Introduction To Operators

Hour 6

Objectives

- Precedence and associativity
- Assignment
- Increment and decrement
- Relational
- Casting
- Type and format conversion functions
- Character testing functions

---

Copyright © 1998-2002 Delroy A. Brinkerhoff. All Rights Reserved.

Operators

Precedence and associativity

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Assoc</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>indexing</td>
<td>↓</td>
</tr>
<tr>
<td>., -&gt; (dot and arrow)</td>
<td>member selection</td>
<td>↓</td>
</tr>
<tr>
<td>( )</td>
<td>function invocation</td>
<td>↓</td>
</tr>
<tr>
<td>++, --</td>
<td>auto increment and decrement</td>
<td>↑</td>
</tr>
<tr>
<td>!</td>
<td>logical negation</td>
<td>↑</td>
</tr>
<tr>
<td>~</td>
<td>bitwise complement</td>
<td>↑</td>
</tr>
<tr>
<td>+, -</td>
<td>unary</td>
<td>↑</td>
</tr>
<tr>
<td>()</td>
<td>grouping and casting</td>
<td>↑</td>
</tr>
<tr>
<td>&amp;</td>
<td>address off</td>
<td>↑</td>
</tr>
<tr>
<td>*</td>
<td>indirection or dereference</td>
<td>↑</td>
</tr>
<tr>
<td>sizeof</td>
<td>size of variable or type</td>
<td>↑</td>
</tr>
<tr>
<td>*, /, %</td>
<td>multiplication, division, modulo</td>
<td>←</td>
</tr>
</tbody>
</table>

---

Copyright © 1998-2002 Delroy A. Brinkerhoff. All Rights Reserved.
## Operators

Continued

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, -</td>
<td>addition and subtraction</td>
<td>➔</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>bitwise left shift</td>
<td>➔</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>bitwise right shift (sign extend)</td>
<td>➔</td>
</tr>
<tr>
<td>&lt;, &gt;, &lt;=, &gt;=</td>
<td>logical relationships</td>
<td>➔</td>
</tr>
<tr>
<td>==, !=</td>
<td>logical equality or inequality</td>
<td>➔</td>
</tr>
<tr>
<td>&amp;</td>
<td>bitwise and</td>
<td>➔</td>
</tr>
<tr>
<td>^</td>
<td>bitwise exclusive-or</td>
<td>➔</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bitwise or</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>logical and</td>
<td>➔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?:</td>
<td>conditional expression</td>
<td>➔</td>
</tr>
<tr>
<td>=, +=, -=, *=, /=, %=, &lt;&lt;=, &gt;&gt;=, &amp;=,</td>
<td>=, ^=</td>
<td>operation and assignment (target must be a variable or l-value)</td>
</tr>
</tbody>
</table>

## Precedence and Associativity

**Evaluation order**

- **Precedence:** which operator is evaluated first
  - `a + b * c` * has a higher precedence than + and is evaluated 1st
  - `(a + b) * c` parentheses force the + to be evaluated 1st

- **Associativity:** which operator is evaluated first if the precedence is the same
  - `a + b + c` is equivalent to `(a + b) + c` because + is left associative (i.e., it is evaluated left to right)
  - `a = b = c` is equivalent to `a = (b = c)` because = is right associative (i.e., it is evaluated right to left)
### Assignment

More unusual C operators

- `=`
  - `x = y + z; /* assignment statement: y & z are rvalues, x is lvalue */`
  - `x = y = 0; /* initialize two or more variables with one statement */`

- `op= (+=, -=, *=, /=, %=, ~=, <<=, >>=)`
  - `variable op= expression; ← variable = variable op expression;`
  - `x += 10; ← x = x + 10;`
  - `i -= 2; ← i = i -2;`
  - `a /= b; ← a = a / b;`
  - `index %= size; ← index = index % size;`
  - `mask <<= 2; ← mask = mask << 2;`

### Operator Examples

Unusual C operators

- **Auto increment and decrement**
  - Target must be a variable (i.e., an l-value)
  - `int i, j = 10;`
  - `i = j++; /* postincrement: i is 10, j is 11 */`
  - `i = ++j; /* preincrement: i is 11, j is 11 */`
  - `i = j--; /* postdecrement: i is 10, j is 9 */`
  - `i = --j; /* predecrement: i is 9, j is 9 */`
  - `i++, ++i, i--, and --i are legal (pre and post are equivalent)`
  - May be embedded in expressions
  - Often used in array indexes and for loops

- **Multiple assignment (i.e., the = operator returns a value)**
  - `i = j = k = 0; /* equivalent to i = (j = (k = 0)) */`
  - `while ((c = getc()) != -1) /*`
Relational Operators

Return true or false

- `==` equal to
- `!=` not equal to
- `>` greater than
- `<` less than
- `>=` greater than or equal to
- `<=` less than or equal to

- counter == 0
- counter != 0
- counter > 0
- counter < 0
- counter >= 0
- counter <= 0
- c == max
- a != x
- a > z
- count < min
- lim >= max
- lim <= min

Type Conversions & the Cast Operator

Implicit and explicit

- chars and ints convert back and forth
- Small types are automatically “promoted” to large types in arithmetic expressions
- If a and b are ints, a/b is an int (truncates if necessary)
- Explicit conversions are called type casts: (type)expression
  - Required to convert from a large type to a small type and to force conversions for /
  - int i = (int) 3.14159; /* always truncates */
  - int i = (int)(3.14159 + 2.7);
  - unsigned int i = (unsigned int) 75;
  - double x = (double) 1 / 3;
Format Conversion Functions
From strings to integral values

- #include <stdlib.h>
- int atoi(const char *s); /* ascii (string) to integer */
- long atol(const char *s); /* ascii (string) to long */
- long strtol(const char *s, char **endptr, int radix);
- unsigned long strtoul(const char *s, char **endptr, int radix);

strtol and strtoul parameters
- s string to convert
- endptr address where the conversion stopped
- radix number base: 2 is binary, 10 is decimal, 16 is hexadecimal, etc.

Format Conversion Functions
From strings to floating point (i.e., real) values

- double atof(const char *s);
  - ascii (string) to floating point (type double)
  - #include <math.h>
- double strtod(const char *s, char **endptr);
  - s string to convert
  - endptr address where the conversion stopped
  - #include <stdlib.h>
Format Conversion Functions

Numeric values to strings

- `#include <stdlib.h>
- `char* fcvt(double value, int ndig, int *dec, int *sign);
- `char* ecvt(double value, int ndig, int *dec, int *sign);
- `char* gcvt(double value, int ndec, char *buf);
- `char* itoa(int value, char *string, int radix);
- `char* ltoa(long value, char *string, int radix);
- `char* ultoa(unsigned long value, char *string, int radix);
- Note that f, e, g, i, l, and u are like the printf format specifiers

Ctype Library

Character identification and conversion

- `Using ctype macros / functions
  - Macros return 1 for true and 0 for false
  - `isascii()` is defined on all integer values; the rest are defined only for integers representing characters, or EOF
  - `#include <ctype.h>
- `int isalpha(int c)
- `int isupper(int c)
- `int islower(int c)
- `int isdigit(int c)
- `int isxdigit(int c)
- `int isalnum(int c)
- `int isspace(int c)
- `int ispunct(int c)
- `int isprint(int c)
- `int isgraph(int c)
- `int iscntrl(int c)
- `int isascii(int c)
- `int toupper(int c)
- `int tolower(int c)