```
// Operator overloading
// George F. Riley, Georgia Tech, Spring 2009

#include <stdio.h>
#include <iostream>

using namespace std;

// Define class A with a default constructor, non-default constructor,
// and a "Copy Constructor"
class A {
public:
  A();         // Default constructor
  A(int);      // Non-Default Constructor
  A(const A&); // A copy constructor is used by the compiler whenever
               // a "copy" of an object is needed.
public:
  int x;       // Single data member
};

A::A()
{
  cout << "Hello from A::A() Default constructor" << endl;
}

A::A(int i)
    : x(i)
{
  cout << "Hello from A::A(int) constructor" << endl;
}

A::A(const A& a)
    : x(a.x)
{
  cout << "Hello from A::A(const A&) constructor" << endl;
}

A operator+(const A& lhs, const A& rhs)
{
  cout << "Returning from A::operator+" << endl;
  return A(lhs.x + rhs.x);
}


// Define a class B  similar to A.
// But B uses "member function" operator overloading
class B {
public:
  B();         // Default Constructor
  B(int);      // Non-Default Constructor
  B(const B&); // Copy constructor
  ~B();        // Destructor
  B operator+(const B& rhs); // Define member function addition operator
public:
  int x;       // Single data member
};
```

Program operators.cc

```

B::B()
    : x(0)
{
  cout << "Hello from B::B() default constructor" << endl;
}

B::B(int i)
    : x(i)
{
  cout << "Hello from B::B(int) constructor" << endl;
}

B::B(const B& b)
    : x(b.x)
{
  cout << "Hello from B::B(const B&) constructor" << endl;
}

// Implement the member function addition operator
B B::operator+(const B& rhs)
{
  cout << "Returning from B::operator+" << endl;
  B r(x + rhs.x); // Note LHS is "x", RHS is "rhs.h"
  return r;
}

B::~B()
{
  cout << "Hello from B::~B() destructor" << endl;
}


// Define class C similar to A and B
// addition operator.
class C {
public:
  C();         // Default constructor
  C(int);      // Non-Default Constructor
  C(const C&); // Copy constructor
public:
  int x;       // Single data member
};

C::C()
    : x(0)
{
  cout << "Hello from C::C() default constructor" << endl;
}

C::C(int c)
    : x(c)
{
  cout << "Hello from C::C(int) constructor" << endl;
}

```

Program operators.cc (continued)

```
C::C(const C& c)
    : x(c.x)
{
  cout << "Hello from C::C(const B&) constructor" << endl;
}

// Non-member addition operator for C
C operator+(const C& lhs, const C& rhs)
{
  cout << "Returning from C::operator+, non-member function" << endl;
  return C(lhs.x + rhs.x);
}

// We can also make an addition operator to add an A and B, returning A
A operator+(const A& lhs, const B& rhs)
{
  return A(lhs.x + rhs.x);
}

int main()
{
  cout << "Creating A objects"; getchar();
  A a0(1);
  A a1(10);
  A a2;
  cout << "Adding a0 + a1"; getchar();
  a2 = a0 + a1;

  cout << "Creating B objects"; getchar();
  B b0(2);
  B b1(20);
  B b2;
  B b3;
  cout << "Adding b0 + b1"; getchar();
  b2 = b0 + b1;
  cout << "Done adding" << endl;
  cout << "Adding b0 and 5" << endl;
  b2 = b0 + 5; // Why does this compile?
  cout << "Resulting b2 is " << b2.x << endl;
  // try this
  b2 = B(5) + b0;

  // Add an A and B object
  cout << "Adding a0 + b0"; getchar();
  a2 = a0 + b0;
  cout << "Resulting a2 is " << a2.x << endl;

  cout << "Creating C objects"; getchar();
  C c0(3);
  C c1(30);
  C c2;

  cout << "Adding c0 + c1"; getchar();
  c2 = c0 + c1;
  cout << "Resulting c2 is " << c2.x << endl;
  return 0;
```

Program operators.cc (continued)

```
}
```

Program operators.cc (continued)