Министерство образования и науки Российской Федерации

Федеральное государственное образовательное автономное учреждение высшего образования

"Пермский национальный исследовательский политехнический университет"

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

Дисциплина: Основы алгоритмизации и программирования

Тема: Перегрузка операций

|  |  |
| --- | --- |
| Выполнил работу | |
| Студент группы РИС-22-1б | |
| Деревнин И.В. | |
|  | |
| Проверил работу | |
| Доцент кафедры ИТАС | |
| Полякова О.А. | |
|  | |

Пермь – 2023

**Анализ предметной области**

**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 операции: присваивания, вывода и ввода объектов с помощью потоков, операции указанные в варианте
6. Написать демонстрационную программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

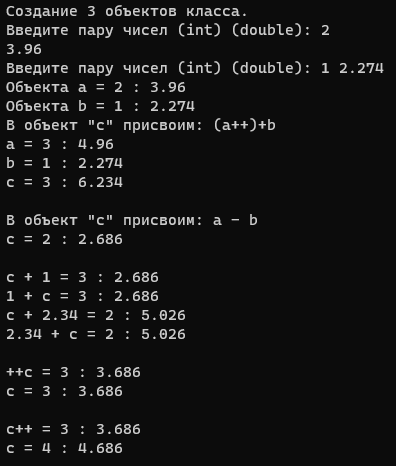
Вариант 15: создать класс Pair (пара чисел). Пара должна быть представлена двумя полями: типа int для первого числа и типа double для второго. Первое число при выводе на экран должно быть отделено от второго числа двоеточием. Реализовать:

- вычитание пар чисел

- добавление константы к паре (увеличивается первое число, если константа целая, второе, если константа вещественная).

**Анализ задачи**

Класс Pair представляет из себя тип данных пара чисел, первое поле которого – имеет тип данных int, а второе double. Для данного типа данных перегрузить операторы: присваивания (метод), ввода и вывода (дружественные методы), вычитания пар чисел (метод), добавление константы к паре (дружественные методы). При добавлении константы необходимо учесть, что константу можно добавлять как справа, так и слева.

****Тестирование программы**

*Рис. 1 – Тестирование программы и класса Pair.*

**Заключение**

Была разработана программа, которая реализовывает собственный тип данных Pair. Для реализации данной программы был описан класс, который хранит в себе 2 поля, из которых составляется тип данных Pair, а также методы, благодаря которым можно управлять объектом класса и операторы, которые над типом данных позволяют производить такие же операции, как и с обычными типами данных (сложение, вычитание, присваивание).

# Ответы на контрольные вопросы

1. Для чего используются дружественные функции и классы?

Дружественные функции используются, когда необходимо расширить интерфейс класса, то есть получить доступ извне к скрытым полям класса.

2. 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

…

};

3. Каким образом можно перегрузить унарные операции?

Внутри класса (как компонентную функцию). Перегружается с помощью нестатического метода без параметром, при этом операндом является вызывающий класс.

Синтаксис перегрузки:

тип operator знак\_оператора(void);

*Пример*

class Person{

string name;

int age;

public:

Person(string, int);//конструктор

//компонентная функция //префиксная операция

Person& operator ++() {

++age;

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

};

//в основной функции

Person p1(”Ivanov”,20);

++p1;

p1.Show();

Вне класса (как внешнюю функцию). Перегружается как функция с параметром типа класса.

Синтаксис перегрузки:

тип operator знак\_оператора(class A); //передача объекта класса

*Пример*

class Person {

string name;

int age;

public:

Person(string, int);//конструктор

friend Person & operator ++(Person&) ; //внешняя дружественная функция

};

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

++p.age;

return p; }

//в основной функции

Person p1(”Ivanov”,20);

++p1;

p1.Show();

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

Такая операция перегружается с помощью нестатического метода без параметров (с одним неявным параметром this), если инкремент и декремент префиксные. Если же операции постфиксные, то метод должен иметь один параметр типа int (второй неявный параметр - this)

Person & operator ++(); //префиксная форма

Person & operator ++(int d); //постфиксная форма

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

Так как параметр this не передаётся глобальной функции, то такая функция перегружается как функция с одним параметром типа класса, если инкремент и декремент префиксные. Если же операции постфиксные, то параметров должно быть два: параметр типа int и типа класса.

Person & operator ++(Person& p); //префиксная форма

Person & operator ++(int d, Person& p); //постфиксная форма

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

Бинарная функция-операция перегружается нестатическим методом с одним параметром, так как вызвавший ее объект считается первым операндом (по умолчанию передается объект класса this)

*Пример*:

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();

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

Если функция определяется вне класса, то параметр this не передается. Функция должна иметь два параметра типа класса, один из которых передаётся вместо параметра this, а вторым передается тот, с которым взаимодействует исходный класс посредством оператора.

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();

8. Чем отличается перегрузка префиксных и постфиксных унарных операций?

Если операции постфиксные, то метод должен иметь один дополнительный параметр типа int, который позволяет компилятору различать операции.

Операторы могут отличаться по типу возвращаемого значения. Если префиксный оператор модифицирует какое-то поле, то он возвращает ссылку на объект этого класса. Постфиксный же оператор (из-за принципа работы постфикса) должен сохранять состояние объекта класса во временную переменную, затем модифицировать поле класса, а вернуть значение из временной переменной. Таким образом, на постфиксные операторы накладываются ограничения (они не позволяют работать напрямую с объектом класса)

9. Каким образом можно перегрузить операцию присваивания?

Если класс содержит поля, память под которые выделяется динамически, необходимо определить собственную операцию присваивания. Её можно перегрузить только как нестатическую компонентную функцию класса. Перегрузка происходит с помощью ключевого слова operator.

class Person {

string name;

int age;

public:

Person(string, int);//конструктор

Person& operator =(const Person&); //компонентная функция

};

Person& Person::operator = (const& Person p) {

//проверка на самоприсваивание

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

name = p.name;

age = p.age;

return \*this; }

//в основной функции

Person p1(”Ivanov”,20);

Person p2;

p2=p1;

p1.Show();

p2.Show();

10. Что должна возвращать операция присваивания?

Чтобы сохранить семантику присваивания, операция-функция должна возвращать ссылку на объект, для которого она вызвана (то есть левый операнд), и принимать в качестве параметра единственный аргумент — ссылку на присваиваемый объект.

11. 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.

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); }

12. 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?

Так как операция префиксная и перегружена внутри класса, то компилятор воспримет это как вызов метода класса s.operator++();

13. 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?

Так как функция дружественная и префиксная, то компилятор воспримет это как вызов глобальной функции operator++(s);

14. 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?

Приоритет у оператора << выше, чем у оператора <, так что программа компилироваться не будет. Если выполнится cout << (a < b), то компилятор воспримет это как вызов метода класса a.operator<(b);

15. В программе описан класс

class Student {

…

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

….

};

и определены объекты этого класса

Student a,b;

Выполняется операция

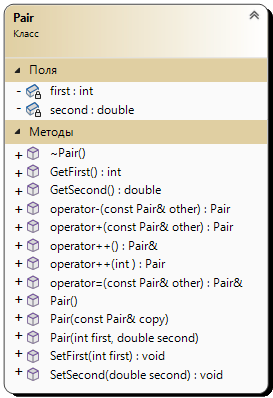
cout<<a>b;

Каким образом, компилятор будет воспринимать вызов функции-операции?

Приоритет у оператора << выше, чем у оператора <, так что программа компилироваться не будет. Если выполнится cout << (a > b), то компилятор воспримет это как вызов глобальной функции operator>(a, b);

**Приложения**

Приложение UML-диаграмма



Приложение Б – код программы

Main.cpp:

#include "Pair.h"

int main()

{

setlocale(LC\_ALL, "ru");

cout << "Создание 3 объектов класса." << endl;

Pair a, b, c;

cout << "Введите пару чисел (int) (double): ";

cin >> a;

cout << "Введите пару чисел (int) (double): ";

cin >> b;

cout << "Объекта а = " << a << endl;

cout << "Объекта b = " << b << endl;

cout << "В объект \"с\" присвоим: (a++)+b" << endl;

c = (a++) + b;

cout << "a = " << a << endl;

cout << "b = " << b << endl;

cout << "c = " << c << endl << endl;

cout << "В объект \"с\" присвоим: a - b" << endl;

c = a - b;

cout << "c = " << c << endl << endl;

cout << "c + 1 = " << c + 1 << endl;

cout << "1 + c = " << 1 + c << endl;

cout << "c + 2.34 = " << c + 2.34 << endl;

cout << "2.34 + c = " << 2.34 + c << endl << endl;

cout << "++c = " << ++c << endl;

cout << "c = " << c << endl << endl;

cout << "c++ = " << c++ << endl;

cout << "c = " << c << endl << endl;

return 0;

}

pair.h:

#pragma once

#include <iostream>

using namespace std;

class Pair

{

public:

Pair(); //конструктор

Pair(int first, double second); //конструктор с параметрами

Pair(const Pair& copy); //конструктор без параметров

~Pair() {} //деструктор

int GetFirst() { return this->first; } //геттеры и сеттеры для первого и второго поля.

double GetSecond() { return this->second; }

void SetFirst(int first) { this->first = first; }

void SetSecond(double second) { this->second = second; }

Pair& operator =(const Pair& other); //перегрузка оператора присваивания

Pair operator +(const Pair& other); //перегрузка оператора сложения 2-х пар чисел

Pair& operator ++(); //инкремент префиксный

Pair operator ++(int); //инкремент постфиксный

Pair operator -(const Pair& other); //оператор вычитания 2-х пар чисел

friend const Pair operator +(const Pair& left, const int& right); //добавление int константы справа и слева

friend const Pair operator +(const int& left, const Pair& right);

friend const Pair operator +(const Pair& left, const double& right);//добавление double константы справа и слева

friend const Pair operator +(const double& left, const Pair& right);

friend ostream& operator <<(ostream& os,const Pair& str); //перегрузка оператора вывода

friend istream& operator >> (istream& is, Pair& str); //перегрузка оператора ввода

private:

int first; //поле 1

double second; //поле 2

};

pair.cpp:

#include "Pair.h"

ostream& operator <<(ostream& os, const Pair& val)

{

os << val.first << " : " << val.second;

return os;

}

istream& operator>>(istream& is, Pair& val)

{

is >> val.first >> val.second;

return is;

}

Pair::Pair()

{

first = 0;

second = 0;

}

Pair::Pair(const Pair& copy)

{

this->first = copy.first;

this->second = copy.second;

}

Pair::Pair(int first, double second)

{

this->first = first;

this->second = second;

}

Pair& Pair::operator=(const Pair& other)

{

this->first = other.first;

this->second = other.second;

return \*this;

}

Pair Pair::operator+(const Pair& other)

{

Pair temp(this->first + other.first, this->second + other.second);

return temp;

}

Pair& Pair::operator++()

{

++this->first;

++this->second;

return \*this;

}

Pair Pair::operator++(int)

{

Pair temp(\*this);

++this->first;

++this->second;

return temp;

}

Pair Pair::operator-(const Pair& other)

{

Pair temp(this->first - other.first, this->second - other.second);

return temp;

}

const Pair operator+(const Pair& left, const int& right)

{

return Pair(left.first + right, left.second);

}

const Pair operator+(const int& left, const Pair& right)

{

return right + left;

}

const Pair operator+(const Pair& left, const double& right)

{

return Pair(left.first, left.second + right);

}

const Pair operator+(const double& left, const Pair& right)

{

return right + left;

}