The Switch Statement

An alternative to if-else statements, switch allows you to express multiple choice branching in a clear way.

For comparison purposes, assume you wanted to write code that, given a variable , adds 5 to its value if it is a 1, adds 7 if its a 2, adds 10 if it is a 3, and multiplies it by 2 in all other cases. Using "if" the solution would be:

```
int number;

cin >> number;

if (number == 1)
number = number + 5;
else if (number == 2)
number = number + 7;
else if (number == 3)
number = number + 10;
else number = 2 * number;
```

The same could be expressed using switch in the following way:

```
switch (number) {

case 1: number = number + 5;
break;
case 2: number = number + 7;
break;
case 3: number = number + 10;
break;
default: number = number * 2;
}
```

The advantage is mainly a matter of clarity, where the multiple branches can be followed best when expressed in this cascading fashion.

Notice that the break statement is not part of the switch, but rather a C++ command that is used here to indicate that, once one case has matched the value of number, we execute the associated statements and then ignore the rest of the cases. Failure to include the "break" would allow the computation to continue cascading through the other cases, even if they don't match. In other words, once a case matches the value in the switch (in this case, the value of number), its statements are executed, and no other case is tested, therefore alloing the execution of their associated commands. If we want to avoid this, we need to use a break command.

Finally, the "default" option will be executed if none of the cases are actually matched. It is not mandatory, but in this case is necessary, since we want to multiply by 2 for all other values not listed in the case statements.