C Programming:
Arithmetic and Logic Operations

A. Sahu and S. V. Rao
Dept of Comp. Sc. & Engg.
Indian Institute of Technology Guwahati
Algebra: BEDMAS/PEDMAS Rule (recap)

• B-E-DM-AS or P-E-DM-AS

• B/P : Bracket or Parenthesis ( )
  – In C, only ( ) used for expression
  – Curly braces {}, and square bracket [] used for some other purpose.
    • Again [] may involves in expression as in the form of array access

• E : Exponentiation

• DM: Division and Multiplication

• AS : Addition and Subtraction
**BEDMAS equivalent in C**

**Arithmetic Operators Precedence Rule (recap)**

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Precedence &amp; Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Evaluated first. If <strong>nested</strong> (embedded), innermost first.</td>
</tr>
<tr>
<td>* / %</td>
<td>Evaluated second. If there are several, evaluated left to right.</td>
</tr>
<tr>
<td>+ -</td>
<td>Evaluated third. If there are several, evaluated left to right.</td>
</tr>
<tr>
<td>=</td>
<td>Evaluated last, right to left.</td>
</tr>
</tbody>
</table>
Using Parentheses

• Use parentheses to change the order in which an expression is evaluated.

\[ a + b \times c \] Would multiply b * c first, then add a to the result.

If you really want the sum of a and b to be multiplied by c, use parentheses to force the evaluation to be done in the order you want.

\[ (a + b) \times c \]

• Also use parentheses to clarify a complex expression.
Practice With Evaluating Expressions
(recap)

Given integer variables \( a, b, c, d, \) and \( e, \)
where \( a = 1, \) \( b = 2, \) \( c = 3, \) \( d = 4, \)
evaluate the following expressions:

\[
\begin{align*}
\text{a} + \text{b} - \text{c} + \text{d} \\
\text{a} \times \text{b} / \text{c} \\
1 + \text{a} \times \text{b} \% \text{c} \\
\text{a} + \text{d} \% \text{b} - \text{c} \\
e = \text{b} = \text{d} + \text{c} / \text{b} - \text{a}
\end{align*}
\]
Practice With Evaluating Expressions (recap)

Given integer variables a, b, c, d, and e, where a = 1, b = 2, c = 3, d = 4, evaluate the following expressions:

\[
\begin{align*}
\text{a + b - c + d} &= 3 - 3 + 4 = 0 + 4 = 4 \\
\text{a * b / c} &= 2 / 3 + 4 = 0 + 4 = 4 \\
\text{1 + a * b \% c} &= 1 + 2 \% 3 = 1 + 2 = 3 \\
\text{a + d \% b - c} &= 1 + 0 - 3 = 1 - 3 = -2 \\
\text{e = b = d + c / b - a} &=
\end{align*}
\]
Increment and Decrement Operators

• The **increment operator**  `++`
• The **decrement operator**  `--`
• Precedence: lower than `()`, but higher than `*` `/` and `%`
• Associativity: right to left
• Increment and decrement operators can only be applied to variables, not to constants or expressions
Increment Operator

• If we want to add one to a variable, we can say:
  \[
  \text{count} = \text{count} + 1 ;
  \]

• Programs often contain statements that increment variables, so to save on typing, C provides these shortcuts:
  \[
  \text{count}++ ; \quad \text{OR} \quad ++\text{count} ;
  \]
Both do the same thing. They change the value of count by adding one to it.
**Postincrement Operator**

- The position of the `++` determines when the value is incremented. If the `++` is after the variable, then the incrementing is done last (a postincrement).

```java
int amount, count;
count = 3;
amount = 2 * count++;
```

- amount gets the value of `2 * 3`, which is 6, and then 1 gets added to count.
- So, after executing the last line, amount is 6 and count is 4.
Preincrement Operator

• If the ++ is before the variable, then the incrementing is done first (a preincrement).

```cpp
int amount, count;
count = 3;
amount = 2 * ++count;
```

• 1 gets added to count first, then amount gets the value of 2 * 4, which is 8.

• So, after executing the last line, amount is 8 and count is 4.
A Hand Trace Example

```c
int ans, val= 4 ;
```

<table>
<thead>
<tr>
<th>Code</th>
<th>Val</th>
<th>Ans</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val = value + 1 ;</code></td>
<td>4</td>
<td>garbage</td>
</tr>
<tr>
<td><code>val++ ;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>++val ;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ans = 2 * val++ ;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ans = ++val / 2 ;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>val-- ;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>--val ;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ans = --val * 2 ;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ans = val-- / 3 ;</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# A Hand Trace Example

```c
int ans, val = 4;

val = value + 1;
val++;
++val;
ans = 2 * val++;
an = ++val / 2;
val--;
--val;
an = --val * 2;
an = val-- / 3;
```

<table>
<thead>
<tr>
<th>Code</th>
<th>Val</th>
<th>Ans</th>
</tr>
</thead>
<tbody>
<tr>
<td>val = value + 1;</td>
<td>4</td>
<td>garbage</td>
</tr>
<tr>
<td>val++ ;</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>++val ;</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>ans = 2 * val++ ;</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>ans = ++val / 2 ;</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>val-- ;</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>--val ;</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>ans = --val * 2 ;</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>ans = val-- / 3 ;</td>
<td>12</td>
<td>1</td>
</tr>
</tbody>
</table>
```c
int main(){
    int ans, val=4;
    val = val + 1;
    printf("ans=%d val=%d\n",ans,val);
    val++ ; ++val ;
    printf("ans=%d val=%d\n",ans,val);
    ans = 2 * val++ ;
    printf("ans=%d val=%d\n",ans,val);
    val--;--val;
    printf("ans=%d val=%d\n",ans,val);
    ans=--val*2;
    printf("ans=%d val=%d\n",ans,val);
    ans = val-- / 3 ;
    printf("ans=%d val=%d\n",ans,val);
    return 0;
}
```
Practice

Given

```java
int a = 1, b = 2, c = 3;
```

What is the value of this expression?

```java
++a * b - c--
```

What are the new values of `a`, `b`, and `c`?
Practice

Given

\[ \text{int } a = 1, \ b = 2, \ c = 3; \]

What is the value of this expression?

\[ \text{++a * b - c--} = 1 \]

What are the new values of a, b, and c?

\[ a = 2, \ c = 2 \]
More Practice

Given

```java
int a = 1, b = 2, c = 3, d = 4;
```

What is the value of this expression?

```
++b / c + a * d++
```

What are the new values of a, b, c, and d?
More Practice

Given

```java
int a =1, b =2, c =3, d =4 ;
```

What is the value of this expression?

```java
++b / c + a * d++
```

= 1+4 =5

What are the new values of a, b, c, and d?
a=1, b=3, c=3, d=5
## Assignment Operators

<table>
<thead>
<tr>
<th>Statement</th>
<th>Equivalent Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = a + 2 ;</td>
<td>a += 2 ;</td>
</tr>
<tr>
<td>a = a - 3 ;</td>
<td>a -= 3 ;</td>
</tr>
<tr>
<td>a = a * 2 ;</td>
<td>a *= 2 ;</td>
</tr>
<tr>
<td>a = a / 4 ;</td>
<td>a /= 4 ;</td>
</tr>
<tr>
<td>a = a % 2 ;</td>
<td>a %= 2 ;</td>
</tr>
<tr>
<td>b = b + (c + 2) ;</td>
<td>b += c + 2 ;</td>
</tr>
<tr>
<td>d = d * (e - 5) ;</td>
<td>d *= e - 5 ;</td>
</tr>
</tbody>
</table>
**Practice with Assignment Operators**

```c
int i = 1, j = 2, k = 3, m = 4;
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i += j + k</td>
<td></td>
</tr>
<tr>
<td>j *= k = m + 5</td>
<td></td>
</tr>
<tr>
<td>k -= m /= j * 2</td>
<td></td>
</tr>
</tbody>
</table>
Practice with Assignment Operators

```c
int i = 1, j = 2, k = 3, m = 4;
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>i += j + k</code></td>
<td><code>i=6</code></td>
</tr>
<tr>
<td><code>j *= k = m + 5</code></td>
<td><code>k=9, j=18</code></td>
</tr>
<tr>
<td><code>k -= m /= j * 2</code></td>
<td><code>m=1, k=2</code></td>
</tr>
</tbody>
</table>
Relational Expressions and Evaluation
### Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>is equal to</td>
</tr>
<tr>
<td>!=</td>
<td>is not equal to</td>
</tr>
</tbody>
</table>

Relational expressions evaluate to the integer values 1 (true) or 0 (false).

All of these operators are called **binary operators** because they take two expressions as operands.
Practice with Relational Expressions

```c
int a = 1, b = 2, c = 3;
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a &lt; c</td>
<td></td>
<td>(a + b) &gt;= c</td>
<td></td>
</tr>
<tr>
<td>b &lt;= c</td>
<td></td>
<td>(a + b) == c</td>
<td></td>
</tr>
<tr>
<td>c &lt;= a</td>
<td></td>
<td>a != b</td>
<td></td>
</tr>
<tr>
<td>a &gt; b</td>
<td></td>
<td>(a + b) != c</td>
<td></td>
</tr>
<tr>
<td>b &gt;= c</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Practice with Relational Expressions

```c
int a = 1, b = 2, c = 3;
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a &lt; c</td>
<td>T</td>
<td>(a + b) &gt;= c</td>
<td>T</td>
</tr>
<tr>
<td>b &lt;= c</td>
<td>T</td>
<td>(a + b) == c</td>
<td>T</td>
</tr>
<tr>
<td>c &lt;= a</td>
<td>F</td>
<td>a != b</td>
<td>T</td>
</tr>
<tr>
<td>a &gt; b</td>
<td>F</td>
<td>(a + b) != c</td>
<td>F</td>
</tr>
<tr>
<td>b &gt;= c</td>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arithmetic Expressions: True or False

- **Arithmetic expressions** evaluate to numeric values.

- An arithmetic expression that has a **value of zero** is false.

- An arithmetic expression that has a value **other than zero** is true.
# Practice with Arithmetic Expressions

```c
int    a = 1, b = 2, c = 3;
float  x = 3.33, y = 6.66;
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Numeric Value</th>
<th>True/False</th>
</tr>
</thead>
<tbody>
<tr>
<td>a + b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b - 2*a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c - b - a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c - a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y - x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y - 2*x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Practice with Arithmetic Expressions

```plaintext
int a = 1, b = 2, c = 3;
float x = 3.33, y = 6.66;
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Numeric Value</th>
<th>True/False</th>
</tr>
</thead>
<tbody>
<tr>
<td>a + b</td>
<td>3</td>
<td>T</td>
</tr>
<tr>
<td>b - 2 * a</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>c - b - a</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>c - a</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>y - x</td>
<td>3.33</td>
<td>T</td>
</tr>
<tr>
<td>y - 2 * x</td>
<td>0.0</td>
<td>F</td>
</tr>
</tbody>
</table>
Structured Programming

- All programs can be written in terms of only three control structures
  - Sequence, selection and repetition
- The **sequence** structure
  - Unless otherwise directed, the statements are executed in the order in which they are written.
- The **selection** structure
  - Used to choose among alternative courses of action.
- The **repetition** structure
  - Allows an action to be repeated while some condition remains true.
Selection: the if statement

```c
if ( condition )
{
    statement(s) /* body of the if statement */
}
```

The braces are not required if the body contains only a single statement.

However, they are a good idea and are required by the C Coding Standards.
Examples

```c
if ( age >= 18 )
{
    printf("Vote!\n");
}

if ( value == 0 )
{
    printf("You entered a zero.\n");
    printf ("Please try again.\n");
}
```
Good Programming Practice

• Always place braces around the body of an if statement.

• Advantages:
  – Easier to read
  – Will not forget to add the braces if you go back and add a second statement to the body
  – Less likely to make a semantic error

• **Indent the body of the if statement 3 to 4 spaces** -- **be consistent!**
Selection: the if-else statement

```java
if ( condition )
{
    statement(s) /*if clause */
}
else
{
    statement(s) /*else clause */
}
```
Example

```c
if ( age >= 18 )
{
    printf("Vote!\n") ;
}
else
{
    printf("Maybe next time!\n") ;
}
```
Good Programming Practice

• Always place braces around the bodies of the if and else clauses of an if-else statement.

• Advantages:
  – Easier to read
  – Will not forget to add the braces if you go back and add a second statement to the clause
  – Less likely to make a semantic error

• Indent the bodies of the if and else clauses 3 to 5 spaces -- be consistent!
Nesting of if-else Statements

```java
if ( condition1 )
{
    statement(s)
}
else if ( condition2 )
{
    statement(s)
}
... /* more else clauses may be here */
else
{
    statement(s) /* the default case */
}
```
if ( value == 0 )
{
    printf("Value you entered was 0\n");
}
else if ( value < 0 )
{
    printf("%d is negative.\n", value);
}
else
{
    printf("%d is positive.\n", value);
}
Bad Example: 2 if 1 else

```c
if ( n > 0 )
    if ( a > b )
        z=a;
    else
        z=b;
else
    z=b;
```

```c
if ( n > 0 )
{
    if (a> b)
        z=a;
}
else
    z=b;
```

Indentation will not ensure result:

`else` match with closest `if`.

int  a = 2 ;
if ( a = 1 ) /* semantic(logic) Err! */
{
    printf ("a is one\n") ;
}
else  if (a == 2 )
{
    printf ("a is two\n") ;
}  else {
    printf ("a is %d\n", a) ;
}
Gotcha ..

• The statement `if (a = 1)` is syntactically correct, so no error message will be produced. (Some compilers will produce a warning.) However, a semantic (logic) error will occur.

• An assignment expression has a value -- the value being assigned. In this case the value being assigned is 1, which is true.

• If the value being assigned was 0, then the expression would evaluate to 0, which is false.

• This is a VERY common error. So, if your if-else structure always executes the same, look for this typographical error.
Logical Operators

• So far we have seen only simple conditions.
  
  ```
  if ( count > 10 ) . . .
  ```

• Sometimes we need to test multiple conditions in order to make a decision.

• **Logical operators** are used for combining simple conditions to make complex conditions.

  ```
  && is AND if ( x > 5 && y < 6 )
  || is OR if ( z == 0 || x > 10 )
  ! is NOT if (! (bob > 42) )
  ```
Example Use of &&

```c
if ( age < 1 && gender == 'm') {
    printf ("Infant boy\n") ;
}
```
### Truth Table for `&&`

<table>
<thead>
<tr>
<th>Expression₁</th>
<th>Expression₂</th>
<th>Expression₁ &amp;&amp; Expression₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>nonzero</td>
<td>0</td>
</tr>
<tr>
<td>nonzero</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nonzero</td>
<td>nonzero</td>
<td>1</td>
</tr>
</tbody>
</table>

Exp₁ && Exp₂ && ... && Expₙ will evaluate to 1 (true) only if ALL subconditions are true.
if (grade==‘E’ || grade==‘F’)
{
    printf("See you next semester!\n") ;
}
### Truth Table for $||$

| $\text{Expression}_1$ | $\text{Expression}_2$ | $\text{Expression}_1 || \text{Expression}_2$ |
|-----------------------|-----------------------|---------------------------------------------|
| 0                     | 0                     | 0                                           |
| 0                     | nonzero               | 1                                           |
| nonzero               | 0                     | 1                                           |
| nonzero               | nonzero               | 1                                           |

$\text{Exp}_1 \&\& \text{Exp}_2 \&\& \ldots \&\& \text{Exp}_n$ will evaluate to 1 (true) if only ONE subcondition is true.
Example Use of !

```c
if (!(x==2)) /* Same as (x!=2) */
{
    printf("x is not equal to 2") ;
}
```
<table>
<thead>
<tr>
<th>Expression</th>
<th>! Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>nonzero</td>
<td>0</td>
</tr>
</tbody>
</table>
# Operator Precedence and Associativity

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>left to right/inside-out</td>
</tr>
<tr>
<td>++ -- ! + (unary) - (unary) (type)</td>
<td>right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
</tr>
<tr>
<td>+ (addition) - (subtraction)</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>left ot right</td>
</tr>
<tr>
<td>== !=</td>
<td>left to right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>= += -= *= /= %=</td>
<td>right to left</td>
</tr>
<tr>
<td>, (comma)</td>
<td>right to left</td>
</tr>
</tbody>
</table>
Thanks