Basic C Operators

- Arithmetic operators
  - Unary operators
  - Binary operators
- Assignment operators
- Equalities and relational operators
- Logical operators
- Conditional operator
Arithmetic Operators I

In C, we have the following operators (note that all these examples are using 9 as the value of its first operand):

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operator</th>
<th>Operand</th>
<th>Value after</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Increment</td>
<td>++</td>
<td>-</td>
<td>10</td>
</tr>
<tr>
<td>Decrement</td>
<td>- -</td>
<td>-</td>
<td>8</td>
</tr>
<tr>
<td>Modulus</td>
<td>% (!!!)</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Arithmetic Operators II

There are 2 types of arithmetic operators in C:

- unary operators
  - operators that require only one operand.
- binary operators.
  - operators that require two operands.
Unary Operator

<table>
<thead>
<tr>
<th>C Operation</th>
<th>Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>+</td>
<td>a = +3</td>
</tr>
<tr>
<td>Negative</td>
<td>-</td>
<td>b = -a</td>
</tr>
<tr>
<td>Increment</td>
<td>++</td>
<td>i++</td>
</tr>
<tr>
<td>Decrement</td>
<td>--</td>
<td>i--</td>
</tr>
</tbody>
</table>

- The first assigns positive 3 to a
- The second assigns the negative value of a to b.
- i++ is equivalent to i = i + 1
- i-- is equivalent to i = i-1
PRE- / POST-Increment

- It is also possible to use ++i and --i instead of i++ and i--
- However, the two forms have a slightly yet important difference.
- Consider this example:
  ```c
  int a = 9;
  printf("%d\n", a++);
  printf("%d", a);
  ```

- The output would be:
  
  9
  10
PRE- / POST-Increment cont…

- But if we have:
  ```
  int a = 9;
  printf("%d\n", ++a);
  printf("%d", a);
  ```
- The output would be:
  
  10
  10
- `a++` would return the current value of `a` and then increment the value of `a`
- `++a` on the other hand increment the value of `a` before returning the value
The following table illustrates the difference between the prefix and postfix modes of the increment and decrement operator.

```c
int R = 10, count=10;
```

<table>
<thead>
<tr>
<th>++ Or -- Statement</th>
<th>Equivalent Statements</th>
<th>R value</th>
<th>Count value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R = count++;</td>
<td>R = count; count = count + 1</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>R = ++count;</td>
<td>count = count + 1; R = count;</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>R = count --;</td>
<td>R = count; count = count – 1;</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>R = --count;</td>
<td>Count = count – 1; R = count;</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>
Binary Operators

<table>
<thead>
<tr>
<th>C Operation</th>
<th>Operator</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>a + 3</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>a - 6</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>a * b</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>a / c</td>
</tr>
<tr>
<td>Modulus</td>
<td>%</td>
<td>a % x</td>
</tr>
</tbody>
</table>

- The division of variables of type int will always produce a variable of type int as the result.
- You could **only use** modulus (%) operation on int variables.
Assignment Operators

- Assignment operators are used to combine the '=' operator with one of the binary arithmetic operators.
- In the following slide, all operations starting from $c = 9$.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent Statement</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>$c += 7$</td>
<td>$c = c + 7$</td>
<td>$c = 16$</td>
</tr>
<tr>
<td>-=</td>
<td>$c -= 8$</td>
<td>$c = c - 8$</td>
<td>$c = 1$</td>
</tr>
<tr>
<td>*=</td>
<td>$c *= 10$</td>
<td>$c = c * 10$</td>
<td>$c = 90$</td>
</tr>
<tr>
<td>/=</td>
<td>$c /= 5$</td>
<td>$c = c / 5$</td>
<td>$c = 1$</td>
</tr>
<tr>
<td>%=</td>
<td>$c %= 5$</td>
<td>$c = c % 5$</td>
<td>$c = 4$</td>
</tr>
</tbody>
</table>
Precedence Rules

- Precedence rules come into play when there is a mixed of arithmetic operators in one statement. For example: \( x = 3 \times a - ++b \% 3; \)
- The rules specify which of the operators will be evaluated first.

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Associativity Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (highest)</td>
<td>()</td>
<td>left to right</td>
</tr>
<tr>
<td>2</td>
<td>unary</td>
<td>right to left</td>
</tr>
<tr>
<td>3</td>
<td>* / %</td>
<td>left to right</td>
</tr>
<tr>
<td>4</td>
<td>+ -</td>
<td>left to right</td>
</tr>
<tr>
<td>5 (lowest)</td>
<td>= += -= *= /= %=</td>
<td>right to left</td>
</tr>
</tbody>
</table>
Precedence Rules cont...

- For example: \( x = 3 \times a - ++b \% 3; \)
  
  *how would this statement be evaluated?*

- If we intend to have the statement evaluated differently from the way specified by the precedence rules, we need to specify it using parentheses ( ).

- Using parenthesis, we will have

  \[ x = 3 \times ((a - ++b)\%3); \]

- The expression inside a parentheses will be evaluated first.

- The inner parentheses will be evaluated earlier compared to the outer parentheses.
Equality and Relational Operators

- **Equality Operators:**
  - **Operator** | **Example** | **Meaning**
  - `==` | `x == y` | x is equal to y
  - `!=` | `x != y` | x is not equal to y

- **Relational Operators:**
  - **Operator** | **Example** | **Meaning**
  - `>` | `x > y` | x is greater than y
  - `<` | `x < y` | x is less than y
  - `>=` | `x >= y` | x is greater than or equal to y
  - `<=` | `x <= y` | x is less than or equal to y

Logical Operators

- Logical operators are useful when we want to test multiple conditions.

- There are 3 types of logical operators and they work the same way as the boolean AND, OR and NOT operators.

- && - Logical AND
  - All the conditions must be true for the whole expression to be true.
  - Example: if (a == 10 && b == 9 && d == 1) means the if statement is only true when $a == 10$ and $b == 9$ and $d == 1$. 
Logical Operators cont...

- **||** - Logical OR
  - The truth of one condition is enough to make the whole expression true.
  - Example: if \((a == 10 \text{ || } b == 9 \text{ || } d == 1)\)
    means the if statement is true when either one of \(a\), \(b\) or \(d\) has the right value.

- **!** - Logical NOT (also called logical negation)
  - Reverse the meaning of a condition
  - Example: if \(!((points > 90))\)
    means if points not bigger than 90.
Conditional Operator

• The conditional operator (?:) is used to simplify an if/else statement.

• Syntax:
  
  Condition ? Expression1 : Expression2

• The statement above is equivalent to:

  ```java
  if (Condition)
      Expression1
  else
      Expression2
  ```
Conditional Operator cont...

Example 1:

if/else statement:
if (total > 60)
    grade = ‘P’
else
    grade = ‘F’;

conditional statement:
total > 60 ? grade = ‘P’: grade = ‘F’;

OR

grade = total > 60 ? ‘P’: ‘F’;
Example 2:

**if/else statement:**

```c
if (total > 60)
    printf("Passed!!\n");
else
    printf("Failed!!\n");
```

**Conditional Statement:**

```c
printf("%s!!\n", total > 60? "Passed": "Failed");
```
SUMMARY

- This chapter exposed you the operators used in C
  - Arithmetic operators
  - Assignment operators
  - Equalities and relational operators
  - Logical operators
  - Conditional operator

- Precedence levels come into play when there is a mixed of arithmetic operators in one statement.
- Pre/post fix - effects the result of statement