Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

Семестр: 2

На тему: «ООП. Перегрузка операций»

Выполнил студент ИВТ-22-2б:

Казанцев Антон Васильевич

(дата, подпись)

Проверила:

Полякова Ольга Андреевна

(дата, подпись)

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 15:*

Создать класс Pair (пара чисел). Пара должна быть представлено 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**

1. Pair::Pair(int first, double second) - конструктор с параметрами

2. istream& operator>>(istream& in, Pair& m\_pair) - перегрузка оператора ввода

3. ostream& operator<<(ostream& out, const Pair& m\_pair) – перегрузка оператора вывода

4. Pair::Pair() - конструктор без параметров

5. Pair::~Pair() - деструктор

6. Pair::Pair(const Pair& m\_pair) - конструктор копирования

7. Pair& Pair::operator=(const Pair& m\_pair) - перегрузка оператора присваивания

8. Pair Pair::operator-(const Pair& m\_pair) - перегрузка оператора вычитания

9. Pair& Pair::operator+(int first) - перегрузка оператора сложения - целый аргумент

10. Pair& Pair::operator+(double second) - перегрузка оператора сложения - дробный аргумент

11. Pair& Pair::operator++() - перегрузка оператора ++ префикс

12. Pair Pair::operator ++(int) - перегрузка оператора ++ постфикс

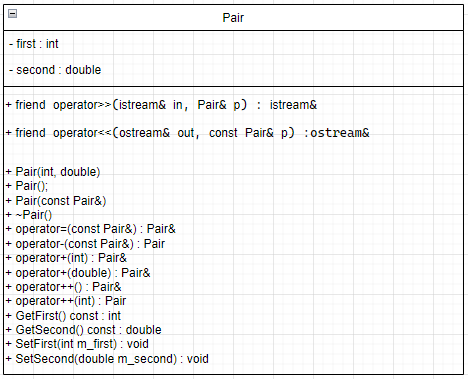
13. int Pair::GetFirst() const – геттер для первого числа

14. double Pair::GetSecond() const - геттер для второго числа

15. void Pair::SetFirst(int m\_first) – сеттер для первого числа

16. void Pair::SetSecond(double m\_second) – сеттер для второго числа

**UML-Диаграмма**



**Программное решение**

***Pair.h***

#pragma once

#include <iostream>

using namespace std;

class Pair {

int first;

double second;

public:

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

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

Pair(int, double);

Pair();

Pair(const Pair&);

~Pair();

Pair& operator=(const Pair&);

Pair operator-(const Pair&);

Pair& operator+(int);

Pair& operator+(double);

Pair& operator++();

Pair operator++(int);

int GetFirst() const;

double GetSecond() const;

void SetFirst(int m\_first);

void SetSecond(double m\_second);

};

***Pair.cpp***

#include "Pair.h"

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

this->first = first;

this->second = second;

}

istream& operator>>(istream& in, Pair& m\_pair) {//перегрузка оператора ввода

in >> m\_pair.first;

in >> m\_pair.second;

return in;

}

ostream& operator<<(ostream& out, const Pair& m\_pair) {//перегрузка оператора вывода

return (out << m\_pair.first << " : " << m\_pair.second);

}

Pair::Pair() {//конструктор без параметров

first = 0;

second = 0;

}

Pair::~Pair() {//деструктор

}

Pair::Pair(const Pair& m\_pair) {//конструктор копирования

first = m\_pair.first;

second = m\_pair.second;

}

Pair& Pair::operator=(const Pair& m\_pair) {//перегрузка оператора присваивания

if (&m\_pair != this) {

first = m\_pair.first;

second = m\_pair.second;

}

return \*this;

}

Pair Pair::operator-(const Pair& m\_pair) {//перегрузка оператора вычитания

Pair result(first - m\_pair.first, second - m\_pair.second);

return result;

}

Pair& Pair::operator+(int first) {//перегрузка оператора сложения - целый аргумент

this->first += first;

return \*this;

}

Pair& Pair::operator+(double second) {//перегрузка оператора сложения - дробный аргумент

this->second += second;

return \*this;

}

Pair& Pair::operator++() {//перегрузка оператора ++ префикс

++first;

++second;

return \*this;

}

Pair Pair::operator ++(int) {//перегрузка оператора ++ постфикс

Pair temp = \*this;

this->first++;

this->second++;

return temp;//сначала нужно вывести без изменений

}

int Pair::GetFirst() const{

return first;

}

double Pair::GetSecond() const{

return second;

}

void Pair::SetFirst(int m\_first) {

first = m\_first;

}

void Pair::SetSecond(double m\_second) {

second = m\_second;

}

***LabaOOP3.cpp***

#include <iostream>

#include "Pair.h"

using namespace std;

Pair& operator+(int number, Pair& pair) { //заменяем местами для коммуникативности - сложение с целой константой

return pair + number;

}

Pair& operator+(double number, Pair& pair) { //заменяем местами для коммуникативности - сложение с вещественной константой

return pair + number;

}

int main() {

system("chcp 1251 > NULL");

Pair pair1;

Pair pair2(3, 4.5);

cout << "Вывод значений pair2:\n" << pair2 << '\n';

cout << "Ввод значений для pair1:\n";

cin >> pair1;

cout << "Вывод значений pair1:\n" << pair1 << '\n';

cout << "Вычитание из значений pair2 соответствующие значения pair1:\n" << pair2 - pair1 << '\n';

cout << "Присваивние: pair1 = pair2\n";

pair1 = pair2;

cout << "Вывод значений pair1:\n" << pair1 << '\n';

Pair tmp = pair1;//сохраняем значение

cout << "Сложение pair1 с целой константой 3:\n";

cout << pair1 + 3 << '\n';

pair1 = tmp;//возвращаем предыдущее значение

cout << "Проверка коммуникативности:\n";

cout << 3 + pair1 << '\n';

tmp = pair1;//сохраняем значение

cout << "Сложение pair1 с вещественной константой 4.5:\n";

cout << pair1 + 4.5 << '\n';

pair1 = tmp;//возвращаем предыдущее значение

cout << "Проверка коммуникативности:\n";

cout << 4.5 + pair1 << '\n';

cout << "++ префикс для pair1:\n";

cout << ++pair1 << '\n';

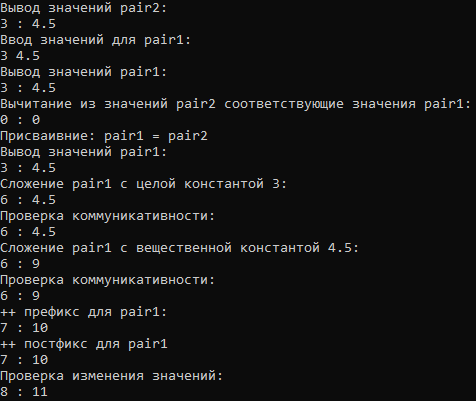
cout << "++ постфикс для pair1\n";

cout << pair1++ << '\n';

cout << "Проверка изменения значений:\n";

cout << pair1 << '\n';

}



**Контрольные вопросы**

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

Если необходимо иметь доступ извне к скрытым полям класса, то есть расширить интерфейс класса, то можно использовать дружественные функции и дружественные классы. Метод, как правило, описывает свойство объекта, а в виде дружественных функций оформляются действия, не являющиеся свойствами класса, но концептуально входящие в его интерфейс и нуждающиеся в доступе к его скрытым полям, например, переопределенные операции ввода/вывода объектов.

*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.

*Пример:*

class Pair {

private:

int a;

double b;

public:

//Перегрузка операторов ввода/вывода

friend istream& operator>>(istream& in, Pair& N);

friend ostream& operator<<(ostream& out, const Pair& N);

};

- 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.

*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

. . .

};

- Одна функция может быть дружественной сразу нескольким классам.

*Пример:*

class student; //Предварительное описание класса

class teacher

{

. . .

friend void teach(student&); //Дружественная функция

. . .

};

class student

{

. . .

friend void teach(student&); //Такая же дружественная функция

. . .

};

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

*3. Каким образом можно перегрузить унарные операции?*

Унарную операцию можно перегрузить:

1. **Компонентная функция класса**.

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

*Пример:*

class Person

{

string name;

int age;

public:

Person() { name = "Ivan"; age = 18; } //конструктор

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

++age;

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

}

};

int main() {

Person s1;

++s1; //age=19

return 0;

}

2. **Внешняя (глобальная) функция**.

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

*Пример:*

class Person

{

string name;

int age;

public:

Person() { name = "Ivan"; age = 18; } //конструктор

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

};

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

{

++p.age;

return p;

}

int main() {

Person s1;

++s1; //age=19

return 0;

}

*4. 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** (с неявным параметром this) **для префиксных** инкремента и декремента.

Person& operator ++();

Операции **постфиксного** инкремента и декремента должны иметь **первый параметр типа int** (второй параметр неявный – this). Это используется только для того, чтобы отличить их от префиксной формы.

Person& operator ++(int);

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

Если функция определяется вне класса (параметр this не передается), она должна иметь **один параметр** типа класса для **префиксных** операция.

Person& operator ++(Person& p);

**Два параметра**: типа класса и параметр int для **постфиксных** операций.

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

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

Бинарная функция-операция, определяемая **внутри класса**, должна быть представлена с помощью нестатического метода с **одним явным параметром**, при этом вызвавший ее объект считается первым операндом (неявный параметр this).

*Пример:*

class Person

{

string name;

int age;

public:

Person() { name = "Ivan"; age = 18; } //конструктор

Person& operator +(int x) //Компонентная функция

{

age += x;

return \*this;

}

};

int main() {

Person s1;

s1+2; //age=20

return 0;

}

*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**, один из которых передаётся вместо параметра this и имеет тип класса, а вторым передается тот, с которым взаимодействует исходный объект класс посредством оператора.

*Пример:*

class Person

{

string name;

int age;

public:

Person() { name = "Ivan"; age = 18; } //конструктор

friend Person& operator +(Person& p, int x); //внешняя дружественная функция

};

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

{

p.age += x;

return p;

}

int main() {

Person s1;

s1+2; //age=20

return 0;

}

*8. Чем отличается перегрузка префиксных и постфиксных унарных операций?*

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

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

*Пример (префикс):*

class Person

{

string name;

int age;

public:

Person() { name = "Ivan"; age = 18; } //Конструктор

Person& operator ++() { //Префиксная операция

++age;

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

}

};

int main() {

Person s1;

k=++s1; //age=19, k=19

return 0;

}

*Пример(постфикс)*

class Person

{

string name;

int age;

public:

Person() { name = "Ivan"; age = 18; } //Конструктор

Person& operator ++() { //Префиксная операция

Person temp(\*this);

++(\*this);

return temp; //Указатель на объект, вызвавший метод

}

};

int main() {

Person s1;

k=s1++; //age=19, k=18

return 0;

}

*9. Каким образом можно перегрузить операцию присваивания?*

Операция присваивания определена в любом классе по умолчанию как поэлементное копирование. Эта операция вызывается каждый раз, когда одному существующему объекту присваивается значение другого. Если класс содержит поля, память под которые выделяется динамически, необходимо определить собственную операцию присваивания. Её можно перегрузить только как нестатическую компонентную функцию класса. Функция должна принимать в качестве параметра единственный аргумент — ссылку на присваиваемый объект.

Перегрузка происходит с помощью ключевого слова operator.

*Пример:*

class Person

{

string name;

int age;

public:

Person() { name = "Ivan"; age = 18; } //Конструктор

Person& operator = (const Person& p) {

if (&p == this) return\*this; //Проверка на самоприсваивание

name = p.name;

age = p.age;

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

}

};

int main() {

Person s1;

Person s2;

s1 = s1;

return 0;

}

*10. Что должна возвращать операция присваивания?*

Чтобы сохранить семантику присваивания, операция-функция должна возвращать ссылку на объект, для которого она вызвана (левый операнд).

*Пример:*

Person& operator = (const Person& p) {

if (&p == this) return\*this; //Проверка на самоприсваивание

name = p.name;

age = p.age;

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

}

*11. Каким образом можно перегрузить операции ввода-вывода?*

Операции ввода (operator >>) вывода (operator<<) всегда реализуются как внешние дружественные функции, т. к. левым операндом этих операций являются потоки. Функции в качестве параметров принимают ссылку на поток ввода/вывода и ссылку на объект (неконстантная для вывода, константная для вывода), для которого нужно выполнить операцию. Функции возвращают имя потока.

*Пример:*

class Person

{

string name;

int age;

public:

Person() { name = "Ivan"; age = 18; } //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) {

out << p.name << endl;

out << p.age;

return out;

}

*12. В программе описан класс:*

class Student

{ . . .

Student& operator++();

. . .

};

*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&);

. . .

};

*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);

. . .

};

*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); и выведет одно из логических значений, полученных в результате операции.

*15. В программе описан класс:*

class Student

{ . . .

friend bool operator>(const Student&, Student&);

. . .

};

*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); и выведет одно из логических значений, полученных в результате операции.