**运算符重载**

#include <iostream>

#include <string>

#ifndef Pan\_H

#define Pan\_H

class Pan

{

friend Pan operator+(Pan &left, Pan &right);

friend std::ostream &operator<<(std::ostream &left, Pan &right);

friend std::istream &operator>>(std::istream &left, Pan &right);

private:

/\* data \*/

int \_num = 0;

public:

Pan(/\* args \*/) : Pan(0){};

Pan(int num) : \_num(num){};

~Pan();

// 成员函数重载运算符，这里返回类型使用值，因为我们不希望 pan1 + pan2 会改变 pan1 的值

Pan operator+(Pan &right)

{

Pan sum = \*this;

sum += right; // 使用类的复合运算符

return sum;

};

// 一般定义了算术运算符也会定义复合运算符

Pan &operator+=(Pan &right)

{

this->\_num += right.\_num;

return \*this;

}

// 重载下标运算符

std::string operator[](std::size\_t n);

// 前置递增

Pan &operator++();

// 后置递增

Pan &operator++(int);

// 重载 \* 运算符， \*pan 可以使用

std::string &operator\*() const;

// 重载函数调用运算符，对象可以像函数一样使用，pan(10) 可以使用

// 重载了函数运算符的对象叫做 函数对象，其中我们声明的 lambda 就是匿名的函数对象

int operator()(int num) const{

return num + this->\_num;

}

// 重载类型转换，bool 是转换的值

// 使用 explicit 是因为我们不希望系统帮我们隐式，转换但有一个特例，在 if，while中系统会帮我们隐式转换

explicit operator bool() const {

return this->\_num != 0;

}

// 同理还有 -> 的重载，但我们不希望重载他们，这样会让别人误解

};

Pan::Pan(/\* args \*/)

{

}

Pan::~Pan()

{

}

// 非成员函数重载运算符

Pan operator+(Pan &left, Pan &right)

{

return Pan(left.\_num + left.\_num);

};

// 重载 << 运算符， cout << pan 可以使用了

std::ostream &operator<<(std::ostream &left, Pan &right)

{

left << right.\_num;

return left;

};

// 重载 >> 运算符

std::istream &operator>>(std::istream &left, Pan &right)

{

left >> right.\_num;

if (!left)

{

right = Pan();

}

return left;

}

// 比较运算符

bool operator==(Pan &left, Pan &right);

#endif

**可调用对象域function**

#include <iostream>

#include <vector>

#include <functional>

using namespace std;

int sum(int l, int r);

int main()

{

// 可调用对象与 function

// c++中几种可调用对象：函数，函数指针，lambda表达式，bind创建的对象，重载了函数调用运算符的对象

// 调用形式：函数 int sum(int, int) 和 lambda [](int l, int r){ return l + r } 具有相同的调用形式 int(int, int)

// 虽然这些可调用对象具有相同的调用形式，但他们却不能相互转换，即我们不能将他们放到一个容器中

// function 可以解决此问题

vector<function<int(int, int)>> list;

list.push\_back(sum);

list.push\_back([](int l, int r){ return l + r; });

function<int(int, int)> fun1([](int l, int r){ return l + r; });

fun1(1, 2);

cout << "enter key" << endl;

cin.get();

return 0;

}