4. Test and debug the program.
5. Document and maintain the program.

Using functions helps the programmer develop programs that can be easily coded, debugged, and maintained. Keep the following guidelines in mind when building programs of more than one function.

1. **Organization.** A large program is easier to read and modify if it is logically organized into functions. It is easier to work with a program in parts, rather than one large chunk. A well-organized program, consisting of multiple functions, is easier to read and debug. Once a single function is tested and performs properly, you can set it aside and concentrate on problem areas.
2. **Autonomy**. Programs should be designed so that they consist mainly of stand-alone functions or modules. Each function is autonomous, meaning the function does not depend on data or code outside the function any more than necessary.
3. **Encapsulation**. The term ***encapsulation*** refers to enclosing the details of a function within the function itself, so that those details do not have to be known in order to use the function.
4. **Reusability**. Because functions typically perform a single and well-defined task, they may be reused in the same program or even in other programs.

Functions may be written for any purpose. For example, you could create a function that converts Fahrenheit temperatures to Celsius or a function that gets input from the user. A function can also be a go-between for other parts of the program, as illustrated in the **handle_choice** function of Figure 9-2.

On the Net

For more information about program design and coding standards, see http://www.ProgramCPP.com. See topic 9.1.1.

Program Design

*There are two popular methods of designing programs. The first method, called **top-down design,** begins with the functions at the top of the VTOC and works toward the functions at the bottom of the VTOC. In other words, the general organization and flow of the program is decided before the details are coded.*

***Bottom-up design** involves beginning with the bottom of the VTOC and working your way up. Some programmers prefer to work out the details of how the program will perform specific tasks and then bring the details together to create the overall organization and flow.*

Whether you use top-down or bottom-up design, it is important to take an organized approach to writing a multi-function program.

THE SYNTAX OF FUNCTIONS

With each program you have written, you have created a main function. You can use a similar syntax to create other functions. But before we look at other functions, let's take another look at the main function. Up to this point, the main functions shown in this book have looked like the one below.

```
main()
{
 // body of program
 return 0;
}
```


When the program reaches the **return 0;** statement, the value zero is returned to the operating system. This value tells the operating system that the program ended normally. The value returned is a standard integer because we did not specify otherwise. The **int** type is assumed because we did not specify another type. To be more explicit, programs are often written using a main function like the one below.

```
int main()
{
 // body of program
 return 0;
}
```

To prevent a value from being returned, the **void** keyword is used in place of a data type. You may have seen programs with a main function like the one below.

```
void main()
{
 // body of program
}
```

In a **void** function, no value is returned, therefore no **return** statement is included. Newer operating systems are more likely to take advantage of the value returned by the main function. Therefore, you should get into the habit of creating main functions which return a zero when they terminate normally. The **void** main functions are used less frequently now than in the past.

As mentioned earlier, creating other functions in C++ programs is similar to creating the main function. Let's begin by looking at a simple function that prints a message to the screen.

```
void print_title()
 {
  cout << "Tennis Tournament Scheduler Program\n";
  cout << "By Jennifer Baker\n";
 }
```

The name of the function is **print_title**. The **void** keyword indicates that no value is returned. The parentheses after the name let the compiler know that **print_title** is a function. The statements between the braces are executed when the function **print_title** is "called." The main function below includes an example of a call to the **print_title** function.

```
int main()
 {
  print_title(); // call to print_title

  // insert the rest of the program here

  return 0;
 }
```