Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе № 15.3**

Дисциплина: «Информатика»

Тема: Классы и объекты. Перегрузка операторов

Вариант 12

Выполнила:

Студентка группы РИС-22-1б

Черкасова А.А.

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь 2023**

**Содержание**

**Введение**

**Цель:**

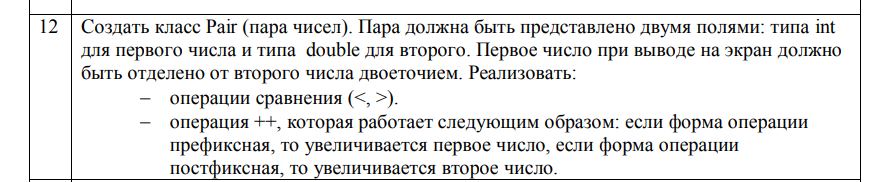
* Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
* Использование перегруженных функций

Для достижения поставленной цели, необходимо решить следующие **задачи**:

* Провести анализ задачи
* Реализовать задачу на языке С++
* Составить блок-схему

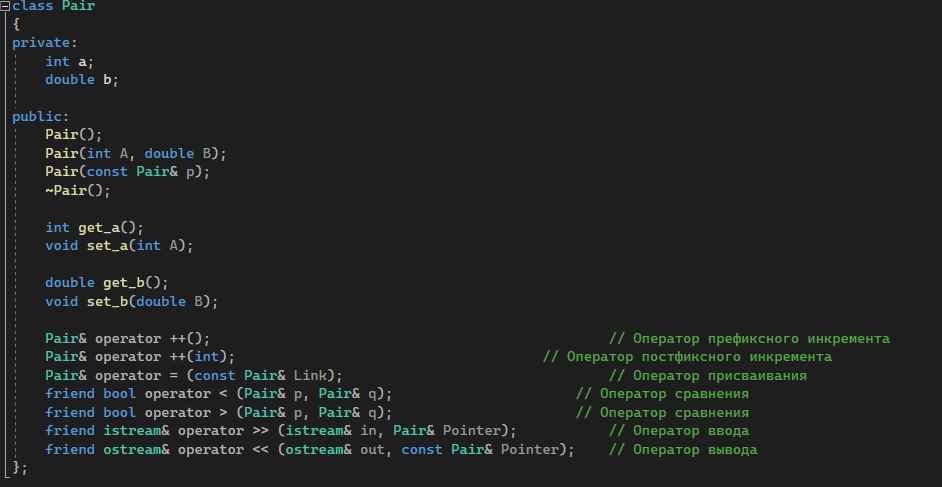
**Постановка задачи:**

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 операцию ввода и вывода объектов с помощью потоков.
7. Перегрузить операции, указанные в варианте.
8. Написать демонстрационную программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

****

# **Анализ задачи**

1. Для решения задачи необходимо:
   1. Реализовать определение класса Pair.



* 1. Реализовать определение конструкторов без параметра, с параметрами и копирования, деструктора, селекторов и модификаторов класса.
  2. Реализовать определение перегрузки операторов.
  3. Реализовать применение этих функций в главной функции.

1. В ходе работы были использованы типы данных:
   1. Для конструктора без параметров Pair класса Pair не используются аргументы:

Сам конструктор заполняет атрибуты объекта класса “пустотами”

* 1. Для конструктора с параметрами Pair класса Pair используются следующие аргументы:
  2. Тип int: переменная a
  3. Тип double: переменная b

Сам конструктор заполняет атрибуты объекта класса параметрами.

* 1. Для конструктора копирования Pair класса Pair используются следующие аргументы:
  2. Адресация типа const Pair: объект копирования.

Сам конструктор заполняет атрибуты объекта класса атрибутами объекта копирования.

* 1. Для деструктора Pair класса Pair не используются аргументы:

Сам деструктор удаляет атрибуты объекта класса.

* 1. Для селектора get\_a не используются аргументы.

Сам селектор имеет тип int и возвращает значение поля a.

* 1. Для селектора get\_b не используются аргументы.

Сам селектор имеет тип double и возвращает значение поля b.

* 1. Для модификатора set\_a используются следующие аргументы:

1. Тип int: переменная a.

Сам модификатор имеет тип void, поскольку при работе селектора не нужно возвращать значение.

* 1. Для модификатора set\_b используются следующие аргументы:

1. Тип double: переменная b.

Сам модификатор имеет тип void, поскольку при работе селектора не нужно возвращать значение.

* 1. Для метода перегрузки оператора префиксного инкремента не используются аргументы.

Сам метод имеет тип Pair& и возвращает указатель на объект this.

* 1. Для метода перегрузки постфиксного инкремента используются следующие аргументы:
  2. Тип int: показатель того, что происходит перегрузка постфиксного инкремента.

Сам метод имеет тип Pair& и возвращает указатель на объект this.

* 1. Для метода перегрузки оператора присваивания используются следующие аргументы:
  2. Тип const Pair&: ссылка на объект.

Сам метод имеет тип Pair& и возвращает указатель на объект this.

* 1. Для дружественной функции перегрузки оператора ввода используются следующие аргументы:
  2. Тип istream&: поток ввода.
  3. Тип const Pair&: ссылка на объект.

Сама функция имеет тип istream& и возвращает вводимые данные.

* 1. Для дружественной функции перегрузки оператора вывода используются следующие аргументы:
  2. Тип ostream&: поток вывода.
  3. Тип const Pair&: ссылка на объект.

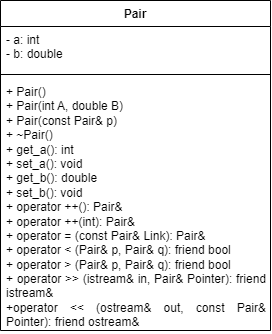
Сама функция имеет тип ostream& и возвращает выводимые данные.

* 1. Для метода перегрузки операторов сравнения используются следующие аргументы:
  2. Тип Pair&: ссылка на 1-ый сравниваемый объект.
  3. Тип Pair&: ссылка на 2-ой сравниваемый объект.

Сам метод имеет тип Pair& и возвращает указатель на объект this.

1. Для решения задачи данные были представлены в следующем виде:
   1. Для работы с данными используются атрибуты класса.
2. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Ввод данных реализован с помощью оператора cin, используемых при реализации в главной функции и функций перегрузки оператора.
   2. Вывод данных реализован с помощью оператора cout, используемого при реализации в главной функции и функций перегрузки операторов ввода-вывода.
3. Поставленные задачи решены следующими действиями:
   1. Определение класса Pair было реализовано в заголовочном файле Pair.h, определение конструкторов, деструктора, селекторов и модификаторов класса – в файле Pair.cpp. Основной блок программы описан в главном файле main.cpp.
   2. При работе с объектами класса Pair обращение к конструктору без параметра аналогично выделению памяти под переменную. Обращение к конструктору с параметрами аналогично обращению к методу Init. Обращение к конструктору копирования аналогично оператору присваивания. Обращение к селекторам и модификаторам аналогично обращению к методам или к полям структуры.
   3. При работе с перегруженными операторами работа без объектов созданного класса происходит по умолчанию. Если при вызове перегруженных операторов используются объекты класса, то операторы работают согласно их определению в этом классе.

# **UML – диаграмма**



# **Приложение А**

**Листинг программы**

*A) Заголовочный файл Pair.h*

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

int a;

double b;

public:

Pair();

Pair(int A, double B);

Pair(const Pair& p);

~Pair();

int get\_a();

void set\_a(int A);

double get\_b();

void set\_b(double B);

Pair& operator ++(); // Оператор префиксного инкремента

Pair& operator ++(int); // Оператор постфиксного инкремента

Pair& operator = (const Pair& Link); // Оператор присваивания

friend bool operator < (Pair& p, Pair& q); // Оператор сравнения

friend bool operator > (Pair& p, Pair& q); // Оператор сравнения

friend istream& operator >> (istream& in, Pair& Pointer); // Оператор ввода

friend ostream& operator << (ostream& out, const Pair& Pointer); // Оператор вывода

};

*B) Файл с описанием методов класса Pair.cpp*

#include "Pair.h"

Pair::Pair()

{

a = 0;

b = 0.0;

}

Pair::Pair(int A, double B)

{

a = A;

b = B;

}

Pair::Pair(const Pair& p)

{

a = p.a;

b = p.b;

}

Pair::~Pair()

{

}

int Pair:: get\_a()

{

return a;

}

void Pair::set\_a(int A)

{

a = A;

}

double Pair::get\_b()

{

return b;

}

void Pair::set\_b(double B)

{

b = B;

}

Pair& Pair::operator ++() // Оператор префиксного инкремента

{

++a;

return \*this;

}

Pair& Pair::operator ++(int) // Оператор постфиксного инкремента

{

b++;

return \*this;

}

Pair& Pair::operator = (const Pair& Link) // Оператор присваивания

{

if (&Link != this)

{

a = Link.a;

b = Link.b;

}

return \*this;

}

bool operator < (Pair& p, Pair& q) // Оператор сравнения

{

if (p.a < q.a)

{

return true;

}

else

{

if (p.a > q.a)

{

return false;

}

else

{

if (p.b < q.b)

{

return true;

}

else

{

return false;

}

}

}

}

bool operator > (Pair& p, Pair& q) // Оператор сравнения

{

if (p.a > q.a)

{

return true;

}

else

{

if (p.a < q.a)

{

return false;

}

else

{

if (p.b > q.b)

{

return true;

}

else

{

return false;

}

}

}

}

istream& operator >> (istream& in, Pair& Pointer) // Оператор ввода

{

cout << "Введите a: ";

in >> Pointer.a;

cout << "Введите b: ";

in >> Pointer.b;

return in;

}

ostream& operator << (ostream& out, const Pair& Pointer) // Оператор вывода

{

cout << "a:b - ";

return (out << Pointer.a << " : " << Pointer.b);

}

*C) Файл с главной программой main.cpp*

#include <iostream>

#include "Pair.h"

using namespace std;

int main()

{

system("chcp 1251>nul");

Pair A;

cout << "Инициализация объекта A\n";

cin >> A;

cout << "\nВывод объекта A\n";

cout << A << endl;

cout << "\nИзменение объекта A\n";

for (int i = 0; i < 10; i++)

{

++A; // префиксная форма

}

cout << "\nВывод объекта A\n";

cout << A << endl;

cout << "\nИнициализация объекта В\n";

Pair B(4, 1.98);

cout << "\nВывод объекта В\n";

cout << B << endl;

cout << "\nИзменение объекта B\n";

for (int i = 0; i < 10; i++)

{

B++; // постфиксная форма

}

cout << "\nВывод объекта В\n";

cout << B << endl;

Pair C;

cout << "\nИнициализация объекта C\n";

cout << "Введите a: ";

int a;

cin >> a;

cout << "Введите b: ";

double b;

cin >> b;

C.set\_a(a);

C.set\_b(b);

cout << "\nКопирование полей объекта C в объект D\n";

Pair D = C;

cout << "\nОбъект D\n" << D << endl << "Объект С\n" << C << endl;

if (A < B)

{

cout << "\nОбъект A меньше объекта B\n";

}

else

{

if (A > B)

{

cout << "\nОбъект A больше объекта B\n";

}

else

{

cout << "\nОбъект A равен объекту B\n";

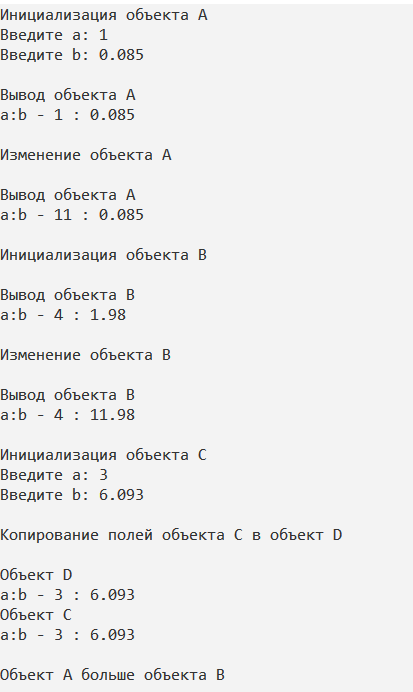
}

}

}

# **Приложение Б**

**Результаты выполнения программы**



**Ответы на вопросы**

1. Для чего используются дружественные функции и классы?

Дружественные функции применяются для доступа к скрытым полям класса и представляют собой альтернативу методам. Дружественный класс применяется для доступа методов одного класса к скрытым полям другого.

1. Сформулировать правила описания и особенности дружественных функций.

* Объявление дружественной функции – внутри нужного класса, ключевым словом friend. Передаваемый параметр - объект или ссылка на объект.
* Дружественная функция – обычная функция или метод другого класса. Не важно, где функция описана в классе + спецификаторы доступа не распространяются.
* Одна и та же дружественная функция может быть в нескольких классах.

*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:

* Компонентная функция класса
* Внешняя (глобальная) функция

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();

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

Унарная функция-операция, определяемая внутри класса, не имеет операндов.

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

Унарная функция-операция, определяемая вне класса, имеет один операнд – ссылка на вызвавший её объект.

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

Бинарная функция-операция, определяемая внутри класса, имеет один операнд – переменная, хранящая значение.

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

Бинарная функция-операция, определяемая вне класса, имеет два операнда. Первый – вызвавший её объект, второй – переменная, хранящая значение.

1. Чем отличается перегрузка префиксных и постфиксных унарных операций?

Перегрузка префиксной унарной операции не имеет передаваемых параметров, перегрузка постфиксной унарной операции имеет один передаваемый параметр – показатель того, что это постфиксная форма.

1. Каким образом можно перегрузить операцию присваивания?

Перегрузка операции присваивания описывается как метод класса. В качестве принимаемого аргумента используется ссылка на присваиваемый объект, в качестве возвращаемого значения – ссылку на объект, для которого она вызвана.

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();

1. Что должна возвращать операция присваивания?

Операция присваивания возвращает ссылку на объект, для которого она вызвана.

1. Каким образом можно перегрузить операции ввода-вывода?

Перегрузка операций ввода-вывода описываются как дружественные функции, т. к. левый операнд – потоки ввода-вывода.

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?

Компилятор воспринимает данный вызов как перегруженную функцию-операцию префиксного инкремента, определенную внутри класса.

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?

Компилятор воспринимает данный вызов как дружественную перегруженную функцию-операцию префиксного инкремента.

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?

Компилятор воспринимает данный вызов как перегруженную функцию-операцию оператора сравнения, определенную внутри класса.

1. В программе описан класс

class Student

{

……

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

……

};

И определён объект этого класса

Student a, b;

Выполняется операция

cout << a > b;

Каким образом, компилятор будет воспринимать вызов функции-операции?

Компилятор воспринимает данный вызов как дружественную перегруженную функцию-операцию сравнения.