6.5 Overloading Binary Operators

Binary operators can be overloaded just as easily as unary operators. We'll look at examples that overload arithmetic operators, comparison operators, and arithmetic assignment operators.

6.5.1 Arithmetic Operators

In the English Distance program we showed how two English Distance objects could be added using a member function add_dist():

```
dist3.add_dist(dist1, dist2);
By overloading the + operator we can reduce this dense-looking expression to
dist3 = dist1 + dist2;
Here's the listing for ENGLPLUS, which does just that:
// englplus.cpp
// overloaded '+' operator adds two Distances
#include <iostream.h>
class Distance //English Distance class
{
private:
int feet;
float inches;
public: //constructor (no args)
Distance() : feet(0), inches(0.0)
{ } //constructor (two args)
Distance(int ft, float in) : feet(ft), inches(in)
{ }
void getdist() //get length from user
{
cout << "\nEnter feet: "; cin >> feet;
cout << "Enter inches: "; cin >> inches;
}
void showdist() const //display distance
{ cout << feet << "\'-" << inches << '\"'; }
Distance operator + ( Distance ) const; //add 2 distances
};
//add this distance to d2
Distance Distance::operator + (Distance d2) const //return sum
{
int f = feet + d2.feet; //add the feet
float i = inches + d2.inches; //add the inches
if(i >= 12.0) //if total exceeds 12.0,
{ //then decrease inches
i -= 12.0; //by 12.0 and
```

```
f++; //increase feet by 1
} //return a temporary Distance
return Distance(f,i); //initialized to sum
}
int main()
{
Distance dist1, dist3, dist4; //define distances
dist1.getdist(); //get dist1 from user
Distance dist2(11, 6.25); //define, initialize dist2
dist3 = dist1 + dist2; //single '+' operator
dist4 = dist1 + dist2 + dist3; //multiple '+' operators
//display all lengths
cout << "dist1 = "; dist1.showdist(); cout << endl;
cout << "dist2 = "; dist2.showdist(); cout << endl;
cout << "dist3 = "; dist3.showdist(); cout << endl;
cout << "dist4 = "; dist4.showdist(); cout << endl;
return 0;
}
```

To show that the result of an addition can be used in another addition as well as in an assignment, another addition is performed in main (). We add dist1, dist2, and dist3 to obtain dist4 (which should be double the value of dist3), in the statement

dist4 = dist1 + dist2 + dist3;//Nameless Temporary Object will hold the intermediate result from adding dist1 and dist2.

Here's the output from the program:

```
Enter feet: 10
Enter inches: 6.5
dist1 = 10'-6.5" // from user
dist2 = 11'-6.25" // initialized in program
dist3 = 22'-0.75" // dist1+dist2
dist4 = 44'-1.5" // dist1+dist2+dist3
```

In class Distance the declaration for the operator+ () function looks like this:

Distance operator + (Distance);

This function has a return type of Distance, and takes one argument of type Distance. In expressions like

dist3 = dist1 + dist2;

It's important to understand how the return value and arguments of the operator relate to the objects. When the compiler sees this expression it looks at the argument types, and finding only type Distance, it realizes it must use the Distance member function operator+(). But what does this function use as its argument—dist1 or

dist2? And doesn't it need two arguments, since there are two numbers to be added?

Here's the key: The argument on the left side of the operator (dist1 in this case) is the object of which the operator is a member. The object on the right side of the operator (dist2) must be furnished as an argument to the operator. The operator returns a value, which can be assigned or used in other ways; in this case it is assigned to dist3. Figure 2 shows how this looks.

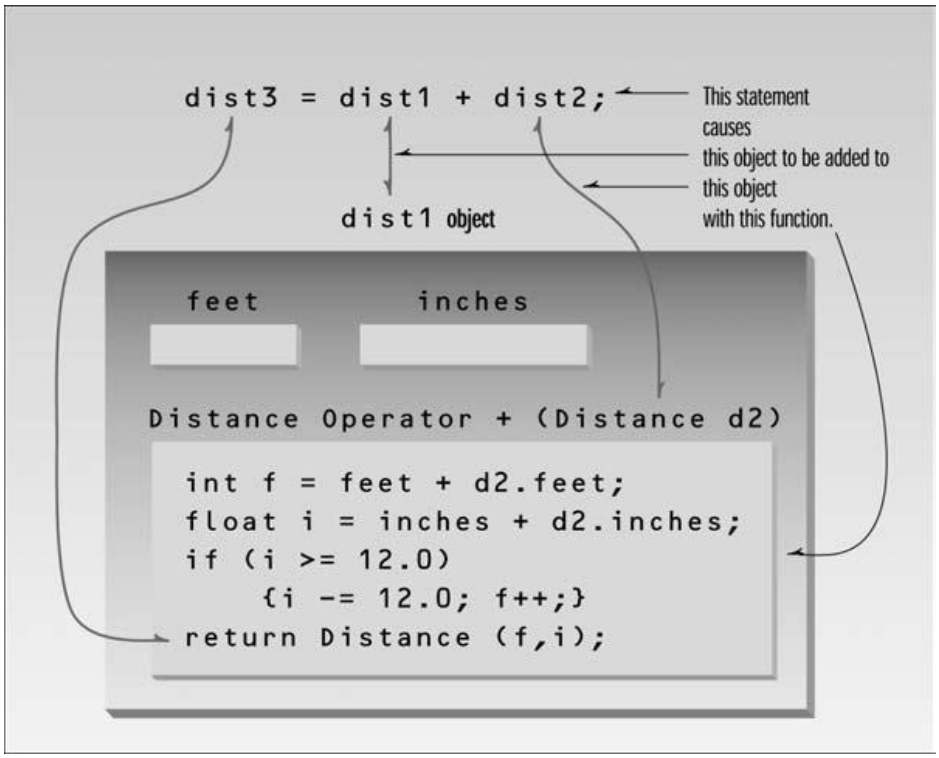

FIGURE 2

6.5.2 Overloaded binary operator: one argument.

In the operator+ () function, the left operand is accessed directly—since this is the object of which the operator is a member—using feet and inches. The right operand is accessed as the function's argument, as d2.feet and d2.inches. We can generalize and say that an overloaded operator always requires one less argument than its number of operands, since one operand is the object of which the operator is a

member. That's why unary operators require no arguments. To calculate the return value of operator+() in ENGLPLUS, we first add the feet and inches from the two operands (adjusting for a carry if necessary). The resulting values, f and i, are then used to initialize a nameless Distance object, which is returned in the statement

```
return Distance (f, i);
```

This is similar to the arrangement used in COUNTPP3, except that the constructor takes two arguments instead of one. The statement

```
dist3 = dist1 + dist2;
```

In main () then assigns the value of the nameless Distance object to dist3. Compare this intuitively obvious statement with the use of a function call to perform the same task, as in the ENGLCON example. Similar functions could be created to overload other operators in the Distance class, so you could subtract, multiply, and divide objects of this class in natural-looking ways.

6.5.3 Arithmetic Assignment Operators

Let's finish up our exploration of overloaded binary operators with an arithmetic assignment operator: the += operator. Recall that this operator combines assignment and addition into one step. We'll use this operator to add one English distance to a second, leaving the result in the first. This is similar to the ENGLPLUS example shown earlier, but there is a subtle difference.

Here's the listing for ENGLPLEQ:

```
// englpleq.cpp
// overloaded '+=' assignment operator
#include <iostream.h>
class Distance //English Distance class
{private:
int feet;
float inches;
public: //constructor (no args)
Distance() : feet(0), inches(0.0)
{ } //constructor (two args)
Distance(int ft, float in) : feet(ft), inches(in)
{ }
void getdist() //get length from user
```

```
{
cout << "\nEnter feet: "; cin >> feet;
cout << "Enter inches: "; cin >> inches;
}
void showdist() const //display distance
{ cout << feet << "\'-" << inches << '\"'; }
void operator += ( Distance );
};
//add distance to this one
void Distance::operator += (Distance d2)
{feet += d2.feet; //add the feet
inches += d2.inches; //add the inches
if(inches >= 12.0) //if total exceeds 12.0,
{ //then decrease inches
inches -= 12.0; //by 12.0 and
feet++; //increase feet
} //by 1
}
int main()
{Distance dist1; //define dist1
dist1.getdist(); //get dist1 from user
cout << "\ndist1 = "; dist1.showdist();
Distance dist2(11, 6.25); //define, initialize dist2
cout << "\ndist2 = "; dist2.showdist();
dist1 += dist2; //dist1 = dist1 + dist2
cout << "\nAfter addition,";
cout << "\ndist1 = "; dist1.showdist();
cout << endl;
return 0;}
```

In this program we obtain a distance from the user and add to it a second distance, initialized to 11'–6.25'' by the program. Here's a sample of interaction with the program:

```
Enter feet: 3
Enter inches: 5.75
dist1 = 3'-5.75"
dist2 = 11'-6.25"
After addition,
dist1 = 15'-0"
```

In this program the addition is carried out in main() with the statement

```
dist1 += dist2;
```

This causes the sum of dist1 and dist2 to be placed in dist1. Notice the difference between the function used here, operator+=(), and that used in ENGLPLUS, operator+(). In the earlier operator+() function, a new object of type Distance had to be created and returned by the function so it could be assigned to a third Distance object, as in

```
dist3 = dist1 + dist2;
```

In the operator+=() function in ENGLPLEQ, the object that takes on the value of the sum is the object of which the function is a member. Thus it is feet and inches that are given values, not temporary variables used only to return an object. The operator+= () function has no return value; it returns type void. A return value is not necessary with arithmetic assignment operators such as +=, because the result of the assignment operator is not assigned to anything. The operator is used alone, in expressions like the one in the program.

dist1 += dist2;

Exercise

1. To the Distance class in the ENGLPLUS program in this chapter, add an overloaded - operator that subtracts two distances. It should allow statements like dist3= dist1-dist2;. Assume that the operator will never be used to subtract a larger number from a smaller one (that is, negative distances are not allowed).

soluation

1).

```
// ex8_1.cpp
// overloaded '-' operator subtracts two Distances
#include <iostream.h>
class Distance //English Distance class
{
private:
int feet;
float inches;
public: //constructor (no args)
Distance() : feet(0), inches(0.0)
{ } //constructor (two args)
Distance(int ft, float in) : feet(ft), inches(in) { }
void getdist() //get length from user
{
cout << "\nEnter feet: "; cin >> feet;
cout << "Enter inches: "; cin >> inches;
}
void showdist() //display distance
{ cout << feet << "\'-" << inches << '\"'; }
Distance operator + ( Distance ); //add two distances
Distance operator - ( Distance ); //subtract two distances
};
//add d2 to this distance
```

```
Distance Distance::operator + (Distance d2) //return the sum
{
int f = feet + d2.feet; //add the feet
float i = inches + d2.inches; //add the inches
if(i >= 12.0) //if total exceeds 12.0,
{ //then decrease inches
i -= 12.0; //by 12.0 and
f++; //increase feet by 1
} //return a temporary Distance
return Distance(f,i); //initialized to sum
}
//subtract d2 from this dist
Distance Distance::operator - (Distance d2) //return the diff
{
int f = feet - d2.feet; //subtract the feet
float i = inches - d2.inches; //subtract the inches
if(i < 0) //if inches less than 0,
{ //then increase inches
i += 12.0; //by 12.0 and
f--; //decrease feet by 1
} //return a temporary Distance
return Distance(f,i); //initialized to difference
}
int main()
{
Distance dist1, dist3; //define distances
dist1.getdist(); //get dist1 from user
Distance dist2(3, 6.25); //define, initialize dist2
dist3 = dist1 - dist2; //subtract
//display all lengths
cout << "\ndist1 = "; dist1.showdist();
cout << "\ndist2 = "; dist2.showdist();
cout << "\ndist3 = "; dist3.showdist();
cout << endl;
                                                    return 0; }
```