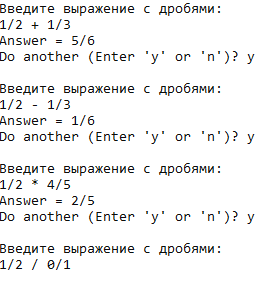
cout << "Do another (Enter 'y' or 'n')? ";

cin >> ch;

} while (ch != 'n');

return 0;

}

**Вывод:**  


**Задание 8**

Модифицируйте класс bМоnеу из упражнения 12 лабораторной работы №5 «Массивы и строки», включив арифметические операции, выполненные с помощью перегруженных операций:

bМоnеу = bМоnеу + bМоnеу

bМоnеу = bМоnеу - bМоnеу

bМоnеу = bМоnеу \* long double (цена за единицу времени, затраченного на изделие)

long double = bМоnеу / bМоnеу (общая цена, деленная на цену за изделие)

bМоnеу = bМоnеу / long double (общая цена, деленная на количество изделий)

Заметим, что операция / перегружена дважды. Компилятор может различить оба варианта, так как их аргументы разные. Помним, что легче выполнять арифметические операции с объектами класса bМоnеу, выполняя те же операции с его long double данными.

Убедитесь, что программа main() запросит ввод пользователем двух денежных строк и числа с плавающей точкой. Затем она выполнит все пять операций и выведет результаты. Это должно происходить в цикле, так, чтобы пользователь мог ввести еще числа, если это понадобится.

Некоторые операции с деньгами не имеют смысла: bМоnеу\*bМоnеу не представляет ничего реального, так как нет такой вещи, как денежный квадрат; вы не можете прибавить bМоnеу к long double (что же будет, если рубли сложить с изделиями?). Чтобы сделать это невозможным, скомпилируйте такие неправильные операции, не включая операции преобразования для bМоnеу в long double или long double в bМоnеу. Если вы это сделаете и запишете затем выражение типа:

bmon2 = bmonl + widgets; //это не имеет смысла

то компилятор будет автоматически преобразовывать widgets в ЬМопеу и выполнять сложение. Без них компилятор будет отмечать такие преобразования как ошибки, что позволит легче найти концептуальные ошибки. Также сделайте конструкторы преобразований явными.

**Код:**

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

class money

{

private:

long double \_sum;

public:

money() {}

explicit money(long double sum) : \_sum(sum) {}

long double mstold(string sum)

{

string res = "";

for (int i = 0; i < sum.length(); i++)

{

if (isdigit(sum[i]) || sum[i] == ',')

res += sum[i];

}

return stold(res.c\_str());

}

string ldtoms(long double sum)

{

if (sum > 9999999999999990.00) return "";

ostringstream strs;

strs << fixed << sum;

string ustr = strs.str();

string newstr = "$";

int i, j;

for (i = 0; i < ustr.length() && (ustr[i] == '0' || ustr[i] == '.'); i++);

if (i > 1 && ustr[i - 1] == '.') i -= 2;

if (i == ustr.length()) return "$0,00";

for (j = ustr.length() - 1; j >= 0 && ustr[j] != '.'; j--);

j += 2;

for (; i <= j; i++)

{

if (i != 0 && i < j - 2 && (j - i + 1) % 3 == 0)

newstr += '.';

newstr += ustr[i] == '.' ? ',' : ustr[i];

}

return newstr;

}

void getmoney()

{

string str;

cout << "Введите денежную сумму:\n";

cin >> str;

\_sum = mstold(str);

}

void putmoney()

{

cout << "Денежная сумма равна: " << ldtoms(\_sum) << endl;

}

money operator+(money m)

{

return money(\_sum + m.\_sum);

}

money operator-(money m)

{

return money(\_sum - m.\_sum);

}

money operator\*(long double ld)

{

return money(\_sum \* ld);

}

long double operator/(money m)

{

return \_sum / m.\_sum;

}

money operator/(long double ld)

{

return money(\_sum / ld);

}

};

int main()

{

setlocale(LC\_ALL, "");

money m1, m2(4512.34L);

long double s = 42.5L;

char ch;

do

{

m1.getmoney();

cout << "m1 + m2: ";

(m1 + m2).putmoney();

cout << "m1 - m2: ";

(m1 - m2).putmoney();

cout << "m1 \* s: ";

(m1 \* s).putmoney();

cout << "m1 / m2: ";

cout << (m1 / m2) << endl;

cout << "m1 / s: ";

(m1 / s).putmoney();

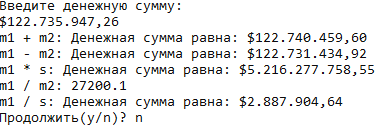
cout << "Продолжить(y/n)? ";

cin >> ch;

} while (ch != 'n');

return 0;

}

**Вывод:**  


**Задание 9**

Дополните класс safearray из программы ARR0VER3 так, чтобы пользователь мог определять и верхнюю, и нижнюю границы массива (например, индексы, начинающиеся с 100 и заканчивающиеся 200). Имеем перегруженную операцию доступа к членам массива, проверяющую индексы каждый раз, когда к массиву нужен доступ, для проверки того, что мы не вышли за пределы массива. Вам понадобится конструктор с двумя аргументами, который определяет верхнюю и нижнюю границы. Так как мы еще не изучили, как выделять память динамически, то данные класса все еще будут размещаться в массиве, состоящем из 100 элементов, но вообще вы можете преобразовывать индексы массива safearray в индексы реального массива целых чисел произвольным образом. Например, если пользователь определил диапазон от 100 до 175, то вы можете преобразовать его в диапазон от агг[0] до агг[75].

**Код:**

#include <iostream>

using namespace std;

const int LIMIT = 100;

class safearray

{

private:

int \_arr[LIMIT];

int \_loffset;

int \_uoffset;

public:

safearray() : \_loffset(0), \_uoffset(LIMIT)

{}

safearray(int loffset, int uoffset) : \_loffset(loffset), \_uoffset(uoffset)

{}

int& operator[](int n)

{

if (n - \_loffset < 0 || n >= \_uoffset)

{

cout << "\nIndex out of bounds"; exit(1);

}

return \_arr[n - \_loffset];

}

};

int main()

{

safearray sa1(100, 115);

for (int j = 100; j < 115; j++)

sa1[j] = j \* 10;

for (int j = 100; j < 115; j++)

{

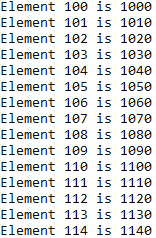
int temp = sa1[j];

cout << "Element " << j << " is " << temp << endl;

}

return 0;

}

**Вывод:**  


**Задание 10**

Создайте класс Polar, который предназначен для хранения полярных координат (радиуса и угла). Перегрузите операцию + для выполнения сложения для объектов класса Polar. Сложение двух объектов выполняется путем сложения координат X объектов, а затем координат Y. Результат будет координатами новой точки. Таким образом, вам нужно будет преобразовать полярные координаты к прямоугольным, сложить их, а затем обратно преобразовать прямоугольные координаты результата к полярным.

**Код:**

#include <cmath>

#include <iostream>

using namespace std;

const float Pi = 3.14159;

class Polar

{

private:

float \_radius;

float \_angle;

public:

Polar() : \_radius(0), \_angle(0) {}

Polar(float radius, float angle) : \_radius(radius), \_angle(angle) {}

Polar operator+(Polar p)

{

float x = \_radius \* cos(\_angle) + p.\_radius \* cos(p.\_angle);

float y = \_radius \* sin(\_angle) + p.\_radius \* sin(p.\_angle);

float tan = y / x;

return Polar(sqrt(x\*x + y\*y), tan < 0 ? Pi + atan(tan) : atan(tan));

}

void display()

{

cout << "Радиус: " << \_radius << ", угол: " << \_angle;

}

};

int main()

{

setlocale(LC\_ALL, "");

Polar p1(10, Pi), p2(5, Pi/2);

cout << "\np1 ";

p1.display();

cout << "\np2 ";

p2.display();

cout << "\nСумма ";

(p1 + p2).display();

return 0;

}

**Вывод:**  


**Задание 11**

Помните структуру sterling? Мы встречались с ней в упражнении 10 лабораторной работы №1 «Основы программирования на С++», в упражнении 11 лабораторной работы №3 и в других местах. Преобразуйте ее в класс, имеющий переменные для фунтов (типа long), шиллингов (типа int) и пенсов (типа int). Создайте в классе следующие функции:

конструктор без аргументов;

конструктор с одним аргументом типа double (для преобразования от десятичных фунтов);

конструктор с тремя аргументами: фунтами, шиллингами и пенсами;

метод getSterLing() для получения от пользователя значений количества фунтов, шиллингов и пенсов в формате £9.19.11;

метод putSterling() для вывода значений количества фунтов, шиллингов и пенсов в формате £9.19.11;

метод для сложения (sterling + sterling), используя перегруженную операцию +;

метод вычитания (sterling - sterling), используя перегруженную операцию -;

метод умножения (sterling \* double), используя перегруженную операцию \*;

метод деления (sterling / sterling), используя перегруженную операцию /;

метод деления (sterling / double), используя перегруженную операцию /;

операцию double (для преобразования к типу double)

Выполнять вычисления вы можете, например, складывая отдельно данные объекта: сложить сначала пенсы, затем шиллинги и т. д. Однако легче использовать операцию преобразования для преобразования объекта класса sterling к типу double, выполнить вычисления с типами double, а затем преобразовать обратно к типу sterling.

**Код:**

#include <iostream>

using namespace std;

const int PoundsToOldShilling = 20;

const int OldShillingToOldPennies = 12;

class Sterling

{

private:

long \_pounds;

int \_shillings;

int \_pence;

public:

Sterling() : \_pounds(0), \_shillings(0), \_pence(0) {}

Sterling(long pounds, int shillings, int pence) : \_pounds(pounds), \_shillings(shillings), \_pence(pence)

{

\*this = normalize(\*this);

}

Sterling(double newPounds)

{

\_pounds = (long)newPounds;

newPounds = (newPounds - \_pounds) \* PoundsToOldShilling;

\_shillings = (int)newPounds;

newPounds = (newPounds - \_shillings) \* OldShillingToOldPennies;

\_pence = (int)newPounds;

}

Sterling operator+(Sterling s)

{

return Sterling((double)\*this + (double)s);

}

Sterling operator-(Sterling s)

{

return Sterling((double)\*this - (double)s);

}

Sterling operator\*(double d)

{

return Sterling((double)\*this \* d);

}

double operator/(Sterling s)

{

return (double)\*this / (double)s;

}

Sterling operator/(double d)

{

return Sterling((double)\*this \* d);

}

operator double()

{

return \_pounds + (\_shillings + \_pence / (double)OldShillingToOldPennies) / (double)PoundsToOldShilling;

}

void get()

{

char ch;

cin >> ch >> \_pounds >> ch >> \_shillings >> ch >> \_pence;

}

void put()

{

cout << '£' << \_pounds << '.' << \_shillings << '.' << \_pence;

}

Sterling normalize(Sterling s)

{

s.\_shillings += s.\_pence / OldShillingToOldPennies;

s.\_pounds += s.\_shillings / PoundsToOldShilling;

s.\_pence %= OldShillingToOldPennies;

s.\_shillings %= PoundsToOldShilling;

return s;

}

};

int main()

{

setlocale(LC\_ALL, "");

Sterling s1(46, 15, 11), s2;

double d = 4.4;

cout << "Введите сумму: ";

s2.get();

cout << "\ns1+s2 ";

(s1 + s2).put();

cout << "\ns1-s2 ";

(s1 - s2).put();

cout << "\ns1\*d ";

(s1 \* d).put();

cout << "\ns1/s2 ";

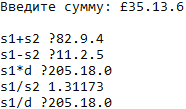
cout << (s1 / s2);

cout << "\ns1/d ";

(s1 / d).put();

return 0;

}

**Вывод:**  


**Задание 12**

Напишите программу, объединяющую в себе классы bМоnеу из упражнения 8 и sterling из упражнения 11. Напишите операцию преобразования для преобразования между классами bМоnеу и sterling, предполагая, что один фунт (£1.0.0) равен пятидесяти долларам ($50.00). Это приблизительный курс обмена для XIX века, когда Британская империя еще использовала меру фунты-шиллинги-пенсы. Напишите программу main(), которая позволит пользователю вводить суммы в каждой из валют и преобразовывать их в другую валюту с выводом результата. Минимизируйте количество изменений в существующих классах bМоnеу и sterling.

**Код:**

#include <iostream>

#include "Sterling.h"

#include "money.h"

using namespace std;

//Sterling.cpp

//Sterling::operator money()

//{

// return money((double)\*this \* 50.0);

//}

//money.cpp

//money::operator Sterling()

//{

// return Sterling(\_sum / 50);

//}

int main()

{

setlocale(LC\_ALL, "");

money m;

Sterling s;

char ch;

do

{

cout << "Какую валюту вы хотите ввести(s/d)?";

cin >> ch;

switch (ch)

{

case 's':

s.get();

((money)s).putmoney();

break;

case 'd':

m.getmoney();

((Sterling)m).put();

break;

default: cout << "Ошибка символа\n";

break;

}

cout << "\nПродолжить(y/n)?";

cin >> ch;

} while (ch != 'n');

return 0;

}

**Вывод:**  
