

ECE 462

Object-Oriented Programming

using C++ and Java

Operator Overloading in C++

Yung-Hsiang Lu
yunglu@purdue.edu

Operator Overloading in C++

You have been using overloaded operators for years.

- $3 + 5$ // integer addition (bit-wise + carries)
- $3.5 + 0.0059$ // floating point addition
 - // 1. align the decimal points
 - // 2. add significands
 - // 3. normalized and update exponent
- “hello” + “ world” // intuitively, it means append
- Operator overloading is *not essential* in object-oriented programming. In fact, Java does not allow programmer-defined operator overloading.

Overloading Operators in C++

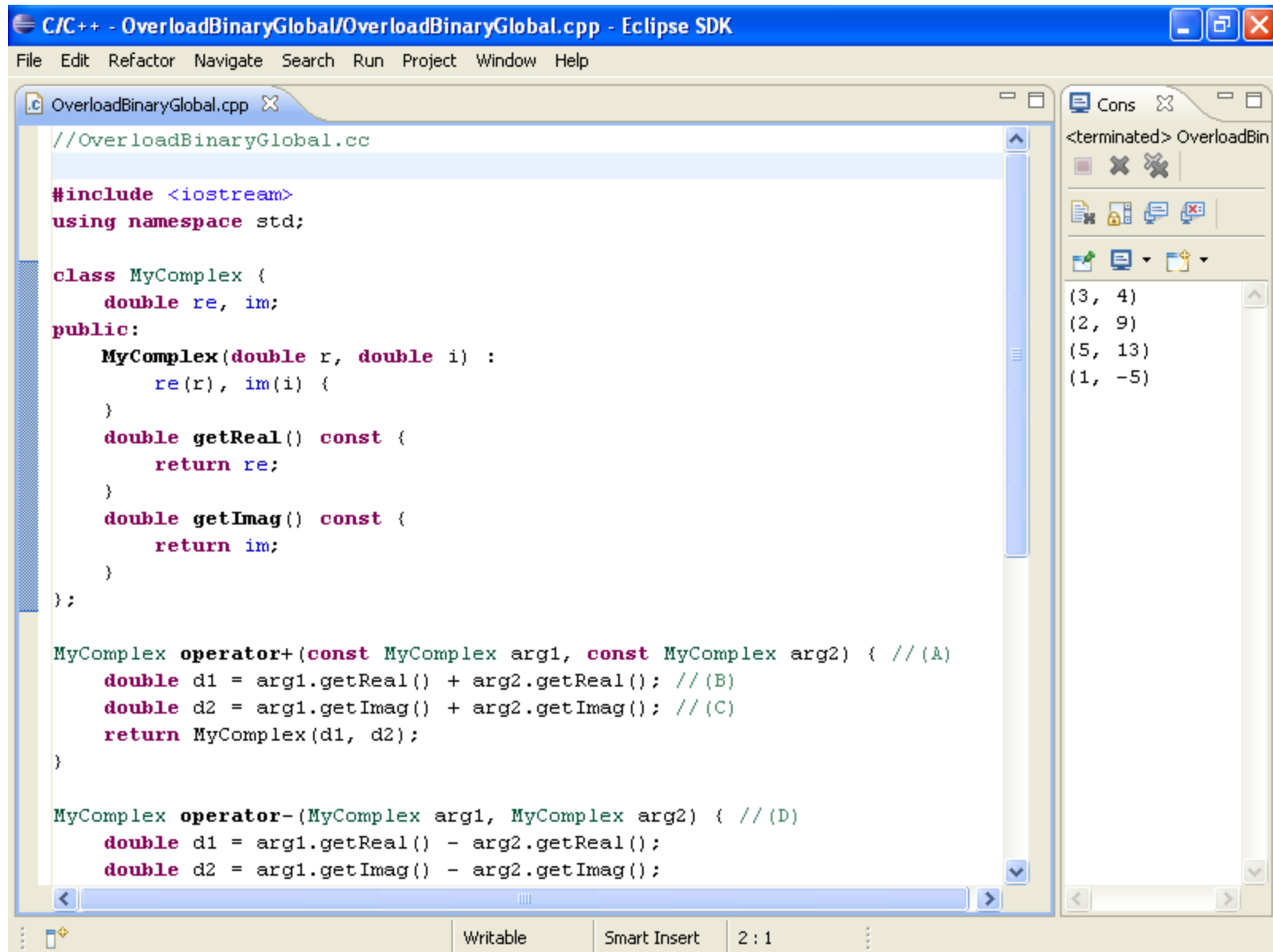
- We have seen overloaded operator for << >> and =
- at least one operand (for binary operators) must be an object or enumeration type (i.e. not a built-in type)
- precedence not changed
- arity not changed (! always unary)
- argument(s) may be passed by value (copy) or by reference, **not** by pointer
- default argument value(s) illegal
- cannot overload :: . * . ?:

Binary Operators

- If the first operand is an object, a binary operator can be implemented in two forms
 - a member function, the first operand is *this* object
 - a “free” function (not a member of any class), usually declared as a **friend** to access private attributes
 - not both
- If the first operand is not an object (such as int), the operator must be a free function.
- In most cases, the operand object(s) should use reference to prevent calling copy constructor.

Global Function

(not a member function of a class)



```
C/C++ - OverloadBinaryGlobal/OverloadBinaryGlobal.cpp - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help

OverloadBinaryGlobal.cpp

    double d2 = arg1.getImag() + arg2.getImag(); // (C)
    return MyComplex(d1, d2);
}

MyComplex operator-(MyComplex arg1, MyComplex arg2) { // (D)
    double d1 = arg1.getReal() - arg2.getReal();
    double d2 = arg1.getImag() - arg2.getImag();
    return MyComplex(d1, d2);
}

ostream& operator<<(ostream& os, const MyComplex& arg) { // (E)
    double d1 = arg.getReal();
    double d2 = arg.getImag();
    os << "(" << d1 << ", " << d2 << ")" << endl;
    return os;
}

int main() {
    MyComplex first(3, 4);
    MyComplex second(2, 9);

    cout << first; // (3, 4)
    cout << second; // (2, 9)
    cout << first + second; // (5, 13)
    cout << first - second; // (1, -5)
    return 0;
}

Cons
<terminated> OverloadBin
(3, 4)
(2, 9)
(5, 13)
(1, -5)

Writable Smart Insert 2:1
```

Copy Constructor and Operator Overloading


```
C/C++ - OverloadBinaryGlobal/OverloadBinaryGlobal.cpp - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help

OverloadBinaryGlobal.cpp
//OverloadBinaryGlobal.cc

#include <iostream>
using namespace std;
int copyConstructorCounter = 0;
class MyComplex {
    double re, im;
public:
    MyComplex(double r, double i) :
        re(r), im(i) {
    }
    MyComplex(const MyComplex & orig) {
        // unnecessary since there is no pointer
        // used to show when copy constructor is called
        cout << "MyComplex(const MyComplex & orig)" << endl;
        copyConstructorCounter ++;
        re = orig.re;
        im = orig.im;
    }
    double getReal() const {
        return re;
    }
    double getImag() const {
        return im;
    }
};

MyComplex operator+(const MyComplex arg1, const MyComplex arg2) { //(A)
```

The screenshot shows the Eclipse IDE interface. The top window displays the source code for `OverloadBinaryGlobal.cpp`. The code defines a `MyComplex` class (partially visible), an overloaded `<<` operator for `ostream`, and a `main` function. The `main` function creates two `MyComplex` objects, `first` (3, 4) and `second` (2, 9), and prints their values and the result of `first + second` and `first - second`. It also prints the value of `copyConstructorCounter`.

```
ostream& operator<<(ostream& os, const MyComplex& arg) { //(E)
    double d1 = arg.getReal();
    double d2 = arg.getImag();
    os << "(" << d1 << ", " << d2 << ")" << endl;
    return os;
}

int main() {
    MyComplex first(3, 4);
    MyComplex second(2, 9);

    cout << first; // (3, 4)
    cout << second; // (2, 9)
    cout << first + second; // (5, 13)
    cout << first - second; // (1, -5)
    cout << "copyConstructorCounter = " << copyConstructorCounter << endl;
    return 0;
}
```

The bottom window shows the console output. The first line is `<terminated> OverloadBinaryGlobal.exe [C/C++ Local Application] C:\yunglu\eclipse\workspace\OverloadBinaryGlobal\Debug\OverloadBinaryGlobal.exe (5/19/08 1`. The second line is `(1, -5)`. The third line is `copyConstructorCounter = 4`, with a yellow arrow pointing to the number 4.

Member Function (first operand is "this")

```
C/C++ - OverloadBinaryMember/OverloadBinaryMember.cc - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help

OverloadBinaryMember.cc
//OverloadBinaryMemb.cc (modified)
#include <iostream>
using namespace std;
int copyConstructorCounter = 0;
class MyComplex {
    double re, im;
public:
    MyComplex(double r, double i) :
        re(r), im(i) {
    }
    MyComplex(const MyComplex & orig) {
        // unnecessary since there is no pointer
        // used to show when copy constructor is called
        cout << "MyComplex(const MyComplex & orig)" << endl;
        copyConstructorCounter ++;
        re = orig.re;
        im = orig.im;
    }
    // getReal and getImag unnecessary
    double getReal() const {
        return re;
    }
    double getImag() const {
        return im;
    }
    MyComplex operator+(MyComplex) const;
    MyComplex operator-(MyComplex) const;
    // ostream& operator<< ( const MyComplex& ); // WRONG

```

```
C/C++ - OverloadBinaryMember/OverloadBinaryMember.cc - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help

OverloadBinaryMember.cc x
// ostream& operator<< ( const MyComplex& ); // WRONG
friend ostream& operator<<(ostream&, const MyComplex&);
};

/*
MyComplex operator+(const MyComplex & arg1, const MyComplex & arg2) {
double d1 = arg1.getReal() + arg2.getReal();
double d2 = arg1.getImag() + arg2.getImag();
return MyComplex(d1, d2);
}

MyComplex operator-(MyComplex & arg1, MyComplex & arg2) {
double d1 = arg1.getReal() - arg2.getReal();
double d2 = arg1.getImag() - arg2.getImag();
return MyComplex(d1, d2);
}

ostream& operator<<(ostream& os, const MyComplex& arg) {
double d1 = arg.getReal();
double d2 = arg.getImag();
os << "(" << d1 << ", " << d2 << ")" << endl;
return os;
}
*/

MyComplex MyComplex::operator+( const MyComplex arg ) const {
double d1 = re + arg.re;
double d2 = im + arg.im;
```

binary operator with only one parameter



```
C/C++ - OverloadBinaryMember/OverloadBinaryMember.cc - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help

OverloadBinaryMember.cc

    double d1 = re + arg.re;
    double d2 = im + arg.im;
    return MyComplex( d1, d2 );
}

MyComplex MyComplex::operator-( const MyComplex arg ) const {
    double d1 = re - arg.re;
    double d2 = im - arg.im;
    return MyComplex( d1, d2 );
}

ostream& operator<<(ostream& os, const MyComplex& c) {
    os << "(" << c.re << ", " << c.im << ")" << endl;
    return os;
}

int main() {
    MyComplex first(3, 4);
    MyComplex second(2, 9);

    cout << first; // (3, 4)
    cout << second; // (2, 9)
    cout << first + second; // (5, 13)
    cout << first - second; // (1, -5)
    cout << "copyConstructorCounter = "
         << copyConstructorCounter << endl;
    return 0;
}

Writable Smart Insert 7 : 8
```

The screenshot shows the Eclipse IDE with a C++ project named 'OverloadBinaryMember'. The editor displays the source code for 'OverloadBinaryMember.cc', which defines a 'MyComplex' class with real ('re') and imaginary ('im') parts. The code implements the addition (+), subtraction (-), and output stream (<<) operators. A 'main' function creates two 'MyComplex' objects, 'first' (3, 4) and 'second' (2, 9), and prints their values and the results of 'first + second' and 'first - second'. It also prints the 'copyConstructorCounter', which is set to 2.

```
double d1 = re + arg.re;
double d2 = im + arg.im;
return MyComplex( d1, d2 );
}

MyComplex MyComplex::operator-( const MyComplex arg ) const
{
    double d1 = re - arg.re;
    double d2 = im - arg.im;
    return MyComplex( d1, d2 );
}

ostream& operator<<(ostream& os, const MyComplex& c) {
    os << "(" << c.re << ", " << c.im << ")" << endl;
    return os;
}

int main() {
    MyComplex first(3, 4);
    MyComplex second(2, 9);

    cout << first; // (3, 4)
    cout << second; // (2, 9)
    cout << first + second; // (5, 13)
    cout << first - second; // (1, -5)
    cout << "copyConstructorCounter = "
         << copyConstructorCounter << endl;
    return 0;
}
```

The console window shows the output of the program, which matches the comments in the code: (3, 4), (2, 9), MyComplex(const MyComplex & orig) (5, 13), MyComplex(const MyComplex & orig) (1, -5), and copyConstructorCounter = 2.

Binary Operator, One Input?

```
first + second;
```

can be thought of as

```
first.operator + (second);
```

This operator is a member function. Hence, the first operand is given by the first object.

```
cout << first;
```

can be thought of as

```
cout.operator << (first);
```


Unary Operator, Global Function

```
C/C++ - OverloadUnaryGlobal/OverloadUnaryGlobal.cc - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help

OverloadUnaryGlobal.cc
//OverloadUnaryGlobal.cc
#include <iostream>
using namespace std;
class MyComplex {
    double re, im;
public:
    MyComplex( double r, double i ) : re(r), im(i) {}
    double getReal() const { return re; }
    double getImag() const { return im; }
};
// global overload definition for "-" as a unary operator
MyComplex operator-( const MyComplex arg ) {
    return MyComplex( -arg.getReal(), -arg.getImag() );
}
// global overload definition for "<<" as a binary operator
ostream& operator<< ( ostream& os, const MyComplex& arg ) {
    os << "(" << arg.getReal() << ", " << arg.getImag() << ")" << endl;
    return os;
}
int main()
{
    MyComplex c(3, 4);
    cout << c;           // (3, 4)
    cout << -c;         // (-3, -4)
    return 0;
}
```

unary operator with only one parameter



Unary Operator, Member Function

```
C/C++ - OverloadUnaryGlobal/OverloadUnaryMember.cc - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help

OverloadUnaryMember.cc
//OverloadUnaryMemb.cc
#include <iostream>
using namespace std;
class MyComplex {
    double re, im;
public:
    MyComplex( double r, double i ) : re(r), im(i) {}
    MyComplex operator-() const;
    friend ostream& operator<< ( ostream&, const MyComplex& );
};

//Member-function overload definition for "-"
MyComplex MyComplex::operator-() const { // (A)
    return MyComplex( -re, -im );
}

//This overload definition has to stay global
ostream& operator<< ( ostream& os, const MyComplex& c ) {
    os << "(" << c.re << ", " << c.im << ")" << endl;
    return os;
}

int main()
{
    MyComplex c(3, 4);
    MyComplex z = -c; // (B)
    cout << z << endl; // (-3, -4)
    return 0;
}
```

unary operator without parameter ("this" is the parameter)



Comparison Operator <

The screenshot shows the Eclipse IDE interface. The main editor window displays the source code for `CompareObject.cc`. The code defines a `Student` class with a `string sName` member and a `<` operator that compares the `sName` members of two `Student` objects. The `main` function creates four `Student` objects: `s1("John")`, `s2("Mary")`, `s3("Amy")`, and `s4("Tom")`. It then prints the results of three comparisons: `s1 < s2` (true), `s1 < s3` (false), and `s3 < s4` (true). The console window on the right shows the output of the program, which matches the comments in the code: `1`, `0`, and `1` on separate lines.

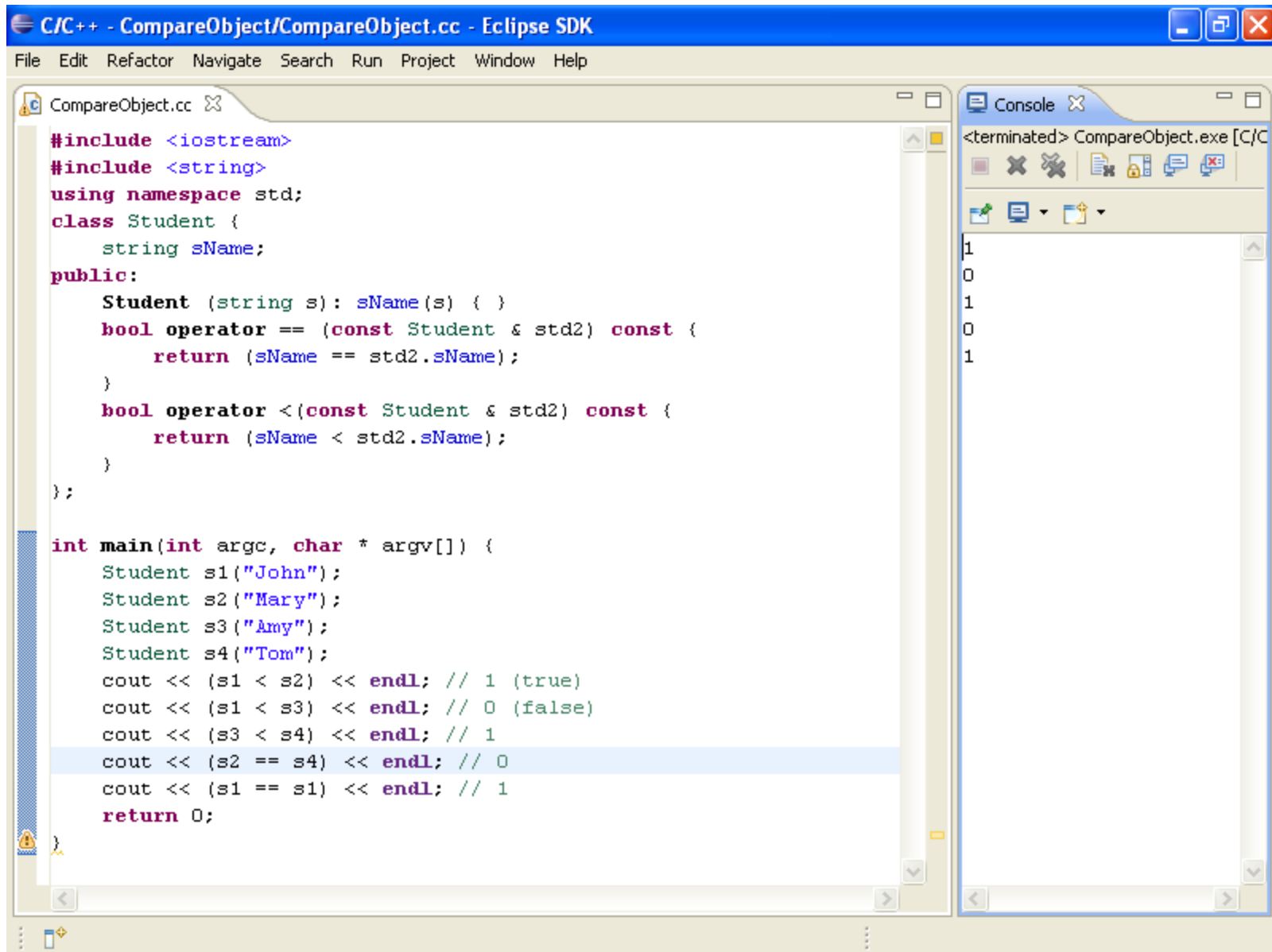
```
#include <iostream>
#include <string>
using namespace std;
class Student {
    string sName;
public:
    Student (string s): sName(s) { }
    bool operator <(const Student & std2) const {
        return (sName < std2.sName);
    }
};

int main(int argc, char * argv[]) {
    Student s1("John");
    Student s2("Mary");
    Student s3("Amy");
    Student s4("Tom");
    cout << (s1 < s2) << endl; // 1 (true)
    cout << (s1 < s3) << endl; // 0 (false)
    cout << (s3 < s4) << endl; // 1
    return 0;
}
```

Console Output:

```
<terminated> CompareObject.exe [C/C
1
0
1
```

Equality Operator ==



Self Test

ECE 462
Object-Oriented Programming
using C++ and Java

Small Int and Conversion

Yung-Hsiang Lu
yunglu@purdue.edu

Increment and Decrement Operators (Small Int)

```
//prefix increment operator, such as ++i
inline SmallInt& SmallInt::operator++() {
    if (value < MAX) { value ++; }
    else { cerr << "Error: Range of SmallInt violated" << endl; }
    return *this;
}
```

```
//postfix increment operator, , such as i++
inline const SmallInt SmallInt::operator++(int) {
    SmallInt oldValue = *this;
    ++(*this); // call the prefix increment operator
    return oldValue;
}
```

prefix and postfix increment

```
void f1(int i) {  
    cout << i << endl;  
}  
int x = 5;  
f1(++x); // increment before calling f1, output 6  
cout << x << endl; // 6  
f1(x ++); // increment after calling f1, output 6  
cout << x << endl; // 7
```

inline

- inline \Rightarrow **suggest** C++ compiler to replace the function call by the function body to reduce the call overhead
- function call overhead
 - push the current location to the call stack
 - push the parameters' values to the call stack
 - change the program counter to the called function's address
 - execute the code in the function // useful work
 - pop the stack to get the return address
 - change the program counter to the return address
- inline replaces the call by the code inside the function

```
C/C++ - SmallInt/SmallInt.cc - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help

*SmallInt.cc
//SmallIntWithIncrDecr.cc (modified)

#include <iostream>
#include <cstdlib>
using namespace std;
class SmallInt {
    int value;
    bool rangeCheck(int i);
public:
    static const int MAX = 255;
    static const int MIN = 0;
    explicit SmallInt(int ival = 0); // (A)
    // explicit suppress implicit conversion by the compiler
    SmallInt(const SmallInt& other);
    SmallInt& operator=(const SmallInt& other);
    SmallInt& operator=(const int i);
    //unsafe implicit conversion:
    operator int() const {
        cout << "operator int() const {" << endl;
        // implicit conversion
        return value;
    } // (B)
    SmallInt& operator++();
    SmallInt& operator--();
    const SmallInt operator++(int);
    const SmallInt operator--(int);
    friend ostream& operator<<(ostream& os, const SmallInt& s);
};

Writable Smart Insert 1:1
```

```
C/C++ - SmallInt/SmallInt.cc - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help

*SmallInt.cc

};

//constructor:
inline SmallInt::SmallInt(int ival) {
    cout << "SmallInt::SmallInt(int ival) {" << endl;
    if (rangeCheck(ival)) {
        value = ival;
    } else {
        cout << "Error: Range of SmallInt violated" << endl;
    }
}

//copy constructor:
inline SmallInt::SmallInt(const SmallInt& other) {
    cout << "SmallInt::SmallInt(const SmallInt& other) {" << endl;
    value = other.value;
}

//copy assignment operator:
inline SmallInt& SmallInt::operator=( const SmallInt& other ) {
    cout << "SmallInt& SmallInt::operator=( const SmallInt& other ) {" << endl;
    if ( this != & other ) {
        value = other.value;
    }
    return *this;
}

//assignment from an int:

Writable Smart Insert 1:1
```


The screenshot shows the Eclipse IDE window titled "C/C++ - SmallInt/SmallInt.cc - Eclipse SDK". The menu bar includes "File", "Edit", "Refactor", "Navigate", "Search", "Run", "Project", "Window", and "Help". The editor window shows the following C++ code:

```
//assignment from an int:
inline SmallInt& SmallInt::operator=( const int i ) {
    cout << "SmallInt& SmallInt::operator=( const int i ) {" << endl;
    if (rangeCheck(i)) {
        value = i;
    } else {
        cout << "Error: Range of SmallInt violated" << endl;
    }
    return *this;
}

//prefix increment operator:
inline SmallInt& SmallInt::operator++() {
    cout << "SmallInt& SmallInt::operator++() {" << endl;
    if (value < MAX) {
        value ++;
    } else {
        cout << "Error: Range of SmallInt violated" << endl;
    }
    return *this;
}

//prefix decrement operator:
inline SmallInt& SmallInt::operator--() {
    cout << "SmallInt& SmallInt::operator--() {" << endl;
    if (value > MIN) {
        value --;
    } else {
```

The IDE status bar at the bottom shows "Writable", "Smart Insert", and "1 : 1".

The screenshot shows the Eclipse IDE window titled "C/C++ - SmallInt/SmallInt.cc - Eclipse SDK". The menu bar includes File, Edit, Refactor, Navigate, Search, Run, Project, Window, and Help. The editor window displays the following C++ code:

```
//prefix decrement operator:
inline SmallInt& SmallInt::operator--() {
    cout << "SmallInt& SmallInt::operator--() {" << endl;
    if (value > MIN) {
        value --;
    } else {
        cout << "Error: Range of SmallInt violated" << endl;
    }
    return *this;
}

//postfix incretor:
inline const SmallInt SmallInt::operator++(int) {
    cout << "SmallInt SmallInt::operator++(int) {" << endl;
    SmallInt oldValue = *this;
    ++(*this);
    return oldValue;
}

//postfix decretor:
inline const SmallInt SmallInt::operator--(int) {
    cout << "SmallInt SmallInt::operator--(int) {" << endl;
    SmallInt oldValue = *this;
    --(*this);
    return oldValue;
}
```

The status bar at the bottom shows icons for a toolbar, a list of icons, and the text "Writable Smart Insert 1 : 1".

```
C/C++ - SmallInt/SmallInt.cc - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help

SmallInt.cc

//range check function:
inline bool SmallInt::rangeCheck(int i) {
    cout << "SmallInt::rangeCheck(int i) {" << endl;
    if ((i < MIN) || (i > MAX)) {
        return false;
    }
    return true;
}

//overload for the output stream operator:
ostream& operator<<(ostream& os, const SmallInt& s) {
    os << s.value;
    return os;
}

int main() {
    SmallInt si( 3);
    cout << si + 3.14159 << endl << endl; // 6.14159
    cout << ++si << endl << endl; // 4
    cout << si++ << endl << endl; // 4
    cout << si << endl << endl; // 5
    cout << --si << endl << endl; // 4
    cout << si << endl << endl; // 4
    cout << si-- << endl << endl; // 4
    cout << si << endl << endl; // 3

    si = 255;
}
```

```
C/C++ - SmallInt/SmallInt.cc - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help

SmallInt.cc
//overload for the output stream operator:
ostream& operator<<(ostream& os, const SmallInt& s) {
    os << s.value;
    return os;
}

int main() {
    SmallInt si( 3);
    cout << si + 3.14159 << endl << endl; // 6.14159
    cout << ++si << endl << endl; // 4
    cout << si++ << endl << endl; // 4
    cout << si << endl << endl; // 5
    cout << --si << endl << endl; // 4
    cout << si << endl << endl; // 4
    cout << si-- << endl << endl; // 4
    cout << si << endl << endl; // 3

    si = 255;
    cout << si << endl << endl;
    si++; // range violated, error message
    si = 300; // range violated, error message

    cout << si + 400 << endl << endl; // (C)
    // 655 (shows the dangers of
    // implicit conversion)
    return 0;
}

Writable Smart Insert 1 : 1
```

The screenshot shows the Eclipse IDE's console window. The title bar reads "C/C++ - Eclipse SDK". The menu bar includes "File", "Edit", "Refactor", "Navigate", "Search", "Run", "Project", "Window", and "Help". The console window title is "Console". The output text is as follows:

```
<terminated> CompareObject.exe (1) [C/C++ Local Application] C:\yunglu\eclipse\workspace\SmallInt\Debug\CompareObject.exe (5/21/08 11:49 PM)
SmallInt::SmallInt(int ival) {
SmallInt::rangeCheck(int i) {
operator int() const {
6.14159

SmallInt& SmallInt::operator++() {
4

SmallInt SmallInt::operator++(int) {
SmallInt::SmallInt(const SmallInt& other) {
SmallInt& SmallInt::operator++() {
4
5

SmallInt& SmallInt::operator--() {
4

4

SmallInt SmallInt::operator--(int) {
SmallInt::SmallInt(const SmallInt& other) {
SmallInt& SmallInt::operator--() {
4
3
```

The screenshot shows the Eclipse IDE's console window. The title bar reads "C/C++ - Eclipse SDK". The menu bar includes "File", "Edit", "Refactor", "Navigate", "Search", "Run", "Project", "Window", and "Help". The console window title is "Console". The output text is as follows:

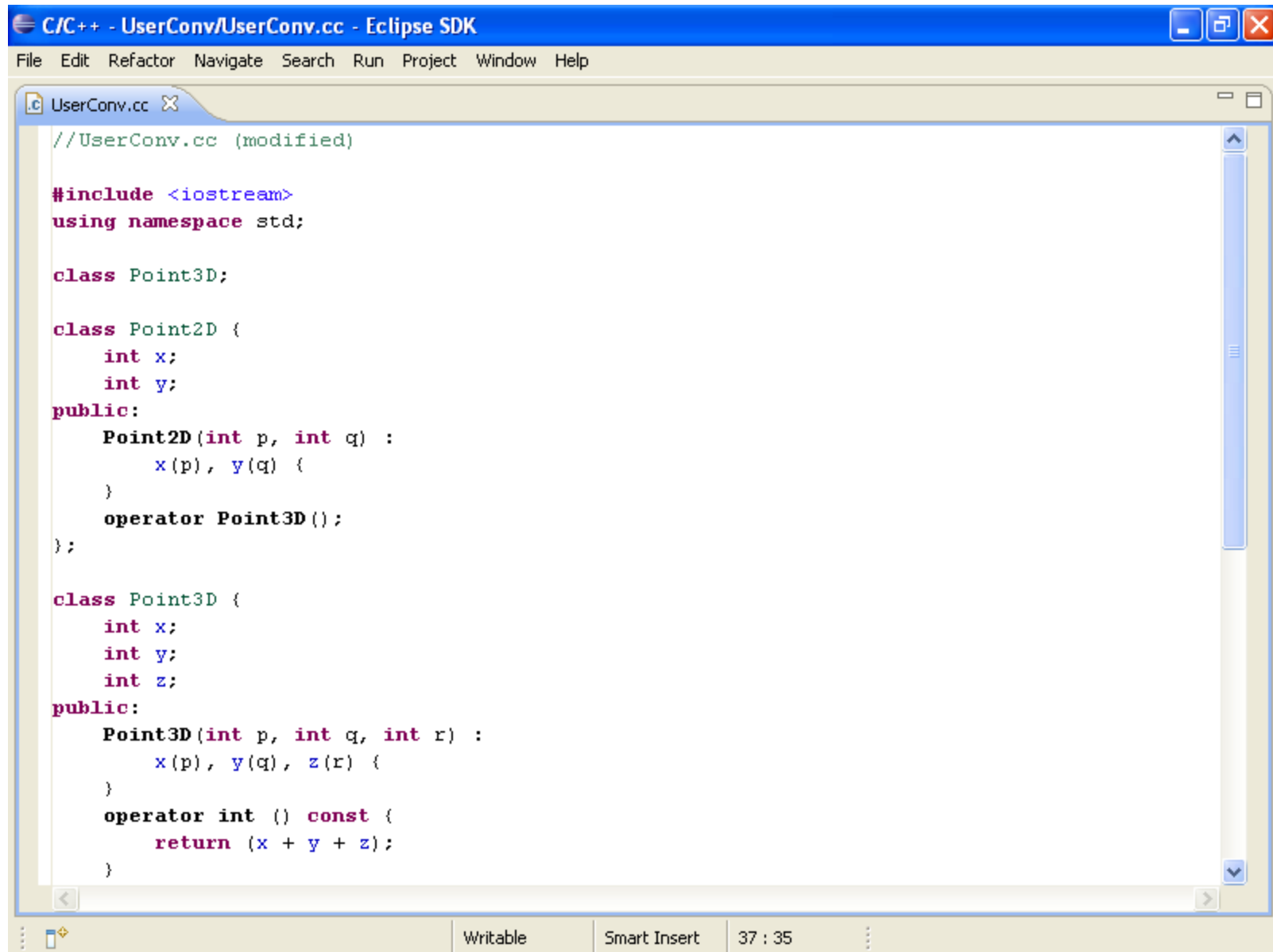
```
<terminated> CompareObject.exe (1) [C/C++ Local Application] C:\yunglu\eclipse\workspace\SmallInt\Debug\CompareObject.exe (5/21/08 11:49 PM)
SmallInt& SmallInt::operator--() {
4
4

SmallInt SmallInt::operator--(int) {
SmallInt::SmallInt(const SmallInt& other) {
SmallInt& SmallInt::operator--() {
4
3

SmallInt& SmallInt::operator=( const int i ) {
SmallInt::rangeCheck(int i) {
255

SmallInt SmallInt::operator++(int) {
SmallInt::SmallInt(const SmallInt& other) {
SmallInt& SmallInt::operator++() {
Error: Range of SmallInt violated
SmallInt& SmallInt::operator=( const int i ) {
SmallInt::rangeCheck(int i) {
Error: Range of SmallInt violated
operator int() const {
655
```

Programmer Defined (Implicit) Class Conversion



```
//UserConv.cc (modified)

#include <iostream>
using namespace std;

class Point3D;

class Point2D {
    int x;
    int y;
public:
    Point2D(int p, int q) :
        x(p), y(q) {
    }
    operator Point3D ();
};

class Point3D {
    int x;
    int y;
    int z;
public:
    Point3D(int p, int q, int r) :
        x(p), y(q), z(r) {
    }
    operator int () const {
        return (x + y + z);
    }
};
```

Writable Smart Insert 37 : 35


```
C/C++ - UserConv/UserConv.cc - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help

UserConv.cc

Point3D(int p, int q, int r) :
    x(p), y(q), z(r) {
}
operator int () const {
    return (x + y + z);
}
friend ostream& operator<<(ostream& os, const Point3D& point);
};

ostream& operator<<(ostream& os, const Point3D& point) {
    os << "(" << point.x << ", " << point.y << ", " << point.z << ")";
    return os;
}

inline Point2D::operator Point3D() {
    cout << "inline Point2D::operator Point3D() {" << endl;
    return Point3D( x, y, 0 );
}

int main() {
    Point2D p2D( 3, 4);
    cout << p2D << endl;
    // (3, 4, 0) using Point3D's operator<<
    Point3D p3D(1, 2, 3);
    cout << p3D - 1 << endl; // 5
    return 0;
}

Writable Smart Insert 50 : 1
```

Explicit in C++

```
C/C++ - UserConv/explicit.cc - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help

explicit.cc
// modified from http://www.glenmcccl.com/tip_023.htm
#include <iostream>
using namespace std;
class A {
    int value;
public:
    A(int i) {
        cout << "A::A(int i)" << endl;
        value = i;
    }
    friend void f(const A & aobj);
};

void f(const A & aobj) {
    cout << aobj.value << endl;
}

void g() {
    A a1 = 37;
    A a2 = A(47);
    A a3(57);
    a1 = 67;
    f(77);
}

int main(int argc, char * argv[]) {
    g();
    return 0;
}

Console
<terminated> CompareObject.exe (2) [C/C++
A::A(int i)
A::A(int i)
A::A(int i)
A::A(int i)
A::A(int i)
A::A(int i)
77

Writable Smart Insert 11 : 29
```

C/C++ - UserConv/explicit.cc - Eclipse SDK

File Edit Refactor Navigate Search Run Project Window Help

```
explicit.cc
// modified from http://www.glenmcccl.com/tip_023.htm
#include <iostream>
using namespace std;
class A {
    int value;
public:
    explicit A(int i) {
        cout << "A::A(int i)" << endl;
        value = i;
    }
    friend void f(const A & aobj);
};

void f(const A & aobj) {
    cout << aobj.value << endl;
}

void g() {
    A a1 = 37;
    A a2 = A(47);
    A a3(57);
    a1 = 67;
    f(77);
}

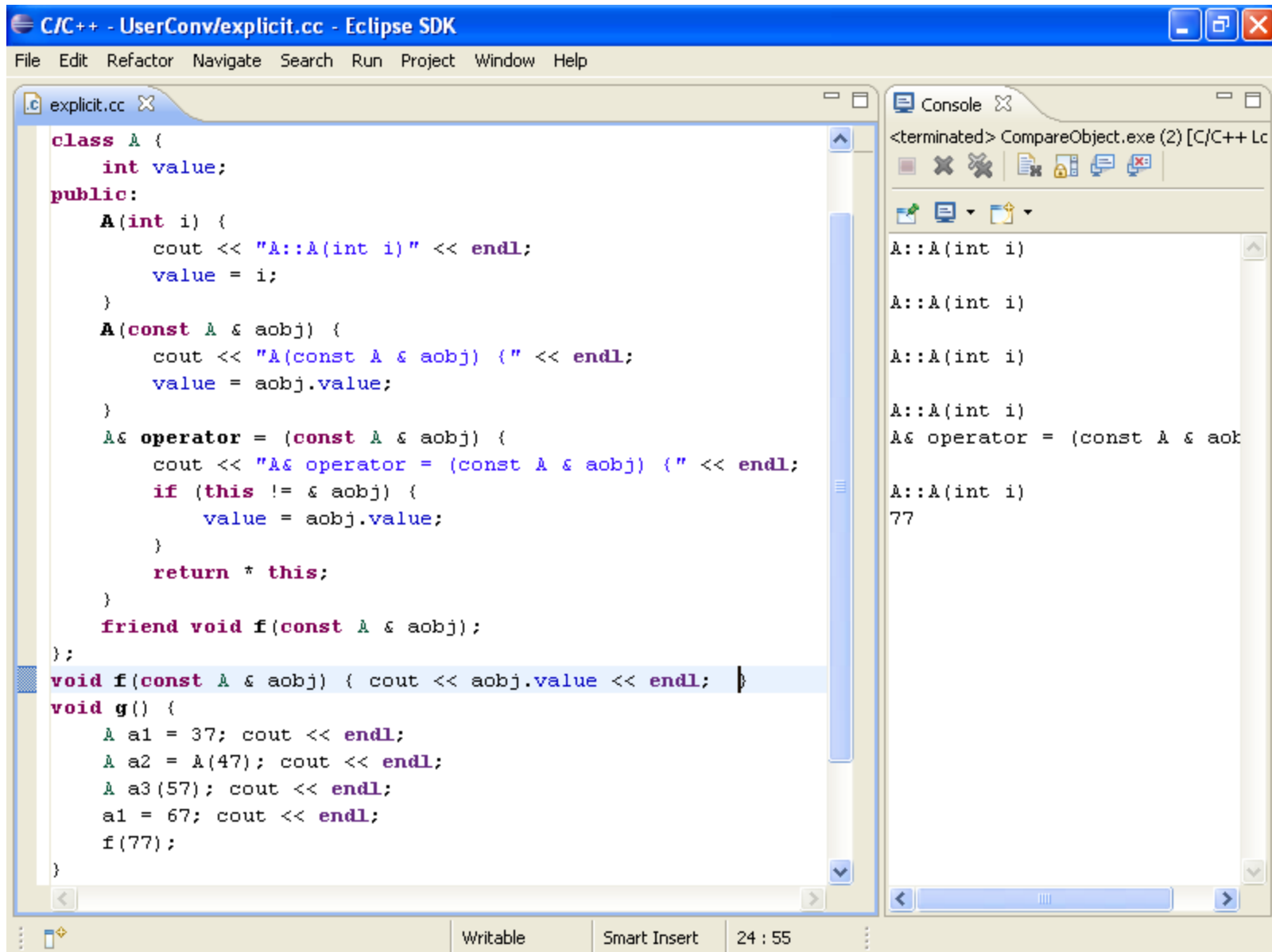
int main(int argc, char * argv[]) {
    g();
    return 0;
}
```

Console

C-Build [UserConv]

```
-MP -MF"explicit.d"
-MT"explicit.d"
-o"explicit.o"
"../explicit.cc"
../explicit.cc: In
function `void g()':
../explicit.cc:18: error:
conversion from `int' to
non-scalar type `A'
requested
../explicit.cc:21: error:
no match for `operator='
in `a1 = 67'
../explicit.cc:4: note:
candidates are: A&
A::operator=(const A&)
../explicit.cc:22: error:
invalid initialization of
reference of type `const
A&' from expression of
type `int'
../explicit.cc:14: error:
in passing argument 1 of
`void f(const A&)'
make: *** [explicit.o]
Error 1
```

Writable Smart Insert 17 : 11



Self Test