C++中，重载运算符是具有特殊名称的函数。

运算符就是操作符。

重载操作符的一般形式：

返回值 operator 操作符 (参数表)

例如重载操作符->

例：程序operator\_test1

base.h

#ifndef BASE\_H

#define BASE\_H

class Base

{

public:

Base\* operator->(); // 重载->运算符

void func();

};

#endif // BASE\_H

base.cpp

#include "base.h"

#include <iostream>

using std::cout;

using std::endl;

Base\* Base::operator ->()

{

cout << "Base 重载的->运算符" << endl;

return this;

}

void Base::func()

{

cout << "func()" << endl;

}

#include <iostream>

using std::cout;

using std::endl;

#include "base.h"

int main(int argc, char\* argv[])

{

Base b;

b->func(); // 可以这样调用，相当于b.operator ->()->func()

return 0;

}

除了函数调用操作符”()”之外，重载操作符的形参数目（包括成员函数的隐式this指针）与操作符的操作数数目相同，就是说除了函数调用操作符之外，重载操作符的形参数目只能是1个或2个，因为C++中的三目运算符”?:”是不能重载的。

重载操作符要注意：

1. 重载操作符必须具有至少一个类类型或枚举类型的形参，就是说重载操作符的形参的类型不能都是内置类型的，如int operator + (int, int)是非法的。
2. 重载操作符时，除了函数调用操作符”()”外，使用默认实参是非法的。这主要针对有些操作符既可作一元操作符又可作二元操作符。
3. 重载操作符的优先级和结合性是固定的。
4. 大多数重载操作符可以定义为普通非成员函数或类成员函数。这两者是有区别的：
   1. 定义为普通非成员函数，重载操作符的形参数目与操作符的操作数数目相同，即重载单目操作符有一个形参，重载双目操作符有两个形参。
   2. 定位为类成员函数，重载操作符的形参数目看起来比操作符的操作数数目少1个，原因是成员函数有一个隐含的形参this指针，限定为第一个操作数（左操作数）。即重载单目操作符没有显式的形参，重载双目操作符有一个形参。

例：

// 重载操作符为类成员函数

Sales\_item& Sales\_item :: operator += (const Sales\_item&);

// 重载操作符为普通非成员函数

Sales\_item operator + (const Sales\_item&, const Sales\_item&);

一般将算术和关系操作符定义为普通非成员函数，将赋值运算符定义为类成员函数。

重载输入操作符”<<”和输出操作符”>>”时，最好把它们定义为普通非成员函数，这是为了与标准库iostream规范的输入或输出操作符相一致。标准的输出操作符的左操作数为ostream&，输入操作符的左操作数为istream&。如果将重载输入操作符或输出操作符定义为类成员函数，其第一个形参为隐含的this指针，则其左操作数为该类的类对象。

1. 重载操作符定义为非成员函数时，通常把它们设置为所操作类的友元。
2. 重载操作符时，赋值（=），下标（[ ]），调用（()），成员访问箭头（->），必须定义为类成员函数，否则编译会出错。
3. 赋值操作符(=)必须返回类的引用，即\*this。
4. 复合赋值操作符通常定义为类的成员函数，但不是必须地。
5. 自增自减操作符通常定义为类的成员函数。
6. 算术操作符、关系操作符、位操作符和输入输出操作符通常定义为非成员函数。

转换与类类型

在C++学习7-类的构造函数.docx中，可以将其他类型隐式转换为类类型，前提是不用explicit关键字去定义构造函数。

我们可以定义类的转换操作符，将类类型转换为其他类型。

转换操作符是特殊的类成员函数，通常采用如下形式：

operator type();

其中，type表示内置类型名、类类型名或由类型别名定义的名字。对于任何可以作为函数返回类型的类型（void除外）都可以定义转换函数。一般而言，不允许转换为函数或数组类型。

转换函数必须是成员函数，不能指定返回类型，并且形参表必须为空。转换操作符通常定义为const成员。

例：程序class\_test25

// 转换与类类型

#include <iostream>

class SmallInt

{

public:

SmallInt(int i = 0): val(i)

{

if ((i < 0) || (i > 255))

{

throw std::out\_of\_range("Bad SmallInt initializer"); // 超出范围

}

}

public:

operator int() const // 转换操作符，将SmallInt转换为int型

{

return val;

}

private:

std::size\_t val;

};

int main(int argc, char\* argv[])

{

SmallInt si = 3;

double f = si + 3.14; // 将si转换为int，然后再转换为double

if (si >= f)

{

}

if (si)

{

}

return 0;

}

只要存在转换，编译器可以在使用内置转换的地方自动调用它。

类类型转换和标准类型转换

1. 将类类型转换为标准类型后，还可以继续转换，如：

SmallInt si = 3;

double f = si + 3.14; // 将si转换为int，然后再转换为double

1. 将类类型转换为另一个类类型后，不允许再进行转换。

转换可能带来的二义性：

例：程序class\_test26

// 类类型与转换

#include <iostream>

class SmallInt

{

public:

SmallInt(int i = 0): val(i)

{

}

SmallInt(double f) : val(f)

{

}

public:

operator int() const // 转换操作符，将SmallInt转换为int型

{

return val;

}

operator double() const

{

return val;

}

private:

std::size\_t val;

};

void compute(int i)

{

}

void compute(double f)

{

}

int main(int argc, char\* argv[])

{

SmallInt si;

// compute(si); // error，重载函数调用存在二义性，SmallInt既可以转换为int，也可以转换为double

compute(static\_cast<int>(si)); // ok，强制性消除二义性

}