**Министерство образования и науки Российской Федерации**

**Пермское федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский Национальный Исследовательский Политехнический Университет»**

**Электротехнический факультет**

**Кафедра «Информационные технологии и автоматизированные системы»**

**ОТЧЁТ**

По лабораторной работе №18.3 на тему

«Перегрузка операторов»

Вариант №11

Выполнил студент группы ИВТ-20-2б

Сабуров Павел Алексеевич

Проверил доцент кафедры ИТАС

Полякова О.А.

Пермь 2021

**Цель работы** –научиться перегружать операторы классов в C++.

**Постановка задачи**

Задача – реализовать на языке программирования C++ программу, демонстрирующую работу перегруженных операторов классов.

**Исходные данные для варианта №11:**

* Операции сравнения:
  + Больше;
  + Меньше;
* Инкременты
  + Префиксный;
  + Постфиксный;

Класс «деньги»:

1. Копейки;
2. Рубли;

**Анализ задачи**

Для решения задачи были использованы следующие средства:

1. Язык программирования C++ (Microsoft Visual C++)
2. Текстовый редактор Microsoft Visual Studio Code

Объявление класса «Деньги» в заголовочном файле:

Деньги состоят из двух чисел:

1. Количество рублей – целой части валюты;
2. Количество копеек – дробной части валюты;

#pragma once

class Money

{

private:

long \_dollars;

int \_cents;

public:

int Dollars() { return \_dollars; }

void SetDollars(long dollars);

int Cents() { return \_cents; }

void SetCents(int cents);

void PrintMoney();

friend bool operator > (Money& first, Money& second);

friend bool operator < (Money& first, Money& second);

Money& operator++ ();

Money operator++ (int);

Money();

Money(long dollars, int cents);

Money(Money& parentMoney);

~Money();

};

Поскольку количество копеек не может быть больше 100, закладывается соответствующая проверка на значение в методе SetCents.

Конструкторы класса «Деньги»:

Money::Money()

{

\_dollars = 0;

\_cents = 0;

}

Money::Money(long dollars, int cents)

{

SetDollars(dollars);

SetCents(cents);

}

Money::Money(Money& parentMoney)

{

\_dollars = parentMoney.\_dollars;

\_cents = parentMoney.\_cents;

}

Методы задания данных о деньгах:

void Money::SetDollars(long dollars)

{

if (dollars < 0)

cout << "Error: dollars less than 0;\n";

else

\_dollars = dollars;

}

void Money::SetCents(int cents)

{

if (cents < 0 || cents > 100)

cout << "Error: cents less than 0 or bigger than 100;\n";

else

\_cents = cents;

}

Перегрузка операторов сравнения, наивысший приоритет при сравнении имеет целая часть валюты, затем дробная:

bool operator>(Money& first, Money& second)

{

if (first.Dollars() == second.Dollars())

return first.Cents() > second.Cents();

else

return first.Dollars() > second.Dollars();

}

bool operator<(Money& first, Money& second)

{

if (first.Dollars() == second.Dollars())

return first.Cents() < second.Cents();

else

return first.Dollars() < second.Dollars();

}

Префиксный инкремент:

Money Money::operator++(int)

{

Money temp(\_dollars, \_cents);

++(\*this);

return temp;

}

Постфиксный инкремент (за основу берётся префиксный инкремент, но возвращается старое значение):

Money& Money::operator++()

{

\_cents++;

if (\_cents == 100)

{

\_cents = 0;

\_dollars++;

}

return \*this;

}

Основная программа, отвечающая за демонстрацию работы операторов с меню:

#include <iostream>

#include <windows.h>

#include <string>

#include "Money.h"

using namespace std;

Money \_first, \_second;

int InputNaturalNum()

{

int number;

do cin >> number; while (number < 1);

return number;

}

string ReadLine()

{

string s;

getline(cin, s);

return s;

}

Money ReadMoneyData()

{

cout << "dollars> ";

int dollars = InputNaturalNum();

cout << "cents> ";

int cents;

do

cents = InputNaturalNum();

while (cents >= 100);

Money money(dollars, cents);

return money;

}

void PrintMoney()

{

cout << "First: ";

\_first.PrintMoney();

cout << "Second: ";

\_second.PrintMoney();

}

void PrintComparison()

{

PrintMoney();

if (\_first > \_second)

cout << endl << "First has more money than second;" << endl;

else if (\_first < \_second)

cout << endl << "Second has more money than first" << endl;

else

cout << "They has the same money" << endl;

}

bool AcceptCommand()

{

cout << ">> ";

string command = ReadLine();

if (command == "0")

return false;

else

{

if (command == "read")

{

cout << "Enter the first item value:" << endl;

\_first = ReadMoneyData();

cout << "Enter the second item value:" << endl;

\_second = ReadMoneyData();

}

else if (command == "compare")

PrintComparison();

else if (command == "print")

PrintMoney();

else if (command == "increment")

{

\_first.PrintMoney();

cout << "type> ";

string subcommand = ReadLine();

if (subcommand == "prefix")

{

++\_first;

\_first.PrintMoney();

}

else

{

\_first++;

\_first.PrintMoney();

}

}

else if (command == "clear")

{

\_first = Money();

\_second = Money();

}

cout << endl;

return true;

}

}

int main()

{

while (AcceptCommand());

return 0;

}

**Полный исходный код программы на языке программирования C++:**

main.cpp:

#include <iostream>

#include <windows.h>

#include <string>

#include "Money.h"

using namespace std;

Money \_first, \_second;

int InputNaturalNum()

{

int number;

do cin >> number; while (number < 1);

return number;

}

string ReadLine()

{

string s;

getline(cin, s);

return s;

}

Money ReadMoneyData()

{

cout << "dollars> ";

int dollars = InputNaturalNum();

cout << "cents> ";

int cents;

do

cents = InputNaturalNum();

while (cents >= 100);

Money money(dollars, cents);

return money;

}

void PrintMoney()

{

cout << "First: ";

\_first.PrintMoney();

cout << "Second: ";

\_second.PrintMoney();

}

void PrintComparison()

{

PrintMoney();

if (\_first > \_second)

cout << endl << "First has more money than second;" << endl;

else if (\_first < \_second)

cout << endl << "Second has more money than first" << endl;

else

cout << "They has the same money" << endl;

}

bool AcceptCommand()

{

cout << ">> ";

string command = ReadLine();

if (command == "0")

return false;

else

{

if (command == "read")

{

cout << "Enter the first item value:" << endl;

\_first = ReadMoneyData();

cout << "Enter the second item value:" << endl;

\_second = ReadMoneyData();

}

else if (command == "compare")

{

PrintComparison();

}

else if (command == "print")

{

PrintMoney();

}

else if (command == "increment")

{

\_first.PrintMoney();

cout << "type> ";

string subcommand = ReadLine();

if (subcommand == "prefix")

{

++\_first;

\_first.PrintMoney();

}

else

{

\_first++;

\_first.PrintMoney();

}

}

else if (command == "clear")

{

\_first = Money();

\_second = Money();

}

cout << endl;

return true;

}

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

while (AcceptCommand());

return 0;

}

money.h:

#pragma once

class Money

{

private:

long \_dollars;

int \_cents;

public:

int Dollars() { return \_dollars; }

void SetDollars(long dollars);

int Cents() { return \_cents; }

void SetCents(int cents);

void PrintMoney();

friend bool operator > (Money& first, Money& second);

friend bool operator < (Money& first, Money& second);

Money& operator++ ();

Money operator++ (int);

Money();

Money(long dollars, int cents);

Money(Money& parentMoney);

~Money();

};

money.cpp:

#include "Money.h"

#include <string>

#include <iostream>

using namespace std;

void Money::SetDollars(long dollars)

{

if (dollars < 0)

cout << "Error: dollars less than 0;\n";

else

\_dollars = dollars;

}

void Money::SetCents(int cents)

{

if (cents < 0 || cents > 100)

cout << "Error: cents less than 0 or bigger than 100;\n";

else

\_cents = cents;

}

void Money::PrintMoney()

{

string centStr = (\_cents < 10) ? '0' + to\_string(\_cents) : to\_string(\_cents);

cout << "Money value = " << \_dollars << '.' << centStr << "$;\n";

}

Money& Money::operator++()

{

\_cents++;

if (\_cents == 100)

{

\_cents = 0;

\_dollars++;

}

return \*this;

}

Money Money::operator++(int)

{

Money temp(\_dollars, \_cents);

++(\*this);

return temp;

}

Money::Money()

{

\_dollars = 0;

\_cents = 0;

}

Money::Money(long dollars, int cents)

{

SetDollars(dollars);

SetCents(cents);

}

Money::Money(Money& parentMoney)

{

\_dollars = parentMoney.\_dollars;

\_cents = parentMoney.\_cents;

}

Money::~Money() {}

bool operator>(Money& first, Money& second)

{

if (first.Dollars() == second.Dollars())

return first.Cents() > second.Cents();

else

return first.Dollars() > second.Dollars();

}

bool operator<(Money& first, Money& second)

{

if (first.Dollars() == second.Dollars())

return first.Cents() < second.Cents();

else

return first.Dollars() < second.Dollars();

}

**Ответы на вопросы:**

**Для чего используются дружественные функции и классы?**

Их можно использовать, если необходимо иметь доступ извне к скрытым полям класса, то есть расширить интерфейс класса.

**Сформулировать правила описания и особенности дружественных функций.**

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.

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.

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

…

};

Использования дружественных функций нужно по возможности избегать, по- скольку они нарушают принцип инкапсуляции и, таким образом, затрудняют отладку и модификацию программы.

Если все методы какого-либо класса должны иметь доступ к скрытым полях другого, весь класс объявляется дружественным с помощью ключевого слова friend.

class student;//предварительное описание класса class teacher

{

…

void teach(student &S);

….

};

class student

{

...

friend class teacher;//все функции класса teacher являются

// дружественными для класса student

…

};

**Каким образом можно перегрузить унарные операции?**

Унарную операцию можно перегрузить:

Как компонентную функцию класса

Как внешнюю (глобальную) функцию

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

Один. Операнд - вызвавший ее объект.

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

Один. Операнд - параметр функции.

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

Два. Первый операнд - вызвавший ее объект, второй - параметр функции.

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

Два. Первый и второй - параметры функции.

**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.

**Каким образом можно перегрузить операцию присваивания?**

Операция может быть перегруженна только внутри области оперделения класса. Операция-функция должна возвращать ссылку на объект, для которого она вызвана, и принимать в качестве параметра единственный аргумент - ссылку на присваиваемый объект.

**Что должна возвращать операция присваивания?**

Операция-функция должна возвращать ссылку на объект, для которого она вызвана.

**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.Пример:

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

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

**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?**

Как вызов унарной функции-операции постфиксной инкрементации с одним операндом - объектом класса.

**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?**

Как вызов дружественной унарной функции-операции постфиксной инкрементации с одним операндом - объектом класса.

**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?**

Как вызовбинарной функции-операции постфиксной инкрементации с двумя операндами - объектом класса и аргументом.

**В программе описан класс class Student**

**{**

**…**

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

**….**

**};**

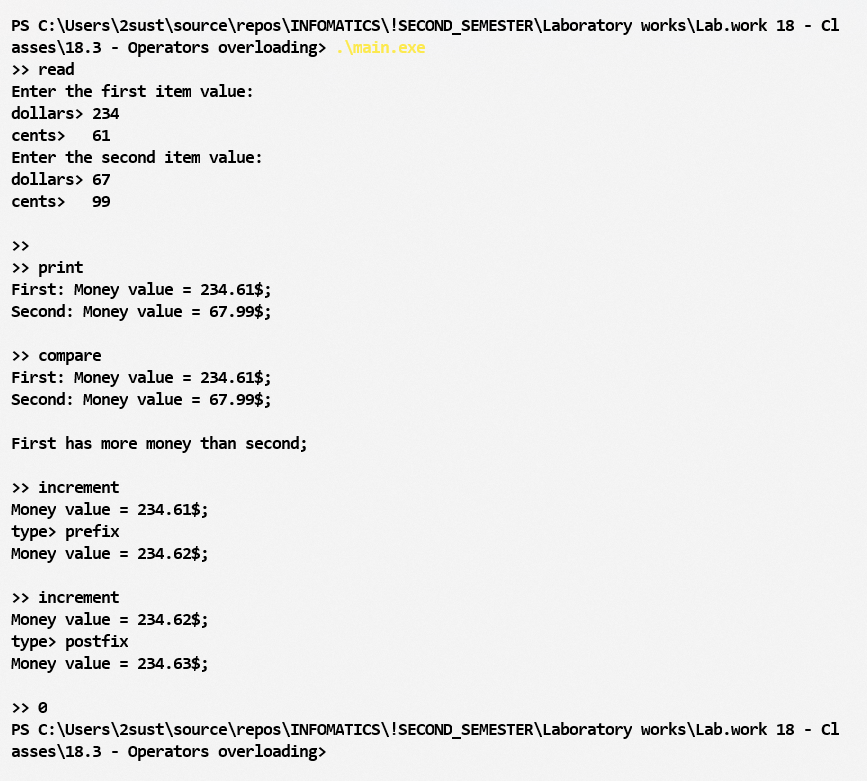
**и определены объекты этого класса Student a,b;**

**Выполняется операция cout<<a>b;**

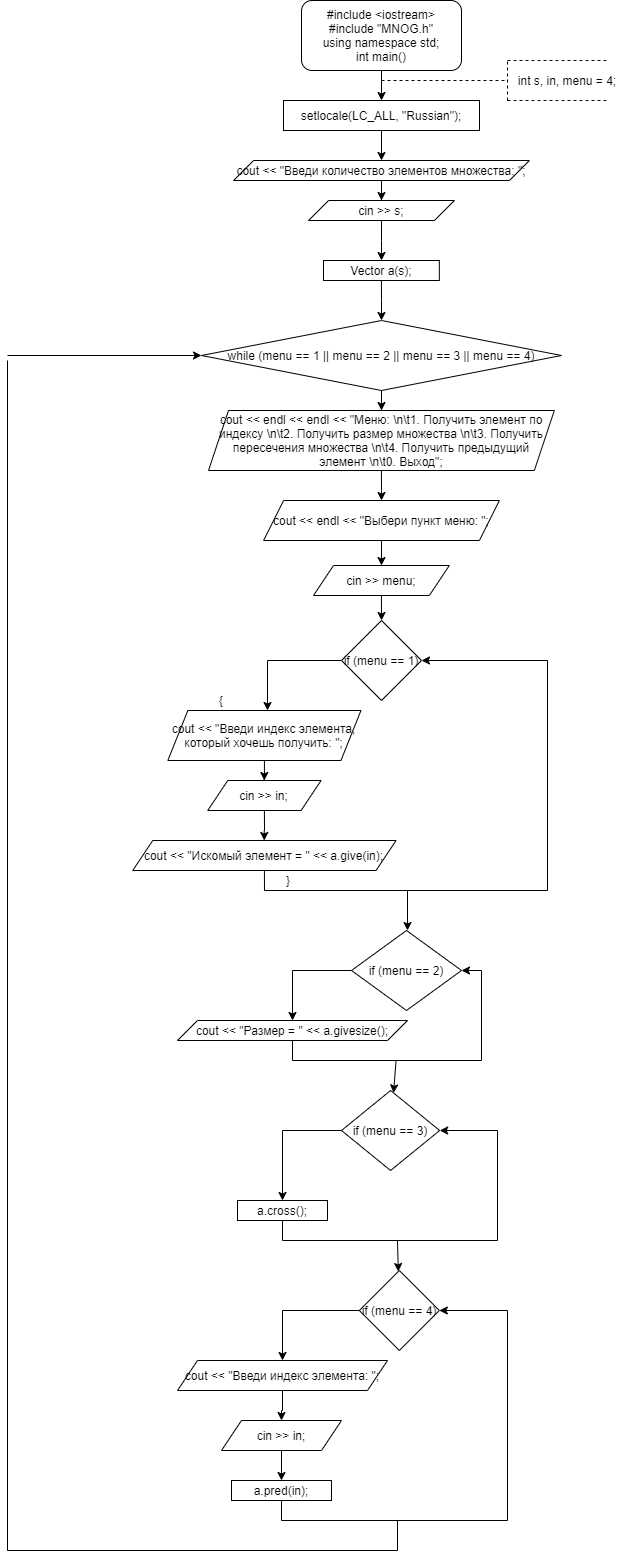
**Каким образом, компилятор будет воспринимать вызов функции-операции?**

Как вызов дружественной бинарной функции-операции постфиксной инкрементации с двумя операндами - объектом класса и аргументом.

Скриншот выполненной программы:



Блок-схема



Vector.cpp:

