ESc101: Operators and Variable Type Conversion

Instructor: Krithika Venkataramani
Semester 2, 2011-2012

The content of these slides are taken from the lecture slides of Prof. Arnab Bhattacharya

Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>int</th>
<th>double, float</th>
<th>char</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>yes</td>
<td>yes</td>
<td>restricted</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>yes</td>
<td>yes</td>
<td>restricted</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>yes</td>
<td>yes</td>
<td>best to avoid</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>integer</td>
<td>yes</td>
<td>best to avoid</td>
</tr>
<tr>
<td>%</td>
<td>modulus</td>
<td>yes</td>
<td>no</td>
<td>best to avoid</td>
</tr>
</tbody>
</table>

Examples

- 33 / 5 = 6
- 33 % 5 = 3
- -33 / 5 = -6 (non-standard)
- -33 % 5 = -3 (non-standard)
- 33.0 / 5.0 = 6.6
Examples on Character Operations

- Characters are treated as small integers, so it is possible to operate (add, subtract, compare) on them.
- 'a' + 2 = 'c'
- 'a' - 2 = ' '
- 'A' + '1' = 65 + 49 = 114 = 'r' (avoid)
- '1' * 2 = 49 * 2 = 98 = 'b' (avoid)

Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>int</th>
<th>float, double</th>
<th>char</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>equal to</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Boolean Operations

- There is no boolean or truth type in C
- Integers are treated as booleans
- Value 0 represents false
- Any non-zero value (typically 1) represents true
- Examples:
  - -4 is evaluated as TRUE
  - !(4) is evaluated as FALSE
  - 0.0 is evaluated as FALSE
  - -4.0 is also evaluated as TRUE
  - !(4.0) is evaluated as FALSE

Examples of relational operations

- 33 > 5
  - 1 = 1
- 'a' > 'b'
  - 0 = 0
- 'a' == 97
  - 1 = 1
- 'a' == 353
  - 0 = 0
- 'a' == 97.0
  - 0 = 1
- 96.0 == 97
  - 0 = 0
- 97.0 == 97
  - 1 = 1 (avoid such automatic type conversions)
### Logical or boolean operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>int</th>
<th>float, double</th>
<th>char</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>logical and</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>logical or</td>
<td>yes</td>
</tr>
</tbody>
</table>

### Examples of logical operators on characters

- !'a' =
  - 0
- "0" =
  - 0
- Numbers enclosed in single quotes denote characters
- "0" =
  - 1
- "a" || "0" =
  - 1

### Examples of logical operators

- !4 =
  - 0
- !4.0 =
  - 0
- 4 && 5 =
  - 1
- 4.0 && 5.0 =
  - 1
- 4 && 0 =
  - 0
- 4 || 0 =
  - 1

### Unary Operators

Unary operators: Operate on a single variable/constant

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>int</th>
<th>float, double</th>
<th>char</th>
</tr>
</thead>
<tbody>
<tr>
<td>(unary) +</td>
<td>positive</td>
<td>yes</td>
<td>yes</td>
<td>best to avoid</td>
</tr>
<tr>
<td>(unary) -</td>
<td>negative</td>
<td>yes</td>
<td>yes</td>
<td>best to avoid</td>
</tr>
<tr>
<td>++</td>
<td>increment by 1</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>--</td>
<td>decrement by 1</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>=</td>
<td>assignment</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Examples of unary operations

- $45 + -33 = 12$
- $45 + +33 = 78$
- $i = 33; i++; i$ becomes $34$
- $i = 33; i--; i$ becomes $32$
- $c = 'g'; ++c; c$ becomes $'h'$
- $c = 'g'; --c; c$ becomes $'f'$

Brackets in expressions

- An expression is any legal combination of variables, constants and operators
- Brackets ( ) are needed to enforce particular order
- $(12 + 6) / 3 =$
  - $6$
- $24 / (6 * 2) =$
  - $2$
- $24 / (6 / 2) =$
  - $8$
- $12 - (6 - 3) =$
  - $9$
- Best to use brackets to avoid confusion

Expressions

- An expression is any legal combination of variables, constants and operators
- $12 + 6 / 3 =$
  - $14$
  - / evaluated earlier than + due to higher precedence
- $24 / 6 * 2 =$
  - $8$
  - / is of same precedence as * and order of evaluation is left to right
- $24 / 6 / 2 =$
  - $2$
  - / is evaluated from left to right
- $12 - 6 - 3 =$
  - $3$
  - - is evaluated from left to right

Precedence and Associativity

- $- 6 / 3 * - 5 < 8 || 0 > 5 - 6 =$
  - $1$
  - ((((-6) / 3) * (-5)) < 8) || (0 > (5 - 6)) = 1
- Decide using rules of operator precedence and associativity
- Precedence: among multiple operators, precedence rules guide the order in which they will be evaluated
  - Operators having higher precedence are evaluated earlier
  - $12 + 6 / 3$ is really $12 * (6 / 3)$
- Associativity: among multiple instances of the same operator, associativity rules guide the order in which they will be evaluated
  - If an operator is left-to-right associative, the leftmost instance of the operator is evaluated first, and so on
  - $12 - 6 - 3$ is really $(12 - 6) - 3$
- It is best to avoid using such expressions without brackets
Operator precedence and associativity

- Brackets have highest precedence
- Unary operators: right-to-left
- Arithmetic operators: left-to-right
- Relational operators: left-to-right
- Logical operators: left-to-right

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Highest</td>
<td>left to right</td>
</tr>
<tr>
<td>! ++ - - (unary)+ (unary)-</td>
<td>right to left</td>
<td></td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
<td></td>
</tr>
<tr>
<td>+ -</td>
<td>left to right</td>
<td></td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>left to right</td>
<td></td>
</tr>
<tr>
<td>== !=</td>
<td>left to right</td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>left to right</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>Lowest</td>
<td>right to left</td>
</tr>
</tbody>
</table>

Statements and Blocks

- A statement is an expression followed by a semicolon (;)
  - `i = 12 + 6 / 3;` Assignment (`=`) is an expression
- A series of statements grouped together using braces `{ }` is a block of statements or a compound statement
  ```
  { 
  i = 5;
  i++;
  }
  ```
- A block of statements is treated as a single statement

Type Conversion

- For arithmetic/relational operators, lower variable type is promoted to higher variable type
- Among common variable types, the type order is below.
  - double (highest)
  - float
  - int
  - char (lowest)
- For assignments, value of right side is converted to type of left
- The value in code snippets like below depend on the rules
  - `double d = 12.56;` int `i = d;`
  - `i = 34.56;`
- Value of `i` is first 12, and then 34
Examples

- $33.0 / 5 = 6.6$
- $33.0 \% 5 = \text{error}$
- $-33 / 5.0 = -6.6$
- $33.5 + 5 = 38.5$
- $10 * 5.5 = 55.0$

Explicit Type conversion of variables

- Explicit type casting of variables can be done by
  double $d = -12.56$;
  int $i = (\text{int}) d$;
- Value of $i$ gets truncated to -12
- Conversion from higher type to lower type may lose information
- Conversion from lower type to higher type should not lose information
- Avoid implicit type conversion as much as possible

Examples of relational operations

- $5 == 5.0$
  $= 1$
- $-4.0 != 4$
  $= 0$