Bit Shift Operators

Igor Pro 7 or later supports the bit shift operators << and >>. `a<<n` shifts the bits of the integer a left by n bits. `a>>n` shifts the bits of the integer a right by n bits.

Normally you should apply bit shift operators to integer values. If you apply a bit shift operator to a floating point value, any fractional part is truncated.

Increment and Decrement Operators

In Igor7 and later, in a user-defined function, you can use these operators on local variables:

++var	Pre-increment
var++	Post-increment
--var	Pre-decrement
var--	Post-decrement

`var` must be a real local variable.

The pre-increment functions increment the variable and then return its new value. In this example, a is initialized to 0. `++a` then increments a to 1 and returns the new value. Consequently 1 is stored in b.

```
Function PreIncrementDemo()
   int a = 0
   int b = ++a
End
```

The post-increment functions return the variable's original value and then increment it. In this example, a is initialized to 0. `a++` returns the original value, 0, and then increments a to 1. Consequently 0 is stored in b.

```
Function PostIncrementDemo()
   int a = 0
   int b = a++
End
```

The pre-decrement and post-decrement work the same but decrement the variable rather than incrementing it.

Switch Statements

The switch construct can sometimes be used to simplify complicated flow control. It chooses one of several execution paths depending on a particular value.

Instead of a single form of switch statement, as is the case in C, Igor has two types: *switch* for numeric expressions and *strswitch* for string expressions. The basic syntax of these switch statements is as follows:

```
switch(<numeric expression>)        // numeric switch
   case <literal number or numeric constant>:
      <code>
      [break]
   case <literal number or numeric constant>:
      <code>
      [break]
   . . .
   [default:
      <code>]
endswitch
```

```
strswitch(<string expression>)      // string switch
   case <literal string or string constant>:
      <code>
      [break]
   case <literal string or string constant>:
      <code>
      [break]
   . . .
   [default:
      <code>]
endswitch
```

The switch numeric or string expression is evaluated. In the case of numeric switches, the result is rounded to the nearest integer. Execution proceeds with the code following the matching case label. When none of the case labels match, execution continues at the default label, if it is present, or otherwise the switch exits with no action taken.

All of the case labels must be numeric or string constant expressions and they must all have unique values within the switch statement. The constant expressions can either be literal values or they must be declared using the Constant and StrConstant keywords for numeric and string switches respectively.

Literal numbers used as case labels are required by the compiler to be integers. Numeric constants used as case labels are rounded by the compiler to the nearest integers.

Execution proceeds within each case until a break statement is encountered or the endswitch is reached. The break statement explicitly exits the switch construct. Usually, you should put a break statement at the end of each case. If you omit the break statement, execution continues with the next case label. Do this when you want to execute a single action for more than one switch value.

The following examples illustrate how switch constructs can be used in Igor:

```
Constant kThree=3
StrConstant ksHow="how"

Function NumericSwitch(a)
   Variable a

   switch(a)                        // numeric switch
      case 1:
         print "a is 1"
         break
      case 2:
         print "a is 2"
         break
      case kThree:
      case 4:
         print "a is 3 or 4"
         break
      default:
         print "a is none of those"
         break
   endswitch
End

Function StringSwitch(a)
   String a

   strswitch(a)                     // string switch
      case "hello":
         print "a is hello"
         break
      case ksHow:
         print "a is how"
```