Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

Лабораторная работа №3

Перегрузка операций

Выполнил

Студент группы РИС-22-2б

Вековшинин Д. А.

Проверила

Доцент кафедры ИТАС

Полякова О. А.

Пермь 2023

Постановка задачи

1. Определить пользовательский класс.

2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.

3. Определить в классе деструктор.

4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).

5. Перегрузить операцию присваивания.

6. Перегрузить операции ввода и вывода объектов с помощью потоков.

7. Перегрузить операции, указанные в варианте.

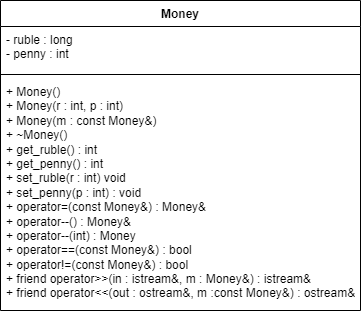
8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

Вариант 10: Создать класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long для рублей и типа int для копеек. Дробная часть числа при выводе на экран должна быть отделена от целой части запятой. Реализовать:

- операции сравнения (==, !=).

- вычитание копеек (--) (постфиксная и префиксная формы).

UML-диаграмма



Определение компонентных функций

Money() { ruble = 0; penny = 0; };

Money(int r, int p) { ruble = r; penny = p; }

Money(const Money& m) { ruble = m.ruble; penny = m.penny; }

~Money() {}

int get\_ruble() { return ruble; }

int get\_penny() { return penny; }

void set\_ruble(int r) { ruble = r; }

void set\_penny(int p) { penny = p; }

//перегрузка операции присваивания

Money& Money::operator=(const Money& m)

{

//проверка на самоприсваивание

if (&m == this) return \*this;

ruble = m.ruble;

penny = m.penny;

return \*this;

}

//перегрузка префиксной операции декремент

Money& Money::operator--()

{

int temp = ruble \* 100 + penny;

temp--;

ruble = temp / 100;

penny = temp % 100;

return \*this;

}

//перегрузка постфиксной операции декремент

Money Money::operator --(int)

{

int temp = ruble \* 100 + penny;

temp--;

Money t(ruble, penny);

ruble = temp / 100;

penny = temp % 100;

return t;

}

//перегрузка бинарной операции сравнения

bool Money::operator==(const Money& t)

{

int temp1 = ruble \* 100 + penny;

int temp2 = t.ruble \* 100 + t.penny;

bool compare = (temp1 == temp2);

return compare;

}

bool Money::operator!=(const Money& t)

{

int temp1 = ruble \* 100 + penny;

int temp2 = t.ruble \* 100 + t.penny;

bool compare = (temp1 != temp2);

return compare;

}

Определение глобальных функций

//перегрузка глобальной функции-операции ввода

istream& operator>>(istream& in, Money& t)

{

cout << "rubles? "; in >> t.ruble;

cout << "pennies? "; in >> t.penny;

return in;

}

//перегрузка глобальной функции-операции вывода

ostream& operator<<(ostream& out, const Money& t)

{

return (out << t.ruble << "," << t.penny);

}

Функция main()

void main()

{

Money a;//конструктор без параметров

Money b; //конструктор без параметров

bool compare;

cin >> a;//ввод переменной

cin >> b;//ввод переменной

--a;//префиксная операция декремент

cout << a << endl;//вывод переменной

b--;//постфиксная операция декремент

cout << b << endl;//вывод переменной

compare = a == b;//сравнение

cout << "a=" << a << endl; //вывод переменной

cout << "b=" << b << endl; //вывод переменной

cout << "c=" << compare << endl; //вывод переменной

compare = a != b;//сравнение

cout << "a=" << a << endl; //вывод переменной

cout << "b=" << b << endl; //вывод переменной

cout << "c=" << compare << endl; //вывод переменной

a = b; //операция присваивания

cout << "a=" << a << endl; //вывод переменной

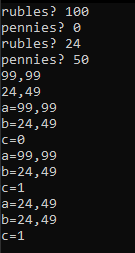
cout << "b=" << b << endl; //вывод переменной

compare = a == b;//сравнение

cout << "c=" << compare << endl; //вывод переменной

}

Результаты работы программы



Ответы на контрольные вопросы

1. Для чего используются дружественные функции и классы?

Для доступа к скрытым полям класса

ostream& operator<<(ostream& out, const Money& t)

{

return (out << t.ruble << "," << t.penny);

}

1. Сформулировать правила описания и особенности дружественных функций.

1) Дружественная функция объявляется внутри класса, к элементам которого

ей нужен доступ, с ключевым словом friend. В качестве параметра ей должен

передаваться объект или ссылка на объект класса, поскольку указатель this

ей не передается.

2) Дружественная функция может быть обычной функцией или методом другого ранее

определенного класса. На нее не распространяется действие спецификаторов доступа, место

размещения ее объявления в классе безразлично.

3) Одна функция может быть дружественной сразу нескольким классам.

Пример

class student;//предварительное описание класса

class teacher

{

…

void teach(student &S);

….

};

class student

{

...

friend void teacher::teach(student&);//дружественная функция,

//имеет доступ к элементам класса student

…

};

1. Каким образом можно перегрузить унарные операции?

Унарную операцию можно перегрузить:

- Как компонентную функцию класса

- Как внешнюю (глобальную) функцию

ostream& operator<<(ostream& out, const Money& t)

{

return (out << t.ruble << "," << t.penny);

}

1. Сколько операндов должна иметь унарная функция-операция, определяемая внутри класса?

Унарная функция-операция, определяемая внутри класса, должна быть представлена с помощью нестатического метода без параметров, при этом операндом является вызвавший ее объект, например:

class Person

{

string name;

int age;

public:

Person(string, int);//конструктор

.....

//компонентная функция

Person& operator ++() //префиксная операция

{

++age;

return \*this; //указатель на объект, вызвавший метод

}

};

//в основной функции

Person p1(”Ivanov”,20);

++p1;

p1.Show();

1. Сколько операндов должна иметь унарная функция-операция, определяемая вне класса?

Если функция определяется вне класса, она должна иметь один параметр типа класса:

class Person

{

string name;

int age;

public:

Person( string, int);//конструктор

…..

//внешняя дружественная функция

friend Person & operator ++(Person&) ;

};

Person & operator ++(Person& p) //префиксная операция

{

++p.age;

return p;

}

//в основной функции

Person p1(”Ivanov”,20);

++p1;

p1.Show();

1. Сколько операндов должна иметь бинарная функция-операция, определяемая внутри класса?

Бинарная функция-операция, определяемая внутри класса, должна быть представлена с помощью нестатического метода с параметрами, при этом вызвавший ее объект считается первым операндом:

class Person

{

string name;

int age;

public:

Person( string, int);//конструктор

…..

//компонентная функция

Person & operator +(int x)

{

age+=x;

return \*this;

}

};

//в основной функции

Person p1(”Ivanov”,20);

p1+2;

p1.Show();

1. Сколько операндов должна иметь бинарная функция-операция, определяемая вне класса?

Если функция определяется вне класса, она должна иметь два

параметра типа класса:

class Person

{

string name;

int age;

public:

Person(string, int);//конструктор

…..

//внешняя дружественная функция

friend Person & operator +(Person& p, int x) ;

};

Person & operator +(Person &p, int x)

{

p.age+=x;

return p;

}

//в основной функции

Person p1(“Ivanov”,20);

p1+2;

p1.Show();

1. Чем отличается перегрузка префиксных и постфиксных унарных операций?

Операции постфиксного инкремента и декремента должны иметь первый параметр типа int. Он используется только для того, чтобы отличить их от префиксной формы.

//перегрузка префиксной операции декремент

Money& Money::operator--()

//перегрузка постфиксной операции декремент

Money Money::operator --(int)

1. Каким образом можно перегрузить операцию присваивания?

Чтобы сохранить семантику присваивания, операция-функция должна возвращать ссылку на объект, для которого она вызвана, и принимать в качестве параметра единственный аргумент — ссылку на присваиваемый объект.//перегрузка операции присваивания

Money& Money::operator=(const Money& m)

{

//проверка на самоприсваивание

if (&m == this) return \*this;

ruble = m.ruble;

penny = m.penny;

return \*this;

}

1. Что должна возвращать операция присваивания?

Ссылку на объект, для которого она вызвана

//перегрузка операции присваивания

Money& Money::operator=(const Money& m)

{

//проверка на самоприсваивание

if (&m == this) return \*this;

ruble = m.ruble;

penny = m.penny;

return \*this;

}

1. Каким образом можно перегрузить операции ввода-вывода?

Операции ввода-вывода operator>> и operator<< всегда реализуются как внешние дружественные функции, т. к. левым операндом этих операций являются потоки. Для класса Person соответствующие операции могут выглядеть следующим образом:

class Person

{

string name;

int age;

public:

Person(string, int);//конструктор

…..

//дружественная глобальная функция

friend istream& operator>>(istream&in, Person&p);

friend ostream& operator<<(ostream&out, const Person&p);

};

……

istream&operator>>(istream&in, Person &p)

{

cout<<"name?"; in>>p.name;

cout<<"age?"; in>>p.age;

return in;

}

ostream&operator<<(ostream&out, const Person&p)

{

return (out<<p.name<<","<<p.age);

}

1. В программе описан класс

class Student

{

…

Student& operator++();

….

};

и определен объект этого класса

Student s;

Выполняется операция

++s;

Каким образом, компилятор будет воспринимать вызов функции-операции?

Произойдёт ошибка, так как нет доступа к скрытому полю Student& operator++();



1. В программе описан класс

class Student

{…

friend Student& operator ++( Student&);

….

};

и определен объект этого класса

Student s;

Выполняется операция

++s;

Каким образом, компилятор будет воспринимать вызов функции-операции?

Компилятор не найдёт проблем.



1. В программе описан класс

class Student

{

…

bool operator<(Student &P);

….

};

и определены объекты этого класса

Student a,b;

Выполняется операция

cout<<a<b;

Каким образом, компилятор будет воспринимать вызов функции-операции?

Произойдёт ошибка, так как нет доступа к скрытому полю bool operator<(Student& P);



1. В программе описан класс

class Student

{

…

friend bool operator >(const Person&, Person&)

….

};

и определены объекты этого класса

Student a,b;

Выполняется операция

cout<<a>b;

Каким образом, компилятор будет воспринимать вызов функции-операции?

Произойдёт ошибка, так как в качестве параметра функции friend bool operator >() должны передаваться объекты или ссылки на объекты класса Student, а не Person

