Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Перегрузка операций»

Семестр 2

Выполнил работу:

Студент группы ИВТ-22-2б

Голубцов Никита Валерьевич

Проверил:

Доцент кафедры ИТАС

Полякова Ольга Андреевна

Г. 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.

**Описание класса**

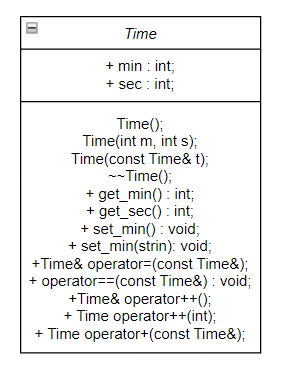
Создать класс Time для работы с временными интервалами. Интервал должен быть

представлен в виде двух полей: минуты типа int и секунды типа int. при выводе минуты

отделяются от секунд двоеточием. Реализовать:  сложение временных интервалов (учесть, что в минуте не может быть более 60

секунд)  сравнение временных интервалов (==)

**UML - Диаграмма**

****

**Рис. 1 – UML-диаграмма класса Time**

**Контрольные вопросы**

1. Для чего используются дружественные функции и классы?

Дружественные функции применяются для доступа к скрытым полям класса и представляют собой альтернативу методам. 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 вывода объектов.

1. 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 передаётся.

- Дружественная функция может быть обычной функцией или методом другого ранее определённого класса. На неё не распространяется действие спецификаторов доступа, место размещения её объявления в классе безразлично.

- Одна функция может быть дружественной сразу нескольким классам:

class daughter;//предварительное описание класса

class family

{

public:

int members;

void help(daughter & S);

};

class daughter

{

int age;

friend void family::help(daughter&);//дружественная функция, доступ к методу

};

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 daughter

{

int age;

friend void family;//дружественная функция

};

1. Каким образом можно перегрузить унарные операции?

Унарную операцию можно перегрузить:

-Как компонентную функцию класса

-Как внешнюю (глобальную) функцию

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

class Person

{

string name;

int age;

public:

Person(); // Конструктор без параметров

Person(string, int);//конструктор с параметрами

Person(const Person&); // Конструктор копирования

~Person(); // Деструктор

//компонентная функция

Person& operator ++() //префиксная операция

{

++age;

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

}

};

void main()

{

Person first("Polina", 18);

++first;

}

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

class Person

{

string name;

int age;

public:

Person(); // Конструктор без параметров

Person(string, int);//конструктор с параметрами

Person(const Person&); // Конструктор копирования

~Person(); // Деструктор

friend Person& operator ++(Person&);

};

Person& operator ++(Person&p)

{

++p.age;

return p;

}

void main()

{

Person first("Polina", 18);

++first;

}

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

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 (кроме неявного this), если инкремент и декремент префиксные: Person& operator ++()

- Операции постфиксного инкремента и декремента должны иметь первый параметр типа int: Person& operator ++(int n)

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

- Унарная функция-операция, определяемая вне класса, должна иметь один параметр типа класса, если инкремент и декремент префиксные:

Person& operator ++(Person&p)

- Если функция-операция постфиксного инкремента или декремента, то она должна иметь два параметра - типа int и типа класса:

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

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 её объект считается первым операндом:

class Person

{

string name;

int age;

public:

Person(); // Конструктор без параметров

Person(string, int);//конструктор с параметрами

Person(const Person&); // Конструктор копирования

~Person(); // Деструктор

Person& operator +(int n)

{

age += n;

return \*this;

}

};

void main()

{

Person first("Polina", 18);

first+5;

}

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

Если функция определяется вне класса, она должна иметь два параметра типа класса, причём второй должен быть того типа, с которым взаимодействует функция:

class Person

{

string name;

int age;

public:

Person(); // Конструктор без параметров

Person(string, int);//конструктор с параметрами

Person(const Person&); // Конструктор копирования

~Person(); // Деструктор

friend Person& operator +(Person&, int age);

};

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

{

++p.age;

return p;

}

void main()

{

Person first("Polina", 18);

first+5;

}

1. Чем отличается перегрузка префиксных и постфиксных унарных операций?

- Если функция постфиксная, то она должна иметь дополнительный операнд int, который служит знаком для компилятора о том, какая это функция.

1. Каким образом можно перегрузить операцию присваивания?

Если класс содержит поля, память под которые выделяется динамически, необходимо определить собственную операцию присваивания.

#include <string>

#include <iostream>

using namespace std;

class Person

{

string name;

int age;

public:

Person(string, int);//конструктор

Person(); // конструктор по умолчанию

~Person(); // деструктор

//компонентная функция

Person& operator =(const Person&);

};

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

{

//проверка на самоприсваивание if (&p==this)

age = p.age;

return \*this;

}

Person::Person()

{

name = "";

age = 0;

}

Person::Person(string name, int age)

{

this->name = name;

this->age = age;

}

Person::~Person()

{

cout << "all was deleted";

}

void main()

{

Person first("Polina", 18);

Person second;

second = first;

}

1. Что должна возвращать операция 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.

1. Каким образом можно перегрузить операции ввода-вывода?

class Person

{

string name;

int age;

public:

Person(string, int);//конструктор

Person();

~Person();

//дружественная глобальная функция

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

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

};

Person::Person(string name, int age)

{

this->name = name;

this->name = name;

}

Person::Person()

{

name = "";

age = 0;

}

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 определён объект этого класса Student s;

Выполняется операция

++s;

Каким образом, компилятор будет воспринимать вызов функции-операции?

Данная операция префиксная, определённая внутри класса, следовательно, компилятор воспримет её как вызов метода класса s.operator++().

1. В программе описан класс class Student

{

…

friend Student& operator ++( Student&);

….

};

и определён объект этого класса Student s;

Выполняется операция

++s;

Каким образом, компилятор будет воспринимать вызов функции-операции?

Данная операция дружественная, префиксная и определённая вне класса, следовательно, компилятор воспримет её как вызов глобальной функции operator++(s).

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?

У оператора << приоритетность выполнения выше, чем у оператора <, поэтому программа компилироваться не будет.

Если изменить операцию, обозначив приоритетность необходимым способом, cout << (a < b), то компилятор воспримет это как вызов метода класса a.operator<(b).

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?

У оператора << приоритетность выполнения выше, чем у оператора <, поэтому программа компилироваться не будет.

Если изменить операцию, обозначив приоритетность необходимым способом, cout << (a > b), то компилятор воспримет это как вызов метода класса operator>(a, b).