Lecture 6: Operators
Recap to previous lecture!

- Define a constant (how?)
- Identifier in C
- Arithmetic expression
Defining Names for Constants

- Constant is a value that can not be changed during the execution.

- Using a feature known as macro definition, we can name this constant:

```c
#define MY_CONSTANT 166
```
Identifiers

- Identifier is the name for a variable, function, constant and any other entity in your program.

- C places *no limit* on the maximum length of an identifier.

- An identifier in C may contain letters, digits, and underscores, *but must begin with a letter or underscore*:
  Correct identifiers: `times10` `get_char_N` `_done`
  Incorrect identifiers: `10times` `get-next-char`
Expressions

• Operators are the basic tools for building an expression in C.

• Three types of operators exist in C:
  – *Arithmetic operators*: a + b
  – *Relational operators* (comparisons): a > b
  – *Logical operators* (condition): a > 0 and a < 100
Arithmetic Operators

• C provides five binary *arithmetic operators*:
  
  – Unary: Unary plus and Unary minus
  
  – Binary: Additive and Multiplicative
    
    1. Additive: +, -
    
    2. Multiplicative: x, / and %
Compound Assignment

- Some programmers use the compound assignment operator as follows:

```c
i += 2;
/* same as i = i + 2; */
```
Compound Assignment

• There are nine other compound assignment operators, including the following:
  \[-=  *=  /=  %=\]

• All compound assignment operators work in much the same way:
  \[v += e \text{ adds } v \text{ to } e, \text{ storing the result in } v\]
  \[v -= e \text{ subtracts } e \text{ from } v, \text{ storing the result in } v\]
  \[v *= e \text{ multiplies } v \text{ by } e, \text{ storing the result in } v\]
  \[v /= e \text{ divides } v \text{ by } e, \text{ storing the result in } v\]
Increment and Decrement Operators

• C provides special `++` (*increment*) and `--` (*decrement*) operators.
• The `++` operator adds 1 to its operand. The `--` operator subtracts 1.

```
i = i + 1;   i += 1;   i++;  

j = j - 1;   j -= 1;   j--;  
```
Selection Statements in C
Most of C’s remaining statements fall into three categories:

- **Selection statements**: if and switch
- **Iteration statements**: while, do, and for
- **Jump statements**: break, continue, return, and goto.
Statements

• C also supports two more statements:
  – *Compound statement*: Groups several statements into a single statement.
  – *Null statement*: Performs no action!
More on Expressions!
Logical Expressions

• Several of C’s statements must test the value of an expression to see if it is “true” or “false”
• For example, we need to create a statement that tests *if the value of variable i is greater than the value of variable j*.

• Expressions of this type in many programming languages are called *logical expressions*.

• Such logical expressions are of special type named “*Boolean*”.
Logical Expressions

• The “Boolean” or “Logical” type would have only two values:

  1. true (integer value 1)
  2. false (integer value 0)

• There exist a set of operators that are used to build such logical expressions.
Relational Operators

• C’s relational operators:

<table>
<thead>
<tr>
<th>Symbol</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>
</tbody>
</table>
Guess the value of the logical expressions below.

<table>
<thead>
<tr>
<th>32</th>
<th>&lt;</th>
<th>64</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>&lt;=</td>
<td>64</td>
<td>?</td>
</tr>
<tr>
<td>32.5</td>
<td>&lt;=</td>
<td>64</td>
<td>?</td>
</tr>
<tr>
<td>32.5</td>
<td>&lt;</td>
<td>64.0</td>
<td>?</td>
</tr>
</tbody>
</table>

- The relational operators can be used to compare integers and floating-point numbers, with operands of **mixed types allowed**.
Relational Operators

• The precedence of the relational operators is *lower than* that of the arithmetic operators.

\[ i + j < k - 1 \implies (i + j) < (k - 1). \]

• The relational operators are *left associative*. 
Equality Operators

• C provides two *equality operators*:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>

• The equality operators are also *left associative* and have lower precedence than the relational operators.

\[
i < j == j < k \quad \Rightarrow \quad (i < j) == (j < k)
\]
What are these symbols ?!

\[
\begin{align*}
\leq \\
\geq \\
\ne \\
= \\
\end{align*}
\]
Logical Operators

• More complicated logical expressions can be built from simpler ones by using the logical operators:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Logical negation</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Logical and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Logical Operators .. Negation (!)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>!(1) = 0</td>
</tr>
<tr>
<td>0</td>
<td>!(0) = 1</td>
</tr>
</tbody>
</table>
**Logical Operators .. Logical and (&&)**

<table>
<thead>
<tr>
<th>Expression A</th>
<th>Expression B</th>
<th>Result (A &amp;&amp; B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Logical Operators .. Logical or (||)

| Expression A | Expression B | Result (A || B) |
|--------------|--------------|----------------|
| 0            | 0            | 0              |
| 0            | 1            | 1              |
| 1            | 0            | 1              |
| 1            | 1            | 1              |
Logical Operators

- Behavior of the logical operators:
  - !expr has the value 1 if expr has the value 0.
  - expr1 && expr2 has the value 1 if the values of expr1 and expr2 are both nonzero.
  - expr1 || expr2 has the value 1 if either expr1 or expr2 (or both) has a nonzero value.

- In all other cases, these operators produce the value 0.
Logical Operators

• The ! operator is unary, while && and || are binary.
• The logical operators produce 0 or 1 as their result.
• The logical operators treat any nonzero operand as a true value and any zero operand as a false value.
Logical Operators

Both && and || perform “short-circuit” evaluation: they first evaluate the left operand, then the right one. (Why?)
Logical Operators

• If the value of the expression can be deduced from the left operand alone, *the right operand isn’t evaluated.*

• Example:

\[(i \neq 0) \land (j > 0)\]

• If \(i\) is 0, the entire expression must be *false*, so there’s no need to evaluate \((j > 0)\).

• \((i \neq 0)\) is evaluated first. If \(i\) isn’t equal to 0, then \((j/ i > 0)\) is evaluated.
Relational Operators

- The expression $i < j < k$ is legal, but does not test whether $j$ lies between $i$ and $k$ (why?).

- Since the $<$ operator is *left associative*, this expression is equivalent to: $(i < j) < k$
  The 1 or 0 produced by $i < j$ is then compared to $k$.

- The correct expression is: $i < j \&\& j < k$
Guess the output

- \( A = 1; \quad B = 2; \quad C = 3; \quad D = 1; \)

Expression : \((A \neq D) \lor (A < B) \land (C \geq 3)\)

Expression : \((A \neq D) \land (A < B) \lor (C \geq 3)\)

Expression : \((1 = 3) \land (A < B) \lor (C \geq 3)\)
Selection Statements in C
The if Statement

- The if statement allows a program to choose between *two alternatives* by testing an expression.
- In its simplest form, the if statement has the form

```
if ( expression )
{
    Statements
}
```
The if Statement

• When the expression is true, the statements are executed.

• Example:

```c
int x = 100;
if ( x >= 30 )
{
    printf("the value of X is greater than 30");
}
```
Example

• Build a simple program that asks the user for his zipcode (3digits), and print “welcome” if the zipcode is between 400 and 500

• Update the above program to print “incorrect” if the zipcode is less than 100 or greater than 500