**Chapter 18 Notes**

**Alex Yang**

**(5/17/2021)**

1. ostream in C++

#include <iostream>

using namespace std;

int main()

{

ostream &os = cout;

os<<"Testing"<<"\n";

cout<<"Hello World";

return 0;

}

#include<iostream>

using std::cout;

using std::endl;

using std::ostream;

// class CPoint{ //Error example

// public:

// CPoint(int x\_,int y\_):x(x\_),y(y\_){}

// int x,y;

// };

class CPoint

{

public:

CPoint(int x\_,int y\_):x(x\_),y(y\_){}

friend ostream & operator <<(ostream & os,const CPoint & p){

return os << "x = "<<p.x << " y = "<< p.y << endl;

}

int x,y;

};

int main(void){

CPoint point(1,2);

cout << point; //redefine “<<” as follows

//(cout << point)=> calling ostream & operator <<(cout, point)

return 0;

}

*//-------------------------------------------------*

#include <iostream>

using namespace std;

class Box{

public:

Box(int da){data = da;}

void output(ostream& out) const;

private:

int data;

};

void Box :: output(ostream& out) const{

out << "Operator overloading: ";

out << data << endl;

}

ostream& operator<<(ostream& out, const Box& x){

x.output(out);

return out;

}

int main(void){

Box box(10);

cout << box;

return 0;

}

*//-------------------------------------------------*

#include<iostream>

using std::cout;

class Box{

public:

Box(int da){data = da;}

Box operator+(const Box& b){

Box c(0);

c.data = data + b.data;

return c;

}

int getData(){ return data;}

private:

int data;

};

int main(void){

Box a(10);

Box b(15);

Box c(0);

c = a + b; //a+b ==> a.operator+(b);

cout << c.getData()<<"\n";

return 0;

}

*//-------------------------------------------------*

#include <iostream>

using namespace std;

class person{

private:

int age;

public:

person(int nAge){

this->age = nAge;

}

bool operator==(const person& ps){

if (this->age == ps.age){

return true;

}

return false;

}

};

int main(void){

person p1(10);

person p2(10);

if (p1 == p2){ //call p1.operator==(p2)

cout << "p1 is equal with p2." << endl;

}

else{

cout << "p1 is not equal with p2." << endl;

}

return 0;

}

*//-------------------------------------------------*

#include <iostream>

using namespace std;

class person{

public:

int age;

};

// The left operand must be specified explicitly

bool operator==(person const& p1 ,person const& p2){

if (p1.age == p2.age){

return true;

}

else{

return false;

}

}

int main(void){

person p1;

person p2;

p1.age = 18;

p2.age = 18;

if (p1 == p2){ //call: operator==(p1, p2)

cout << "p1 is equal with p2." << endl;

}

else{

cout << "p1 is NOT equal with p2." << endl;

}

return 0;

}

*//-------------------------------------------------*

#include<iostream>

using std::cout;

class upDate{

public:

upDate(int d){data=d;}

upDate operator+(int x){

this->data += x;

return \*this;

}

int get(){return data;}

private:

int data;

};

int main(void){

int a=123;

upDate b(456),c(0);

c=b+a; //b.operator+(a); and b.operator+(a)’s return is b

//c=a+b; //Error

cout<<"c.day="<<c.get()<<"\n";

return 0;

}

1. explicit keyword in C++

//Example.h

#pragma once

class CExample{

public:

CExample(void);

public:

~CExample(void);

public:

int m\_iFirst;

int m\_iSecond;

public:

//explicit CExample(int iFirst, int iSecond = 4);

CExample(int iFirst, int iSecond = 4);

};

//Example.cpp

#include "Example.h"

CExample::CExample(void): m\_iFirst(0){}

CExample::~CExample(void){}

CExample::CExample(int iFirst, int iSecond):m\_iFirst(iFirst), m\_iSecond(iSecond)

{}

//main.h

#include "Example.h"

int main(void){

CExample objOne;

CExample objTwo(12, 12);

CExample objThree(12);

//CExample objFour(12);

CExample objFour=12; //like CExample temp(12); objFour(temp)

return 0;

}

1. Converting between Types

#include <iostream>

class NewInt{

int data;

public:

NewInt(int val=0){

data = val;

}

// Note that conversion-type-id "int" is the implied return type.

// Returns by value so "const" is a better fit in this case.

operator int() const{

return data;

}

};

int main(void){

NewInt A(10);

std::cout << int(A);

return 0;

}

*//-------------------------------------------------*

#include <iostream>

// operator Type() cast operator

// operator int()

// operator double()

// ...

//////////////////////////////////////////////////////////

class Rectangle{

public:

Rectangle(const int w, const int h): width(w), height(h){};

~Rectangle() {};

operator int(); //after casting int(), Rectangle is like int

public:

int width;

int height;

};

//////////////////////////////////////////////////////////

Rectangle::operator int(){

return width \* height;

}

//////////////////////////////////////////////////////////

void printInt(const int v) {

std::cout << v+5 << std::endl;

}

int main(void){

Rectangle a(40, 10);

int v1 = a; // 400

int v2 = 1 + a; // 401

int v3 = static\_cast<int>(a); // 400

std::cout << a << std::endl; // output: 400

printInt(a); // output: 405

return 0;

}

1. Overloading ++ and –

#include<iostream>

using namespace std;

class Test {

friend Test & operator--(Test &obj);

friend Test operator--(Test &obj, int);

public:

Test(int a = 0, int b = 0){

this->a = a;

this->b = b;

}

void display(){

cout << "a:" << a << " b:" << b << endl;

}

public:

//prefix ++

Test & operator++(){ // or Test & operator++(0)

this->a++;

this->b++;

return \*this;

}

//postfix ++

Test operator++(int){

Test temp = \*this;

++\*this;

return temp;

}

private:

int a;

int b;

};

//prefix --

Test & operator--(Test &obj){

obj.a--;

obj.b--;

return obj;

}

//postfix --

Test operator--(Test &obj,int){ //adding int is to differentiate postfix from prefix

Test temp = obj;

--obj;

return temp;

}

int main(void){

Test t1(1, 2);

t1.display();

++t1;

t1.display();

--t1;

t1.display();

Test t2(3, 4);

t2.display();

t2++;

t2.display();

t2--;

t2.display();

cout << "hello world!\n";

return 0;

}