[Previous Section | Next Section | Chapter Index | Main Index]

Section 1.4

Fundamental Building Blocks of Programs

THERE ARE TWO BASIC ASPECTS of programming: data and instructions. To work with data, you need to understand variables and types; to work with instructions, you need to understand control structures and subroutines. You'll spend a large part of the course becoming familiar with these concepts.

A variable is just a memory location (or several consecutive locations treated as a unit) that has been given a name so that it can be easily referred to and used in a program. The programmer only has to worry about the name; it is the compiler's responsibility to keep track of the memory location. As a programmer, you need to keep in mind that the name refers to a kind of "box" in memory that can hold data, even though you don't have to know where in memory that box is located.

In Java and in many other programming languages, a variable has a type that indicates what sort of data it can hold. One type of variable might hold integers -- whole numbers such as 3, -7, and 0 -- while another holds floating point numbers -- numbers with decimal points such as 3.14, -2.7, or 17.0. (Yes, the computer does make a distinction between the integer 17 and the floating-point number 17.0; they actually look quite different inside the computer.) There could also be types for individual characters ('A', ';', etc.), strings ("Hello", "A string can include many characters", etc.), and less common types such as dates, colors, sounds, or any other kind of data that a program might need to store.

Programming languages always have commands for getting data into and out of variables and for doing computations with data. For example, the following "assignment statement," which might appear in a Java program, tells the computer to take the number stored in the variable named "principal", multiply that number by 0.07, and then store the result in the variable named "interest":

```
interest = principal * 0.07;
```

There are also "input commands" for getting data from the user or from files on the computer's disks, and there are "output commands" for sending data in the other direction.

These basic commands -- for moving data from place to place and for performing computations -- are the building blocks for all programs. These building blocks are combined into complex programs using control structures and subroutines.

A program is a sequence of instructions. In the ordinary "flow of control," the computer executes the instructions in the sequence in which they occur in the program, one after the other. However, this is obviously very limited: the computer would soon run out of instructions to execute. Control structures are special instructions that can change the flow of control. There are two basic types of control structure: loops, which allow a sequence of instructions to be repeated over and over, and branches, which allow the computer to decide between two or more different courses of action by testing conditions that occur as the program is running.

For example, it might be that if the value of the variable "principal" is greater than 10000, then the "interest" should be computed by multiplying the principal by 0.05; if not, then the interest should be

computed by multiplying the principal by 0.04. A program needs some way of expressing this type of decision. In Java, it could be expressed using the following "if statement":

```
if (principal > 10000)
    interest = principal * 0.05;
else
    interest = principal * 0.04;
```

(Don't worry about the details for now. Just remember that the computer can test a condition and decide what to do next on the basis of that test.)

Loops are used when the same task has to be performed more than once. For example, if you want to print out a mailing label for each name on a mailing list, you might say, "Get the first name and address and print the label; get the second name and address and print the label; get the third name and address and print the label..." But this quickly becomes ridiculous -- and might not work at all if you don't know in advance how many names there are. What you would like to say is something like "While there are more names to process, get the next name and address, and print the label." A loop can be used in a program to express such repetition.

Large programs are so complex that it would be almost impossible to write them if there were not some way to break them up into manageable "chunks." Subroutines provide one way to do this. A subroutine consists of the instructions for performing some task, grouped together as a unit and given a name. That name can then be used as a substitute for the whole set of instructions. For example, suppose that one of the tasks that your program needs to perform is to draw a house on the screen. You can take the necessary instructions, make them into a subroutine, and give that subroutine some appropriate name -- say, "drawHouse()". Then anyplace in your program where you need to draw a house, you can do so with the single command:

```
drawHouse();
```

This will have the same effect as repeating all the house-drawing instructions in each place.

The advantage here is not just that you save typing. Organizing your program into subroutines also helps you organize your thinking and your program design effort. While writing the house-drawing subroutine, you can concentrate on the problem of drawing a house without worrying for the moment about the rest of the program. And once the subroutine is written, you can forget about the details of drawing houses -- that problem is solved, since you have a subroutine to do it for you. A subroutine becomes just like a built-in part of the language which you can use without thinking about the details of what goes on "inside" the subroutine.

Variables, types, loops, branches, and subroutines are the basis of what might be called "traditional programming." However, as programs become larger, additional structure is needed to help deal with their complexity. One of the most effective tools that has been found is object-oriented programming, which is discussed in the next section.

[Previous Section | Next Section | Chapter Index | Main Index]