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

Precendence

Evaluation order

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

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

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; "i = i + 10;"
  - i = 2; "a = a / b;"
  - 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 12 */
  - i = i--; /* postdecrement: i is 9, j is 9 */
  - j = j--; /* predecrement: i is 9, j is 8 */
  - i++, ++i; l- 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 = i = k = 0; /* equivalent to i = (i = (k = 0)) */
  - while ((c = getc()) != -1)
Relational Operators

Return true or false

- == equal to  
  counter == 0  c == max
- != not equal to  
  counter != 0  a != x
- > greater than  
  counter > 0  a > z
- < less than  
  counter < 0  count < min
- >= greater than or equal to  
  counter >= 0  lim >= max
- <= less than or equal to  
  counter <= 0  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

```c
#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);
```

Format Conversion Functions

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

```c
#include <math.h>

double atof(const char *s);
double strtod(const char *s, char **endptr);
```

Format Conversion Functions

Numeric values to strings

```c
#include <stdlib.h>

char* fcvt(double value, int ndig, int *dec, int *sign);
char* gcvt(double value, int ndig, 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);
```

Ctype Library

Character identification and conversion

```c
#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 topper(int c)
int tolower(int c)
```