```
   int var = 100;

   cout << var << endl;
   myFunc();
   cout << var << endl;
   return 0;
}

// Definition of function myFunc
void myFunc()
{
   int var = 50;

   cout << var << endl;
}
```

6.18 What is the output of the following program?

```
#include <iostream>
using namespace std;

void showVar(); // Function prototype

int main()
{
   for (int count = 0; count < 10; count++)
      showVar();
   return 0;
}

// Definition of function showVar
void showVar()
{
   static int var = 10;

   cout << var << endl;
   var++;
}
```

6.12 Default Arguments

CONCEPT: Default arguments are passed to parameters automatically if no argument is provided in the function call.

It's possible to assign *default arguments* to function parameters. A default argument is passed to the parameter when the actual argument is left out of the function call. The default arguments are usually listed in the function prototype. Here is an example:

```
void showArea(double = 20.0, double = 10.0);
```

Default arguments are literal values or constants with an = operator in front of them, appearing after the data types listed in a function prototype. Since parameter names are optional in function prototypes, the example prototype could also be declared as

```
void showArea(double length = 20.0, double width = 10.0);
```

In both example prototypes, the function `showArea` has two `double` parameters. The first is assigned the default argument 20.0 and the second is assigned the default argument 10.0. Here is the definition of the function:

```
void showArea(double length, double width)
{
   double area = length * width;
   cout << "The area is " << area << endl;
}
```

The default argument for `length` is 20.0 and the default argument for `width` is 10.0. Because both parameters have default arguments, they may optionally be omitted in the function call, as shown here:

```
showArea();
```

In this function call, both default arguments will be passed to the parameters. The parameter `length` will take the value 20.0 and `width` will take the value 10.0. The output of the function will be

```
The area is 200
```

The default arguments are only used when the actual arguments are omitted from the function call. In the call below, the first argument is specified, but the second is omitted:

```
showArea(12.0);
```

The value 12.0 will be passed to `length`, while the default value 10.0 will be passed to `width`. The output of the function will be

```
The area is 120
```

Of course, all the default arguments may be overridden. In the function call below, arguments are supplied for both parameters:

```
showArea(12.0, 5.5);
```

The output of the function call above will be

```
The area is 66
```

NOTE: If a function does not have a prototype, default arguments may be specified in the function header. The `showArea` function could be defined as follows:

```
void showArea(double length = 20.0, double width = 10.0)
{
   double area = length * width;
   cout << "The area is " << area << endl;
}
```

WARNING! A function's default arguments should be assigned in the earliest occurrence of the function name. This will usually be the function prototype.

Program 6-24 uses a function that displays asterisks on the screen. Arguments are passed to the function specifying how many columns and rows of asterisks to display. Default arguments are provided to display one row of 10 asterisks.

Program 6-24

```
// This program demonstrates default function arguments.
#include <iostream>
using namespace std;

// Function prototype with default arguments
void displayStars(int = 10, int = 1);

int main()
{
   displayStars();       // Use default values for cols and rows.
   cout << endl;
   displayStars(5);      // Use default value for rows.
   cout << endl;
   displayStars(7, 3);   // Use 7 for cols and 3 for rows.
   return 0;
}

//********************************************************
// Definition of function displayStars.                  *
// The default argument for cols is 10 and for rows is 1.*
// This function displays a square made of asterisks.    *
//********************************************************

void displayStars(int cols, int rows)
{
   // Nested loop. The outer loop controls the rows
   // and the inner loop controls the columns.
   for (int down = 0; down < rows; down++)
   {
      for (int across = 0; across < cols; across++)
         cout << "*";
      cout << endl;
   }
}
```

Program Output

```
**********

*****

*******
*******
*******
```

Although C++'s default arguments are very convenient, they are not totally flexible in their use. When an argument is left out of a function call, all arguments that come after it must be left out as well. In the `displayStars` function in Program 6-24, it is not possible to omit the argument for `cols` without also omitting the argument for `rows`. For example, the following function call would be illegal:

```
displayStars(, 3); // Illegal function call.
```

It's possible for a function to have some parameters with default arguments and some without. For example, in the following function (which displays an employee's gross pay), only the last parameter has a default argument:

```
// Function prototype
void calcPay(int empNum, double payRate, double hours = 40.0);

// Definition of function calcPay
void calcPay(int empNum, double payRate, double hours)
{
   double wages;

   wages = payRate * hours;
   cout << fixed << showpoint << setprecision(2);
   cout << "Gross pay for employee number ";
   cout << empNum << " is " << wages << endl;
}
```

When calling this function, arguments must always be specified for the first two parameters (`empNum` and `payRate`) since they have no default arguments. Here are examples of valid calls:

```
calcPay(769, 15.75);      // Use default arg for 40 hours
calcPay(142, 12.00, 20);  // Specify number of hours
```

When a function uses a mixture of parameters with and without default arguments, the parameters with default arguments must be defined last. In the `calcPay` function, `hours` could not have been defined before either of the other parameters. The following prototypes are illegal:

```
// Illegal prototype
void calcPay(int empNum, double hours = 40.0, double payRate);

// Illegal prototype
void calcPay(double hours = 40.0, int empNum, double payRate);
```

Here is a summary of the important points about default arguments:

- The value of a default argument must be a literal value or a named constant.
- When an argument is left out of a function call (because it has a default value), all the arguments that come after it must be left out too.
- When a function has a mixture of parameters both with and without default arguments, the parameters with default arguments must be declared last.

6.13 Using Reference Variables as Parameters

CONCEPT: When used as parameters, reference variables allow a function to access the parameter's original argument. Changes to the parameter are also made to the argument.

Earlier you saw that arguments are normally passed to a function by value, and that the function cannot change the source of the argument. C++ provides a special type of variable