6.14 Overloading Functions

CONCEPT: Two or more functions may have the same name, as long as their parameter lists are different.

Sometimes you will create two or more functions that perform the same operation, but use a different set of parameters or parameters of different data types. For instance, in Program 6-13 there is a `square` function that uses a `double` parameter. But, suppose you also wanted a `square` function that works exclusively with integers, accepting an `int` as its argument. Both functions would do the same thing: return the square of their argument. The only difference is the data type involved in the operation. If you were to use both these functions in the same program, you could assign a unique name to each function. For example, the function that squares an `int` might be named `squareInt`, and the one that squares a `double` might be named `squareDouble`. C++, however, allows you to *overload* function names. That means you may assign the same name to multiple functions, as long as their parameter lists are different. Program 6-27 uses two overloaded `square` functions.

Program 6-27

```
1  // This program uses overloaded functions.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  // Function prototypes
7  int square(int);
8  double square(double);
9
10 int main()
11 {
12    int userInt;
13    double userFloat;
14
15    // Get an int and a double.
16    cout << fixed << showpoint << setprecision(2);
17    cout << "Enter an integer and a floating-point value: ";
18    cin >> userInt >> userFloat;
19
20    // Display their squares.
21    cout << "Here are their squares: ";
22    cout << square(userInt) << " and " << square(userFloat);
23    return 0;
24 }
25
26 //*************************************************************
27 // Definition of overloaded function square.                  *
28 // This function uses an int parameter, number. It returns the *
29 // square of number as an int.                                *
30 //*************************************************************
```

```
31
32  int square( int number)
33  {
34     return number * number;
35  }
36
37  //*************************************************************
38  // Definition of overloaded function square.                  *
39  // This function uses a double parameter, number. It returns  *
40  // the square of number as a double.                          *
41  //*************************************************************
42
43  double square( double number)
44  {
45     return number * number;
46  }
```

Program Output with Example Input Shown in Bold

```
Enter an integer and a floating-point value: 12 4.2 [Enter]
Here are their squares: 144 and 17.64
```

Here are the headers for the `square` functions used in Program 6-27:

```
int square( int number)

double square( double number)
```

In C++, each function has a signature. The *function signature* is the name of the function and the data types of the function's parameters in the proper order. The `square` functions in Program 6-27 would have the following signatures:

```
square( int)

square( double)
```

When an overloaded function is called, C++ uses the function signature to distinguish it from other functions with the same name. In Program 6-27, when an `int` argument is passed to `square`, the version of the function that has an `int` parameter is called. Likewise, when a `double` argument is passed to `square`, the version with a `double` parameter is called.

Note that the function's return value is not part of the signature. The following functions could not be used in the same program because their parameter lists aren't different.

```
int square( int number)
{
   return number * number
}

double square( int number)  // Wrong! Parameter lists must differ
{
   return number * number
}
```

Overloading is also convenient when there are similar functions that use a different number of parameters. For example, consider a program with functions that return the sum of integers. One returns the sum of two integers, another returns the sum of three integers, and yet another returns the sum of four integers. Here are their function headers:

```
int sum(int num1, int num2)

int sum(int num1, int num2, int num3)

int sum(int num1, int num2, int num3, int num4)
```

Because the number of parameters is different in each, they all may be used in the same program. Program 6-28 is an example that uses two functions, each named `calcWeeklyPay`, to determine an employee's gross weekly pay. One version of the function uses an `int` and a `double` parameter, while the other version only uses a `double` parameter.

Program 6-28

```
1  // This program demonstrates overloaded functions to calculate
2  // the gross weekly pay of hourly paid or salaried employees.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  // Function prototypes
8  void getChoice(char &);
9  double calcWeeklyPay(int, double);
10 double calcWeeklyPay(double);
11
12 int main()
13 {
14    char selection;    // Menu selection
15    int worked;        // Hours worked
16    double rate;       // Hourly pay rate
17    double yearly;     // Yearly salary
18
19    // Set numeric output formatting.
20    cout << fixed << showpoint << setprecision(2);
21
22    // Display the menu and get a selection.
23    cout << "Do you want to calculate the weekly pay of\n";
24    cout << "(H) an hourly paid employee, or \n";
25    cout << "(S) a salaried employee?\n";
26    getChoice(selection);
27
28    // Process the menu selection.
29    switch (selection)
30    {
31       // Hourly paid employee
32       case 'H' :
33       case 'h' :  cout << "How many hours were worked? ";
```

```
34                   cin >> worked;
35                   cout << "What is the hourly pay rate? ";
36                   cin >> rate;
37                   cout << "The gross weekly pay is $";
38                   cout << calcWeeklyPay( worked, rate) << endl;
39                   break;
40
41       // Salaried employee
42       case 'S' :
43       case 's' :  cout << "What is the annual salary? ";
44                   cin >> yearly;
45                   cout << "The gross weekly pay is $";
46                   cout << calcWeeklyPay( yearly) << endl;
47                   break;
48    }
49    return 0;
50 }
51
52 //*************************************************************
53 // Definition of function getChoice.                          *
54 // The parameter letter is a reference to a char.             *
55 // This function asks the user for an H or an S and returns   *
56 // the validated input.                                       *
57 //*************************************************************
58
59 void getChoice( char & letter)
60 {
61    // Get the user's selection.
62    cout << "Enter your choice ( H or S): ";
63    cin >> letter;
64
65    // Validate the selection.
66    while ( letter != 'H' && letter != 'h' &&
67            letter != 'S' && letter != 's')
68    {
69       cout << "Please enter H or S: ";
70       cin >> letter;
71    }
72 }
73
74 //*************************************************************
75 // Definition of overloaded function calcWeeklyPay.           *
76 // This function calculates the gross weekly pay of           *
77 // an hourly paid employee. The parameter hours holds the     *
78 // number of hours worked. The parameter payRate holds the    *
79 // hourly pay rate. The function returns the weekly salary.   *
80 //*************************************************************
```

(program continues)

Program 6-28 *(continued)*

```
81
82  double calcWeeklyPay( int hours, double payRate)
83  {
84     return hours * payRate;
85  }
86
87  //*********************************************************
88  // Definition of overloaded function calcWeeklyPay.       *
89  // This function calculates the gross weekly pay of       *
90  // a salaried employee. The parameter holds the employee's *
91  // annual salary. The function returns the weekly salary.  *
92  //*********************************************************
93
94  double calcWeeklyPay( double annSalary)
95  {
96     return annSalary / 52;
97  }
```

Program Output with Example Input Shown in Bold

```
Do you want to calculate the weekly pay of
(H) an hourly paid employee, or
(S) a salaried employee?
Enter your choice (H or S): H [Enter]
How many hours were worked? 40 [Enter]
What is the hourly pay rate? 18.50 [Enter]
The gross weekly pay is $740.00
```

Program Output with Example Input Shown in Bold

```
Do you want to calculate the weekly pay of
(H) an hourly paid employee, or
(S) a salaried employee?
Enter your choice (H or S): S [Enter]
What is the annual salary? 68000.00 [Enter]
The gross weekly pay is $1307.69
```

6.15 The `exit()` Function

CONCEPT: The `exit()` function causes a program to terminate, regardless of which function or control mechanism is executing.

A C++ program stops executing when the `return` statement in function `main` is encountered. When other functions end, however, the program does not stop. Control of the program goes back to the place immediately following the function call. Sometimes, rare circumstances make it necessary to terminate a program in a function other than `main`. To accomplish this, the `exit` function is used.

When the `exit` function is called, it causes the program to stop, regardless of which function contains the call. Program 6-29 demonstrates its use.